

# From Description Logics to Satisfiability Modulo Theories (and back?)

Roberto Sebastiani

Dept. of Information Science and Engineering, University of Trento, Italy

j.w.w.

Gilles Audemard, Marco Bozzano, Roberto Bruttomesso, **Alessandro Cimatti**, Anders Franzén, **Alberto Griggio**, Tommi Junttila, Arthur Kornilowicz, **Fausto Giunchiglia**, Enrico Giunchiglia, Silvio Ranise, Peter vanRossum, Stephan Schultz, Cristian Stenico, Armando Tacchella, Silvia Tomasi, **Michele Vescovi**,...

The 23<sup>rd</sup> International Workshop on Description Logics (DL 2010)  
Waterloo, Ontario, Canada, May 4-7th, 2010.

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities
  - Proofs and unsatisfiable cores
  - Interpolants
  - All-SMT
- 4 ... and back to DL?

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities
  - Proofs and unsatisfiable cores
  - Interpolants
  - All-SMT
- 4 ... and back to DL?

# Satisfiability Modulo Theories (SMT( $\mathcal{T}$ ))

## Satisfiability Modulo Theories (SMT( $\mathcal{T}$ ))

is the problem of deciding the satisfiability of (typically quantifier-free) formulas in some decidable first-order theory  $\mathcal{T}$ .

### Some theories of interest (e.g., for formal verification)

- *Equality and Uninterpreted Functions* ( $\mathcal{EUF}$ ):  
 $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- *Difference logic* ( $\mathcal{DL}$ ):  $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- *Linear arithmetic over the rationals* ( $\mathcal{LA}(\mathbb{Q})$ ):  
 $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- *Linear arithmetic over the integers* ( $\mathcal{LA}(\mathbb{Z})$ ):  
 $(x := x_l + 2^{16}x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- *Arrays*:  $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- *Bit vectors*:  $x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) \ll w_{[8]}[3 : 0]$

# From KSAT ... [Giunchiglia & Sebastiani CADE'96;KR'96]

```

function KSAT( $\varphi, \mu$ )
  if ( $\varphi == \top$ )                                /* base */
    then return KSATA( $\mu$ );
  if ( $\varphi == \perp$ )                                /* backtrack */
    then return False;
  if {a unit clause ( $l$ ) occurs in  $\varphi$ }           /* unit */
    then return KSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ );
   $l := \text{choose-literal}(\varphi);$                   /* split */
  return KSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ ) or
        KSAT(assign( $\neg l, \varphi$ ),  $\mu \wedge \neg l$ );

```

*/\*  $\mu$  is  $\bigwedge_i \Box_1 \alpha_{1i} \wedge \bigwedge_j \neg \Box_1 \beta_{1j} \wedge \dots \wedge \bigwedge_i \Box_m \alpha_{mi} \wedge \bigwedge_j \neg \Box_m \beta_{mj} \wedge \bigwedge_k A_k \wedge \bigwedge_h \neg A_h$  \*/*

```

function KSATA( $\mu$ )
  for each box index  $r \in \{1 \dots m\}$  do
    for each literal  $\neg \Box_r \beta_{rj} \in \mu$  do
      if not (KSAT( $\bigwedge_i \alpha_{ri} \wedge \neg \beta_{rj}, \top$ ))
        then return False;
  return True;

```

# From KSAT ... [Giunchiglia & Sebastiani CADE'96;KR'96]

```

function KSAT( $\varphi, \mu$ )
  if ( $\varphi == \top$ )                                /* base */
    then return KSATA( $\mu$ );
  if ( $\varphi == \perp$ )                                /* backtrack */
    then return False;
  if {a unit clause ( $l$ ) occurs in  $\varphi$ }            /* unit */
    then return KSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ );
   $l := \text{choose-literal}(\varphi);$                     /* split */
  return KSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ ) or
        KSAT(assign( $\neg l, \varphi$ ),  $\mu \wedge \neg l$ );

```

- a DPLL-based procedure for  $K_m/\mathcal{ALC}$
- idea: DPLL as assignment enumerator instead of tableaux rules
- recursive calls to DPLL over the modal depth
- enhancements in KSAT [Giunchiglia & Sebastiani, KR'96,...], Fact [Horrocks TABLEAUX'98;...], DLP [Patel-Schneider DL'98;...], \*SAT [Giunchiglia et al. JAR'00;...],...:  
*early-pruning, atom normalization, backjumping, memo-izing, ...*

## ... to TSAT ... [Armando et al. ECP'99]

```

function TSAT( $\varphi, \mu$ )
  if ( $\varphi == \top$ )                                /* base */
    then return TSATW( $\mu$ );
  if ( $\varphi == \perp$ )                                /* backtrack */
    then return False;
  if {a unit clause ( $l$ ) occurs in  $\varphi$ }            /* unit */
    then return TSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ );
   $l := \text{choose-literal}(\varphi);$                     /* split */
  return TSAT(assign( $l, \varphi$ ),  $\mu \wedge l$ ) or
        TSAT(assign( $\neg l, \varphi$ ),  $\mu \wedge \neg l$ );

```

- a DPLL-based for *disjunctive temporal constraints (DTC)*:  
 $\bigwedge_i \bigvee_j (t_{ij1} - t_{ij2} \leq c_{ij}), \quad t_x, c_x \in \mathbb{Q}$
- TSAT<sub>W</sub> based on LPsolver (symplex)
- v.0 built on top of KSAT C++ code of [Giunchiglia et al. KR'98]
- outperformed previous tableau-based procedure for DTC
- contemporarily, LPSAT for *resource planning* [Wolfman & Weld, IJCAI'99]

## ... to “modern” CDCL SMT solvers

[Audemard et al. CADE'02; Barret et al. CAV'02; de Moura et al. CADE'02; ...]

```

function  $\mathcal{T}$ -DPLL( $\mathcal{T}$ -formula:  $\varphi$ ,  $\mathcal{T}$ -assignment &  $\mu$ ) {
  status :=  $\mathcal{T}$ -preprocess( $\varphi, \mu, \varphi^p, \mu^p$ ); //  $\varphi^p \Leftrightarrow \mathcal{T}2\mathcal{B}(\varphi)$ 
  while (1) { //  $\mu^p \Leftrightarrow \mathcal{T}2\mathcal{B}(\mu)$ 
     $\mathcal{T}$ -decide_next_branch( $\mu, \varphi^p, \mu^p$ );
    while (1) {
      status :=  $\mathcal{T}$ -deduce( $\varphi^p, \mu^p, \eta^p$ ); //  $\eta^p \Leftrightarrow \mathcal{T}2\mathcal{B}(\eta)$ 
      if (status == Sat)
        res :=  $\mathcal{T}$ -solver( $\mu, \eta$ );
        if (res==Sat)
          return Sat;
      if (status==Conflict || res==Unsat) {
        blevel :=  $\mathcal{T}$ -analyze_conflict( $\varphi^p, \mu^p, \eta^p$ );
        if (blevel == 0)
          return Unsat;
        else backtrack(blevel,  $\varphi^p, \mu^p$ );
      }
      else break;
    }
  }
}

```



## ... to “modern” CDCL SMT solvers

[Audemard et al. CADE'02; Barret et al. CAV'02; de Moura et al. CADE'02; ...]

- based on “modern” Conflict-Driven Clause-Learning DPLL solvers  
[Silva & Sakallah'96; Moskewicz et al.'01; Een & Sörensson, SAT'03]
- many theories ( $\mathcal{LA}(\mathbb{Q})$ ,  $\mathcal{LA}(\mathbb{Z})$ ,  $\mathcal{DL}$ ,  $\mathcal{UTVPI}$ ,  $\mathcal{AR}$ ,  $\mathcal{BV}$ ) and their combinations
- target FV (e.g., timed & hybrid systems, SW, HW RTL designs.,....)
- many enhancements

# Outline

1 From DL to SMT ...

2 **Efficient SMT solving**

- Modern SAT solvers
- Modern SMT solvers
- Theory Solvers and their combination

3 Beyond Solving: advanced SMT functionalities

- Proofs and unsatisfiable cores
- Interpolants
- All-SMT

4 ... and back to DL?

# Outline

1 From DL to SMT ...

2 **Efficient SMT solving**

- **Modern SAT solvers**
- Modern SMT solvers
- Theory Solvers and their combination

3 Beyond Solving: advanced SMT functionalities

- Proofs and unsatisfiable cores
- Interpolants
- All-SMT

4 ... and back to DL?

# Modern DPLL implementations

[Silva & Sakallah'96; Moskewicz et al.'01]

*Conflict-Driven Clause-Learning (CDCL)* DPLL solvers:

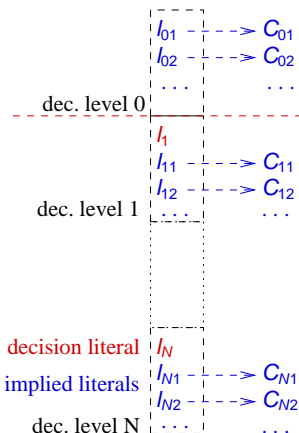
- Non-recursive: stack-based representation of data structures
- Efficient data structures for doing and undoing assignments (e.g., two-watched-literal scheme)
- Perform conflict-directed backtracking (backjumping) and learning
- May perform search restarts

Dramatically efficient: solve industrial-derived problems with  $\approx 10^7$  Boolean variables and  $\approx 10^7 - 10^8$  clauses

# Stack-based representation of a truth assignment $\mu$

- stack partitioned into *decision levels*:
  - one *decision literal*
  - its *implied literals*
  - each implied literal tagged with the clause causing its unit-propagation (*antecedent clause*)
- equivalent to an *implication graph*:
  - a node without incoming edges represent a *decision literal*
  - the graph contains  $l_1 \xrightarrow{c} l, \dots, l_n \xrightarrow{c} l$  iff  $c \stackrel{\text{def}}{=} \bigvee_{j=1}^n \neg l_j \vee l$  is the antecedent clause of  $l$

representation of the dependencies between literals in  $\mu$



# Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$c_6 : \neg A_5 \vee \neg A_6$$

$$c_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$c_8 : A_1 \vee A_8$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

...

# Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$c_6 : \neg A_5 \vee \neg A_6$$

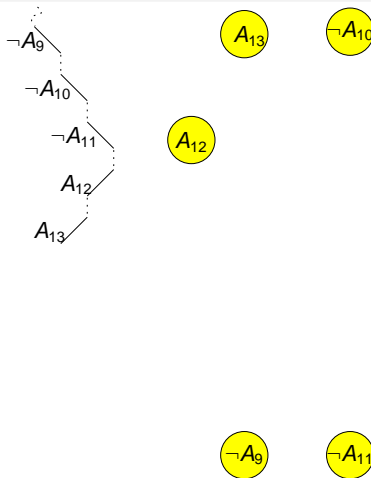
$$c_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$c_8 : A_1 \vee A_8$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

...

$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \dots\}$   
(initial assignment)



# Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$c_6 : \neg A_5 \vee \neg A_6$$

$$c_7 : A_1 \vee A_7 \vee \neg A_{12} \quad \checkmark$$

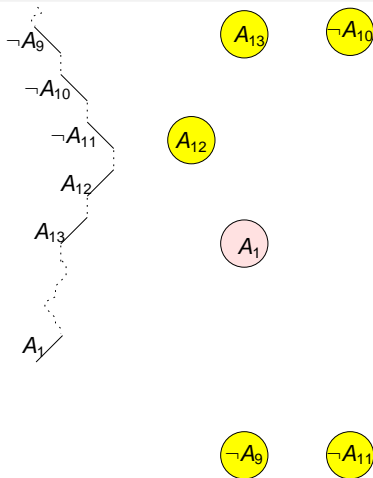
$$c_8 : A_1 \vee A_8 \quad \checkmark$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

...

