Computer science

— Theory vs Experimentation —

Christine Solnon

INSA de Lyon - 5IF

2023 / 2024

Computer science is a science...

...but not the science of computers!

(...) The topic became prematurely known as 'computer science' – which, actually, is like referring to surgery as 'knife science' – and it was firmly implanted in people's minds that computing science is about machines and their peripheral equipment. Quod non. *Dijkstra (1986)*

What does it mean to be a science (according to Wikipedia)?

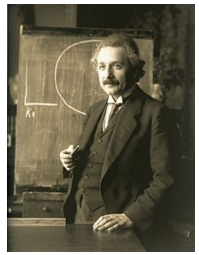
Science is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe.
A scientific process is usually composed of the following steps:

- *Formulate a question*
- *Make conjectures/hypothesis that may answer the question*
- *Use these hypothesis to predict consequences that can be tested*
- *Test hypothesis by conducting experiments*
- *Analyse results to support or falsify hypothesis... and publish!?*

Theory versus Experimentation

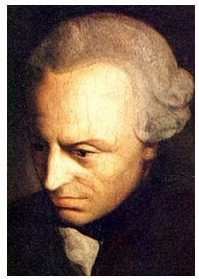
In theory, theory and practice are the same.
In practice, they are not.

(A. Einstein)



Experience without theory is blind,
but theory without experience is mere intellectual play.

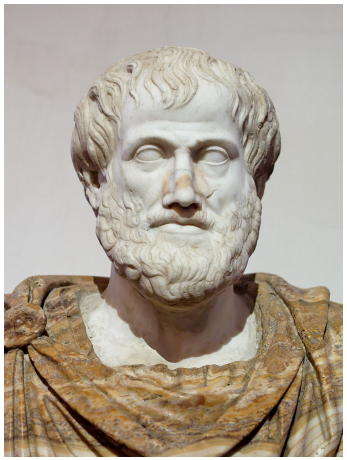
(I. Kant)



If you find that you're spending almost all your time on theory, start turning some attention to practical things; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.

(D. Knuth)

A young science with old foundations

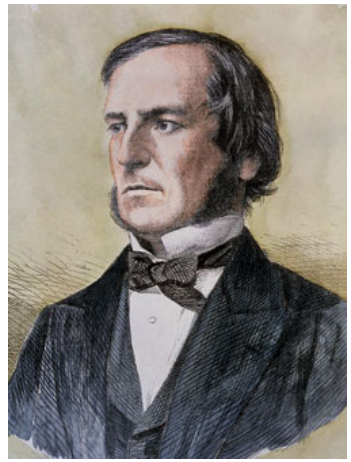


Aristote
(384–322 BC)

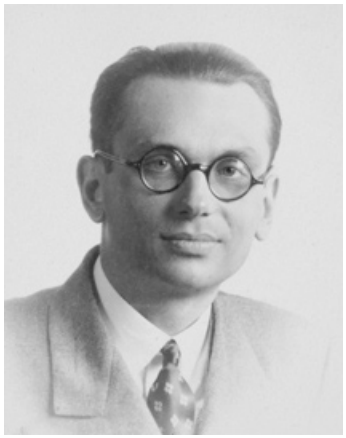
Euclide
(∼ 300 BC)

Blaise Pascal
(1623-1662)

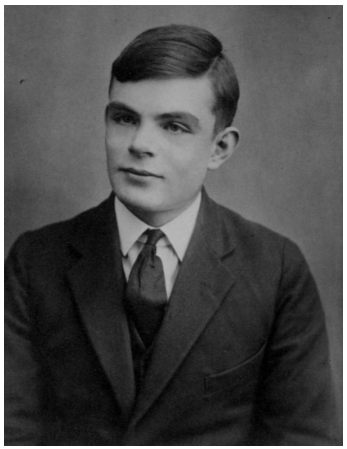
Ada Lovelace
(1815-1852)



George Boole
(1815-1864)



Kurt Gödel
(1906-1978)



Alan Turing
(1912-1954)

J. von Neumann
(1903-1957)

Grace Hopper
(1906-1992)

Donald Knuth
(1938-?)

Plan

Problems, Instances and Algorithms (Recalls)

Specification of a problem:

- Input parameters
- Output parameters
- Optionally: Preconditions on input parameters
- Postrelation between input and output parameter values

Instance of a problem:

Values of input parameters which satisfy preconditions

Algorithm for a problem:

Sequence of elementary instructions to compute output parameter values from input parameter values, for any instance of the problem

Example 1: Searching for a value in a sorted array

Specification of the problem:

- Input: an array a of n integers (indexed from 0 to $n-1$) and an integer e
- Output: an integer i
- Precondition: $\forall j \in [0, n-2], a[j] \leq a[j+1]$
- Postrelation:
  - if $\forall j \in [0, n-1], a[j] \neq e$ then $i = n$
  - else $i \in [0, n-1]$ and $a[i] = e$

Examples of instances:

- Input: $e = 8$ and $a =$

4	4	7	8	10	11	12

  $\leadsto$ Output: $i = 3$
- Input: $e = 9$ and $a =$

4	4	7	8	10	11	12

  $\leadsto$ Output: $i = 7$

Example 2: Sorting an array

Specification of the problem:

- Input: an array a^- of n integers
- Output: an array a^+ of n integers
- Postrelation:
  - a^+ is a permutation of a^-
  - $\forall i \in [0, n-2], a^+[i] \leq a^+[i+1]$

Examples of instances:

- Input: $a^- =$ | 4 | 2 | 9 | 4 | 0 | 7 | 1 |

  $\leadsto$ Output: $a^+ =$ | 0 | 1 | 2 | 4 | 4 | 7 | 9 |
- Input: $a^- =$ | 4 | 7 | 7 | 4 |

  $\leadsto$ Output: $a^+ =$ | 4 | 4 | 7 | 7 |

Example 3: Satisfiability of a Boolean Formula (SAT)

Specification of the problem:

- Input: a Boolean formula F defined on a set X of n variables
- Output: a Boolean value V
- Precondition: F is in Conjunctive Normal Form (CNF)
- Postrelation: $V = \mathit{true}$ iff F can be satisfied

Examples of instances:

- $X = \{a, b, c\}$ and $F = (a \vee \overline{b}) \wedge (b \vee c) \wedge (\overline{a} \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee c)$
  $\rightsquigarrow V = \mathit{true}$
- $X = \{a, b, c\}$ and $F = (a \vee \overline{b}) \wedge (b \vee c) \wedge (\overline{a} \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (a \vee b \vee \overline{c})$
  $\rightsquigarrow V = \mathit{false}$

Reference:

Knuth: The Art of Computer Programming, Vol. 4B, 2022

Complexity of an algorithm

Estimation of resources required by an algorithm:

- Time $\rightsquigarrow$ Number of elementary instructions
- Space $\rightsquigarrow$ Memory consumption

The estimation depends on the size of input param. and is a growth rate:
How do time or space requirements grow as the input size grows?

Growth of a function $f(n)$:

- $\mathcal{O}(g(n))$: $\exists c, n_0 \in \mathbb{R}^+$ such that $\forall n > n_0, |f(n)| \leq c.|g(n)|$
- $\Omega(g(n))$: $\exists c, n_0 \in \mathbb{R}^+$ such that $\forall n > n_0, |f(n)| \geq c.|g(n)|$
- $\Theta(g(n))$: $\exists c_1, c_2, n_0 \in \mathbb{R}^+$ such that $\forall n > n_0, c_1.|g(n)| \leq |f(n)| \leq c_2.|g(n)|$

Quiz: Complexity of some sorting algorithms

Specification of the problem (recall):

- Input: an array a^- of n integers
- Output: an array a^+ of n integers
- Postrelation: a^+ is a permutation of a^- and a^+ is sorted

Complexity of Selection Sort?

```
1 begin
2     for i ranging from 0 to n − 2 do
          /* Invariant:  ∀j ∈ [0, i − 2], a[j] ≤ a[j + 1]   */
          /*             ∀j ∈ [i, n − 1], a[i − 1] ≤ a[j]   */
3         search for the index s of the smallest value in a[i..n − 1]
4         exchange a[s] and a[i]
```

Quiz: Complexity of some sorting algorithms

Specification of the problem (recall):

- Input: an array a^- of n integers
- Output: an array a^+ of n integers
- Postrelation: a^+ is a permutation of a^- and a^+ is sorted

Complexity of Insertion Sort?

```
1 begin
2     for i ranging from 1 to n − 1 do
          /* Invariant:  ∀j ∈ [0, i − 2], a[j] ≤ a[j + 1]   */
3         insert a[i] in a[0..i − 1]
```

Quiz: Complexity of some sorting algorithms

Specification of the problem (recall):

- Input: an array a^- of n integers
- Output: an array a^+ of n integers
- Postrelation: a^+ is a permutation of a^- and a^+ is sorted

Complexity of Quick Sort?

```
1 begin
2     if n > 1 then
3         Choose a pivot value in a[0..n − 1]
4         Find i and permut values of a[0..n − 1] so that:
5         → ∀j ∈ [0, i − 1], a[j] ≤ pivot
6         → a[i] = pivot
7         → ∀j ∈ [i + 1, n − 1], a[j] > pivot
8         Recursively sort a[0..i − 1]
9         Recursively sort a[i + 1..n − 1]
```

Quiz: Complexity of some sorting algorithms

Specification of the problem (recall):

- Input: an array a^- of n integers
- Output: an array a^+ of n integers
- Postrelation: a^+ is a permutation of a^- and a^+ is sorted

Complexity of Counting Sort?

```
1  begin
2      Let min and max be the smallest and largest values of a⁻, respectively
3      Initialise to 0 an array nbOcc indexed from min to max
4      for i ranging from 0 to n − 1 do
5          nbOcc[a[i]] ← nbOcc[a[i]] + 1
6      i ← 0
7      for v ranging from min to max do
8          for k ranging from 1 to nbOcc[v] do
9              a[i] ← v
10             i ← i + 1
```

Some Pitfalls of Algorithm Complexity

When complexity depends on the considered instance

Examples: Insertion sort, Simplex

- Worst-case/Best-case complexity
  $\rightsquigarrow$ Identify the worst/best possible instance
- Average-case complexity
  $\rightsquigarrow$ Depends on a probability distribution of input values

When complexity depends on input values

Examples: Counting sort, Dynamic programming for the knapsack problem

- Pseudo-polynomial complexity

When the output has an exponential size wrt the input

Example: Frequent itemset mining in a database

- Complexity of the step between two consecutive outputs

Plan

Complexity of Problems

Complexity of a problem = Complexity of its best algorithm

But this best algorithm may not (yet) be known!

How can we compute the complexity of a problem X?

- Each algorithm for X gives an upper bound
- Lower bounds may be found by analysing the problem

The complexity of X is known if largest lower bound = smallest upper bound
Otherwise the complexity is open...

Examples of complexity lower bounds

Output size

- Give a trivial lower bound for:
  - Generating all permutations of n values
  - Multiplying two $n \times n$ matrices
- Compare with the complexity of the best known algorithm

Can we use the input size as a lower bound?

Bounds computed by reasoning on decision trees

Ex: Sort of an array of n values (for binary comparison-based sorts)

- There exist $n!$ different permutations of the array
  and the algorithm must be able to compute each of them
- If the sort is based on the comparison of couples of values,
  then each comparison can eliminate at most half of the permutations
- Hence, we need at least $\log_2(n!)$ comparisons $\rightsquigarrow \Omega(n \log n)$

Examples of complexity lower bounds

Output size

- Give a trivial lower bound for:
  - Generating all permutations of n values
  - Multiplying two $n \times n$ matrices
- Compare with the complexity of the best known algorithm

Can we use the input size as a lower bound?

Bounds computed by reasoning on decision trees

Ex: Sort of an array of n values (for binary comparison-based sorts)

- There exist $n!$ different permutations of the array
  and the algorithm must be able to compute each of them
- If the sort is based on the comparison of couples of values,
  then each comparison can eliminate at most half of the permutations
- Hence, we need at least $\log_2(n!)$ comparisons $\rightsquigarrow \Omega(n \log n)$

Examples of complexity lower bounds

Output size

- Give a trivial lower bound for:
  - Generating all permutations of n values
  - Multiplying two $n \times n$ matrices
- Compare with the complexity of the best known algorithm

Can we use the input size as a lower bound?

Bounds computed by reasoning on decision trees

Ex: Sort of an array of n values (for binary comparison-based sorts)

- There exist $n!$ different permutations of the array
  and the algorithm must be able to compute each of them
- If the sort is based on the comparison of couples of values,
  then each comparison can eliminate at most half of the permutations
- Hence, we need at least $\log_2(n!)$ comparisons $\rightsquigarrow \Omega(n \log n)$

Decision Problems

What is a Decision Problem?

- Output of the problem = `yes` or `no`
- Postrelation = Binary question on input parameters

Example: Description of the *Search* Decision Problem

- Input = an array a of n values and a value e
- Question = Does a contain e?

Many complexity classes are defined for decision problems only

Class $\mathcal{P}$

$\mathcal{P}$ is the class of decision problems with polynomial complexities

$\rightsquigarrow$ $\mathcal{P}$ is the class of tractable problems

Examples of decision problems in $\mathcal{P}$

- Decide if a value belongs to an array
- Decide if there exists a path between two vertices in a graph
- Decide if there exists a path of bounded cost between two vertices
- Decide if there exists a spanning tree of bounded cost in a graph
- ...
- Decide if a given value is a prime number
  $\rightsquigarrow$ Prime is in $\mathcal{P}$ [Agrawal - Kayal - Saxena 2002]!

Class $\mathcal{NP}$

Condition for a decision problem X to belong to class $\mathcal{NP}$:

- $X \in \mathcal{NP} \Rightarrow \exists$ Polynomial algo. for a Non deterministic Turing machine
- In other words: $X \in \mathcal{NP}$ if, for every instance I of X such that answer(I) = yes, there exists a certificate $c(I)$ which allows one to check in polynomial time that answer(I) = yes

  $\rightsquigarrow$ **The problem of deciding if a certificate is solution belongs to $\mathcal{P}$**

Example: SAT $\in \mathcal{NP}$

- Description of SAT (recall):
  - Input = a Boolean formula F over a set X of n Boolean variables
  - Question = Does there exist a valuation of X which satisfies F?
- Certificate = an assignment $A : X \rightarrow \{true, false\}$

  $\rightsquigarrow$ Deciding whether A satisfies F or not is a polynomial problem

Relation between $\mathcal{P}$ and $\mathcal{NP}$:

- $\mathcal{P} \subseteq \mathcal{NP}$
- Conjecture: $\mathcal{P} \neq \mathcal{NP}$

  1 million dollar prize to win!

