SAT solver essentials, SAT modeling
Encodings

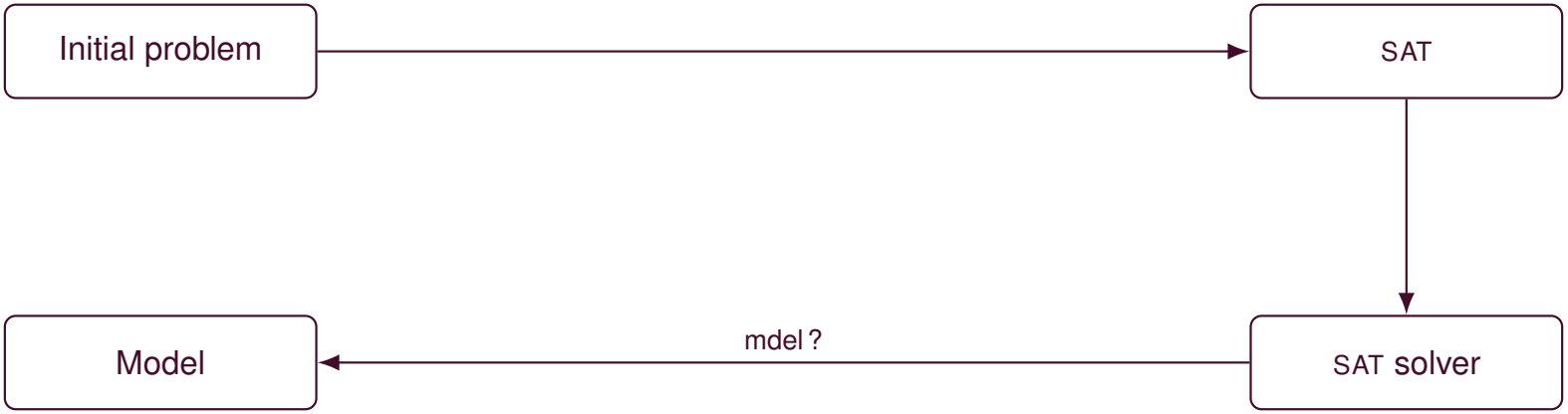
Gilles Audemard

VTSA School - Liege - 2021

Thanks to N. Szczepanski and L. Simon

Introduction

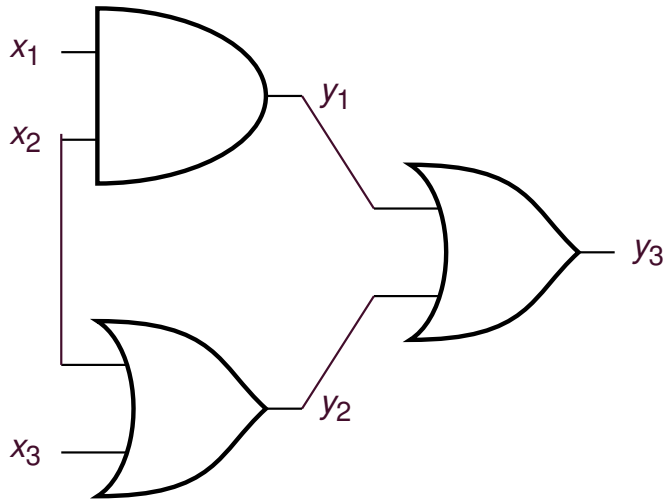
- Take a given problem, transform it into SAT
- If the resulting formula is SAT, transform it in the original form



Encoding circuits

- 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

$(y_1 \vee \overline{x_1} \vee \overline{x_2})$
$(\overline{y_1} \vee x_1)$
$(\overline{y_1} \vee x_2)$

$(\overline{y_2} \vee x_2 \vee x_3)$
$(y_2 \vee \overline{x_2})$
$(y_2 \vee \overline{x_3})$

$(\overline{y_3} \vee y_1 \vee y_2)$
$(y_3 \vee \overline{y_1})$
$(y_3 \vee \overline{y_2})$

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 $x_i = k$ $\quad x_a^i, x_b^i, x_c^i$
  - A clause (*at least*) saying that x_i has to be assign to a value $x_a^i \vee x_b^i \vee x_c^i$
  - Clauses (*at most*) saying that x_i can not have two different values $(\overline{x_a^i} \vee \overline{x_b^i})$, $(\overline{x_a^i} \vee \overline{x_c^i})\ldots$

