
SMT: Enhancing SAT with Special-purpose Algorithms

SAT'09 Swansea

Robert Nieuwenhuis

(+ Albert Oliveras, Enric Rodríguez, Morgan Deters, Javier Larrosa, ...)

Barcelogic 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 to CP applications: scheduling, timetabling...
- How to use SMT to get the **best of both worlds:**
 - From SAT:
efficiency, robustness, no need for tuning.
 - From (complete methods in) CP:
expressiveness, rich modeling languages, special-purpose algorithms for arithmetic, global constraints....

Outline of this talk

Outline of this talk

- Good vs Bad

Outline of this talk

- Good vs Bad in SAT and CP.
- SAT Solvers. Why do they work so well?
- Why did attempts to do this in CP fail?
- What is SAT Modulo Theories (SMT)?
- Lazy approach, improved Lazy approach.
- Our DPLL(T) approach: $DPLL(T) = DPLL(X) + T\text{-Solver}$.
- The *Barcelogic* SMT solver.
- Theories and T -Solvers
- CP-like theories and T -solvers. Applications and examples.
- Proof complexity and other insights
- SMT for Optimization.

What is meant by **CP solver** in this talk?

CP = Constraint Programming.

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

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

Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT (EDA \$)

Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT (EDA \$)

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

Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT (EDA \$)

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

What's GOOD? Complete solvers that:

- outperform by far the other methods (see later why)
- on real-world problems from many sources
- with a single, fully automatic, push-button strategy!
- Hence modeling is essentially declarative.

Good vs Bad in SAT Solvers

Decades of academic and industrial efforts in SAT (EDA \$)

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

What's GOOD? Complete solvers that:

- outperform by far the other methods (see later why)
- on real-world problems from many sources
- with a single, fully automatic, push-button 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 CP Solvers

Good vs Bad in CP Solvers

- What's GOOD?
 - Expressive modeling constructs and languages
 - Specialized algorithms for many (global) constraints
 - Optimization aspects better studied

Good vs Bad in CP Solvers

- What's GOOD?
 - Expressive modeling constructs and languages
 - Specialized algorithms for many (global) constraints
 - Optimization aspects better studied

What's BAD?

- Performance(?) (hard to assess, but...)
- Not quite automatic or push-button
 - Tuning needed per problem (and even per instance!)
- Research is sometimes a bit “academic”:
 - on random or artificial problems (sometimes not realistic)
 - no big database of real-world/industrial instances
- No common underlying syntax or language (this hinders many good things, see later)

DPLL (or CDCL) SAT Solvers

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

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset

$\parallel \bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2} \Rightarrow$

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2$, $\bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4 5 $\bar{6}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$		

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4 5 $\bar{6}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backtrack)

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4 5 $\bar{6}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backtrack)
1 2 3 4 $\bar{5}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$		

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4 5 $\bar{6}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backtrack)
1 2 3 4 $\bar{5}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$		model found!

DPLL (or CDCL) SAT Solvers

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

An **Abstract DPLL state** has the form $A \parallel F$ (cf. [NOT JACM06]):

Assignment A :

Clause set F :

\emptyset	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
1 2 3 4 5	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(UnitPropagate)
1 2 3 4 5 $\bar{6}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backtrack)
1 2 3 4 $\bar{5}$	\parallel	$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$		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 $\bar{5}$.

But note that decision level 3 4 is unrelated to the conflict $6 \vee \bar{5} \vee \bar{2}$:

\emptyset		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
\vdots	\vdots	\vdots		
1 2 3 4 5 $\bar{6}$		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backjump)

Backtrack vs. Backjump

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

But note that decision level 3 4 is unrelated to the conflict $6 \vee \bar{5} \vee \bar{2}$:

\emptyset		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
\vdots	\vdots	\vdots		
1 2 3 4 5 $\bar{6}$		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backjump)
1 2 $\bar{5}$		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	...

Backtrack vs. Backjump

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

But note that decision level 3 4 is unrelated to the conflict $6 \vee \bar{5} \vee \bar{2}$:

\emptyset		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Decide)
\vdots	\vdots	\vdots		
1 2 3 4 5 $\bar{6}$		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	(Backjump)
1 2 $\bar{5}$		$\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, 6 \vee \bar{5} \vee \bar{2}$	\Rightarrow	...

