Representations for Automated Reasoning

Ruben Martins

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

Automated Reasoning and Satisfiability, September 10, 2019

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...

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...

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

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?

x	y	$XOR(x, y)$
0	0	0
0	1	1
1	0	1
1	1	0

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?

x	y	$XOR(x, y)$
0	0	0
0	1	1
1	0	1
1	1	0

$XOR(x_1, \ldots, x_n)$ is *true* when an odd number of x_i is assigned to *true*.

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:

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

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:

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

$$\begin{aligned} XOR(x, y, z) = & (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge \\ & (\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \end{aligned}$$

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:

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

$$\begin{aligned} XOR(x, y, z) = & (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge \\ & (\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \end{aligned}$$

Question: How many solutions does this formula have?

Exclusive OR (2)

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

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

into SAT?

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

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

$$\begin{aligned} XOR(x, y, z) = &(x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge \\ &(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \end{aligned}$$

Question: How many solutions does this formula have? 4

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:

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

Can we encode large XORs with less clauses?

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:

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

Can we encode large XORs with less clauses?

Make it compact: $\mathrm{XOR}\ (x_1, x_2, y) \wedge \mathrm{XOR}\ (\bar{y}, x_3, \ldots, x_n)$
Tradeoff: increase the number of variables but decreases the number of clauses!

AtMostOne (1)

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

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

into SAT?

AtMostOne (1)

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:

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

AtMostOne (1)

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:

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

Is it possible to use fewer clauses?

AtMostOne (2)

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?

AtMostOne (2)

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

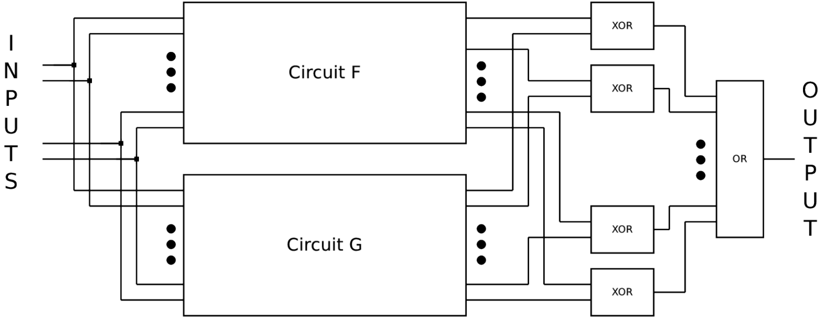
$$\textsc{AtMostOne}\ (x_1, x_2, x_3, y) \wedge \textsc{AtMostOne}\ (\bar{y}, x_4, \ldots, x_n)$$

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

AtMostOne (3)

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:



AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

φ_1 (direct encoding)	φ_2 (split encoding)
$\bar{x}_1 \lor \bar{x}_2$	$\bar{x}_1 \lor \bar{y}$
	$y \lor \bar{x}_2$

Question: Is φ_1 equivalent to φ_2?
Note: $\varphi_1 \leftrightarrow \varphi_2$ is valid if $\neg\varphi_1 \land \varphi_2$ and $\varphi_1 \land \neg\varphi_2$ are unsatisfiable.

AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

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

Is $\neg\varphi_1 \wedge \varphi_2$ unsatisfiable?
Note: $\neg\varphi_1 \equiv x_1 \wedge x_2$

AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

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

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

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

AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

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

Is $\varphi_1 \wedge \neg\varphi_2$ unsatisfiable?
Note: $\neg\varphi_2 \equiv (x_1 \vee y) \wedge (x_1 \vee x_2) \wedge (\neg y \vee x_2)$

AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

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

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

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

AtMostOne (3)

Are these two formulas that encode $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

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

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.

How to encode a problem into SAT?

```
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
```

How to encode a problem into SAT?

```
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]
```

Tseitin Transformation (1)

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

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!

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!

Tseitin Transformation (2)

$$\boxed{P \to (Q \wedge R)}$$

Tseitin Transformation (2)

$$\boxed{P \rightarrow (Q \wedge R)}$$

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

Tseitin Transformation (2)

$$\boxed{P \to (Q \wedge R)}$$

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

$$T_1 \quad \leftrightarrow \quad P \to T_2$$

Tseitin Transformation (2)

$$\boxed{P \to (Q \land R)}$$

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

$$\begin{aligned} T_1 &\leftrightarrow P \to T_2 \\ T_2 &\leftrightarrow Q \land R \end{aligned}$$

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

$$\begin{array}{rcl} T_1 & \leftrightarrow & P \rightarrow T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