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 $x_i = k$ $\quad x_a^i, x_b^i, x_c^i$
  - A clause (*at least*) saying that x_i has to be assign to a value $x_a^i \vee x_b^i \vee x_c^i$
  - Clauses (*at most*) saying that x_i can not have two different values $(\overline{x_a^i} \vee \overline{x_b^i}), (\overline{x_a^i} \vee \overline{x_c^i}) \ldots$

- 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

- $x_1 + x_2 \ldots + x_n <= 1$
- The previous encoding is quadratic
- Can we do better ? Yes... By adding auxilliary 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 clauses are
  - $x_1 \rightarrow s_1$
  - $(x_i \vee s_{i-1}) \rightarrow s_i$ 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 : $p_1^i \vee p_2^i \ldots p_{(n-1)}^i$
  - Pigeons i and j can not be inside the same hole : $\overline{p_i^p} \vee \overline{p_j^p}$
- 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

Dangerous encoding

- 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 themp
- 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

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

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 $\neg x_{i_1} \vee \ldots \vee \neg x i_{k+1}$

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 $\neg x_{i_1} \vee \ldots \vee \neg x i_{k+1}$

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 $\neg x_{i_1} \vee \ldots \vee \neg x i_{k+1}$

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

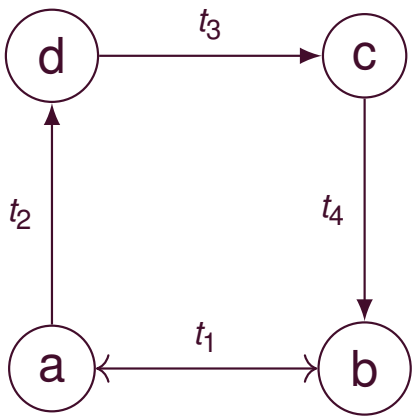
Other encodings

- Sorting networks, BDD....
- PySAT propose different encodings for cardinality constraints

Exercise : Peacable Queens

BMC

- One of the first domain where SAT usage outperformed previous ad-hoc techiques
- Given an finite state automaton, one wants to check if a given property is verified
- Used in formal verification

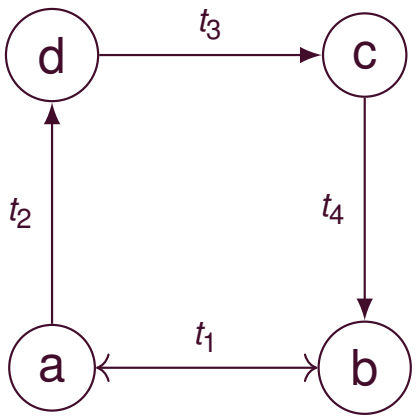


property : Starting from d, is it possible to reach a?

How it works

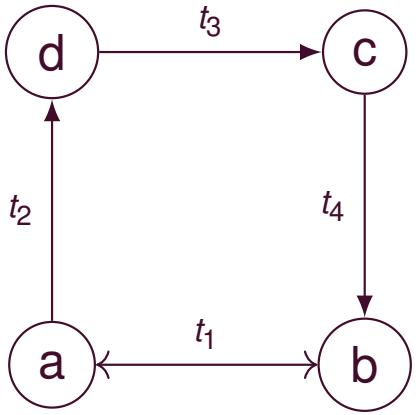
- In BMC, instead of proofing that the property is true, find an counter example within less than k steps
  - At step i, we are in state s_i
- SAT Formula : $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \overline{p(s_k)}$
- If formula is false, we have a bug
- If formula is UNSAT : we know... nothing ! !
- In this case, we need to increase the bound k

Example



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

Example

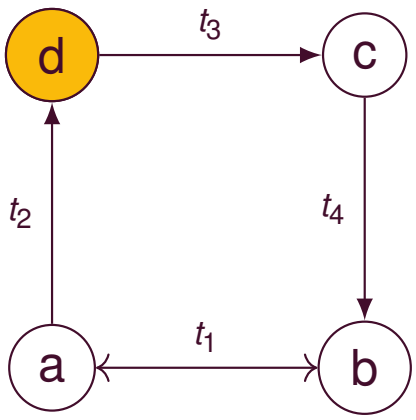


- $a_0, a_1, \ldots$
- $b_0, \ldots, c_0, \ldots, d_0, \ldots$
- $t_1^0, t_1^1, t_1^2 \ldots$
- $t_2^0, t_2^1, t_2^2 \ldots$