  See [Millenium Problems of the Clay Mathematics Institute]

$\mathcal{NP}$-complete Problems:

- Hardest problems of $\mathcal{NP}$:
  $\rightsquigarrow$ X is $\mathcal{NP}$-complete if $(X \in \mathcal{NP})$ and $(X \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP})$
- Theorem of [Cook 1971]: SAT is $\mathcal{NP}$-complete
- Since 1971, hundreds of problems have been shown $\mathcal{NP}$-complete
  See [Karp 1972], [Garey and Johnson 1979], [Wikipedia dynamic list]

For more information on $\mathcal{P}$ vs. $\mathcal{NP}$:

Fortnow: Fifty Years of P vs. NP and the Possibility of the Impossible, 2022

How to prove that a problem P is $\mathcal{NP}$-complete?

Step 1: Show that P belongs to $\mathcal{NP}$

- Define a polynomial-size certificate
- Define a polynomial-time procedure to decide whether a certificate is a solution or not

Step 2: Reduce a known $\mathcal{NP}$-complete problem P' to P

- Find a polynomial-time algorithm A_r to reduce P' to P:
  Given an instance I' of P', A_r returns an instance I of P such that the answer of I' for P' is equal to the answer of I for P
- Use A_r to define a lower bound on the complexity of P':
  Complexity of $P' \leq$ Complexity of A_r + Complexity of P
- If P belongs to $\mathcal{P}$, then P' also belongs to $\mathcal{P}$...
  ... and you have shown that $\mathcal{P} = \mathcal{NP}$

Exercise 1

Description of the Clique Problem:

- Input: a graph $G = (V, E)$ and a positive integer k
- Question: Does there exist $S \subseteq V$ such that $\#S = k$ and $\forall i, j \in S, i \neq j \Rightarrow (i, j) \in E$?

Show that Clique $\in \mathcal{NP}$:

$\rightsquigarrow$ Certificate?

Show that Clique is $\mathcal{NP}$-complete:

$\rightsquigarrow$ Reduction from SAT to Clique:

- Give a polynomial algorithm to solve the reduction problem:
  - Input: an instance of SAT = a CNF formula F
  - Output: an instance of Clique = a graph G and an integer k
  - Postrelation: F is satisfiable $\Leftrightarrow$ G contains a clique of order k

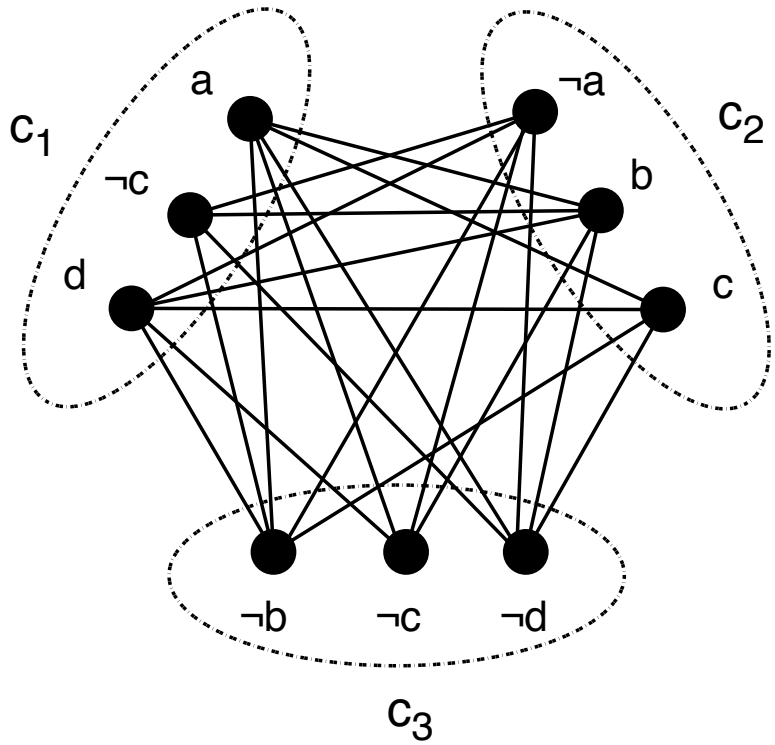
Solution to Exercise 1

Reduction from SAT to Clique:

- Define the graph $G_F = (V, E)$ associated with a formula F :
  - V associates a vertex with every literal of every clause of F
    $\rightsquigarrow$ $c(u)$ and $l(u)$ = clause and literal associated with vertex u
  - $E = \{\{u, v\} \subseteq V \mid c(u) \neq c(v) \text{ and } l(u) \neq \neg l(v)\}$
- Define k = Number of clauses in F

Example:

Formula F :
$(a \vee \neg c \vee d) \wedge$
$(\neg a \vee b \vee c) \wedge$
$(\neg b \vee \neg c \vee \neg d)$



Solution to Exercise 1

Reduction from SAT to Clique:

- Define the graph $G_F = (V, E)$ associated with a formula F :
  - V associates a vertex with every literal of every clause of F $\rightsquigarrow$ $c(u)$ and $l(u)$ = clause and literal associated with vertex u
  - $E = \{\{u, v\} \subseteq V \mid c(u) \neq c(v) \text{ and } l(u) \neq \neg l(v)\}$
- Define k = Number of clauses in F

Homework: Demonstrate that

- If F is satisfiable, then G_F contains a clique of k vertices
- If G_F contains a clique of k vertices, then F is satisfiable

Conclusion:

- Clique is at least as hard as SAT because if Clique can be solved in polynomial time, then SAT can also be solved in polynomial time
- As Clique belongs to $\mathcal{NP}$, Clique is $\mathcal{NP}$-complete

Exercise 2

Description of the Subset Sum Problem (SSP):

- Input: a set S of n integers
- Question: Does there exist $X \subseteq S$ such that $X \neq \emptyset$ and $\sum_{i \in X} i = 0$?

Show that SSP $\in \mathcal{NP}$:

$\rightsquigarrow$ Certificate?

Show that SSP is $\mathcal{NP}$-complete:

$\rightsquigarrow$ Reduction from 3-SAT to SSP:

- Give a polynomial algorithm for the reduction problem:
  - Input: a 3-SAT instance (each clause contains 3 literals)
  - Output: an SSP instance = a set S of integers
  - Postrelation: F is satisfiable $\Leftrightarrow \exists X \subseteq S, \sum_{i \in X} i = 0$

Ex.: $F = (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg d) \wedge (a \vee \neg b \vee d)$

	a	b	c	d	c_1	c_2	c_3
i_a	1	0	0	0	0	0	1
$i_{\neg a}$	1	0	0	0	1	0	0
i_b		1	0	0	0	0	0
$i_{\neg b}$		1	0	0	0	1	1
i_c			1	0	1	0	0
$i_{\neg c}$			1	0	0	1	0
i_d				1	0	0	1
$i_{\neg d}$				1	1	1	0
$i^1_{c_1}$					1	0	0
$i^2_{c_1}$					2	0	0
$i^1_{c_2}$						1	0
$i^2_{c_2}$						2	0
$i^1_{c_3}$							1
$i^2_{c_3}$							2
t	-1	1	1	1	4	4	4

$$\{\neg a, \neg b, c, \neg d\}$$

$$\Leftrightarrow$$

$$\begin{array}{rr} & 1000100 \\ + & 100011 \\ + & 10100 \\ + & 1110 \\ + & 100 \\ + & 20 \\ + & 1 \\ + & 2 \\ - & 1111444 \\ \hline = & 0 \end{array}$$

Algorithm to solve SSP

Input: n integers $x_1, x_2, \ldots, x_n$
Output: true if there exists $X \subseteq \{x_1, \ldots, x_n\}$ s.t. $X \neq \emptyset$ and $(\sum_{k \in X} k) = 0$

```
/* Let min be the sum of all negative values in {x_1,...,x_n}                    */
/* Let max be the sum of all positive values in {x_1,...,x_n}                    */
/* Let a[1..n][min..max] be a Boolean array:                                     */
/*      a[i][v]=true iff ∃X ⊆ {x_1,...,x_i} such that X ≠ ∅ and (Σ_{k∈X} k) = v  */
1 begin
2     Initialise all elements of a to false
3     a[1][x_1] ← true
4     for i ranging from 2 to n do
5         for v ranging from min to max do
6             a[i][v] ← a[i − 1][v] || (x_i == v) || a[i − 1][v − x_i]
7     return a[n][0]
```

Compute the complexity of this algorithm:

Have we shown that $\mathcal{P} = \mathcal{NP}$?

Algorithm to solve SSP

Input: n integers $x_1, x_2, \ldots, x_n$
Output: true if there exists $X \subseteq \{x_1, \ldots, x_n\}$ s.t. $X \neq \emptyset$ and $(\sum_{k \in X} k) = 0$

```
/* Let min be the sum of all negative values in {x_1,...,x_n}                    */
/* Let max be the sum of all positive values in {x_1,...,x_n}                    */
/* Let a[1..n][min..max] be a Boolean array:                                     */
/*      a[i][v]=true iff ∃X ⊆ {x_1,...,x_i} such that X ≠ ∅ and (Σ_{k∈X} k) = v  */
1 begin
2     Initialise all elements of a to false
3     a[1][x_1] ← true
4     for i ranging from 2 to n do
5         for v ranging from min to max do
6             a[i][v] ← a[i − 1][v] || (x_i == v) || a[i − 1][v − x_i]
7     return a[n][0]
```

Compute the complexity of this algorithm:

Have we shown that $\mathcal{P} = \mathcal{NP}$?
No: The algorithm is Pseudo-polynomial!

Parameterized Complexity

Strongly $\mathcal{NP}$-complete Problems:

Problems which are still $\mathcal{NP}$-complete even when all numerical input values are bounded by a polynome with respect to the input size

Parameterized Problems:

Problems for which an input parameter k has a fixed value

Example: Parameterized SSP

- Input: a set S of n integers
- Parameter: $k = max - min$ with $min = \sum_{i \in S, i<0} i$ and $max = \sum_{i \in S, i>0} i$
- Question: Does there exist $X \subseteq S$ such that $X \neq \emptyset$ and $(\sum_{i \in X} i) = 0$?

Fixed Parameter Tractable (FPT) Problems:

Problems for which there exists an algorithm in $\mathcal{O}(f(k) \cdot n^c)$ where n = input size, k = fixed parameter and c = constant independent from input values

Example of FPT Problem: Vertex Cover

Definition of Vertex Cover

- Input: a non directed graph $G = (V, E)$ and an integer k
- Question: does there exist $S \subseteq V$ such that $\#S \leq k$ and $\forall (i, j) \in E, i \in S \vee j \in S$?

Kernelisation of *Vertex Cover*

Transform G by iteratively applying the following rules:

- If $k > 0$ and $\exists v \in V, d^\circ(v) > k$: remove v from G and decrement k
- If $\exists v \in V, d^\circ(v) = 0$: remove v from G

Let $G' = (V', E')$ be the resulting graph (s.t. $\forall v \in V', 1 \leq d^\circ(v) \leq k$)

- If $\#E' > k^2$, then answer no
  (each vertex covers at most k edges $\Rightarrow$ S covers at most k^2 edges)
- Else, G' is a kernel: answer(G) = answer(G'), $\#E' \leq k^2$ and $\#V' \leq 2k^2$
  $\rightsquigarrow$ Brute force algorithm in $\mathcal{O}(2^{2k^2})$ + transformation in $\mathcal{O}(\#V + \#E)$

Conclusion: Vertex Cover is FPT for the parameter k

$\mathcal{NP}$-intermediate Problems

Ladner Theorem:

If $\mathcal{P} \neq \mathcal{NP}$ then there exist problems of $\mathcal{NP}$ which are $\mathcal{NP}$-intermediate $\rightsquigarrow$ neither in $\mathcal{P}$ nor $\mathcal{NP}$-complete

Problems of $\mathcal{NP}$ not known to be $\mathcal{NP}$-complete nor to belong to $\mathcal{P}$:

- Graph isomorphism
  but quasi-polynomial algorithm proposed in [Babai 2016]
- Factoring of integers
- Rotation distance between binary trees
- ...

$\mathcal{NP}$-hard Problems (1/2)

Problems at least as hard as those in $\mathcal{NP}$:

- X is $\mathcal{NP}$-hard if: $X \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP}$
  $\rightsquigarrow$ Checking that a certificate is a solution of X may not belong to $\mathcal{P}$
- $\mathcal{NP}$-complete $\subset \mathcal{NP}$-hard

Example: k^{th} Subset Problem

- Input: a set S of n integers and 2 integers k and l
- Question: Does there exist k distinct subsets of S such that, for each subset, the sum of its elements is greater than l?

Does this problem belong to $\mathcal{NP}$?

