Representations for Automated Reasoning

Ruben Martins

Carnegie
Mellon
University

http://www.cs.cmu.edu/~mheule/15816-f22/

Automated Reasoning and Satisfiability
September 7, 2022

rubenm@cs.cmu.edu

Basic Constraints

Solver Input

Representing Integers

Cardinality Constraints

Lazy Encodings

Basic Constraints

Solver Input

Representing Integers

Cardinality Constraints

Lazy Encodings

rubenm@cs.cmu.edu

AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtLeastOne}\,(x_1, \ldots, x_n)$$

into SAT?

Hint: This is easy...

rubenm@cs.cmu.edu

AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtLeastOne}\ (x_1, \ldots, x_n)$$

into SAT?

Hint: This is easy...

$$(x_1 \lor x_2 \lor \cdots \lor x_n)$$

rubenm@cs.cmu.edu

Exclusive OR (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

rubenm@cs.cmu.edu

Exclusive OR (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

$\mathrm{XOR}(x_1, \ldots, x_n)$ is *true* when an odd number of x_i is assigned to *true*. Consider the case with two literals:

x_1	x_2	$\mathrm{XOR}(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

rubenm@cs.cmu.edu

Exclusive OR (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

$\mathrm{XOR}(x_1, \ldots, x_n)$ is *true* when an odd number of x_i is assigned to *true*. Consider the case with two literals:

x_1	x_2	$\mathrm{XOR}(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$$

rubenm@cs.cmu.edu

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

rubenm@cs.cmu.edu

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

$$\begin{aligned} \mathrm{XOR}(x_1, x_2, x_3) \quad = \quad & (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge \\ & (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \end{aligned}$$

rubenm@cs.cmu.edu

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\text{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

$$\begin{aligned} \text{XOR}(x_1, x_2, x_3) \quad = \quad & (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge \\ & (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \end{aligned}$$

Question: How many solutions does this formula have?

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

$$\begin{aligned}\mathrm{XOR}(x_1, x_2, x_3) \;=\;& (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge \\ & (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3)\end{aligned}$$

Question: How many solutions does this formula have? 4

rubenm@cs.cmu.edu

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

Can we encode large XORs with fewer clauses?

rubenm@cs.cmu.edu

Exclusive OR (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

Can we encode large XORs with fewer clauses?

Make it compact: XOR $(x_1, x_2, y) \wedge$ XOR $(\overline{y}, x_3, \ldots, x_n)$

Tradeoff: increase the number of variables but decreases the number of clauses!

rubenm@cs.cmu.edu

AtMostOne: Pairwise Encoding

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\ (x_1, \ldots, x_n)$$

into SAT?

rubenm@cs.cmu.edu

AtMostOne: Pairwise Encoding

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires $n(n-1)/2$ binary clauses:

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

rubenm@cs.cmu.edu

AtMostOne: Pairwise Encoding

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires $n(n-1)/2$ binary clauses:

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

Is it possible to use fewer clauses?

rubenm@cs.cmu.edu

AtMostOne: Linear Encoding

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

rubenm@cs.cmu.edu

AtMostOne: Linear Encoding

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\ (x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

By splitting the constraint using additional variables. Apply the direct encoding if $n \leq 4$ otherwise replace $\textsc{AtMostOne}$ $(x_1, \ldots, x_n)$ by

$$\textsc{AtMostOne}\ (x_1, x_2, x_3, y) \land \textsc{AtMostOne}\ (\overline{y}, x_4, \ldots, x_n)$$

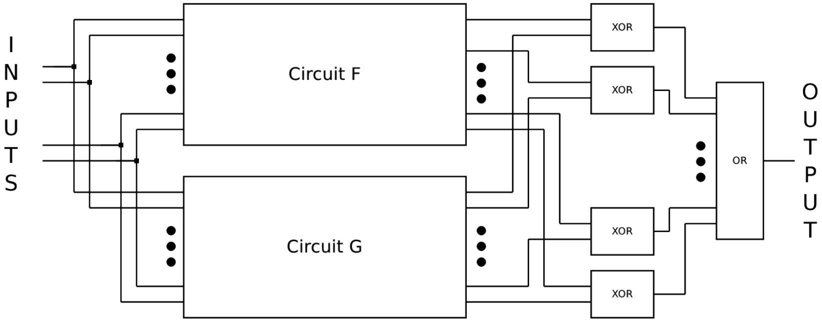
resulting in $3n - 6$ clauses and $(n - 3)/2$ new variables

rubenm@cs.cmu.edu

AtMostOne: Equivalence

How to show that two encodings of $\textsc{AtMostOne}(x_1, x_2)$ are equivalent?

If we have a circuit representation of each encoding then we can use a miter circuit to show that for the same inputs, the output variables are equivalent:



rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Question: Is φ_1 equivalent to φ_2?

Note: $\varphi_1 \leftrightarrow \varphi_2$ is valid if $\neg\varphi_1 \wedge \varphi_2$ and $\varphi_1 \wedge \neg\varphi_2$ are unsatisfiable.

rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\neg\varphi_1 \wedge \varphi_2$ unsatisfiable?

Note: $\neg\varphi_1 \equiv x_1 \wedge x_2$

rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\neg\varphi_1 \wedge \varphi_2$ unsatisfiable? yes!

Note: $\neg\varphi_1 \equiv x_1 \wedge x_2$

rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\varphi_1 \wedge \neg\varphi_2$ unsatisfiable?

Note: $\neg\varphi_2 \equiv (x_1 \vee y) \wedge (x_1 \vee x_2) \wedge (\overline{y} \vee x_2)$

rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\varphi_1 \wedge \neg\varphi_2$ unsatisfiable? no!

Note: $\neg\varphi_2 \equiv (x_1 \vee y) \wedge (x_1 \vee x_2) \wedge (\overline{y} \vee x_2)$

rubenm@cs.cmu.edu

AtMostOne: Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$ $\overline{y} \vee \overline{x}_2$

φ_1 and φ_2 are equisatisfiable:

- φ_1 is satisfiable iff φ_2 is satisfiable.

