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}@cril.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
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**

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

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

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

$$C_1 \models \neg(a \land \neg c) = \neg a \lor c$$
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.
**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**

\[
C_1 = \{\neg a \lor b, \neg b \lor c\} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b).(b' + c)
\]

\[
C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c
\]

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

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

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

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

SAT answers can be checked: trusted model oracle
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.
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

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

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

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**

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

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

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
Definition

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

- increasing number of constraints
- provided some “assumptions” are satisfied
Definition
Allow the solver to decide the satisfiability of a formula with:
- increasing number of constraints
- provided some "assumptions" are satisfied

Example

\[ C = \{ s_1 \lor \neg a \lor b, s_1 \lor \neg b \lor c, s_2 \lor a, s_2 \lor \neg c \} \]

\[ C_1 \equiv C \land \neg s_1 \land s_2 \]

\[ C_2 \equiv C \land \neg s_1 \land \neg s_2 \]
Definition

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

- increasing number of constraints
- provided some "assumptions" are satisfied

Example

\[ C = \{s_1 \lor \neg a \lor b, s_1 \lor \neg b \lor c, s_2 \lor a, s_2 \lor \neg c\} \]

\[ C_1 \equiv C \land \neg s_1 \land s_2 \]

\[ C_2 \equiv C \land \neg s_1 \land \neg s_2 \]

The solver is considered as a **stateful system**: as long as the constraints are satisfiable, learn 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** \( S \).
  
  \[ (\neg a \lor b, 6) \land (\neg b \lor c, 8) \]

- Special weight \((\infty)\) for clauses that cannot be violated: **hard clauses** \( H \)

**Definition (Partial Weighted MaxSat)**

Find a model \( M \) of \( H \) that minimizes \( \text{weight}(M, S) \) 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 $S$.
  
  $$(\neg a \lor b, 6) \land (\neg b \lor c, 8)$$

- Special weight ($\infty$) for clauses that cannot be violated: hard clauses $H$
  
  $$(a, \infty) \land (\neg c, \infty)$$

**Definition (Partial Weighted MaxSat)**

Find a model $M$ of $H$ that minimizes 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* $S$. 
  $$(-a \lor b, 6) \land (-b \lor c, 8)$$

- Special weight ($\infty$) for clauses that cannot be violated: *hard clauses* $H$ 
  $$(a, \infty) \land (\neg c, \infty)$$

**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

<table>
<thead>
<tr>
<th></th>
<th>( x_6, x_2 )</th>
<th>( \neg x_6, x_2 )</th>
<th>( \neg x_2, x_1 )</th>
<th>( \neg x_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \neg x_6, x_8 )</td>
<td>( x_6, \neg x_8 )</td>
<td>( x_2, x_4 )</td>
<td>( \neg x_4, x_5 )</td>
</tr>
<tr>
<td></td>
<td>( x_7, x_5 )</td>
<td>( \neg x_7, x_5 )</td>
<td>( \neg x_5, x_3 )</td>
<td>( \neg x_3 )</td>
</tr>
</tbody>
</table>

Example CNF formula (\( k = 1 \) for each clause, not displayed)
Linear Search for solving MaxSAT

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_2, b_7$</td>
<td>$\neg x_6, x_2, b_8$</td>
<td>$\neg x_2, x_1, b_1$</td>
<td>$\neg x_1, b_2$</td>
</tr>
<tr>
<td>$\neg x_6, x_8, b_9$</td>
<td>$x_6, \neg x_8, b_{10}$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
</tr>
<tr>
<td>$x_7, x_5, b_{11}$</td>
<td>$\neg x_7, x_5, b_{12}$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
</tr>
</tbody>
</table>

Add selector or **blocking** variables $b_i$
Linear Search for solving MaxSAT

<table>
<thead>
<tr>
<th>( x_6, x_2, b_7 )</th>
<th>( \neg x_6, x_2, b_8 )</th>
<th>( \neg x_2, x_1, b_1 )</th>
<th>( \neg x_1, b_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg x_6, x_8, b_9 )</td>
<td>( x_6, \neg x_8, b_{10} )</td>
<td>( x_2, x_4, b_3 )</td>
<td>( \neg x_4, x_5, b_4 )</td>
</tr>
<tr>
<td>( x_7, x_5, b_{11} )</td>
<td>( \neg x_7, x_5, b_{12} )</td>
<td>( \neg x_5, x_3, b_5 )</td>
<td>( \neg x_3, b_6 )</td>
</tr>
</tbody>
</table>