Example 2: Exact Clique Problem

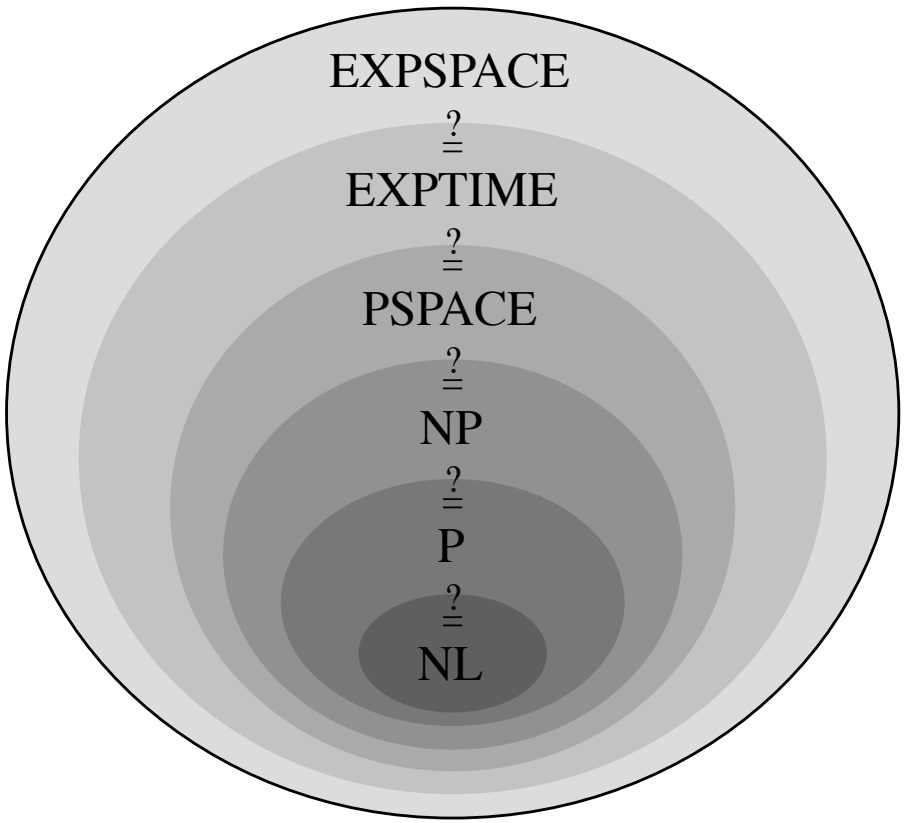
- Input: a graph $G = (V, E)$ and a positive integer k
- Question: Is the largest clique of G of order k?

Does this problem belong to $\mathcal{NP}$?

$\mathcal{NP}$-hard Problems (2/2)

Classification of $\mathcal{NP}$-hard Problems

- $\mathcal{P}$-SPACE: There exists a polynomial-space complexity algorithm
- $\mathcal{EXP}$-TIME: There exists an exponential-time complexity algorithm
- $\mathcal{EXP}$-SPACE: There exists an exponential-space complexity algorithm



The co-$\mathcal{NP}$ Class

Complementary problem $\overline{X}$ of a decision problem X

- Input of $\overline{X}$ = Input of X
- Question of $\overline{X}$ = Negation of the question of X

Example 1: $\overline{\text{SAT}}$ = Complementary of SAT

- Input: A set of variables X and a Boolean formula F
- Question: Is F inconsistent?

Example 2: $\overline{\text{Clique}}$ = Complementary of Clique

- Input: A graph $G = (V, E)$ and a positive integer k
- Question: Is the order of every clique of G smaller than k?

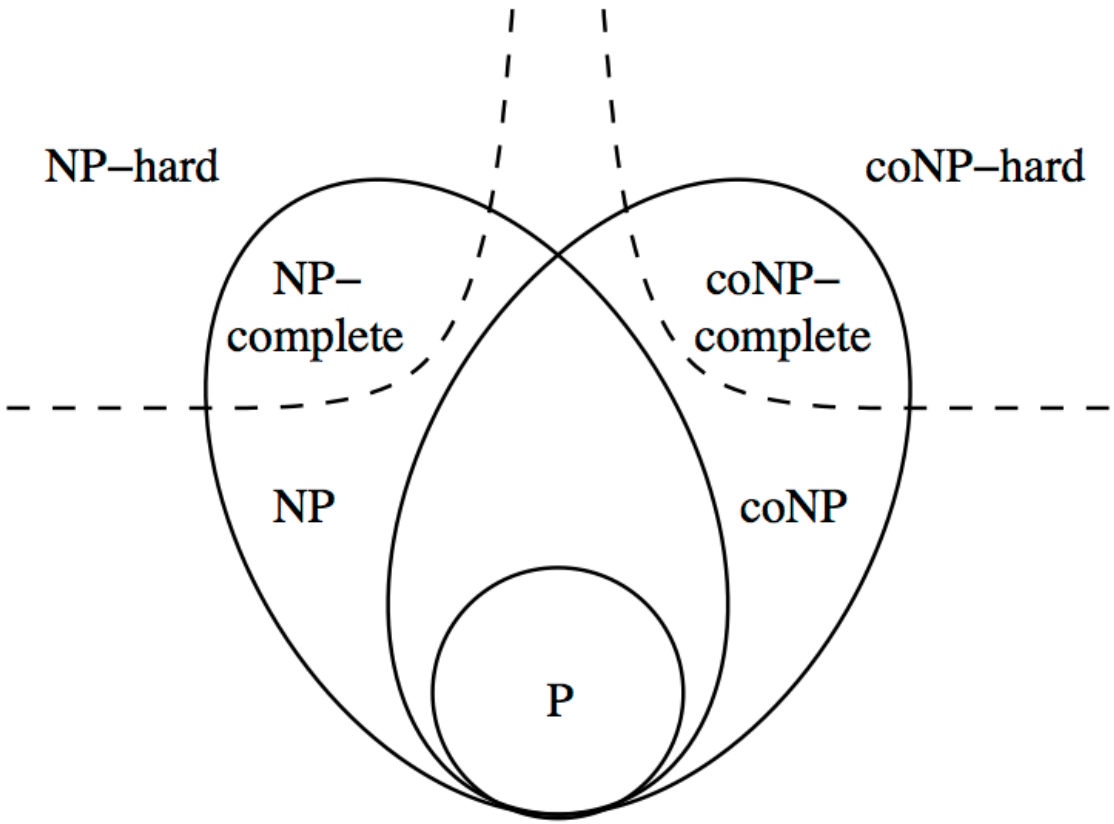
Co-$\mathcal{NP}$ Class:

$\overline{P} \in$ co-$\mathcal{NP}$ iff $P \in \mathcal{NP}$
Examples: $\overline{\text{SAT}} \in$ co-$\mathcal{NP}$ and $\overline{\text{Clique}} \in$ co-$\mathcal{NP}$

Relations between $\mathcal{NP}$ and co-$\mathcal{NP}$

Theorems:

- If $A \in \mathcal{P}$ then $\overline{A} \in \mathcal{P}$
- If there exists A st A is $\mathcal{NP}$-complete and $\overline{A} \in \mathcal{NP}$ then $\mathcal{NP} =$ co-$\mathcal{NP}$
- If A is $\mathcal{NP}$-complete then $\overline{A}$ is co-$\mathcal{NP}$-complete



Plan

Why Matching Graphs?

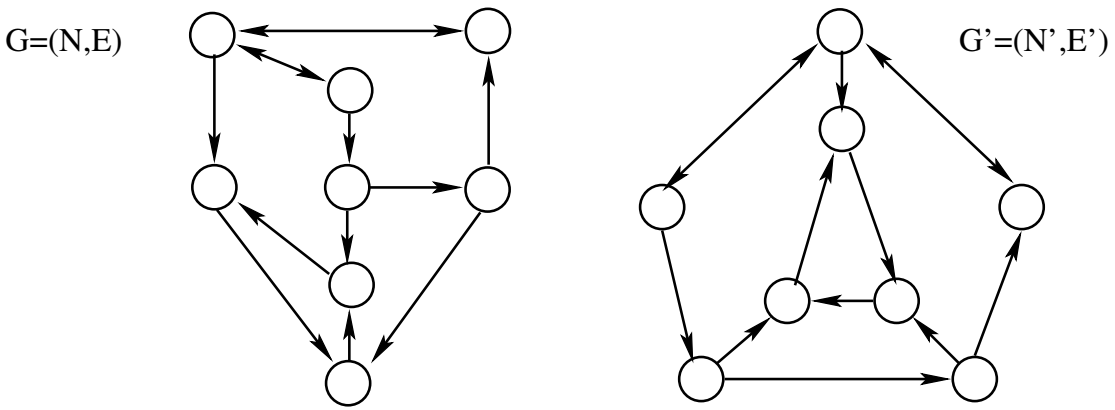
Graphs are used to model:

- Images
- 3D Objects
- Molecules
- Interaction networks
- Social networks
- Ontologies
- Documents (XML)
- Resources on the Web (RDF)
- ...

To compare graphs, we may match their vertices and edges

Graph Matching Problems

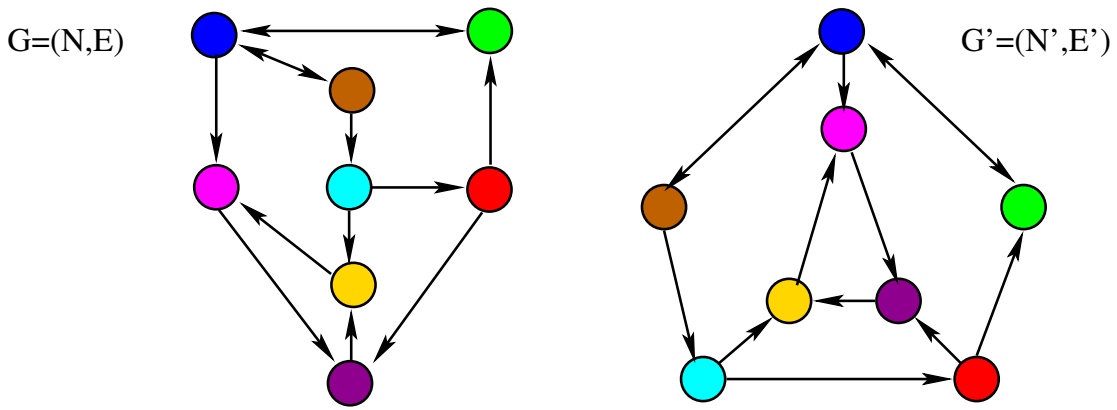
- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost



$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2))$

Graph Matching Problems

- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost



$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$

$\rightsquigarrow$ Bijection $f : V_1 \to V_2$ such that $\forall i, j \in V_1 : (i, j) \in E_1 \Leftrightarrow (f(i), f(j)) \in E_2$

$\rightsquigarrow$ Complexity in the general case: Conjectured $\mathcal{NP}$-intermediary
[Babai 2016]: Quasi-polynomial algorithm ... to be continued!?

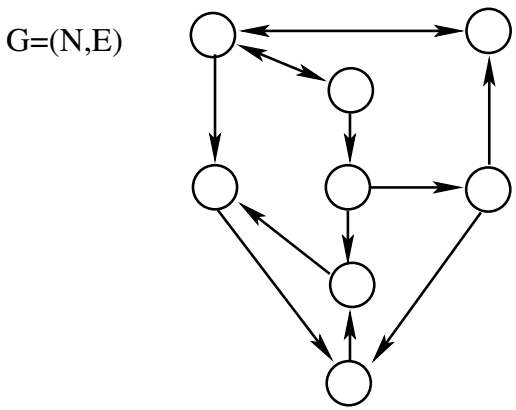
$\rightsquigarrow$ Particular cases:

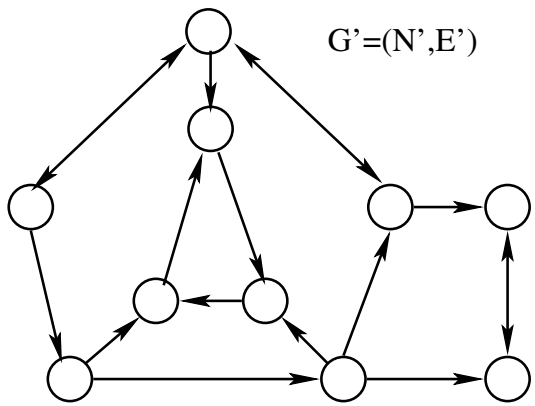
- Polynomial algorithm: Planar graphs, Ordered graphs, ...
- FPT algorithm: Bounded degree, Bounded treewidth, ...

Graph Matching Problems

- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost

$G_p = (V_p, E_p)$

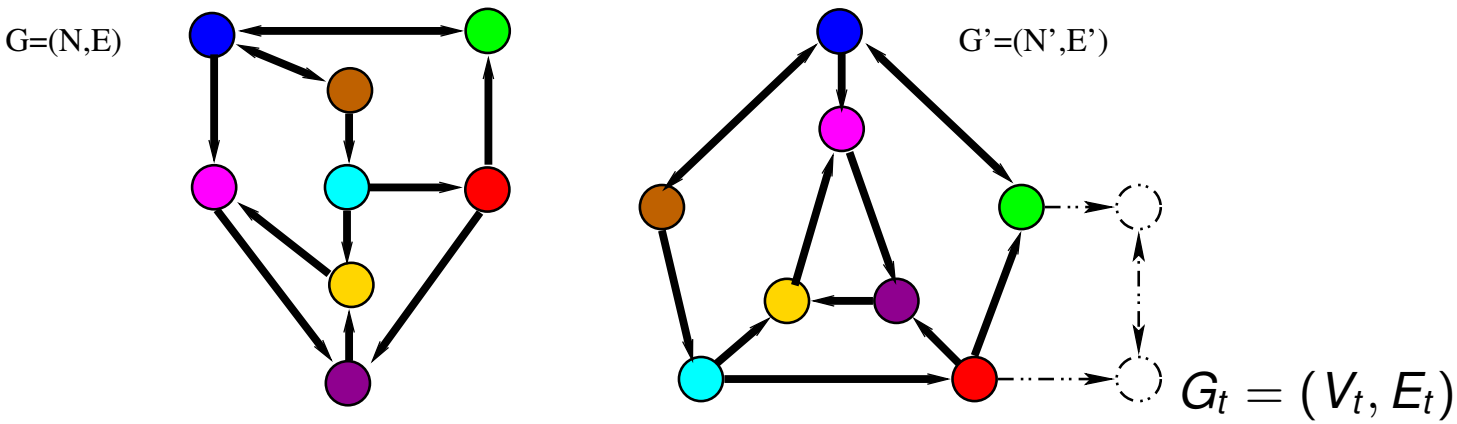




$G_t = (V_t, E_t)$

Graph Matching Problems

- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

$\rightsquigarrow$ Injection $f : V_p \rightarrow V_t$ such that
- Induced case: $\forall i, j \in V_p : (i, j) \in E_p \Leftrightarrow (f(i), f(j)) \in E_t$
- Non-induced case: $\forall i, j \in V_p : (i, j) \in E_p \Rightarrow (f(i), f(j)) \in E_t$

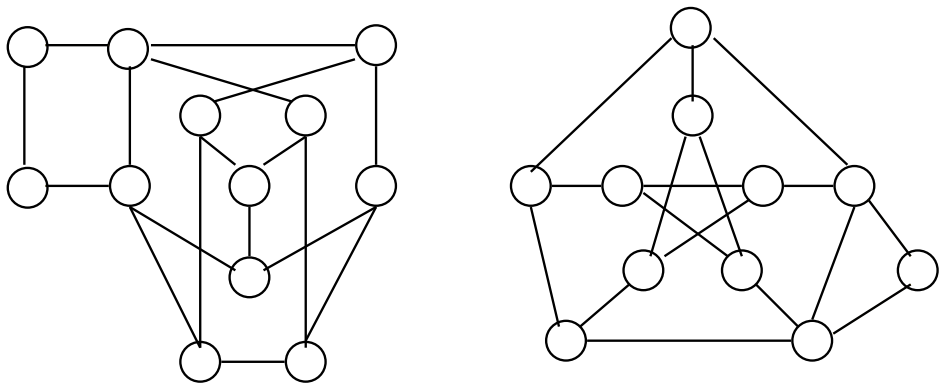
$\rightsquigarrow$ Complexity in the general case: $\mathcal{NP}$-complete (Demonstration?)

