课程：Constraint Solving

Shaowei Cai （蔡少伟）

Institute of Software, Chinese Academy of Sciences

Constraint Solving (2022. Autumn)

Constraints are everywhere…

Recognizing constraints

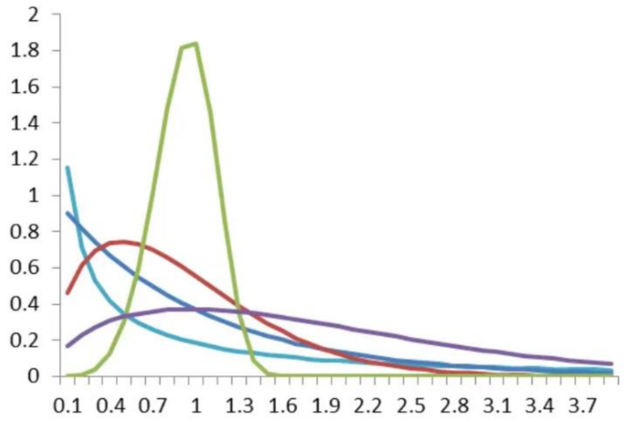
- Facts/Rules （当按钮被按下时，门就打开；一天有24小时；John是Alice的老师）
- Must do （门必须有人看着）
- Cannot violate （不能闯红灯）

Data types:

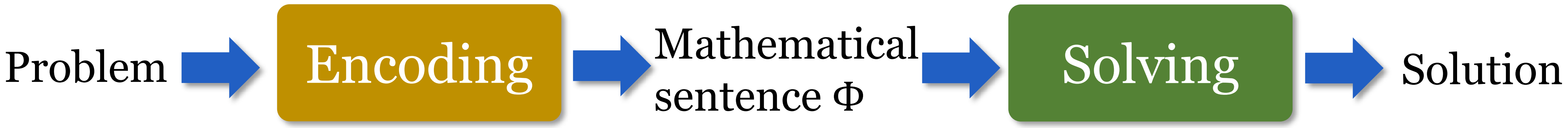
- Discrete

- Continuous

- Hybrid



Constraint Solving 约束求解



Constraints

布尔约束（SAT）：

$$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \rightarrow x_4)$$

数学规划:

$y^2 + 2xy < 100$
$x \leq 60$

谓词逻辑（SMT）：

$$((y^2 + 2xy < 100) \vee ((f(y) < 30) \rightarrow \neg(x - y < 30) \wedge (x \leq 60))$$

CSP:

$$AllDifferent(x_i, x_j) \wedge |x_i - x_j| \neq |i - j|$$

And, more:

Differential Equations 微分方程

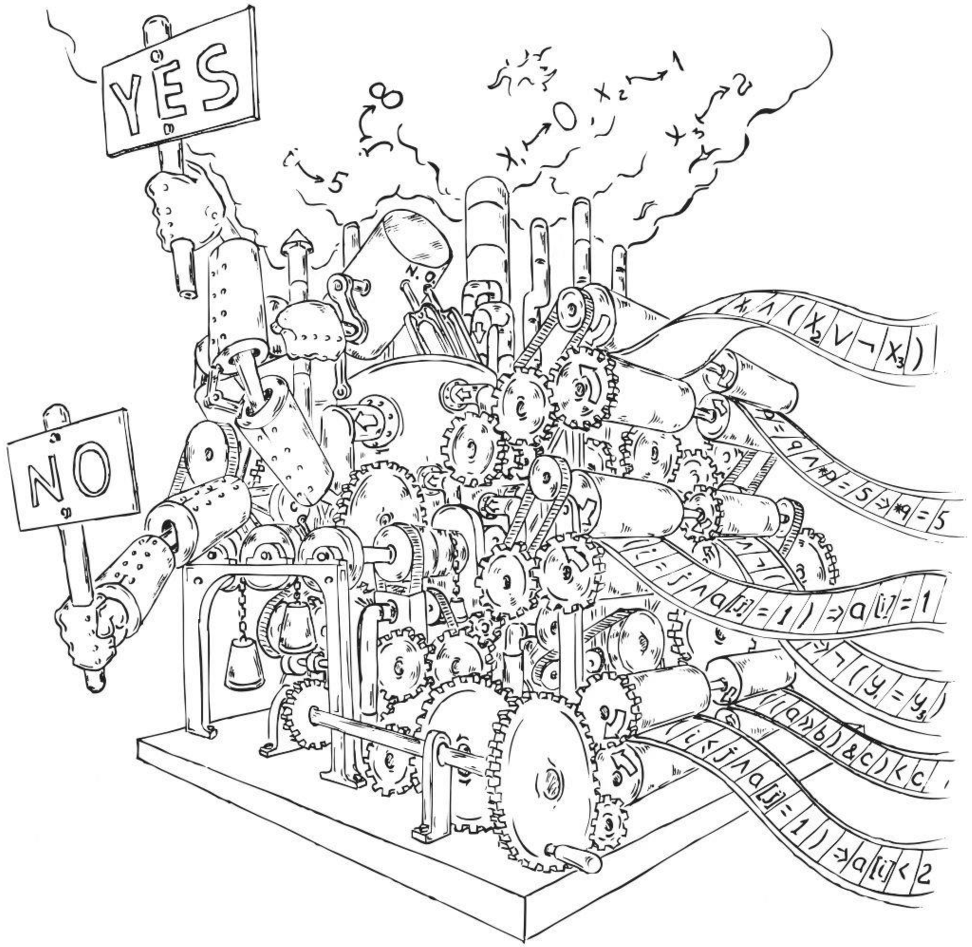
Geometrical Constraints 几何约束

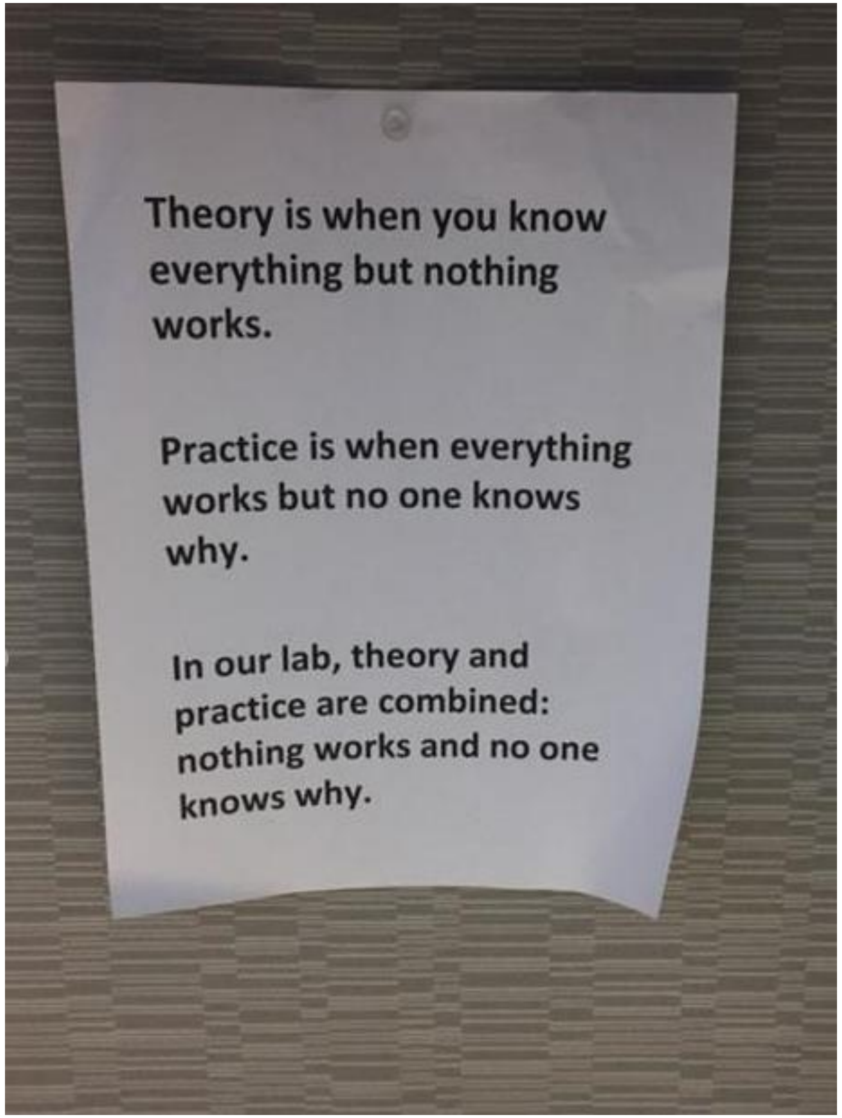
…

Model vs. Problem

Formal (Mathematic)
Generic
Solvable (most cases)

Solving


YES
NO


Theory is when you know everything but nothing works.
Practice is when everything works but no one knows why.
In our lab, theory and practice are combined: nothing works and no one knows why.

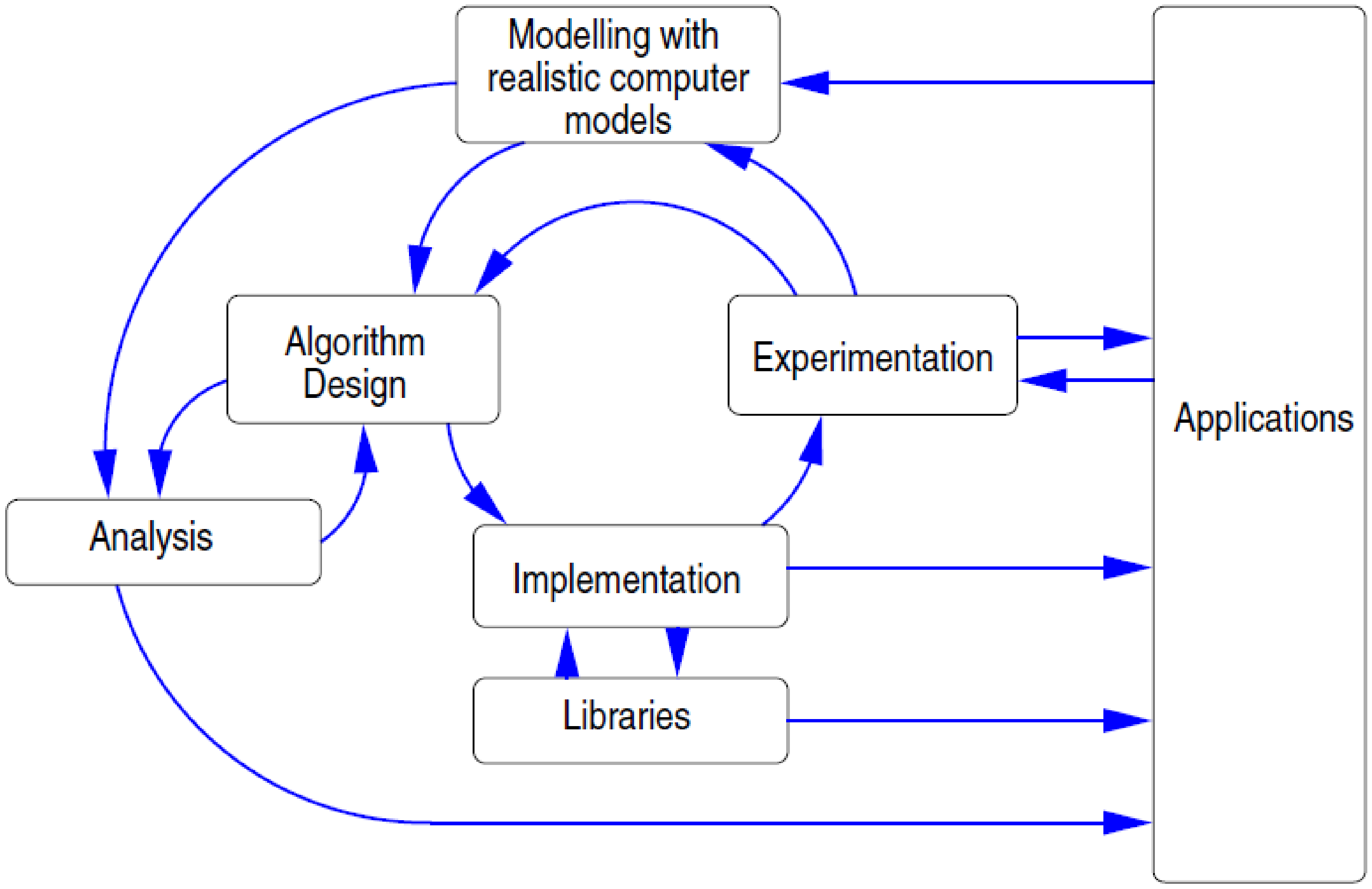
Solvers

- <u>Informally, a solver is a program that solves a constraint model.</u>

Powerful, but not always best.

