SAT solver essentials, SAT modeling
Encodings

Gilles Audemard

VTSA School - Liege - 2021

Thanks to N. Szczepanski and L. Simon
- Take a given problem, transform it into SAT
- If the resulting formula is SAT, transform it in the original form
SAT is perfect to encode circuit

Widely used: EDA (Electronic Design Automation)

It is the same for cryptanalysis, plenty of xor

Dedicated solvers: CryptoMinisat
Encoding circuits

- 6 boolean variables: $x_1, x_2, x_3, y_1, y_2, y_3$
- input variables $x_i$
- Auxiliary variables $y_i$

$y_3$
Exercise: Encoding gates
Encoding a variable with finite domain

- How to encode a variable $x_i$ that can have some value inside the set \{a, b, c\}

- A first solution
  - A boolean variable that represent $x_i = k$ \(x^i_a, x^i_b, x^i_c\)
  - A clause (at least) saying that $x_i$ has to be assign to a value \(x^i_a \lor x^i_b \lor x^i_c\)
  - Clauses (at most) saying that $x_i$ can not have two different values \((\overline{x^i_a} \lor \overline{x^i_b}), (\overline{x^i_a} \lor \overline{x^i_c})\)
How to encode a variable $x_i$ that can have some value inside the set \{a, b, c\}

A first solution

- A boolean variable that represent $x_i = k$ $x_i^a, x_i^b, x_i^c$
- A clause (at least) saying that $x_i$ has to be assign to a value $x_i^a \lor x_i^b \lor x_i^c$
- Clauses (at most) saying that $x_i$ can not have two different values $(x_i^a \lor \overline{x_i^b}), (\overline{x_i^a} \lor x_i^c)$...

For each variable $x_i$ of size $n$, One needs

- $n$ boolean variables
- $1 + \frac{n \times (n-1)}{2}$ clauses
A different encoding for the at most one constraint

- \( x_1 + x_2 \ldots + x_n \leq 1 \)
- The previous encoding is quadratic
- Can we do better? Yes... By adding auxiliary variables

- Let \( s_i \) be a fresh variable indicating that the count has reached 1 by \( i \) (for \( 1 \leq i < n \))
- The resulting set of clauses are
  - \( x_1 \rightarrow s_1 \)
  - \( (x_i \lor s_{i-1}) \rightarrow s_i \) for \( 1 < i < n \)
  - \( s_{i-1} \rightarrow \neg x_i \) for \( 1 < i \leq n \)

- \( n - 1 \) additional variables
- \( 1 + 2 \times (n - 2) + n - 1 = 3 \times n - 4 \) clauses
Exercise : Graph coloring
Symmetry breaking

- Many problems contain symmetries
- Break them can help
- An example

Pigeon hole problem

- We have to put \( n \) pigeons in \( (n - 1) \) pigeon holes: only one pigeon per pigeon hole
- Trivially UNSAT
- Yet....
Encoding pigeonhole

- Variables $p_h^p$ : Pigeon $p$ is in hole $h$

- Clauses
  - Pigeon $i$ is in one hole : $p_1^i \lor p_2^i \ldots p_{(n-1)}^i$
  - Pigeons $i$ and $j$ can not be inside the same hole : $\overline{p_i^p} \lor \overline{p_j^p}$

- It is time to test it

- We can add some constraints
  - All pigeons are the same, holes too : **Symmetries**
  - By breaking these symmetries we have a very simple problem
Suppose an encoded problem hard to solve

It results from
- the initial problem itself?
- the encoding?

In [Hertel07], authors proposed two different encodings of the same problem and exhibited an exponential separation between them.

In 90s, an important challenge:
- Solve instances par32
- Specialized solvers were developed to this end (eqSatz, [Li 00])
- In [Bailleux 03], authors showed that the problem is not very difficult, but it is the case for the SAT encoding.

