

# From satisfaction to optimization, and beyond

## SAT-based guided problem solving

Daniel Le Berre

*joint work with Mutsunori Banbara, Tiago de Lima, Jean-Marie Lagniez,  
Valentin Montmirail, Stéphanie Roussel, Naoyuki Tamura, Takehide Soh*

CNRS, Université d'Artois, FRANCE  
[{leberre}@crl.univ-artois.fr](mailto:{leberre}@crl.univ-artois.fr)

SAT+SMT school, IIT Bombay, India, 10 December 2019



# Purpose of this talk

- ▶ Using SAT solvers are black boxes
- ▶ Importance of the interaction with the solver
- ▶ When encodings are too large

# The SAT problem: textbook definition

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : Is there an assignment of the  $n$  variables that satisfies all those clauses?

# The SAT problem: textbook definition

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : Is there an assignment of the  $n$  variables that satisfies all those clauses?

## Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For  $C_1$ , the answer is **yes**, for  $C_2$  the answer is **no**

$$C_1 \models \neg(a \wedge \neg c) = \neg a \vee c$$

# The SAT problem solver: practical point of view 1/3

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else answer UNSAT.

# The SAT problem solver: practical point of view 1/3

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else answer UNSAT.

## Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For  $C_1$ , one answer is  $\{a, b, c\}$ , for  $C_2$  the answer is UNSAT.

# The SAT problem solver: practical point of view 1/3

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else answer UNSAT.

## Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For  $C_1$ , one answer is  $\{a, b, c\}$ , for  $C_2$  the answer is UNSAT.

SAT answers can be checked: trusted model oracle

# The SAT problem solver: practical point of view 2/3

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else provide a subset of  $C$  which cannot be satisfied.

# The SAT problem solver: practical point of view 2/3

## Definition

Input : A set of clauses  $C$  built from a propositional language with  $n$  variables.

Output : If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else provide a subset of  $C$  which cannot be satisfied.

## Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For  $C_1$ , one answer is  $\{a, b, c\}$ , for  $C_2$  the answer is  $C_2$

# The SAT problem solver: practical point of view 2/3

## Definition

**Input :** A set of clauses  $C$  built from a propositional language with  $n$  variables.

**Output :** If there is an assignment of the  $n$  variables that satisfies all those clauses, provide such assignment, else provide a subset of  $C$  which cannot be satisfied.

## Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For  $C_1$ , one answer is  $\{a, b, c\}$ , for  $C_2$  the answer is  $C_2$

UNSAT core may explain inconsistency if much smaller than  $C$ :  
informative UNSAT oracle

# The SAT problem solver: practical point of view 3/3

## Definition

Allow the solver to decide the satisfiability of a formula with:

- ▶ increasing number of constraints
- ▶ provided some “assumptions” are satisfied

# The SAT problem solver: practical point of view 3/3

## Definition

Allow the solver to decide the satisfiability of a formula with:

- ▶ increasing number of constraints
- ▶ provided some “assumptions” are satisfied

## Example

$$C = \{s_1 \vee \neg a \vee b, s_1 \vee \neg b \vee c, s_2 \vee a, s_2 \vee \neg c\}$$

$$C_1 \equiv C \wedge \neg s_1 \wedge s_2$$

$$C_2 \equiv C \wedge \neg s_1 \wedge \neg s_2$$

# The SAT problem solver: practical point of view 3/3

## Definition

Allow the solver to decide the satisfiability of a formula with:

- ▶ increasing number of constraints
- ▶ provided some “assumptions” are satisfied

## Example

$$C = \{s_1 \vee \neg a \vee b, s_1 \vee \neg b \vee c, s_2 \vee a, s_2 \vee \neg c\}$$

$$C_1 \equiv C \wedge \neg s_1 \wedge s_2$$

$$C_2 \equiv C \wedge \neg s_1 \wedge \neg s_2$$

The solver is considered as a **stateful system**: as long as the constraints are satisfiable, learned clauses can be kept: **incremental SAT oracle**

# How to solve MaxSat MinUnsat with SAT?

- ▶ Associate to each clause a weight (penalty)  $w_i$  taken into account if the clause is violated: **Soft clauses  $S$** .
- ▶ Special weight ( $\infty$ ) for clauses that cannot be violated: **hard clauses  $H$**

## Definition (Partial Weighted MaxSat)

Find a model  $M$  of  $H$  that minimizes  $weight(M, S)$  such that:

- ▶  $weight(M, (c_i, w_i)) = 0$  if  $M$  satisfies  $c_i$ , else  $w_i$ .
- ▶  $weight(M, S) = \sum_{wc \in S} weight(M, wc)$

Simply called MaxSAT if  $k = 1$  and  $H = \emptyset$

# How to solve MaxSat MinUnsat with SAT?

- ▶ Associate to each clause a weight (penalty)  $w_i$  taken into account if the clause is violated: **Soft clauses  $S$ .**  
 $(\neg a \vee b, 6) \wedge (\neg b \vee c, 8)$
- ▶ Special weight ( $\infty$ ) for clauses that cannot be violated: **hard clauses  $H$**

## Definition (Partial Weighted MaxSat)

Find a model  $M$  of  $H$  that minimizes  $weight(M, S)$  such that:

- ▶  $weight(M, (c_i, w_i)) = 0$  if  $M$  satisfies  $c_i$ , else  $w_i$ .
- ▶  $weight(M, S) = \sum_{wc \in S} weight(M, wc)$

Simply called MaxSAT if  $k = 1$  and  $H = \emptyset$

# How to solve MaxSat MinUnsat with SAT?

- ▶ Associate to each clause a weight (penalty)  $w_i$  taken into account if the clause is violated: **Soft clauses  $S$** .  
 $(\neg a \vee b, 6) \wedge (\neg b \vee c, 8)$
- ▶ Special weight ( $\infty$ ) for clauses that cannot be violated: **hard clauses  $H$**   
 $(a, \infty) \wedge (\neg c, \infty)$

## Definition (Partial Weighted MaxSat)

Find a model  $M$  of  $H$  that minimizes  $\text{weight}(M, S)$  such that:

- ▶  $\text{weight}(M, (c_i, w_i)) = 0$  if  $M$  satisfies  $c_i$ , else  $w_i$ .
- ▶  $\text{weight}(M, S) = \sum_{wc \in S} \text{weight}(M, wc)$

Simply called MaxSAT if  $k = 1$  and  $H = \emptyset$

# How to solve MaxSat MinUnsat with SAT?

- ▶ Associate to each clause a weight (penalty)  $w_i$  taken into account if the clause is violated: **Soft clauses  $S$** .  
 $(\neg a \vee b, 6) \wedge (\neg b \vee c, 8)$
- ▶ Special weight ( $\infty$ ) for clauses that cannot be violated: **hard clauses  $H$**   
 $(a, \infty) \wedge (\neg c, \infty)$

## Definition (Partial Weighted MaxSat)

Find a model  $M$  of  $H$  that minimizes  $weight(M, S)$  such that:

- ▶  $weight(M, (c_i, w_i)) = 0$  if  $M$  satisfies  $c_i$ , else  $w_i$ .
- ▶  $weight(M, S) = \sum_{wc \in S} weight(M, wc)$  weight of  $\{a, \neg b, \neg c\}$  is 6

Simply called MaxSAT if  $k = 1$  and  $H = \emptyset$

# Linear Search for solving MaxSAT

 $x_6, x_2$  $\neg x_6, x_2$  $\neg x_2, x_1$  $\neg x_1$  $\neg x_6, x_8$  $x_6, \neg x_8$  $x_2, x_4$  $\neg x_4, x_5$  $x_7, x_5$  $\neg x_7, x_5$  $\neg x_5, x_3$  $\neg x_3$ 

Example CNF formula ( $k = 1$  for each clause, not displayed)

# Linear Search for solving MaxSAT

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1$  $\neg x_1, b_2$  $\neg x_6, x_8, b_9$  $x_6, \neg x_8, b_{10}$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5$  $\neg x_3, b_6$ 

