Algorithms for Computational Logic

Introduction

Emmanuel Hebrard (adapted from **João Marques Silva**, Inês Lynce and Vasco Manquinho)

Outline

1. **Applications of SAT**

Outline

CNF Encodings

- CNF-SAT is **NP**-complete, and therefore as *powerful* as general SAT as a language
- Most research has focused on algorithms for CNF-SAT
- However, if polynomial encodings necessarily exist, they are not always easy to find
- Not all encodings are equal
  - What is a *good* encoding ?
  - How to design a *good* encoding

- From SAT to CNF-SAT via the rules of Boolean algebra:

$$(a \implies (c \wedge d)) \vee (b \implies (c \wedge e))$$

- Decompose the implications

$$(a \implies c) \wedge (a \implies d)) \vee ((b \implies c) \wedge (b \implies e))$$

- Rearrange disjunctions and conjunctions (conjunctions and disjunctions are distributive)

$$((a \implies c) \vee (b \implies c)) \wedge ((a \implies c) \vee (b \implies e)) \wedge ((a \implies d) \vee (b \implies c)) \wedge ((a \implies d) \vee (b \implies e))$$

- Rewrite implications as disjunctions

$$(\bar{a} \vee c \vee \bar{b}) \wedge (\bar{a} \vee c \vee \bar{b} \vee e) \wedge (\bar{a} \vee d \vee \bar{b} \vee c) \wedge (\bar{a} \vee d \vee \bar{b} \vee e)$$

- Remove subsumed clauses

$$(\bar{a} \vee c \vee \bar{b}) \wedge (\bar{a} \vee d \vee \bar{b} \vee e)$$

- Distributing is not efficient:

$$(x_1^1 \wedge x_2^1 \wedge \ldots \wedge x_k^1) \vee (x_1^2 \wedge x_2^2 \wedge \ldots \wedge x_k^2) \vee \ldots \vee (x_1^n \wedge x_2^n \wedge \ldots \wedge x_k^n)$$

- Up to k^n clauses of size up to n
- *Tseitin's encoding is polynomial in every case.* Idea ?
- Add extra variables

- Rewrite implications as disjunctions
- For every *nested* conjunction $(a \wedge \bar{b} \wedge c)$, introduce a fresh variable f and the clauses $(a \wedge \bar{b} \wedge c) \iff f$:

$$(a \wedge \bar{b} \wedge c) \implies f : \quad (\bar{a} \vee b \vee \bar{c} \vee f)$$

$$f \implies (a \wedge \bar{b} \wedge c) : \quad \begin{cases} \bar{f} \vee a \\ \bar{f} \vee \bar{b} \\ \bar{f} \vee c \end{cases}$$

- For instance for $(a \implies (c \wedge d)) \vee (b \implies (c \wedge e)) = (\bar{a} \vee (c \wedge d)) \vee (\bar{b} \vee (c \wedge e))$:
- $(\bar{c} \vee \bar{d} \vee f_1) \wedge (\bar{f_1} \vee c) \wedge (\bar{f_1} \vee d) \wedge$
  $(\bar{c} \vee \bar{e} \vee f_2) \wedge (\bar{f_2} \vee c) \wedge (\bar{f_2} \vee e) \wedge$
  $(\bar{a} \vee f_1) \wedge (\bar{b} \vee f_2)$

Playing Sudoku

	2		1	7	8		3	
	4		3		2		9	
1								6
		8	6		3	5		
3								4
		6	7		9	2		
9								2
	8		9		1		6	
	1		4	3	6		5	

- Fill empty cells such that each row, each colum and each 3x3 grid contains all of the digits 1 to 9.