Tradeoff: Generic vs. Specific

Solving as Algorithm Engineering



from 《 Algorithm engineering: Bridging the Gap between Algorithm Theory and Practice 》

How ?

- Reading
  - Books
  - Papers
  - Codes

- Discussions

- Hands in
  - Modelling
  - Implementing
  - Analyzing

- Presentation
  - Talk
  - Writing

Books

- 《The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiabiliy》
- 《Handbook of Satisfiablity》
- 《Handbook of Constraint Programming》
- 《 Decision Procedures: An Algorithmic Point of View》
- 《The Calculus of Computation: Decision Procedures with Applications to Verification》
- 《A Guide to Experimental Algorithms》
- 《Stochastic Local Search: Foundation and Application》

Constraint Models

Shaowei Cai （蔡少伟）

Institute of Software, Chinese Academy of Sciences

Constraint Solving (2022. Autumn)

Models

- SAT

SAT

- Boolean Satisfiability / Propositional Satisfiability / SAT
布尔可满足性问题/命题可满足性问题

- E.g. $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \rightarrow x_4)$

- Conjunctive Normal Form (CNF):

  e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee \neg x_4)$

  $\varphi = \{\{x_1, \neg x_2\},\{x_2, x_3\},\{x_1 \vee \neg x_4\}\}$

Every propositional formulas can be converted into CNF efficiently.

SAT

- Boolean variables: $x_1, x_2, \dots$

- A literal is a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal)

- A clause is a disjunction ($\vee$) of literals

  $x_2 \vee x_3,$

  $\neg x_1 \vee \neg x_3 \vee x_4$

- A Conjunctive Normal Form(CNF) formula is a conjunction ($\wedge$) of clauses.

SAT

Examples:

$$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

Solution: Satisfiable. Assign **T** to *p, q,* and *r.*

$$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$$

Solution: Not satisfiable. Check each possible assignment of truth values to the propositional variables and none will make the proposition true.

Hello, SAT!

- The first NP-Complete problem [Cook, 1971]
- A core problem in computer science and a basic problem in logic
- SAT solvers widely used in industry and science

> The SAT problem is evidently a killer app, because it is key to the solution of so many other problems. SAT-solving techniques are among computer science's best success stories so far, and these volumes tell that fascinating tale in the words of the leading SAT experts.
>
> *Donald Knuth*

> . . . Clearly, efficient SAT solving is a key technology for 21st century computer science. I expect this collection of papers on all theoretical and practical aspects of SAT solving will be extremely useful to both students and researchers and will lead to many further advances in the field.
>
> *Edmund Clarke*

SAT solvers

- Using SAT solvers
  - To find a certain structure
  - To prove something

Constraints
Scheduling,
Resource allocation,
Logistics,
Hardware design,
Software Engineering,
…

Reasoning
Theorem proving,
System verification,
Knowledge representation,
Decision procedure,
Agents,
…

ty

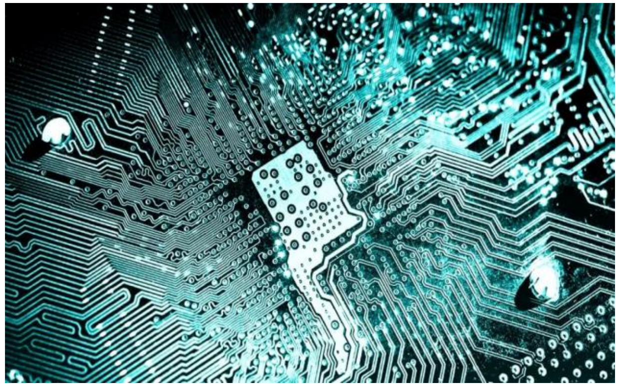
Finding a solution

Prove that
$x \geq 1, y \geq 1 \Rightarrow x + y \geq 2$

Is
$x \geq 1, y \geq 1, not(x + y \geq 2)$
UNSAT?

Checking validity

SAT revolution



EDA

original C code

```
if(!a && !b) h();
else if(!a) g();
else f();
```

⇓

```
if(!a) {
  if(!b) h();
  else g();
} else f();
```

⇒

optimized C code

```
if(a) f();
else if(b) g();
else h();
```

⇑

```
if(a) f();
else {
  if(!b) h();
  else g(); }
```

How to check that these two versions are equivalent?

Program Analysis

Planning

cryptography

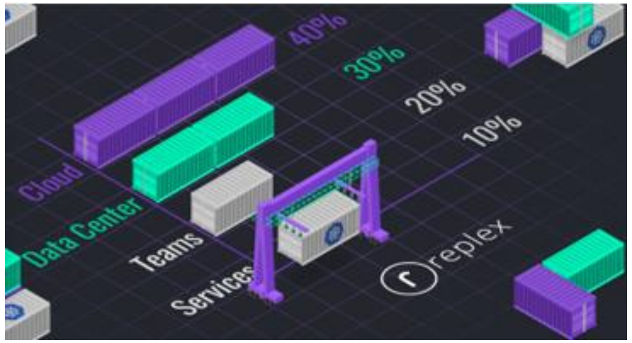
$x^m + y^m = z^m \pmod{p}$ *vdW(6) = 1132*

Schur's Theorem *Ramsey Theory*

Pythagorean Tuples Conjecture

3n+1 Conjecture?

Math



Resource Allocation

DIMACS Format of CNF

MiniSat: A open-source SAT solver widely used in industries.

<u>Input file: DIMACS format.</u>

```
c example
p cnf 4 4
1 -4 -3 0
1 4 0
-1 0
-4 3 0
```

lines starting ”c” are comments and are ignored by the SAT solver.
a line starting with ”p cnf” is the problem definition line containing the number of variables and clauses.
the rest of the lines represent clauses, literals are integers (starting with variable 1), clauses are terminated by a zero.

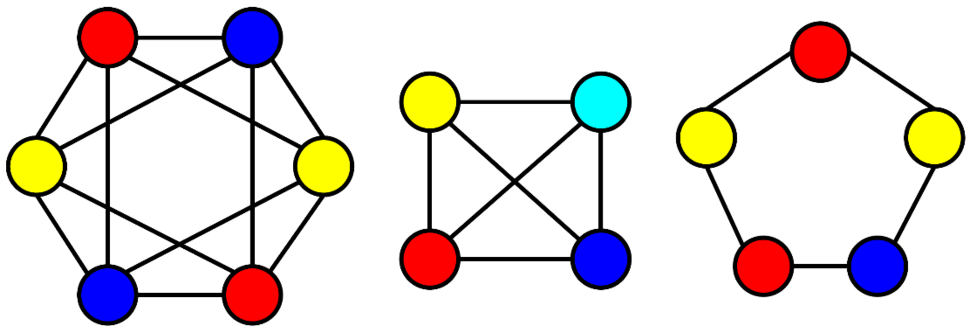
<u>Output format</u>

```
c comments, usually stastitics about the solving
s SATISFIABLE
v 1 2 -3 -4  5 -6 -7 8 9 10
v -11 12 13 -14 15 0
```

the solution line (starting with ”s”) can contain SATISFIABLE, UNSATISFIABLE and UNKNOWN.
For SATISFIABLE case, the truth values of variables are printed in lines starting with ”v”, the last value is followed by a ”0”

SAT Encoding: graph coloring problem

A coloring is an assignment of colors to vertices such that no two adjacent vertices share the same color.



The Graph Coloring Problem (GCP) is to find a coloring of a graph while minimizing the number of colors.

The decision version: given a positive number k, decide whether a graph can be colored with k colors.

SAT Encoding: graph coloring problem

- 3-coloring problem:
  - for each vertex, uses *3* variables (n vertices), $4 \times 3 = 12$ in all.

    $x_{11},\ x_{12},\ x_{13},\ x_{21},\ x_{22},\ x_{23},\ x_{31},\ x_{32},\ x_{33},\ x_{41},\ x_{42},\ x_{43}$

  - *For each edge, produces 3 negative 2-clause*

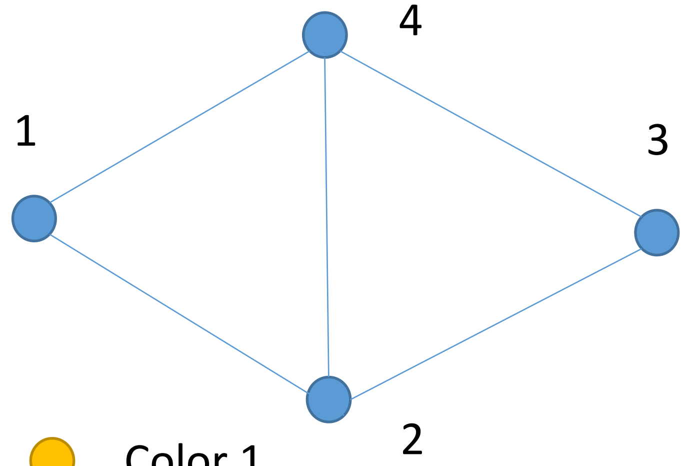
    edge1-2: $\neg x_{11} \vee \neg x_{21}$, $\neg x_{12} \vee \neg x_{22}$, $\neg x_{13} \vee \neg x_{23}$;
    edge1-4: $\neg x_{11} \vee \neg x_{41}$, $\neg x_{12} \vee \neg x_{42}$, $\neg x_{13} \vee \neg x_{43}$;
    edge2-3: $\neg x_{21} \vee \neg x_{31}$, $\neg x_{22} \vee \neg x_{32}$, $\neg x_{23} \vee \neg x_{33}$;
    edge2-4: $\neg x_{21} \vee \neg x_{41}$, $\neg x_{22} \vee \neg x_{42}$, $\neg x_{23} \vee \neg x_{43}$;
    edge3-4: $\neg x_{31} \vee \neg x_{41}$, $\neg x_{32} \vee \neg x_{42}$, $\neg x_{33} \vee \neg x_{43}$;

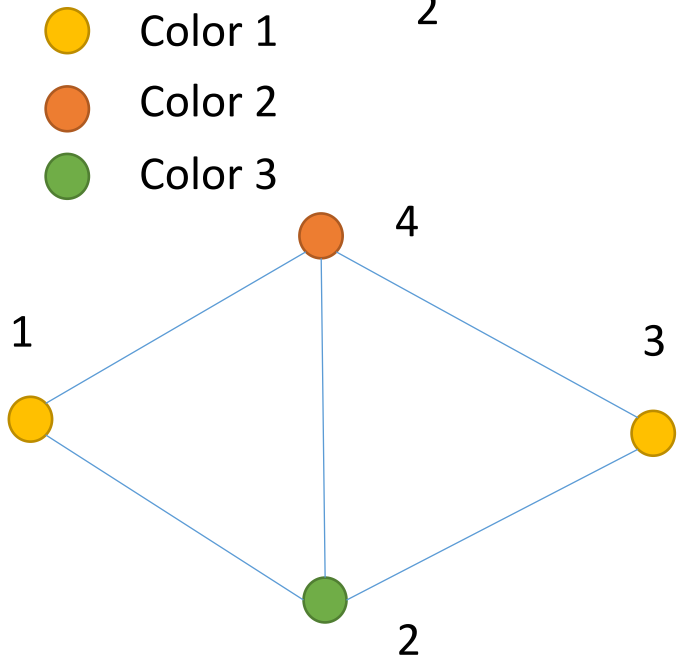
  - For each vertex, produces a positive *k*-clauses

    $x_{11} \vee x_{12} \vee x_{13}$, $x_{21} \vee x_{22} \vee x_{23}$, $x_{31} \vee x_{32} \vee x_{33}$, $x_{41} \vee x_{42} \vee x_{43}$

  - Result:

    $\mathbf{x_{11}}$, $\neg x_{12}$, $\neg x_{13}$, $\neg x_{21}$, $\mathbf{x_{22}}$, $\neg x_{23}$, $\mathbf{x_{31}}$, $\neg x_{32}$, $\neg x_{33}$, $\neg x_{41}$, $\neg x_{42}$, $\mathbf{x_{43}}$





SAT Encoding: Meeting Scheduling

Scheduling a meeting consider the following constraints

- Adam can only meet on Monday or Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

- $F = (x_1 \vee x_3) \wedge (\overline{x}_3) \wedge (\overline{x}_5) \wedge (x_4 \vee x_5) \wedge AtMostOne(x_1, x_2, x_3, x_4, x_5)$

SAT Encoding: Meeting Scheduling

$$F = (x_1 \vee x_3) \wedge (\overline{x}_3) \wedge (\overline{x}_5) \wedge (x_4 \vee x_5)$$
$$\wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_5)$$
$$\wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_5)$$
$$\wedge (\overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_3 \vee \overline{x}_5)$$
$$\wedge (\overline{x}_4 \vee \overline{x}_5)$$

- Solution: Unsatisfiable, i.e., it is impossible to schedule a meeting with these constraints

SAT Encoding: logic puzzle

- Question: at least one of them speak truth. Who speaks the truth?
  - A: B is lying.
  - B: C is lying.
  - C: A and B is lying.