Backjump =

1. “Find” a **backjump clause** $C \vee l$ (in this example, $\bar{2} \vee \bar{5}$) that:
 - is a logical consequence of F
 - 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: ...6... $\bar{7}$...9 and let F contain:

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

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

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

$$\begin{array}{r}
 \frac{\frac{\frac{\frac{\frac{\frac{\bar{6} \vee 8 \vee 4}{8 \vee 7 \vee \bar{5}}}{\bar{6} \vee 8 \vee 7 \vee 5}}{\bar{4} \vee \bar{1}}}{\bar{4} \vee 5 \vee 2}}{\bar{4} \vee 5 \vee 7 \vee 1}}{\frac{5 \vee 7 \vee \bar{3}}{5 \vee 7 \vee 1 \vee \bar{2}}}}{1 \vee \bar{2} \vee 3} \\
 \bar{4} \vee 5 \vee 2 \\
 \bar{4} \vee 5 \vee 7 \vee 1 \\
 \bar{6} \vee 8 \vee 4 \\
 5 \vee 7 \vee \bar{4} \\
 8 \vee 7 \vee \bar{5} \\
 \bar{6} \vee 8 \vee 7 \vee 5 \\
 8 \vee 7 \vee \bar{6}
 \end{array}$$

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

Can use this backjump clause $8 \vee 7 \vee \bar{6}$ for **Backjump** to ...6... $\bar{7}$ 8.

Yes, but why is it really **that** good?

Three **key** ingredients that **only work if used TOGETHER**:

Yes, but why is it 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 future **similar** conflicts (i.e., repeated work)

Yes, but why is it 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 future **similar** conflicts (i.e., repeated work)
2. **Decide** on variables with **many occurrences in recent conflicts**:
 - so-called **activity-based heuristics**
 - idea: **work off**, one by one, **clusters** of tightly related vars

Yes, but why is it 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 future **similar** conflicts (i.e., repeated work)
2. **Decide** on variables with **many occurrences in recent conflicts**:
 - so-called **activity-based heuristics**
 - idea: **work off**, one by one, **clusters** of tightly related vars
3. **Forget** from time to time **low-activity lemmas**:
 - **crucial** to keep **UnitPropagate** fast and memory low
 - idea: lemmas from **worked off clusters** no longer needed!

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

- **First-fail** heuristics used (not very dynamic ones)
 - effect: work simultaneously on **too unrelated** variables
 - requires storing **too many** nogoods at the same time

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

- **First-fail** heuristics used (not very dynamic ones)
 - effect: work simultaneously on **too unrelated** variables
 - requires storing **too many** nogoods at the same time
- **No simple underlying syntax** as there is in SAT:
 - hard to express nogoods
 - hard to understand conflict analysis
 - hard to implement things **really** efficiently

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

- **First-fail** heuristics used (not very dynamic ones)
 - effect: work simultaneously on **too unrelated** variables
 - requires storing **too many** nogoods at the same time
- **No simple underlying syntax** as there is in SAT:
 - hard to express nogoods
 - 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!

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

- **First-fail** heuristics used (not very dynamic ones)
 - effect: work simultaneously on **too unrelated** variables
 - requires storing **too many** nogoods at the same time
- **No simple underlying syntax** as there is in SAT:
 - hard to express nogoods
 - 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]

Why did attempts to do this in CP fail?

It's not easy to get everything **together** right. But also...

- **First-fail** heuristics used (not very dynamic ones)
 - effect: work simultaneously on **too unrelated** variables
 - requires storing **too many** nogoods at the same time
- **No simple underlying syntax** as there is in SAT:
 - hard to express nogoods
 - 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 the satisfiability of an existential 1st-order formula w.r.t. 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:

$A = \text{write}(B, i+1, x), \quad \text{read}(A, j+3) = y \vee f(i-1) \neq f(j+1)$

Typical verification examples, where SMT is method of choice.

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

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

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

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

SAT solver returns model $[1, 2, 3, \bar{4}]$

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

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

SAT solver returns model $[1, 2, 3, \bar{4}]$

Theory solver says $[1, 2, 3, \bar{4}]$ is *T-inconsistent*

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

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

SAT solver returns model $[1, 2, 3, \bar{4}]$

Theory solver says $[1, 2, 3, \bar{4}]$ is *T-inconsistent*

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

The **Lazy** approach to SMT

Aka Lemmas on demand [dMR,'02].

Same EUF example:

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

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

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

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

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

SAT solver returns model $[1, 2, 3, \bar{4}]$

Theory solver says $[1, 2, 3, \bar{4}]$ is *T-inconsistent*

3. Send $\{\bar{1} \vee 2, 3, \bar{4}, 1 \vee \bar{3} \vee 4, \bar{1} \vee \bar{2} \vee \bar{3} \vee 4\}$ 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**

Our DPLL(T) approach ([NOT], JACM Nov 06)

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

- **Modular** and **flexible**: can plug in any T -Solvers into the DPLL(X) engine.
- T -Solvers specialized and fast in **Theory Propagation**:
 - Propagate input literals that are theory consequences
 - **more pruning** in improved lazy SMT
 - T -Solver also **guides** search, instead of only **validating** it
 - fully exploited in conflict analysis (non-trivial)
- DPLL(T) approach is being quite widely adopted (cf. Google).