Constraint Model for Sudoku

			1		5		6	8
						7		1
9		1					3	
		7		2	6			
5								3
			8	7		4		
	3					8		5
1		5						
7	9		4		1			

medium

- Modeling the problem with integer variables:
  - Rows: $i = 1, \dots, 9$
  - Columns: $j = 1, \dots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \dots, 9\}$, $i, j \in \{1, \dots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \dots, 9\}$, for $j < k \in \{1, \dots, 9\}$: $v_{i,j} \neq v_{i,k}$
  - Each value used exactly once in each column:
    - For $j \in \{1, \dots, 9\}$, for $i < k \in \{1, \dots, 9\}$: $v_{i,j} \neq v_{k,j}$
  - Each value used exactly once in each 3×3 sub-grid:
    - For $i, j, k, l \in \{1, 9\}$, if ($k \neq i$ OR $l \neq j$) AND $\lceil \frac{i}{3} \rceil = \lceil \frac{k}{3} \rceil$ AND $\lceil \frac{j}{3} \rceil = \lceil \frac{l}{3} \rceil$: $v_{i,j} \neq v_{k,l}$
  - Each clue corresponds to a variable assignment:

$$v_{1,4} = 1, v_{1,6} = 5, v_{1,8} = 6, v_{1,9} = 8, v_{2,7} = 7, v_{2,9} = 1$$
$$v_{3,1} = 9, v_{3,3} = 1, v_{3,8} = 3, v_{4,3} = 7, v_{4,5} = 2, v_{4,6} = 6, \dots$$

Constraint Satisfaction Problems

Constraint Satisfaction Problem (CSP)

Data: a triplet $\mathcal{X}, \mathcal{D}, \mathcal{C}$ where:

- $\mathcal{X}$ is a ordered set of *variables*
- $\mathcal{D}$ is a *domain*
- $\mathcal{C}$ is a set of *constraints*, where for $c \in \mathcal{C}$:
  - its *scope* $S(c)$ is a list of variables
  - its *relation* $R(c)$ is a subset of $\mathcal{D}^{|S(c)|}$

Question: does there exist a solution $\sigma \in \mathcal{D}^{|\mathcal{X}|}$ such that for every $c \in \mathcal{C}$, $\sigma(S(c)) \in R(c)$?

Projection

The projection $\sigma(X)$ of a tuple σ on a set of variables $X = (x_{i_1}, \dots, x_{i_k}) \subseteq \mathcal{X}$ as the tuple $(\sigma(x_{i_1}), \dots, \sigma(x_{i_k}))$

- Example: the constraint $x + y = z$ (on the Boolean ring)

x	y	z	$S(x + y = z)$
0	0	0	
0	1	1	$R(x + y = z)$
1	0	1	
1	1	0	

Sudoku Example

Sudoku

- $\mathcal{X} = (v_{1,1}, \ldots, v_{9,9})$
- $\mathcal{D} = \{1, \ldots, 9\}$
- $\mathcal{C}$: inequalities on rows, columns and subsquares; clues

x	y	$S(x \neq y)$
1	2	
1	3	
$\vdots$	$\vdots$	
1	9	
2	1	$R(x \neq y)$
2	3	
$\vdots$	$\vdots$	
2	9	
$\vdots$	$\vdots$	
9	8	

Encoding Integer Domains

- Variable x with domain $\{0, \ldots, n-1\}$:
  - Log encoding: Boolean variables $x_0, \ldots, x_{\lfloor \log n \rfloor}$ stands for $x = \sum_{j=0}^{\lfloor \log n \rfloor} 2^j$
  - Direct encoding: Boolean variable x_j stands for variable x takes value j
- Direct encoding requires *consistency clauses* because it is not a bijection:
  - $\sum_{j=1}^{n} x_j \geq 1$: encode with $(x_1 \vee x_2 \vee \ldots \vee x_n)$
  - $\sum_{j=1}^{n} x_j \leq 1$ encode with: *Pairwise encoding* or *Sequential counters*

Pairwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with pairwise incompatibilities:
  - $x = i$ implies $x \neq j$

$$\bigwedge_{1 \leq i < j \leq n} (\bar{x}_i \vee \bar{x}_j)$$

  - $\mathcal{O}(n^2)$ binary clauses
  - Encoding relations is easy and efficient:

