SATisfiability Solving: How to solve problems with SAT?

Ruben Martins

University of Oxford

February 13, 2014

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]
```

Encoding to CNF

- What to encode?
  - Boolean formulas
    - Tseitin's encoding
    - Plaisted&Greenbaum's encoding
    - ...
  - Natural numbers
  - Cardinality constraints
  - Pseudo-Boolean (PB) constraints
  - ...

Encoding to CNF

- What to encode?
  - Boolean formulas
    - Tseitin's encoding
    - Plaisted&Greenbaum's encoding
    - ...
  - Natural numbers
  - Cardinality constraints
  - Pseudo-Boolean (PB) constraints
  - ...

- There are no CNF problems **!**

  [Source: Peter J. Stuckey 2013]
  - Structure is lost when encoding to CNF

Why CNF?

- Any propositional formula may be converted into an equisatisfiable CNF formula with only linear increase in size:
  - Use Tseitin's encoding **!**

- CNF makes it possible to perform interesting deductions (resolution)

- SAT solvers use CNF as the standard input format

Tseitin's encoding

Convert $\varphi = (a \vee b) \rightarrow (a \vee \bar{c})$ to an equisatisfiable CNF formula

- For each subformula, introduce new variables: t_1 for φ, t_2 for $(a \vee b)$, t_3 for $(a \vee \bar{c})$, and t_4 for $\bar{c}$
- Stipulate equivalences and convert them to CNF:
  - $t_1 \leftrightarrow (t_2 \rightarrow t_3) \Rightarrow \varphi_1 = (\bar{t}_1 \vee \bar{t}_2 \vee t_3) \wedge (t_2 \vee t_1) \vee (\bar{t}_3 \vee t_1)$
  - $t_2 \leftrightarrow (a \vee b) \Rightarrow \varphi_2 = (\bar{t}_2 \vee a \vee b) \wedge (\bar{a} \vee t_2) \wedge (\bar{b} \vee t_2)$
  - $t_3 \leftrightarrow (a \vee \bar{t}_4) \Rightarrow \varphi_3 = (\bar{t}_3 \vee a) \wedge (\bar{t}_3 \vee t_4) \wedge (\bar{a} \vee \bar{t}_4 \vee t_3)$
  - $t_4 \leftrightarrow \bar{c} \Rightarrow \varphi_4 = (t_4 \vee \bar{c}) \wedge (t_4 \vee c)$
- The formula $t_1 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ is equisatisfiable to φ and is in CNF

Tseitin's encoding

- Using automated tools to encode to CNF:

  e.g **limboole**: `http://fmv.jku.at/limboole`

Tseitin's encoding

- Using automated tools to encode to CNF:

  e.g **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

How to encode natural numbers?

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

How to encode natural numbers?

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

- Unary encoding:
  - Each variable x_n is true iff the number is equal to or greater than n:

    e.g. $x_2 = 1$ represents that the number is equal to or greater than 2
  - x_i implies x_{i+1}: $\bigwedge_{i<j} \bar{x}_i \vee x_j$

How to encode natural numbers?

- 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:

  e.g. $x_2 = 1$ represents that the number is equal to or greater than 2
  - x_i implies x_{i+1}: $\bigwedge_{i<j} \bar{x}_i \vee x_j$
- Binary encoding:
  - Use $\lceil log_2 n \rceil$ auxiliary variables to represent n in binary

  e.g. Consider $n = 3$:
  x_0 (number 0) corresponds to the binary representation 00
  $\bar{x}_0 \vee \bar{b}_0$, $\bar{x}_0 \vee \bar{b}_1$

How to encode cardinality constraints?

At-most-one constraints:

- Naive (pairwise) encoding for at-most-one constraints:
  - Cardinality constraint: $x_1 + x_2 + x_3 + x_4 \leq 1$
  - 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 cardinality constraints?