Formula is \textbf{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

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8, b_9 \quad x_6, \neg x_8, b_{10} \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

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

Bound the number of constraints to be relaxed: \(|M \cap B| = 5\)
Linear Search for solving MaxSAT

\[
\begin{align*}
&x_6, x_2, b_7 & \neg x_6, x_2, b_8 & \neg x_2, x_1, b_1 & \neg x_1, b_2 \\
&\neg x_6, x_8, b_9 & x_6, \neg x_8, b_{10} & x_2, x_4, b_3 & \neg x_4, x_5, b_4 \\
&x_7, x_5, b_{11} & \neg x_7, x_5, b_{12} & \neg x_5, x_3, b_5 & \neg x_3, b_6 \\
\sum_{i=1}^{12} b_i & < 5
\end{align*}
\]

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

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

Bound the number of constraints to be relaxed \(|M \cap B| = 2\)
Linear Search for solving MaxSAT

<table>
<thead>
<tr>
<th>$x_6, x_2, b_7$</th>
<th>$\neg x_6, x_2, b_8$</th>
<th>$\neg x_2, x_1, b_1$</th>
<th>$\neg x_1, b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg x_6, x_8, b_9$</td>
<td>$x_6, \neg x_8, b_{10}$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
</tr>
<tr>
<td>$x_7, x_5, b_{11}$</td>
<td>$\neg x_7, x_5, b_{12}$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
</tr>
</tbody>
</table>

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

Instance is now **UNSAT**
Linear Search for solving MaxSAT

\[
\begin{align*}
&x_6, x_2, b_7 & \neg x_6, x_2, b_8 & \neg x_2, x_1, b_1 & \neg x_1, b_2 \\
&\neg x_6, x_8, b_9 & x_6, \neg x_8, b_{10} & x_2, x_4, b_3 & \neg x_4, x_5, b_4 \\
&x_7, x_5, b_{11} & \neg x_7, x_5, b_{12} & \neg x_5, x_3, b_5 & \neg x_3, b_6 \\
&\sum_{i=1}^{12} b_i < 2
\end{align*}
\]

MaxSAT solution is \(|\varphi| - |M \cap B| = 12 - 2 = 10\)
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?
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 $\{ r_i \lor C_i | C_i \in C' \}$
- Add the constraint $\sum r_i \leq 1$ to $C$
- Repeat until the formula is satisfiable
- If $MinUnsat(C) = k$, 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.
### Example CNF formula

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_2$</td>
<td>$\neg x_6, x_2$</td>
<td>$\neg x_2, x_1$</td>
<td>$\neg x_1$</td>
</tr>
<tr>
<td>$\neg x_6, x_8$</td>
<td>$x_6, \neg x_8$</td>
<td>$x_2, x_4$</td>
<td>$\neg x_4, x_5$</td>
</tr>
<tr>
<td>$x_7, x_5$</td>
<td>$\neg x_7, x_5$</td>
<td>$\neg x_5, x_3$</td>
<td>$\neg x_3$</td>
</tr>
</tbody>
</table>
Fu & Malik's Algorithm: msu1.0

<table>
<thead>
<tr>
<th>Variable Pair</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_2$</td>
<td>$\neg x_6, x_2$</td>
</tr>
<tr>
<td>$\neg x_6, x_8$</td>
<td>$x_6, \neg x_8$</td>
</tr>
<tr>
<td>$x_7, x_5$</td>
<td>$\neg x_7, x_5$</td>
</tr>
<tr>
<td>$\neg x_2, x_1$</td>
<td>$\neg x_1$</td>
</tr>
<tr>
<td>$x_2, x_4$</td>
<td>$\neg x_4, x_5$</td>
</tr>
<tr>
<td>$\neg x_5, x_3$</td>
<td>$\neg x_3$</td>
</tr>
</tbody>
</table>

Formula is **UNSAT**; Get unsat core
Fu&Malik’s Algorithm: msu1.0

\[
x_6, x_2 \quad \neg x_6, x_2 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2
\]

\[
\neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4
\]

\[
x_7, x_5 \quad \neg x_7, x_5 \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6
\]

\[
\sum_{i=1}^{6} b_i \leq 1
\]