$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1\}$

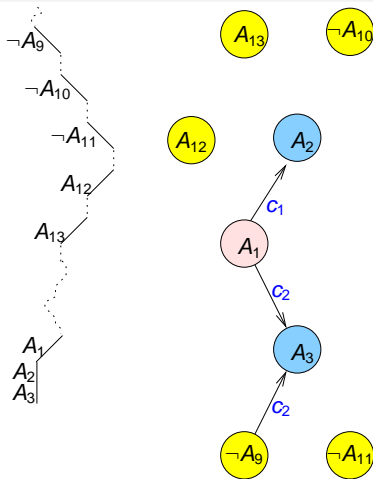
... (branch on  $A_1$ )





# Example

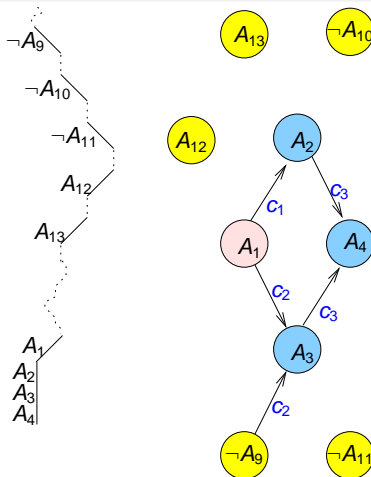
- $c_1 : \neg A_1 \vee A_2$  ✓  
 $c_2 : \neg A_1 \vee A_3 \vee A_9$  ✓  
 $c_3 : \neg A_2 \vee \neg A_3 \vee A_4$   
 $c_4 : \neg A_4 \vee A_5 \vee A_{10}$   
 $c_5 : \neg A_4 \vee A_6 \vee A_{11}$   
 $c_6 : \neg A_5 \vee \neg A_6$   
 $c_7 : A_1 \vee A_7 \vee \neg A_{12}$  ✓  
 $c_8 : A_1 \vee A_8$  ✓  
 $c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$   
 ...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1, A_2, A_3\}$   
 (unit  $A_2, A_3$ )

# Example

- $C_1 : \neg A_1 \vee A_2 \quad \checkmark$   
 $C_2 : \neg A_1 \vee A_3 \vee A_9 \quad \checkmark$   
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4 \quad \checkmark$   
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$   
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$   
 $C_6 : \neg A_5 \vee \neg A_6$   
 $C_7 : A_1 \vee A_7 \vee \neg A_{12} \quad \checkmark$   
 $C_8 : A_1 \vee A_8 \quad \checkmark$   
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$   
 ...



$\{ \dots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \dots, A_1, A_2, A_3, A_4 \}$   
 (unit  $A_4$ )



# State-of-the-art backjumping and learning

- Idea: when a branch  $\mu$  fails,
  - (i) *conflict analysis*: find the *conflict set*  $\eta \subseteq \mu$  by generating the *conflict clause*  $C \stackrel{\text{def}}{=} \neg\eta$  via resolution from the falsified clause
  - (ii) *learning*: add the conflict clause  $C$  to the clause set
  - (iii) *backjumping*: backtrack to the highest branching point s.t. the stack contains all-but-one literals in  $\eta$ , and then unit-propagate the unassigned literal on  $C$ 
    - $\implies$  may climb up to many decision levels in the stack
- if  $\eta (\neg C)$  entirely assigned at level 0, then return unsat

# Conflict analysis: build a conflict clause by resolution

1.  $C :=$  falsified clause (*conflicting clause*)
2. repeat
  - (i) resolve the current clause  $C$  with the antecedent clause of the last unit-propagated literal  $l$  in  $C$until  $C$  verifies some given termination criteria

# Conflict analysis: build a conflict clause by resolution

1.  $C :=$  falsified clause (*conflicting clause*)
2. repeat
  - (i) resolve the current clause  $C$  with the antecedent clause of the last unit-propagated literal  $l$  in  $C$
 until  $C$  verifies some given termination criteria

criterion: **decision**

...until  $C$  contains only decision literals

$$\begin{array}{c}
 \text{Conflicting cl.} \\
 \overline{A_4} \vee A_6 \vee A_{11} \quad \overline{A_5} \vee \overline{A_6} \quad (A_6) \\
 \hline
 \overline{A_4} \vee \overline{A_5} \vee A_{11} \quad (A_5) \\
 \hline
 \overline{A_4} \vee A_5 \vee A_{10} \quad \overline{A_4} \vee A_{10} \vee A_{11} \quad (A_4) \\
 \hline
 \overline{A_2} \vee \overline{A_3} \vee A_4 \quad \overline{A_2} \vee \overline{A_3} \vee A_{10} \vee A_{11} \quad (A_3) \\
 \hline
 \overline{A_1} \vee A_3 \vee A_9 \quad \overline{A_2} \vee \overline{A_1} \vee A_9 \vee A_{10} \vee A_{11} \quad (A_2) \\
 \hline
 \overline{A_1} \vee A_2 \quad \overline{A_1} \vee A_9 \vee A_{10} \vee A_{11}
 \end{array}$$

# Conflict analysis: build a conflict clause by resolution

1.  $C :=$  falsified clause (*conflicting clause*)
2. repeat
  - (i) resolve the current clause  $C$  with the antecedent clause of the last unit-propagated literal  $l$  in  $C$
 until  $C$  verifies some given termination criteria

criterion: 1st UIP

... until  $C$  contains only one literal assigned at current decision level  
(1st UIP)

$$\begin{array}{c}
 \text{Conflicting cl.} \\
 \overline{A_4} \vee A_5 \vee A_{10} \quad \overline{A_4} \vee A_6 \vee A_{11} \quad \overbrace{\overline{A_5} \vee \overline{A_6}} \\
 \hline
 \overline{A_4} \vee \overline{A_5} \vee A_{11} \quad (A_6) \\
 \hline
 \overline{A_4} \vee A_{10} \vee A_{11} \quad (A_5) \\
 \hline
 \underbrace{\overline{A_4}}_{\text{1st UIP}} \vee A_{10} \vee A_{11}
 \end{array}$$

# Conflict analysis: build a conflict clause by resolution

1.  $C :=$  falsified clause (*conflicting clause*)
2. repeat
  - (i) resolve the current clause  $C$  with the antecedent clause of the last unit-propagated literal  $l$  in  $C$until  $C$  verifies some given termination criteria

## Note:

Equivalent to finding a partition in the implication graph of  $\mu$  with all decision literals on one side and the conflict on the other.

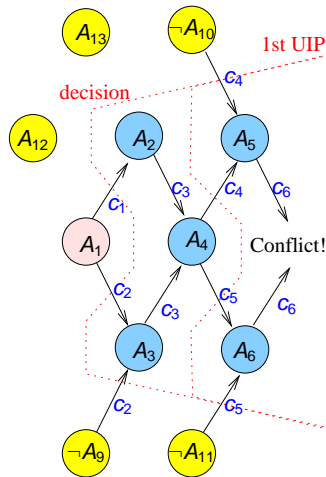
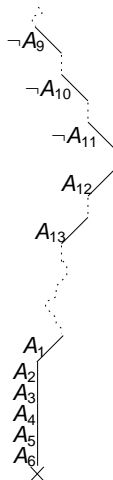
## Note

$\varphi \models C$ , so that  $C$  can be safely added to  $C$ .



# Example

- $C_1 : \neg A_1 \vee A_2$  ✓  
 $C_2 : \neg A_1 \vee A_3 \vee A_9$  ✓  
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4$  ✓  
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$  ✓  
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$  ✓  
 $C_6 : \neg A_5 \vee \neg A_6$  ✗  
 $C_7 : A_1 \vee A_7 \vee \neg A_{12}$  ✓  
 $C_8 : A_1 \vee A_8$  ✓  
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$  ✓  
 ...



$\Rightarrow$  Conflict set:  $\{\neg A_{10}, \neg A_{11}, A_4\}$ , learn  $c_{10} := A_{10} \vee A_{11} \vee \neg A_4$

# Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$c_6 : \neg A_5 \vee \neg A_6$$

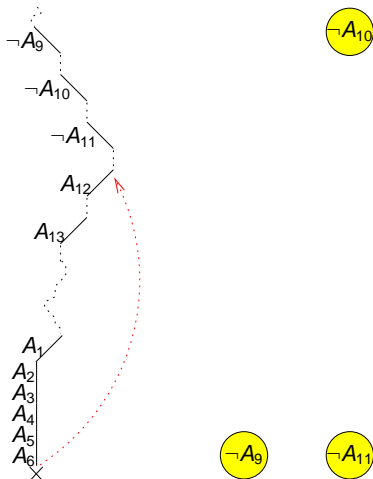
$$c_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$c_8 : A_1 \vee A_8$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

$$c_{10} : A_{10} \vee A_{11} \vee \neg A_4$$

...



$\Rightarrow$  backtrack up to  $A_{11} \Rightarrow \{..., \neg A_9, \neg A_{10}, \neg A_{11}\}$

# Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10} \quad \checkmark$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11} \quad \checkmark$$

$$c_6 : \neg A_5 \vee \neg A_6$$

$$c_7 : A_1 \vee A_7 \vee \neg A_{12}$$

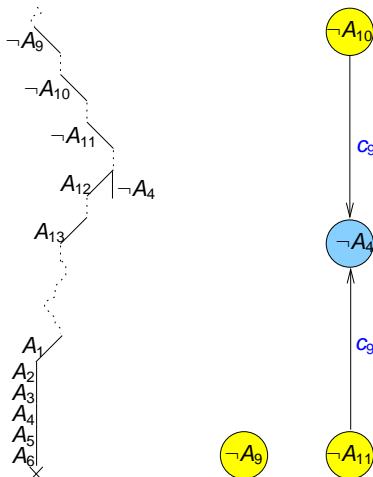
$$c_8 : A_1 \vee A_8$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

$$c_{10} : A_{10} \vee A_{11} \vee \neg A_4 \quad \checkmark$$

...

$\Rightarrow$  unit propagate  $\neg A_4 \Rightarrow \{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_4\}...$



## Learning – example

$$C_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$C_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$C_6 : \neg A_5 \vee \neg A_6$$

$$C_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$C_8 : A_1 \vee A_8$$

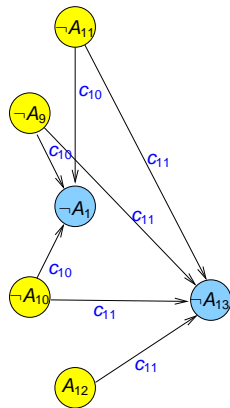
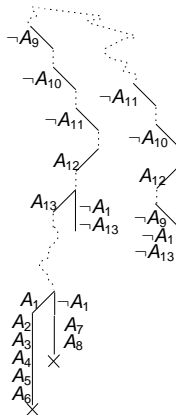
$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

$$C_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$$

$$C_{11} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_{12} \vee \neg A_{13} \checkmark$$

...

$\Rightarrow$  Unit:  $\{\neg A_1, \neg A_{13}\}$



# State-of-the-art backjumping and learning: intuitions

- *Backjumping*: allows for climbing up to many decision levels in the stack
  - intuition: “go back to the oldest decision where you’d have done something different if only you had known  $C$ ”  
⇒ may avoid lots of redundant search
- *Learning*: in future branches, when all-but-one literals in  $\eta$  are assigned, the remaining literal is assigned to false by unit-propagation:
  - intuition: “when you’re about to repeat the mistake, do the opposite of the last step”  
⇒ avoids finding the same conflict again

# Drawbacks of Learning

- Learning prunes drastically the search.
- Problem: may cause a blowup in space
  - ⇒ techniques to drop learned clauses when necessary
    - according to their size
    - according to their **activity**.