Note: Equisatisfiability is weaker than equivalence but useful if all we want we want to do is determine satisfiability.

rubenm@cs.cmu.edu

Basic Constraints

Solver Input

Representing Integers

Cardinality Constraints

Lazy Encodings

rubenm@cs.cmu.edu

Solver Input: DIMACS format

```
c famous problem (in CNF)
p cnf 6 9
1 4 0
2 5 0
3 6 0
-1 -2 0
-1 -3 0
-2 -3 0
-4 -5 0
-4 -6 0
-5 -6 0
```

rubenm@cs.cmu.edu

Solver Input: DIMACS format

```
c pigeon hole problem
p cnf 6 9
1 4 0         #   pigeon[1]@hole[1] ∨  pigeon[1]@hole[2]
2 5 0         #   pigeon[2]@hole[1] ∨  pigeon[2]@hole[2]
3 6 0         #   pigeon[3]@hole[1] ∨  pigeon[3]@hole[2]
-1 -2 0       # ¬pigeon[1]@hole[1] ∨ ¬pigeon[2]@hole[1]
-1 -3 0       # ¬pigeon[1]@hole[1] ∨ ¬pigeon[3]@hole[1]
-2 -3 0       # ¬pigeon[2]@hole[1] ∨ ¬pigeon[3]@hole[1]
-4 -5 0       # ¬pigeon[1]@hole[2] ∨ ¬pigeon[2]@hole[2]
-4 -6 0       # ¬pigeon[1]@hole[2] ∨ ¬pigeon[3]@hole[2]
-5 -6 0       # ¬pigeon[2]@hole[2] ∨ ¬pigeon[3]@hole[2]
```

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (1)

- SAT solvers take as input a formula in CNF
- What is the complexity of transformation any formula φ in CNF?

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (1)

- SAT solvers take as input a formula in CNF
- What is the complexity of transformation any formula φ in CNF?

In some cases, converting a formula to CNF can have an exponential explosion on the size of the formula.

If we convert $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$ using De Morgan's laws and distributive law to CNF:

$$(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \ldots \vee x_n) \wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$$

- How can we avoid the exponential blowup? In this case, the equivalent formula would have 2^n clauses!

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (1)

- SAT solvers take as input a formula in CNF
- What is the complexity of transformation any formula φ in CNF?

- Tseitin's transformation converts a formula φ into an equisatisfiable CNF formula that is linear in the size of φ!
- Key idea: introduce auxiliary variables to represent the output of subformulas, and constrain those variables using CNF clauses!

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \land r)}$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \land r)}$$

1. Introduce a fresh variable for every non-atomic subformula

$$t_1 \quad \leftrightarrow \quad p \rightarrow t_2$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \land r)}$$

1. Introduce a fresh variable for every non-atomic subformula

$$\begin{array}{rcl} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \land r \\ \hline \end{array}$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF

$$\begin{array}{rcl} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \wedge r \\ \hline \end{array}$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF

$$\begin{array}{ccc} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \wedge r \end{array}$$

$$(t_1 \vee p) \wedge (t_1 \vee \overline{t}_2) \wedge (\overline{t}_1 \vee \overline{p} \vee t_2)$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF

$$\begin{array}{lcl} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \wedge r \end{array}$$

$$(t_1 \vee p) \wedge (t_1 \vee \overline{t}_2) \wedge (\overline{t}_1 \vee \overline{p} \vee t_2)$$
$$(\overline{t}_2 \vee q) \wedge (\overline{t}_2 \vee r) \wedge (t_2 \vee \overline{q} \vee \overline{r})$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF
3. Assert the conjunction of t_1 and the CNF-converted equivalences

$$\begin{array}{rcl} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \wedge r \end{array}$$

$$(t_1 \vee p) \wedge (t_1 \vee \overline{t}_2) \wedge (\overline{t}_1 \vee \overline{p} \vee t_2)$$
$$(\overline{t}_2 \vee q) \wedge (\overline{t}_2 \vee r) \wedge (t_2 \vee \overline{q} \vee \overline{r})$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF
3. Assert the conjunction of t_1 and the CNF-converted equivalences

$$\begin{array}{ccc} t_1 & \leftrightarrow & p \rightarrow t_2 \\ t_2 & \leftrightarrow & q \wedge r \end{array}$$

$$F_1 : (t_1 \vee p) \wedge (t_1 \vee \overline{t}_2) \wedge (\overline{t}_1 \vee \overline{p} \vee t_2)$$
$$F_2 : (\overline{t}_2 \vee q) \wedge (\overline{t}_2 \vee r) \wedge (t_2 \vee \overline{q} \vee \overline{r})$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (2)

$$\boxed{p \rightarrow (q \wedge r)}$$

1. Introduce a fresh variable for every non-atomic subformula
2. Convert each equivalence into CNF
3. Assert the conjunction of t_1 and the CNF-converted equivalences

$$\begin{aligned} t_1 &\leftrightarrow p \rightarrow t_2 \\ t_2 &\leftrightarrow q \wedge r \end{aligned}$$

$$F_1 : (t_1 \vee p) \wedge (t_1 \vee \overline{t}_2) \wedge (\overline{t}_1 \vee \overline{p} \vee t_2)$$

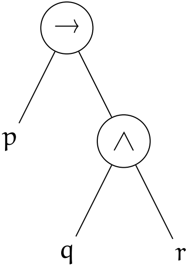
$$F_2 : (\overline{t}_2 \vee q) \wedge (\overline{t}_2 \vee r) \wedge (t_2 \vee \overline{q} \vee \overline{r})$$

$$t_1 \wedge F_1 \wedge F_2$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (3)

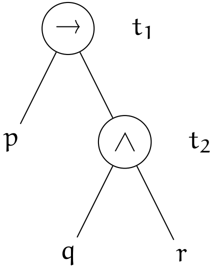
Tree representation of the Tseitin Transformation:



rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (3)

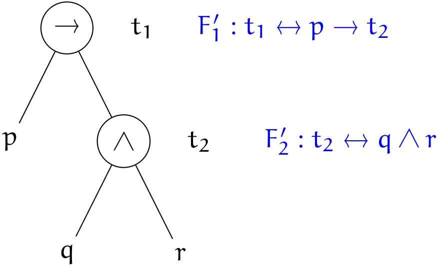
Tree representation of the Tseitin Transformation:



rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (3)

Tree representation of the Tseitin Transformation:

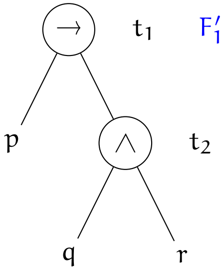


$F'_1 : t_1 \leftrightarrow p \rightarrow t_2$

$F'_2 : t_2 \leftrightarrow q \wedge r$

Solver Input: Tseitin Transformation (3)