- Encoding:
  - 3 variables: a, b, c present A, B, C speak truth, while ¬a, ¬b, ¬c present lying.
  - clauses:

  a ∨ b ∨ c ;   %at least one speak truth.
  ¬ a ∨ ¬ b; a ∨ b;   %a-> ¬b, ¬a -> b
  ¬ b ∨ ¬ c; b ∨ c;   %b-> ¬c, ¬b -> c
  ¬ c ∨ ¬ a; ¬c ∨ ¬b; c ∨ a ∨ b   %c->(¬a∧ ¬b), ¬c->¬ (¬a∧ ¬b)

- Result: ¬a, b, ¬c

  B speaks truth, A and C are lying

SAT Encoding: Pythagorean Tuples Conjecture

Problem Definition:

Is it possible to assign to each integer 1,2,.....n one of two colors such that if $a^2 + b^2 = c^2$ then a, b and c do not all have the same color?

- Solution : Nope
- for n=7825 it is not possible
- proof obtained by a SAT solver (2016)

How to encode this?

- for each integer i we have a Boolean variable x_i, x_i= 1 if color of i is 1, x_i=0 otherwise.
- for each a, b, c such that $a^2 + b^2 = c^2$ we have two clauses: $(x_a \lor x_b \lor x_c)$ and $(\bar{x}_a \lor \bar{x}_b \lor \bar{x}_c)$

SAT Encoding: Sudoku

- A **Sudoku puzzle** is represented by a 9×9 grid made up of nine 3×3 blocks. Some of the 81 cells of the puzzle are assigned one of the numbers 1,2, …, 9.
- Goal: assign numbers to each blank cell so that every row, column and block contains each of the nine possible numbers.

  - Let $p(i,j,n)$ denote the proposition that is true when the cell in the i-th row and the j-th column has number n.
  - There are 9×9 × 9 = 729 such propositions.
  - In the sample puzzle p(5,1,6) is true, but p(5,j,6) is false for j = 2,3,…9

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

SAT Encoding: Sudoku

- For each cell with a given value, assert $p(i,j,n)$, when the cell in row i and column j has the given value.

- Assert that every row contains every number.

$$\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i,j,n)$$

- Assert that every column contains every number.

$$\bigwedge_{j=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{9} p(i,j,n)$$

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

SAT Encoding: Sudoku

- Assert that each of the 3 x 3 blocks contain every number.

$$\bigwedge_{r=0}^{2}\bigwedge_{s=0}^{2}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{3}\bigvee_{j=1}^{3} p(3r+i,3s+j,n)$$

- Assert that no cell contains more than one number.

$$n \neq n'$$

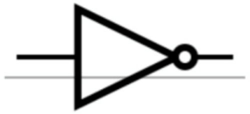
$$\neg p(i,j,n) \vee \neg p(i,j,n')$$

通过枚举二元负文字对保证独一性

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

Encoding Circuit to CNF

Tseitin Transformation

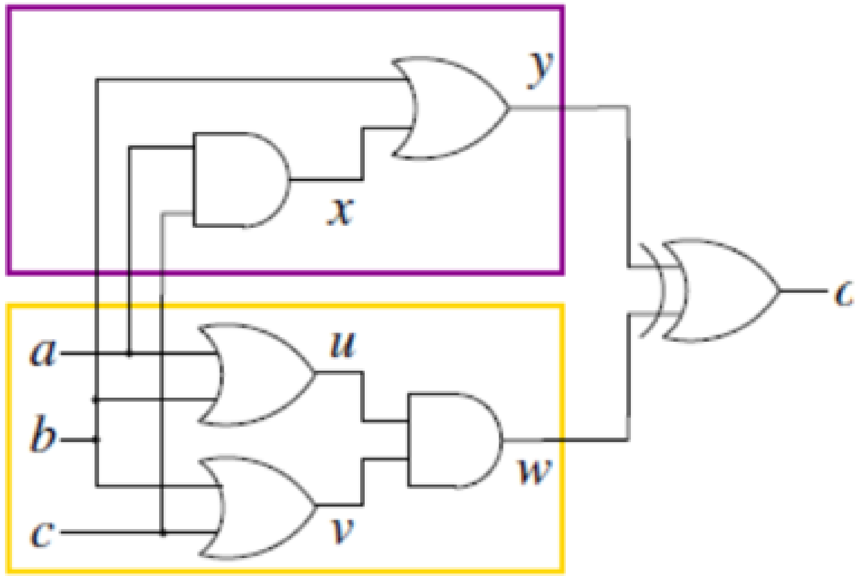
Type	Operation	CNF Sub-expression
 AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
 NAND	$C = \overline{A \cdot B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
 OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
 NOR	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
 NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
 XOR	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$

$$C = A \cdot B$$

$$C \rightarrow A \wedge B \equiv \neg C \vee (A \wedge B)$$
$$\equiv (A \vee \neg C) \wedge (B \vee \neg C)$$

$$A \wedge B \rightarrow C \equiv \neg(A \wedge B) \vee C$$
$$\equiv \neg A \vee \neg B \vee C$$

Encoding Circuit to CNF

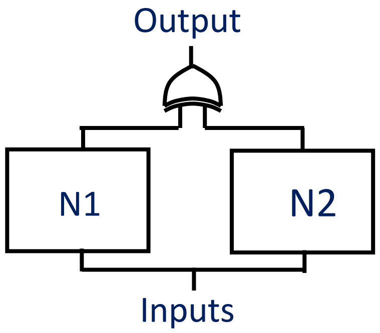


$$
\begin{aligned}
&o \wedge \\
&(x \leftrightarrow a \wedge c) \wedge \\
&(y \leftrightarrow b \vee x) \wedge \\
&(u \leftrightarrow a \vee b) \wedge \\
&(v \leftrightarrow b \vee c) \wedge \\
&(w \leftrightarrow u \wedge v) \wedge \\
&(o \leftrightarrow y \oplus w)
\end{aligned}
$$

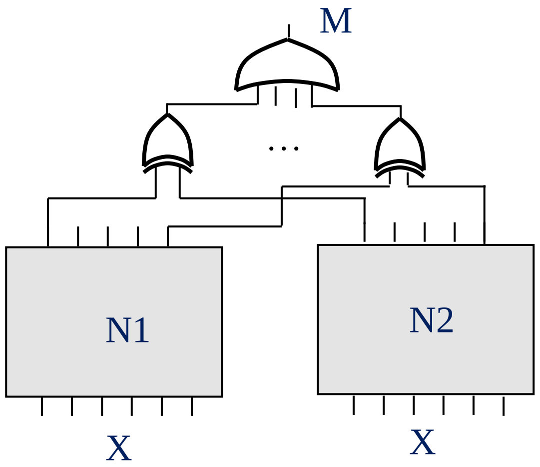
$$o \wedge (x \rightarrow a) \wedge (x \rightarrow c) \wedge (x \leftarrow a \wedge c) \wedge \ldots$$

$$\boxed{o \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge \ldots}$$

Equivalence Checking to SAT



Single output Miter

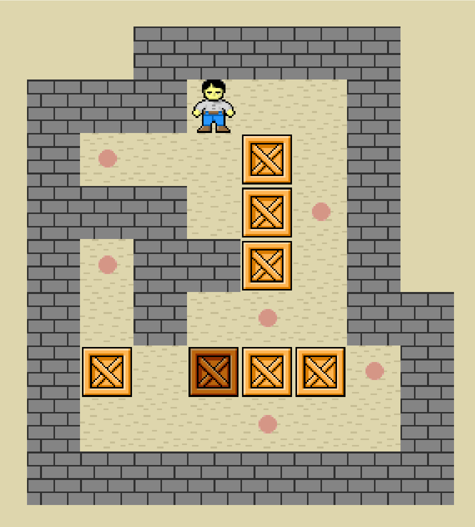


Multi-output Miter

- Build a miter circuit
- Transform Miter Circuit to CNF
- Call a SAT solver
  - If SAT, we find a counter-example
  - If UNSAT, N1=N2

Encoding Sokoban to SAT

- Variables – For each location we have variable, the domain is WORKER, BOX, EMPTY
- Initial State – assign values based on the picture
- Goal – goal position variables have value BOX
- Actions – move and push for each possible location
  - push(L1; L2; L3) = ({L1 = W; L2 = B; L3 = E}; {L1 = E; L2 = W; L3 = B}).
  - move(L1; L2) = ({L1 = W; L2 = E}; {L1 = E; L2 = W})



- We cannot encode the existence of a plan in general
- But we can encode the existence of plan up to some length

[example taken from SAT lecture by Carsten Sinz, Toma´s Balyo]

Planning Problem Definition

- A planning problem instance Π is a tuple (χ, A, S_I, S_G) where
- χ is a set of multivalued variables with finite domains.

  each variable $x \in \chi$ has a finite possible set of values $dom(x)$
- A is a set actions. Each action $a \in A$ is a tuple $\left(pre(a), eff(a)\right)$

  $pre(a)$ is a set of preconditions of action a

  $eff(a)$ is a set of effects of action a

  both are sets of equalities of the form $x = v$ where $x \in \chi$ and $v \in dom(x)$
- s_I is the initial state, it is a **full** assignment of the variables in χ
- s_G is the set of goal conditions, it is a set of equalities(same as $pre(a)$ and $eff(a)$)

Planning Problem Definition

The task

- Given a planning problem instance $\Pi = (\chi, A, S_I, S_G)$ and $k \in \mathrm{N}$ construct a CNF formula F sus that F satisfiable if and only if there is plan of length k for Π.

- We will need two kinds of variables
  - Variables to encode the actions:

    a_i^t for each $t \in \{1, \dots, k\}$ and $a_i \in A$
  - Variables to encode the states:

    $b_{x=v}^t$ for each $t \in \{1, \dots, k+1\},\ x \in \chi\ and\, v \in dom(x)$
  - In total we have $k|A| + (k+1)\sum_{x \in \chi} dom(x)$ variables

Planning Problem Definition

We will need 8 kinds of clauses

- The first state is the initial state
- The goal conditions are satisfied in the end
- Each state variable has at least one value
- Each state variable has at most one value
- If an action is applied it must be applicable
- If an action is applied its effects are applied in the next step
- State variables cannot change without an action between steps
- At most one action is used in each step

Planning Problem Definition

The first state is the initial state:

$$(b^1_{x=v})$$

$$\forall(x = v) \in s_I$$

The goal conditions are satisfied in the end:

$$(b^{n+1}_{x=v})$$

$$\forall(x = v) \in s_G$$

Planning Problem Definition

Each state variable has at least one value:

$$(b^t_{x=v_1} \vee b^t_{x=v_2} \vee \cdots b^t_{x=v_d})$$

$$\forall x \in \chi, dom(x) = \{v_1, v_1, \ldots, v_d\}, \forall t \in \{1, \ldots, k+1\}$$

Each state variable has at most one value:

$$(\neg b^t_{x=v_i} \vee \neg b^t_{x=v_j})$$

$$\forall x \in \chi, v_i \neq v_j, \{v_i, v_j\} \subseteq dom(x), \forall t \in \{1, \ldots, k+1\}$$

Planning Problem Definition

If an action is applied it must be applicable:

(preconditions是action的必要条件 $a^t \rightarrow \wedge_{\forall(x=v)\in pre(a)} b^t_{x=v}$)

$$(\neg a^t \vee b^t_{x=v})$$

$$\forall a \in A, \forall(x = v) \in pre(a), \forall t \in \{1, \ldots, k\}$$

If an action is applied its effects are applied in the next step:

(action是effects的充分条件 $a^t \rightarrow \wedge_{\forall(x=v)\in eff(a)} b^{t+1}_{x=v}$)

$$(\neg a^t \vee b^{t+1}_{x=v})$$

$$\forall a \in A, \forall(x = v) \in eff(a), \forall t \in \{1, \ldots, k\}$$

Planning Problem Definition

State variables cannot change without an action between steps

$$(\neg b_{x=v}^{t} \wedge b_{x=v}^{t+1}) \rightarrow a_{s_1}^{t} \vee \cdots \vee a_{s_j}^{t}$$
$$\Leftrightarrow (b_{x=v}^{t} \vee \neg b_{x=v}^{t+1} \vee a_{s_1}^{t} \vee \cdots \vee a_{s_j}^{t})$$

$$\forall x \in \chi, \forall v \in dom(x), support(x = v) = \left\{a_{s_1}, \ldots, a_{s_j}\right\}, \forall t \in \{1, \ldots, k\}$$

By support$(x = v) \subseteq A$ we mean the set of supporting actions of the assignment $x = v$,i.e., the set of actions that have $x = v$ as one of their effects.