Étape

a
b
e
d

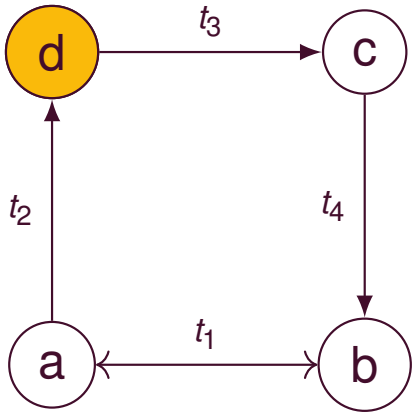
Example



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

Étape	0
a	F
b	F
e	F
d	T

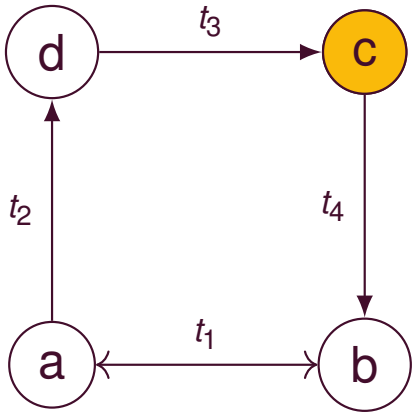
Example



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

Étape	0	t_3^0
a	F	
b	F	
e	F	
d	T	

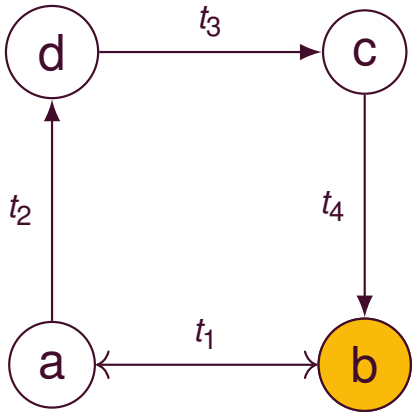
Example



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

Étape	0	t_3^0	1
a	F		F
b	F		F
e	F		T
d	T		F

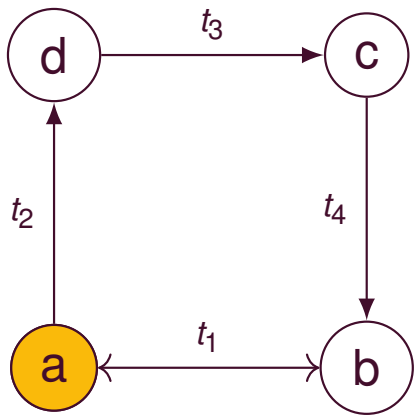
Example



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

Étape	0	t_3^0	1	t_4^1	2
a	F		F		F
b	F		F		T
e	F		T		F
d	T		F		F

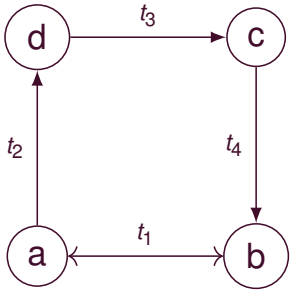
Example



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

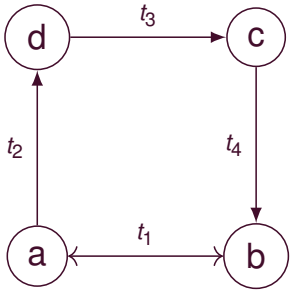
Étape	0	t_3^0	1	t_4^1	2	t_1^2	3
a	F		F		F		T
b	F		F		T		F
e	F		T		F		F
d	T		F		F		F

Example : Resulting clauses



Initial state :
d_0

Example : Resulting clauses



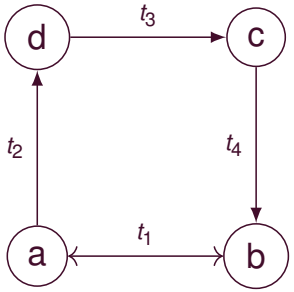
Initial state :

d_0

In only one state at each step :

$(a_0 \vee b_0 \vee c_0 \vee d_0) \wedge (\neg a_0 \vee \neg b_0) \ldots (\neg c_0 \vee \neg d_0)$
$(a_1 \vee b_1 \vee c_1 \vee d_1) \wedge (\neg a_1 \vee \neg b_1) \ldots (\neg c_1 \vee \neg d_1)$
$(a_2 \vee b_2 \vee c_2 \vee d_2) \wedge (\neg a_2 \vee \neg b_2) \ldots (\neg c_2 \vee \neg d_2)$
$(a_3 \vee b_3 \vee c_3 \vee d_3) \wedge (\neg a_3 \vee \neg b_3) \ldots (\neg c_3 \vee \neg d_3)$