Add blocking variables and AtMost1 constraint
Fu&Malik’s Algorithm: msu1.0

\[ x_6, x_2 \quad \neg x_6, x_2 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5 \quad \neg x_7, x_5 \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

\[ \sum_{i=1}^{6} b_i \leq 1 \]

Formula is (again) **UNSAT**; Get unsat core
Add new blocking variables and AtMost1 constraint
Fu&Malik’s Algorithm: msu1.0

\[\begin{align*}
x_6, x_2, b_7 & \quad \neg x_6, x_2, b_8 & \quad \neg x_2, x_1, b_1, b_9 & \quad \neg x_1, b_2, b_{10} \\
\neg x_6, x_8 & \quad x_6, \neg x_8 & \quad x_2, x_4, b_3 & \quad \neg x_4, x_5, b_4 \\
x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} & \quad \neg x_5, x_3, b_5, b_{13} & \quad \neg x_3, b_6, b_{14} \\
\sum_{i=1}^{6} b_i \leq 1 & \quad \sum_{i=7}^{14} b_i \leq 1
\end{align*}\]

Instance is now SAT
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?
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

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8, b_9 \quad x_6, \neg x_8, b_{10} \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

\[ \text{Cores} = \{\} \]

\[ HS = \emptyset \]
MaxHS principle

\[
\begin{align*}
\neg x_6, x_2, b_7 & \quad \neg x_6, x_2, b_8 \\
\neg x_6, x_8, b_9 & \quad x_6, \neg x_8, b_{10} \\
x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} \\
\end{align*}
\]

\[
\{\{b_1, b_2, b_3, b_4, b_5, b_6\}\} \quad HS = \{b_4\}
\]
MaxHS principle

<table>
<thead>
<tr>
<th></th>
<th>$x_6, x_2, b_7$</th>
<th>$\neg x_6, x_2, b_8$</th>
<th>$\neg x_2, x_1, b_1$</th>
<th>$\neg x_1, b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\neg x_6, x_8, b_9$</td>
<td>$x_6, \neg x_8, b_{10}$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
</tr>
<tr>
<td></td>
<td>$x_7, x_5, b_{11}$</td>
<td>$\neg x_7, x_5, b_{12}$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
</tr>
</tbody>
</table>

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

$HS = \{b_1\}$
MaxHS principle

\[\begin{align*}
x_6, x_2, b_7 & \quad \neg x_6, x_2, b_8 & \quad \neg x_2, x_1, b_1 & \quad \neg x_1, b_2 \\
\neg x_6, x_8, b_9 & \quad x_6, \neg x_8, b_{10} & \quad x_2, x_4, b_3 & \quad \neg x_4, x_5, b_4 \\
x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} & \quad \neg x_5, x_3, b_5 & \quad \neg x_3, b_6
\end{align*}\]

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

\[HS = \{b_2, b_5\}\]
MaxHS principle

<table>
<thead>
<tr>
<th>$x_6, x_2, b_7$</th>
<th>$\neg x_6, x_2, b_8$</th>
<th>$\neg x_2, x_1, b_1$</th>
<th>$\neg x_1, b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg x_6, x_8, b_9$</td>
<td>$x_6, \neg x_8, b_{10}$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
</tr>
<tr>
<td>$x_7, x_5, b_{11}$</td>
<td>$\neg x_7, x_5, b_{12}$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
</tr>
</tbody>
</table>

Instance is SAT. MaxSAT solution is $12 - |\{b_2, b_5\}| = 10$
3 ways to solve the same [optimization] problem

- 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 $x_{ij} = 1 \iff (i, j) \in A$ belongs to a cycle.

\[
\sum_{(i,j) \in A} x_{ij} = 1 \quad \text{for each } i = 1, \ldots, n \text{ (out-degree)}
\]
\[
\sum_{(i,j) \in A} x_{ij} = 1 \quad \text{for each } j = 1, \ldots, n \text{ (in-degree)}
\]
\[
\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad S \subset V, \, 2 \leq |S| \leq n - 2 \text{ (connectivity)}
\]

- 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)

But the SAT solver may be lucky!
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).
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).