对于某个位置x，如果第t步它不在状态x=v,而第t+1步它在状态x=v,
则必定是发生了某个支持x=v的action

Planning Problem Definition

At most one action is used in each step:

$$(\neg a_i^t \vee \neg a_j^t)$$

$$\forall \{a_i, a_j\} \subseteq A, a_i \neq a_j \ \forall t \in \{1, \ldots, k\}$$

Models

- SAT
- MaxSAT

MaxSAT

- When the formula is not satisfiable, we concern about satisfying as many clauses as possible -> Maximum Satisfiability.

Example: A Simple MAX-SAT Instance

$$
\begin{aligned}
F := \quad & (\neg x_1) \\
& \wedge (\neg x_2 \vee x_1) \\
& \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\
& \wedge (x_1 \vee x_2) \\
& \wedge (\neg x_4 \vee x_3) \\
& \wedge (\neg x_5 \vee x_3)
\end{aligned}
$$

- minimum number of unsatisfied clauses? 1

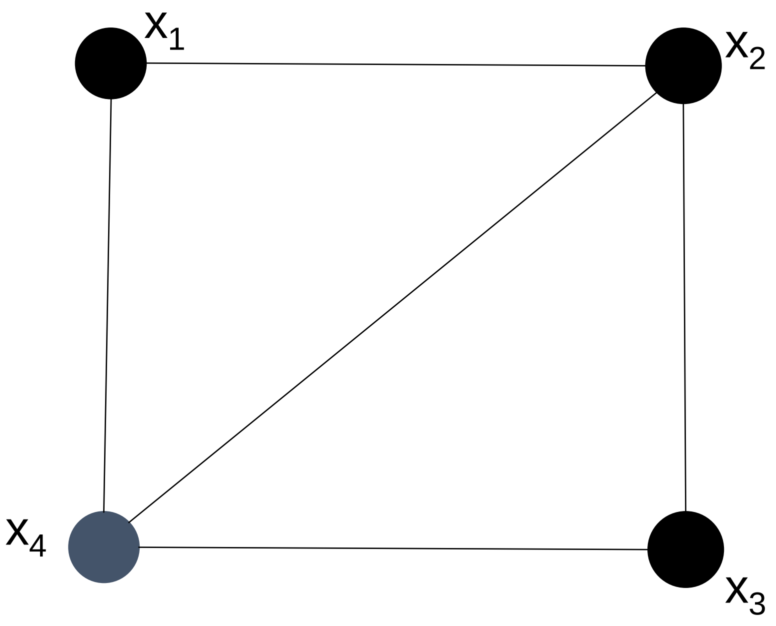
  (*e.g.*, $x_1 := x_2 := x_3 := x_4 := x_5 := \perp$)

Variants of MaxSAT

- Weighted MaxSAT
  - Each clause is associated with a weight, the goal: maximize the total weight of satisfied clauses
- Partial MaxSAT
  - hard clauses: must be satisfied
  - soft clauses: to satisfy as many as possible
  - the goal: satisfy all hard clauses and as many soft clauses as possible.
- Weighted Partial MaxSAT
  - Each soft clause is associated with a weight
  - The goal: satisfy all hard clauses and maximize the total weight of satisfied soft clauses.

Encoding MaxCut to MaxSAT

MaxCut: to maximize the numbers of edges in a graph that are “cut” by partitioning the vertices into two sets.



Graph : G = (E, V)

- non-Partial MaxSAT
- soft clauses：

$x_1 \vee x_2$
$\neg x_1 \vee \neg x_2$

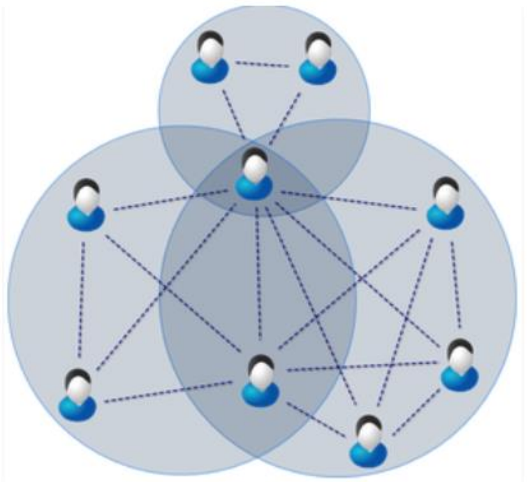
$x_1 \vee x_4$
$\neg x_1 \vee \neg x_4$

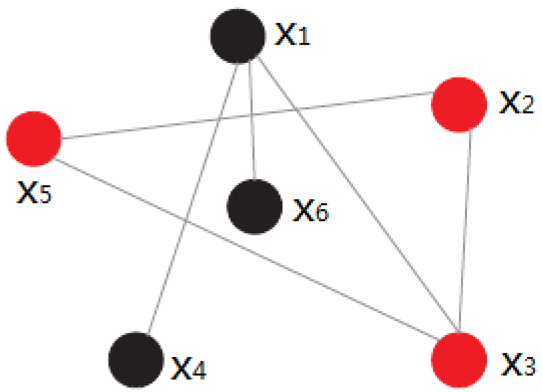
…

$x_i \vee x_j, \forall e_{ij} \in \mathrm{E}$
$\neg x_i \vee \neg x_j, \forall e_{ij} \in E$

Encoding MaxClique to MaxSAT

- MaxClique Problem





A **clique** is a vertex subset C such that every vertex in C is adjacent to any other vertices in C.

- hard clauses:

$\neg x_1 \vee \neg x_5$

$\neg x_1 \vee \neg x_2$

…

$\neg x_i \vee \neg x_j, \forall e_{ij} \notin E$

- soft clauses:

x_1

…

x_6

Encoding Set Cover to Weighted Partial MaxSAT

Set Cover Problem
U=$\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$

- S1: $\{x_1, x_2\}$, 2
- S2: $\{x_1, x_2, x_3, x_4\}$, 3
- S3: $\{x_2, x_3, x_5\}$, 2
- S4: $\{x_2, x_4, x_5\}$, 2
- S5: $\{x_3, x_4, x_5, x_6\}$,7
- S6: $\{x_4, x_5, x_6, x_7\}$,5
- S7: $\{x_6, x_7, x_8\}$,3
- S8: $\{x_7, x_8\}$,4

Hard clauses:	**Soft clauses:**
$h1: \{v_1, v_2\}$	$s1: \{\neg v_1\}, 2$
$h2: \{v_1, v_2, v_3, v_4\}$	$s2: \{\neg v_2\}, 3$
$h3: \{v_2, v_3, v_5\}$	$s3: \{\neg v_3\}, 2$
$h4: \{v_2, v_4, v_5\}$	$s4: \{\neg v_4\}, 2$
$h5: \{v_3, v_4, v_5, v_6\}$	$s5: \{\neg v_5\}, 7$
$h6: \{v_5, v_6, v_7\}$	$s6: \{\neg v_6\}, 5$
$h7: \{v_6, v_7, v_8\}$	$s7: \{\neg v_7\}, 3$
$h8: \{v_7, v_8\}$	$s8: \{\neg v_8\}, 4$

Cardinality constraints and CNF

- Cardinality constraints:
  - $l_1 + l_2 + \ldots + l_n \geq k,\ \mathrm{k} \in \mathbb{Z},\ \ \boldsymbol{l_i} \in \{\boldsymbol{x_i}, \neg \boldsymbol{x_i}\},\ \ \boldsymbol{x_i} \in \{\mathbf{0}, \mathbf{1}\}$

- A naïve encoding to CNF: Forbidding all illegal assignments
- Example: $atLeast\_2\{x_1, x_2, x_3, x_4\}$

h1: $x_1 \lor x_2 \lor x_3 \lor x_4$

h1: $\neg x_1 \lor x_2 \lor x_3 \lor x_4$

h1: $x_1 \lor \neg x_2 \lor x_3 \lor x_4$

h1: $x_1 \lor x_2 \lor \neg x_3 \lor x_4$

h1: $x_1 \lor x_2 \lor x_3 \lor \neg x_4$

Linear objective function and CNF

- Example:
- Min z = $2 * x_1 + 3 * x_2 - 4 * x_3$
- Generate soft unit clauses:

  s1 = $(2, \{\neg x_1\})$

  s2 = $(3, \{\neg x_2\})$

  s3 = $(4, \{x_3\})$

Models

- SAT
- MaxSAT
- Integer Linear Programming

Pseudo Boolean Constraints / 0-1 Integer Linear Programming

- Pseudo Boolean constraints:
  - $\mathrm{a}_1 l_1 + \mathrm{a}_2 l_2 + \ldots + \mathrm{a}_n l_n \geq k,\ \boldsymbol{a_i}, \mathrm{k} \in \mathbb{Z},\ \ \boldsymbol{l_i} \in \{\boldsymbol{x_i}, \neg \boldsymbol{x_i}\},\ \boldsymbol{x_i} \in \{\mathbf{0}, \mathbf{1}\}$

One PB constraint: $\sum_{i=1}^{i=n} a_i\, l_i \geq \mathrm{k}$

O(nlogn) additional variables and clauses of **CNF**

$O(\sum_{i=1}^{i=n} a_i)$ additional variables and clauses of **ECNF**

Linear Programming (LP)

- Linear Programming has been studied for many years and achieved great success in real word situations.
- Standard form:

$$\min z = c'x$$

$$\text{s. t.} \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

c is a weight vector; **x** represents the vector of variables

A represents a matrix; **b** is a column vector

LP

- Linear function is in the form of $a'x \infty b$, *where* ∞ *can be* =、≥、≤
- $\mathrm{a'x} = \mathrm{b}$ → *standard form*
- $\mathrm{a'x} \geq \mathrm{b}$ → $\mathrm{a'x} - x_s = \mathrm{b}$, *where* $x_s \geq 0$
- $\mathrm{a'x} \leq \mathrm{b}$ → $\mathrm{a'x} + x_s = \mathrm{b}$, *where* $x_s \geq 0$
- $x_i \leq 0$ → $y_i = -x_i$
- $x_i \in R$ → $x_i = z_1 - z_2$, where $z_1, z_2 \geq 0$
- max $c'x$ → $min\ -c'x$

LP

Example:

$$\max 100x_1 + 200x_2$$

$$\text{s. t.} \begin{cases} 30x_1 + 40x_2 \leq 500 \\ 40x_1 + 60x_2 \leq 700 \\ x_1, x_2 \geq 0 \end{cases}$$

→

$$\min -(100x_1 + 200x_2)$$

$$\text{s. t.} \begin{cases} 30x_1 + 40x_2 + c_1 = 500 \\ 40x_1 + 60x_2 + c_2 = 700 \\ x_1, x_2, c_1, c_2 \geq 0 \end{cases}$$

1. Turn to standard form
2. Apply the general LP solver

ILP and MILP

- If all the variables in LP are restricted to integers, the resulting problem is Integer Linear Programming (ILP)

- If only a part of the variables in LP is restricted to integers, the resulting problem is Mix Integer Linear Programming (MILP)

MILP Example:

Food	Cost per serving	Vitamin A	Vitamin B
Corn	\$0.18	107	72
Milk	\$0.23	500	121
Wheat Bread	\$0.05	0	65

s.t. : 1. the total intake of vitamin A is not less than 500

2. the total intake of Vitamin B is not less than 1000

Goal: minimize the total cost

Min: $0.18x_{corn} + 0.23\, x_{milk} + 0.05\, x_{bread}$

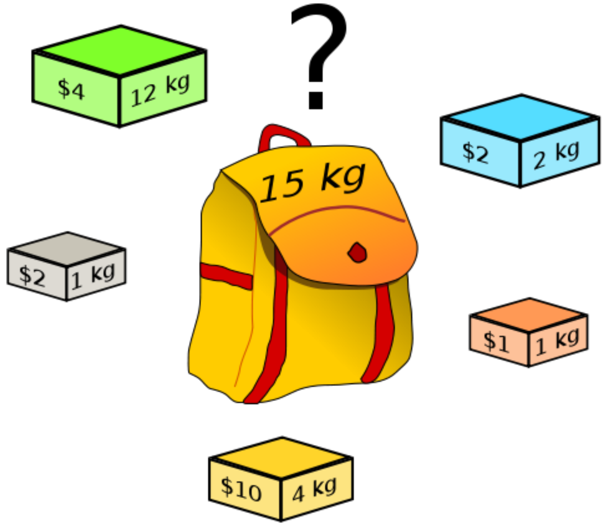
s.t. $107x_{corn} + 500\, x_{milk} >= 500$

$72x_{corn} + 121\, x_{milk} + 65\, x_{bread} >= 1000$

$x_{corn}, x_{milk}, x_{bread} \geq 0$; x_{corn} is integer

ILP:Knapsack Problem

- Given a set of items, each with a weight and a profit, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.



$$
\begin{cases}
\text{Max} \sum_{i=1}^{N} p_i x_i \\
\text{s.t} \sum_{i=1}^{N} w_i x_i \leq C; \\
x_i \in \{0,1\}; \forall i = 1, \cdots, N
\end{cases}
$$

capacity constraint of resources

N: the number of items

p_i: the profit of the i-th item

w_i: the weight of the i-th item

C: the capacity of the knapsack

x_i: binary decision variable

it equals to 1 if i-th item is selected, and 0 otherwise.