At-most-k constraints:

- Naive encoding for at-most-k 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

Encodings for cardinality constraints

Encoding	Clauses	Variables	Type
Pairwise	$\mathcal{O}(n^2)$	0	at-most-one
Ladder	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Bitwise	$\mathcal{O}(n\ log_2\ n)$	$\mathcal{O}(log_2\ n)$	at-most-one
Commander	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Product	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Sequential	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	at-most-k
Totalizer	$\mathcal{O}(nk)$	$\mathcal{O}(n\ log_2\ n)$	at-most-k
Sorters	$\mathcal{O}(n\ log_2^2\ n)$	$\mathcal{O}(n\ log_2^2\ n)$	at-most-k

Encodings for cardinality constraints

Encoding	Clauses	Variables	Type
Pairwise	$\mathcal{O}(n^2)$	0	at-most-one
Ladder	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Bitwise	$\mathcal{O}(n\ log_2\ n)$	$\mathcal{O}(log_2\ n)$	at-most-one
Commander	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Product	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Sequential	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	at-most-k
Totalizer	$\mathcal{O}(nk)$	$\mathcal{O}(n\ log_2\ n)$	at-most-k
Sorters	$\mathcal{O}(n\ log_2^2\ n)$	$\mathcal{O}(n\ log_2^2\ n)$	at-most-k

- Example on the board

Encodings for cardinality constraints

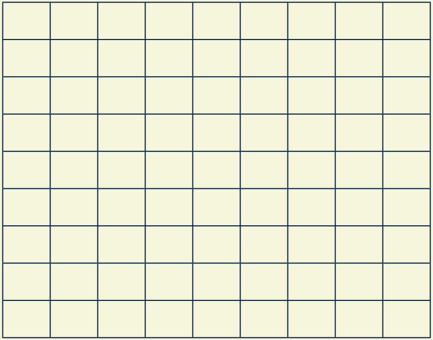
Encoding	Clauses	Variables	Type
Pairwise	$\mathcal{O}(n^2)$	0	at-most-one
Ladder	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Bitwise	$\mathcal{O}(n\ log_2\ n)$	$\mathcal{O}(log_2\ n)$	at-most-one
Commander	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Product	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Sequential	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	at-most-k
Totalizer	$\mathcal{O}(nk)$	$\mathcal{O}(n\ log_2\ n)$	at-most-k
Sorters	$\mathcal{O}(n\ log_2^2\ n)$	$\mathcal{O}(n\ log_2^2\ n)$	at-most-k

- Many more encodings exist
- They can also be generalized to pseudo-Boolean constraints:
  - $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq k$

Simplification of encodings

- Many problems are highly symmetrical

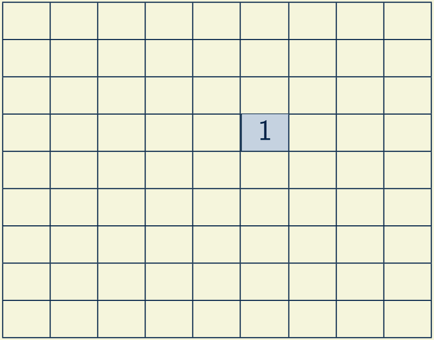
e.g Quasigroups:



Simplification of encodings

- Many problems are highly symmetrical

  e.g Quasigroups:



Simplification of encodings

- Many problems are highly symmetrical

e.g Quasigroups:

4	8	7	5	1	2	9	3	6

Simplification of encodings

- Many problems are highly symmetrical

  e.g Quasigroups:

		3						
		7						
		5						
		2						
		8						
		1						
		9						
		6						
		4						

Simplification of encodings

- Many problems are highly symmetrical

e.g Quasigroups:

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

Simplification of encodings

- Many problems are highly symmetrical

e.g Quasigroups:

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

Simplification of encodings

- Many problems are highly symmetrical

  e.g Quasigroups:

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

Simplification of encodings

- Many problems are highly symmetrical

e.g Quasigroups:

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

Simplification of encodings

- Many problems are highly symmetrical

e.g Quasigroups:

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

Simplification of encodings

- Many problems are highly symmetrical
  e.g Quasigroups:
- Breaking symmetries:
  - Change the search algorithm of the SAT solver?