Add selector or **blocking** variables  $b_i$

# Linear Search for solving MaxSAT

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1$  $\neg x_1, b_2$  $\neg x_6, x_8, b_9$  $x_6, \neg x_8, b_{10}$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5$  $\neg x_3, b_6$ 

Formula is **SAT**; eg model M contains

$b_1, \neg b_2, b_3, \neg b_4, b_5, \neg b_7, \neg b_8, \neg b_9, b_{10}, \neg b_{11}, b_{12}$

# Linear Search for solving MaxSAT

$x_6, x_2, b_7$

$\neg x_6, x_2, b_8$

$\neg x_2, x_1, b_1$

$\neg x_1, b_2$

$\neg x_6, x_8, b_9$

$x_6, \neg x_8, b_{10}$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5, b_{11}$

$\neg x_7, x_5, b_{12}$

$\neg x_5, x_3, b_5$

$\neg x_3, b_6$

$$\sum_{i=1}^{12} b_i < 5$$

Bound the number of constraints to be relaxed:  $|M \cap B| = 5$

# Linear Search for solving MaxSAT

 $x_6, x_2, b_7$ 
 $\neg x_6, x_2, b_8$ 
 $\neg x_2, x_1, b_1$ 
 $\neg x_1, b_2$ 
 $\neg x_6, x_8, b_9$ 
 $x_6, \neg x_8, b_{10}$ 
 $x_2, x_4, b_3$ 
 $\neg x_4, x_5, b_4$ 
 $x_7, x_5, b_{11}$ 
 $\neg x_7, x_5, b_{12}$ 
 $\neg x_5, x_3, b_5$ 
 $\neg x_3, b_6$ 

$$\sum_{i=1}^{12} b_i < 5$$

Formula is (again) **SAT**; eg model contains  
 $b_1, \neg b_2, \neg b_3, \neg b_4, \neg b_5, \neg b_7, \neg b_8, \neg b_9, \neg b_{10}, \neg b_{11}, b_{12}$

# Linear Search for solving MaxSAT

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1$  $\neg x_1, b_2$  $\neg x_6, x_8, b_9$  $x_6, \neg x_8, b_{10}$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5$  $\neg x_3, b_6$ 

$$\sum_{i=1}^{12} b_i < 2$$

Bound the number of constraints to be relaxed  $|M \cap B| = 2$

# Linear Search for solving MaxSAT

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1$  $\neg x_1, b_2$  $\neg x_6, x_8, b_9$  $x_6, \neg x_8, b_{10}$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5$  $\neg x_3, b_6$ 

$$\sum_{i=1}^{12} b_i < 2$$

Instance is now **UNSAT**

# Linear Search for solving MaxSAT

$x_6, x_2, b_7$

$\neg x_6, x_2, b_8$

$\neg x_2, x_1, b_1$

$\neg x_1, b_2$

$\neg x_6, x_8, b_9$

$x_6, \neg x_8, b_{10}$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5, b_{11}$

$\neg x_7, x_5, b_{12}$

$\neg x_5, x_3, b_5$

$\neg x_3, b_6$

$$\sum_{i=1}^{12} b_i < 2$$

MaxSAT solution is  $|\varphi| - |M \cap B| = 12 - 2 = 10$

Note that ...

- ▶ No initial upper or lower bounds: **the first model provides a first upper bound.**
- ▶ In practice, the objective function can be used to guide the search
- ▶ The procedure follows a SAT, SAT, SAT, SAT, ..., UNSAT pattern with linear search
- ▶ Binary search is possible but:
  - ▶ SAT answer is usually faster than UNSAT
  - ▶ the solver must be reset in case on unsatisfiability
- ▶ **In lucky case, two calls to the SAT solver are sufficient (one SAT + one UNSAT).**
- ▶ Used in Sat4j since 2006, was state-of-the-art in 2009
- ▶ **Main issue: how to represent the bound constraint?**

# From Unsat Core computation to MaxSat: MSU

Z. Fu and S. Malik, On solving the partial MAX-SAT problem, in International Conference on Theory and Applications of Satisfiability Testing, August 2006, p 252-265.

10

Other SAT-based approaches in practical Max Sat solving rely on unsat core computation [Fu and Malik 2006]:

- ▶ Compute one unsat core  $C'$  of the formula  $C$
- ▶ Relax it by replacing  $C'$  by  $\{ r_i \vee C_i | C_i \in C'\}$
- ▶ Add the constraint  $\sum r_i \leq 1$  to  $C$
- ▶ Repeat until the formula is satisfiable
- ▶ If  $\text{MinUnsat}(C) = k$ , requires  $k + 1$  loops.

Many improvement since then (PM1, PM2, MsUncore, etc): works for Weighted Max Sat, reduction of the number of relaxation variables, etc.

# Fu&Malik's Algorithm: msu1.0

$x_6, x_2$

$\neg x_6, x_2$

$\neg x_2, x_1$

$\neg x_1$

$\neg x_6, x_8$

$x_6, \neg x_8$

$x_2, x_4$

$\neg x_4, x_5$

$x_7, x_5$

$\neg x_7, x_5$

$\neg x_5, x_3$

$\neg x_3$

Example CNF formula

# Fu&Malik's Algorithm: msu1.0

 $x_6, x_2$  $\neg x_6, x_2$  $\neg x_2, x_1$  $\neg x_1$  $\neg x_6, x_8$  $x_6, \neg x_8$  $x_2, x_4$  $\neg x_4, x_5$  $x_7, x_5$  $\neg x_7, x_5$  $\neg x_5, x_3$  $\neg x_3$ 

Formula is **UNSAT**; Get unsat core

# Fu&Malik's Algorithm: msu1.0

$x_6, x_2$

$\neg x_6, x_2$

$\neg x_2, x_1, b_1$

$\neg x_1, b_2$

$\neg x_6, x_8$

$x_6, \neg x_8$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5$

$\neg x_7, x_5$

$\neg x_5, x_3, b_5$

$\neg x_3, b_6$

$$\sum_{i=1}^6 b_i \leq 1$$

Add **blocking variables** and AtMost1 constraint

# Fu&Malik's Algorithm: msu1.0

$x_6, x_2$

$\neg x_6, x_2$

$\neg x_2, x_1, b_1$

$\neg x_1, b_2$

$\neg x_6, x_8$

$x_6, \neg x_8$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5$

$\neg x_7, x_5$

$\neg x_5, x_3, b_5$

$\neg x_3, b_6$

$$\sum_{i=1}^6 b_i \leq 1$$

Formula is (again) **UNSAT**; Get unsat core

# Fu&Malik's Algorithm: msu1.0

$x_6, x_2, b_7$

$\neg x_6, x_2, b_8$

$\neg x_2, x_1, b_1, b_9$

$\neg x_1, b_2, b_{10}$

$\neg x_6, x_8$

$x_6, \neg x_8$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5, b_{11}$

$\neg x_7, x_5, b_{12}$

$\neg x_5, x_3, b_5, b_{13}$

$\neg x_3, b_6, b_{14}$

$$\sum_{i=1}^6 b_i \leq 1$$

$$\sum_{i=7}^{14} b_i \leq 1$$

Add new **blocking variables** and AtMost1 constraint

# Fu&Malik's Algorithm: msu1.0

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1, b_9$  $\neg x_1, b_2, b_{10}$  $\neg x_6, x_8$  $x_6, \neg x_8$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5, b_{13}$  $\neg x_3, b_6, b_{14}$ 

$$\sum_{i=1}^6 b_i \leq 1$$

$$\sum_{i=7}^{14} b_i \leq 1$$

Instance is now **SAT**

# Fu&Malik's Algorithm: msu1.0

$x_6, x_2, b_7$

$\neg x_6, x_2, b_8$

$\neg x_2, x_1, b_1, b_9$

$\neg x_1, b_2, b_{10}$

$\neg x_6, x_8$

$x_6, \neg x_8$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5, b_{11}$

$\neg x_7, x_5, b_{12}$

$\neg x_5, x_3, b_5, b_{13}$

$\neg x_3, b_6, b_{14}$

$$\sum_{i=1}^6 b_i \leq 1$$

$$\sum_{i=7}^{14} b_i \leq 1$$

MaxSAT solution is  $|\varphi| - I = 12 - 2 = 10$

Note that ...