Multiple Dimensions 0-1 Knapsack Problem (MKP)

- Each item i consumes an amount $w_{ji} \geq 0$ from each dimension j.
- Each dimension has a capacity $C_j > 0$.

$$\begin{cases} \text{Max} \displaystyle\sum_{i=1}^{N} p_i x_i \\ \text{s.t} \displaystyle\sum_{i=1}^{N} w_{ji} x_i \leq C_j; \forall j = 1, \cdots, d \\ x_i \in \{0,1\}; \forall i = 1, \cdots, N \end{cases}$$

capacity constraint of resources in each dimension

N : the number of items

p_i : the profit of the i -th item

w_{ji} : the j-th dimension weight of the i -th item

C_j : the j-th dimension capacity of the knapsack

x_i : binary decision variable

it equals to 1 if i -th item is selected, and 0 otherwise.

Multiple MKP

- Consider multiple knapsacks where each one (say knapsack k), has d dimensions with limited capacity C_{kj}.

$$
\begin{cases}
Max \sum_{k=1}^{M} \sum_{i=1}^{N} p_i x_{ik} \\
s.t \sum_{i=1}^{N} w_{ij}^{k} x_{ik} \leq C_j^k; \forall j = 1, \cdots, d; \forall k = 1, \cdots, M; \\
\sum_{k=1}^{M} x_{ik} \leq 1; \forall i = 1, \cdots, N; \\
x_{ik} \in \{0,1\}; \forall i = 1, \cdots, N; k = 1, \cdots, M;
\end{cases}
$$

each item appears at most once in all knapsacks

x_{ik}*: binary decision variable*

$x_{ik} = 1$ *if* i*-th item is selected and packed into knapsack k,*
and $x_{ik} = 0$ *otherwise.*

Encoding Nurse Rostering to ILP

Nurse Rostering :

The basic problem consists in the weekly scheduling of a fixed number of nurses using a set of shifts, such that in each day a nurse works a shift or has a day-off.

Nurses may have multiple skills, and for each skill we are given different coverage requirements.

Data and variables

- N: the set of nurses
- W: the set of all weeks in the scheduling period
- D: the set of days in each week (D={1,2,3,4,5,6,7})
- S: the set of shift types

Variables:

$\forall n \in N, \forall d \in D, \forall s \in S$:

$x_{n,w,d,s} = 1$ if nurse $\boldsymbol{n}$ works shift type $\boldsymbol{s}$ on the $\boldsymbol{dth}$ day of week $\boldsymbol{w}$

$= 0$ otherwise.

Hard (H) constraint types:

- H1. Single assignment per day:
  →A nurse can be assigned to at most one shift per day

- ILP: $\forall n \in N, \forall w \in W, \forall d \in D: \sum_{s \in S} x_{n,w,d,s} \leq 1$

Hard (H) constraint types:

- H2. Sufficient-staffing:

  →The number of nurses for each shift for each skill must be at least equal to the minimum requirement

- ILP: Let $C_{w,d,s}^{min}$ denote the minimum number of nurses required for covering a shift ***s*** on the ***dth*** day of week ***w***, then:

  $\forall \mathrm{w} \in \mathrm{W}, \forall \mathrm{d} \in \mathrm{D}, \forall \mathrm{s} \in \mathrm{S}: \ \sum_{n \in N} x_{n,w,d,s} \geq C_{w,d,s}^{min}$

Hard (H) constraint types:

- H3. Shift type successions:

  →The shift type assignments of one nurse in two consecutive days must belong to the legal successions provided in the scenario.

- ILP: Let ***F*** be the set of forbidden shift type successions. Each ***f*** ∈ ***F*** represents a sequence of two shift types ***s1*** and ***s2*** that is forbidden. T.i. a shift ***s2*** cannot follow a shift type ***s1***:

  $\forall n \in N, \forall w \in W, \forall d \in D, \forall f \in F : x_{n,w,d,s1} + x_{n,w,d+1,s2} \leq 1$

  (ps: if d = 7, then $x_{n,w,7,s1} + x_{n,w+1,1,s2} \leq 1$)

Soft (S) constraints types:

Soft constraints:
- Allowed falsified
- Incur a penalty to the cost

Alternatively, we can have an objective function.

Objectives

- Complete weekend

→Every nurse that has the complete weekend value set to true, must work both week-end days or none.

If a nurse works only one of the two days *Sat* and *Sun*, this is penalized by the corresponding penalty weight w_1.

(the penalty weight can vary among the nurses. Here for simplicity, we assume all nurses have the same penalty weight.)

Objectives

Complete weekend

- $p_{n,w,d}$ =1 if nurse $\boldsymbol{n}$ works any shift type on the $\boldsymbol{dth}$ day of week $\boldsymbol{w}$

  $\boldsymbol{s.t.}\ p_{n,w,d} = \sum_{s \in S} x_{n,w,d,s}$
- Mathematical constraint:

  →∀n ∈ N, ∀w ∈ W: $p_{n,w,6}$ - $p_{n,w,7}$ = 0

- ILP: ∀n ∈ N, ∀w ∈ W:
- s.t. $p_{n,w,6}$ - $p_{n,w,7}$ + $y_{n,w,1} \geq 0$ and $p_{n,w,7}$ - $p_{n,w,6}$ + $y_{n,w,2} \geq 0$

  obj: min z1 := w_1*($\sum_{\mathrm{n} \in N, w \in W} y_{n,w,1} + y_{n,w,2}$)

Objectives

- S2: Total assignments

→For each nurse the total number of assignments (working days) must at least reach the minimum requirement. The difference, multiplied by its weight, is added to the objective function.

Objectives

- S2: Total assignments
- $p_{n,w,d}$ =1 if nurse $\boldsymbol{n}$ works any shift type on the $\boldsymbol{dth}$ day of week $\boldsymbol{w}$

  $\boldsymbol{s.t.}\ p_{n,w,d} = \sum_{s\in S} x_{n,w,d,s}$

Mathematical constraint:

→$\forall \mathrm{n} \in \mathrm{N}, \sum_{w\in W,d\in D} p_{n,w,d} \geq \mathrm{T}^{\mathrm{min}}$ 每个护士必须上班的最少天数

ILP: $\forall \mathrm{n} \in \mathrm{N}$

$\quad$ s.t. $\sum_{w\in W,d\in D} p_{n,w,d} + q_n \geq \mathrm{T}^{\mathrm{min}}$

$\quad$ Obj: min z2 := $w_2 * (\sum_{n\in N} q_n)$

注意q_n是对于缺失的天数的惩罚。

Overall:

Min z = z1 +z2 = $w_1 \sum_{n \in N, w \in W} (y_{n,w,1} + y_{n,w,2}) + w_2 (\sum_{n \in N} q_n)$

s.t. $\forall n \in N, \forall w \in W, \forall d \in D: \; \sum_{s \in S} x_{n,w,d,s} \leq 1$ (H1)

$\forall w \in W, \forall d \in D, \forall s \in S: \; \sum_{n \in N} x_{n,w,d,s} \geq C_{w,d,s}^{min}$ (H2)

$\forall n \in N, \forall w \in W, \forall d \in D, \forall f \in F: x_{n,w,d,s1} + x_{n,w,d+1,s2} \leq 1$ (H3)

$\forall n \in N, \forall w \in W, \forall d \in D, \quad p_{n,w,d} = \sum_{s \in S} x_{n,w,d,s}$ (H4)

$\forall n \in N, \forall w \in W, p_{n,w,6} - p_{n,w,7} + y_{n,w,1} \geq 0$ and $p_{n,w,7} - p_{n,w,6} + y_{n,w,2} \geq 0$ (S1)

$\forall n \in N \sum_{w \in W, d \in D} p_{n,w,d} + \sum q_n \geq T^{min}$ (S2)

- Transfrom ILP to 0-1 ILP (PBO)

ILP: $\forall n \in N$

s.t. $\sum_{w\in W,d\in D} p_{n,w,d} + q_n \geq T^{\min}$

Obj: min z2 := $w_2 * (\sum_{n\in N} q_n)$

0-1 ILP: $\forall n \in N$

s.t. $\sum_{w\in W,d\in D} p_{n,w,d} + \sum_{t\in[1,T^{\min}]} q_{n,t} \geq T^{\min}$

Obj: min z2 := $w_2 * (\sum_{n\in N} \sum_{t\in[1,T^{\min}]} q_{n,t})$

$q_{n,t}$为0-1变量。

Models

- SAT
- MaxSAT
- Integer Linear Programming
- CSP

Constraint Satisfaction Problem (CSP)

- Constraint Satisfiability Problem
- P=<X,D,C>
  - X: variables
  - D: domains
  - C: constraints

- Express constraints
  - Extensional
  - Intensional

Encoding the n-queue problem to CSP

	Q		
			Q
Q			
		Q	

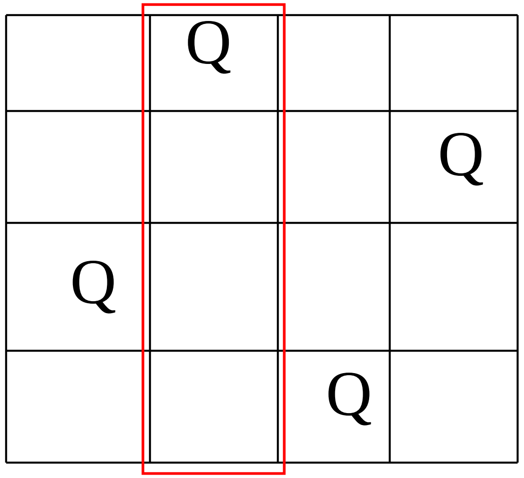
Variables: x_1, x_2, x_3, x_4

Constraint Satisfiability Problem (CSP)

- Constraint Satisfiability Problem
- P=<X,D,C>
  - X: variables
  - D: domains
  - C: constraints

- Express constraints
  - Extensional
  - Intensional

Encoding the n-queue problem to CSP



Variables: x_1, x_2, x_3, x_4

D_1, D_2, D_3, D_4 (D_i = {1,2,3,4})

$x_i \neq x_j \quad (0<i<j\leq n)$;

$|x_i - x_j| \neq |i-j|. \quad (0<i<j\leq n)$

CSP: Global Constraints

- `alldifferent`(X)

  //Enforce all variables of the collection X to take distinct values.
- `all`**equal**(X)

  //Enforce all variables of the collection X to take the same value.
- `atleast`(N,X,value)

  //At least N variables of the collection are assigned to the value.
- regular(X,DFA)