- Unit propagation of $x = 3$

j	1	2	3	4	5	6	7	8	9
$x_j(x = j)$	0	0	1	0	0	0	0	0	0

- One clause to forbid every *non-tuple* in the relation $R(c)$, e.g. for $x \neq y$:

x	y	conflict clauses
1	1	$\bar{x}_1 \vee \bar{y}_1$
2	2	$\bar{x}_2 \vee \bar{y}_2$
3	3	$\bar{x}_3 \vee \bar{y}_3$
4	4	$\bar{x}_4 \vee \bar{y}_4$
5	5	$\bar{x}_5 \vee \bar{y}_5$
6	6	$\bar{x}_6 \vee \bar{y}_6$
$\vdots$	$\vdots$	$\vdots$

Sequential Counter Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter
- Introduce Boolean variables $s_1, \ldots, s_{n-1}$ with s_i standing for $x \leq i$
  - $x = i$ implies $x \leq i$
  - $x = i$ implies $x > i - 1$
  - $x \leq i - 1$ implies $x \leq i$

$$\bigwedge_{1 < i < n} ((\neg x_i \vee s_i) \wedge (\neg x_i \vee \neg s_{i-1})) \wedge (\neg s_{i-1} \vee s_i))$$
$$\wedge (\neg x_1 \vee s_1) \wedge (\neg x_n \vee \neg s_{n-1})$$

  - $\mathcal{O}(n)$ binary clauses ; $\mathcal{O}(n)$ auxiliary variables

- Unit propagation of $x = 3$

j	1	2	3	4	5	6	7	8	9
$x_j(x = j)$	0	0	1	0	0	0	0	0	0
$s_j(x \leq j)$	0	0	1	1	1	1	1	1	1

- Unit propagation of $3 \leq x \leq 6$

j	1	2	3	4	5	6	7	8	9
$x_j(x = j)$	0	0					0	0	0
$s_j(x \leq j)$	0	0				1	1	1	1

Encoding Relations in Bitwise Encoding

x			y			conflict clauses
x_{22}	x_{21}	x_{20}	y_{22}	y_{21}	y_{20}	
0	0	0	0	0	0	$x_{22} \vee x_{21} \vee x_{20} \vee y_{22} \vee y_{21} \vee y_{20}$
0	0	1	0	0	1	$x_{22} \vee x_{21} \vee \bar{x_{20}} \vee y_{22} \vee y_{21} \vee \bar{y_{20}}$

- Assume $x = 1$, that is : $x_{22} = 0$, $x_{21} = 0$ and $x_{20} = 1$
- The clause $(x_{22} \vee x_{21} \vee \bar{x_{20}} \vee y_{22} \vee y_{21} \vee \bar{y_{20}})$ does not unit propagate !
- Unit propagation is *weaker* on the log encoding
- Notion of *Arc Consistency*

Arc Consistency

- Let $\mathcal{X}$ be a set of variables and $\mathcal{D}$ be a domain:
  - $\mathcal{D}(x)$ is the set of possible values for variable $x \in \mathcal{X}$

Arc Consistency

A constraint c is *Arc Consistent* on domain $\mathcal{D}$ if and only if for every $x \in S(c)$ and for every $j \in \mathcal{D}(x)$, there exists a tuple $\sigma \in R(c)$ such that $\sigma(x) = j$.

- *Achieving Arc Consistency* on domain $\mathcal{D}$ with respect to constraint c corresponds to reducing $\mathcal{D}$ to the maximum $\mathcal{D}' \subseteq \mathcal{D}$ such that $\mathcal{D}'$ is arc consistent
  - If $\mathcal{D}'$ is empty there is no solution satisfying relation c on domain $\mathcal{D}$

	x	y	z
	1	2	3
	1	3	4
	1	4	5
	1	5	6
	1	6	7
	2	3	5
	2	4	6
	2	5	7
	3	4	7
$R(c)(\cdot)$	$\{1, 2, 3\}$	$\{2, 3, 4, 5, 6\}$	$\{3, 4, 5, 6, 7\}$
$\mathcal{D}(\cdot)$	$\{1, 2, 3, 4, 5\}$	$\{1, 2, 5, 6, 7\}$	$\{1, 2, 3, 4, 5, 6\}$