  - Remodel the problem
  - Add symmetry breaking constraints
    e.g. Impose lexicographical order
  - Automated tools for finding symmetries:
    - **shatter** `http://www.aloul.net/Tools/shatter/`
- Other simplifications:
  - Formula simplification by preprocessing
    - **CP3** `http://tools.computational-logic.org/content/riss3g.php`

Incremental SAT solving

- Calling a SAT solver solver multiple times
- Changing the formula at each iteration
  - Adding new clauses is easy!
  - How to remove clauses?

Incremental SAT solving

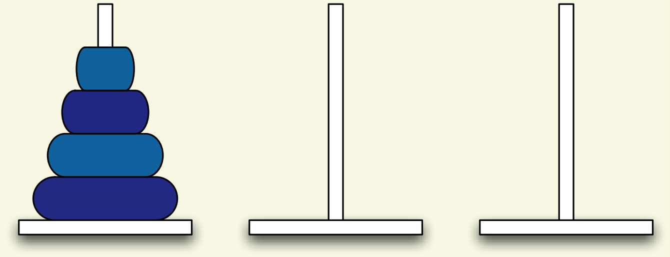
- Calling a SAT solver solver multiple times
- Changing the formula at each iteration
  - Adding new clauses is easy!
  - How to remove clauses?

- Use assumptions
- Add a fresh variables to clauses that you may want to remove:
  - $(a \vee b \vee f)$, where f is a fresh variable
  - Set f to 0 to consider the clause
  - Set f to 1 to remove the clause

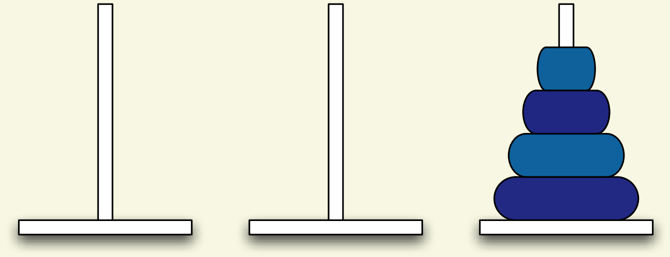
Other tips for encodings

- Tweaking solver parameters
  - Changing the set of decision variables
  - Bumping activity of more important variables
  - ...
  - Disclaimer: I would not recommend on doing this !
- Order of variable indexes
  - Close variables are usually related
- Solutions close to zero
  - SAT solvers usually branch on 0 first

Encoding a problem into SAT – Towers of Hanoi



Encoding a problem into SAT – Towers of Hanoi



- Only one disk may be moved at a time;
- No disk may be placed on the top of a smaller disk;
- Each move consists in taking the upper disk from one of the towers and sliding it onto the top of another tower.

How to encode ToH?

STRIPS planning mode:

- Variables
- Actions: preconditions $\rightarrow$ postconditions
- Initial state
- Goal state

How to encode ToH?

[Source: Selman & Kautz 1996]

- Variables: $on(d, dt, i); clear(dt, i)$
- Actions: $move(d, dt, dt, i) = obj(d, i) \wedge from(dt, i) \wedge to(dt, i)$
  - preconditions:
    $clear(d, i), clear(dt', i), on(d, dt, i)$
  - postconditions:
    $on(d, dt', i + 1), clear(dt, i + 1), \neg on(d, dt, i), \neg clear(dt', i + 1)$
- Initial state:
  - $on(d_1, d_2, 1), \ldots, on(d_{n-1}, d_n, 1), on(d_n, t_1, 1)$
    $clear(d_1, 1), clear(t_1, 1), clear(t_2, 1), clear(t_3, 1)$
  - All other variables initialized to false
- Goal state:
  - $on(d_1, d_2, 2^n - 1), \ldots, on(d_{n-1}, d_n, 2^n - 1), on(d_n, t_1, 2^n - 1)$

How to encode ToH?

[Source: Selman & Kautz 1996]

Constraints:

- Exactly one disk is moved at each time step
- There is exactly one movement at each time step
- There are no movements to exactly the same position
- For a movement to be done the preconditions must be satisfied
- After performing a movement the postconditions are implied
- No disks can be moved to the top of smaller disks
- Initial state holds at time step 0
- Goal state holds ate time step $2^n - 1$
- Preserve the value of variables that were unaffected by movements

How good is this encoding?

