# 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.
- DPLL($T$) = DPLL($X$) + $T$-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.
- DPLL($T$) = 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.
- DPLL($T$) = DPLL($X$) + $T$-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)

# 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.

# 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 $A \| F$ (see [NOT], JACM'06):

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

$$\emptyset \quad \| \quad \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow$$

# 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):

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

| | | | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2}$ | $\Rightarrow$ | (Decide) |
| 1 | $\|$ | $\overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| 1 2 | $\|$ | $\overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2}$ | $\Rightarrow$ | (Decide) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$ | $\Rightarrow$ | (UnitPropagate) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4\ 5\ \overline{6}$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | | |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4\ 5\ \overline{6}$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Backtrack) |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4\ 5\ \overline{6}$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Backtrack) |
| $1\ 2\ 3\ 4\ \overline{5}$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | | |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4\ 5\ \overline{6}$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | $\Rightarrow$ | (Backtrack) |
| $1\ 2\ 3\ 4\ \overline{5}$ | $\|$ | $\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$ | | model found! |

# 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):

| **Assignment** $A$ : | | **Clause set** $F$ : | | |
|---|---|---|---|---|
| $\emptyset$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Decide) |
| $1\ 2\ 3\ 4\ 5$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (UnitPropagate) |
| $1\ 2\ 3\ 4\ 5\ \overline{6}$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | $\Rightarrow$ | (Backtrack) |
| $1\ 2\ 3\ 4\ \overline{5}$ | $\|$ | $\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$ | | model found! |

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

# Backtrack vs. Backjump

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

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

$$\begin{array}{llll}
\emptyset & \parallel & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \quad (\text{Decide}) \\
\vdots & \vdots & \quad\quad\quad\quad \vdots & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \parallel & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Rightarrow \quad (\text{Backjump})
\end{array}$$

# Backtrack vs. Backjump

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

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

$$
\begin{array}{llll}
\emptyset & \| \quad \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots & \vdots \qquad\qquad\quad \vdots & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \| \quad \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & (\text{Backjump}) \\
1\ 2\ \overline{5} & \| \quad \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & \ldots
\end{array}
$$

# Backtrack vs. Backjump

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

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

$$
\begin{array}{llllllll}
\emptyset & \| & \overline{1} \vee 2, & \overline{3} \vee 4, & \overline{5} \vee \overline{6}, & 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots & \vdots & & \vdots & & & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \| & \overline{1} \vee 2, & \overline{3} \vee 4, & \overline{5} \vee \overline{6}, & 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & (\text{Backjump}) \\
1\ 2\ \overline{5} & \| & \overline{1} \vee 2, & \overline{3} \vee 4, & \overline{5} \vee \overline{6}, & 6 \vee \overline{5} \vee \overline{2} & \Rightarrow & \ldots
\end{array}
$$

Backjump =

1. Conflict Analysis: “Find” a backjump clause $C \vee l$ (here, $\overline{2} \vee \overline{5}$)
   - 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: $\ldots 6 \ldots \overline{7} \ldots 9$ and let $F$ contain:

$\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}$, $8 \vee 7 \vee \overline{5}$, $\overline{6} \vee 8 \vee 4$, $\overline{4} \vee \overline{1}$, $\overline{4} \vee 5 \vee 2$, $5 \vee 7 \vee \overline{3}$, $1 \vee \overline{2} \vee 3$.

UnitPropagate gives $\ldots 6 \ldots \overline{7} \ldots 9\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$. Conflict w/ $1 \vee \overline{2} \vee 3$!

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

$$
\dfrac{8 \vee 7 \vee \overline{5} \qquad \dfrac{\overline{6} \vee 8 \vee 4 \qquad \dfrac{\overline{4} \vee \overline{1} \qquad \dfrac{\overline{4} \vee 5 \vee 2 \qquad \dfrac{5 \vee 7 \vee \overline{3} \qquad 1 \vee \overline{2} \vee 3}{5 \vee 7 \vee 1 \vee \overline{2}}}{\overline{4} \vee 5 \vee 7 \vee 1}}{5 \vee 7 \vee \overline{4}}}{\overline{6} \vee 8 \vee 7 \vee 5}}{8 \vee 7 \vee \overline{6}}
$$

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

Can use this backjump clause $8 \vee 7 \vee \overline{6}$ 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...)