## Definition

A clause is currently *active* if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).

## Property (see, e.g., [Nieuwenhuis et al. JACM'06])

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

⇒ **CDCL solvers require polynomial space**

# Drawbacks of Learning

- Learning prunes drastically the search.
- Problem: may cause a blowup in space
  - ⇒ techniques to drop learned clauses when necessary
    - according to their size
    - according to their **activity**.

## Definition

A clause is currently **active** if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).

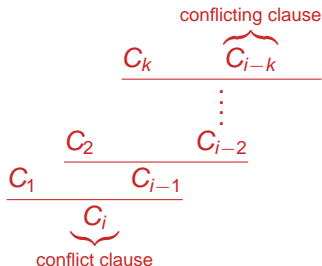
Property (see, e.g., [Nieuwenhuis et al. JACM'06])

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

⇒ **CDCL solvers require polynomial space**

# Building Proofs of Unsatisfiability in CDCL SAT solvers

- recall: each conflict clause  $C_i$  learned is computed from the conflicting clause  $C_{i-k}$  by backward resolving with the antecedent clause of one literal



- each resolution (sub)proof can be easily tracked:

$k \quad i-k \rightarrow i-k-1$

$\dots$

$2 \quad i-2 \rightarrow i-1$

$1 \quad i-1 \rightarrow i$



# Building Proofs of Unsatisfiability in CDCL SAT solvers

- ... in particular, if  $\varphi$  is unsatisfiable, the last step produces “false” as conflict clause:

$$\begin{array}{c}
 \text{conflicting clause} \\
 \overline{C_k \quad C_{i-k}} \\
 \vdots \\
 \overline{C_2 \quad C_{i-2}} \\
 \overline{C_1 \quad C_{i-1}} \\
 \perp
 \end{array}$$

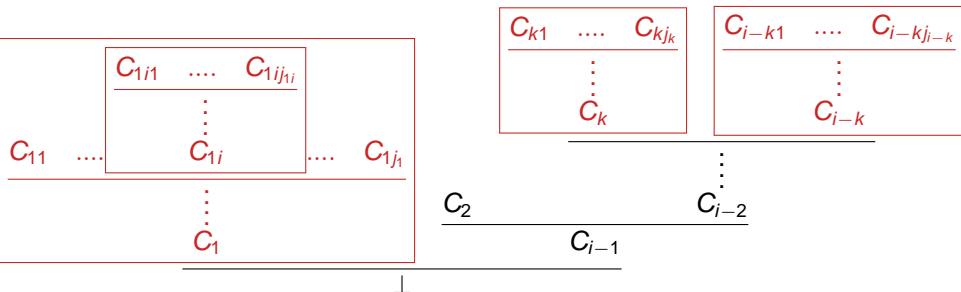
- note:  $C_1 = l$ ,  $C_{i-1} = \neg l$  for some literal  $l$
- $C_1, \dots, C_k$ , and  $C_{i-k}$  can be original or learned clauses...

# Building Proofs of Unsatisfiability in CDCL SAT solvers

Starting from the previous proof of unsatisfiability, repeat recursively:

- for every **learned** leaf clause  $C_i$ , substitute  $C_i$  with the resolution proof generating it

until all leaf clauses are original clauses



⇒ we obtain a resolution proof of unsatisfiability for (a subset of) the clauses in  $\varphi$

# SAT under assumptions: $SAT(\varphi, \{l_1, \dots, l_n\})$

- Many SAT solvers allow for solving a CNF formula  $\varphi$  **under a set of assumption literals**  $\mathcal{A} \stackrel{\text{def}}{=} \{l_1, \dots, l_n\}$  :  $SAT(\varphi, \{l_1, \dots, l_n\})$ 
  - $SAT(\varphi, \{l_1, \dots, l_n\})$ : same result as  $SAT(\varphi \wedge \bigwedge_{i=1}^n l_i)$
- idea:
  - $l_1, \dots, l_n$  “decided” at decision level 0 before starting the search
  - if backjump to level 0 on  $C \stackrel{\text{def}}{=}} \neg \eta$  s.t.  $\eta \subseteq \mathcal{A}$ , then return unsat
  - if the “decision” strategy for conflict analysis is used, then  $\eta$  is the **subset of assumptions causing the inconsistency**
- incremental** calls  $SAT(\varphi, \mathcal{A}_1), \dots, SAT(\varphi, \mathcal{A}_n)$  without restarting
  - stack-based interface for  $\{l_1, \dots, l_n\}$
  - reuse of search (e.g. learned clauses) from call to call

# Outline

1 From DL to SMT ...

2 **Efficient SMT solving**

- Modern SAT solvers
- **Modern SMT solvers**
- Theory Solvers and their combination

3 Beyond Solving: advanced SMT functionalities

- Proofs and unsatisfiable cores
- Interpolants
- All-SMT

4 ... and back to DL?

# Modern “lazy” SMT( $\mathcal{T}$ ) solvers

## A prominent “lazy” approach

- a *SAT solver* is used to enumerate truth assignments  $\mu_i$  for (the Boolean abstraction of) the input formula  $\varphi$
- a theory-specific solver  *$\mathcal{T}$ -solver* checks the  $\mathcal{T}$ -consistency of the *set of  $\mathcal{T}$ -literals* corresponding to each assignment

Note: wrt. DPLL for modal logic, no nesting of DPLL calls

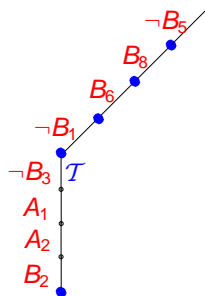
- Built on top of modern CDCL solvers
  - benefit for free from all modern CDCL techniques (e.g., Boolean preprocessing, backjumping & learning, restarts,...)
  - benefit for free from all state-of-the-art data structures and implementation tricks (e.g., two-watched literals,...)
- Many techniques to maximize the benefits of integration (see [Sebastiani, JSAT'07])
- Many lazy SMT tools available  
( *Ario*, *Barcelogic*, *CVC3*, *MathSAT*, *SATeen*, *Yices*, *Z3*, ... )

# Basic schema: example

$\varphi =$   
 $C_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $C_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $C_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $C_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $C_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $C_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $C_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$

*true, false*

$\varphi^p =$   
 $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$



$$\begin{aligned}
 \mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
 \mu &= \{\neg(3v_1 - v_3 \leq 6), \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
 &\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
 \end{aligned}$$

$\Rightarrow$  inconsistent in  $\mathcal{LA}(\mathbb{Q}) \Rightarrow$  backtrack



# Early pruning: example

$$\begin{aligned}\varphi = & \{ \neg(2v_2 - v_3 > 2) \vee A_1 \} \wedge \\ & \{ \neg A_2 \vee (2v_1 - 4v_5 > 3) \} \wedge \\ & \{ (3v_1 - 2v_2 \leq 3) \vee A_2 \} \wedge \\ & \{ \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (3v_1 - 2v_2 \leq 3) \} \wedge \\ & \{ (v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 \}.\end{aligned}$$

$$\begin{aligned}\varphi^p = & \{ \neg B_1 \vee A_1 \} \wedge \\ & \{ \neg A_2 \vee B_2 \} \wedge \\ & \{ B_3 \vee A_2 \} \wedge \\ & \{ \neg B_4 \vee \neg B_5 \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee B_3 \} \wedge \\ & \{ B_6 \vee B_7 \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee B_8 \vee A_2 \}.\end{aligned}$$

- Suppose it is built the intermediate assignment:

$$\mu'^p = \neg B_1 \wedge \neg A_2 \wedge B_3 \wedge \neg B_5.$$

corresponding to the following set of  $\mathcal{T}$ -literals

$$\mu' = \neg(2v_2 - v_3 > 2) \wedge \neg A_2 \wedge (3v_1 - 2v_2 \leq 3) \wedge \neg(3v_1 - v_3 \leq 6).$$

- If  $\mathcal{T}$ -solver is invoked on  $\mu'$ , then it returns unsat, and DPLL backtracks **without exploring any extension of  $\mu'$** .



# Early pruning: remark

## Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to  $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo $\mu_4, \mu_3, \mu_2$	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$  Desirable features of  $\mathcal{T}$ -solvers:

- incrementality*:  $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$  reuses computation of  $\mathcal{T}\text{-solver}(\mu_1)$  without restarting from scratch
- backtrackability (resetability)*:  $\mathcal{T}\text{-solver}$  can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$   $\mathcal{T}$ -solver requires a **stack-based interface**

# $\mathcal{T}$ -Propagation [Armando et al.'99; Audemard et al.'02; Ganzinger et al. '04]

- strictly related to early pruning
- important property of  $\mathcal{T}$ -solver.
  - **$\mathcal{T}$ -deduction**: *when a partial assignment  $\mu$  is  $\mathcal{T}$ -satisfiable,  $\mathcal{T}$ -solver may be able to return also an assignment  $\eta$  to some unassigned atom occurring in  $\varphi$  s.t.  $\mu \models_{\mathcal{T}} \eta$ .*

E.g., if  $(v_1 - v_3 \geq 2)$ ,  $(v_2 = v_3) \in \mu$  and  $(v_1 - v_2 < 1) \notin \mu$  and occurs in  $\varphi$ , then  $\mathcal{T}$ -solver can derive  $\neg(v_1 - v_2 < 1)$  from  $\mu$ .

- If so:
    - the literal  $\eta$  is then unit-propagated;
    - optionally, a  **$\mathcal{T}$ -deduction clause  $C := \neg\mu' \vee \eta$**  can be learned,  $\mu'$  being the subset of  $\mu$  which caused the deduction ( $\mu' \models_{\mathcal{T}} \eta$ )
- E.g.,  $\neg(v_1 - v_3 \geq 2) \vee \neg(v_2 = v_3) \vee \neg(v_1 - v_2 < 1)$

$\Rightarrow$  may prune drastically the search

# $\mathcal{T}$ -propagation: example

 $\varphi =$ 

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$ 

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

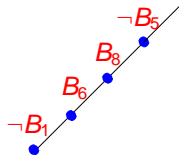
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



*true, false*

$\Rightarrow$  propagate  $\neg B_3$  [and learn the deduction clause  $B_5 \vee B_1 \vee \neg B_3$ ]

# $\mathcal{T}$ -propagation: example

 $\varphi =$ 

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$ 

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

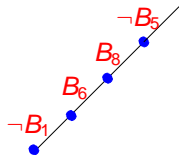
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



*true, false*

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2)\}$$

$$\models_{\mathcal{LA}(\mathbb{Q})} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}$$

$\Rightarrow$  propagate  $\neg B_3$  [and learn the deduction clause  $B_5 \vee B_1 \vee \neg B_3$ ]

# $\mathcal{T}$ -propagation: example

 $\varphi =$ 

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$ 

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

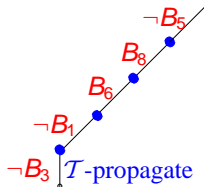
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



*true, false*

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2)\}$$

$$\models_{\mathcal{LA}(\mathbb{Q})} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}$$

$\Rightarrow$  propagate  $\neg B_3$  [and learn the deduction clause  $B_5 \vee B_1 \vee \neg B_3$ ]