Example : Resulting clauses



Initial state :

d_0

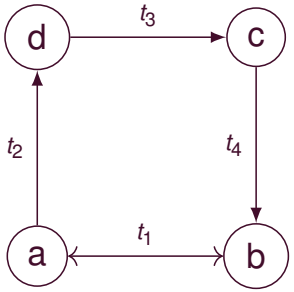
In only one state at each step :

$(a_0 \vee b_0 \vee c_0 \vee d_0) \wedge (\neg a_0 \vee \neg b_0) \ldots (\neg c_0 \vee \neg d_0)$
$(a_1 \vee b_1 \vee c_1 \vee d_1) \wedge (\neg a_1 \vee \neg b_1) \ldots (\neg c_1 \vee \neg d_1)$
$(a_2 \vee b_2 \vee c_2 \vee d_2) \wedge (\neg a_2 \vee \neg b_2) \ldots (\neg c_2 \vee \neg d_2)$
$(a_3 \vee b_3 \vee c_3 \vee d_3) \wedge (\neg a_3 \vee \neg b_3) \ldots (\neg c_3 \vee \neg d_3)$

Only one transition per step :

$(t_1^0 \vee t_2^0 \vee t_3^0 \vee t_4^0) \wedge (\neg t_1^0 \vee \neg t_2^0) \ldots$
$(t_1^1 \vee t_2^1 \vee t_3^1 \vee t_4^1) \wedge (\neg t_1^1 \vee \neg t_2^1) \ldots$
$(t_1^2 \vee t_2^2 \vee t_3^2 \vee t_4^2) \wedge (\neg t_1^2 \vee \neg t_2^2) \ldots$

Example : Resulting clauses



Initial state :

d_0

In only one state at each step :

$(a_0 \vee b_0 \vee c_0 \vee d_0) \wedge (\neg a_0 \vee \neg b_0) \ldots (\neg c_0 \vee \neg d_0)$
$(a_1 \vee b_1 \vee c_1 \vee d_1) \wedge (\neg a_1 \vee \neg b_1) \ldots (\neg c_1 \vee \neg d_1)$
$(a_2 \vee b_2 \vee c_2 \vee d_2) \wedge (\neg a_2 \vee \neg b_2) \ldots (\neg c_2 \vee \neg d_2)$
$(a_3 \vee b_3 \vee c_3 \vee d_3) \wedge (\neg a_3 \vee \neg b_3) \ldots (\neg c_3 \vee \neg d_3)$

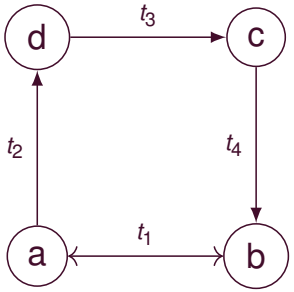
Only one transition per step :

$(t_1^0 \vee t_2^0 \vee t_3^0 \vee t_4^0) \wedge (\neg t_1^0 \vee \neg t_2^0) \ldots$
$(t_1^1 \vee t_2^1 \vee t_3^1 \vee t_4^1) \wedge (\neg t_1^1 \vee \neg t_2^1) \ldots$
$(t_1^2 \vee t_2^2 \vee t_3^2 \vee t_4^2) \wedge (\neg t_1^2 \vee \neg t_2^2) \ldots$

Transition can only occur if the current state is true

$(\neg d_0 \vee t_3^0) \wedge (\neg c_O \vee t_4^0) \ldots$
$(\neg d_1 \vee t_3^1) \wedge (\neg c_1 \vee t_4^1) \ldots$
$\ldots$

Example : Resulting clauses



Initial state :

d_0

In only one state at each step :

$(a_0 \vee b_0 \vee c_0 \vee d_0) \wedge (\neg a_0 \vee \neg b_0) \ldots (\neg c_0 \vee \neg d_0)$
$(a_1 \vee b_1 \vee c_1 \vee d_1) \wedge (\neg a_1 \vee \neg b_1) \ldots (\neg c_1 \vee \neg d_1)$
$(a_2 \vee b_2 \vee c_2 \vee d_2) \wedge (\neg a_2 \vee \neg b_2) \ldots (\neg c_2 \vee \neg d_2)$
$(a_3 \vee b_3 \vee c_3 \vee d_3) \wedge (\neg a_3 \vee \neg b_3) \ldots (\neg c_3 \vee \neg d_3)$

Only one transition per step :

$(t_1^0 \vee t_2^0 \vee t_3^0 \vee t_4^0) \wedge (\neg t_1^0 \vee \neg t_2^0) \ldots$
$(t_1^1 \vee t_2^1 \vee t_3^1 \vee t_4^1) \wedge (\neg t_1^1 \vee \neg t_2^1) \ldots$
$(t_1^2 \vee t_2^2 \vee t_3^2 \vee t_4^2) \wedge (\neg t_1^2 \vee \neg t_2^2) \ldots$