Beware of the encoding
Many problems contain at least, at most or exactly $k$ constraints

Different encodings were proposed last years

More or less efficient, depending the number of variables and the value of $k$
At least \( k \) encodings

- Many problems contain at least, at most or exactly \( k \) constraints
- Different encodings were proposed last years
- More or less efficient, depending the number of variables and the value of \( k \)

Pairwise encoding

- Related to the simple at-most 1 encoding
- For each \( i_1 \ldots i_{k+1} \in 1 \ldots n \), add the clause \( \neg x_{i_1} \lor \ldots \lor \neg x_{i_{k+1}} \)
At least k encodings

- Many problems contain at least, at most or exactly $k$ constraints
- Different encodings were proposed last years
- More or less efficient, depending the number of variables and the value of $k$

Pairwise encoding

- Related to the simple at-most 1 encoding
- For each $i_1 \ldots i_{k+1} \in 1 \ldots n$, add the clause $\neg x_{i_1} \lor \ldots \lor \neg x_{i_{k+1}}$

Generalized sequential encoding

- Related to the second encoding of at most one constraint
- Add a variable $s_{ij}$ true if the sum has reached $j$ with the first $i$ variables
At least k encodings

- Many problems contain at least, at most or exactly $k$ constraints
- Different encodings were proposed last years
- More or less efficient, depending the number of variables and the value of $k$

Pairwise encoding

- Related to the simple at-most 1 encoding
- For each $i_1 \ldots i_{k+1} \in 1 \ldots n$, add the clause $\neg x_{i_1} \lor \ldots \lor \neg x_{i_{k+1}}$

Generalized sequential encoding

- Related to the second encoding of at most one constraint
- Add a variable $s_{ij}$ true if the sum has reached $j$ with the first $i$ variables

Other encodings

- Sorting networks, BDD....
- PySAT propose different encodings for cardinality constraints
Exercise : Peacable Queens
- One of the first domains where SAT usage outperformed previous ad-hoc techniques
- Given a finite state automaton, one wants to check if a given property is verified
- Used in formal verification

**Diagram:**

- Nodes: `d`, `c`, `a`, `b`
- Edges: `t_1`, `t_2`, `t_3`, `t_4`

**Property:** Starting from `d`, is it possible to reach `a`?
How it works

- In BMC, instead of proofing that the property is true, find an counter example within less than \( k \) steps
  - At step \( i \), we are in state \( s_i \)

- SAT Formula: \( l(s_0) \land T(s_0, s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land \overline{p(s_k)} \)

- If formula is false, we have a bug

- If formula is UNSAT : we know... nothing !!

- In this case, we need to increase the bound \( k \)
Example

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_1^0, t_1^1, t_1^2, \ldots$
- $t_2^0, t_2^1, t_2^2, \ldots$

![Diagram](image-url)
Étape

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_1^0, t_1^1, t_1^2, \ldots$
- $t_2^0, t_2^1, t_2^2, \ldots$
Example

Étape  0

\[
\begin{array}{c|c}
  a & F \\
  b & F \\
  e & F \\
  d & T \\
\end{array}
\]

- \( a_0, a_1, \ldots \)
- \( b_0, \ldots, c_0, \ldots, d_0, \ldots \)
- \( t^0_1, t^1_1, t^2_1 \ldots \)
- \( t^0_2, t^1_2, t^2_2 \ldots \)
Example

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_1^0, t_1^1, t_1^2, \ldots$
- $t_2^0, t_2^1, t_2^2, \ldots$

<table>
<thead>
<tr>
<th>Étape</th>
<th>0</th>
<th>$t_3^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$e$</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Example

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t^0_1, t^1_1, t^2_1, \ldots$
- $t^0_2, t^1_2, t^2_2, \ldots$

<table>
<thead>
<tr>
<th>Étape</th>
<th>$t^0_3$</th>
<th>$t^1_3$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$F$</td>
<td>$F$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$F$</td>
<td>$F$</td>
<td></td>
</tr>
<tr>
<td>$e$</td>
<td>$F$</td>
<td>$T$</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>$T$</td>
<td>$F$</td>
<td></td>
</tr>
</tbody>
</table>
Example

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_0^0, t_1^1, t_2^2 \ldots$
- $t_0^0, t_1^1, t_2^2 \ldots$

<table>
<thead>
<tr>
<th>Étape</th>
<th>0</th>
<th>$t_0^0$</th>
<th>1</th>
<th>$t_1^1$</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$b$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$e$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$d$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Example

- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_1^0, t_1^1, t_1^2, \ldots$
- $t_2^0, t_2^1, t_2^2, \ldots$

<table>
<thead>
<tr>
<th>Étape</th>
<th>0</th>
<th>$t_3^0$</th>
<th>1</th>
<th>$t_4^1$</th>
<th>2</th>
<th>$t_1^2$</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>$b$</td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>T</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>$e$</td>
<td>F</td>
<td></td>
<td>T</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>$d$</td>
<td>T</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>
Example: Resulting clauses

State variables: $S \times (k + 1)$

Transitions variables: $T \times k$

Clauses for states: $1 + (S \times (S - 1)) \times (k + 1)$

Clauses for transitions: $(1 + T \times (T - 1)) \times k$

Clauses for performing a transition: $S \times k$

Clauses for after a transition: $S \times k$

28 variables and 82 clauses

Initial state: $d_0$

In only one state at each step:

\[
\begin{align*}
(a_0 \lor b_0 \lor c_0 \lor d_0) & \land (\neg a_0 \lor \neg b_0) \\
(a_1 \lor b_1 \lor c_1 \lor d_1) & \land (\neg a_1 \lor \neg b_1) \\
(a_2 \lor b_2 \lor c_2 \lor d_2) & \land (\neg a_2 \lor \neg b_2) \\
(a_3 \lor b_3 \lor c_3 \lor d_3) & \land (\neg a_3 \lor \neg b_3)
\end{align*}
\]

Transition can only occur if the current state is true:

\[
\begin{align*}
(\neg d_0 \lor t_0^3) & \land (\neg c_0 \lor t_0^4) \\
(\neg d_1 \lor t_1^3) & \land (\neg c_1 \lor t_1^4) \\
\vdots \\
(\neg d_n \lor t_n^3) & \land (\neg c_n \lor t_n^4)
\end{align*}
\]

After a transition, one reaches a given state:

\[
\begin{align*}
(\neg t_0^3 \lor c_1) & \land (\neg t_0^4 \lor b_1) \\
(\neg t_1^3 \lor c_2) & \land (\neg t_1^4 \lor b_2) \\
\vdots
\end{align*}
\]
Example : Resulting clauses

Initial state :
\[ d_0 \]

In only one state at each step :
\[ \begin{align*}
(a_0 \lor b_0 \lor c_0 \lor d_0) \land (-a_0 \lor -b_0) & \land \ldots (-c_0 \lor -d_0) \\
(a_1 \lor b_1 \lor c_1 \lor d_1) \land (-a_1 \lor -b_1) & \land \ldots (-c_1 \lor -d_1) \\
(a_2 \lor b_2 \lor c_2 \lor d_2) \land (-a_2 \lor -b_2) & \land \ldots (-c_2 \lor -d_2) \\
(a_3 \lor b_3 \lor c_3 \lor d_3) \land (-a_3 \lor -b_3) & \land \ldots (-c_3 \lor -d_3)
\end{align*} \]
Example: Resulting clauses

Initial state:

\[ d_0 \]

In only one state at each step:

\[
\begin{align*}
(a_0 &\lor b_0 &\lor c_0 &\lor d_0) \land (\neg a_0 &\lor \neg b_0) \land \ldots (\neg c_0 &\lor \neg d_0) \\
(a_1 &\lor b_1 &\lor c_1 &\lor d_1) \land (\neg a_1 &\lor \neg b_1) \land \ldots (\neg c_1 &\lor \neg d_1) \\
(a_2 &\lor b_2 &\lor c_2 &\lor d_2) \land (\neg a_2 &\lor \neg b_2) \land \ldots (\neg c_2 &\lor \neg d_2) \\
(a_3 &\lor b_3 &\lor c_3 &\lor d_3) \land (\neg a_3 &\lor \neg b_3) \land \ldots (\neg c_3 &\lor \neg d_3)
\end{align*}
\]