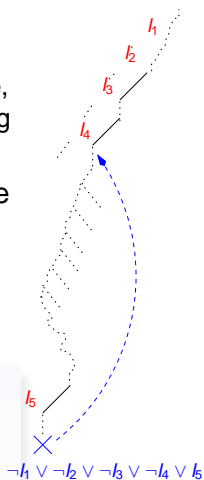
# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning

[Horrocks et al. '98; Wolfman&Weld, '99; Audemard et al.02]

- Similar to Boolean backjumping & learning
- important property of  $\mathcal{T}$ -solver:
  - **extraction of  $\mathcal{T}$ -conflict sets**: if  $\mu$  is  $\mathcal{T}$ -unsatisfiable, then  $\mathcal{T}$ -solver( $\mu$ ) returns the subset  $\eta$  of  $\mu$  causing the  $\mathcal{T}$ -inconsistency of  $\mu$  ( $\mathcal{T}$ -conflict set)
- If so, the  **$\mathcal{T}$ -conflict clause**  $C := \neg\eta$  is used to drive the backjumping & learning mechanism of DPLL  $\implies$  lots of search saved
- **the less redundant is  $\eta$ , the more search is saved**

## Definition: $\mathcal{T}$ -lemmas

Both  $\mathcal{T}$ -deduction clauses and  $\mathcal{T}$ -conflict clauses are called  **$\mathcal{T}$ -lemmas** since they are valid in  $\mathcal{T}$



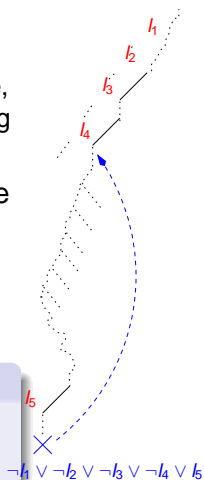
# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning

[Horrocks et al. '98; Wolfman&Weld, '99; Audemard et al.02]

- Similar to Boolean backjumping & learning
- important property of  $\mathcal{T}$ -solver:
  - **extraction of  $\mathcal{T}$ -conflict sets**: if  $\mu$  is  $\mathcal{T}$ -unsatisfiable, then  $\mathcal{T}$ -solver( $\mu$ ) returns the subset  $\eta$  of  $\mu$  causing the  $\mathcal{T}$ -inconsistency of  $\mu$  ( $\mathcal{T}$ -conflict set)
- If so, the  **$\mathcal{T}$ -conflict clause**  $C := \neg\eta$  is used to drive the backjumping & learning mechanism of DPLL  $\implies$  lots of search saved
- **the less redundant is  $\eta$ , the more search is saved**

## Definition: $\mathcal{T}$ -lemmas

Both  $\mathcal{T}$ -deduction clauses and  $\mathcal{T}$ -conflict clauses are called  **$\mathcal{T}$ -lemmas** since they are valid in  $\mathcal{T}$



# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

 $\varphi =$ 

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

$$c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$$

 $\varphi^p =$ 

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

$$B_3 \vee A_2$$

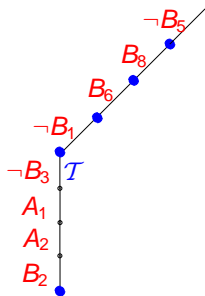
$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$

$$B_5 \vee \neg B_8 \vee \neg B_2$$



*true*, *false*

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$



# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

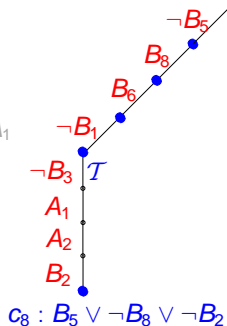
 $\varphi =$ 

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$   
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$

true, false

 $\varphi^p =$ 

- $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$   
 $B_5 \vee \neg B_8 \vee \neg B_2$



$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$

# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

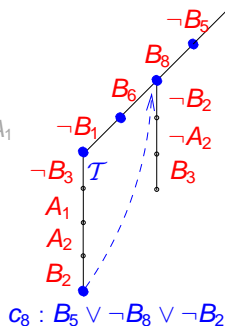
 $\varphi =$ 

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$   
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$

true, false

 $\varphi^p =$ 

- $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$   
 $B_5 \vee \neg B_8 \vee \neg B_2$



$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$

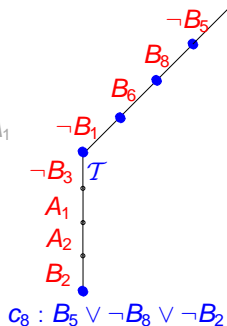
# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example (2)

 $\varphi =$ 

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$   
 $c'_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
*true, false*

 $\varphi^p =$ 

- $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$   
 $B_5 \vee \neg B_8 \vee B_1$   
 $B_5 \vee \neg B_8 \vee \neg B_2$


 $c_8$ : theory conflicting clause

$$\begin{array}{c}
 \overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_2} \quad \overbrace{\neg A_2 \vee B_2}^{c_3} \quad (B_2) \quad \overbrace{B_3 \vee A_2}^{c_T} \\
 \hline
 B_5 \vee \neg B_8 \vee \neg A_2 \quad \neg A_2 \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \\
 \hline
 B_5 \vee \neg B_8 \vee B_3 \quad \neg A_2 \quad (B_3) \\
 \hline
 \overbrace{B_5 \vee \neg B_8 \vee B_1}^{c'_8}
 \end{array}$$

 $c'_8$ : mixed Boolean+theory conflict clause

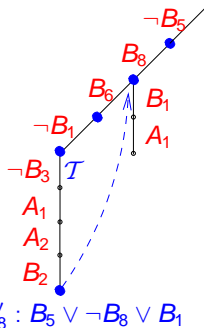
# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example (2)

 $\varphi =$ 

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$   
 $c'_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
*true, false*

 $\varphi^p =$ 

- $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$   
 $B_5 \vee \neg B_8 \vee B_1$   
 $B_5 \vee \neg B_8 \vee \neg B_2$


 $c'_8 : B_5 \vee \neg B_8 \vee B_1$ 
 $c_8$ : theory conflicting clause

$$\begin{array}{c}
 \overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_2} \quad \overbrace{\neg A_2 \vee B_2}^{c_2} \quad (B_2) \quad \overbrace{B_3 \vee A_2}^{c_3} \quad (\neg A_2) \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \quad (B_3) \\
 \hline
 B_5 \vee \neg B_8 \vee \neg A_2 \\
 \hline
 B_5 \vee \neg B_8 \vee B_3 \\
 \hline
 B_5 \vee \neg B_8 \vee B_1 \\
 \hline
 \end{array}$$

 $c'_8$ : mixed Boolean+theory conflict clause

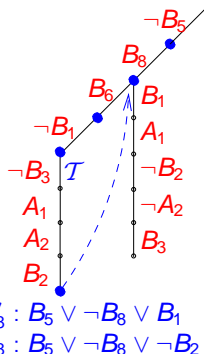
# $\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example (2)

 $\varphi =$ 

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$   
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$   
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$   
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$   
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$   
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$   
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$   
 $c'_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$   
*true, false*

 $\varphi^p =$ 

- $\neg B_1 \vee A_1$   
 $\neg A_2 \vee B_2$   
 $B_3 \vee A_2$   
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$   
 $A_1 \vee B_3$   
 $B_6 \vee B_7 \vee \neg A_1$   
 $A_1 \vee B_8 \vee A_2$   
 $B_5 \vee \neg B_8 \vee B_1$   
 $B_5 \vee \neg B_8 \vee \neg B_2$


 $c_8$ : theory conflicting clause

$$\begin{array}{c}
 \overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_2} \quad \overbrace{\neg A_2 \vee B_2}^{c_2} \quad (B_2) \quad \overbrace{B_3 \vee A_2}^{c_3} \quad (\neg A_2) \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \quad (B_3) \\
 \hline
 B_5 \vee \neg B_8 \vee \neg A_2 \\
 \hline
 B_5 \vee \neg B_8 \vee B_3 \\
 \hline
 B_5 \vee \neg B_8 \vee B_1 \\
 \hline
 c'_8: \text{mixed Boolean+theory conflict clause}
 \end{array}$$

 $c'_8$ : mixed Boolean+theory conflict clause

# Pure-literal filtering [Giunchiglia et al.'99; Audemard et al.'02]

## Property

If we have non-Boolean  $\mathcal{T}$ -atoms occurring only positively [negatively] **in the original formula**  $\varphi$  (learned clauses are not considered), we can drop every negative [positive] occurrence of them from the assignment to be checked by  $\mathcal{T}$ -solver (and from the  $\mathcal{T}$ -deducible ones).

- increases the chances of finding a model
- reduces the effort for the  $\mathcal{T}$ -solver
- eliminates unnecessary “nasty” negated literals  
(e.g. negative equalities like  $\neg(3v_1 - 9v_2 = 3)$  in  $\mathcal{LA}(\mathbb{Z})$  force splitting:  $(3v_1 - 9v_2 > 3) \vee (3v_1 - 9v_2 < 3)$ ).
- may weaken the effect of early pruning.

# Pure literal filtering: example

$$\begin{aligned}
 \varphi = & \{ \neg(2v_2 - v_3 > 2) \vee A_1 \} \wedge \\
 & \{ \neg A_2 \vee (2v_1 - 4v_5 > 3) \} \wedge \\
 & \{ (3v_1 - 2v_2 \leq 3) \vee A_2 \} \wedge \\
 & \{ \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 \} \wedge \\
 & \{ A_1 \vee (3v_1 - 2v_2 \leq 3) \} \wedge \\
 & \{ (v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 \} \wedge \\
 & \{ A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 \} \wedge \\
 & \{ (2v_2 - v_3 > 2) \vee \neg(3v_1 - 2v_2 \leq 3) \vee (3v_1 - v_3 \leq 6) \} \text{ \textit{learned}}
 \end{aligned}$$

$$\mu' = \{ \neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \leq 6) \}.$$

$\Rightarrow$  Sat:  $v_1 = v_2 = v_3 = 0, v_5 = -4/3$  is a solution

N.B.  $(3v_1 - v_3 \leq 6)$  “filtered out” from  $\mu'$  because it occurs only negatively in the original formula  $\varphi$

# Other optimization techniques

- *Preprocessing literals*
- *Static learning*
- *$T$ -deduced-literal filtering*
- *Ghost-literal filtering*
- *$T$ -solver layering*
- *$T$ -solver clustering*
- ...

(see [Sebastiani, JSAT'07])



# Outline

1 From DL to SMT ...

2 **Efficient SMT solving**

- Modern SAT solvers
- Modern SMT solvers
- **Theory Solvers and their combination**

3 Beyond Solving: advanced SMT functionalities

- Proofs and unsatisfiable cores
- Interpolants
- All-SMT

4 ... and back to DL?

# $\mathcal{T}$ -solvers for theories of interest I

- *Equality and uninterpreted functions ( $\mathcal{EUF}$ ):*

- EX:  $\{(x = y), (y = f(z)), \neg(g(x) = g(f(z)))\}$
- polynomial:  $O(n \cdot \log(n))$
- based of congruence closure data structures

[Detlefs et al JACM'05; Nieuwenhuis & Oliveras LPAR'03]

- *Difference logic ( $\mathcal{DL}$ ):*

- EX:  $\{(x - y \leq 0), (y - z \leq 4), (z - x \leq -5)\}$
- polynomial:  $O(n \cdot m)$
- variants of the Bellman-Ford shortest-path algorithm

[Nieuwenhuis & Oliveras CAV'05; Cotton & Maler SAT'06]

- *Linear arithmetic over the rationals ( $\mathcal{LA}(\mathbb{Q})$ ):*

- EX:  $\{(s_1 - s_2 \leq 5.2), (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0), \neg(s_1 = s_0)\}$
- polynomial
- variants of the simplex LP algorithm

[Dutertre & Demoura CAV'06]

# $\mathcal{T}$ -solvers for theories of interest II

- *Linear arithmetic over the integers* ( $\mathcal{LA}(\mathbb{Z})$ ):

- EX:  $\{(x := x_l + 2^{16}x_h), (x \geq 0), (x \leq 2^{16} - 1)\}$
- NP-complete
- combination of many techniques: simplex, branch&bound, cutting planes, ... [Dutertre & Demoura CAV'06, Griggio SAT'10,...]