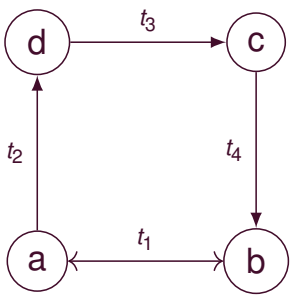
Transition can only occur if the current state is true

$(\neg d_0 \vee t_3^0) \wedge (\neg c_O \vee t_4^0) \ldots$
$(\neg d_1 \vee t_3^1) \wedge (\neg c_1 \vee t_4^1) \ldots$
$\ldots$

After a transition, one reaches a given state :

$(\neg t_3^0 \vee c_1) \wedge (\neg t_4^0 \vee b_1) \ldots$
$(\neg t_3^1 \vee c_2) \wedge (\neg t_4^1 \vee b_2) \ldots$

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 :

$(a_0 \vee b_0 \vee c_0 \vee d_0) \wedge (\neg a_0 \vee \neg b_0) \ldots (\neg c_0 \vee \neg d_0)$
$(a_1 \vee b_1 \vee c_1 \vee d_1) \wedge (\neg a_1 \vee \neg b_1) \ldots (\neg c_1 \vee \neg d_1)$
$(a_2 \vee b_2 \vee c_2 \vee d_2) \wedge (\neg a_2 \vee \neg b_2) \ldots (\neg c_2 \vee \neg d_2)$
$(a_3 \vee b_3 \vee c_3 \vee d_3) \wedge (\neg a_3 \vee \neg b_3) \ldots (\neg c_3 \vee \neg d_3)$

Only one transition per step :

$(t_1^0 \vee t_2^0 \vee t_3^0 \vee t_4^0) \wedge (\neg t_1^0 \vee \neg t_2^0) \ldots$
$(t_1^1 \vee t_2^1 \vee t_3^1 \vee t_4^1) \wedge (\neg t_1^1 \vee \neg t_2^1) \ldots$
$(t_1^2 \vee t_2^2 \vee t_3^2 \vee t_4^2) \wedge (\neg t_1^2 \vee \neg t_2^2) \ldots$

Transition can only occur if the current state is true

$(\neg d_0 \vee t_3^0) \wedge (\neg c_O \vee t_4^0) \ldots$
$(\neg d_1 \vee t_3^1) \wedge (\neg c_1 \vee t_4^1) \ldots$
$\ldots$

After a transition, one reaches a given state :

$(\neg t_3^0 \vee c_1) \wedge (\neg t_4^0 \vee b_1) \ldots$
$(\neg t_3^1 \vee c_2) \wedge (\neg t_4^1 \vee b_2) \ldots$

The order encoding

- Proposed to solve schedulling problems [Tamura 09].
- Very efficient
- Variables $x \in \{l_x, 2, 3 \ldots u_x\}$
- l_x et 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}
  - $\overline{x_{k-1}} \vee x_k$ for $l_x \leq k \leq u_x$
- Clauses represent intervals

x_{l_x-1}	x_{l_x}	x_{l_x+1}	$\ldots$	x_{u_k}
0	1	1		1
0	1	1		1
0	0	1		1
			$\ldots$	
0	0	0		1

Order encodings... 2

- Particularly suited for scheduling, knapsack problems.....
- Let $x, y \in \{2, 3, 4, 5, 6\}$ be 2 variables
- How to encode the constraint $x + y \leq 7$

Order encodings... 2

- Particularly suited for scheduling, knapsack problems.....
- Let $x, y \in \{2, 3, 4, 5, 6\}$ be 2 variables
- How to encode the constraint $x + y \leq 7$

$$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \qquad y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6$$

$$\overline{x_1} \quad x_6 \quad (\overline{x_1} \vee x_2) \quad (\overline{x_2} \vee x_3) \quad (\overline{x_3} \vee x_4) \quad (\overline{x_4} \vee x_5) \quad (\overline{x_5} \vee x_6)$$

$$\overline{y_1} \quad y_6 \quad (\overline{y_1} \vee y_2) \quad (\overline{y_2} \vee y_3) \quad (\overline{y_3} \vee y_4) \quad (\overline{y_4} \vee y_5) \quad (\overline{y_5} \vee y_6)$$

$$(x_1 \vee y_5) \quad (x_2 \vee y_4) \quad (x_3 \vee y_3) \quad (x_4 \vee y_2) \quad (x_5 \vee y_1)$$

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 tow 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