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

$$\begin{array}{lcl} T_1 & \leftrightarrow & P \rightarrow T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

$$(T_1 \vee P) \wedge (T_1 \vee \neg T_2) \wedge (\neg T_1 \vee \neg P \vee T_2)$$

Tseitin Transformation (2)

$$\boxed{P \to (Q \wedge R)}$$

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

$$\begin{array}{rcl} T_1 & \leftrightarrow & P \to T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

$$(T_1 \vee P) \wedge (T_1 \vee \neg T_2) \wedge (\neg T_1 \vee \neg P \vee T_2)$$
$$(\neg T_2 \vee Q) \wedge (\neg T_2 \vee R) \wedge (T_2 \vee \neg Q \vee \neg R)$$

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

$$\begin{array}{ccc} T_1 & \leftrightarrow & P \rightarrow T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

$$(T_1 \vee P) \wedge (T_1 \vee \neg T_2) \wedge (\neg T_1 \vee \neg P \vee T_2)$$
$$(\neg T_2 \vee Q) \wedge (\neg T_2 \vee R) \wedge (T_2 \vee \neg Q \vee \neg R)$$

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

$$\begin{array}{rcl} T_1 & \leftrightarrow & P \rightarrow T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

$$F_1 : (T_1 \vee P) \wedge (T_1 \vee \neg T_2) \wedge (\neg T_1 \vee \neg P \vee T_2)$$
$$F_2 : (\neg T_2 \vee Q) \wedge (\neg T_2 \vee R) \wedge (T_2 \vee \neg Q \vee \neg R)$$

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

$$\begin{array}{rcl} T_1 & \leftrightarrow & P \rightarrow T_2 \\ T_2 & \leftrightarrow & Q \wedge R \end{array}$$

$$F_1 : (T_1 \vee P) \wedge (T_1 \vee \neg T_2) \wedge (\neg T_1 \vee \neg P \vee T_2)$$
$$F_2 : (\neg T_2 \vee Q) \wedge (\neg T_2 \vee R) \wedge (T_2 \vee \neg Q \vee \neg R)$$

$$T_1 \wedge F_1 \wedge F_2$$

Tseitin Transformation (3)

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

Tseitin Transformation (3)

- 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

Boolean representation of Integers (1)

Onehot encoding:

- Each number is represented by a boolean variable: $x_0 \ldots x_n$
- At most one number: $\bigwedge_{i \neq j} \bar{x}_i \vee \bar{x}_j$

Boolean representation of Integers (1)

Onehot encoding:

- Each number is represented by a boolean variable: $x_0 \dots x_n$
- At most one number: $\bigwedge_{i \neq j} \bar{x}_i \vee \bar{x}_j$

Unary encoding:

- Each variable x_n is true iff the number is equal to or greater than n:
  $x_2 = 1$ represents that the number is equal to or greater than 2
- x_i implies x_{i+1}: $\bigwedge_{i<j} \bar{x}_i \vee x_j$

Boolean representation of Integers (2)

Binary encoding:

- Use $\lceil log_2 n \rceil$ auxiliary variables to represent n in binary
  Consider $n = 3$:
  x_0 (number 0) corresponds to the binary representation 00
  $\bar{x}_0 \vee \bar{b}_0$, $\bar{x}_0 \vee \bar{b}_1$

Boolean representation of Integers (2)

Binary encoding:

- Use $\lceil log_2 n \rceil$ auxiliary variables to represent n in binary
  Consider $n = 3$:
  x_0 (number 0) corresponds to the binary representation 00
  $\bar{x}_0 \vee \bar{b}_0$, $\bar{x}_0 \vee \bar{b}_1$

Order encoding:

- Encode the comparison $x \leq a$ by a different Boolean variable for each integer variable x and integer value a
- Useful if you want to capture the order between integers: $\{x \leq a, \neg(y \leq a)\}$ can be used to represent the constraint among integer variables, i.e. $x \leq y$

How to encode linear constraints?

Recall AtMostOne constraints:

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

$$\left.\begin{array}{c}(x_1 \Rightarrow \neg x_2)\\(x_1 \Rightarrow \neg x_3)\\(x_1 \Rightarrow \neg x_4)\\\ldots\end{array}\right\}\begin{array}{c}\neg x_1 \vee \neg x_2\\\neg x_1 \vee \neg x_3\\\neg x_1 \vee \neg x_4\\\ldots\end{array}$$

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

How to encode linear constraints?