- ▶ Unsat core may not be minimal
- ▶ Nice property: if  $k$  constraints must be relaxed, then the procedure requires exactly  $k + 1$  calls to the SAT solver.
- ▶ How to represent the cardinality constraints?

# MaxHS: SAT and MIP solver interplay

Jessica Davies, Fahiem Bacchus: Solving MAXSAT by Solving a Sequence of Simpler S Instances. CP 2011: 225-239

13

- ▶ Core guided MAXSAT solver can be seen as a two step procedure:
  - ▶ Discover UNSAT cores of the formula
  - ▶ Stop as soon as one minimal Hitting Set of the cores satisfies the formula
- ▶ The size of the HS provides the number of constraints to relax
- ▶ May require to enumerate all MUS of a formula
- ▶ Or less if lucky

# MaxHS principle

 $x_6, x_2, b_7$ 
 $\neg x_6, x_2, b_8$ 
 $\neg x_2, x_1, b_1$ 
 $\neg x_1, b_2$ 
 $\neg x_6, x_8, b_9$ 
 $x_6, \neg x_8, b_{10}$ 
 $x_2, x_4, b_3$ 
 $\neg x_4, x_5, b_4$ 
 $x_7, x_5, b_{11}$ 
 $\neg x_7, x_5, b_{12}$ 
 $\neg x_5, x_3, b_5$ 
 $\neg x_3, b_6$ 

*Cores* = {}

*HS* =  $\emptyset$

# MaxHS principle

 $x_6, x_2, b_7$ 
 $\neg x_6, x_2, b_8$ 
 $\neg x_2, x_1, b_1$ 
 $\neg x_1, b_2$ 
 $\neg x_6, x_8, b_9$ 
 $x_6, \neg x_8, b_{10}$ 
 $x_2, x_4, b_3$ 
 $\neg x_4, x_5, b_4$ 
 $x_7, x_5, b_{11}$ 
 $\neg x_7, x_5, b_{12}$ 
 $\neg x_5, x_3, b_5$ 
 $\neg x_3, b_6$ 
 $\{\{b_1, b_2, b_3, b_4, b_5, b_6\}\}$ 
 $HS = \{b_4\}$

# MaxHS principle

 $x_6, x_2, b_7$ 
 $\neg x_6, x_2, b_8$ 
 $\neg x_2, x_1, b_1$ 
 $\neg x_1, b_2$ 
 $\neg x_6, x_8, b_9$ 
 $x_6, \neg x_8, b_{10}$ 
 $x_2, x_4, b_3$ 
 $\neg x_4, x_5, b_4$ 
 $x_7, x_5, b_{11}$ 
 $\neg x_7, x_5, b_{12}$ 
 $\neg x_5, x_3, b_5$ 
 $\neg x_3, b_6$ 
 $\{\{b_1, b_2, b_3, b_4, b_5, b_6\}, \{b_1, b_2, b_7, b_8\}\}$ 
 $HS = \{b_1\}$

# MaxHS principle

$x_6, x_2, b_7$

$\neg x_6, x_2, b_8$

$\neg x_2, x_1, b_1$

$\neg x_1, b_2$

$\neg x_6, x_8, b_9$

$x_6, \neg x_8, b_{10}$

$x_2, x_4, b_3$

$\neg x_4, x_5, b_4$

$x_7, x_5, b_{11}$

$\neg x_7, x_5, b_{12}$

$\neg x_5, x_3, b_5$

$\neg x_3, b_6$

$\{\{b_1, b_2, b_3, b_4, b_5, b_6\}, \{b_1, b_2, b_7, b_8\}, \{b_{11}, b_{12}, b_5, b_6\}\}$

$HS = \{b_2, b_5\}$

# MaxHS principle

 $x_6, x_2, b_7$  $\neg x_6, x_2, b_8$  $\neg x_2, x_1, b_1$  $\neg x_1, b_2$  $\neg x_6, x_8, b_9$  $x_6, \neg x_8, b_{10}$  $x_2, x_4, b_3$  $\neg x_4, x_5, b_4$  $x_7, x_5, b_{11}$  $\neg x_7, x_5, b_{12}$  $\neg x_5, x_3, b_5$  $\neg x_3, b_6$ 

Instance is SAT. MaxSAT solution is  $12 - |\{b_2, b_5\}| = 10$

- ▶ Take advantage of SAT solvers feedback: model or core
- ▶ No single approach outperforms the others
- ▶ Core-guided and MaxHS work best currently **on "application" benchmarks (not crafted ones)**

Linear Search or Core-Guided approaches require encoding cardinality constraints in CNF (or use native support for such constraints as found in Sat4j)

## Hamiltonian Cycle Problem SAT-encoding

Let  $G = (V, A)$  a digraph where  $V$  is a set of  $n$  vertices and  $A$  is a set of arcs. Let  $x_{ij}$  be Boolean variables such that  $x_{ij} = 1 \leftrightarrow (i, j) \in A$  belongs to a cycle.

$$\sum_{(i,j) \in A} x_{ij} = 1 \quad \text{for each } i = 1, \dots, n \text{ (out-degree)}$$

$$\sum_{(i,j) \in A} x_{ij} = 1 \quad \text{for each } j = 1, \dots, n \text{ (in-degree)}$$

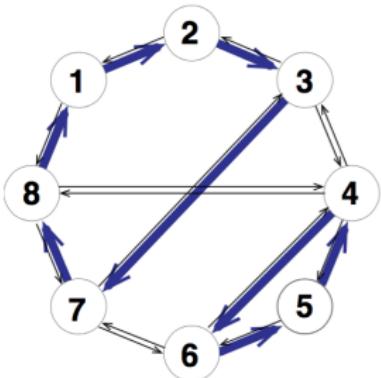
$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad S \subset V, 2 \leq |S| \leq n - 2 \text{ (connectivity)}$$

- ▶ in/out-degree constraints ensure that in/out-degrees are respectively exactly one for each node in solution cycles
- ▶ connectivity constraints prohibits sub-cycles

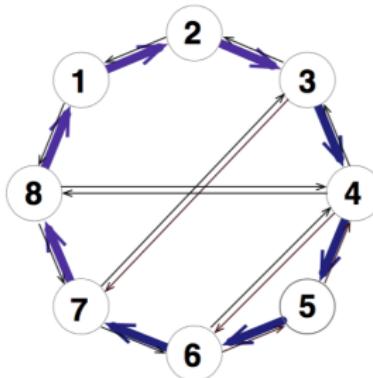
Encoding requires  $O(n^3)$  clauses [Pre03]

# How to solve HCP efficiently with SAT?

- ▶ With only in/out-degree constraints, we have cycles but they may not be connected (Case A)
- ▶ With all constraints, we can find a Hamiltonian cycle (Case B)



(Case A)  
in/out-degree



(Case B)  
in/out-degree + connectivity

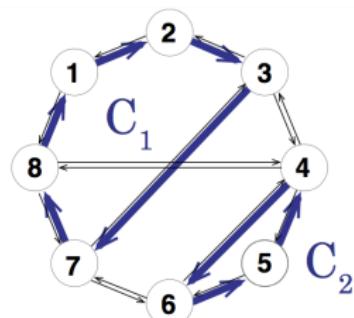
But the SAT solver may be lucky!

# Incremental SAT-based method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem

T. Soh, D. Le Berre, S. Roussel, M. Banbara, and N. Tamura, JELIA'14, pages 684–693

18

- ▶ Do not encode cardinality constraints in CNF (Sat4j)
- ▶ Ask the SAT solver for a cycle
- ▶ We can get lucky and find an Hamiltonian Cycle quickly
- ▶ Else add new clauses to block the sub-cycles (connectivity constraints generated lazily).