Comparison of Encodings

- The size of the encoding is an important feature
  - Sequential counters are more concise than pairwise incompatibilities
- We have seen that unit propagation might not be the same on two logically equivalent encodings
  - Log vs. direct encoding of the constraint $x \neq y$
- We can ask whether a Boolean encoding of a constraint c achieves arc consistency on domain $\mathcal{D}$
  - Defined in the same way using the natural isomorphism between domain encodings

Encodings of the "Less than" Constraint

x	$\neq$	y
1		2
1		3
1		4
2		3
2		4
3		4

Negative encoding

one clause for every *non-tuple* in $R(c)$

$$(\bar{x}_1 \vee \bar{y}_1)$$
$$(\bar{x}_2 \vee \bar{y}_1)$$
$$(\bar{x}_2 \vee \bar{y}_2)$$
$$(\bar{x}_3 \vee \bar{y}_1)$$
$$(\bar{x}_3 \vee \bar{y}_2)$$
$$(\bar{x}_3 \vee \bar{y}_3)$$
$$(\bar{x}_4 \vee \bar{y}_1)$$
$$(\bar{x}_4 \vee \bar{y}_2)$$
$$(\bar{x}_4 \vee \bar{y}_3)$$
$$(\bar{x}_4 \vee \bar{y}_4)$$

Suport encoding

one clause for every value, encoding its *support values* in $R(c)$

$$(\bar{y}_1)$$
$$(\bar{y}_2 \vee x_1)$$
$$(\bar{y}_3 \vee x_2 \vee x_1)$$
$$(\bar{y}_4 \vee x_3 \vee x_2 \vee x_1)$$
$$(\bar{x}_1 \vee y_2 \vee y_3 \vee y_4)$$
$$(\bar{x}_2 \vee y_3 \vee y_4)$$
$$(\bar{x}_3 \vee y_4)$$
$$(\bar{x}_4)$$

- The support encoding unit propagates $\bar{y}_1$ and $\bar{x}_4$, whereas the negative encoding does not
- Suppose that we know $x \neq 1$ ($\bar{x}_1$ is a new true literal)
- Unit propagation on the support encoding achieves arc consistency

- Support encoding is only defined for binary relations
- For dense relations, negative encoding is efficient.
  - For instance the constraint $x \neq y$ contains $\frac{n-1}{n}$ tuples, and the negative encoding achieves arc consistency
- Alternative to negative encoding for sparse constraints ?

Tseitin encoding

one extra variable and $1 + |\sigma|$ clauses for every *tuple* $\sigma \in R(c)$

x	$\neq$ y
1	2
1	3
1	4
2	3
2	4
3	4

$$(\bar{x_1} \vee \bar{y_2} \vee z_{1,2}) \wedge (\bar{z_{1,2}} \vee x_1) \wedge (\bar{z_{1,2}} \vee y_2)$$
$$z_{1,3} \iff (x_1 \wedge y_3)$$
$$z_{1,4} \iff (x_1 \wedge y_4)$$
$$z_{2,3} \iff (x_2 \wedge y_3)$$
$$z_{2,4} \iff (x_2 \wedge y_4)$$
$$z_{3,4} \iff (x_3 \wedge y_4)$$
$$(z_{1,2} \vee z_{1,3} \vee z_{1,4} \vee z_{2,3} \vee z_{2,4} \vee z_{3,4})$$
$$(\bar{x_1} \vee z_{1,2} \vee z_{1,3} \vee z_{1,4}) \quad (\bar{y_1})$$
$$(\bar{x_2} \vee z_{2,3} \vee z_{2,4}) \quad (\bar{y_2} \vee z_{1,2})$$
$$(\bar{x_3} \vee z_{3,4}) \quad (\bar{y_3} \vee z_{1,3} \vee z_{2,3})$$
$$(\bar{x_4}) \quad (\bar{y_4} \vee z_{1,4} \vee z_{2,4} \vee z_{3,4})$$