- *Arrays*:

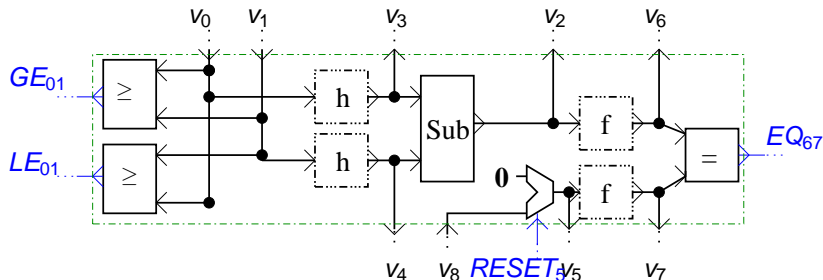
- EX:  $\{\neg(i = j), read(write(a, i, e), j) = read(a, j)\}$
- NP-complete
- congruence closure ( $\mathcal{EUF}$ ) plus on-the-fly instantiation of array's axioms [Detlefs et al JACM'05; Goel et al. SMT'08,...]

- *Bit vectors*:

- EX:  $\{(x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) \ll w_{[16]}[3 : 0]), \dots\}$
- NP-complete
- combination of rewriting & simplification techniques with either:
  - final encoding into  $\mathcal{LA}(\mathbb{Z})$  [Bruttomesso et. al CAVa'07;...]
  - final encoding into SAT [Brummaryer & Biere JSAT'09;...]

# Lazy SMT for combined theories: $SMT(\bigcup_i \mathcal{T}_i)$

*Problem: Many problems can be expressed as SMT problems only in combination of theories  $\bigcup_i \mathcal{T}_i$  —  $SMT(\bigcup_i \mathcal{T}_i)$*



$\mathcal{LA}(\mathbb{Z})$  :  $(GE_{01} \leftrightarrow (v_0 \geq v_1)) \wedge (LE_{01} \leftrightarrow (v_0 \leq v_1)) \wedge$

$\mathcal{EUF}$  :  $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge$

$\mathcal{LA}(\mathbb{Z})$  :  $(v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$

$\mathcal{EUF}$  or  $\mathcal{LA}(\mathbb{Z})$  :  $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge$

$\mathcal{EUF}$  :  $(v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$

$\mathcal{EUF}$  or  $\mathcal{LA}(\mathbb{Z})$  :  $(EQ_{67} \leftrightarrow (v_6 = v_7)) \wedge \dots$

# SMT( $\bigcup_i \mathcal{T}_i$ ) via Nelson-Oppen

## Main idea

Combine two or more  $\mathcal{T}_i$ -solvers into one  $(\bigcup_i \mathcal{T}_i)$ -solver via  
*Nelson-Oppen/Shostak (N.O.) combination procedure*

[Nelson&Oppen TOCL 79; Shostak JACM 84]

- based on the deduction and exchange of equalities between shared variables/terms (*interface equalities,  $e_{ij}s$* )
- important improvements and evolutions

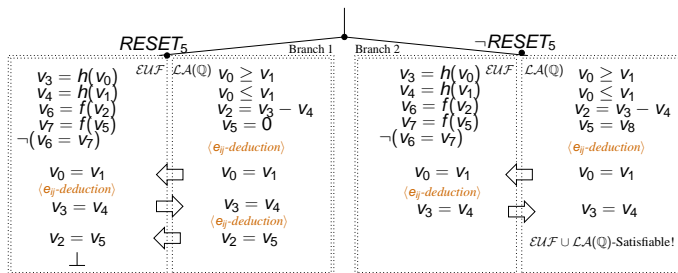
[Ruess & Shankar LICS01; Barrett et al., FroCoS'02; Detlefs et al. JACM05,...]

# N.O.: example

$\mathcal{EUF}$  :  $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$

$\mathcal{LA}(\mathbb{Q})$  :  $(v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$

Both :  $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).$



$\mathcal{EUF}$ -conflict :  $((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_5)) \rightarrow \perp$

$\mathcal{LA}(\mathbb{Q})$ -deduction :  $((v_2 = v_3 - v_4) \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5)$

$\mathcal{EUF}$ -deduction :  $((v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4)$

$\mathcal{LA}(\mathbb{Q})$ -deduction :  $((v_0 \geq v_1) \wedge (v_0 \leq v_1)) \rightarrow (v_0 = v_1)$

$\Rightarrow$

$\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ -conflict :  $((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_3 - v_4) \wedge (v_5 = v_8) \wedge (v_0 = v_1) \wedge (v_3 = v_4) \wedge (v_2 = v_5)) \rightarrow \perp$

# $SMT(\bigcup_i \mathcal{T}_i)$ via Delayed Theory Combination (DTC)

## Main idea

Delegate to DPLL part/most of the (possibly very expensive) reasoning effort on interface equalities previously due to the  $\mathcal{T}_i$ -solvers ( $e_{ij}$ -deduction, case-split). [Bozzano et al. CAV05, Inf.&Comp. 06; LPAR06 ]

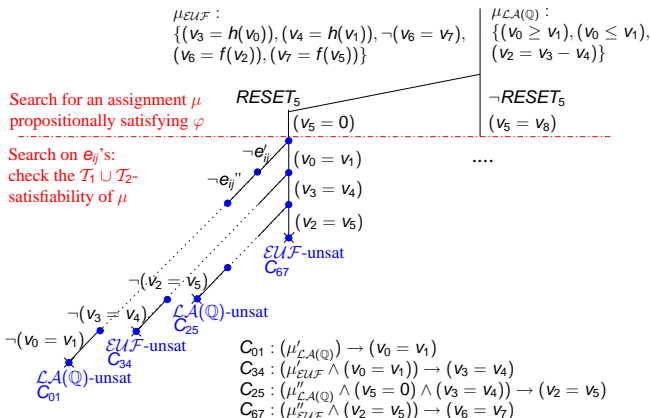
- based on Boolean reasoning on interface equalities via DPLL (+  $\mathcal{T}$ -propagation)
- important improvements and evolutions  
[Dutertre&deMoura SMT-COMP'06; Barret et al. LPAR'06; DeMoura&Bjorner, SMT'07;...]
- feature wrt N.O. [Bozzano et al. CAV05, Inf.&Comp.06, LPAR06, AI&Math09]
  - do not require (possibly expensive) deduction capabilities from  $\mathcal{T}_i$ -solvers
  - [ with non-convex theories ] case-splits on  $e_{ij}$ 's handled by DPLL
  - generate  $\mathcal{T}_i$ -lemmas with interface equalities  
 $\implies$  backjumping & learning from  $e_{ij}$ -reasoning

# DTC: example

$$\mathcal{EUF} : (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$$

$$\mathcal{LA}(\mathbb{Q}) : (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$$

$$Both : (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).$$





# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities**
  - Proofs and unsatisfiable cores
  - Interpolants
  - All-SMT
- 4 ... and back to DL?

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities**
  - Proofs and unsatisfiable cores**
  - Interpolants
  - All-SMT
- 4 ... and back to DL?

# Building (Resolution) Proofs of $\mathcal{T}$ -Unsatisfiability

## Resolution proof of $\mathcal{T}$ -unsatisfiability

Very similar to building proofs with plain SAT:

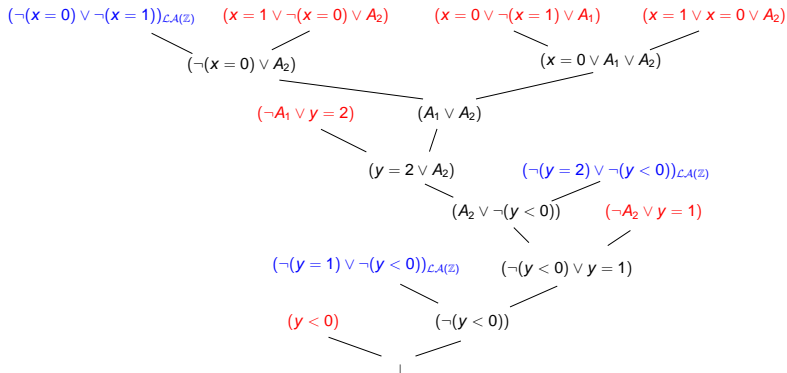
- resolution proofs whose leaves are original clauses and  $\mathcal{T}$ -lemmas returned by the  $\mathcal{T}$ -solver (i.e.,  $\mathcal{T}$ -conflict and  $\mathcal{T}$ -deduction clauses)
- built by backward traversal of implication graphs, as in DPLL
- Sub-proofs of  $\mathcal{T}$ -lemmas can be built in some  $\mathcal{T}$ -specific deduction framework if requested

Important for:

- certifying  $\mathcal{T}$ -unsatisfiability results
- computing unsatisfiable cores
- computing interpolants

# Building Proofs of $\mathcal{T}$ -Unsatisfiability: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



relevant original clauses, irrelevant original clauses,  $\mathcal{T}$ -lemmas

# Extraction of $\mathcal{T}$ -unsatisfiable cores

## The problem

Given a  $\mathcal{T}$ -unsatisfiable set of clauses, extract from it a (possibly small/minimal/minimum)  $\mathcal{T}$ -unsatisfiable subset ( $\mathcal{T}$ -unsatisfiable core)

- wide literature in SAT
- Despite some implementations, substantially no literature on the topic for SMT (apart from [Cimatti et al. SAT'07] )
- We recognize three approaches:
  - *Proof-based* approach (CVClite, MathSAT):  
byproduct of finding a resolution proof
  - *Assumption-based* approach (Yices):  
use extra variables labeling clauses, as in the plain Boolean case
  - *Lemma-Lifting* approach [Cimatti et al. SAT'07]:  
use an external (possibly-optimized) Boolean unsat-core extractor

# The proof-based approach to $\mathcal{T}$ -unsat cores

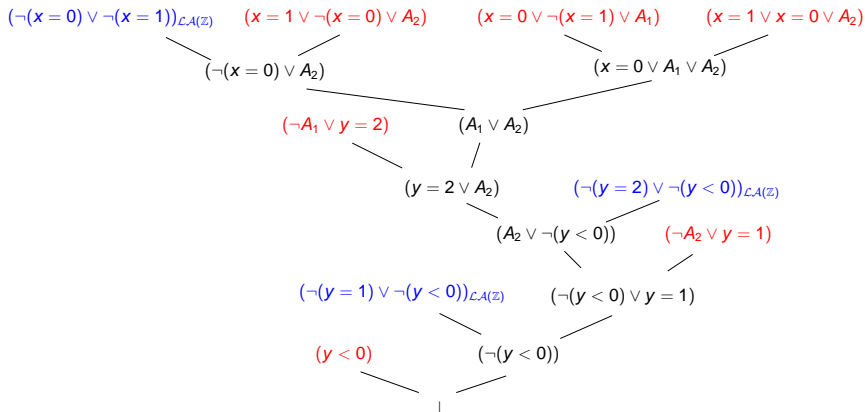
Idea (adapted from [Zhang & Malik SAT'03])

Unsatisfiable core of  $\varphi$ :

- in SAT: the set of leaf clauses of a resolution proof of unsatisfiability of  $\varphi$
- in  $\text{SMT}(\mathcal{T})$ : the set of leaf clauses of a resolution proof of  $\mathcal{T}$ -unsatisfiability of  $\varphi$ , minus the  $\mathcal{T}$ -lemmas
- implemented in MathSAT and CVC3

# The proof-based approach to $\mathcal{T}$ -unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



# The assumption-based approach to $\mathcal{T}$ -unsat cores

Let  $\varphi$  be  $\bigwedge_{i=1}^n C_i$  s.t.  $\varphi$  inconsistent.