## Blocking Clauses

$C_1 \neg x_{12} \vee \neg x_{23} \vee \neg x_{37} \vee \neg x_{78} \vee \neg x_{81}$

$C'_1 \neg x_{87} \vee \neg x_{73} \vee \neg x_{32} \vee \neg x_{21} \vee \neg x_{18}$

$C_2 \neg x_{46} \vee \neg x_{65} \vee \neg x_{54}$

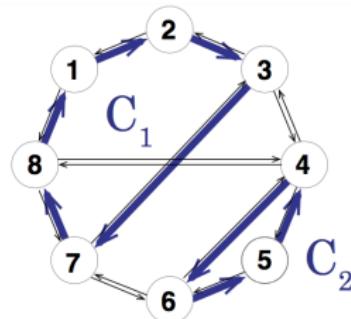
$C'_2 \neg x_{45} \vee \neg x_{56} \vee \neg x_{64}$

# Incremental SAT-based method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem

T. Soh, D. Le Berre, S. Roussel, M. Banbara, and N. Tamura, JELIA'14, pages 684–693

18

- ▶ Do not encode cardinality constraints in CNF (Sat4j)
- ▶ Ask the SAT solver for a cycle
- ▶ We can get lucky and find an Hamiltonian Cycle quickly
- ▶ Else add new clauses to block the sub-cycles (connectivity constraints generated lazily).



## Blocking Clauses

$C_1 \neg x_{12} \vee \neg x_{23} \vee \neg x_{37} \vee \neg x_{78} \vee \neg x_{81}$   
 $C'_1 \neg x_{87} \vee \neg x_{73} \vee \neg x_{32} \vee \neg x_{21} \vee \neg x_{18}$   
 $C_2 \neg x_{46} \vee \neg x_{65} \vee \neg x_{54}$   
 $C'_2 \neg x_{45} \vee \neg x_{56} \vee \neg x_{64}$

This idea of going step by step and refining each step is called:

**CEGAR**: CounterExample Guided Abstraction Refinement

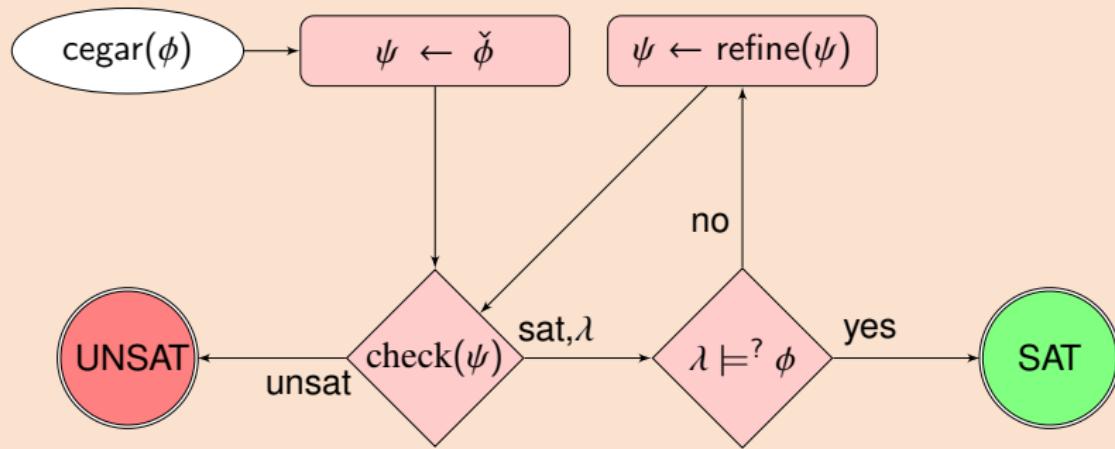
## CEGAR: CounterExample Guided Abstraction Refinement

To solve a problem, we may need to consider only a small part of it  
[CGJ<sup>+</sup>03]

- ▶ To abstract problems: hoping it will be easier to solve
- ▶ Two variants of abstraction:
  - ▶ Under-abstraction: abstraction has **more** solutions
  - ▶ Over-abstraction: abstraction has **less** solutions
- ▶ CEGAR-over: CEGAR approach using over-abstractions
- ▶ CEGAR-under: CEGAR approach using under-abstractions

# CEGAR using under-abstractions

## CEGAR-under

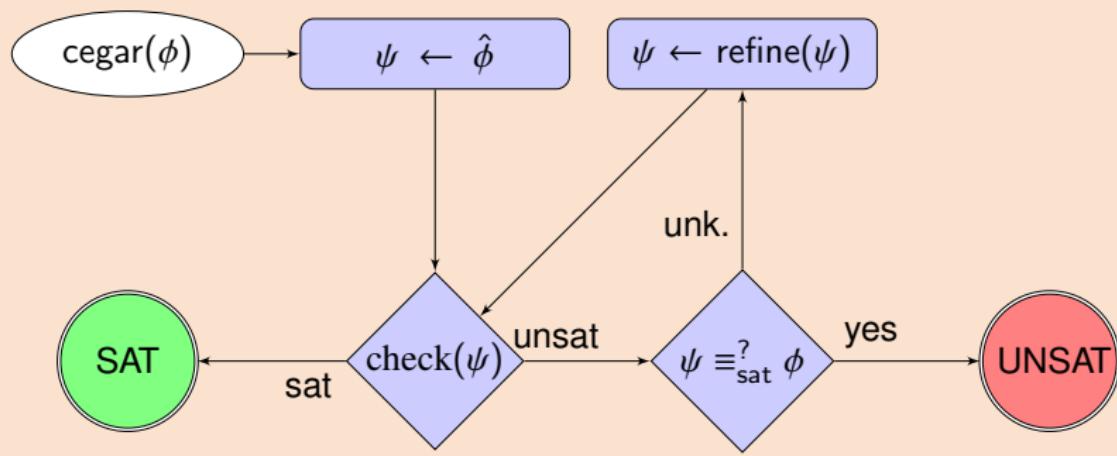


## Example

Hamiltonian cycle problem

# CEGAR using over-abstractions

## CEGAR-over



## Example

Planning problem, by increasing step by step the horizon; Bounded Model Checking

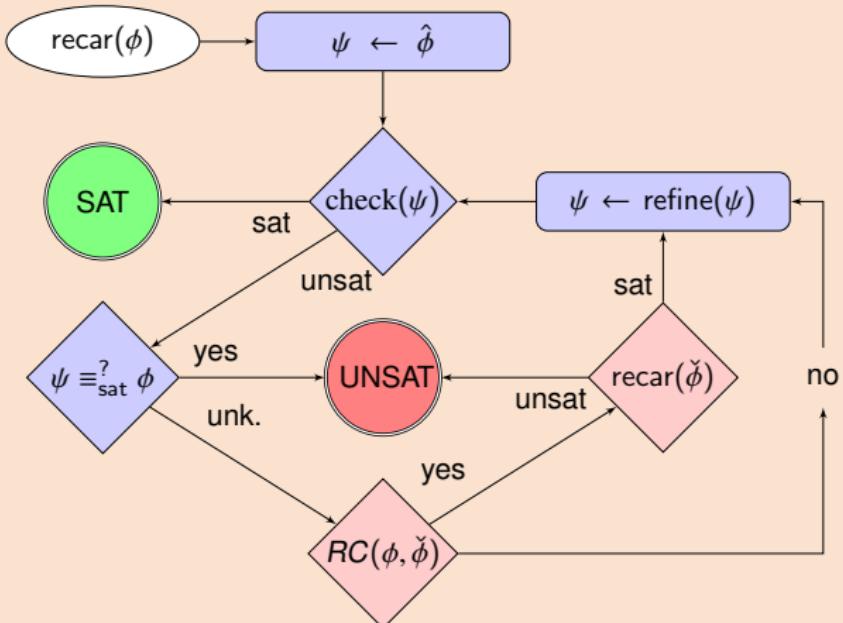
## Advantages

- ▶ If problem mainly satisfiable: CEGAR-over
- ▶ If problem mainly unsatisfiable: CEGAR-under
- ▶ When check improves, CEGAR improves
- ▶ Many applications already use CEGAR

## Drawbacks

- ▶ Not efficient when 50/50 chances of being SAT/UNSAT
- ▶ Not efficient when we need many refinement steps

## RECAR



**Recursive Explore and Check Abstraction Refinement**

- ▶ Called *RECAR* [LLdLM17]
- ▶ Inspired by CEGAR [CGJ<sup>+</sup>03]
- ▶ Rely on 5 very important assumptions

**RECAR Assumptions**

1. Function ‘check’ is sound, complete and terminates
2.  $\text{isSAT}(\hat{\phi})$  implies  $\text{isSAT}(\text{refine}(\hat{\phi}))$
3.  $\exists n \in \mathbb{N}$  s.t.  $\text{refine}^n(\hat{\phi}) \equiv_{\text{sat}}^? \phi$ .
4.  $\text{isUNSAT}(\check{\phi})$  implies  $\text{isUNSAT}(\phi)$
5.  $\exists n \in \mathbb{N}$  s.t.  $RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$  is false.