Only one transition per step:

\[
\begin{align*}
(t_1^0 &\lor t_2^0 &\lor t_3^0 &\lor t_4^0) \land (\neg t_1^0 &\lor \neg t_2^0) \land \ldots \\
(t_1^1 &\lor t_2^1 &\lor t_3^1 &\lor t_4^1) \land (\neg t_1^1 &\lor \neg t_2^1) \land \ldots \\
(t_1^2 &\lor t_2^2 &\lor t_3^2 &\lor t_4^2) \land (\neg t_1^2 &\lor \neg t_2^2) \land \ldots
\end{align*}
\]
Example: Resulting clauses

Initial state:
\[d_0\]

In only one state at each step:
\[(a_0 \lor b_0 \lor c_0 \lor d_0) \land (\neg a_0 \lor \neg b_0) \ldots (\neg c_0 \lor \neg d_0)\]
\[(a_1 \lor b_1 \lor c_1 \lor d_1) \land (\neg a_1 \lor \neg b_1) \ldots (\neg c_1 \lor \neg d_1)\]
\[(a_2 \lor b_2 \lor c_2 \lor d_2) \land (\neg a_2 \lor \neg b_2) \ldots (\neg c_2 \lor \neg d_2)\]
\[(a_3 \lor b_3 \lor c_3 \lor d_3) \land (\neg a_3 \lor \neg b_3) \ldots (\neg c_3 \lor \neg d_3)\]

Only one transition per step:
\[(t_0^0 \lor t_2^0 \lor t_3^0 \lor t_4^0) \land (\neg t_1^0 \lor \neg t_2^0) \ldots \]
\[(t_1^1 \lor t_2^1 \lor t_3^1 \lor t_4^1) \land (\neg t_1^1 \lor \neg t_2^1) \ldots \]
\[(t_2^2 \lor t_2^2 \lor t_3^2 \lor t_4^2) \land (\neg t_1^2 \lor \neg t_2^2) \ldots \]

Transition can only occur if the current state is true
\[(\neg d_0 \lor t_3^0) \land (\neg c_0 \lor t_4^0) \ldots \]
\[(\neg d_1 \lor t_3^1) \land (\neg c_1 \lor t_4^1) \ldots \]
\[\ldots \]
Example: Resulting clauses

Initial state:
\[ d_0 \]

In only one state at each step:
\[
(a_0 \lor b_0 \lor c_0 \lor d_0) \land (\neg a_0 \lor \neg b_0) \land \ldots \land (\neg c_0 \lor \neg d_0) \\
(a_1 \lor b_1 \lor c_1 \lor d_1) \land (\neg a_1 \lor \neg b_1) \land \ldots \land (\neg c_1 \lor \neg d_1) \\
(a_2 \lor b_2 \lor c_2 \lor d_2) \land (\neg a_2 \lor \neg b_2) \land \ldots \land (\neg c_2 \lor \neg d_2) \\
(a_3 \lor b_3 \lor c_3 \lor d_3) \land (\neg a_3 \lor \neg b_3) \land \ldots \land (\neg c_3 \lor \neg d_3)
\]

Only one transition per step:
\[
(t_0^1 \lor t_0^2 \lor t_0^3 \lor t_0^4) \land (\neg t_0^1 \lor \neg t_0^2) \land \ldots \\
(t_1^1 \lor t_1^2 \lor t_1^3 \lor t_1^4) \land (\neg t_1^1 \lor \neg t_1^2) \land \ldots \\
(t_2^1 \lor t_2^2 \lor t_2^3 \lor t_2^4) \land (\neg t_2^1 \lor \neg t_2^2) \land \ldots
\]

Transition can only occur if the current state is true
\[
(\neg d_0 \lor t_0^0) \land (\neg c_0 \lor t_0^4) \land \ldots \\
(\neg d_1 \lor t_1^0) \land (\neg c_1 \lor t_1^4) \land \ldots \\
\ldots
\]

After a transition, one reaches a given state:
\[
(\neg t_3^0 \lor c_1) \land (\neg t_4^0 \lor b_1) \land \ldots \\
(\neg t_3^1 \lor c_2) \land (\neg t_4^1 \lor b_2) \land \ldots
\]
Example : Resulting clauses