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

**CEGAR**: CounterExample Guided Abstraction Refinement
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+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

Example

Hamiltonian cycle problem
CEGAR using over-abstractions

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
Recursive Explore and Check Abstraction Refinement

RECAR

\[ \text{recar}(\phi) \rightarrow \psi \leftarrow \hat{\phi} \]

\[ \text{check}(\psi) \rightarrow \psi \leftarrow \text{refine}(\psi) \]

\[ \psi \equiv \text{sat} \rightarrow \psi \]

\[ \psi \equiv \text{unsat} \rightarrow \text{UNSAT} \]

\[ \text{RC}(\phi, \hat{\phi}) \rightarrow \text{recar}(\hat{\phi}) \]

SAT

check(\psi)
sat

unsat

yes

no

yes

unsat

sat

unk.

yes

no
Recursive Explore and Check Abstraction Refinement

Called *RECAR* [LLdLM17]

Inspired by CEGAR [CGJ+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. $\exists n \in \mathbb{N} \ s.t. \ \text{RC}(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$ is false.
∃n ∈ N s.t. RC(under^n(φ), under^{n+1}(φ)) is false.

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

- □ϕ means ϕ is necessarily true
- ◊ϕ means ϕ is possibly true

◊ϕ ↔ ¬□¬ϕ

□ϕ ↔ ¬◊¬ϕ
Preliminaries: Kripke Structure

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

Kripke Structure $[\text{Kri59}]$

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

- $W$, a non-empty set of possible worlds
- $R$, a binary relation on $W$
- $V$, a function that associates 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 $W$
Definition (Satisfaction Relation)

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

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

$K$ that satisfied a formula $\phi$ will be called “Kripke model of $\phi$”
Preliminaries: Example of a Kripke Structure

\[ \phi_1 = \Box(\bullet) \]

\[ \phi_2 = \Box\Diamond(\bullet) \]

\[ \phi_3 = \Diamond(\bullet \land \Diamond \neg \bullet) \]

\[ \phi_4 = (\bullet \lor \bullet \lor \bullet) \]

\[ \phi_5 = \Diamond\Diamond(\bullet \land \Box \neg \bullet) \]

Figure: Example \( K \)
Preliminaries: Example of a Kripke Structure

\[ \phi_1 = \square(\bullet) \]

\[ \phi_2 = \square\Diamond(\bullet) \]

\[ \phi_3 = \Diamond(\bullet \land \Diamond\neg\bullet) \]

\[ \phi_4 = (\bullet \lor \bullet \lor \bullet) \]

\[ \phi_5 = \Diamond\Diamond(\bullet \land \square\neg\bullet) \]

Figure: Example $\mathcal{K}$
MoSaiC

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

RECAR Assumptions:
1. Function 'check' is sound, complete and terminates
2. isSAT(\hat{\varphi}) \Rightarrow isSAT(refine(\hat{\varphi}))
3. \exists n \in \mathbb{N} \text{ s.t. } refine(n)(\hat{\varphi}) \equiv \text{sat} \varphi
4. isUNSAT(\tilde{\varphi}) \Rightarrow isUNSAT(\varphi)
5. \exists n \in \mathbb{N} \text{ s.t. } RC(\text{under } n(\varphi), \text{under } n+1(\varphi)) \text{ is false}
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. \( \text{RC}(\text{under}^n(\phi), \text{under}^{n+1}(\phi)) \) is false
φ always in NNF and over(φ, i) in CNF using Tseitin’s translation

\[
\begin{align*}
\text{over}(\phi, n) &= \text{over}'(\phi, 0, n) \\
\text{over}'(p_k, i, n) &= p_{k,i} \quad \text{over}'(\neg p_k, i, n) = \neg p_{k,i} \\
\text{over}'(\Box \phi, i, n) &= \bigwedge_{j=0}^{n} (r_{i,j} \rightarrow \text{over}'(\phi, j, n)) \\
\text{over}'(\Diamond \phi, i, n) &= \bigvee_{j=0}^{n} (r_{i,j} \land \text{over}'(\phi, j, n))
\end{align*}
\]

- \( p_{k,i} \) 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
Are there known bounds on the number of worlds to consider?

\[ UB(\phi) = \text{Atom}(\phi) \cdot \text{depth}(\phi) \] 

where \( \text{Atom}(\phi) \) denotes the number of propositional variables and \( \text{depth}(\phi) \) the modal depth of \( \phi \).
Are there known bounds on the number of worlds to consider?