- Consider a constraint c of *arity* $|S(c)| := a$
- Let $S = \prod_{x \in S(c)} \mathcal{D}(x)$ be the set of *valid* tuples (allowed by the domain $\mathcal{D}$) with $|S| := s$
- Let $R(c) \cap S$ be the set of *consistent* tuples (valid and allowed by the constraint) with $|R(c) \cap S| := t$
- The negative encoding requires $\Theta(a(s-t))$ space
- Tseitin's encoding requires $\Theta(at)$ space

Tseitin encoding and Arc Consistency

Unit propagation on Tseitin's encoding is an *optimal* algorithm to achieve Arc Consistency on a table constraint.

- Tseitin's encoding takes linear space
- Unit propagation takes linear time
- There is no sublinear algorithm to achieve arc consistency

Upgrading a Linux Distribution

- Let $U = \{p_1, \ldots, p_n\}$ be all versions of all linux packages
- Let $C \subseteq U^2$ be a set of *conflicts* (packages pairwise incompatible)
- For every package $p_i \in U$, we have:
  - a set D_i of *dependencies* with $d \in D_i$ a set of packages such at least one of them is required for package p
- An installation profile $P \subseteq U$ is *valid* iff, for every $p_i \in P$:
  - $C_i \cap P = \emptyset$ (there is no incompatibilities)
  - For each $d \in D_i, \; d \cap P \neq \emptyset$ (the dependencies are satisfied)
- An installation profile $P \subseteq U$ is *non-regressive with respect to profile* P^o iff for each $p_i \in P^o, \; V_i \cap P \neq \emptyset$

Upgradeability problem

Given a current installation profile P^o and a package p, *does there exist two sets of packages P^+ and P^- such that $P \cup P^+ \setminus P^-$ is a valid non-regressive installation profile and contains p*

Encoding

- A variable p_i for every $p_i \in U$: *whether package p_i should be in the installation profile*
- Constraints
  - **Compatibilities:** $\bar{p}_i \vee \bar{p}_j$ $\qquad \forall p_i \in U, \; \forall (p_i, p_j) \in C$
  - **Dependencies:** $\bar{p}_i \vee \bigvee_{p_j \in d} p_j$ $\qquad \forall p_i \in d, \; \forall d \in D_i$
  - **Non-regression:** $p_i \vee \bigvee_{p_j \in V_i} p_j$ $\qquad \forall p_i \in P^o$

Objective

- *Minimize the number of changes*
- Introduce new variables to encode the delta

$$
\begin{array}{llll}
p_i^{\Delta} \iff \bar{p}_i : & p_i^{\Delta} \vee p_i \wedge \bar{p_i^{\Delta}} \vee \bar{p}_i & & \forall p_i \in P^o \\
p_i^{\Delta} \iff p_i : & p_i^{\Delta} \vee \bar{p}_i \wedge \bar{p_i^{\Delta}} \vee p_i & & \forall p_i \notin P^o
\end{array}
$$

- Optimization is usually done by successive constraints
  - Top-down: $\sum_{i=1}^{n} p_i < ub_0$; $\sum_{i=1}^{n} p_i < ub_1$; $\ldots$; $\sum_{i=1}^{n} p_i < ub_k$ (with ub_i a *feasible* number of packages)
  - Bottom-up: $\sum_{i=1}^{n} p_i > lb_0$; $\sum_{i=1}^{n} p_i > lb_1$; $\ldots$; $\sum_{i=1}^{n} p_i > lb_k$ (with ub_i a *infeasible* number of packages)
  - Binary search
- How to encode a *cardinality* constraint?