  //accepted by a DFA

a

- abs_value
- all_differ_from_at_least_k_pos
- all_differ_from_at_most_k_pos
- all_differ_from_exactly_k_pos
- all_equal
- all_equal_peak
- all_equal_peak_max
- all_equal_valley
- all_equal_valley_min
- all_incomparable
- all_min_dist
- alldifferent
- alldifferent_between_sets
- alldifferent_consecutive_values
- alldifferent_cst
- alldifferent_except_0
- alldifferent_interval
- alldifferent_modulo
- alldifferent_on_intersection
- alldifferent_partition
- alldifferent_same_value
- allperm
- among
- among_diff_0
- among_interval
- among_low_up
- among_modulo
- among_seq
- among_var
- and
- arith
- arith_or
- arith_sliding
- assign_and_counts
- assign_and_nvalues
- atleast
- atleast_nvalue
- atleast_nvector
- atmost
- atmost1
- atmost_nvalue
- atmost_nvector

b

- balance
- balance_cycle
- balance_interval
- balance_modulo
- balance_partition
- balance_path
- balance_tree
- between_min_max
- big_peak
- big_valley
- bin_packing
- bin_packing_capa
- binary_tree
- bipartite

c

- calendar
- cardinality_atleast
- cardinality_atmost
- cardinality_atmost_partition
- change
- change_continuity
- change_pair
- change_partition
- change_vectors
- circuit
- circuit_cluster
- circular_change
- clause_and
- clause_or
- clique
- colored_matrix
- coloured_cumulative
- coloured_cumulatives
- common
- common_interval
- common_modulo
- common_partition
- compare_and_count
- cond_lex_cost
- cond_lex_greater
- cond_lex_greatereq
- cond_lex_less
- cond_lex_lesseq
- connect_points
- connected
- consecutive_groups_of_ones
- consecutive_values
- contains_sboxes
- correspondence
- count
- counts
- coveredby_sboxes
- covers_sboxes
- crossing
- cumulative
- cumulative_convex
- cumulative_product
- cumulative_two_d
- cumulative_with_level_of_priority
- cumulatives
- cutset
- cycle
- cycle_card_on_path
- cycle_or_accessibility
- cycle_resource
- cyclic_change
- cyclic_change_joker

d

- dag
- decreasing
- decreasing_peak
- decreasing_valley
- deepest_valley
- derangement
- differ_from_at_least_k_pos
- differ_from_at_most_k_pos
- differ_from_exactly_k_pos
- diffn
- diffn_column
- diffn_include
- discrepancy
- disj
- disjoint
- disjoint_sboxes
- disjoint_tasks
- disjunctive
- disjunctive_or_same_end
- disjunctive_or_same_start
- distance
- distance_between
- distance_change
- divisible
- divisible_or
- dom_reachability
- domain
- domain_constraint

e

- elem
- elem_from_to
- element
- element_greatereq
- element_lesseq
- element_matrix
- element_product
- element_sparse
- elementn
- elements
- elements_alldifferent
- elements_sparse
- eq
- eq_cst
- eq_set
- equal_sboxes
- equilibrium
- equivalent
- exactly

f

g

gcd geost geost_time geq geq_cst global_cardinality
global_cardinality_low_up global_cardinality_low_up_no_loop global_cardinality_no_loop global_cardinality_with_costs global_contiguity golomb
graph_crossing graph_isomorphism group group_skip_isolated_item gt

h

highest_peak

i

imply in in_interval in_interval_reified in_intervals in_relation
in_same_partition in_set incomparable increasing increasing_global_cardinality increasing_nvalue
increasing_nvalue_chain increasing_peak increasing_sum increasing_valley indexed_sum inflexion
inside_sboxes int_value_precede int_value_precede_chain interval_and_count interval_and_sum inverse
inverse_offset inverse_set inverse_within_range ith_pos_different_from_0

k

k_alldifferent k_cut k_disjoint k_same k_same_interval k_same_modulo
k_same_partition k_used_by k_used_by_interval k_used_by_modulo k_used_by_partition

l

length_first_sequence length_last_sequence leq leq_cst lex2 lex_alldifferent
lex_alldifferent_except_0 lex_between lex_chain_greater lex_chain_greatereq lex_chain_less lex_chain_lesseq
lex_different lex_equal lex_greater lex_greatereq lex_less lex_lesseq
lex_lesseq_allperm link_set_to_booleans longest_change longest_decreasing_sequence longest_increasing_sequence lt

m

map max_decreasing_slope max_increasing_slope max_index max_n max_nvalue
max_occ_of_consecutive_tuples_of_values max_occ_of_sorted_tuples_of_values max_occ_of_tuples_of_values max_size_set_of_consecutive_var maximum maximum_modulo
meet_sboxes min_decreasing_slope min_dist_between_inflexion min_increasing_slope min_index min_n
min_nvalue min_size_full_zero_stretch min_size_set_of_consecutive_var min_width_peak min_width_valley minimum
minimum_except_0 minimum_greater_than minimum_modulo minimum_weight_alldifferent multi_global_contiguity multi_inter_distance
multiple

n

nand nclass neq neq_cst nequivalence next_element
next_greater_element ninterval no_peak no_valley non_overlap_sboxes nor
not_all_equal not_in npair nset_of_consecutive_values nvalue nvalue_on_intersection
nvalues nvalues_except_0 nvector nvectors nvisible_from_end nvisible_from_start

o

open_alldifferent
open_among
open_atleast
open_atmost
open_global_cardinality
open_global_cardinality_low_up
open_maximum
open_minimum
opposite_sign
or
orchard
order
ordered_atleast_nvector
ordered_atmost_nvector
ordered_global_cardinality
ordered_nvector
orth_link_ori_siz_end
orth_on_the_ground
orth_on_top_of_orth
orths_are_connected
overlap_sboxes

p

path
path_from_to
pattern
peak
period
period_except_0
period_vectors
permutation
place_in_pyramid
polyomino
power
precedence
product_ctr
proper_circuit
proper_forest

r

range_ctr
relaxed_sliding_sum
remainder
roots

s

same
same_and_global_cardinality
same_and_global_cardinality_low_up
same_intersection
same_interval
same_modulo
same_partition
same_sign
scalar_product
sequence_folding
set_value_precede
shift
sign_of
size_max_seq_alldifferent
size_max_starting_seq_alldifferent
sliding_card_skip0
sliding_distribution
sliding_sum
sliding_time_window
sliding_time_window_from_start
sliding_time_window_sum
smooth
soft_all_equal_max_var
soft_all_equal_min_ctr
soft_all_equal_min_var
soft_alldifferent_ctr
soft_alldifferent_var
soft_cumulative
soft_same_interval_var
soft_same_modulo_var
soft_same_partition_var
soft_same_var
soft_used_by_interval_var
soft_used_by_modulo_var
soft_used_by_partition_var
soft_used_by_var
some_equal
sort
sort_permutation
stable_compatibility
stage_element
stretch_circuit
stretch_path
stretch_path_partition
strict_lex2
strictly_decreasing
strictly_increasing
strongly_connected
subgraph_isomorphism
sum
sum_ctr
sum_cubes_ctr
sum_free
sum_of_increments
sum_of_weights_of_distinct_values
sum_powers4_ctr
sum_powers5_ctr
sum_powers6_ctr
sum_set
sum_squares_ctr
symmetric
symmetric_alldifferent
symmetric_alldifferent_except_0
symmetric_alldifferent_loop
symmetric_cardinality
symmetric_gcc

t

temporal_path
tour
track
tree
tree_range
tree_resource
twin
two_layer_edge_crossing
two_orth_are_in_contact
two_orth_column
two_orth_do_not_overlap
two_orth_include

u

used_by
used_by_interval
used_by_modulo
used_by_partition
uses

v

valley
vec_eq_tuple
visible

CSP Encoding: Graph Coloring

From www.minizinc.org

AUST ≡ [DOWNLOAD]

```
% Colouring Australia using nc colours
int: nc = 3;

var 1..nc: wa;   var 1..nc: nt;  var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;   var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
solve satisfy;

output ["wa=", show(wa), "\t nt=", show(nt),
        "\t sa=", show(sa), "\n", "q=", show(q),
        "\t nsw=", show(nsw), "\t v=", show(v), "\n",
         "t=", show(t),  "\n"];
```

CSP Encoding: Puzzle

• SEND+MORE=MONEY, what is the value of each letter in the equation?

SEND-MORE-MONEY ≡ [DOWNLOAD]

```
include "alldifferent.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint           1000 * S + 100 * E + 10 * N + D
                   + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   ",show(S),show(E),show(N),show(D),"\n",
        "+  ",show(M),show(O),show(R),show(E),"\n",
        "= ",show(M),show(O),show(N),show(E),show(Y),"\n"];
```

From www.minizinc.org

CSP Encoding: Job Shop Scheduling

Assign jobs to a machine

- Sequential
- Handle one job at any time

```
int: jobs;                                        % no of jobs
int: tasks;                                       % no of tasks per job
array [1..jobs,1..tasks] of int: d;               % task durations
int: total = sum(i in 1..jobs, j in 1..tasks)
             (d[i,j]);                            % total duration
int: digs = ceil(log(10.0,int2float(total))); % digits for output
array [1..jobs,1..tasks] of var 0..total: s;  % start times
var 0..total: end;                                % total end time


constraint %% ensure the tasks occur in sequence
    forall(i in 1..jobs) (
        forall(j in 1..tasks-1)
            (s[i,j] + d[i,j] <= s[i,j+1]) /\
        s[i,tasks] + d[i,tasks] <= end
    );

constraint %% ensure no overlap of tasks
    forall(j in 1..tasks) (
        forall(i,k in 1..jobs where i < k) (
            s[i,j] + d[i,j] <= s[k,j] \/
            s[k,j] + d[k,j] <= s[i,j]
        )
    );

solve minimize end;

output ["end = ", show(end), "\n"] ++
       [ show_int(digs,s[i,j]) ++ " " ++
         if j == tasks then "\n" else "" endif |
         i in 1..jobs, j in 1..tasks ];
```

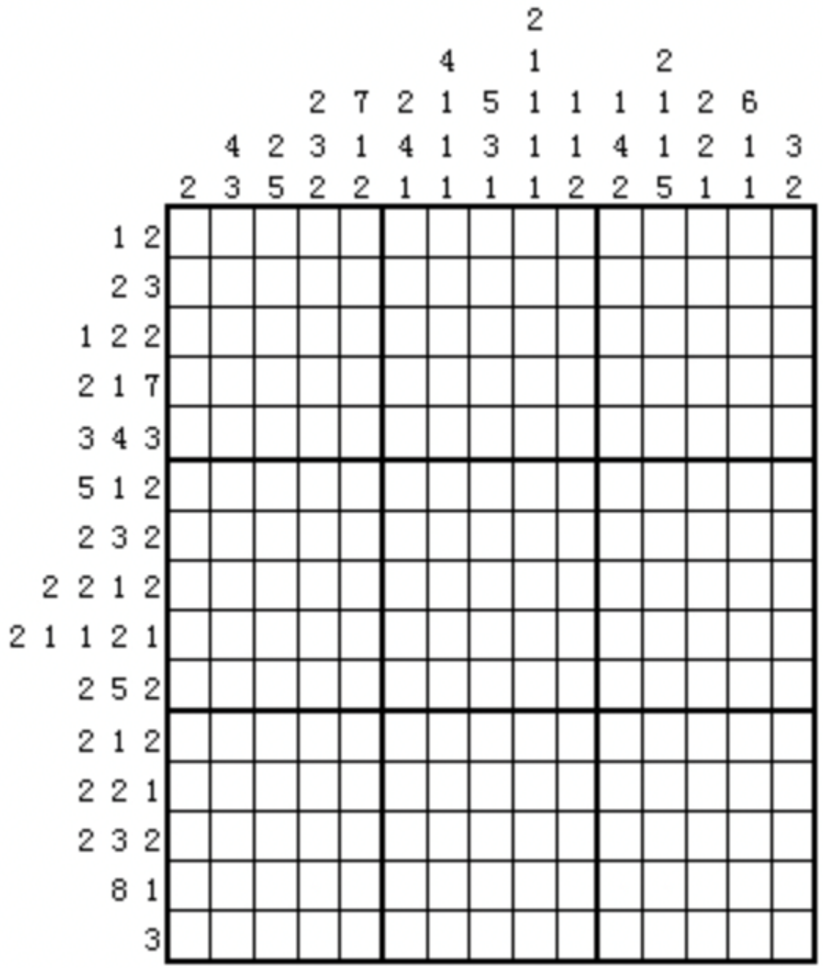
CSP Encoding: Nonogram

/** * CSPLib prob012:
 * "Nonograms are a popular puzzles, which goes by different names in different countries. *

Models have to shade in squares in a grid so that blocks of consecutive shaded squares satisfy constraints given for each row and column.

Constraints typically indicate the sequence of shaded blocks (e.g. 3,1,2 means that there is a block of 3, then a gap of unspecified size, a block of length 1, another gap, and then a block of length 2)."