# 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!

# 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, ...]

# 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: $f(g(a)) \neq f(c) \vee g(a) = d, \quad g(a) = c, \quad c \neq d$

Example 2: several (how many?) combined theories:

2 clauses: $A = write(B, i+1, x), \quad read(A, j+3) = y \vee f(i-1) \neq f(j+1)$

Typical verification examples, where SMT is method of choice.

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2, \quad 3, \quad \overline{4}\ \}$ to SAT solver

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a))\neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \quad \underbrace{g(a)=c}_{3}, \quad \underbrace{c\neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1}\vee 2,\quad 3,\quad \overline{4}\ \}$ to SAT solver

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

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4}\ \}$ to SAT solver

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

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

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4},\quad 1 \vee \overline{3} \vee 4\ \}$ to SAT solver

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2, \quad 3, \quad \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2, \quad 3, \quad \overline{4}, \quad 1 \vee \overline{3} \vee 4\ \}$ 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:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ 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:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4},\quad 1 \vee \overline{3} \vee 4\ \}$ 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 $\{\overline{1} \vee 2,\quad 3,\quad \overline{4},\quad 1 \vee \overline{3} \vee 4,\quad \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\ \}$ to SAT solver

# The Lazy approach to SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4}\ \}$ 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 $\{\ \overline{1} \vee 2,\quad 3,\quad \overline{4},\quad 1 \vee \overline{3} \vee 4\ \}$ 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 $\{\overline{1} \vee 2,\quad 3,\quad \overline{4},\quad 1 \vee \overline{3} \vee 4,\quad \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\ \}$ 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)

**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($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

$$\emptyset \quad \| \quad \overline{1} \vee 2,\ \ 3,\ \ \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate})$$

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

$$\begin{array}{llllll} \emptyset & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\ 3 & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \end{array}$$

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

$$\begin{array}{lclcl}
\emptyset & \parallel & \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
3 & \parallel & \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
3\ 1 & \parallel & \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{UnitPropagate})
\end{array}$$

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

$$\begin{array}{llll}
\emptyset & \| \quad \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
3 & \| \quad \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
3\ 1 & \| \quad \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
3\ 1\ 2 & \| \quad \overline{1} \vee 2,\ \ 3,\ \ \overline{4} & \Rightarrow & (\text{T-Propagate})
\end{array}$$

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a))\neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \quad \underbrace{g(a)=c}_{3}, \quad \underbrace{c\neq d}_{\overline{4}}$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\emptyset$ | $\parallel$ | $\overline{1}\vee 2,$ | $3,$ | $\overline{4}$ | $\Rightarrow$ | (UnitPropagate) |
| $3$ | $\parallel$ | $\overline{1}\vee 2,$ | $3,$ | $\overline{4}$ | $\Rightarrow$ | (T-Propagate) |
| $3\ 1$ | $\parallel$ | $\overline{1}\vee 2,$ | $3,$ | $\overline{4}$ | $\Rightarrow$ | (UnitPropagate) |
| $3\ 1\ 2$ | $\parallel$ | $\overline{1}\vee 2,$ | $3,$ | $\overline{4}$ | $\Rightarrow$ | (T-Propagate) |
| $3\ 1\ 2\ 4$ | $\parallel$ | $\overline{1}\vee 2,$ | $3,$ | $\overline{4}$ | $\Rightarrow$ | |

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a))\neq f(c)}_{\overline{1}} \vee \underbrace{g(a)=d}_{2}, \qquad \underbrace{g(a)=c}_{3}, \qquad \underbrace{c\neq d}_{\overline{4}}$$

$$\begin{array}{llllll}
\emptyset & \| & \overline{1}\vee 2, & 3, & \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
3 & \| & \overline{1}\vee 2, & 3, & \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
3\ 1 & \| & \overline{1}\vee 2, & 3, & \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
3\ 1\ 2 & \| & \overline{1}\vee 2, & 3, & \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
3\ 1\ 2\ 4 & \| & \overline{1}\vee 2, & 3, & \overline{4} & \Rightarrow & \mathit{unsat}
\end{array}$$

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

# DPLL($T$) Example (the same EUF one)

Notation used: Abstract DPLL Modulo Theories:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

$$\begin{array}{llllll}
\emptyset & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
3 & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
3\ 1 & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
3\ 1\ 2 & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
3\ 1\ 2\ 4 & \| & \overline{1} \vee 2, & 3, & \overline{4} & \Rightarrow \quad \mathit{unsat}
\end{array}$$

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

Explanation for last T-Propagate:

$2 \wedge 3 \rightarrow 4$ or, equivalently, $\overline{2} \vee \overline{3} \vee 4$

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
  $l_1 \wedge \ldots \wedge l_n \rightarrow lit$ 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
  $l_1 \wedge \ldots \wedge l_n \to lit$ 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 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) = 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 $1 \dots n$.