$\rightsquigarrow$ Particular cases:
- Polynomial algorithm: Trees, Outerplanar 2-connected graphs, ...
- FPT algorithm: $(\#V_p, \mathit{treeWidth}(G_t))$ bounded

Graph Matching Problems

- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost

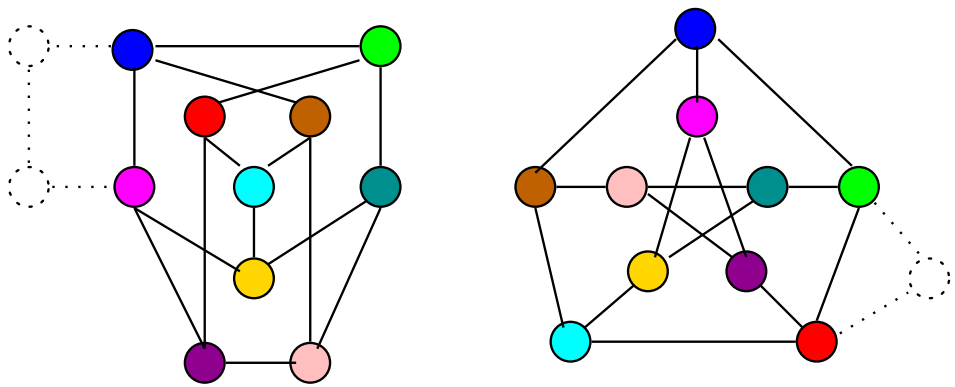
$G_1 = (V_1, E_1)$



$G_2 = (V_2, E_2)$

Graph Matching Problems

- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost



$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$

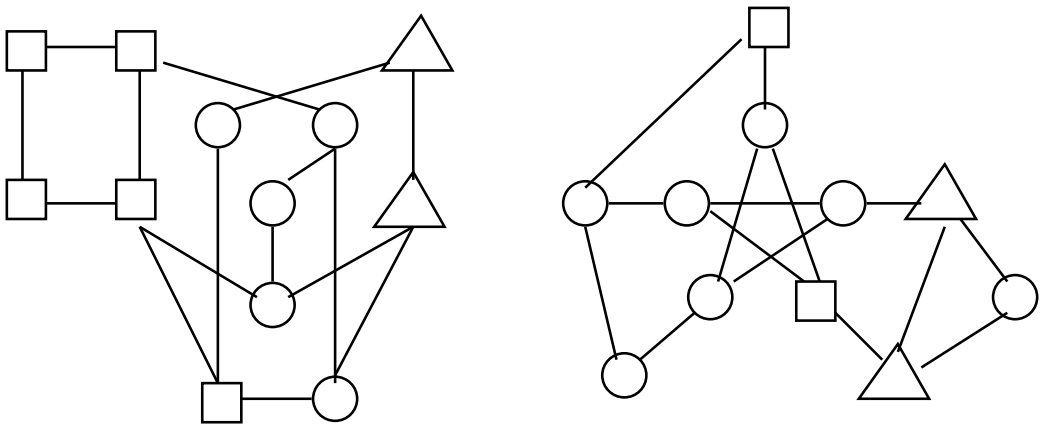
$\rightsquigarrow$ Partial injection $f : V_1 \to V_2 \cup \{\epsilon\}$ such that
- Induced case: Maximise $\#\{i \in V_1, f(i) \neq \epsilon\}$ so that $\forall i, j \in V_1, f(i) \neq \epsilon, f(j) \neq \epsilon : (i, j) \in E_1 \Leftrightarrow (f(i), f(j)) \in E_2$
- Non-induced case: Maximise $\#\{(i, j) \in E_1, (f(i), f(j)) \in E_2\}$

$\rightsquigarrow$ Complexity in the general case: $\mathcal{NP}$-hard (Demonstration?)

$\rightsquigarrow$ Particular cases
- Polynomial algorithm: Trees
- FPT algorithm: Outerplanar graphs with bounded degree

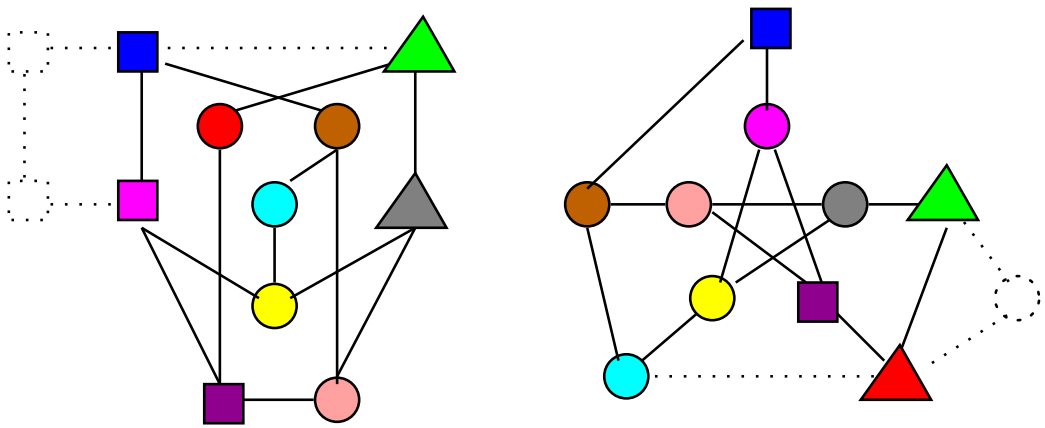
Graph Matching Problems

- Isomorphism $\leadsto$ Equivalence
- Sub-isomorphism $\leadsto$ Inclusion
- Maximum common subgraph $\leadsto$ Intersection
- Edit distance $\leadsto$ Transformation cost



Graph Matching Problems

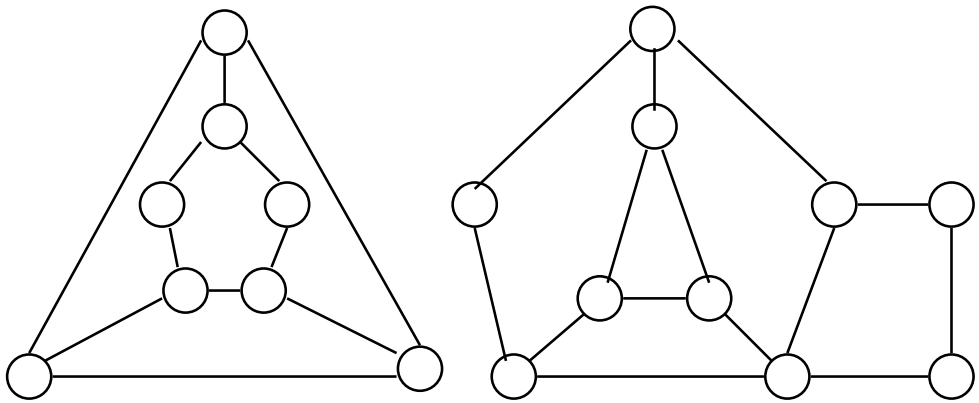
- Isomorphism $\rightsquigarrow$ Equivalence
- Sub-isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Edit distance $\rightsquigarrow$ Transformation cost



- $\rightsquigarrow$ Matching that minimises edit costs
- $\rightsquigarrow$ Complexity in the general case: $\mathcal{NP}$-hard (Demonstration?)
- $\rightsquigarrow$ Polynomial cases: Strings and ordered trees

Zoom on plane graphs and image matching

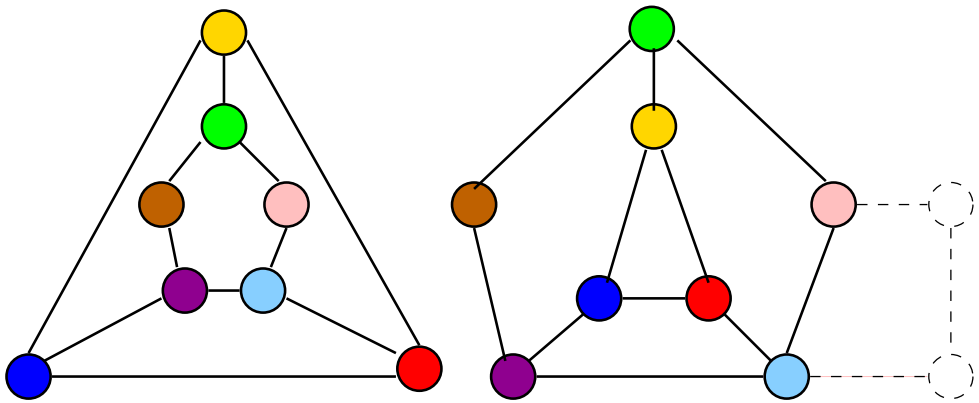
$\rightsquigarrow$ SATTIC (ANR blanc 2007-2011) and Solstice (ANR blanc 2013-2018) projects



Is the left-side graph sub-isomorphic to the right-side one?

Zoom on plane graphs and image matching

$\rightsquigarrow$ SATTIC (ANR blanc 2007-2011) and Solstice (ANR blanc 2013-2018) projects

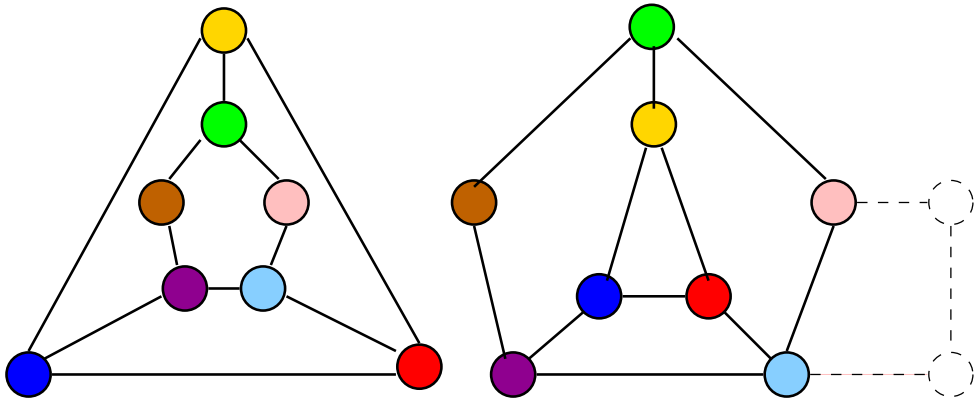


Is the left-side graph sub-isomorphic to the right-side one?
Yes, but are they similar?

- Graphs that model images are embedded in planes
- Let's compare these embeddings $\rightsquigarrow$ Combinatorial maps

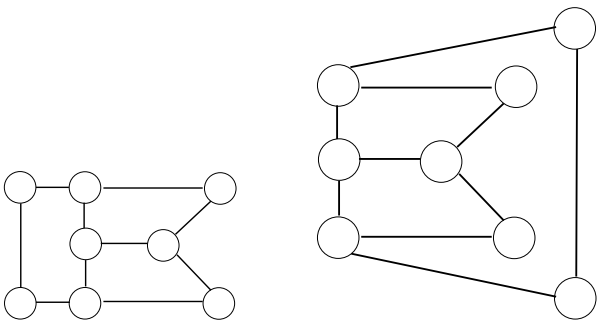
Zoom on plane graphs and image matching

$\rightsquigarrow$ SATTIC (ANR blanc 2007-2011) and Solstice (ANR blanc 2013-2018) projects

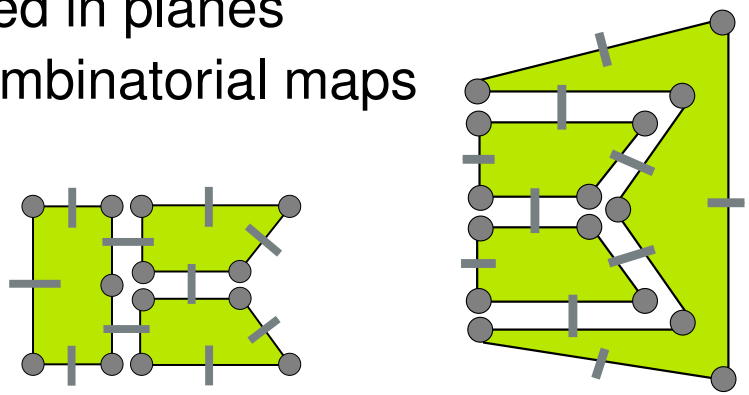


Is the left-side graph sub-isomorphic to the right-side one?
Yes, but are they similar?