S states and T transitions
State variables : \( S \times (k + 1) \)
Transitions variables : \( T \times k \)
Clauses for states : \( 1 + (1 + \frac{S \times (S-1)}{2}) \times (k + 1) \)
Clauses for transitions : \( (1 + \frac{T \times (T-1)}{2}) \times k \)
Clauses for performing a transition : \( S \times k \)
Clauses pour after a transition : \( S \times k \)

28 variables and 82 clauses

Initial state :
\( d_0 \)

In only one state at each step :
\( (a_0 \lor b_0 \lor c_0 \lor d_0) \land (\neg a_0 \lor \neg b_0) \ldots (\neg c_0 \lor \neg d_0) \)
\( (a_1 \lor b_1 \lor c_1 \lor d_1) \land (\neg a_1 \lor \neg b_1) \ldots (\neg c_1 \lor \neg d_1) \)
\( (a_2 \lor b_2 \lor c_2 \lor d_2) \land (\neg a_2 \lor \neg b_2) \ldots (\neg c_2 \lor \neg d_2) \)
\( (a_3 \lor b_3 \lor c_3 \lor d_3) \land (\neg a_3 \lor \neg b_3) \ldots (\neg c_3 \lor \neg d_3) \)

Only one transition per step :
\( (t_0^1 \lor t_2^0 \lor t_3^0 \lor t_4^0) \land (\neg t_0^1 \lor \neg t_2^0) \ldots \)
\( (t_0^1 \lor t_2^0 \lor t_3^1 \lor t_4^1) \land (\neg t_0^1 \lor \neg t_2^1) \ldots \)
\( (t_0^2 \lor t_2^0 \lor t_3^2 \lor t_4^2) \land (\neg t_0^2 \lor \neg t_2^2) \ldots \)

Transition can only occur if the current state is true
\( (\neg d_0 \lor t_3^0) \land (\neg c_0 \lor t_4^0) \ldots \)
\( (\neg d_1 \lor t_3^1) \land (\neg c_1 \lor t_4^1) \ldots \)
\ldots

After a transition, one reaches a given state :
\( (\neg t_3^0 \lor c_1) \land (\neg t_4^0 \lor b_1) \ldots \)
\( (\neg t_3^1 \lor c_2) \land (\neg t_4^1 \lor b_2) \ldots \)
The order encoding

- Proposed to solve scheduling problems [Tamura 09].
- Very efficient
- Variables $x \in \{l_x, 2, 3 \ldots u_x\}$
- $l_x$ and $u_x$ are lower and upper bound for $x$
- We use $(u_x - l_x + 1)$ boolean variables
  - $x_k$ represents $x \leq k$ (for $l_x - 1 \leq k \leq u_x$)
- And the following clauses
  - $\overline{x_{l_x - 1}}$
  - $x_{u_x}$
  - $\overline{x_{k - 1}} \lor x_k$ for $l_x \leq k \leq u_x$
- Clauses represent intervals

<table>
<thead>
<tr>
<th>$x_{l_x - 1}$</th>
<th>$x_{l_x}$</th>
<th>$x_{l_x + 1}$</th>
<th>$\ldots$</th>
<th>$x_{u_x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\ldots$</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\ldots$</td>
<td>1</td>
</tr>
</tbody>
</table>
Particularly suited for scheduling, knapsack problems.

Let $x, y \in \{2, 3, 4, 5, 6\}$ be 2 variables.

How to encode the constraint $x + y \leq 7$. 
Particularly suited for scheduling, knapsack problems.

Let $x, y \in \{2, 3, 4, 5, 6\}$ be 2 variables.

How to encode the constraint $x + y \leq 7$
MaxSAT: definitions

- Optimisation problem associated to SAT
- Maximize the number of satisfied clauses
- Generalisation:
  - Weighted MaxSAT: clauses have weights. Maximize the weights of satisfied clauses
  - Partial MaxSAT: Some clauses are hard and must be satisfied (no two different lectures in the same room). Others are soft (I want to start to work at 10am).
  - Weighted Partial MaxSAT: soft clauses have weights

- Solution: an assignment that satisfies all hard clauses
- Cost of a solution: sum of weights
- Optimal solution: maximize the weights
Exercise: shortest path in a *labyrinth*
Constraint Programming