Cardinality Constraints

- How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$
  - Special case when k = 1 $\sum_{j=1}^{n} x_j \leq 1$
    - ★ AtMost1 constraints was the subject of the previous class
- Solution #1:
  - Use native PB solver, e.g. BSOLO, PBS, Galena, Pueblo, etc.
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - ★ E.g. Minisat+, Open-WBO, QMaxSat, MSUnCore, WPM2, etc.
- Solution #2:
  - Encode cardinality constraints to CNF
  - Use SAT solver

General Cardinality Constraints

- General form: $\sum_{j=1}^{n} x_j \leq k$ (or $\sum_{j=1}^{n} x_j \geq k$)
  - Sequential counters [S05]
    - ★ Clauses/Variables: $\mathcal{O}(n\,k)$
  - BDDs [ES06]
    - ★ Clauses/Variables: $\mathcal{O}(n\,k)$
  - Sorting networks [ES06]
    - ★ Clauses/Variables: $\mathcal{O}(n \log^2 n)$
  - Cardinality Networks: [ANORC09,ANORC11a]
    - ★ Clauses/Variables: $\mathcal{O}(n \log^2 k)$
  - Totalizer [BB03]
    - ★ Clauses: $\mathcal{O}(nk)$, Variables: $\mathcal{O}(n \log k)$
  - Pairwise Cardinality Networks [CZI10]
  - ...

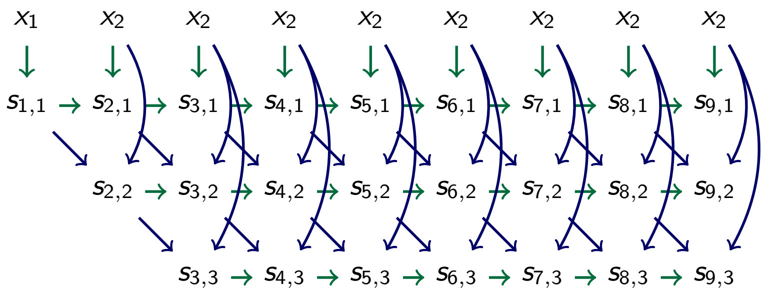
Sequential Counter Encoding

Assume the general form: $\sum_{i=1}^{n} x_i \leq k$

- For each variable x_i, create k additional variables $s_{i,j}$ that are used as counters.
  - $s_{i,j} = 1$ if at least j of variables $\{x_1 \ldots x_i\}$ are assigned value 1
  - $s_{i,j} = 0$ if at most $j-1$ of variables $\{x_1 \ldots x_i\}$ are assigned value 1

Encoding:

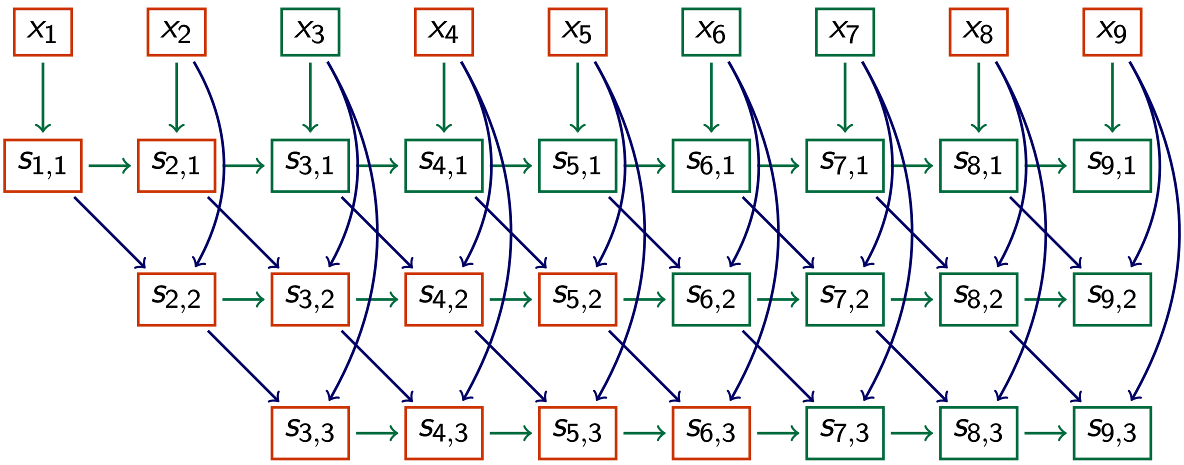
$(\neg s_{1,j})$	$\forall j : 1 < j \leq k$
$(\neg x_i \vee s_{i,1})$	$\forall i : 1 \leq i < n$
$(\neg s_{i-1,j} \vee s_{i,j})$	$\forall i,j : 1 \leq i < n, 1 < j \leq k$
$(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j})$	$\forall i,j : 1 < i < n, 1 < j \leq k$
$(\neg x_i \vee \neg s_{i-1,k})$	$\forall i : 1 < i \leq n$