Time limit of 10,000 seconds using **picosat**

n	Selman
4	0.16
5	8.31
6	54.70
7	5252.27
8	-
9	-
10	-
11	-
12	-

A more compact encoding

[Source: Prestwich 2007]

- Actions: $move(d, dt, dt, i) = obj(d, i) \wedge from(dt, i) \wedge to(dt, i)$
  - Before:
    - Movements from disks/towers to disks/towers
  - Now:
    - Movements from towers to towers
    - Clear variable can be removed

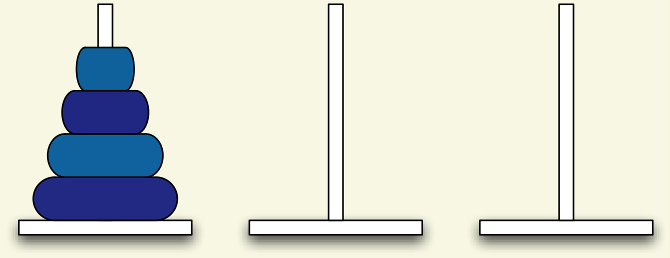
- More compact encoding:
  - Before: 5 towers requires 1,931 variables and 14,468 clauses
  - Now: 5 towers only requires 821 variables and 6,457 clauses

How good is this encoding?

n	Selman	Prestwich
4	0.16	0.01
5	8.31	0.08
6	54.70	0.47
7	5252.27	3.65
8	-	109.7
9	-	7126.57
10	-	-
11	-	-
12	-	-

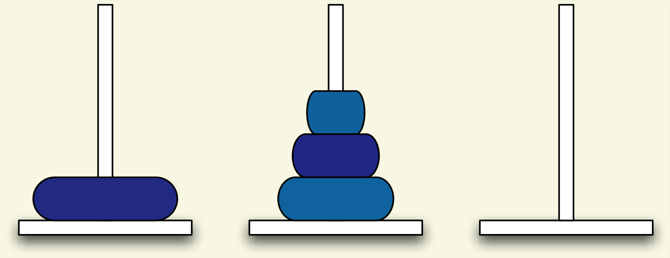
- Can we do better?
  - Look at the properties of the problem **!**

ToH Properties (Recursion)



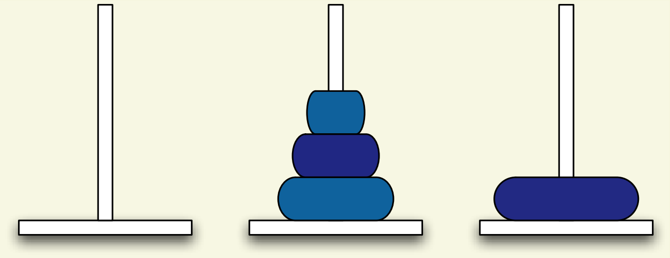
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n - 1$

ToH Properties (Recursion)



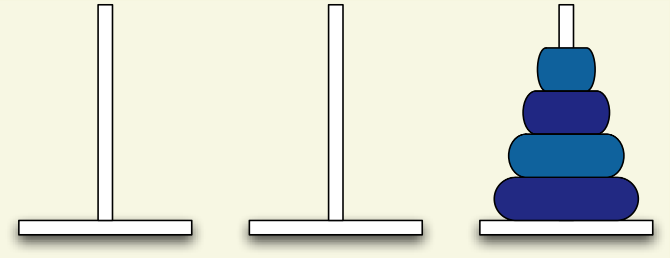
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n-1$

ToH Properties (Recursion)



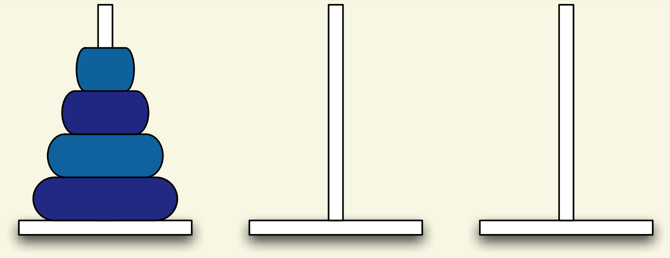
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n-1$