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

Notation used: **Abstract DPLL Modulo Theories:**

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

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

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

Notation used: **Abstract DPLL Modulo Theories:**

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

$$\begin{array}{l} \emptyset \\ 3 \end{array} \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad \begin{array}{l} \text{(UnitPropagate)} \\ \text{(T-Propagate)} \end{array}$$

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

Notation used: **Abstract DPLL Modulo Theories:**

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

$$\begin{array}{l} \emptyset \\ 3 \\ 3 \ 1 \end{array} \quad \parallel \quad \begin{array}{l} \bar{1} \vee 2, \ 3, \ \bar{4} \\ \bar{1} \vee 2, \ 3, \ \bar{4} \\ \bar{1} \vee 2, \ 3, \ \bar{4} \end{array} \quad \Rightarrow \quad \begin{array}{l} (\text{UnitPropagate}) \\ (\text{T-Propagate}) \\ (\text{UnitPropagate}) \end{array}$$

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

Notation used: **Abstract DPLL Modulo Theories:**

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

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

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

Notation used: **Abstract DPLL Modulo Theories:**

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

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

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

Notation used: **Abstract DPLL Modulo Theories:**

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

\emptyset	\parallel	$\bar{1} \vee 2, 3, \bar{4}$	\Rightarrow	(UnitPropagate)
3	\parallel	$\bar{1} \vee 2, 3, \bar{4}$	\Rightarrow	(T-Propagate)
3 1	\parallel	$\bar{1} \vee 2, 3, \bar{4}$	\Rightarrow	(UnitPropagate)
3 1 2	\parallel	$\bar{1} \vee 2, 3, \bar{4}$	\Rightarrow	(T-Propagate)
3 1 2 4	\parallel	$\bar{1} \vee 2, 3, \bar{4}$	\Rightarrow	<i>unsat</i>

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

DPLL(T) Example (the same EUF one)

Notation used: **Abstract DPLL Modulo Theories:**

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

$$\begin{array}{l} \emptyset \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad (\text{UnitPropagate}) \\ 3 \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad (\text{T-Propagate}) \\ 3 \ 1 \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad (\text{UnitPropagate}) \\ 3 \ 1 \ 2 \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad (\text{T-Propagate}) \\ 3 \ 1 \ 2 \ 4 \quad \parallel \quad \bar{1} \vee 2, \quad 3, \quad \bar{4} \quad \Rightarrow \quad \text{unsat} \end{array}$$

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

Explanation for last **T-Propagate**:

$$2 \wedge 3 \rightarrow 4 \quad \text{or, equivalently,} \quad \bar{2} \vee \bar{3} \vee 4$$

Explanations are **T -lemmas**, i.e., **tautologies 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**
 $l_1 \wedge \dots \wedge l_n \rightarrow lit$ provided by **T -Solver**
Too new T -explanations are forbidden!

How should it be implemented?

- **UnitPropagate**: store a pointer to clause C , as in SAT solvers
- **T-Propagate**: (pre-)compute explanations at each **T-Propagate**?
 - **If possible**, only **on demand**, during conflict analysis
 - typically only one Explain for every 250 **T-Propagate**.
 - depends on T

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:
 - see *Barcelogic* in SAT Race 2008:
 - **best** on **unsat** problems
 - **third** on **sat+unsat** problems
 - **new ideas**: for binclauses, subsumption, simplification, ...
 - still lots of room for improvement
- *T-Solvers* for:
 - Congruences (EUF)
 - Integer/Real Difference Logic (*)
 - Linear Integer/Real Arithmetic
 - Arrays (*)
 - New: typical CP filtering algorithms

(*) = *Barcelogic* won this category at SMT-Competition'08

Theories and *T*-Solvers (1)

- Equality with Uninterpreted Functions (EUF)
 - typical application: pipelined processor verification
 - incremental, backtrackable, explaining congr. closure
 - built-in **integer offsets**: $f(a + 3, \dots) = f(b - 4, \dots) \wedge \dots$
 - [NO'07] Inf&Comp.
- Real/Integer Difference Logic (IDL/RDL)
 - Linear arith. w/ only atoms $x + k \geq y$ or $x \geq k$ or $x \leq k$.
 - linear-time incremental consistency check
 - good for timed automata, but also scheduling!
 - our algorithms based on [CM SAT'06]
 - **new faster version**, special treatment of **bounds**.

Theories and *T*-Solvers (2)

- Linear Real Arithmetic (LRA)
 - bounded primal simplex (both tableau and revised)
 - (dis)equalities pre-processed away: faster
 - does cheap Difference Logic checks and propagations first
 - lot of **new work**: representation of rationals, theory propagation, ... Also benefits from new DL solver
 - check SAT'08 paper on our solver vs CPLEX-11.