Tree representation of the Tseitin Transformation:



$F'_1 : t_1 \leftrightarrow p \rightarrow t_2$

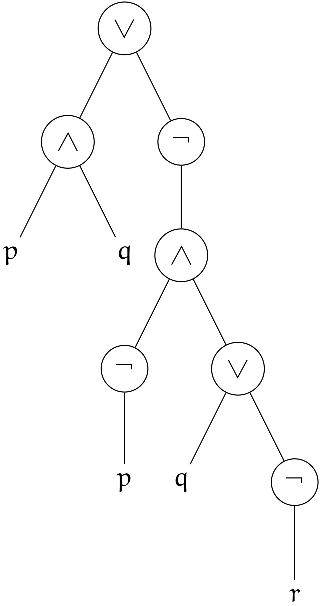
$F'_2 : t_2 \leftrightarrow q \wedge r$

$$\boxed{p \rightarrow (q \wedge r) \equiv t_1 \wedge \text{CNF}(F'_1) \wedge \text{CNF}(F'_2)}$$

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (4)

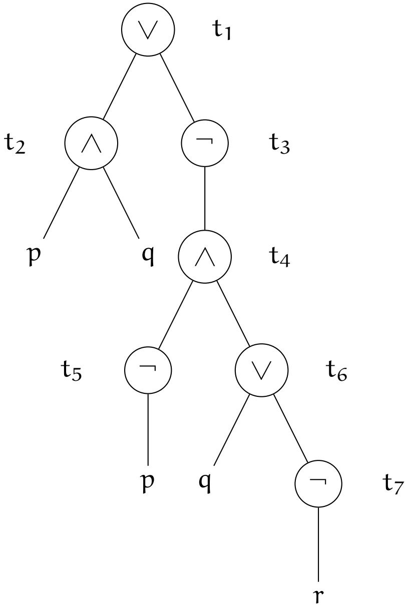
$F : (p \land q) \lor \neg(\neg p \land (q \lor \neg r))$



rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (4)

$F : (p \land q) \lor \neg(\neg p \land (q \lor \neg r))$



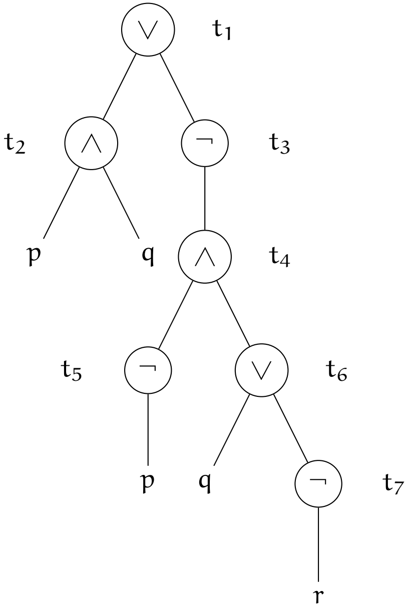
rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (4)

$F : (p \land q) \lor \neg(\neg p \land (q \lor \neg r))$

$F'_1 : t_1 \leftrightarrow t_2 \lor t_3$
$F'_2 : t_2 \leftrightarrow p \land q$
$F'_3 : t_3 \leftrightarrow \neg t_4$
$F'_4 : t_4 \leftrightarrow t_5 \land t_6$
$F'_5 : t_5 \leftrightarrow \neg p$
$F'_6 : t_6 \leftrightarrow q \lor t_7$
$F'_7 : t_7 \leftrightarrow \neg r$

$F \equiv t_1 \land \text{CNF}(F'_1) \land \ldots \land \text{CNF}(F'_7)$



rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (5)

- Using automated tools to encode to CNF:
  limboole: `http://fmv.jku.at/limboole`

rubenm@cs.cmu.edu

Solver Input: Tseitin Transformation (5)

- Using automated tools to encode to CNF:
  limboole: `http://fmv.jku.at/limboole`

- Tseitin's encoding may add many redundant variables/clauses!
- Using **limboole** for the pigeon hole problem ($n = 3$) creates a formula with 40 variables and 98 clauses
- After unit propagation the formula has 12 variables and 28 clauses
- Original CNF formula only has 6 variables and 9 clauses

rubenm@cs.cmu.edu

Basic Constraints

Solver Input

Representing Integers

Cardinality Constraints

Lazy Encodings

rubenm@cs.cmu.edu

Representing Integers: Direct Encoding

- Each number i is represented by a Boolean variable: d_i
- At least one number is true: $d_0 \vee \cdots \vee d_n$
- At most one number is true: $\bigwedge_{i<j} \overline{d}_i \vee \overline{d}_j$
- Expressing in a clause that an integer has a specific value v requires one literal.
- For example, "if the number is 1, then do x", is encoded as $\overline{d}_1 \vee x$.
- Typically effective when reasoning about a small range of integers.

rubenm@cs.cmu.edu

Representing Integers: Order Encoding

Order encoding:

- Variables represent that a number is larger or equal: $o_{\geq i}$
- Requires a linear number of binary clauses: $o_{\geq i} \vee \overline{o}_{\geq i+1}$
- Expressing in a clause that an integer has a specific value v requires two literals.
- For example, "if the number is 1, then do x", is encoded as $\overline{o}_{\geq 1} \vee o_{\geq 2} \vee x$.
- Allows the solver to reason (and produce clauses) that cover multiple cases.

rubenm@cs.cmu.edu

Representing Integers: Binary Encoding

Binary encoding:

- Use $\lceil \log_2 n \rceil$ auxiliary variables b_i to represent n in binary
- All non-occurring numbers $\leq 2^{\lceil \log_2 n \rceil}$ need to be blocked. For example, if we have the numbers 0, 1, and 2, then the number 3 needs to be blocked: $(\neg b_0 \vee \neg b_1)$
- Expressing in a clause that an integer has a specific value v requires $\lceil \log_2 n \rceil$ literals.
- For example, "if the number is 1, then do x", is encoded as $\neg b_0 \vee b_1 \vee x$.
- Typically effective when reasoning about a large range of integers.

rubenm@cs.cmu.edu

Basic Constraints

Solver Input

Representing Integers

Cardinality Constraints

Lazy Encodings

rubenm@cs.cmu.edu

How to encode cardinality constraints?