- The order of the disks to be moved after moving the largest disk is exactly the same as before

ToH Properties (Recursion)



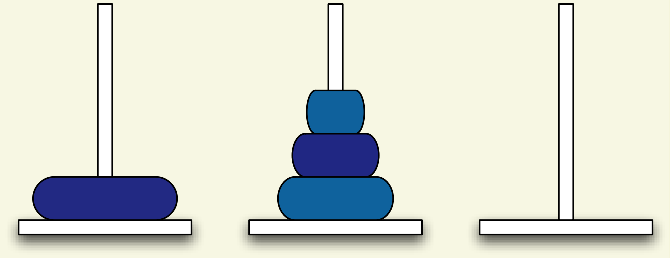
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n - 1$
- The order of the disks to be moved after moving the largest disk is exactly the same as before

ToH Properties (Symmetry)



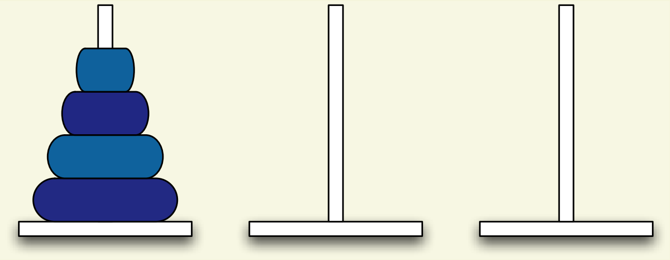
- ToH can be solved in $2^n - 1$ steps
- Considering the relationship between the movement of the disks after/before moving the largest disk we only need to determine the first $2^{n-1} - 1$ steps

ToH Properties (Symmetry)



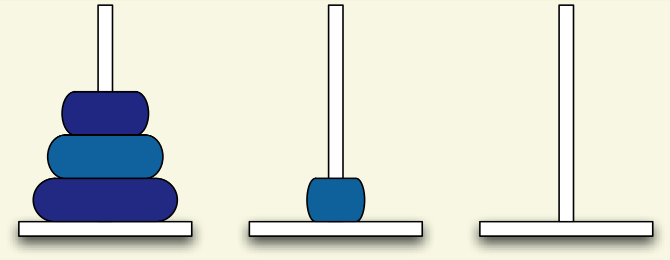
- ToH can be solved in $2^n - 1$ steps
- Considering the relationship between the movement of the disks after/before moving the largest disk we only need to determine the first $2^{n-1} - 1$ steps

ToH Properties (Parity)



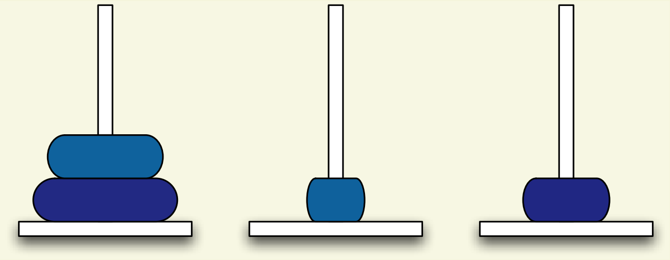
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



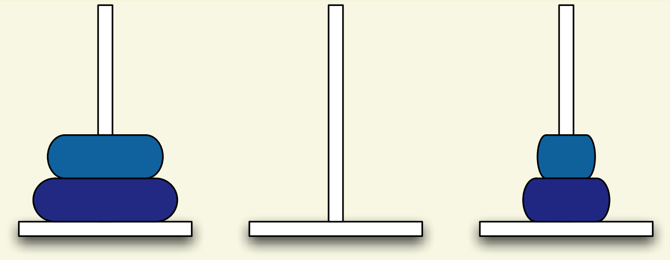
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



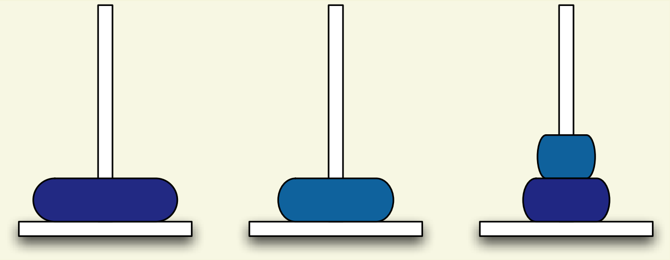
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



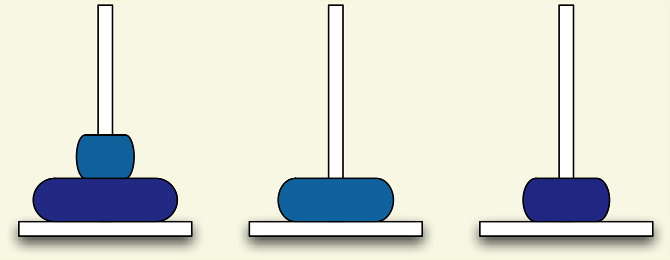
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