AtMostK constraints:

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

$$\left.\begin{array}{c}(x_1 \wedge x_2 \Rightarrow \neg x_3)\\(x_1 \wedge x_2 \Rightarrow \neg x_4)\\(x_2 \wedge x_3 \Rightarrow \neg x_4)\\\dots\end{array}\right\}\begin{array}{c}(\neg x_1 \vee \neg x_2 \vee \neg x_3)\\(\neg x_1 \vee \neg x_2 \vee \neg x_4)\\(\neg x_2 \vee \neg x_3 \vee \neg x_4)\\\dots\end{array}$$

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

How to encode linear constraints?

AtMostK constraints:

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

$$\left.\begin{array}{c}(x_1 \wedge x_2 \Rightarrow \neg x_3)\\(x_1 \wedge x_2 \Rightarrow \neg x_4)\\(x_2 \wedge x_3 \Rightarrow \neg x_4)\\\ldots\end{array}\right\} \begin{array}{c}(\neg x_1 \vee \neg x_2 \vee \neg x_3)\\(\neg x_1 \vee \neg x_2 \vee \neg x_4)\\(\neg x_2 \vee \neg x_3 \vee \neg x_4)\\\ldots\end{array}$$

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

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

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!

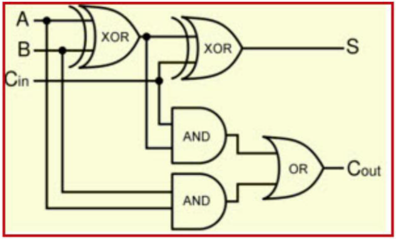
Consistency and Arc-Consistency (2)

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

Adder encoding (1)

Build an adder circuit by using bit-adders as building blocks:



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = C_{in}(A \oplus B) + AB$$

Encodings of this kind are not arc-consistent!
Consider $A + B + C_{in} \leq 0$, i.e. $\neg S \wedge \neg C_{out}$
Then unit propagation should propagate $\neg A, \neg B, \neg C_{in}$

Adder encoding (2)

```
p cnf 9 17 (2,3,5 inputs; 6,9 outputs)
2 3 -4 0
-2 -3 -4 0
2 -3 4 0
-2 3 4 0
4 5 -6 0
-4 -5 -6 0
4 -5 6 0
-4 5 6 0
2 -7 0
3 -7 0
-2 -3 7 0
4 -8 0
5 -8 0
-4 -5 8 0
-7 9 0
-8 9 0
7 8 -9 0
```

Adder encoding (2)

p cnf 9 17 (2,3,5 inputs; 6,9 outputs)
2 3 -4 0
-2 -3 -4 0
2 -3 4 0
-2 3 4 0
4 5 -6 0
-4 -5 -6 0
4 -5 6 0
-4 5 6 0
2 -7 0
3 -7 0
-2 -3 7 0
4 -8 0
5 -8 0
-4 -5 8 0
-7 9 0
-8 9 0
7 8 -9 0

Sinz encoding (1)

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

Sinz encoding (1)

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

Yes! Intuition on the whiteboard!

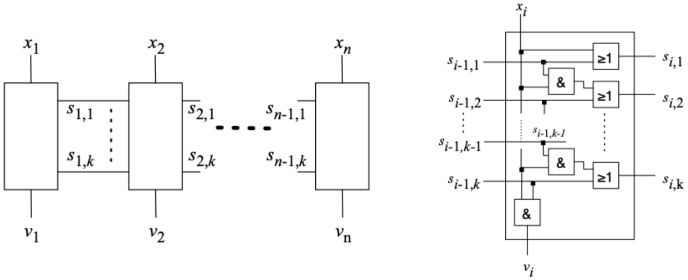
Sinz encoding (2)

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

$$
\begin{array}{ll}
(\neg x_1 \vee s_{1,1}) & \\
(\neg s_{1,j}) & \text{for } 1 < j \leq k \\
\left.\begin{array}{l}
(\neg x_i \vee s_{i,1}) \\
(\neg s_{i-1,1} \vee s_{i,1}) \\
\left.\begin{array}{l}
(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \\
(\neg s_{i-1,j} \vee s_{i,j})
\end{array}\right\} \text{ for } 1 < j \leq k \\
(\neg x_i \vee \neg s_{i-1,k})
\end{array}\right\} & \text{for } 1 < i < n \\
(\neg x_n \vee \neg s_{n-1,k}) &
\end{array}
$$

Sinz encoding (3)

Sinz's encoding can also be viewed as a circuit:

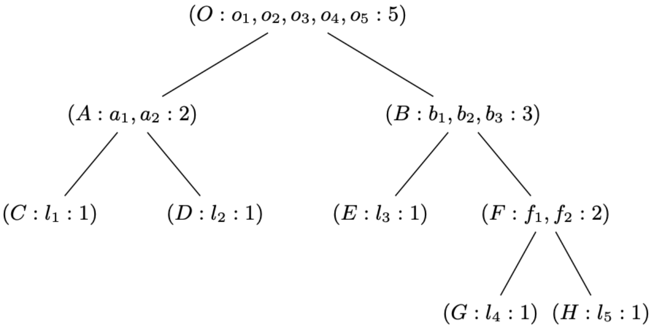


$s_{i,j}$ denotes the j-th digit of the i-th partial sum s_i in unary representation; variables v_i are overflow bits, indicating that the i-th partial sum is greater than k.