Recall AtMostOne constraints:

- Direct encoding for AtMostOne constraints:
- AtMostOne: $x_1 + x_2 + x_3 + x_4 \leq 1$
- Clauses:

$$\left.\begin{array}{c} (x_1 \rightarrow \overline{x}_2) \\ (x_1 \rightarrow \overline{x}_3) \\ (x_1 \rightarrow \overline{x}_4) \\ \dots \end{array}\right\} \begin{array}{c} \overline{x}_1 \vee \overline{x}_2 \\ \overline{x}_1 \vee \overline{x}_3 \\ \overline{x}_1 \vee \overline{x}_4 \\ \dots \end{array}$$

- Complexity: $\mathcal{O}(n^2)$ clauses

rubenm@cs.cmu.edu

How to encode cardinality constraints?

AtMostK constraints:

- Naive encoding for AtMostK constraints:
- Cardinality constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
- Clauses:

$$\left.\begin{array}{c} (x_1 \wedge x_2 \rightarrow \overline{x}_3) \\ (x_1 \wedge x_2 \rightarrow \overline{x}_4) \\ (x_2 \wedge x_3 \rightarrow \overline{x}_4) \\ \ldots \end{array}\right\} \begin{array}{c} (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \\ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \\ \ldots \end{array}$$

- Complexity: $\mathcal{O}(n^k)$ clauses
- What properties should these encodings have?

rubenm@cs.cmu.edu

How to encode cardinality constraints?

AtMostK constraints:

- Naive encoding for AtMostK constraints:
- Cardinality constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
- Clauses:

$$\left.\begin{array}{c} (x_1 \wedge x_2 \rightarrow \overline{x}_3) \\ (x_1 \wedge x_2 \rightarrow \overline{x}_4) \\ (x_2 \wedge x_3 \rightarrow \overline{x}_4) \\ \ldots \end{array}\right\} \begin{array}{c} (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \\ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \\ \ldots \end{array}$$

- Complexity: $\mathcal{O}(n^k)$ clauses
- What properties should these encodings have?
  Number of variables? Number of clauses? Other?

rubenm@cs.cmu.edu

Consistency and Arc-Consistency (1)

- Let us consider an encoding of a constraint C such that there is a correspondence between assignments of the variables in C with Boolean assignments of the variables in the encoding
- The encoding is consistent if whenever M is partial assignment inconsistent wrt C (i.e., cannot be extended to a solution of C), unit propagation leads to conflict

rubenm@cs.cmu.edu

Consistency and Arc-Consistency (1)

- Let us consider an encoding of a constraint C such that there is a correspondence between assignments of the variables in C with Boolean assignments of the variables in the encoding
- The encoding is consistent if whenever M is partial assignment inconsistent wrt C (i.e., cannot be extended to a solution of C), unit propagation leads to conflict
- The encoding is arc-consistent if
  1. it is consistent, and
  2. unit propagation discards arc-inconsistent values (values that cannot be assigned)
- These are good properties for encodings: SAT solvers are very good at unit propagation!

rubenm@cs.cmu.edu

Consistency and Arc-Consistency (2)

In the case of the AtMostOne constraint $x_1 + x_2 + \ldots + x_n \leq 1$:

- Consistency $\equiv$ if there are two variables x_i assigned to *true* then unit propagation should give a conflict
- Arc-consistency $\equiv$ Consistency $+$ if there is one x_i assigned to *true* then all others x_j should be assigned to *false* by unit propagation

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (1)

Can we build an encoding that is arc-consistent and uses a polynomial number of variables/clauses for at-most-k constraints?

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (1)

Can we build an encoding that is arc-consistent and uses a polynomial number of variables/clauses for at-most-k constraints?

Yes! By adding $O(n \cdot k)$ auxiliary variables we only need $O(n \cdot k)$ clauses!

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$$x_1 + x_2 + x_3 \leq 2$$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

Note: this is easy to encode but we will use it to give intuition.
How would you encode this with a single clause?

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

Note: this is easy to encode but we will use it to give intuition.
How would you encode this with a single clause?
$\neg(x_1 \wedge x_2 \wedge x_3) \equiv (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$$x_1 + x_2 + x_3 \leq 2$$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $s_{i,j} \equiv$ At least j variables $x_1, \ldots, x_i$ are assigned 1

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $x_1 \rightarrow$???

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $x_1 \rightarrow s_{1,1}$
- $x_2 \rightarrow s_{2,1}$
- $x_3 \rightarrow s_{3,1}$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $s_{1,1} \rightarrow$???

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$$x_1 + x_2 + x_3 \leq 2$$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $s_{1,1} \rightarrow s_{2,1}$
- $s_{2,1} \rightarrow s_{3,1}$
- $s_{2,2} \rightarrow s_{3,2}$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$$x_1 + x_2 + x_3 \leq 2$$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $(x_2 \wedge s_{1,1}) \rightarrow$???

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$$x_1 + x_2 + x_3 \leq 2$$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- $(x_2 \wedge s_{1,1}) \rightarrow s_{2,2}$
- $(x_3 \wedge s_{2,1}) \rightarrow s_{3,2}$
- $(x_3 \wedge s_{2,2}) \rightarrow s_{3,3}$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- What are we missing?
- We need to enforce that at most two x_i are assigned to 1. How can we do this?