Idea (adapted from [Lynce & Silva SAT'04])

- 1 each clause  $C_i$  in  $\varphi$  is substituted by  $\neg S_i \vee C_i$ , s.t.  $S_i$  fresh “selector” variable
- 2 the resulting formula is checked for **satisfiability under the assumption of all  $S_i$ 's**
- 3 final conflict clause at dec. level 0:  $\bigvee_j \neg S_j$   
 $\implies \{C_j\}_j$  is the unsat core

- extends straightforwardly to  $\text{SMT}(\mathcal{T})$ .
- implemented in YICES and in MATHSAT



# The assumption-based approach to $\mathcal{T}$ -unsat cores: Example

$$\begin{aligned}
 & (S_1 \rightarrow (x = 0 \vee \neg(x = 1) \vee A_1)) \wedge (S_2 \rightarrow (x = 0 \vee x = 1 \vee A_2)) \wedge \\
 & (S_3 \rightarrow (\neg(x = 0) \vee x = 1 \vee A_2)) \wedge (S_4 \rightarrow (\neg A_2 \vee y = 1)) \wedge \\
 & (S_5 \rightarrow (\neg A_1 \vee x + y > 3)) \wedge (S_6 \rightarrow y < 0) \wedge \\
 & (S_7 \rightarrow (A_2 \vee x - y = 4)) \wedge (S_8 \rightarrow (y = 2 \vee \neg A_1)) \wedge (S_9 \rightarrow x \geq 0)
 \end{aligned}$$

Conflict analysis (Yices 1.0.6) returns:

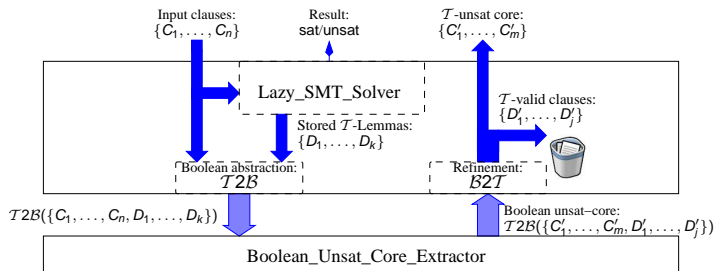
$$\neg S_1 \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8,$$

corresponding to the unsat core in red.

# The lemma-lifting approach to $\mathcal{T}$ -unsat cores

Idea [Cimatti et al. SAT'07]

- (i) The  $\mathcal{T}$ -lemmas  $D_i$  are valid in  $\mathcal{T}$
- (ii) The conjunction of  $\varphi$  with all the  $\mathcal{T}$ -lemmas  $D_1, \dots, D_k$  is propositionally unsatisfiable:  $\mathcal{T}2\mathcal{B}(\varphi \wedge \bigwedge_{i=1}^n D_i) \models \perp$ .



- Implemented in MathSAT [Cimatti et al. SAT'07]
- interfaces with an external Boolean Unsat-core Extractor

# The lemma-lifting approach to $\mathcal{T}$ -unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$

- 1 The SMT solver generates the following set of  $\mathcal{LA}(\mathbb{Z})$ -lemmas:

$$\{(\neg(x = 1) \vee \neg(x = 0)), (\neg(y = 2) \vee \neg(y < 0)), (\neg(y = 1) \vee \neg(y < 0))\}.$$

- 2 The following formula is passed to the external Boolean core extractor

$$(B_0 \vee \neg B_1 \vee A_1) \wedge (B_0 \vee B_1 \vee A_2) \wedge (\neg B_0 \vee B_1 \vee A_2) \wedge \\ (\neg A_2 \vee B_2) \wedge (\neg A_1 \vee B_3) \wedge B_4 \wedge (A_2 \vee B_5) \wedge (B_6 \vee \neg A_1) \wedge B_7 \wedge \\ (\neg B_1 \vee \neg B_0) \wedge (\neg B_6 \vee \neg B_4) \wedge (\neg B_2 \vee \neg B_4)$$

which returns the unsat core in red.

- 3 The unsat-core is mapped back, the three  $\mathcal{T}$ -lemmas are removed  $\implies$  the final  $\mathcal{T}$ -unsat core (in red above).

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities**
  - Proofs and unsatisfiable cores
  - Interpolants**
  - All-SMT
- 4 ... and back to DL?

# Computing (Craig) Interpolants in SMT

## Craig Interpolant

Given an ordered pair  $(A, B)$  of formulas such that  $A \wedge B \models_{\mathcal{T}} \perp$ , a *Craig interpolant* is a formula  $I$  s.t.:

- a)  $A \models_{\mathcal{T}} I$ ,
- b)  $I \wedge B \models_{\mathcal{T}} \perp$ ,
- c)  $I \preceq A$  and  $I \preceq B$ .

“ $I \preceq A$ ” meaning that all uninterpreted (in  $\mathcal{T}$ ) symbols in  $I$  occur in  $A$ .

- Very important in many FV applications
- A few works presented
  - Afaik, very few tools publicly available: *FOCI* [McMillan, TCS'05], *CLP-prover* [Rybalchenko & Sofronie-Stokkemanns, VMCAI'07], *MathSAT* [Cimatti et al. TACAS'08, TOCL'10]
  - Others (*Zap* [Ball et al. LPAR'05], *Lifter* [Kroening & Weissenbaker FMCAD'07]) are not available

# A General Algorithm [Pudlak JSL'97; McMillan CAV'03,TCS'05,CAV'06]

## Algorithm: Interpolant generation for $\text{SMT}(\mathcal{T})$

- (i) Generate a resolution proof of  $\mathcal{T}$ -unsatisfiability  $\mathcal{P}$  for  $A \wedge B$ .
- (ii) **Foreach  $\mathcal{T}$ -lemma  $\neg\eta$  in  $\mathcal{P}$ , generate an interpolant  $I_{\neg\eta}$  for  $(\eta \setminus B, \eta \downarrow B)$ .**
- (iii) For every original leaf clause  $C$  in  $\mathcal{P}$ , set  $I_C \stackrel{\text{def}}{=} C \downarrow B$  if  $C \in A$ , and  $I_C \stackrel{\text{def}}{=} \top$  if  $C \in B$ .
- (iv) For every inner node  $C$  of  $\mathcal{P}$  obtained by resolution from  $C_1 \stackrel{\text{def}}{=} p \vee \phi_1$  and  $C_2 \stackrel{\text{def}}{=} \neg p \vee \phi_2$ , set  $I_C \stackrel{\text{def}}{=} I_{C_1} \vee I_{C_2}$  if  $p$  does not occur in  $B$ , and  $I_C \stackrel{\text{def}}{=} I_{C_1} \wedge I_{C_2}$  otherwise.
- (v) Output  $I_{\perp}$  as an interpolant for  $(A, B)$ .

“ $\eta \setminus B$ ” [resp. “ $\eta \downarrow B$ ”] is the set of literals in  $\eta$  whose atoms do not [resp. do] occur in  $B$ .

- optimized versions for the purely-propositional case
- row 2. only place where  $\mathcal{T}$  comes in to play

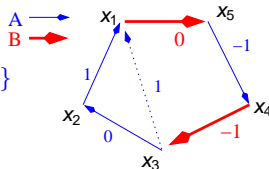
$\implies$  Reduced to the problem of finding an interpolant for two **sets of  $\mathcal{T}$ -literals**

# Example: interpolation algorithms for difference logic

- A graph-based algorithm [Cimatti et al. TACAS'07, TOCL'10]

$$A \stackrel{\text{def}}{=} \overbrace{\{(0 \leq x_1 - x_2 + 1), (0 \leq x_2 - x_3), (0 \leq x_4 - x_5 - 1)\}}^{\text{Chord: } (0 \leq x_1 - x_3 + 1)}$$

$$B \stackrel{\text{def}}{=} \{(0 \leq x_5 - x_1), (0 \leq x_3 - x_4 - 1)\}.$$



$\Rightarrow$  Interpolant:  $(0 \leq x_1 - x_3 + 1) \wedge (0 \leq x_4 - x_5 - 1)$  (still in D.L.)

- An inference-based algorithm [McMillan TCS'05]

$$\begin{array}{c}
 \frac{(0 \leq x_1 - x_2 + 1) \quad (0 \leq x_2 - x_3)}{\text{COMB} \quad (0 \leq x_1 - x_3 + 1)} \quad (0 \leq x_4 - x_5 - 1) \\
 \frac{\text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)}{\text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)} \quad (0 \leq 0) \\
 \frac{\text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)}{\text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)} \quad (0 \leq 0)
 \end{array}$$

$\Rightarrow$  Interpolant:  $(0 \leq x_1 - x_3 + x_4 - x_5)$  (not in D.L., and weaker).

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities**
  - Proofs and unsatisfiable cores
  - Interpolants
  - All-SMT**
- 4 ... and back to DL?



# All-SAT/All-SMT

- **All-SAT**: enumerate all truth assignments satisfying  $\varphi$
- **All-SMT**: enumerate all  **$\mathcal{T}$ -satisfiable** truth assignments propositionally satisfying  $\varphi$ 
  - used in FV for computing **predicate abstraction**:

$$\text{PredAbs}_{\{P_1, \dots, P_n\}}(\varphi) \stackrel{\text{def}}{=} \exists \mathbf{v}. ( \varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v}) )$$

Idea [Lahiri et al, CAV'06; Cavada et al. FMCAD'07]

Each time a  $\mathcal{T}$ -satisfiable assignment  $\{l_1, \dots, l_n\}$  is found, print it and perform conflict-driven backjumping using  $\bigvee_i \neg l_i$  as conflicting clause.

## Remark

To guarantee correctness, completeness & termination it suffices to keep each clause  $\bigvee_i \neg l_i$  only as long as it is active in the implication graph (see, e.g., [Lahiri et al., CAV'06; Nieuwenhuis et al. JACM'06])

$\implies$  All-SAT/All-SMT requires storing a polynomial amount of clauses.

# All-SAT/All-SMT

- **All-SAT**: enumerate all truth assignments satisfying  $\varphi$
- **All-SMT**: enumerate all  **$\mathcal{T}$ -satisfiable** truth assignments propositionally satisfying  $\varphi$ 
  - used in FV for computing **predicate abstraction**:

$$\text{PredAbs}_{\{P_1, \dots, P_n\}}(\varphi) \stackrel{\text{def}}{=} \exists \mathbf{v}. ( \varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v}) )$$

Idea [Lahiri et al, CAV'06; Cavada et al. FMCAD'07]

Each time a  $\mathcal{T}$ -satisfiable assignment  $\{l_1, \dots, l_n\}$  is found, print it and perform conflict-driven backjumping using  $\bigvee_i \neg l_i$  as conflicting clause.

## Remark

To guarantee correctness, completeness & termination it suffices to keep each clause  $\bigvee_i \neg l_i$  only as long as it is active in the implication graph (see, e.g., [Lahiri et al., CAV'06; Nieuwenhuis et al. JACM'06])

$\implies$  All-SAT/All-SMT requires storing a polynomial amount of clauses.