Yes, but quite large: $UB(\phi) = Atom(\phi)^{depth(\phi)}$ [SM97]

where $Atom(\phi)$ denotes the number of propositional variables
and $depth(\phi)$ 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. \( \exists n \in \mathbb{N} \text{ s.t. } \text{refine}^n(\hat{\phi}) \equiv^? \text{sat } \phi \)
4. \( \text{isUNSAT}(\check{\phi}) \) implies \( \text{isUNSAT}(\phi) \)
5. \( \exists n \in \mathbb{N} \text{ s.t. } RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi)) \) is false
Let’s take an example, with $\chi$ huge but satisfiable...

Worst case for CEGAR using our ‘over’ function
Modern SAT solvers returns ‘the reason’ why a formula with $n$ worlds is unsatisfiable ($core = \{s_1, s_2\}$)
We want to cut what is not part of the ‘unsatisfiability’ \( (s_i \notin \text{core}) \)

We just create \( \hat{\phi} \) 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)

\[
\begin{align*}
\text{under}(p, \text{core}) &= p \\
\text{under}(\neg p, \text{core}) &= \neg p \\
\text{under}(\Box \phi, \text{core}) &= \Box(\text{under}(\phi, \text{core})) \\
\text{under}(\Diamond \phi, \text{core}) &= \Diamond(\text{under}(\phi, \text{core})) \\
\text{under}((\phi \land \psi), \text{core}) &= \text{under}(\phi, \text{core}) \land \text{under}(\psi, \text{core}) \\
\text{under}((\psi \lor \chi), \text{core}) &= \begin{cases} 
\text{under}(\chi, \text{core}) & \text{if } \psi = \neg s_i, s_i \in \text{core} \\
\top & \text{if } \psi = \neg s_i, s_i \notin \text{core} \\
\text{under}(\psi, \text{core}) \lor \text{under}(\chi, \text{core}) & \text{otherwise}
\end{cases}
\end{align*}
\]

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. $\exists n \in \mathbb{N} \text{ s.t. } \text{refine}^n(\hat{\phi}) \equiv \text{sat} \phi$
4. $\text{isUNSAT}(\check{\phi})$ implies $\text{isUNSAT}(\phi)$
5. $\exists n \in \mathbb{N} \text{ s.t. } RC(\text{under}^n(\phi), \text{under}^{n+1}(\phi))$ is false
MoSaiC: RECAR for Modal Logic K

\[ MoSaiC(\phi) \]

\[ n \leftarrow 1 \quad \psi \leftarrow over(\phi, n) \]

\[ SAT \quad sat \]

\[ glucose(\psi) \]

\[ UNSAT \quad unsat \]

\[ n > UB(\phi) \]

\[ yes \]

\[ \phi \leftarrow under(\phi, core) \]

\[ no \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]

\[ no \]

\[ \check{\phi} \leftarrow under(\phi, core) \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow max(|M|, n + 1) \]

\[ unsat \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]

\[ unsat \]

\[ \check{\phi} \leftarrow under(\phi, core) \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]

\[ unsat \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]

\[ unsat \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]

\[ unsat \]

\[ \check{\phi} \]

\[ \check{\phi} = \phi \]

\[ yes \]

\[ n \leftarrow n + 1 \]
MoSaiC: RECAR for Modal Logic K
MoSaiC: RECAR for Modal Logic K

The diagram shows the execution time (in seconds) for solving instances of Modal Logic K using various tools. The x-axis represents the number of instances solved, and the y-axis represents the execution time. The tools compared include:

- CEGAR
- FaCT++
- BDDTab
- KS
- Km2SAT
- Vampire
- InKreSAT
- *SAT
- Spartacus
- RECAR

The graph illustrates how each tool performs across different numbers of instances solved, with RECAR showing a notable performance compared to the other tools.
Explanation of the Cactus-Plot

- Spartacus
- RECAR
- Generating Time
- SAT Time

Execution time (s) vs. #instances solved
Some tweaks improve the results
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}@cril.univ-artois.fr

SAT+SMT school, IIT Bombay, India, 10 December 2019
Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.
Counterexample-guided abstraction refinement for symbolic model checking.

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

Saul Kripke.
A completeness theorem in modal logic.
Richard E. Ladner.  
The Computational Complexity of Provability in Systems of Modal Propositional Logic.  

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.  

Steven David Prestwich.  
SAT problems with chains of dependent variables.  