- Graphs that model images are embedded in planes
- Let's compare these embeddings $\rightsquigarrow$ Combinatorial maps



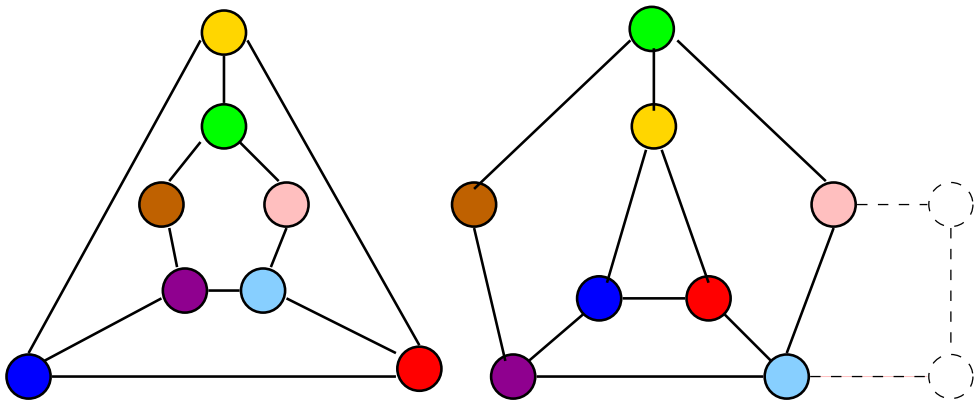
2 isomorphic graphs



2 non-isomorphic maps

Zoom on plane graphs and image matching

$\rightsquigarrow$ SATTIC (ANR blanc 2007-2011) and Solstice (ANR blanc 2013-2018) projects



Is the left-side graph sub-isomorphic to the right-side one?
Yes, but are they similar?

- Graphs that model images are embedded in planes
- Let's compare these embeddings $\rightsquigarrow$ Combinatorial maps

References:

- G. Damiand, C. Solnon, C. de la Higuera, J.-C. Janodet, and E. Samuel: *Polynomial Algorithms for Subisomorphism of nD Open Combinatorial Maps*. Computer Vision and Image Understanding (CVIU), 115(7):996-1010, Elsevier, 2011
- C. Solnon, G. Damiand, C. de la Higuera, J.-C. Janodet: *On the complexity of Submap Isomorphism and Maximum Common Submap Problems*. Pattern Recognition, 48(2):302-316, Elsevier, 2015

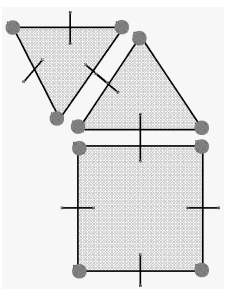
Algorithm for Submap Isomorphism

Basic idea:

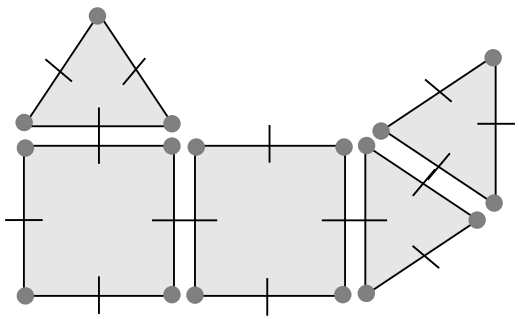
- Choose a dart d in the pattern map M
- For each dart d' in the target map M':
  - Traverse M and M' in parallel and build a dart matching
  - If the dart matching is a subisomorphism then answer Yes
- Answer No

Complexity: $\mathcal{O}(|darts(M)| \cdot |darts(M')|)$

Precondition: The pattern map M must be connected



M



M'

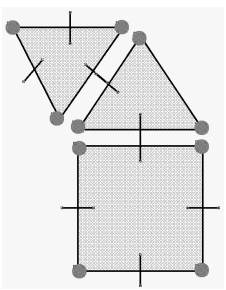
Algorithm for Submap Isomorphism

Basic idea:

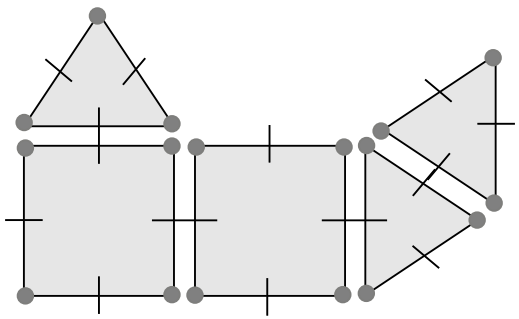
- Choose a dart d in the pattern map M
- For each dart d' in the target map M':
  - Traverse M and M' in parallel and build a dart matching
  - If the dart matching is a subisomorphism then answer Yes
- Answer No

Complexity: $\mathcal{O}(|darts(M)| \cdot |darts(M')|)$

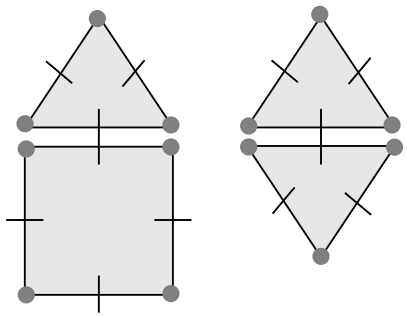
Precondition: The pattern map M must be connected



M



M'



Not connected

Submap isomorphism is $\mathcal{NP}$-complete

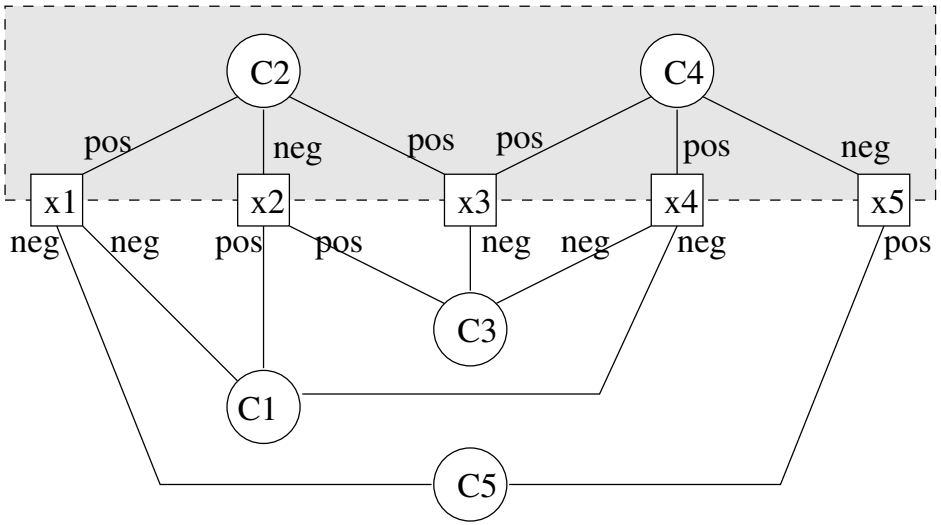
$\rightsquigarrow$ Proof by reduction of Separable Planar 3-SAT

Definition of the $\mathcal{NP}$-complete pb Separable Planar 3-SAT [Lichtenstein 82]

- Input: A CNF formula F over a set X of variables and a plane embedding of the formula graph $G_{(X,F)}$ such that
  - Every clause of F contains 2 or 3 literals
  - For every variable x_i: $d^{\circ}_{pos}(x_i) \geq 1$, $d^{\circ}_{neg}(x_i) \geq 1$, $d^{\circ}(x_i) \leq 3$
  - The cycle $(x_1, x_2, \ldots, x_n, x_1)$ separates the plane in two parts *in* and *out* such that for every variable x_i: $\{pos(x_i), neg(x_i)\} = \{in(x_i), out(x_i)\}$
- Question: Does there exist a truth assignment for X which satisfies F?

Example of instance:

$$
\begin{array}{rcl}
X & = & \{x_1, x_2, x_3, x_4, x_5\} \\
F & = & (\neg x_1 \vee x_2 \vee \neg x_4) \\
 & \wedge & (x_1 \vee \neg x_2 \vee x_3) \\
 & \wedge & (x_2 \vee \neg x_3 \vee \neg x_4) \\
 & \wedge & (x_3 \vee x_4 \vee \neg x_5) \\
 & \wedge & (\neg x_1 \vee x_5)
\end{array}
$$



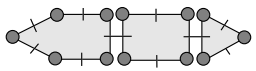
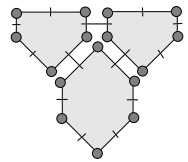
Reduction of Sep. Planar 3-SAT to Submap Isomorphism

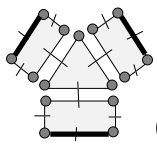
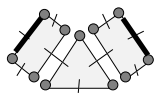
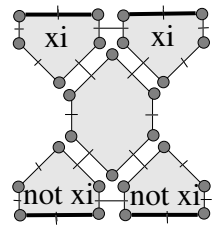
Goal of the reduction:

- Let F be a boolean formula with v variables and c clauses
- Give a polynomial time algorithm for building two maps M and M' such that F is satisfiable iff M is a submap of M'

Basic idea: Associate gadgets with variables and clauses

- The pattern map M is composed of

  c clause gadgets  and v variable gadgets 
- The target map M' is derived from the plane graph associated with F:
  - Replace vertices by gadgets:

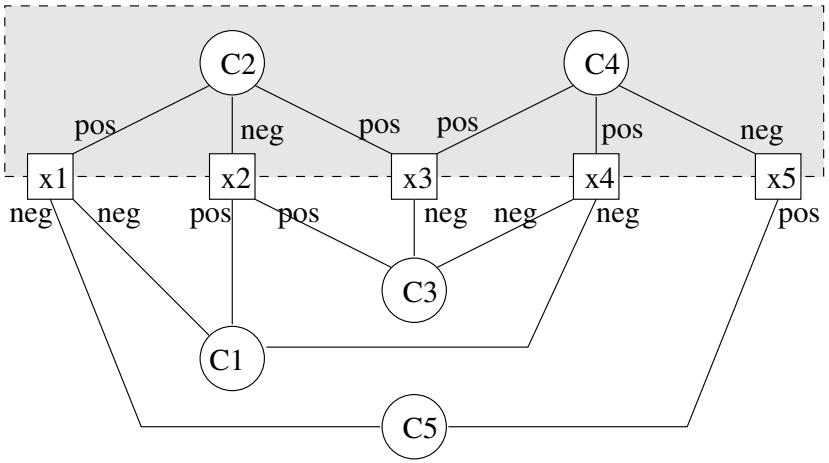
    clause gadgets  or  and variable gadgets 

  - 2-sew patterns associated with adjacent vertices

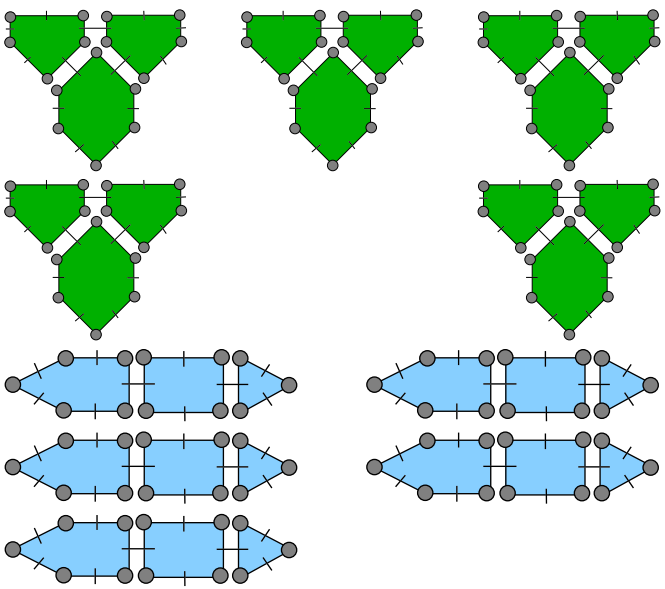
Submap isomorphism $\Leftrightarrow$ Selection of 1 satisfied literal for each clause

Example

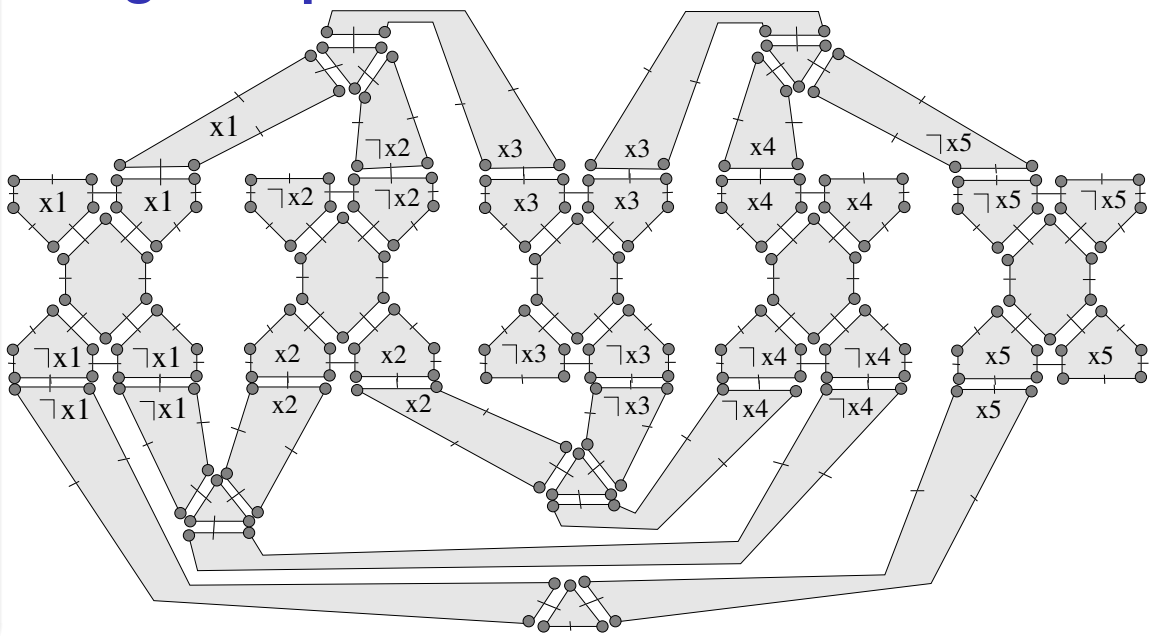
$$
\begin{aligned}
X &= \{x_1, x_2, x_3, x_4, x_5\} \\
F &= (\neg x_1 \vee x_2 \vee \neg x_4) \\
&\wedge (x_1 \vee \neg x_2 \vee x_3) \\
&\wedge (x_2 \vee \neg x_3 \vee \neg x_4) \\
&\wedge (x_3 \vee x_4 \vee \neg x_5) \\
&\wedge (\neg x_1 \vee x_5)
\end{aligned}
$$