# All-SAT/All-SMT

- **All-SAT**: enumerate all truth assignments satisfying  $\varphi$
- **All-SMT**: enumerate all  **$\mathcal{T}$ -satisfiable** truth assignments propositionally satisfying  $\varphi$ 
  - used in FV for computing **predicate abstraction**:

$$\text{PredAbs}_{\{P_1, \dots, P_n\}}(\varphi) \stackrel{\text{def}}{=} \exists \mathbf{v}. ( \varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v}) )$$

Idea [Lahiri et al, CAV'06; Cavada et al. FMCAD'07]

Each time a  $\mathcal{T}$ -satisfiable assignment  $\{l_1, \dots, l_n\}$  is found, print it and perform conflict-driven backjumping using  $\bigvee_i \neg l_i$  as conflicting clause.

## Remark

To guarantee correctness, completeness & termination it suffices to keep each clause  $\bigvee_i \neg l_i$  only as long as it is active in the implication graph (see, e.g., [Lahiri et al., CAV'06; Nieuwenhuis et al. JACM'06])

$\implies$  **All-SAT/All-SMT** requires storing a polynomial amount of clauses.

# Outline

- 1 From DL to SMT ...
- 2 Efficient SMT solving
  - Modern SAT solvers
  - Modern SMT solvers
  - Theory Solvers and their combination
- 3 Beyond Solving: advanced SMT functionalities
  - Proofs and unsatisfiable cores
  - Interpolants
  - All-SMT
- 4 ... and back to DL?

# Back to DL?

## Conjecture

SAT & SMT technology may be exploited for DL reasoning

# Back to DL?

## Conjecture

SAT & SMT technology may be exploited for DL reasoning

Some work: (joint work with Michele Vescovi)

- satisfiability in  $K_m/\mathcal{ALC}$  via SAT encoding  
[Sebastiani & Vescovi SAT'06;JAIR'09]
  - against all odds, competitive!
- Axiom pinpointing in  $\mathcal{EL}^+$  via Horn-SAT and All-SMT  
[Sebastiani & Vescovi CADE'09]

# Back to DL?

## Conjecture

SAT & SMT technology may be exploited for DL reasoning

Some work: (joint work with Michele Vescovi)

- satisfiability in  $K_m/\mathcal{ALC}$  via SAT encoding  
[Sebastiani & Vescovi SAT'06;JAIR'09]
  - against all odds, competitive!
- Axiom pinpointing in  $\mathcal{EL}^+$  via Horn-SAT and All-SMT  
[Sebastiani & Vescovi CADE'09]

# Back to DL?

## Conjecture

SAT & SMT technology may be exploited for DL reasoning

Some work: (joint work with Michele Vescovi)

- satisfiability in  $K_m/\mathcal{ALC}$  via SAT encoding  
[Sebastiani & Vescovi SAT'06;JAIR'09]
  - against all odds, competitive!
- Axiom pinpointing in  $\mathcal{EL}^+$  via Horn-SAT and All-SMT  
[Sebastiani & Vescovi CADE'09]



# Back to DL?

## Conjecture

SAT & SMT technology may be exploited for DL reasoning

Some work: (joint work with Michele Vescovi)

- satisfiability in  $K_m/\mathcal{ALC}$  via SAT encoding  
[Sebastiani & Vescovi SAT'06;JAIR'09]
  - against all odds, competitive!
- Axiom pinpointing in  $\mathcal{EL}^+$  via Horn-SAT and All-SMT  
[Sebastiani & Vescovi CADE'09]

# Motivation: Axiom pinpointing in $\mathcal{EL}^+$

- $\mathcal{EL}^+$  is able to represent widely-used (and often huge) **bio-medical ontologies** like: GENEONTOLOGY, NCI, (the majority of) GALEN and SNOMED-CT
- Operations:
  - infer subsumption relations from an ontology  $\mathcal{T}$  (*classification*)
  - identify the reasons of these relations (*axiom pinpointing*):  
Find minimal sets of axioms (*MinAs*) in  $\mathcal{T}$  which generate a subsumption relation  $C \sqsubseteq D$

## Example: Debugging ontologies

Find minimal sets of axioms (MinAs) in SNOMED-CT which generates the undesired fact that Amputation-of-Finger is a sub-concept of Amputation-of-Arm.

# The logic $\mathcal{EL}^+$

## Concept definitions

	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$X \sqcap Y$	$X^{\mathcal{I}} \cap Y^{\mathcal{I}}$
existential restriction	$\exists r.X$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in X^{\mathcal{I}}\}$

where  $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ , for every concept name  $C$

and  $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ , for every role name  $r$

## Axioms

	Syntax	Semantics
general concept inclusion (GCI)	$X \sqsubseteq Y$	$X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$
role inclusion (RI)	$r_1 \circ \dots \circ r_n \sqsubseteq s$	$r_1^{\mathcal{I}} \circ \dots \circ r_n^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

# Normal Form

It is convenient to establish and work with the following *normal form* of the input TBox (ontology):

## Normal Form for an $\mathcal{EL}^+$ TBox

$$C_1 \sqcap \dots \sqcap C_k \sqsubseteq D \quad k \geq 1$$

$$C \sqsubseteq \exists r.D$$

$$\exists r.C \sqsubseteq D$$

$$r_1 \circ \dots \circ r_n \sqsubseteq s \quad n \geq 1$$

with  $C_1, \dots, C_k, D$  concept names, and  $r_1, \dots, r_n, s$  role names

- Every  $\mathcal{EL}^+$  TBox  $\mathcal{T}$  can be turned into a normalized TBox  $\mathcal{T}'$  which is a *conservative extension* of  $\mathcal{T}$
- $\mathcal{T}'$  is linear w.r.t.  $\mathcal{T}$  and can be computed in linear time

# Reasoning tasks in $\mathcal{EL}^+$

**Concept Subsumption** (polynomial) Given two concepts  $X$  and  $Y$ ,  $Y$  *subsumes*  $X$  wrt. the TBox  $\mathcal{T}$ , written  $X \sqsubseteq_{\mathcal{T}} Y$ , if  $X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$  for every model  $\mathcal{I}$  of  $\mathcal{T}$

**Classification** (polynomial) Find all the possible pairwise concept subsumption relations for all the concept names of a given TBox

**one-nMinA** (polynomial) Find a (possibly non-minimal) set  $\mathcal{S} \subseteq \mathcal{T}$  (written *nMinA*) for  $C \sqsubseteq_{\mathcal{T}} D$  s.t.  $C \sqsubseteq_{\mathcal{S}} D$

**one-MinA** (polynomial) Single *axiom pinpointing* (one-MinA): find one *minimal* axiom set (written *MinA*) for a given subsumption relation  $C \sqsubseteq_{\mathcal{T}} D$

**all-MinAs** (output-exponential) Enumerate all the possible MinAs for the given subsumption relation  $C \sqsubseteq_{\mathcal{T}} D$

# Concept Subsumption and Classification

Polynomial-time algorithm [Baader et al. IJCAI'05, KI'07] for the *classification* of a normalized  $\mathcal{EL}^+$  TBox  $\mathcal{T}$ :

## Completion rules of concept subsumption algorithm

Sub. assertions ( $\in \mathcal{A}$ )	TBox's axiom ( $\in \mathcal{T}$ )	Ass. added to $\mathcal{A}$
$X \sqsubseteq C_1, X \sqsubseteq C_2$	$C_1 \sqcap C_2 \sqsubseteq D$	$X \sqsubseteq D$
$X \sqsubseteq C$	$C \sqsubseteq \exists r.D$	$X \sqsubseteq \exists r.D$
$X \sqsubseteq \exists r.E, E \sqsubseteq C$	$\exists r.C \sqsubseteq D$	$X \sqsubseteq D$
$X \sqsubseteq \exists r.D$	$r \sqsubseteq s$	$X \sqsubseteq \exists s.D$
$X \sqsubseteq \exists r_1.E_1, E_1 \sqsubseteq \exists r_2.D$	$r_1 \circ r_2 \sqsubseteq s$	$X \sqsubseteq \exists s.D$

- The algorithm starts with an initial set of assertions  $\mathcal{A} = \{a_i \in \mathcal{T} \mid a_i \text{ is a GCI}\} \cup \{C \sqsubseteq C\} \cup \{C \sqsubseteq \top\}$
- Applies a rule only if it extends  $\mathcal{A}$
- Stops when no more rules are applicable

# Axiom Pinpointing in $\mathcal{EL}^+$ , Baader et al's algorithms

Build a *propositional formula*  $\phi^{C \sqsubseteq D}$  on  $\{s_{[ax_j]} \mid ax_j \text{ axiom of } \mathcal{T}\}$ , whose minimal valuations are all the MinAs for  $C \sqsubseteq_{\mathcal{T}} D$

- $\phi^{C \sqsubseteq D}$  is *worst-case exponential* wrt. the size of the ontology  
[Baader et al. KI'07]

[Baader et al. KI'07] Modify the classification algorithm to compute **one** nMinA for  $C \sqsubseteq D$

- then *Linear-search/Binary-search* minimization algorithm to obtain **one MinA** (succeed on GALEN, not on SNOMED)

[Baader et al. KR-MED'08] Devised a novel algorithm which, extracting *reachability modules* from the ontology, succeed in finding MinAs on SNOMED-CT

...

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: $\phi_{\mathcal{T}}$

- let  $\mathcal{A}$  be the set of GCI or RI axioms or inferred assertions from  $\mathcal{T}$  (e.g., those obtained by Baader et al's classification)
- let  $\mathcal{EL}^+2sat(a_i)$  be the following definite Horn clause:

$\mathcal{EL}^+2sat(a_i)$

$a_i$	$\mathcal{EL}^+2sat(a_i)$
$C_1 \sqcap C_2 \sqsubseteq D$	$p_{[C_1]} \wedge p_{[C_2]} \rightarrow p_{[D]}$
$C \sqsubseteq \exists r.D$	$p_{[C]} \rightarrow p_{[\exists r.D]}$
$\exists r.C \sqsubseteq D$	$p_{[\exists r.C]} \rightarrow p_{[D]}$

s.t. each  $p_{[X]}$  is uniquely-associated to each normalized concept  $X$

## Proposition

$C \sqsubseteq_{\mathcal{T}} D$  iff  $\phi_{\mathcal{T}} \stackrel{\text{def}}{=} \bigwedge_{a_i \in \mathcal{A}} \mathcal{EL}^+2sat(a_i)$  is **unsatisfiable under the assumptions**  $\{p_{[C]}, \neg p_{[D]}\}$



# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: $\phi_{\mathcal{T}}^{all}$

- let  $s_{[a_i]}$  be a *selector variable* univocally associated to  $a_i$ ,  $\forall a_i \in \mathcal{A}$
- We build (**only once!**) definite Horn formula  $\phi_{\mathcal{T}}^{all}$ 
  - $\phi_{\mathcal{T}}^{all}$  contains the definite Horn clauses:

$$s_{[a_i]} \rightarrow \mathcal{EL}^+ 2sat(a_i)$$

for each  $a_i \in \mathcal{A}$

- $\phi_{\mathcal{T}}^{all}$  contains the definite Horn clause:

$$(s_{[a_1]} [\wedge s_{[a_2]}] \wedge s_{[ax]}) \rightarrow s_{[a]}$$

for each  $a_1, [a_2,]a \in \mathcal{A}$ ,  $ax \in \mathcal{T}$  s.t.  $\{a_1, [a_2,]ax\} \implies a$  is an instantiation of a completion rule