@author Charles Prud'homme @since 08/08/11 */



CSP Encoding: Nonogram

```java
public void buildModel() {
    model = new Model();
    int nR = data.getR().length;
    int nC = data.getC().length;
    vars = new BoolVar[nR][nC];
    for (int i = 0; i < nR; i++) {
        for (int j = 0; j < nC; j++) {
            vars[i][j] = model.boolVar(format("B_%d_%d", i, j));
        }
    }
    for (int i = 0; i < nR; i++) {
        dfa(vars[i], data.getR(i), model);
    }
    for (int j = 0; j < nC; j++) {
        dfa(ArrayUtils.getColumn(vars, j), data.getC(j), model);
    }

}
```

```java
private void dfa(BoolVar[] cells, int[] rest, Model model) {
    StringBuilder regexp = new StringBuilder("0*");
    int m = rest.length;
    for (int i = 0; i < m; i++) {
        regexp.append('1').append('{').append(rest[i]).append('}');
        regexp.append('0');
        regexp.append(i == m - 1 ? '*' : '+');
    }
    IAutomaton auto = new FiniteAutomaton(regexp.toString());
    model.regular(cells, auto).post();
}
```

CSP Modeling

- Choco
- MiniZinc

Models

- SAT
- MaxSAT
- Integer Linear Programming
- CSP
- SMT

The Logic Languages

SAT: Propositional Satisfiability

$(\text{Tie} \vee \text{Shirt}) \wedge (\neg\, \text{Tie} \vee \neg\, \text{Shirt}) \wedge (\neg\, \text{Tie} \vee \text{Shirt})$

FOL: First-order Logic

$\forall X,Y,Z\ [X*Y*Z] = (X*Y)*Z]$

$\forall X[X*inv(X)=e]\ \forall X[X*e=e]$

$\forall n \in \{z|z > 2, z \in Z\} \neg \exists x, y, z \in \mathrm{Z}\ (x^n + y^n = z^n)$

SMT: Satisfiability Modulo background Theories

$b+2 = c \wedge A[3] \neq A[c-b+1]$

First Order Logic (FOL)

- First-order logic (FOL), also called predicate logic and the first-order predicate calculus.
- FOL extends propositional logic with predicates, functions, and quantifiers.
    - variables x, y, z, x1, x2, . . .
    - constants a, b, c, a1, a2, ...
    - Terms evaluate to values other than truth values, integers, people, or cards of a deck. //objects
        - More complicated terms are constructed using functions.

Example: these are terms
a, a constant (or 0-ary function);
x, a variable;
f(a), a unary function f applied to a constant;
g(x, b), a binary function g applied to a variable x and a constant b;
f(g(x, f(b))).

First Order Logic (FOL)

- First-order logic (FOL), also called predicate logic and the first-order predicate calculus.
- FOL extends propositional logic with predicates, functions, and quantifiers.

  - Predicates P, Q… // properties, relations of objects
    - An n-ary predicate takes n terms as arguments.
    - Example: x is a student S(x)
      - Andy is a student S(Andy)
      - Bob is not a student $\neg S(Bob)$
    - Example: y is a teacher of x T(y, x)
      - John is a teacher of Andy T(John, Andy)
  - An atom is $\top$, $\bot$, or an n-ary predicate applied to n terms.

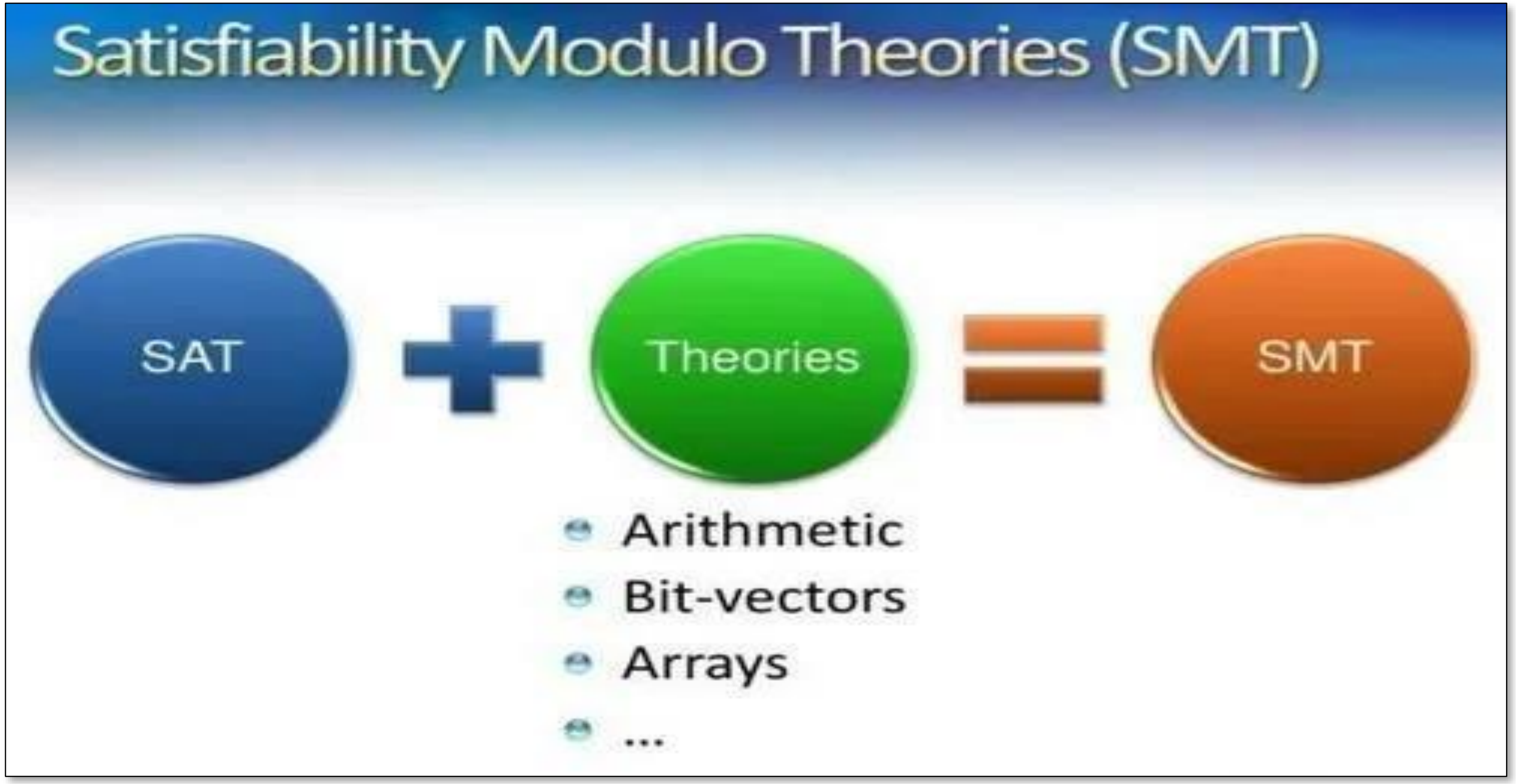
  - A literal is an atom or its negation.

- First-order logic (FOL), also called predicate logic and the first-order predicate calculus.
- FOL extends propositional logic with predicates, functions, and quantifiers.

  - Quantifiers
  - the existential quantifier $\exists x.\ F[x]$, read "there exists an x such that F[x]";
  - the universal quantifier $\forall x.\ F[x]$, read "for all x, F[x]".

  - A FOL formula is
    - a literal,
    - the application of a logical connective $\neg$, $\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$ to a formula or formulae,
    - or the application of a quantifier to a formula.

Satisfiability Modulo Theories



From Propositional to Quantifier-Free Theories

Example:
$\phi := (x_1 - x_2 \leq 13 \vee x_2 \neq x_3) \wedge (x_2 = x_3 \rightarrow x_4 > x_5) \wedge A \wedge \neg B$

Propositional Skeleton $\mathrm{PS}_\Phi = (b_1 \vee \neg b_2) \wedge (b_2 \rightarrow b_3) \wedge A \wedge \neg B$

$b_1 : x_1 - x_2 \leq 13$

$b_2 : x_2 = x_3$

$b_3 : x_4 > x_5$

From Propositional to Quantifier-Free Theories

Example:

- $\mathrm{a} = \mathrm{b} + 2 \land A = write(B, a + 1,4) \land (read(A, b + 3) = 2 \lor f(a - 1) \neq f(b + 1))$

- Propositional Skeleton $\mathrm{PS}_{\Phi} = y_1 \land y_2 \land (y_3 \lor y_4)$

- y_1: $\mathrm{a} = \mathrm{b} + 2$
- y_2: $A = write(B, a + 1,4)$
- y_3: $read(A, b + 3) = 2$
- y_4: $f(a - 1) \neq f(b + 1)$

Language: Signatures

- A first-order theory T is defined by the following components.
  1. Its signature Σ is a set of constant, function, and predicate symbols.
     - A constant can also be viewed as a 0-ary function
     - A FOL propositional variable is a 0-ary predicate, which we write A, B, C, …
  2. Its set of axioms $\mathcal{A}$ is a set of closed FOL formulae in which only constant, function, and predicate symbols of Σ appear.

  - A Σ-formula is constructed from constant, function, and predicate symbols of Σ, as well as variables, logical connectives, and quantifiers.

  - As usual, the symbols of Σ are just symbols without prior meaning.
  - The axioms A provide their meaning.

Interpretation

Recall

- An interpretation I assigns to every propositional variable exactly one truth value. For example, I : $\{P \longmapsto true, Q \longmapsto$ false, ...}
- A formula F is satisfiable iff there exists an interpretation I such that I $\vDash$ F.
- A formula F is valid iff for all interpretations I, I $\vDash$ F

Interpretation

- FOL interpretation I: (D_I, α_I)
- The domain D_I of an interpretation I is a nonempty set of values or objects, such as integers, real numbers, dogs, people, or merely abstract objects…

The **assignment** α_I of interpretation I maps constant, function, and predicate symbols to elements, functions, and predicates over D_I. It also maps variables to elements of D_I:

- each variable symbol x is assigned a value x_I from D_I;
- each n-ary function symbol f is assigned an n-ary function

  $$f_I : D_I^n \rightarrow D_I$$

  that maps n elements of D_I to an element of D_I;
- each n-ary predicate symbol p is assigned an n-ary predicate

  $$p_I : D_I^n \rightarrow \{\text{true}, \text{false}\}$$

  that maps n elements of D_I to a truth value.

Interpretation

Example

- F : x + y > z → y > z − x

- We construct a "standard" interpretation I
- The domain is the integers, $\mathbb{Z}$: $D_I = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- $\alpha_I: \{+ \longmapsto +_{\mathbb{Z}}, - \longmapsto -_{\mathbb{Z}}, > \longmapsto >_{\mathbb{Z}}, x \longmapsto 13, y \longmapsto 42, x \longmapsto 1\}$

T-satisfiability

- Given a FOL formula F and interpretation $I: (D_I, \alpha_I)$, we want to compute if F evaluates to true under interpretation I, I ⊨ F, or if F evaluates to false under interpretation I, I ⊭F.
  - I satisfies F: I ⊨ F

- T – interpretation: an interpretation satisfying $I \vDash$ A for every A $\in \mathcal{A}$.

- A Σ-formula F is satisfiable in T , or T -satisfiable, if there is a T-interpretation I that satisfies F.

Input Format : SMT-LIB2

- First, directives. E.g., asking models to be reported:

(set - option : produce - models true)

- Second, set background theory:

(set - logic QF_LIA)

- Standard theories of interest :
    - QF_BV: quantifier-free bit vector theory
    - QF_LRA : quantifier-free linear real arithmetic
    - QF_LIA: quantifier-free linear integer arithmetic
    - QF_NRA : quantifier-free nonlinear real arithmetic
    - QF_NIA : quantifier-free nonlinear integer arithmetic
    - …

Input Format : SMT-LIB2

- Third, declare variables

(declare-fun x () s), or (declare-const x s) //introducing new symbols `x` of sort `s`

common sorts: `Int Bool Real (_BitVec 3) ((_FixedSizedList 4) Real) (Set (_BitVec 3))`

E.g., integer variable x:

$$(\text{declare} - \text{fun}\ x\ ()\ \text{Int})$$

E.g., real variable z_1_3:

$$(\text{declare} - \text{fun}\ z\_1\_3\ ()\ \text{Real})$$

Input Format : SMT-LIB2

- Fourth, assert formula.
- Expressions should be written in prefix form:

  (< operator > < arg 1 > … < arg n >)

```
( assert
  ( and
    ( or
      ( <=  (+  x  3)  ( *  2  u ) )
      ( >=  (+  v  4)   y )
      ( >=  (+  x  y  z )   2)
    )
    ( = 7
      (+
        ( ite  ( and  ( <=  x  2 )  ( <= 2   (+  x  3  (- 1) ) ) )   3   0)
        ( ite  ( and  ( <=  u  2 )  ( <= 2   (+  u  3  (- 1) ) ) )   4   0)
      )
    )
  )
)
```

- and, or, + have arbitrary arity
- \- is unary or binary
- * is binary
- ite is the if-then-else operator (like ? in C, C++, Java).