Pattern map M:

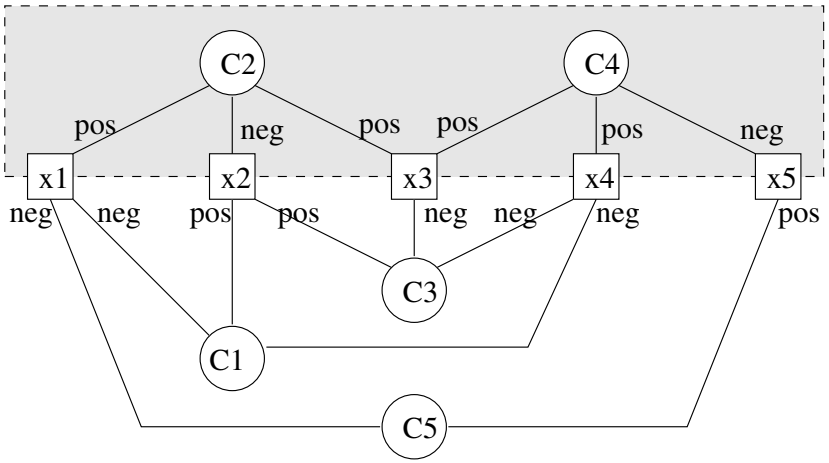


Target map M':

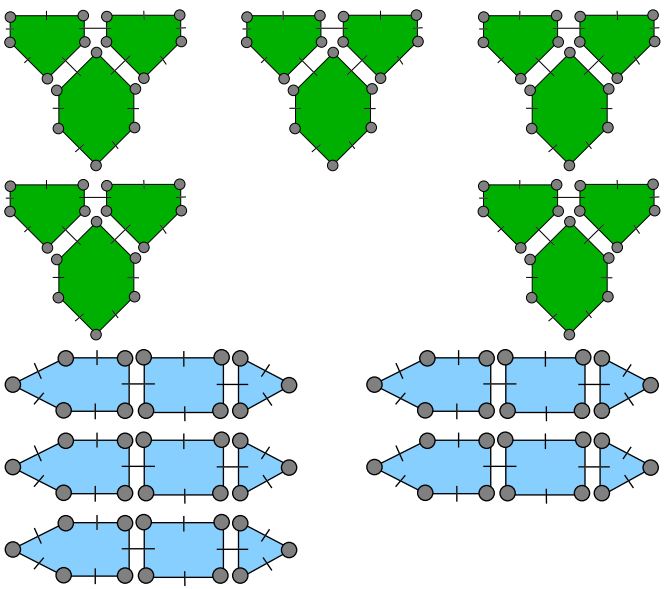


Example

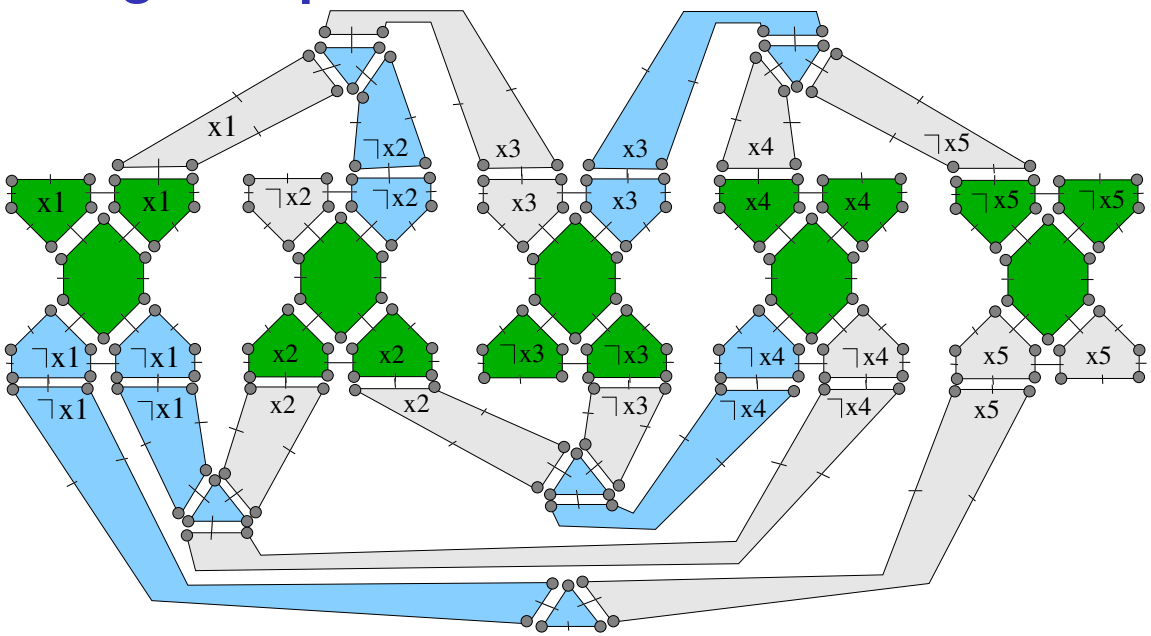
$$
\begin{aligned}
X &= \{x_1, x_2, x_3, x_4, x_5\} \\
F &= (\neg \mathbf{x_1} \vee x_2 \vee \neg x_4) \\
&\wedge (x_1 \vee \neg \mathbf{x_2} \vee x_3) \\
&\wedge (x_2 \vee \neg x_3 \vee \neg \mathbf{x_4}) \\
&\wedge (\mathbf{x_3} \vee x_4 \vee \neg x_5) \\
&\wedge (\neg \mathbf{x_1} \vee x_5)
\end{aligned}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



Pattern map M:



Target map M':



FPT algorithm for Submap isomorphism

Parameterized submap isomorphism:

- Input: Two maps M and M'
- Question: Does there exist a submap of M' which is isomorphic to M?
- Parameter: The number k of connected components in M

FPT Algorithm:

1 Decompose M into its k connected components denoted $M_1, \ldots, M_k$
2 Let V and E be two empty sets
3 **for** *each connected component* M_i *of* M **do**
4 **for** *each partial submap* M'_x *of* M' *which is isomorphic to* M_i **do**
5 add (M'_x, i) to V
6 **for** *each* $(M'_x, i) \in V$ **do**
7 **for** *each* $(M'_y, j) \in V$ *such that* $i \neq j$ **do**
8 **if** $Darts(M'_x) \cap (Darts'_y) = \emptyset$ **then** add $((M'_x, i), (M'_y, j))$ to E;
9 **if** *the graph* $G = (V, E)$ *has a clique of size* k **then return** *True* **else return** *False*

Complexity of the FPT algorithm:

Let $d = |Darts(M)|$, $d_i = |Darts(M_i)|$ and $d' = |Darts(M')|$:

- Decomposition in connected components by a traversal of M in $\mathcal{O}(d)$
- Construction of V in $\mathcal{O}(dd')$
- Construction of E in $\mathcal{O}(dd'^2)$
- Search for a clique of size k in $\mathcal{O}(d'^k)$

$\rightsquigarrow$ Overall time complexity in $\mathcal{O}(d'^k + dd'^2)$.

Conclusion:

The tractability of submap isomorphism depends on the number of connected components in the pattern map

Plan

Parenthesis on Formal Systems

What is a formal system?

- Set of axioms and inference rules
- Theorems are proven by applying rules to axioms
  $\rightsquigarrow$ The set of all theorems is recursively enumerable

Example (inspired from Presburger arithmetic):

- Let N be the set recursively defined by:
  - $0 \in N$
  - $\forall x \in N, s(x) \in N$
- Axiom: $\forall x \in N, p(0, x, x)$
- Rule: $\forall x, y, z \in N, p(x, y, z) \Rightarrow p(s(x), y, s(z))$
- Theorems: $p(0, 0, 0)$, $p(0, s(0), s(0))$, $p(s(0), s(s(0)), s(s(s(0))))$, ...

In other words, in Prolog:

```
p(0,X,X).
p(s(X),Y,s(Z)):- p(X,Y,Z).
```

Properties of a formal system

Consistency:

The negation of a theorem cannot be a theorem
$\rightsquigarrow$ The system cannot prove contradictory theorems

Decidability:

An assertion A is decidable if the system can prove A or $\neg A$
$\rightsquigarrow$ There exists an algorithm to prove or refute A

Completeness:

Every assertion (expressed with the system language) is decidable
$\rightsquigarrow$ The system can prove or refute any assertion

Example of formal system which is consistent and complete:

Presburger arithmetic [Presburger 1929]

From Hilbert to Gödel

Hilbert's programme (1920):

Find a formal system which is both consistent and complete for mathematics $\rightsquigarrow$ Meta-mathematics

> *We are not speaking here of arbitrariness in any sense. Mathematics is not like a game whose tasks are determined by arbitrarily stipulated rules. Rather, it is a conceptual system possessing internal necessity that can only be so and by no means otherwise.*

Gödel's incompleteness theorems (1931):

- A consistent formal system cannot be complete for natural number arithmetic (Peano arithmetic)
- A formal system cannot demonstrate its own consistency

Incompleteness of Formal Systems for Arithmetic

Can we complete an incomplete formal system?

- An incomplete formal system contains undecidable assertions
- Idea: Add a new axiom for each undecidable assertion
- Problem: There will still be undecidable assertions

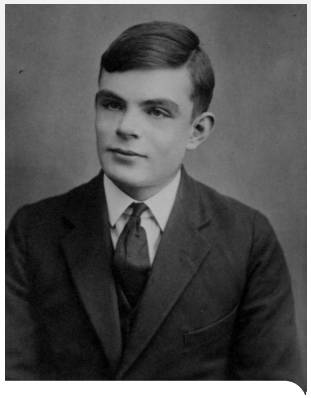
Are there many undecidable assertions?

The probability that a randomly generated assertion of length n is undecidable tends to 1 when n tends to ∞
$\rightsquigarrow$ When $n \to \infty$, all assertions are undecidable!

Reference:

Jean-Paul Delahaye: Presque tout est indécidable
Pour la Science - n° 375 - Janvier 2009 - pages 88 à 93

From Formal Sytems to Computability



It is possible to invent a single machine which can be used to compute any computable sequence. Alan Turing, 1936

Computability of a function f:

There exist formal rules to compute $f(x)$ from x in a finite number of steps

Two different formalisms introduced in 1936:

λ-calculus (Church) and Turing's machine

Church's Thesis: Turing-complete formalisms

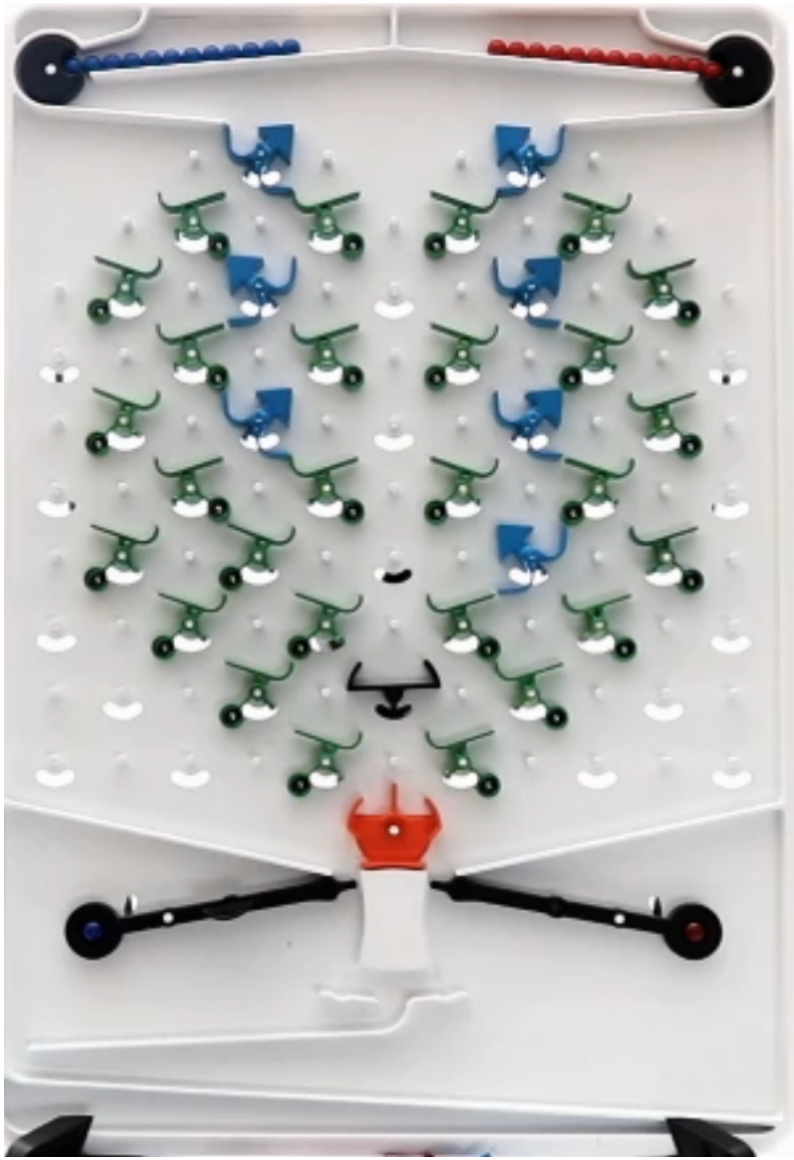
Same power of expression as a Turing machine