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

x_1	x_2	x_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
—	$s_{2,2}$	$s_{3,2}$
—	—	$s_{3,3}$

- What are we missing?
- We need to enforce that at most two x_i are assigned to 1. How can we do this?
- $\overline{s}_{3,3}$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

p cnf 9 10
-1 4 0 # $\overline{x}_1 \vee s_{1,1}$
-2 5 0 # $\overline{x}_2 \vee s_{2,1}$
-3 7 0 # $\overline{x}_3 \vee s_{3,1}$
-4 5 0 # $\overline{s}_{1,2} \vee s_{2,1}$
-5 7 0 # $\overline{s}_{2,1} \vee s_{3,1}$
-6 8 0 # $\overline{s}_{2,2} \vee s_{3,2}$
-2 -4 6 0 # $\overline{x}_2 \vee \overline{s}_{1,1} \vee s_{2,2}$
-3 -5 8 0 # $\overline{x}_3 \vee \overline{s}_{2,1} \vee s_{3,2}$
-3 -6 9 0 # $\overline{x}_3 \vee \overline{s}_{2,2} \vee s_{3,3}$
-9 0 # $\overline{s}_{3,3}$

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

p cnf 9 10

-1 4 0 $\# \ \overline{x}_1 \vee s_{1,1}$

-2 5 0 $\# \ \overline{x}_2 \vee s_{2,1}$

-3 7 0 $\# \ \overline{x}_3 \vee s_{3,1}$

-4 5 0 $\# \ \overline{s}_{1,2} \vee s_{2,1}$

-5 7 0 $\# \ \overline{s}_{2,1} \vee s_{3,1}$

-6 8 0 $\# \ \overline{s}_{2,2} \vee s_{3,2}$

-2 -4 6 0 $\# \ \overline{x}_2 \vee \overline{s}_{1,1} \vee s_{2,2}$

-3 -5 8 0 $\# \ \overline{x}_3 \vee \overline{s}_{2,1} \vee s_{3,2}$

-3 -6 9 0 $\# \ \overline{x}_3 \vee \overline{s}_{2,2} \vee s_{3,3}$

-9 0 $\# \ \overline{s}_{3,3}$

If $x_1 = 1$ and $x_2 = 1$ then by unit propagation we have $x_3 = 0$.

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

p cnf 9 10

-1 4 0	# $\overline{x}_1 \vee s_{1,1}$
-2 5 0	# $\overline{x}_2 \vee s_{2,1}$
-3 7 0	# $\overline{x}_3 \vee s_{3,1}$
-4 5 0	# $\overline{s}_{1,2} \vee s_{2,1}$
-5 7 0	# $\overline{s}_{2,1} \vee s_{3,1}$
-6 8 0	# $\overline{s}_{2,2} \vee s_{3,2}$
-2 -4 6 0	# $\overline{x}_2 \vee \overline{s}_{1,1} \vee s_{2,2}$
-3 -5 8 0	# $\overline{x}_3 \vee \overline{s}_{2,1} \vee s_{3,2}$
-3 -6 9 0	# $\overline{x}_3 \vee \overline{s}_{2,2} \vee s_{3,3}$
-9 0	# $\overline{s}_{3,3}$

If $x_1 = 1$ and $x_2 = 2$ then by unit propagation we have $x_3 = 0$.

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (2)

$x_1 + x_2 + x_3 \leq 2$

`p cnf 9 10`

`-1 4 0` $\# \ \overline{x}_1 \vee s_{1,1}$

`-2 5 0` $\# \ \overline{x}_2 \vee s_{2,1}$

`-3 7 0` $\# \ \overline{x}_3 \vee s_{3,1}$

`-4 5 0` $\# \ \overline{s}_{1,2} \vee s_{2,1}$

`-5 7 0` $\# \ \overline{s}_{2,1} \vee s_{3,1}$

`-6 8 0` $\# \ \overline{s}_{2,2} \vee s_{3,2}$

`-2 -4 6 0` $\# \ \overline{x}_2 \vee \overline{s}_{1,1} \vee s_{2,2}$

`-3 -5 8 0` $\# \ \overline{x}_3 \vee \overline{s}_{2,1} \vee s_{3,2}$

`-3 -6 9 0` $\# \ \overline{x}_3 \vee \overline{s}_{2,2} \vee s_{3,3}$

`-9 0` $\# \ \overline{s}_{3,3}$

If $x_1 = 1$ and $x_2 = 2$ then by unit propagation we have $x_3 = 0$.

rubenm@cs.cmu.edu

Cardinality Constraints: Sinz encoding (3)

Encoding for the general case $x_1 + \ldots + x_n \le k$:

$$\begin{array}{ll}
(\overline{x}_1 \vee s_{1,1}) & \\
(\overline{s}_{1,j}) & \text{for } 1 < j \le k \\
\left.\begin{array}{l}(\overline{x}_i \vee s_{i,1}) \\ (\overline{s}_{i-1,1} \vee s_{i,1}) \\ (\overline{s}_i \vee \overline{s}_{i-1,k})\end{array}\right\} & \text{for } 1 < i < n \\
\left.\begin{array}{l}(\overline{x}_i \vee \overline{s}_{i-1,j-1} \vee s_{i,j}) \\ (\overline{s}_{i-1,j} \vee s_{i,j})\end{array}\right\} & \text{for } 1 < i < n \text{ and } 1 < j \le k \\
(\overline{x}_n \vee \overline{s}_{n-1,k}) &
\end{array}$$

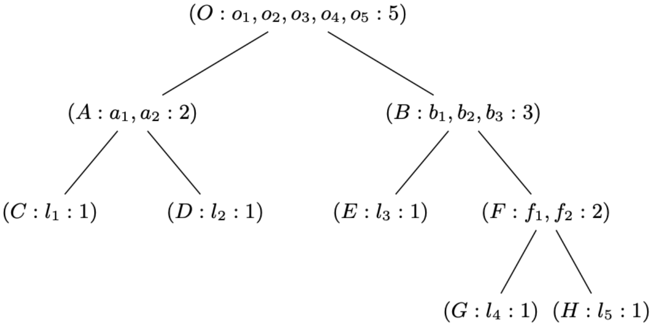
More details in paper: “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”, CP2005

- This version considers extra auxiliary variables that can be removed (e.g., sum at x_1 is never greater than 1)

Cardinality Constraints: Totalizer encoding (1)

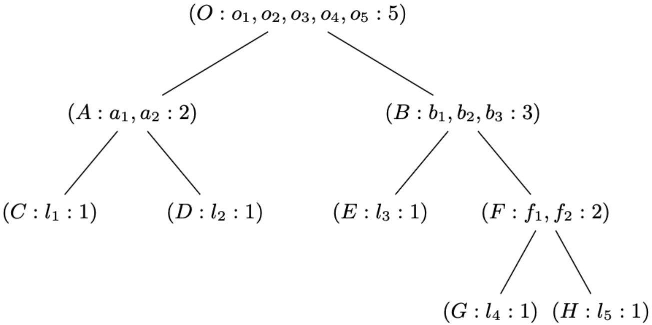
What is another example of an at-most-k encoding for $l_1 + \ldots l_5 \leq k$?