- Linear Integer Arithmetic (LIA)
 - branch-and-cut: cutting-planes + requests to **DPLL(X)** for **splittings on demand**
 - benefits from the **new** improvements in LRA.

Theories and *T*-Solvers (3)

- *Array Solver*: also *new* one
 - dedicated solver, not just axiom instantiation
 - no translation of writes into reads
 - requests *DPLL(X)* to do splittings on demand on equality literals between indices
 - appears to be very good when heavy array reasoning needed

What about more *CP-like theories*??

A DPLL(**alldifferent**) example

Example:

Quasi-Group Completion (QGC)

(aka latin squares).

Each row and column must contain $1 \dots n$.

Currently best is **3-D encoding in SAT**

where p_{ijk} means “row i col j has value k ”:

	3	4		
3	4	5		
4	5			
5				

- 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 one 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:

alldifferent (or Hall's Theorem) infers that x, y consume 1, 2 and hence $z = 3$.

x	y	z		
	3	4		
3	4	5		
4	5			
5				

Idea:

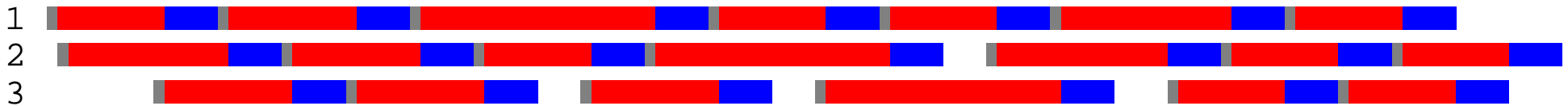
- Use 3-D encoding + SMT where T is **alldifferent**.
As usual in SMT, T -solver knows what p_{ijk} 's mean.
- From time to time invoke T -solver before **Decide**, but do always cheap SAT stuff first: **UnitPropagate**, **Backjump**, etc.
- T -solver = incremental filtering [Regin'94] **but with Explain**:
in our example, the literal p_{133} (meaning $z = 3$) is entailed by $\{ \overline{p_{113}} \ \overline{p_{114}} \ \dots \ \overline{p_{135}} \}$ (meaning $x \neq 3, x \neq 4, \dots, z \neq 5$).

Get CP with learning, backjumping, automatic heuristics...

Naive SMT on steel oven scheduling

- Resources: 3 platforms, 2 ovens, 1 cooler.
- Tasks need 1 hour loading, K h oven, 5 h cooling

Example: Optimal schedule (loading ■, the two ovens ■, the single cooler ■) for 20 tasks: 9 w. 10h oven, 5 w. 12h, 1 w. 15h, 2 w. 16h, 3 w. 22h.:



Naive declarative model in Difference Logic, like this:

- Boolean vars $ov(i, k)$ (“task i uses oven no. k ”)
- Integer vars $s(i)$ (starting time of task i)
- Clauses: $ov(i, k) \wedge ov(j, k) \rightarrow s(i) \geq s(j) + 16 \vee s(j) \geq s(i) + 10$

performs similarly to current best CP model, which is

3 cumulatives + postprocessing for oven/platform asgnmt.

And with DPLL(*cumulative*)?

Modeling by:

- Difference Logic literals (as before), or
- concrete values for the starting times, or
- intervals...

What is then *GOOD*?

And with DPLL(*cumulative*)?

Modeling by:

- Difference Logic literals (as before), or
- concrete values for the starting times, or
- intervals...

What is then **GOOD**? Get all the advantages of CP:

- expressive modeling
- efficiency from specialized filtering algorithms, ...

And with DPLL(**cumulative**)?

Modeling by:

- Difference Logic literals (as before), or
- concrete values for the starting times, or
- intervals...

What is then **GOOD**? Get all the advantages of CP:

- expressive modeling
- efficiency from specialized filtering algorithms, ...

And what is then **ALSO GOOD**? Get all the advantages of SAT:

- additional efficiency from learning
- from backjumping
- No tuning needed: automatic variable selection heuristics...

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, a little 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: $x_1 + \dots + x_n \geq k$ and $x_1 + \dots + x_n < k$

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

$$x_1 \wedge \dots \wedge x_k \rightarrow \overline{x_{k+1}}$$

Here a good SAT encoding (9 AM today) works better and...

Splitting on aux vars can give expon. speedup: **Extended Resol.**

Concluding remarks

- Progress in **optimization** problems.
See SAT'06 and also here next Friday.
Branch-and-bound is just another SMT theory.

Concluding remarks

- Progress in **optimization** problems.
See SAT'06 and also here next Friday.
Branch-and-bound is just another SMT theory.

- **Thank You!**