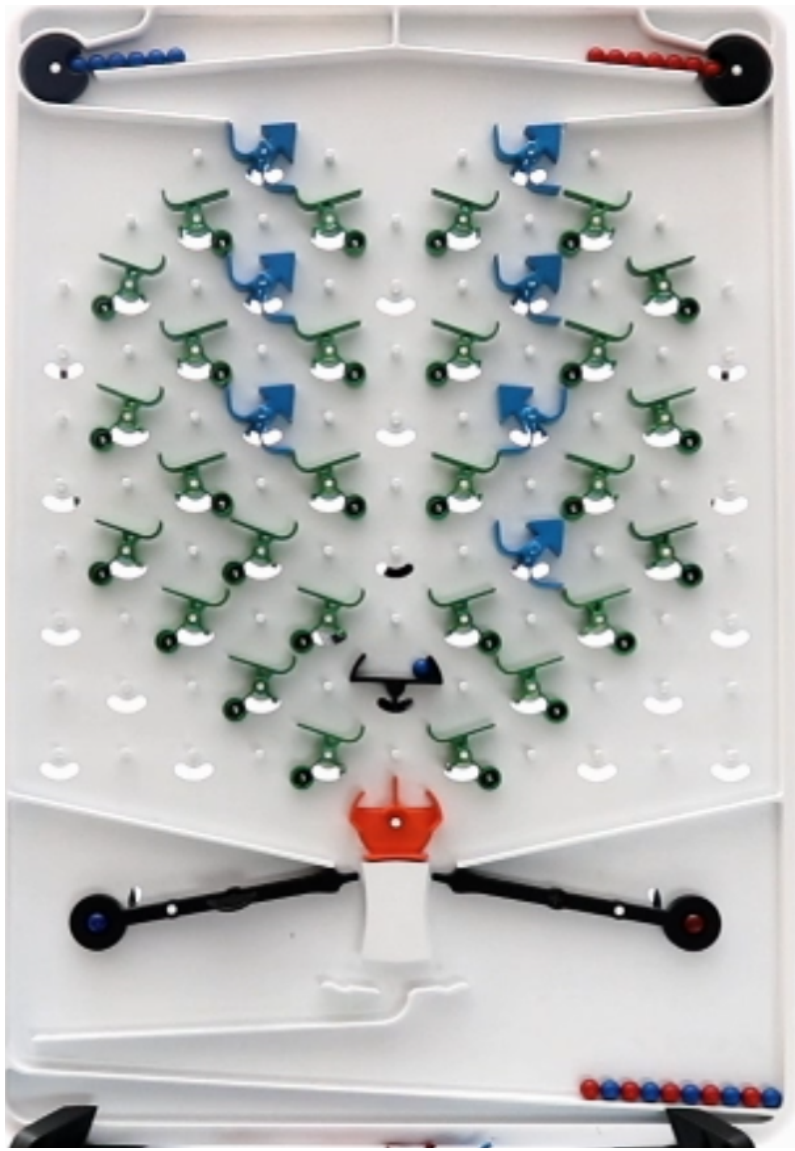
Examples of Turing-complete formalisms:

- Game of life (Conway, 1970)
- ...
- Biochemical reaction networks (F. Fages, G. Le Guludec 2017)

Computing with marbles: $5 + 6 = 11$

$\rightsquigarrow$ See https://www.turingtumble.com/





Lambda Calculus [Church 1936]

Definition of λ-terms:

- A variable x is a λ-term
- Abstraction: $\lambda x.t$ is a λ-term if t is a λ-term and x a variable
- Application: ts is a λ-term if t and s are λ-terms

β-reduction: $(\lambda x.t)s \to t[x := s]$

In other words: Replace x with s in t
$\rightsquigarrow$ For example, $(\lambda x.xy)a \to (xy)[x := a] = ay$

Exercise: Reduce the following λ-terms

- $(\lambda x.x)((\lambda y.y)z)$
- $(\lambda x.xx)(\lambda x.xx)$
- $(\lambda x.xxx)(\lambda x.xxx)$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda h.\lambda z.hhz)fx$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda h.\lambda z.hhz)fx$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda z.ffz)x$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda h.\lambda z.hhz)fx$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda z.ffz)x \rightarrow \lambda f.\lambda x.(\lambda g.\lambda y.gy)fffx$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda h.\lambda z.hhz)fx$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda z.ffz)x \rightarrow \lambda f.\lambda x.(\lambda g.\lambda y.gy)fffx$
$\rightarrow \lambda f.\lambda x.(\lambda y.fy)ffx$

Definition of the addition on $\mathbb{N}$ with λ-calculus

Definition of natural numbers:

- 0 is defined by $\lambda f.\lambda x.x$
- 1 is defined by $\lambda f.\lambda x.fx$
- 2 is defined by $\lambda f.\lambda x.ffx$
- ...

Definition of a function that returns $n+1$: $\lambda n.\lambda f.\lambda x.fnfx$

Exercise: Compute $n+1$ when $n=1$

$(\lambda n.\lambda f.\lambda x.fnfx)\lambda g.\lambda y.gy \rightarrow \lambda f.\lambda x.f(\lambda g.\lambda y.gy)fx \rightarrow \lambda f.\lambda x.f(\lambda y.fy)x \rightarrow \lambda f.\lambda x.ffx$

Definition of a function that returns $m+n$: $\lambda m.\lambda n.\lambda f.\lambda x.mfnfx$

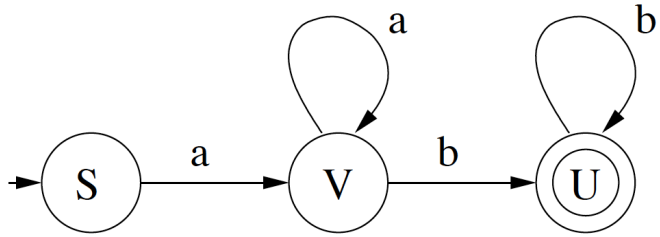
Exercise: Compute $m+n$ when $m=1$ and $n=2$

$(\lambda m.\lambda n.\lambda f.\lambda x.mfnfx)(\lambda g.\lambda y.gy)(\lambda h.\lambda z.hhz)$
$\rightarrow (\lambda n.\lambda f.\lambda x.\lambda g.\lambda y.gyfnfx)(\lambda h.\lambda z.hhz)$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda h.\lambda z.hhz)fx$
$\rightarrow \lambda f.\lambda x.\lambda g.\lambda y.gyf(\lambda z.ffz)x \rightarrow \lambda f.\lambda x.(\lambda g.\lambda y.gy)fffx$
$\rightarrow \lambda f.\lambda x.(\lambda y.fy)ffx \rightarrow \lambda f.\lambda x.fffx$

Finite-state and Pushdown Automata (recalls from 4IF)

Finite-state Automaton $(Q, \Sigma, \delta, I, F)$

- Q = Finite set of states
- Σ = Input alphabet
  $\rightsquigarrow$ Finite set of symbols
- $\delta : Q \times \Sigma \to Q$ = transition function
- $I \subseteq Q$ = Initial states
- $F \subseteq Q$ = Final states



$\rightsquigarrow$ Regular languages ($a^n b^m$, floating point numbers, variable names, etc)

Pushdown Automaton

- Finite-state automaton with a stack (LIFO)
  $\rightsquigarrow$ Each transition may read/write a symbol on top of the stack

$\rightsquigarrow$ Context-free languages ($a^n b^n$, properly matched parenthesis, Java, etc)

Turing Machine [Turing 1936]

Definition: $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$

- Q = Finite set of states with $q_0 \in Q$ (initial state) and $F \subseteq Q$ (final states)
- Γ = alphabet with $b \in \Gamma$ (blank) and $\Sigma \subseteq \Gamma \setminus \{b\}$ (input alphabet)
- $\delta : Q \setminus F \times \Gamma \to Q \times \Gamma \times \{L, R\}$ = (partial) transition function

$\rightsquigarrow$ Finite-state automaton with an infinite tape (infinite sequence of cells)

Execution:

- Initially, the machine is in state q_0, the tape contains an input word, and the head is positioned on the first symbol of this word
- While not Halt:
  - Let q_i be the current state and a be the symbol in the current cell
  - If $\delta(q_i, a)$ is not defined, then Halt
  - Else if $\delta(q_i, a) = (q_j, b, x)$, then:
    - Go to state q_j
    - Replace a with b in the current cell
    - Move the head for one cell left (if $x = L$) or right (if $x = R$)
- If the current state $\in F$, then the input word is accepted by M

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011**:**

State	Tape (Head in blue)

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	0	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	0	b	..
q_1	..	b	1	0	0	0	0	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	0	b	..
q_1	..	b	1	0	0	0	0	b	..
q_h	..	b	1	0	1	0	0	b	..

Exercise:

Turing machine for adding two base-2 numbers?

Example: Increment a base-2 number

Algorithm:

- While symbol $\neq b$, Move the head right
  - $\delta(q_0, 0) = (q_0, 0, R)$
  - $\delta(q_0, 1) = (q_0, 1, R)$
- Move the head left
  - $\delta(q_0, b) = (q_1, b, L)$
- While symbol $= 1$, write 0 and move the head left
  - $\delta(q_1, 1) = (q_1, 0, L)$
- Write 1 and stop...
  - $\delta(q_1, 0) = (q_h, 1, L)$
  - $\delta(q_1, b) = (q_h, 1, L)$

Example when input = 10011:

State	Tape (Head in blue)								
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_0	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	1	b	..
q_1	..	b	1	0	0	1	0	b	..
q_1	..	b	1	0	0	0	0	b	..
q_h	..	b	1	0	1	0	0	b	..

Exercise:

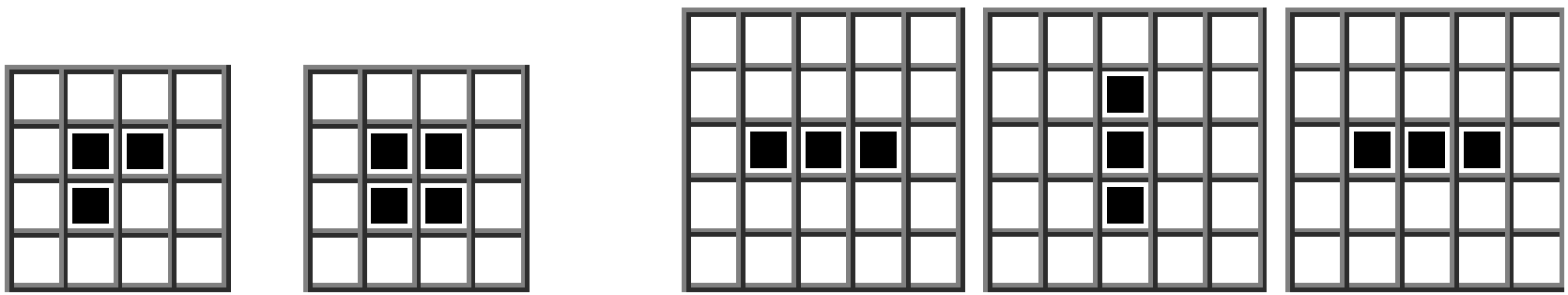
Turing machine for adding two base-2 numbers?

The Game of Life [Conway's game "life" by M. Gardner, 1970]

Definition:

- 2 dimensional grid composed of cells: Each cell is alive or dead
- For each iteration:
  - A dead cell with 3 living neighbours becomes alive
  - A living cell with 2 or 3 living neighbours stays alive
  - A living cell with less than 2 or more than 3 living neighbours dies

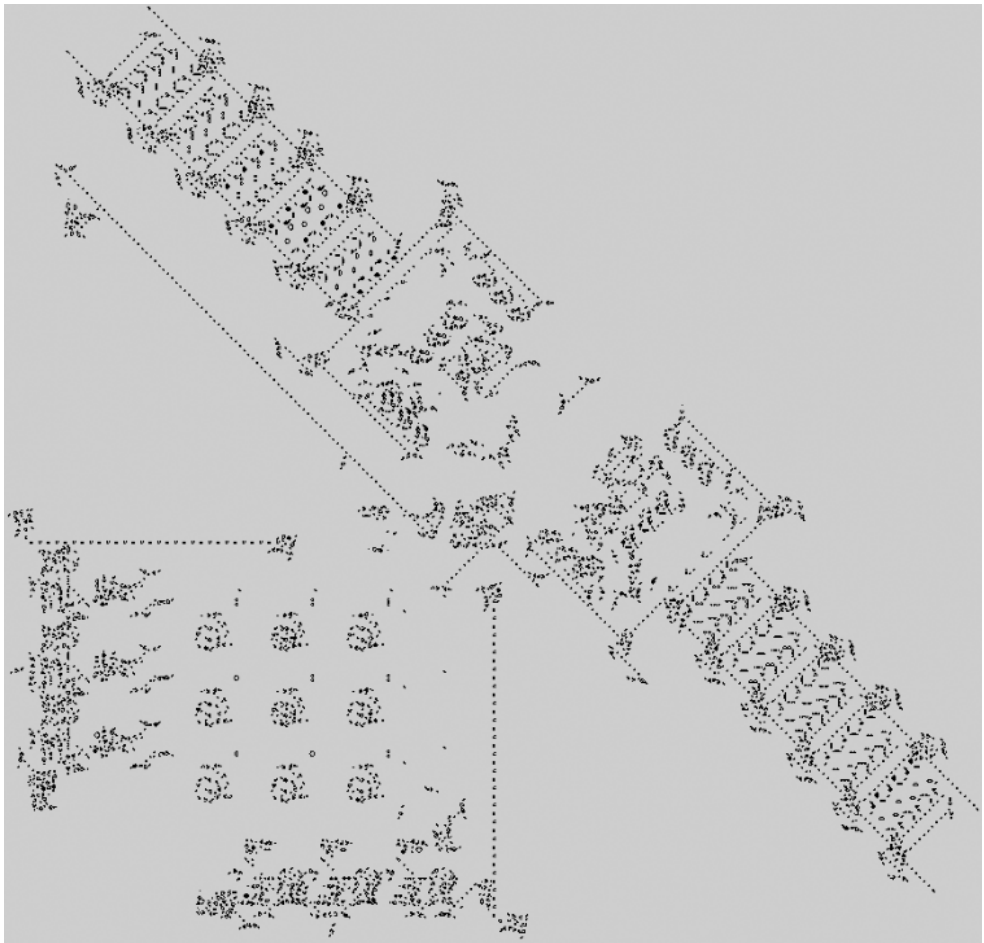
$\rightsquigarrow$ Particular case of cellular automaton



The game of life is a complex system!