Totalizer encoding is based on a tree structure and also only needs $O(n \cdot k)$ clauses/variables.



rubenm@cs.cmu.edu

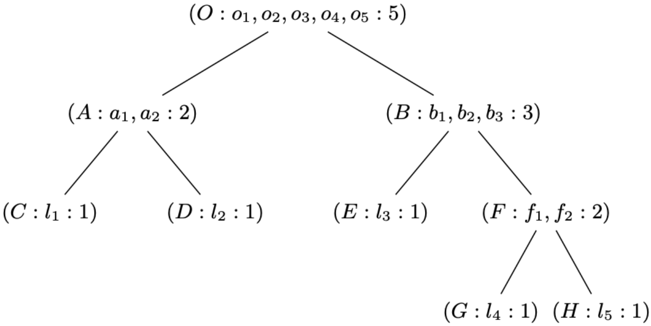
Cardinality Constraints: Totalizer encoding (2)



- Use auxiliary variables to count the sum of the subtree:
  - $f_1 \equiv l_4 + l_5 = 1$
  - $f_2 \equiv l_4 + l_5 = 2$
- Note that only f_1 or f_2 will be assigned to 1.

rubenm@cs.cmu.edu

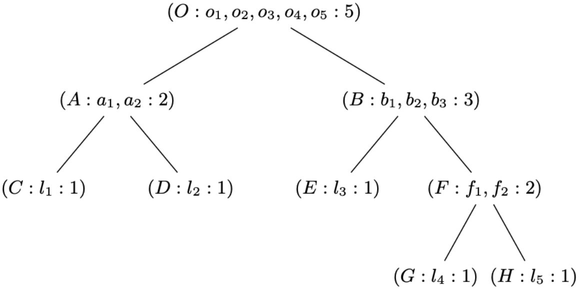
Cardinality Constraints: Totalizer encoding (3)



- Use auxiliary variables to count the sum of the subtree:
  - $b_1 \equiv l_3 + f_1 + 2 \times f_2 = 1$
  - $b_2 \equiv l_3 + f_1 + 2 \times f_2 = 2$
  - $b_3 \equiv l_3 + f_1 + 2 \times f_2 = 3$

rubenm@cs.cmu.edu

Cardinality Constraints: Totalizer encoding (4)



Any parent node P, counting up to n_P, has two children L and R counting up to n_L and n_R respectively s.t. $n_L + n_R = n_P$.

rubenm@cs.cmu.edu

Further reading

More details about cardinality encodings can be found in:

- Sinz's encoding:
  Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. CP 2005. pp. 827-831
  `http://www.carstensinz.de/papers/CP-2005.pdf`
- Totalizer encoding:
  Olivier Bailleux, Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. CP 2003. pp. 108-122
  `https://tinyurl.com/y6ph76au`
- Modulo Totalizer encoding:
  Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, Hiroshi Fujita. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. ICTAI 2013. pp. 9-17 `https://ieeexplore.ieee.org/document/6735224`
- Cardinality networks:
  Roberto Asin, Robert Nieuwenhuis, Albert Oliveras, Enric Rodriguez-Carbonell. Cardinality Networks and Their Applications. SAT 2009. pp. 167-180 `https://tinyurl.com/yxwrxzxo`

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(n \log^2 k)$ variables and clauses

We can also generalize to linear constraints with integer coefficients called pseudo-Boolean constraints:
$a_1x_1 + \ldots + a_nx_n \leq k$

rubenm@cs.cmu.edu

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(n \log^2 k)$ variables and clauses

We can also generalize to linear constraints with integer coefficients called pseudo-Boolean constraints:
$a_1 x_1 + \ldots + a_n x_n \leq k$

Question: Can we generalize Sinz's encoding to pseudo-Boolean constraints?

rubenm@cs.cmu.edu

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(n \log^2 k)$ variables and clauses

We can also generalize to linear constraints with integer coefficients called pseudo-Boolean constraints:
$a_1x_1 + \ldots + a_nx_n \leq k$

Question: Can we generalize Sinz's encoding to pseudo-Boolean constraints? Yes! We just need to consider the coefficient when writing the sum constraints.

rubenm@cs.cmu.edu

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(n \log^2 k)$ variables and clauses

We can also generalize to linear constraints with integer coefficients called pseudo-Boolean constraints:
$a_1x_1 + \ldots + a_nx_n \leq k$

Question: Can we generalize Sinz's encoding to pseudo-Boolean constraints? Yes! We just need to consider the coefficient when writing the sum constraints.

More efficient encodings: Binary merger encoding only requires $O(n^2 \log^2(n) \log(w_{max}))$ clauses and maintains arc-consistency!

rubenm@cs.cmu.edu

Basic Constraints

Solver Input

Representing Integers

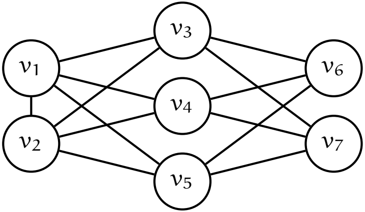
Cardinality Constraints

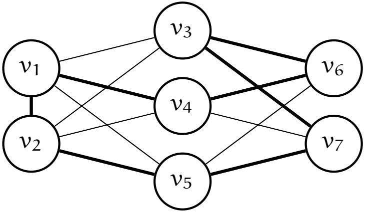
Lazy Encodings

rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?

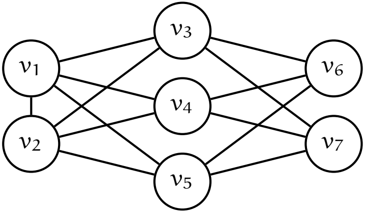


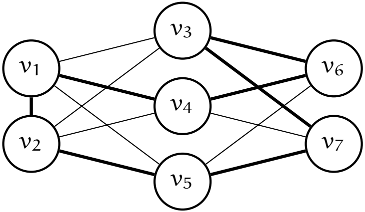


rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?





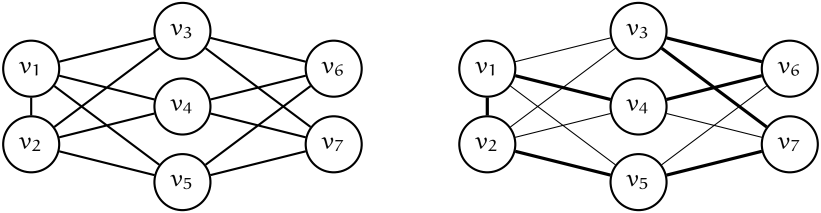
How do we encode this problem into SAT?