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 3 | 4 | | |
| 3 | 4 | 5 | | |
| 4 | 5 | | | |
| 5 | | | | |

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 $p_{ij1} \vee \dots \vee p_{ijn}$
  at most one $k$ per $[i,j]$: 2-lit clauses like $\overline{p_{ij1}} \vee \overline{p_{ij2}}$
- 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 `alldifferent`

QGC Example continued:

| $x$ | $y$ | $z$ | | |
|---|---|---|---|---|
| | 3 | 4 | | |
| 3 | 4 | 5 | | |
| 4 | 5 | | | |
| 5 | | | | |

**`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 `alldifferent`

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

Application to real-world professional round-robin sports scheduling

Sometimes better results with weaker alldiff propagation

# Another example: DPLL(`cumulative`)

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

# 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 $start(t) \leq h$ ( so $\overline{s_{t,h-1}} \wedge s_{t,h}$ means $start(t) = h$ ).
- 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 $start(t) \leq h$ ( so $\overline{s_{t,h-1}} \wedge s_{t,h}$ means $start(t) = h$ ).
- 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 $\overline{s_{t,h-duration(t)}} \wedge s_{t,h} \longrightarrow a_{t,h}$
- T-solver handles, for each hour $h$ and each resource $r$, one Pseudo-Boolean constr. like $3a_{t,h} + 4a_{t',h} + \ldots \leq capacity(r)$

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 $start(t) \leq h$ ( so $\overline{s_{t,h-1}} \wedge s_{t,h}$ means $start(t) = h$ ).
- 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 $\overline{s_{t,h-duration(t)}} \wedge s_{t,h} \longrightarrow a_{t,h}$
- T-solver handles, for each hour $h$ and each resource $r$, one Pseudo-Boolean constr. like $3a_{t,h} + 4a_{t',h} + \ldots \leq capacity(r)$

Very good results.
But... why can SAT sometimes still beat SMT? See below!

# Proof complexity and other insights (I)

The pigeon-hole principle for $n$ pigeons and $n-1$ holes:

Let $PHP^n_{n-1}$ denote the set of clauses:

- $x_{i,1} \vee \ldots \vee x_{i,n-1}$ for $i = 1 \ldots n$
  (every pigeon is in at least one hole)
- $\overline{x_{i,k}} \vee \overline{x_{j,k}}$ for $1 \leq i < j \leq n$ and $1 \leq k \leq n-1$
  (no two pigeons are in the same hole)

[Haken'85]:
Any resolution refutation of $PHP^n_{n-1}$ 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:

$$\dfrac{id_k \;\ldots \qquad \dfrac{id_3 \;\ldots \qquad \dfrac{id_2\!: \; 5 \vee 7 \vee \overline{3} \qquad id_1\!: \; 1 \vee \overline{2} \vee 3}{5 \vee 7 \vee 1 \vee \overline{2}}}{\iddots}}{id\!: \; lemma}$$

One trace line per conflict/lemma: $id \leftarrow \{id_1 \ldots id_k\}$

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

trace file $\geq$ (binary) resolution refutation:

SAT solver runtime $\geq$ size of smallest resolution refutation.

# Proof complexity and other insights (III)

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:

$T$ = 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

$$y_1 \wedge \ldots \wedge y_k \rightarrow \overline{y}$$

# 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:

$T$ = 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

$$y_1 \wedge \ldots \wedge y_k \rightarrow \overline{y}$$

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 $\Longrightarrow$ 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 wich one will be a bottle neck!

# When to use SAT, and when SMT?

- Most constraints are no bottle necks and generate very few explanations $\Longrightarrow$ 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 wich 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 $\Longrightarrow$ 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 wich 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?

# Auxiliary variables & proof complexity

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 $\rightarrow$ 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 $\rightarrow$ 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!