There does not exist an algorithm for deciding if a pattern will appear in a future state given the initial state (if the grid is infinite)

The game of life is Turing-complete



References:

- Rendell: A universal Turing machine in Conway's Game of Life, 2011
- Zucconi: Let's build a computer in Conway's game of life (Video), 2020

Decidability of a Decision Problem

Decidable problems

There exists an algorithm which answers the question within a finite number of instructions

Undecidable problems

There cannot exist an algorithm which answers the question for all instances (but we may find algorithms which are able to answer for some instances)

Semi-decidable problems

There exists an algorithm which always answers when the answer is YES, but may not terminate when the answer is NO

Halting Problem of the Turing Machine

Definition of the halting problem:

- Input:
  - A program P with one input parameter X
  - A value V for the parameter X
- Question: Does the execution of P on V (denoted $P(V)$) ends?

The halting problem is undecidable [Turing 1936]:

- Suppose there exists a program *halt(P,V)* specified as follows:
  - Input: a program P and a value V
  - Postrelation: Return true if $P(V)$ ends, and false otherwise
- Let us define the program *diag* with an input parameter X:
  $diag(X)$ = if $halt(X, X)$ then infinite loop else return true
- *diag(diag)* ends $\Rightarrow$ *halt(diag,diag)* = true $\Rightarrow$ *diag(diag)* does not end
  $\rightsquigarrow$ Contradiction

Is the halting problem semi-decidable?

Other Examples of Undecidable Problems

Tenth Hilbert's problem [Matiyasevic, 1970]:

- Input: a diophantine equation (polynomial equation)
- Question: Does there exist an integer solution?

Post problem:

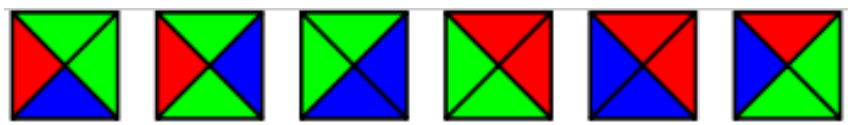
- Input: 2 lists $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $\beta_1, \beta_2, \ldots, \beta_n$ of n words
- Question: Does there exist $i_1, i_2, \ldots, i_k$ s.t. $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$?
- Example of instance: Input =

α_1	α_2	α_3
a	ab	bba

β_1	β_2	β_3
baa	aa	bb

Tilling problem:

- Input: A finite set S of squares with colored edges
- Question: Can we till any $n \times n$ surface with copies of S so that 2 adjacent edges have a same colour?
- Example of instance: Input =



Plan

First Program Proofs?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows.

Alan Turing, 1949

But algorithms appeared long before computers...

... and it is most probable that they were proven correct!?
See the course of Gilles Dowek "Preuve et calcul, des rapports intimes"
(http://www.college-de-france.fr/site/gerard-berry/seminar-2008-02-22-12h00.htm)

Proof of Program Properties

Examples of properties we may want to prove:

- Termination: The program always ends
- Completeness: The program computes output values for every input values that satisfy preconditions
- Correction: The postrelation between input and output values is always satisfied
- Correction of an invariant: The assertion is true for each iteration
- Equivalence of 2 programs P_1 and P_2: P_1 outputs the same values as P_2 given the same input values
- Overflow: No overflow at run time
- etc...

Proof vs Test

Tests may demonstrate the absence of a property:

- Failure of a test $\Rightarrow$ Counter-example of a property:
  $\rightsquigarrow$ Incorrect invariant, Output that does not satisfy the postrelation, etc

Tests cannot prove correction nor termination!

Unless there is a finite number of possible input values, and testing each of them... but in this case we have made a proof!

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

Edsger W. Dijkstra, 1972

Parenthesis 1: Some algorithms are deliberately incomplete

Incomplete approaches to solve $\mathcal{NP}$-hard problems

- Incomplete exploration of the search space
  $\rightsquigarrow$ No guarantee on the result

Example of incomplete algorithm for SAT:

Input: A boolean formula F defined on a set X of n variables

```
1 begin
2     Initialise all variables of X to true in V
3     while Time limit not reached do
4         if V satisfies all clauses of F then return V;
5         Choose a variable x_i ∈ X
6         Flip the value of x_i in V
7     return ?
```

Local search algorithms are usually incomplete!

Parenthesis 2: Some algorithms are deliberately incorrect

Incorrect approaches for $\mathcal{NP}$-hard optimisation problems

- Approximation of the optimal solution computed in polynomial time
- In some cases we have guarantees on the error (ρ-approximation)

Example of 2-approximation algorithm for the TSP

- Compute a Minimum Spanning Tree (MST) of the graph
- Perform a Depth First Search (DFS) of the MST
- Return the cycle C corresponding to the order vertices are discovered during search

Exercise: Prove that length of $C \leq 2 \times$ Length of shortest hamiltonian cycle if distances between vertices satisfy the triangle inequality property

Heuristic algorithms are usually incorrect!

The postrelation must state that the returned solution may not be optimal

A proof may be “handmade” (1/2)

Example: Compute distances in a graph

Let v_0 and v_i be two vertices in a directed graph

- If there exists a path from v_0 to v_i:
  - $\delta(v_0, v_i)$ = length of the shortest path from v_0 to v_i
    (where the length of a path is equal to its number of edges)
- Otherwise: $\delta(v_0, v_i) = \infty$

Algorithm for computing $\delta(v_0, v_i)$?

A proof may be “handmade” (1/2)

Example: Compute distances in a graph

Let v_0 and v_i be two vertices in a directed graph

- If there exists a path from v_0 to v_i:
  - $\delta(v_0, v_i)$ = length of the shortest path from v_0 to v_i
    (where the length of a path is equal to its number of edges)
- Otherwise: $\delta(v_0, v_i) = \infty$

Algorithm for computing $\delta(v_0, v_i)$?

Recalls (3IF): Breadth First Search (BFS) starting from v_0

A proof may be “handmade” (2/2)

Input: A directed graph $G = (V, A)$ and a vertex $v_0 \in S$
Output: An array d such that $d[v_i] = \delta(v_0, v_i)$

```
1  begin
2      for each vertex v_i ∈ S do
3          Colour v_i in white; d[v_i] ← ∞
4      Add v_0 to f and colour v_0 in gray ; d[v_0] ← 0
5      while f not empty do
6          Let v_k be the oldest vertex in f
7          while ∃v_i ∈ succ(v_k) such that v_i is white do
8              Add v_i to f and colour v_i in gray
9              d[v_i] ← d[v_k] + 1
10         Remove v_k from f and colour v_k in black
```

Exercise 1: Prove that this algorithm terminates

A proof may be “handmade” (2/2)

Input: A directed graph $G = (V, A)$ and a vertex $v_0 \in S$
Output: An array d such that $d[v_i] = \delta(v_0, v_i)$

```
1  begin
2      for each vertex v_i ∈ S do
3          Colour v_i in white; d[v_i] ← ∞
4      Add v_0 to f and colour v_0 in gray ; d[v_0] ← 0
5      while f not empty do
6          Let v_k be the oldest vertex in f
7          while ∃v_i ∈ succ(v_k) such that v_i is white do
8              Add v_i to f and colour v_i in gray
9              d[v_i] ← d[v_k] + 1
10         Remove v_k from f and colour v_k in black
```

Exercise 2: Prove that this algorithm is correct

A proof may be “handmade” (2/2)

Input: A directed graph $G = (V, A)$ and a vertex $v_0 \in S$
Output: An array d such that $d[v_i] = \delta(v_0, v_i)$

```
1  begin
2      for each vertex v_i ∈ S do
3          Colour v_i in white; d[v_i] ← ∞
4      Add v_0 to f and colour v_0 in gray ; d[v_0] ← 0
5      while f not empty do
6          Let v_k be the oldest vertex in f
7          while ∃v_i ∈ succ(v_k) such that v_i is white do
8              Add v_i to f and colour v_i in gray
9              d[v_i] ← d[v_k] + 1
10         Remove v_k from f and colour v_k in black
```

Exercise 2: Prove that this algorithm is correct

Help: invariant properties at line 6

1. Every successor of a black vertex is either gray or black
2. For every gray or black vertex v_i, $d[v_i] = \delta(v_0, v_i)$
3. Let $< v_1, v_2, \ldots, v_k >$ be the vertices in f, from the most recent to the oldest: $d[v_1] \geq d[v_2] \geq \ldots \geq d[v_k]$ and $d[v_1] \leq d[v_k] + 1$

Limitations of "handmade" demonstrations

- Often hard to design
- Sometimes hard to check
- Usually done on the algorithm, not on the code!

"Beware of bugs in the above code; I have only proved it correct, not tried it."

D. Knuth

Can we use computers to prove properties?

Interactive checking:

The user finds a proof, the computer checks the proof
$\rightsquigarrow$ The proof must be written in a formal and decidable language

Static analysis:

The user finds an abstraction, the computer builds a proof and checks it
$\rightsquigarrow$ Find the right level of abstraction

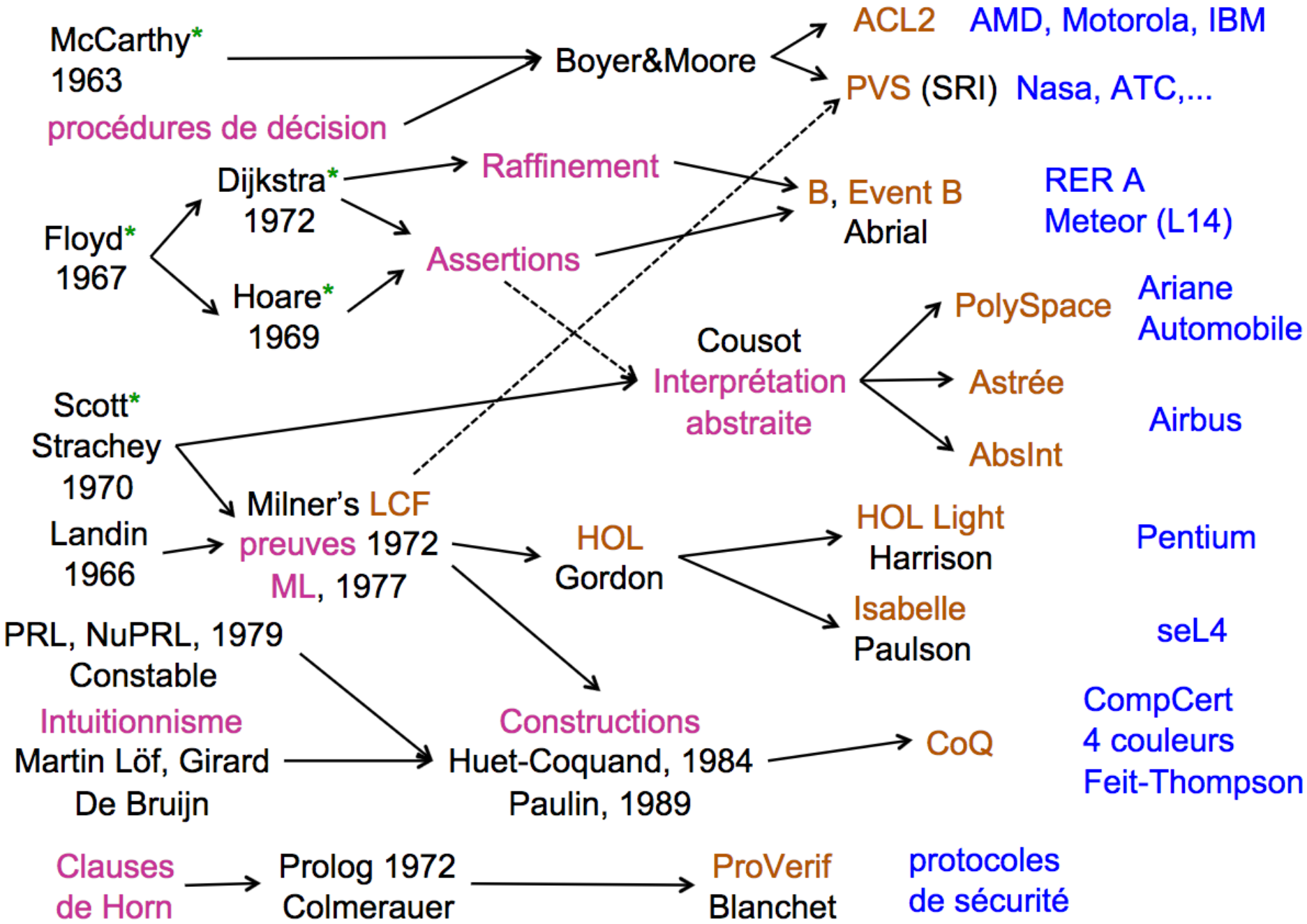
Automatic checking:

The computer finds an abstraction, builds a proof and checks it

Limitations:

- Undecidable problem in the general case $\rightsquigarrow$ Possibility of false alarms
- In general, checking the correctness of a proof is $\mathcal{NP}$-complete

$\rightsquigarrow$ Hot research topic


McCarthy*
1963
procédures de décision
Boyer&Moore
ACL2
AMD, Motorola, IBM
PVS (SRI)
Nasa, ATC,...
Dijkstra*
1972
Raffinement
Floyd*
1967
Assertions
B, Event B
Abrial
RER A
Meteor (L14)
Hoare*
1969
Cousot
Interprétation
abstraite
PolySpace
Ariane
Automobile
Astrée
Scott*
Strachey
1970
Airbus
AbsInt
Milner's LCF
preuves 1972
ML, 1977
Landin
1966
HOL
Gordon
HOL Light
Harrison
Pentium
Isabelle
Paulson
seL4
PRL, NuPRL, 1979
Constable
CompCert
4 couleurs
Feit-Thompson
Intuitionnisme
Martin Löf, Girard
De Bruijn
Constructions
Huet-Coquand, 1984
Paulin, 1989
CoQ
Clauses
de Horn
Prolog 1972
Colmerauer
ProVerif
Blanchet
protocoles
de sécurité

Plan