- $\phi_{\mathcal{T}}^{all}$  **polynomial**:  $|\phi_{\mathcal{T}}^{all}| = \Theta(|\mathcal{A}|^2 \cdot |\mathcal{T}|)$

## Remark

$\phi_{\mathcal{T}}^{all}$  encodes **all the possible ways** of inferring every subsumption relation from  $\mathcal{T}$

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: one-MinA

## Proposition

- $C \sqsubseteq_S D$  iff  $\phi_{\mathcal{T}}^{all}$  is unsatisfiable under the assumptions  $\{p[C], \neg p[D]\} \cup \{s[ax_i] \mid ax_i \in S\}$ , for every  $S \subseteq \mathcal{T}$

- Note: it is possible to “select” any subset  $S$  of  $\mathcal{T}$

- $\phi_{\mathcal{T}}^{all}$  is Horn

⇒ each satisfiability check requires only one run of unit-propagation

⇒ linear in the number of clauses involved in the unit-propagation

- DPLL returns a **conflict clause**  $\bigvee_{ax_j \in S} \neg s[ax_j] \vee \neg p[C] \vee p[D]$   
s.t.  $S \subseteq \mathcal{T}$  is an **nMinA** for  $C \sqsubseteq_{\mathcal{T}} D$ , i.e.  $C \sqsubseteq_S D$

⇒ is refined into a **MinA** from a linear iteration of the process  
(a SAT-based variant of [Baader et al. KI'07])

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: one-MinA

## Proposition

- $C \sqsubseteq_{\mathcal{S}} D$  iff  $\phi_{\mathcal{T}}^{all}$  is unsatisfiable under the assumptions  $\{p[C], \neg p[D]\} \cup \{s[ax_i] \mid ax_i \in \mathcal{S}\}$ , for every  $\mathcal{S} \subseteq \mathcal{T}$

- Note: it is possible to “select” any subset  $\mathcal{S}$  of  $\mathcal{T}$
- $\phi_{\mathcal{T}}^{all}$  is Horn

⇒ each satisfiability check requires only one run of unit-propagation

⇒ linear in the number of clauses involved in the unit-propagation

- DPLL returns a **conflict clause**  $\bigvee_{ax_j \in \mathcal{S}} \neg s[ax_j] \vee \neg p[C] \vee p[D]$   
s.t.  $\mathcal{S} \subseteq \mathcal{T}$  is an **nMinA** for  $C \sqsubseteq_{\mathcal{T}} D$ , i.e.  $C \sqsubseteq_{\mathcal{S}} D$

⇒ is refined into a **MinA** from a linear iteration of the process  
(a SAT-based variant of [Baader et al. KI'07])

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: one-MinA

## Proposition

- $C \sqsubseteq_{\mathcal{S}} D$  iff  $\phi_{\mathcal{T}}^{all}$  is unsatisfiable under the assumptions  $\{p[C], \neg p[D]\} \cup \{s_{[ax_i]} | ax_i \in \mathcal{S}\}$ , for every  $\mathcal{S} \subseteq \mathcal{T}$

- Note: it is possible to “select” any subset  $\mathcal{S}$  of  $\mathcal{T}$
- $\phi_{\mathcal{T}}^{all}$  is Horn

⇒ each satisfiability check requires only one run of unit-propagation

⇒ linear in the number of clauses involved in the unit-propagation

- DPLL returns a **conflict clause**  $\bigvee_{ax_j \in \mathcal{S}} \neg s_{[ax_j]} \vee \neg p[C] \vee p[D]$   
s.t.  $\mathcal{S} \subseteq \mathcal{T}$  is an **nMinA** for  $C \sqsubseteq_{\mathcal{T}} D$ , i.e.  $C \sqsubseteq_{\mathcal{S}} D$

⇒ is refined into a **MinA** from a linear iteration of the process  
(a SAT-based variant of [Baader et al. KI'07])

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: all-MinAs

## Idea

Adapt the all-SMT technique [Lahiri et al. CAV'06] to enumerate all the minimal countermodels of  $\phi_{\mathcal{T}}^{all}$  under the assumptions  $\{p_{[C]}, \neg p_{[D]}\}$

## Basic All-SMT schema:

1. A CDCL solver enumerates a complete set of assignments  $\mu_k$  on  $\{s_{[ax_i]} \mid ax_i \in \mathcal{T}\} \cup \{p_{[C]}, p_{[D]}\}$  satisfying a formula  $\varphi$  (initially  $\top$ )
2. The one-MinA (DPLL-based) procedure is the  $\mathcal{T}$ -solver:
  - checks the *unsatisfiability of  $\phi_{\mathcal{T}}^{all}$  assuming  $\mu_k \cup \{p_{[C]}, \neg p_{[D]}\}$*
  - if unsatisfiable, produces iteratively a minimal axiom set
3. The (negation of the) computed MinA/model  $\mu_k$  is used as conflicting clause by the top-level solver

- some optimizations described in [Sebastiani & Vescovi CADE'09]

# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: remarks

- The size of  $\phi_{\mathcal{T}}^{all}$  and the time to compute it is worst-case *polynomial*
- Once loaded,  $\phi_{\mathcal{T}}^{all}$  can be reused
  - for different queries  $C \sqsubseteq D$
  - for different sub-ontologies  $\mathcal{S} \subseteq \mathcal{T}$

## remark

- all-SMT require polynomial space
- can be used to enumerate MinAs up to a time-out

# Preliminary Experimental Results [Sebastiani & Vescovi CADE'09]

## Classification and Subsumption:

Ontology	NOTGALEN	GENEONT.	NCI	FULLGALEN	SNOMED09
# prim. concepts	2748	20465	27652	23135	310075
# orig. axioms	4379	20466	46800	36544	310025
# norm. axioms	8740	29897	46800	81340	857459
# role names	413	1	50	949	62
# role axioms	442	1	0	1014	12
Size (var#/cls#)					
$\phi_{\mathcal{T}}$	5.4e3/1.8e4	2.2e4/4.2e4	3.2e4/4.7e4	4.8e4/7.3e5	5.3e5/8.4e6
$\phi_{\mathcal{T}}^{one}$	2.3e4/2.7e4	5.5e4/5.4e4	7.8e4/4.7e4	7.3e5/1.4e6	8.4e6/1.6e7
$\phi_{\mathcal{T}}^{all}$ ( $\rho_0$ )	1.7e5/2.2e5	2.1e5/2.6e5	2.9e5/3.0e5	5.3e6/1.2e7	2.6e7/8.4e7
Encode time					
$\phi_{\mathcal{T}}$	0.65	2.37	2.98	35.28	3753.04
$\phi_{\mathcal{T}}^{one}$	2.06	4.15	6.19	68.94	4069.84
$\phi_{\mathcal{T}}^{all}$ ( $\rho_0$ )	1.17	1.56	2.37	178.41	198476.59
Load time					
$\phi_{\mathcal{T}}$	0.11	0.37	1.01	1.93	21.16
$\phi_{\mathcal{T}}^{one}$	0.18	0.55	1.17	5.95	59.88
Subsumpt. ( $10^5$ )					
$\phi_{\mathcal{T}}$	0.00002	0.00002	0.00003	0.00003	0.00004
$\phi_{\mathcal{T}}^{one}$	0.00003	0.00002	0.00003	0.00004	0.00008

# Preliminary Experimental Results [Sebastiani & Vescovi CADE'09]

## Axiom Pinpointing:

Ontology	NOTGAL.	GENEO.	NCI	FULLGAL.	SNOMED09
nMinA $\phi_{\mathcal{T}}^{one}$ (on 5000)	0.00012	0.00027	0.00042	0.00369	0.05938
MinA $\phi_{\mathcal{T}}^{one}$ (on 100)					
- Load time	0.175	0.387	0.694	6.443	63.324
- Extract time	0.066	0.082	0.214	0.303	3.280
- DPLL Search time	0.004	0.004	0.002	0.010	0.093
MinA $\phi_{\mathcal{T}(po)}^{all}$ (on 100)					
- Load time	1.061	1.385	1.370	39.551	150.697
- DPLL Search time	0.023	0.027	0.036	0.331	0.351
allMinA $\phi_{\mathcal{T}(po)}^{all}$ (on 30)					
- 50% #MinA- time	1- 1.50	1- 1.76	4- 1.79	3- 53.40	15- 274.70
- 90% #MinA- time	2- 1.59	4- 2.11	6- 1.86	9- 63.61	32- 493.61
- 100% #MinA- time	2- 1.64	8- 2.79	9- 2.89	15- 150.95	40- 588.33



# Axiom pinpointing in $\mathcal{EL}^+$ via All-SMT: Discussion

- Encoding phase expensive (but done only once forever)
- Formula loading can be done only once per session
- Once the formula is loaded, DPLL time for solving concept subsumption and one nMinA is negligible
- The all-MinAs procedure allows for enumerating MinAs for SNOMED-CT ad libitum (remarkably MinAs are mainly found in the very first part of the search)
- significant improvements in a recent version (unpublished yet)

# Conclusions

- SAT: CDCL solvers very powerful
  - key ingredient for SMT solvers
  - may be exploited in DL reasoning as well?
- SMT solvers
  - $SMT = SAT + \mathcal{T}\text{-solver}$ : idea coming for DL
  - since then, lots of improvements and techniques
  - can ideas from SMT be mapped back to DL?
- SAT & SMT: not only solving
  - lots of advanced functionalities: proofs, unsat cores, interpolants, (cost-minimization, ...)
  - can they be exploited for DL reasoning tools?

# Conclusions

- SAT: CDCL solvers very powerful
  - key ingredient for SMT solvers
  - may be exploited in DL reasoning as well?
- SMT solvers
  - **SMT = SAT + *T-solver***: idea coming for DL
  - since then, lots of improvements and techniques
  - can ideas from SMT be mapped back to DL?
- SAT & SMT: not only solving
  - lots of advanced functionalities: proofs, unsat cores, interpolants, (cost-minimization, ...)
  - can they be exploited for DL reasoning tools?

# Conclusions

- SAT: CDCL solvers very powerful
  - key ingredient for SMT solvers
  - may be exploited in DL reasoning as well?
- SMT solvers
  - **SMT = SAT + *T-solver***: idea coming for DL
  - since then, lots of improvements and techniques
  - can ideas from SMT be mapped back to DL?
- SAT & SMT: not only solving
  - lots of advanced functionalities: proofs, unsat cores, interpolants, (cost-minimization, ...)
  - can they be exploited for DL reasoning tools?

# Links

- a course on SAT & SMT:

[http://disi.unitn.it/~rseba/DIDATTICA/SAT\\_BASED10/](http://disi.unitn.it/~rseba/DIDATTICA/SAT_BASED10/)

- survey papers:

- Lintao Zhang and Sharad Malik, "The Quest for Efficient Boolean Satisfiability Solvers."  
Proc. CAV'02, LNCS, number 2404, Springer, 2002.
- Roberto Sebastiani: "Lazy Satisfiability Modulo Theories".  
Journal on Satisfiability, Boolean Modeling and Computation, JSAT. Vol. 3, 2007. Pag 141–224, ©IOS Press.
- Roberto Sebastiani, Armando Tacchella "SAT Techniques for Modal and Description Logics". Part II, Chapter 25, The Handbook of Satisfiability. 2009. ©IOS press.
- Clark Barrett, Roberto Sebastiani, Sanjit Seshia, Cesare Tinelli "Satisfiability Modulo Theories". Part II, Chapter 26, The Handbook of Satisfiability. 2009. ©IOS press.



©Warner Bros. Inc.