$\exists n \in \mathbb{N}$  s.t.  $RC(under^n(\phi), under^{n+1}(\phi))$  is false.

## RC function

- ▶ ‘true’ if we can do a recursive call, ‘false’ otherwise
- ▶ It compares  $under^i(\phi)$  and  $under^{i+1}(\phi)$
- ▶ It checks if  $under^{i+1}(\phi)$  will be “easier to solve” than  $under^i(\phi)$



## RECAR

- ▶ 2 levels of abstractions
  - ▶ One at the Oracle level ( $\text{check}(\psi)$ )
  - ▶ One at the Domain level (recursive call)
- ▶ Efficient even when 50/50 chance of being SAT/UNSAT
- ▶ When check improves, RECAR improves
- ▶ The return of the recursive call can reduce the number of refinements
- ▶ SAT and UNSAT shortcuts can be inverted if needed
- ▶ Totally generic, can change SAT solver by QBF/SMT/FO solver

## RECAR for Modal Logic K

- ▶ Modal Logic K is **PSPACE**-complete [Lad77, Hal95]
- ▶ What is Modal Logic K?
- ▶ How we over-approximate a formula  $\phi$ ?
- ▶ How we under-approximate a formula  $\phi$ ?
- ▶ Is it competitive against a CEGAR approach?
- ▶ Is it competitive against the state-of-the-art approaches?

# Preliminaries: Modal Logic

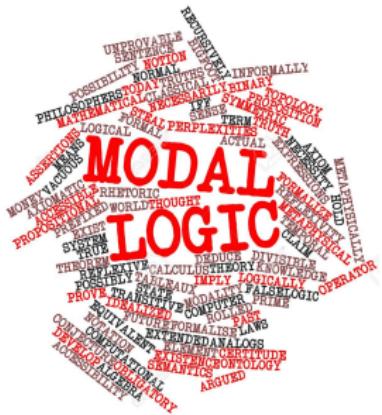
Modal Logic = Propositional Logic +  $\Box$  and  $\Diamond$

## Modal Logic

- ▶  $\Box\phi$  means  $\phi$  is necessarily true
- ▶  $\Diamond\phi$  means  $\phi$  is possibly true

$$\Diamond\phi \leftrightarrow \neg\Box\neg\phi$$

$$\Box\phi \leftrightarrow \neg\Diamond\neg\phi$$



# Preliminaries: Kripke Structure

- ▶  $\mathbb{P}$  finite non-empty set of propositional variables

## Kripke Structure [Kri59]

$M = \langle W, R, V \rangle$  with:

- ▶  $W$ , a non-empty set of possible worlds
- ▶  $R$ , a binary relation on  $W$
- ▶  $V$ , a function that associate to each  $p \in \mathbb{P}$ , the set of possible worlds where  $p$  is true

Pointed Kripke Structure:  $\langle \mathcal{K}, w \rangle$

- ▶  $\mathcal{K}$ : Kripke Structure
- ▶  $w$ : a possible world in  $W$

# Preliminaries: Satisfaction Relation

## Definition (Satisfaction Relation)

The relation  $\models$  between Kripke Structures and formulae is recursively defined as follows:

$\langle \mathcal{K}, w \rangle \models p$	iff	$w \in V(p)$
$\langle \mathcal{K}, w \rangle \models \neg\phi$	iff	$\langle \mathcal{K}, w \rangle \not\models \phi$
$\langle \mathcal{K}, w \rangle \models \phi_1 \wedge \phi_2$	iff	$\langle \mathcal{K}, w \rangle \models \phi_1$ and $\langle \mathcal{K}, w \rangle \models \phi_2$
$\langle \mathcal{K}, w \rangle \models \phi_1 \vee \phi_2$	iff	$\langle \mathcal{K}, w \rangle \models \phi_1$ or $\langle \mathcal{K}, w \rangle \models \phi_2$
$\langle \mathcal{K}, w \rangle \models \Box\phi$	iff	$(w, w') \in R$ implies $\langle \mathcal{K}, w' \rangle \models \phi$
$\langle \mathcal{K}, w \rangle \models \Diamond\phi$	iff	$(w, w') \in R$ and $\langle \mathcal{K}, w' \rangle \models \phi$

$\mathcal{K}$  that satisfied a formula  $\phi$  will be called “Kripke model of  $\phi$ ”

# Preliminaries: Example of a Kripke Structure

✓  $\phi_1 = \square(\bullet)$

✗  $\phi_2 = \square\Diamond(\bullet)$

✓  $\phi_3 = \Diamond(\bullet \wedge \Diamond\neg\bullet)$

✓  $\phi_4 = (\bullet \vee \bullet \vee \bullet)$

✗  $\phi_5 = \Diamond\Diamond(\bullet \wedge \square\neg\bullet)$

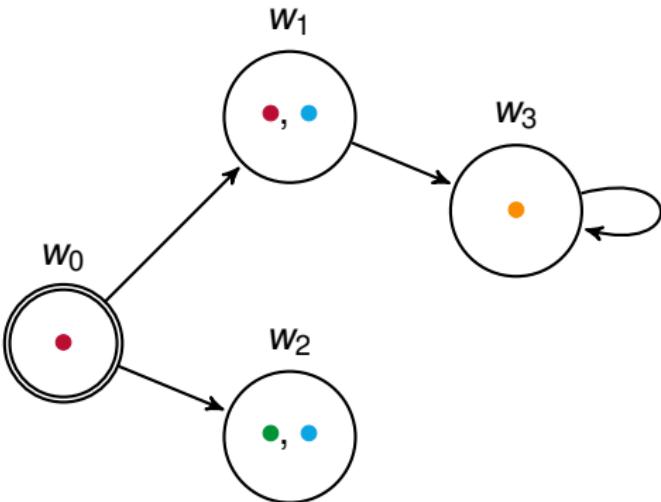


Figure: Example  $\mathcal{K}$

# Preliminaries: Example of a Kripke Structure

✓  $\phi_1 = \square(\bullet)$

✗  $\phi_2 = \square\Diamond(\bullet)$

✓  $\phi_3 = \Diamond(\bullet \wedge \Diamond\neg\bullet)$

✓  $\phi_4 = (\bullet \vee \bullet \vee \bullet)$

✓  $\phi_5 = \Diamond\Diamond(\bullet \wedge \square\neg\bullet)$

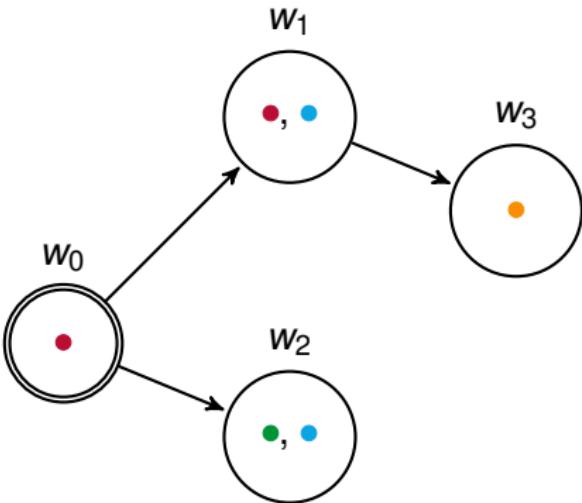


Figure: Example  $\mathcal{K}$

## MoSaiC

- ▶ Modal Logic K solver
- ▶ Uses Glucose as internal SAT solver
- ▶ Uses a RECAR approach