Let a be Boolean and b, c have the same sort S, then (ite a b c) is the expression of sort S equal to:

- b if a holds
- c if a does not hold

Input Format : SMT-LIB2

- Finally ask the SMT solver to check satisfiability …

(check - sat)

- … and report the model

(get - model)

- Anything following a ; up to an end-of-line is a comment
  - You can also use (set-info : comments) to write comments in your files

Input Format : SMT-LIB2

```
( set - option  :  produce - models true )
( set - logic   QF_LIA )
( declare - fun x ()    Int )
( declare - fun y ()    Int )
( declare - fun z ()    Int )          ;           This  is  an  example
( declare - fun u ()    Int )
( declare - fun v ()    Int )
( assert
   ( and
      ( or
        ( <= (+  x  3) (*  2  y ) )
        ( >= (+  x  4)   z )
      )
   (<= x  y))
)
( check - sat )
( get - model )
```

Input Format : SMT-LIB2

```
(set-logic QF_LIA)
(declare-fun x ()   Int)
(declare-fun y ()   Int)
(declare-fun z ()   Int)
(assert ( or  ( >  x  y)  ( >  x  z ) ) )
(assert ( or  ( <  ( +  x  1 )  y )  ( not  ( >  x  y ) ) ) )
(assert ( or  ( >  x  y )  ( >  z  y ) ) ) )
(check-sat)
```

Example

Input Format : SMT-LIB2

```
; There is a fast way to check that fixed size numbers are powers of two.
; It turns out that a bit-vector x is a power of two or zero if and only if x & (x - 1) is zero,
where & represents the bitwise and.
; When using Z3, if you do not set logic, it means all logics supported in Z3.

(define-fun is-power-of-two ((x (_ BitVec 4))) Bool
  (= #x0 (bvand x (bvsub x #x1))))
(declare-const a (_ BitVec 4))
(assert
 (not (= (is-power-of-two a)
         (or (= a #x0)
             (= a #x1)
             (= a #x2)
             (= a #x4)
             (= a #x8)))))
(check-sat)
```

Output Format : SMT-LIB2

- 1st line is sat or unsat

- If satisfiable, then comes a description of the solution in a model expression, where the value of each variable is given by:

$$(\text{define} - \text{fun} < \text{variable} > () < \text{sort} > < \text{value} >)$$

- Example:

```
sat
( model
    ( define - fun y ()   Int 0 )
    ( define - fun x ()   Int (- 3) )
    ( define - fun z ()   Int 2 )
)
```

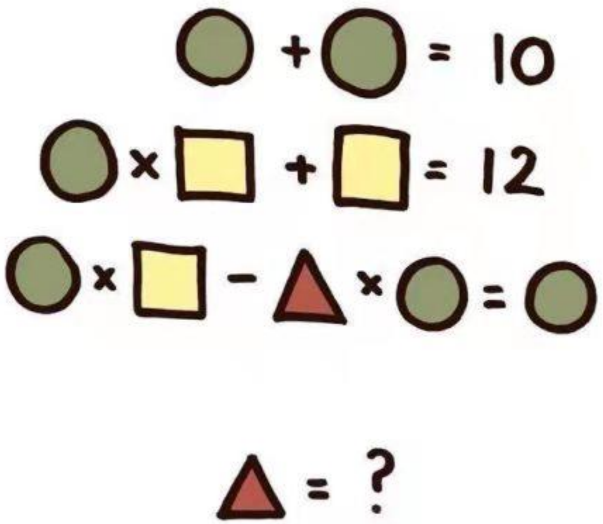
SMT Encoding (Programming) – solving equations

It's that easy to solve it in Z3:

```
#!/usr/bin/python
from z3 import *

circle, square, triangle = Ints('circle square triangle')
s = Solver()
s.add(circle+circle==10)
s.add(circle*square+square==12)
s.add(circle*square-triangle*circle==circle)
print s.check()
print s.model()
```

```
sat
[triangle = 1, square = 2, circle = 5]
```



SMT Encoding (Programming) - Sudoku

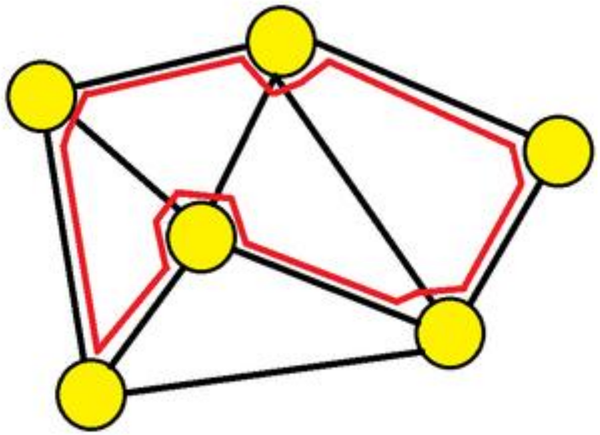
SMT-solvers are so helpful, in that our Sudoku solver has nothing else, we have just defined relationships between variables (cells).

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

```
% time python sudoku2_Z3.py
1 4 5 3 2 7 6 9 8
8 3 9 6 5 4 1 2 7
6 7 2 9 1 8 5 4 3
4 9 6 1 8 5 3 7 2
2 1 8 4 7 3 9 5 6
7 5 3 2 9 6 4 8 1
3 6 7 5 4 2 8 1 9
9 8 4 7 6 1 2 3 5
5 2 1 8 3 9 7 6 4

real    0m0.382s
user    0m0.346s
sys     0m0.036s
```

SMT Encoding (Programming) – Hamiltonian cycle



A **Hamiltonian path** (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once
A **Hamiltonian cycle** (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.
NP complete problem.

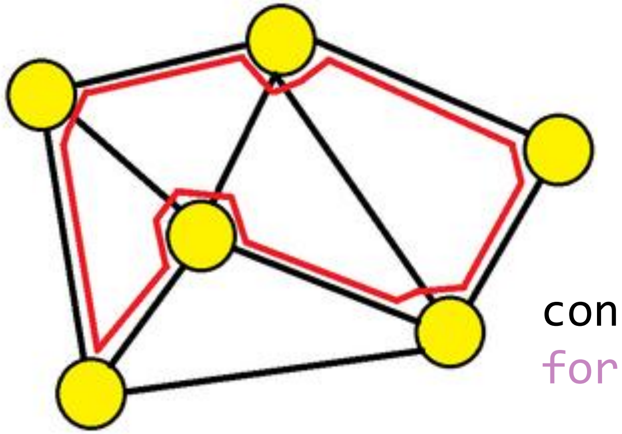
The position of every node in hamiltonian cycle order array should be a integer in [0, N).

$$\forall\, i \in [0, 1, \dots, N\ -1](pos[i] \in [0, 1, \dots, N\ -\ 1] \wedge pos[i] \in \mathrm{Z})$$

For every node, there should be one node which is just next to it in hamiltonian cycle order.

$$\forall\, i \in [0, 1, \dots, N\ -1]\exists j\, \{j \in [0, 1, \dots, N\ -\ 1] \wedge edge(i, j) \in G \wedge pos[j] \equiv (pos[i]\ + 1)\ \%\ N\}$$

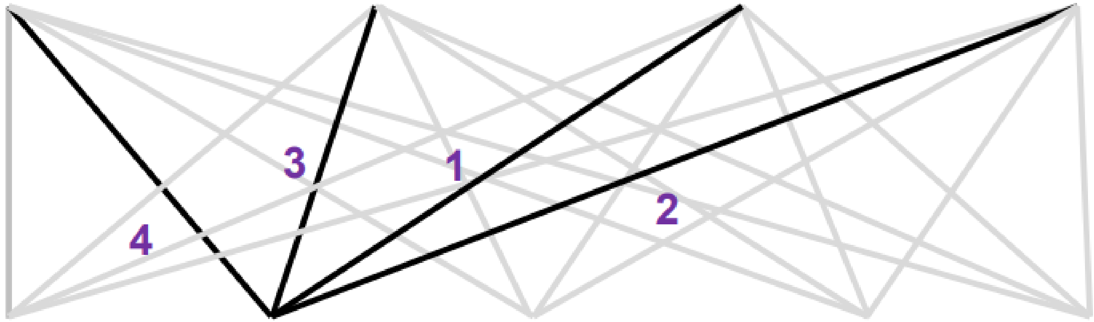
SMT Encoding (Programming) – Hamiltonian cycle



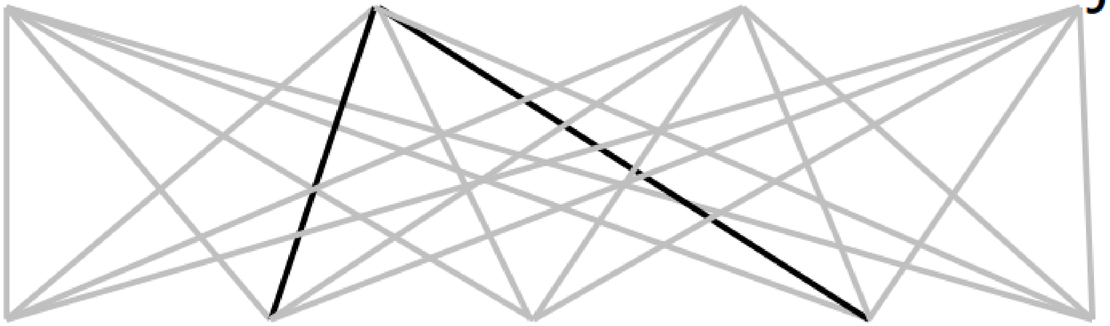
```
constraint <- {}
for i : {i | i in [0, N)} do
    constraint.add_clause(0 <= pos[i] < N and is_integer(pos[i]))
end for
constraint.add_clause(pos[0] == 0)
for i : {i | i in [0, N)} do
    or_clause <- {}
    for j : {j | node j can be reached by node i in graph} do
        or_clause.add_literal(pos[j] == (pos[i] + 1) % N)
    end for
    constraint.add_clause(or_clause)
end for
```

SMT Encoding (Programming) – Job Scheduling

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

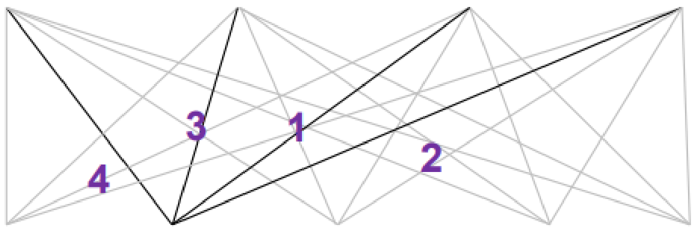


$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

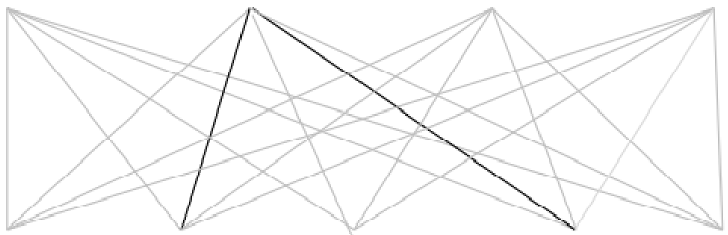
SMT Encoding (Programming) – Job Scheduling

Constraints:

Precedence:



Resource:



$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$

Encoding:

$t_{2,3}$ - start time of job 2 on mach 3

$d_{2,3}$ - duration of job 2 on mach 3

$$t_{2,3} + d_{2,3} \leq t_{2,4}$$

Not convex

$$t_{2,2} + d_{2,2} \leq t_{4,2}$$
$$\vee$$
$$t_{4,2} + \mathrm{d}_{4,2} \leq t_{2,2}$$

SMT Encoding (Programming) – Job Scheduling

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution

$t_{1,1} = 5,\ t_{1,2} = 7,\ t_{2,1} = 2,$
$t_{2,2} = 6,\ t_{3,1} = 0,\ t_{3,2} = 3$

Encoding

$$
\begin{aligned}
&(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
&(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
&(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
&((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
&((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{aligned}
$$

Constraint Modeling

Homework: find an interesting (real world or research) problem and formulate it into a constraint model.