- Create Boolean variables and give them meaning
- Let x_{ij} be a Boolean variable for each edge between v_i, v_j:
    - $x_{ij} = 1$ if this edge is used in the solution cycle
    - $x_{ij} = 0$ if this edge is **not** used in the solution cycle

rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?



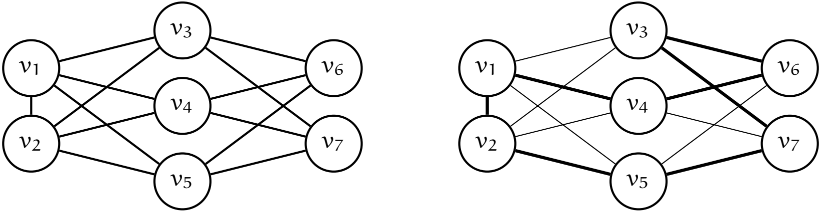
How do we encode this problem into SAT?

- Use the Boolean variables to encode the problem
- Exactly two edges per vertex
- Exactly one cycle

rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?

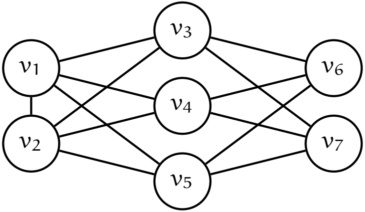


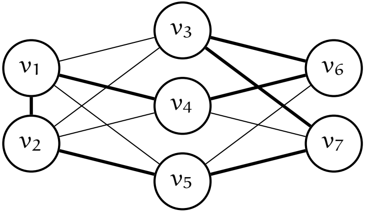
Exactly two edges per vertex:

- $\sum_{(i,j)\in E} x_{i,j} = 2$
- Example: $x_{v_1,v_2} + x_{v_1,v_3} + x_{v_1,v_4} + x_{v_1,v_5} = 2$

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?





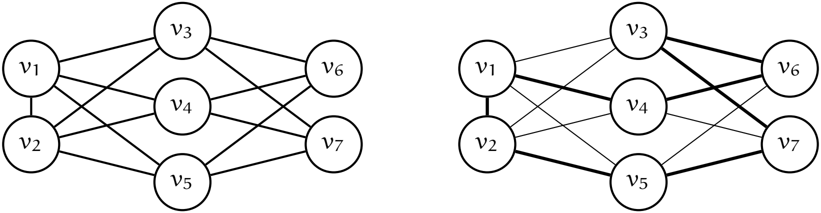
Exactly one cycle:

- How to encode?

rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?



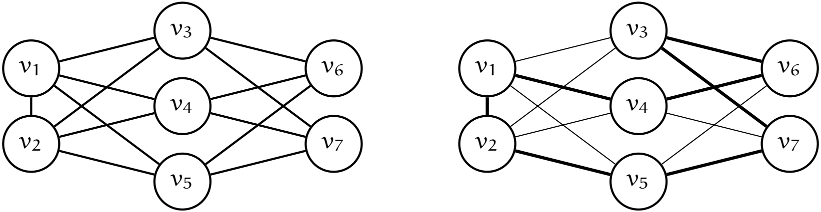
Exactly one cycle:

- $S \subset V, \quad 2 \leq |S| \leq n - 2$
- $\sum_{i,j \in S} x_{i,j} \leq |S| - 1$ (the path must leave S → no cycle)
- Example: $S = \{v_1, v_2, v_4\} : x_{v_1,v_2} + x_{v_1,v_4} + x_{v_2,v_4} \leq 2$

rubenm@cs.cmu.edu

Hamiltonian Cycles: Two Constraints

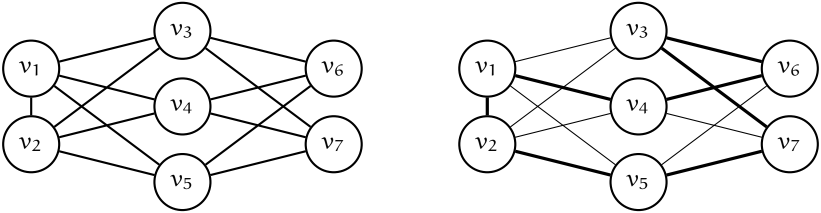
Hamiltonian Cycle Problem (HCP):
Does there exists a cycle that visits all vertices exactly once?



Exactly one cycle:

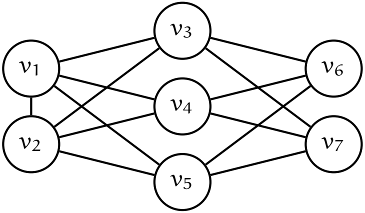
- There is an exponential number of subtours and encoding connectivity constraints with this approach is often not practical!

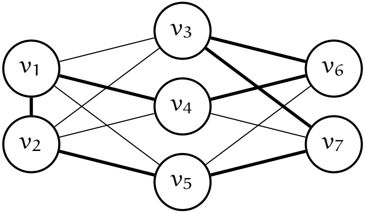
Hamiltonian Cycle Problem (2)



Can we encode this problem using fewer constraints?

Hamiltonian Cycle Problem (2)





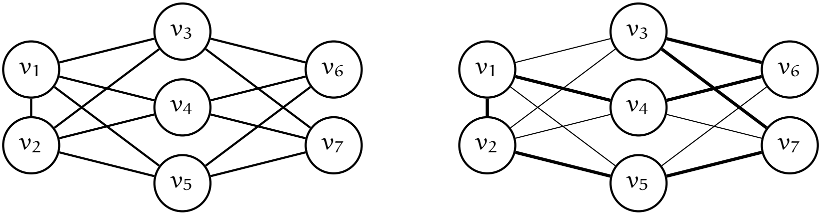
Can we encode this problem using fewer constraints?