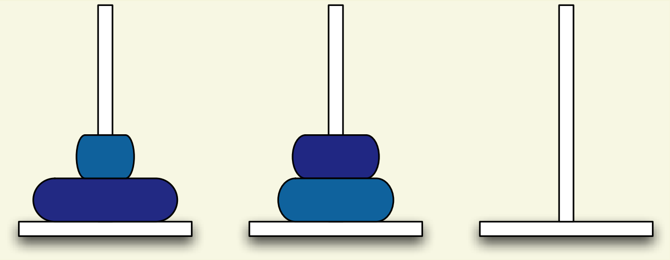
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



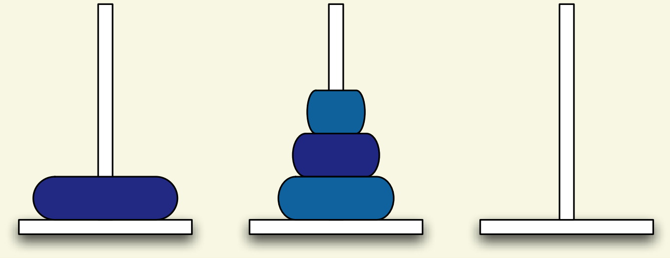
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



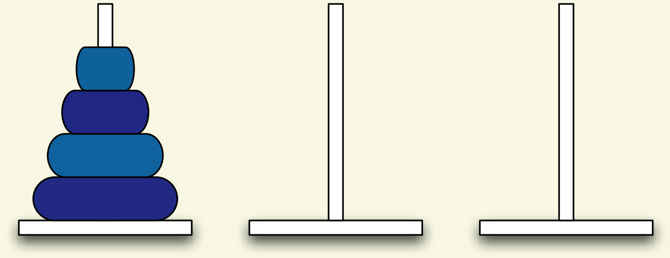
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



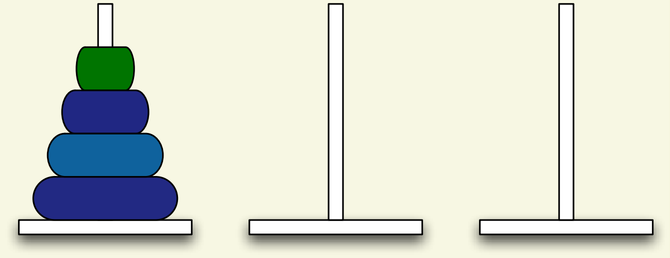
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Cycle)



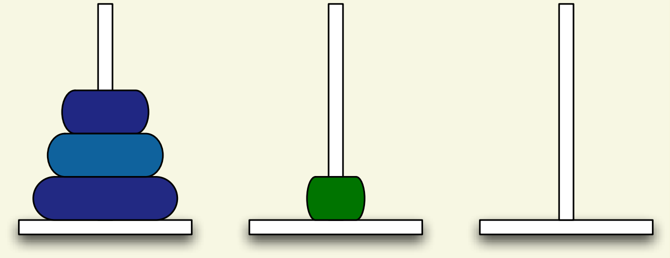
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



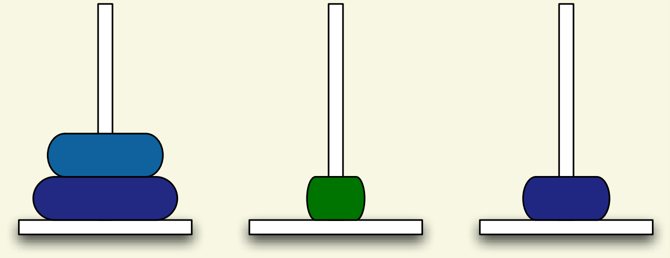
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



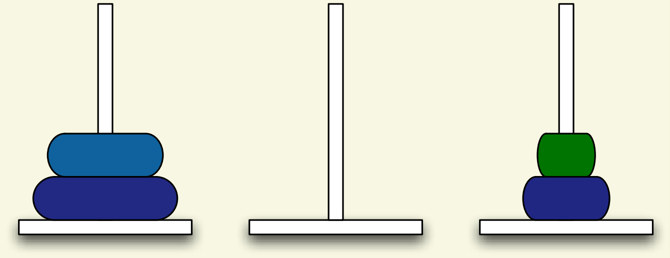
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise $(T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1)$ while the even disks will cycle counterclockwise $(T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1)$
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



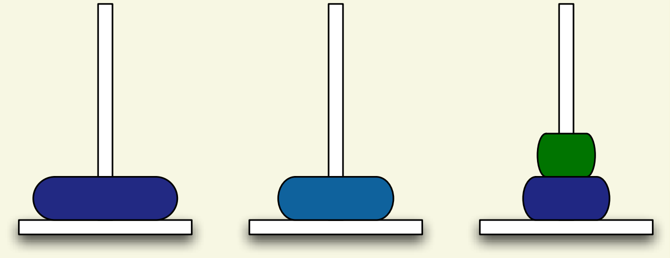
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise $(T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1)$ while the even disks will cycle counterclockwise $(T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1)$
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