## MoSaiC

- ▶ Modal Logic K solver
- ▶ Uses Glucose as internal SAT solver
- ▶ Uses a RECAR approach

## RECAR Assumptions: Reminder

- ✓ 1 Function ‘check’ is sound, complete and terminates
- ? 2  $\text{isSAT}(\hat{\phi})$  implies  $\text{isSAT}(\text{refine}(\hat{\phi}))$
- ? 3  $\exists n \in \mathbb{N}$  s.t.  $\text{refine}^n(\hat{\phi}) \equiv_{\text{sat}}^? \phi$
- 4  $\text{isUNSAT}(\check{\phi})$  implies  $\text{isUNSAT}(\phi)$
- 5  $\exists n \in \mathbb{N}$  s.t.  $RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$  is false

# MoSaiC: Over-Approximation (CNF level)

$\phi$  always in NNF and  $\text{over}(\phi, i)$  in CNF using Tseitin's translation

$$\text{over}(\phi, n) = \text{over}'(\phi, 0, n)$$

$$\text{over}'(p_k, i, n) = p_{k,i} \quad \text{over}'(\neg p_k, i, n) = \neg p_{k,i}$$

$$\text{over}'(\Box\phi, i, n) = \bigwedge_{j=0}^n (r_{i,j} \rightarrow \text{over}'(\phi, j, n))$$

$$\text{over}'(\Diamond\phi, i, n) = \bigvee_{j=0}^n (r_{i,j} \wedge \text{over}'(\phi, j, n))$$

- ▶  $p_{k,i}$  means  $p_k$  is true in the world  $w_i$
- ▶  $r_{i,j}$  means that there is a relation between worlds  $w_i$  and  $w_j$
- ▶  $n$  is a bound on the number of worlds to consider

# Upper Bound for Modal Logic K

- ▶ Are there known bounds on the number of worlds to consider?

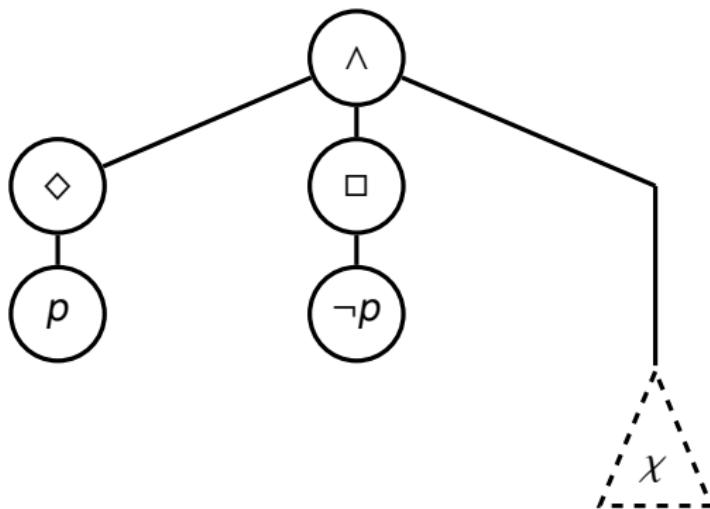
# Upper Bound for Modal Logic K

- ▶ Are there known bounds on the number of worlds to consider?
- ▶ yes, but quite large:  $UB(\phi) = \text{Atom}(\phi)^{\text{depth}(\phi)}$  [SM97]  
where  $\text{Atom}(\phi)$  denotes the number of propositional variables  
and  $\text{depth}(\phi)$  the modal depth of  $\phi$ .

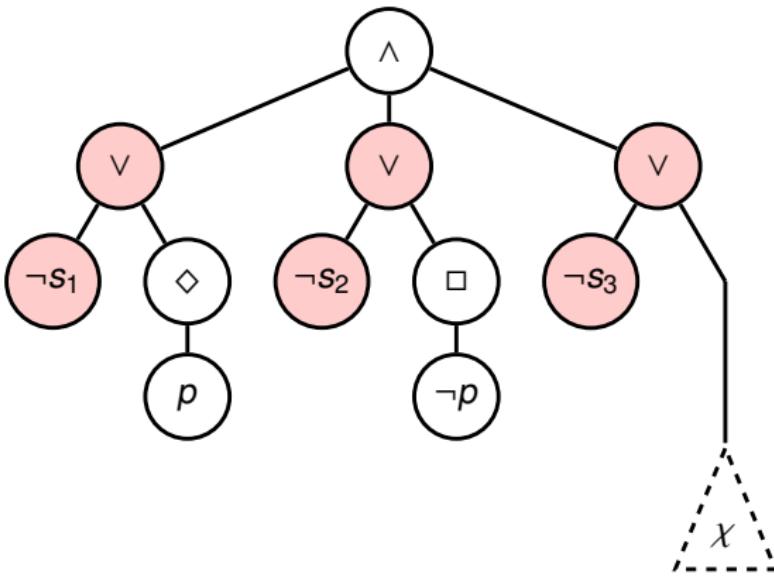
## RECAR Assumptions: Reminder

- ✓ 1 Function ‘check’ is sound, complete and terminates
- ✓ 2  $\text{isSAT}(\hat{\phi})$  implies  $\text{isSAT}(\text{refine}(\hat{\phi}))$
- ✓ 3  $\exists n \in \mathbb{N}$  s.t.  $\text{refine}^n(\hat{\phi}) \equiv_{\text{sat}}^? \phi$
- ? 4  $\text{isUNSAT}(\check{\phi})$  implies  $\text{isUNSAT}(\phi)$
- ? 5  $\exists n \in \mathbb{N}$  s.t.  $RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$  is false

Let's take an example, with  $\chi$  huge but satisfiable...

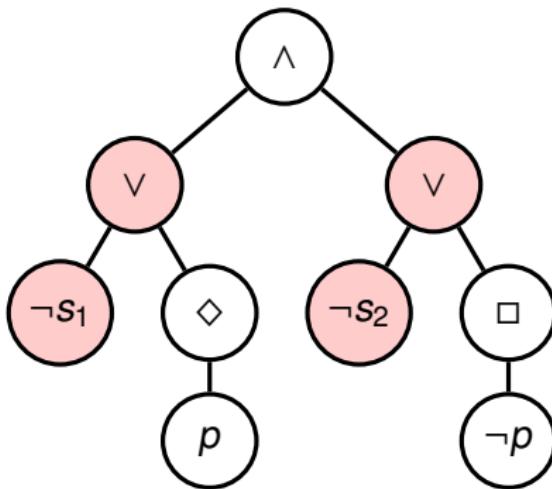


Worst case for CEGAR using our 'over' function



Modern SAT solvers return ‘the reason’ why a formula with  $n$  worlds is unsatisfiable ( $\text{core} = \{s_1, s_2\}$ )

We want to cut what is not part of the ‘unsatisfiability’ ( $s_i \notin \text{core}$ )



We just create  $\check{\phi}$  smaller than  $\phi$  and easier to solve.

The function *RC* from RECAR just says here: did we cut something ?