Boolean variables:

- Consider a path that connects n vertices as a sequence of positions $p_{i,j}$ to denote vertex i occurs jth in the path.
- Example: $p_{1,v_1} = 1, p_{2,v_2} = 1, p_{3,v_5} = 1, \ldots$

rubenm@cs.cmu.edu

Hamiltonian Cycle Problem (2)

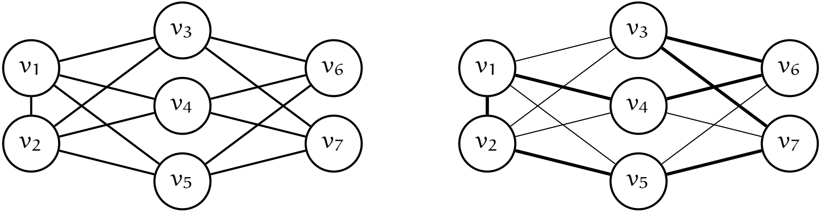


Constraints:

- Each vertex occurs exactly once in the path
- Example: $p_{1,v_1} + p_{2,v_1} + \ldots + p_{7,v_1} = 1$

rubenm@cs.cmu.edu

Hamiltonian Cycle Problem (2)

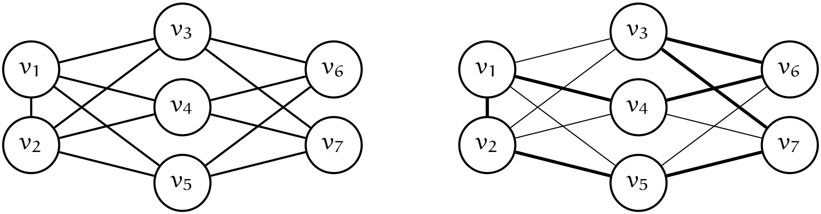


Constraints:

- Each location in the path has exactly one vertex
- Example: $p_{1,v_1} + p_{1,v_2} + \ldots + p_{1,v_7} = 1$

rubenm@cs.cmu.edu

Hamiltonian Cycle Problem (2)

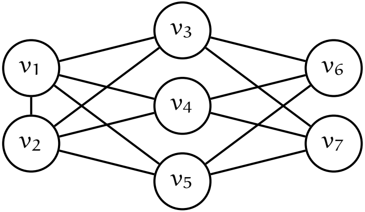


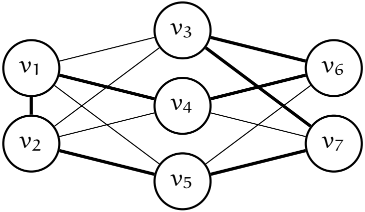
Constraints:

- Two vertices cannot be contiguous in the path if they are not adjacent in the graph
- Example: $p_{1,v_1} \rightarrow \neg p_{2,v_6}$

rubenm@cs.cmu.edu

Hamiltonian Cycle Problem (2)



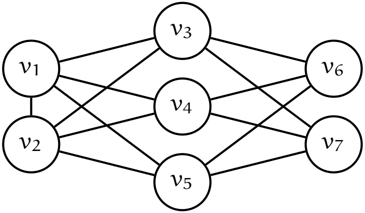


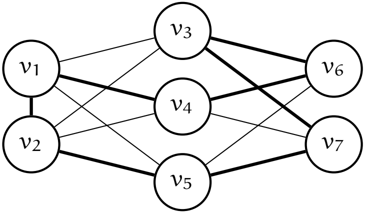
Constraints:

- Each vertex occurs exactly once in the path
- Each location in the path has exactly one vertex
- Two vertices cannot be contiguous in the path if they are not adjacent in the graph

rubenm@cs.cmu.edu

Hamiltonian Cycle Problem (2)





Constraints:

- Each vertex occurs exactly once in the path
- Each location in the path has exactly one vertex
- Two vertices cannot be contiguous in the path if they are not adjacent in the graph
- Still not good enough to handle large graphs!

rubenm@cs.cmu.edu

Hamiltonian Cycles: Incremental SAT

Lazy encoding: instead of encoding the connectivity constraint eagerly, encode it lazily!

Every time the solver returns a solution:
1. Check if it is connected. If it is then we found a solution.
2. Otherwise, add constraints to force connectivity of the current path. Ask for a new solution [Go to 1].

In practice, we can find a solution without adding add subtours! Even though we need to perform several SAT calls to find the solution, this is often faster than most encodings into one large SAT formula.

rubenm@cs.cmu.edu

Hamiltonian Cycles: Better Encodings

More compact encodings exist that can handle large graphs!
See for example:

- Linear-Feedback Shift Register Encoding:
  Michael Haythorpe and Andrew Johnson. Change ringing and Hamiltonian cycles: The search for Erin and Stedman triples. EJGTA 7, 61–75 (2019)
  `https://link.springer.com/content/pdf/10.1007/978-3-030-80223-3_15.pdf`
- Chinese Remainder Encoding:
  Marijn J. H. Heule. Chinese Remainder Encoding for Hamiltonian Cycles. SAT 2021. pp. 216-224
  `https://www.cs.cmu.edu/~mheule/publications/HamiltonianCycle.pdf`

rubenm@cs.cmu.edu

Lazy Encodings: Beyond Propositional Logic

What if our formula looks like this?

$(p \land \overline{q} \lor a = f(b - c)) \land (g(b) \neq c \lor a - c \leq 7)$

Talks about integers, functions, sets, lists, ...

We can transform it into a SAT formula

- can only find solutions within bounds
- very inefficient, so bounds are small

Better idea: combine SAT with special solvers for theories

rubenm@cs.cmu.edu

Lazy Encodings: Satisfiability Modulo Theories (SMT)

Equality and Uninterpreted Functions
EUF $= < f, g, h, \ldots, =,$ axioms of equality & congruence$>$

Linear Integer Arithmetic
LIA $= < 0, 1, \ldots, +, -, =, \leq,$ axioms of arithmetic $>$

Arrays, Strings, bitvectors, datatypes, quantifiers, ...

Theories can be combined!

rubenm@cs.cmu.edu

Lazy Encodings: SMT Solvers

- Z3 (Microsoft): https://github.com/Z3Prover/z3/wiki
- CVC4 (Stanford): http://cvc4.cs.stanford.edu/web/
- Yices (SRI): http://yices.csl.sri.com/
- Boolector (JKU Austria): https://boolector.github.io/

Next lecture we will go over SAT and SMT solvers in practice!

rubenm@cs.cmu.edu

Representations for Automated Reasoning

Ruben Martins

Carnegie
Mellon
University

http://www.cs.cmu.edu/~mheule/15816-f22/

Automated Reasoning and Satisfiability
September 7, 2022

rubenm@cs.cmu.edu