Totalizer encoding (1)

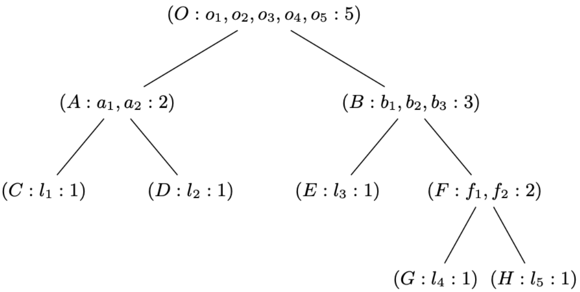
What is another example of a linear at-most-k encoding?

Totalizer encoding is based on a tree structure and also has linear complexity!



Intuition for encoding of $l_1 + \ldots l_5 \leq k$ on the whiteboard!

Totalizer encoding (2)



Any intermediate node P, counting up to n_1, has two children Q and R counting up to n_2 and n_3 respectively such that $n_2 + n_3 = n_1$. In order to ensure that the correct sum is received at P, the following formula is built for P:

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

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 $O(nlog^2 k)$ variables and clauses

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

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(nlog^2k)$ 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?

Other encodings

Many other encodings exist for cardinality constraints!
Majority are based on circuits!
Example: Sorting Networks use $O(nlog^2 k)$ 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.

Other encodings

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

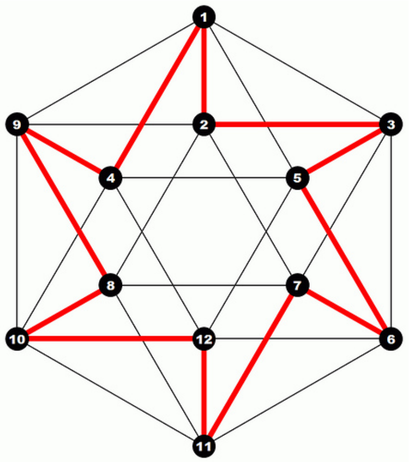
We can also generalize to linear constraints with integer coefficients called pseudo-Boolean constraints:
$a_1 x_1 + \ldots + a_n x_n \leq k$

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

Hamiltonian Cycle Problem (1)

The Hamiltonian cycle problem is the problem of finding a closed loop through a graph that visits each node exactly once!



Hamiltonian Cycle Problem (2)

Let $G = (V, E)$ be a graph where V is a set of n nodes and E is a set of edges.

Let x_{ij} be a Boolean variable for each arc $(i, j) \in E$, which is equal to 1 when (i, j) is used in a solution cycle.

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{for each node } i = 1, \ldots, n. \qquad \text{(out-degree)}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{for each node } j = 1, \ldots, n. \qquad \text{(in-degree)}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1, \qquad S \subset V,\ 2 \leq |S| \leq n - 2 \qquad \text{(connectivity)}$$

Hamiltonian Cycle Problem (3)

The out-degree and in-degree constraints force that, for each node, in-degree and out-degree are respectively exactly one in a solution cycle.

The connectivity constraint prohibits the formation of sub-cycles, i.e., cycles on proper subsets of n nodes.

Hamiltonian Cycle Problem (3)

The out-degree and in-degree constraints force that, for each node, in-degree and out-degree are respectively exactly one in a solution cycle.

The connectivity constraint prohibits the formation of sub-cycles, i.e., cycles on proper subsets of n nodes.

Transitive relations for all possible permutations of three nodes are used to represent the connectivity constraint which results in $O(n^3)$ clauses.

Lazy encodings

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 the $O(n^3)$ clauses! Even though we need to perform several SAT calls to find the solution, this is often faster than solving one large SAT formula.

Beyond Propositional Logic

What if our formula looks like this?
$(p \wedge \neg q \vee a = f(b - c)) \wedge (g(b) \neq c \vee a - c \leq 7)$
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

Satisfiability Modulo Theories

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!

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!

Representations for Automated Reasoning

Ruben Martins

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

Automated Reasoning and Satisfiability, September 10, 2019