# MoSaiC: Under-Approximation (modal logic level)

$$\text{under}(p, \text{core}) = p$$

$$\text{under}(\neg p, \text{core}) = \neg p$$

$$\text{under}(\Box\phi, \text{core}) = \Box(\text{under}(\phi, \text{core}))$$

$$\text{under}(\Diamond\phi, \text{core}) = \Diamond(\text{under}(\phi, \text{core}))$$

$$\text{under}((\phi \wedge \psi), \text{core}) = \text{under}(\phi, \text{core}) \wedge \text{under}(\psi, \text{core})$$

$$\text{under}((\psi \vee \chi), \text{core}) = \begin{cases} \text{under}(\chi, \text{core}) & \text{if } \psi = \neg s_i, s_i \in \text{core} \\ \top & \text{if } \psi = \neg s_i, s_i \notin \text{core} \\ (\text{under}(\psi, \text{core}) \\ \vee \text{under}(\chi, \text{core})) & \text{otherwise} \end{cases}$$

the unsatisfiable-core obtained from the solver drives the under-approximation

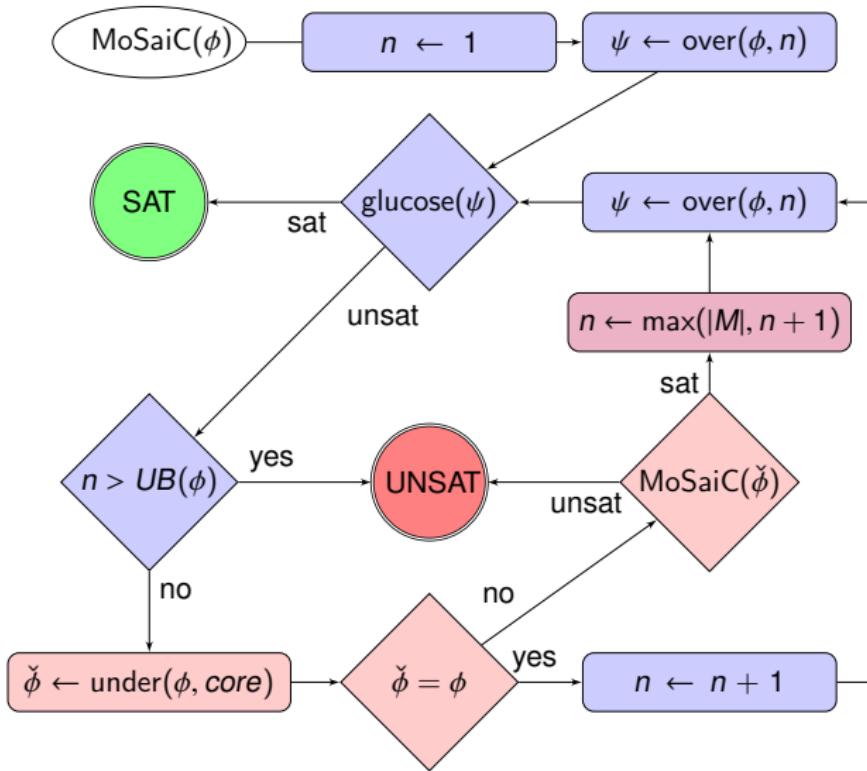
# Behavior of RC predicate

- ▶ RC returns true iff the unsat core is strictly smaller than the input formula
- ▶ eventually the inconsistency will become global
- ▶ thus the predicate will return false

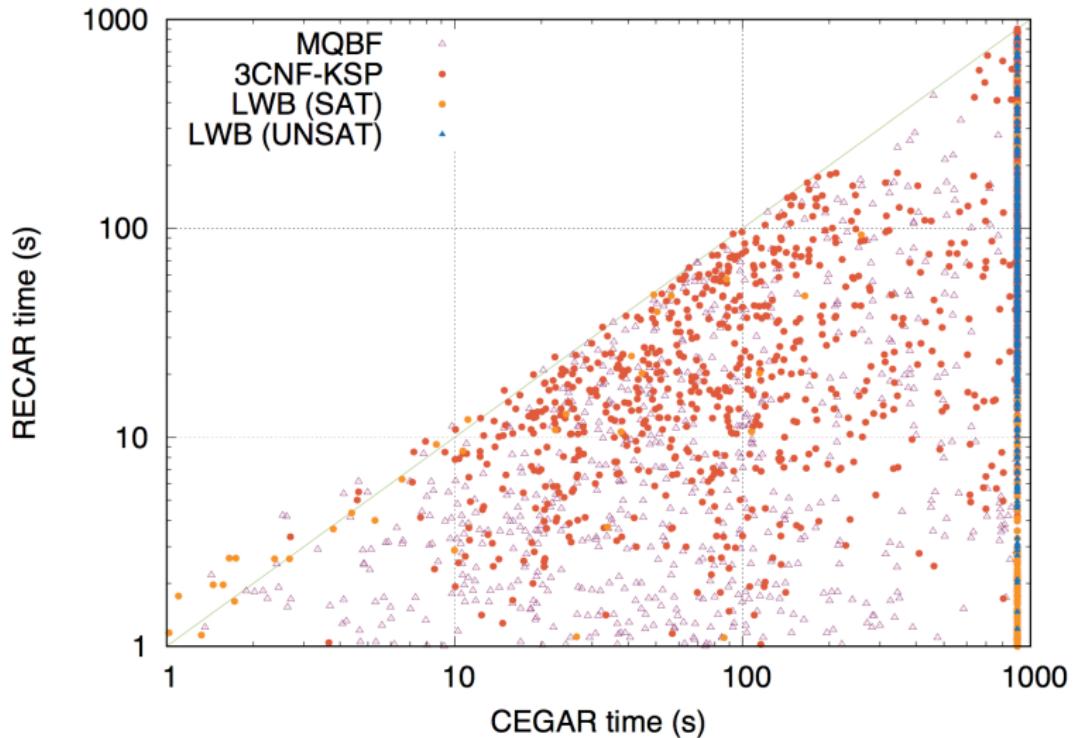
## RECAR Assumptions: Reminder

- ✓ 1 Function ‘check’ is sound, complete and terminates
- ✓ 2  $\text{isSAT}(\hat{\phi})$  implies  $\text{isSAT}(\text{refine}(\hat{\phi}))$
- ✓ 3  $\exists n \in \mathbb{N}$  s.t.  $\text{refine}^n(\hat{\phi}) \equiv_{\text{sat}}^? \phi$
- ✓ 4  $\text{isUNSAT}(\check{\phi})$  implies  $\text{isUNSAT}(\phi)$
- ✓ 5  $\exists n \in \mathbb{N}$  s.t.  $RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$  is false