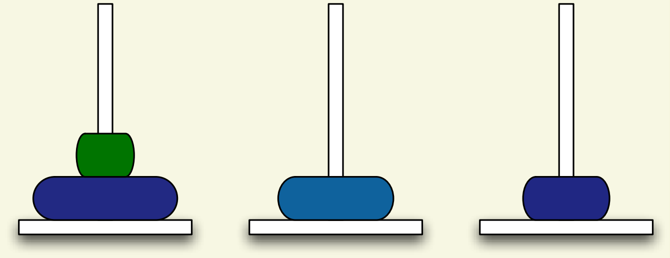
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise $(T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1)$ while the even disks will cycle counterclockwise $(T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1)$
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



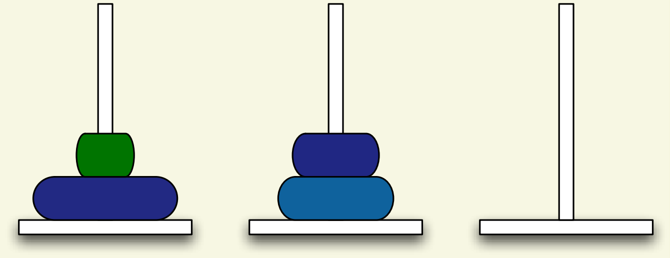
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise $(T_1 \to T_2 \to T_3 \to T_1)$ while the even disks will cycle counterclockwise $(T_1 \to T_3 \to T_2 \to T_1)$
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



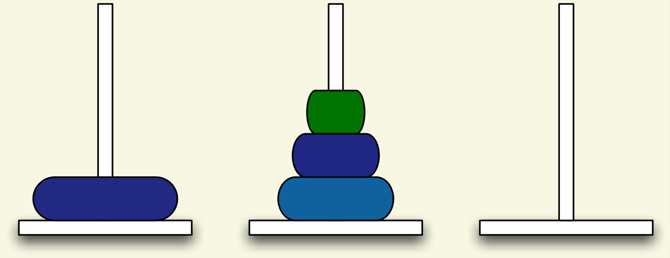
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle
4	0,16	0.01	0	0
5	8.31	0.08	0.01	0.02
6	54.70	0.47	0.03	0.05
7	5252.27	3.65	0.70	0.20
8	-	109.7	5.19	5.18
9	-	7126.57	79.11	7.65
10	-	-	1997.19	973.95
11	-	-	-	1206.37
12	-	-	-	-

- Disk Parity and Disk Cycle encodings use the symmetry property

Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle
4	0,16	0.01	0	0
5	8.31	0.08	0.01	0.02
6	54.70	0.47	0.03	0.05
7	5252.27	3.65	0.70	0.20
8	-	109.7	5.19	5.18
9	-	7126.57	79.11	7.65
10	-	-	1997.19	973.95
11	-	-	-	1206.37
12	-	-	-	-

- Disk Parity and Disk Cycle encodings use the symmetry property
- Can we still do better?