Sequential Counter and Arc Consistency

- Does the sequential counter encoding achieve arc consistency on the cardinality constraint?
- When is the constraint $\sum_{j=1}^{n} x_j \leq k$ *not* arc consistent?
  1. When more than k variables are true
  2. When exactly k variables are true and at least 1 variable can be $\mathtt{true}$
- The value '**false**' is always arc consistent
- In all other cases, unassigned variables are indistinguishable: so any one of them can be $\mathtt{true}$ (in particular if all other are **false**)
- Let see if unit propagation forbids (1) and (2)

Sequential Counter Encoding is *AC*



- $(\neg x_7 \vee \neg s_{6,3}) \wedge (\neg x_8 \vee \neg s_{7,3}) \wedge (\neg x_9 \vee \neg s_{8,3})$
- $(\neg x_6 \vee \neg s_{5,2} \vee s_{6,3})$
- $(\neg x_4 \vee \neg s_{3,1} \vee s_{4,2}) \wedge (\neg x_5 \vee \neg s_{4,1} \vee s_{5,2})$

Totalizer Encoding

Totalizer Encoding

- CNF encoding for cardinality constraints $\sum_{i=1}^{n} x_i \leq k$
- Count in unary how many of the n variables $(x_1 \ldots x_n)$ are assigned value 1
- $O(n \log n)$ new variables
- $O(n^2)$ new clauses
  - Can be improved to $O(n\ k)$

Totalizer Encoding

$(O : o_1, o_2, o_3, o_4, o_5 : 5)$

$(A : a_1, a_2 : 2)$ $(B : b_1, b_2, b_3 : 3)$

$(C : x_1 : 1)$ $(D : x_2 : 1)$ $(E : x_3 : 1)$ $(F : f_1, f_2 : 2)$

$(G : x_4 : 1)$ $(H : x_5 : 1)$

- Visualize the encoding as a tree
  - Each node is $(name : variables : sum)$
  - Literals are at the leaves
  - Each node counts in unary how many leaves are assigned to 1 in its subtree
  - Example: if $b_2 = 1$, then 2 of the leaves (x_3, x_4, x_5) are assigned to 1
- Root node has the output variables $(o_1 \ldots o_5)$ that count how many variables are assigned to 1
- To encode $x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$ just set $o_4 = 0$ and $o_5 = 0$

$$(P : p_1, p_2, \ldots, p_{n_1} : n_1)$$

$$(Q : q_1, q_2, \ldots, q_{n_2} : n_2) \qquad (R : r_1, r_2, \ldots, r_{n_3} : n_3)$$

- Suppose that an intermediate node P that counts up to n_1 has two child nodes Q and R that count up to n_2 and n_3, respectively
- Note that $n_1 = n_2 + n_3$

Encoding:

$$\bigwedge_{\substack{0 \leq \alpha \leq n_2 \\ 0 \leq \beta \leq n_3 \\ 0 \leq \sigma \leq n_1 \\ \alpha + \beta = \sigma}} \neg q_\alpha \vee \neg r_\beta \vee p_\sigma \quad \text{where, } p_0 = q_0 = r_0 = 1$$