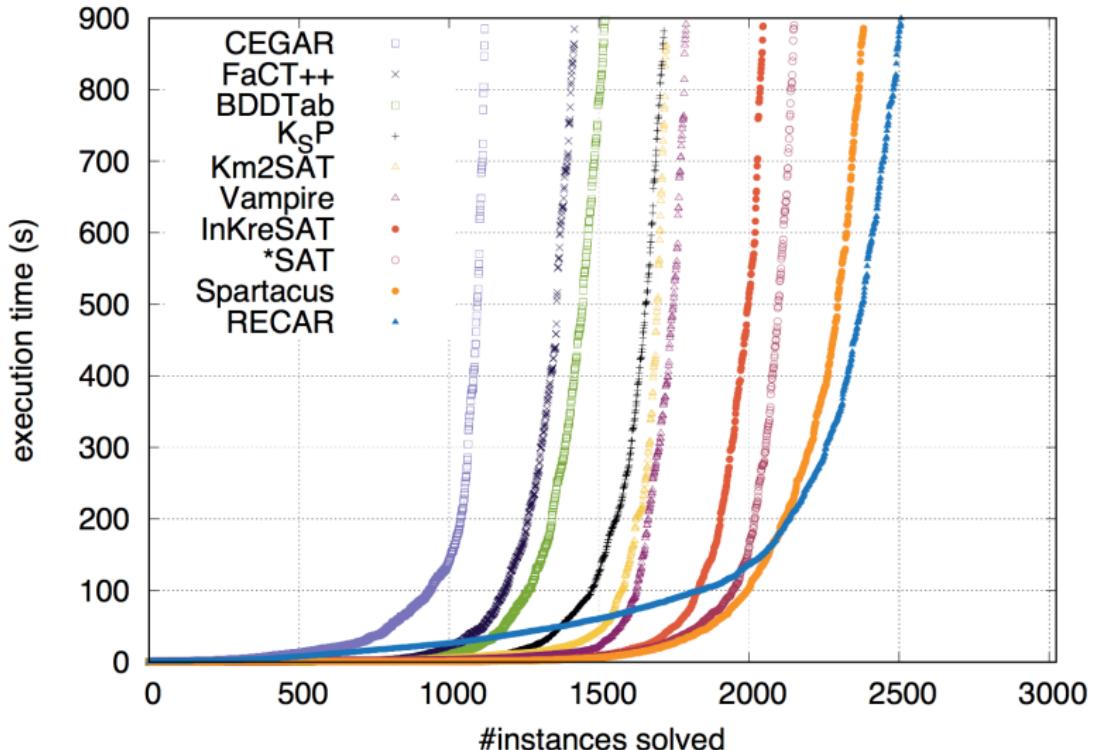
# MoSaiC: RECAR for Modal Logic K



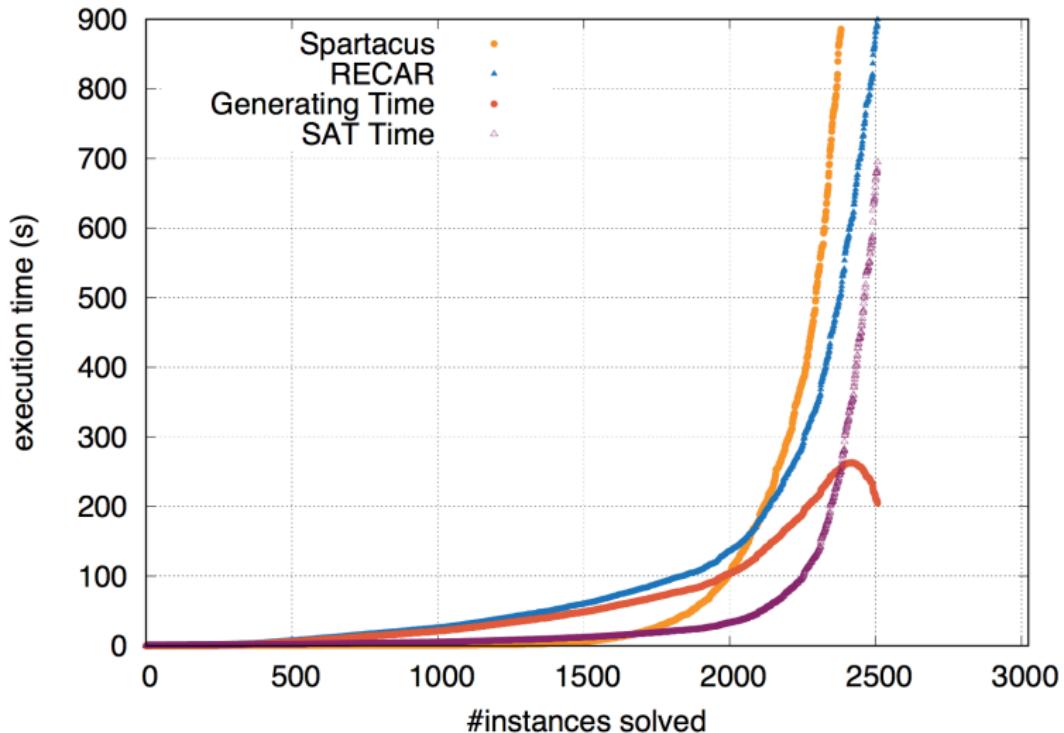
# MoSaiC: RECAR for Modal Logic K



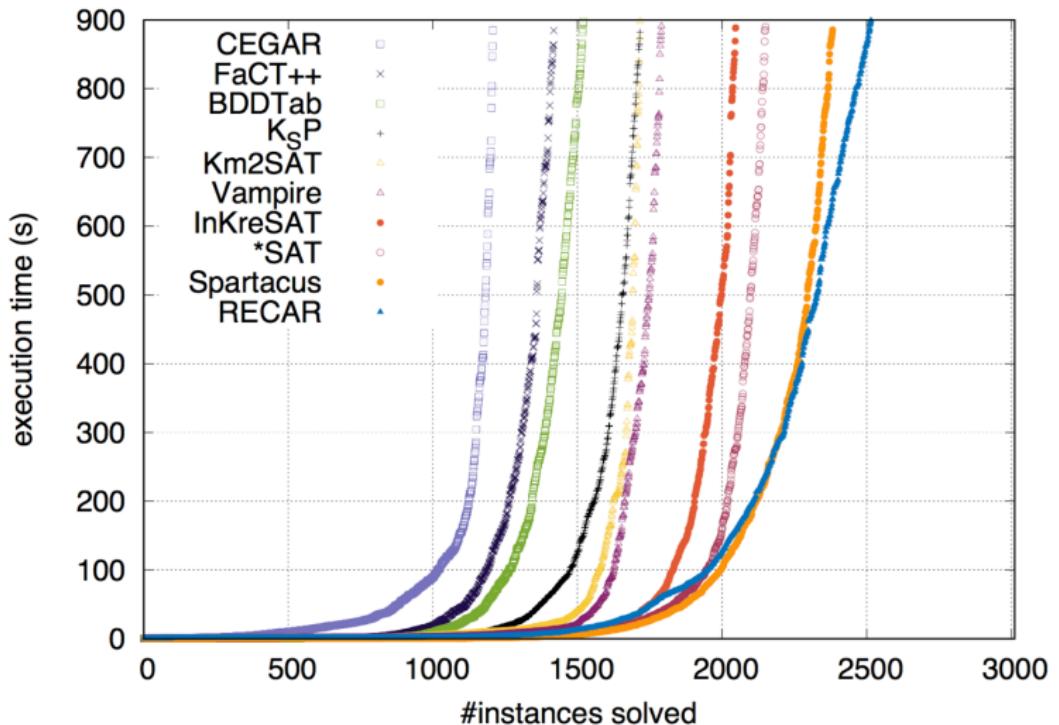
# MoSaiC: RECAR for Modal Logic K



# Explanation of the Cactus-Plot



# Some tweaks improve the results



# Take home message for RECAR

## RECAR

- ▶ New generic approach to solve problems using decision procedures
- ▶ Based on two levels of abstraction:
  - ▶ Decision procedure level as in CEGAR
  - ▶ Domain level for the recursive call
- ▶ Guided by the decision procedure
- ▶ Application to modal logic K satisfiability problem in MoSaiC

## Current limitations:

- ▶ Both domain and decision procedure expertise needed to design the abstractions
- ▶ Upper bound for modal logic K is quite large
- ▶ MoSaiC required tweaks to be efficient in practice

# General conclusion

- ▶ SAT solvers are not just SAT oracles (yes/no answers)
  - ▶ they provide models in case of satisfiability
  - ▶ they provide unsat core in case of unsatisfiability
  - ▶ they work “under assumption”
- ▶ SAT-based algorithm design must use the solver feedback
- ▶ **The solver should "drive" the algorithm**

# From satisfaction to optimization, and beyond

## SAT-based guided problem solving

Daniel Le Berre

*joint work with Mutsunori Banbara, Tiago de Lima, Jean-Marie Lagniez,  
Valentin Montmirail, Stéphanie Roussel, Naoyuki Tamura, Takehide Soh*

CNRS, Université d'Artois, FRANCE  
[{leberre}@crl.univ-artois.fr](mailto:{leberre}@crl.univ-artois.fr)

SAT+SMT school, IIT Bombay, India, 10 December 2019



# Bibliography I

 Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.

Counterexample-guided abstraction refinement for symbolic model checking.

*Journal of the ACM*, 50(5):752–794, 2003.

 Joseph Y. Halpern.

The Effect of Bounding the Number of Primitive Propositions and the Depth of Nesting on the Complexity of Modal Logic.

*Artificial Intelligence*, 75(2):361–372, 1995.

 Saul Kripke.

A completeness theorem in modal logic.

*J. Symb. Log.*, 24(1):1–14, 1959.

## Bibliography II

-  Richard E. Ladner.  
The Computational Complexity of Provability in Systems of  
Modal Propositional Logic.  
*SIAM J. Comput.*, 6(3):467–480, 1977.
-  Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima, and  
Valentin Montmirail.  
A Recursive Shortcut for CEGAR: Application To The Modal  
Logic K Satisfiability Problem.  
*In Proc. of IJCAI'17*, 2017.
-  Steven David Prestwich.  
SAT problems with chains of dependent variables.  
*Discrete Applied Mathematics*, 130(2):329–350, 2003.

## Bibliography III

-  Roberto Sebastiani and David McAllester.  
New Upper Bounds for Satisfiability in Modal Logics the  
Case-study of Modal K.  
Technical Report 9710-15, IRST, Trento, Italy, October 1997.