A new encoding for ToH

- The Disk Sequence encoding:
  - The recursive property determines the disks to be moved at each step
  - Taking into consideration this we can keep only the variables *on* and drop all the others
  - **Recursion+Symmetry+Parity**:
    - Problem can be solved with just unit propagation **!**

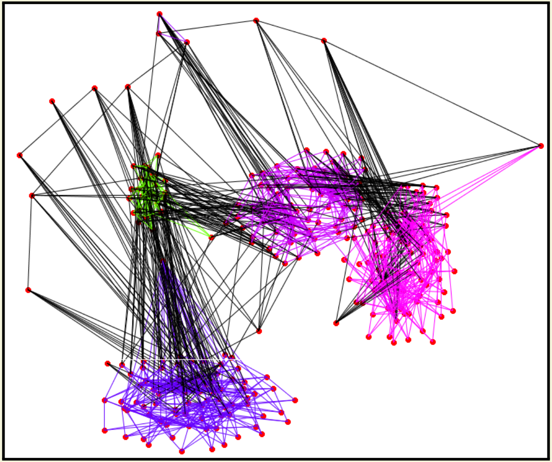
Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle	Disk Sequence
4	0.16	0.01	0	0	0
5	8.31	0.08	0.01	0.02	0
6	54.70	0.47	0.03	0.05	0
7	5252.27	3.65	0.70	0.20	0.01
8	-	109.7	5.19	5.18	0.03
9	-	7126.57	79.11	7.65	0.09
10	-	-	1997.19	973.95	0.23
11	-	-	-	1206.37	0.56
12	-	-	-	-	1.32

How is the structure of these formulas?

Selman encoding ($n = 3$)

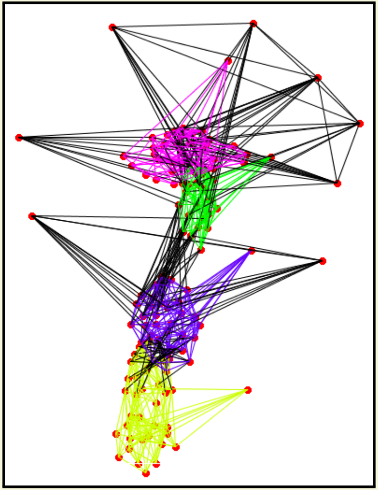
SATGraf– `https://ece.uwaterloo.ca/~vganesh/EvoGraph/Download.html`



How is the structure of these formulas?

Prestwich encoding ($n = 3$)

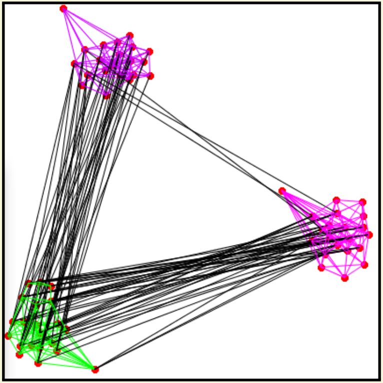
SATGraf– `https://ece.uwaterloo.ca/~vganesh/EvoGraph/Download.html`



How is the structure of these formulas?

Disk Sequence encoding ($n = 3$)

SATGraf– `https://ece.uwaterloo.ca/~vganesh/EvoGraph/Download.html`



Conclusions

- Encoding is an art !
  - Hard to evaluate which encoding is the best
  - Small encoding not necessarily means better one
- Each problem is unique !
  - Use your domain knowledge
  - Encode the properties of the problem
  - Break symmetries
- Automated tools ?
  - Can make your life easier
  - Not as good as handmade encodings