Pina Kolling
piko0011

Summary: Lecture 6

Summary for the chapter *8.2.* [1, 7]

Completeness

Let C be a complexity class and let L be a language in C. L is called *C-complete* if any language $L' \in C$ can be reduced to L.

(Every language of a complexity class can be reduced to L.)

- reducitbility is transitive → problems are ordered by difficulty
- complete problems can capture the difficulty of a class
- problem is seen as cpmpletely understood if the problem is complete

Question:
Which problems can be reduced to a formal language?

SAT can be expressed as formal language. [6]
⇒ SAT can be reduced to a formal language. (?)
SAT is in NP. [7]

Because CIRCUIT SAT can be reduced to SAT: CIRCUIT SAT can be reduced to a formal language. (?) CIRCUIT SAT is NP-complete. [3]
Any formal language $L \in$ NP can be reduced to CIRCUIT SAT? OR the other way around?

Formal language

Formal languages are abstract languages, which define the syntax of the words that get accepted by that language. It is a set of words that get accepted by the language and has a set of symbols that is called alphabet, which contains all the possible characters of the words. Those characters are called nonterminal symbols. [2, 8]

Kleene star
The Kleene star Σ^* of an alphabet Σ is the set of all words that can be created through concatenation of the symbols of the alphabet Σ. The empty word ϵ is included.

Formal language
A formal language L over an alphabet Σ is a subset of the Kleene star of the alphabet: $L \subseteq \Sigma^*$

Where to set the line between lanuguage decisions and other problems? Can every problem be contrcuted as a formal language?

Is everything that is reducable to SAT reducable to a formal language because of the transitivity?

I assume it does not have an influence on the complexity of a problem if it can be expressed as a formal language? Are formal languages part of specific complexity classes?

Closed under reduction

The following complexity classes are all closed under reductions:

P NP coNP L NL PSPACE EXP

A class C is closed under reductions if whenever L is reducible to L' and $L' \in C'$, then $L\ in C'$.

If a complete problem in C belongs in a class $C' \subseteq C$, $C = C'$.

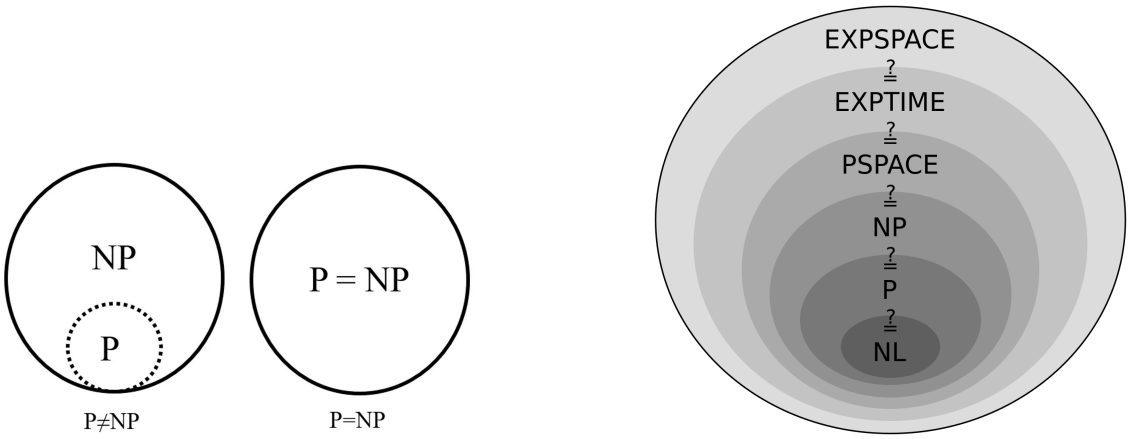

Figure 1: P and NP sets [5] and complexity classes [4]

- examples:
  - if an NP-complete language is in P, then NP = P
  - if a P-complete language is in L, then P = L
  - if a P-complete language is in NL, then P = NL
  - no EXP-complete language can be in P

Table method – time complexity

- table method to understand time complexity
- Turing Machine M decides language L on an input x
- computation on input x as $|x|^k \times |x|^k$ computation table

$\rightarrow$ $|x|^k$ is the time bound

- rows i are the time steps (from 0 to $|x|^k - 1$)
- columns j are the string positions
- T_{ij} represents the content of position j of the string of M at time i (after i steps)

Example:

- Turing Machine M deciding palindromes in time n^2

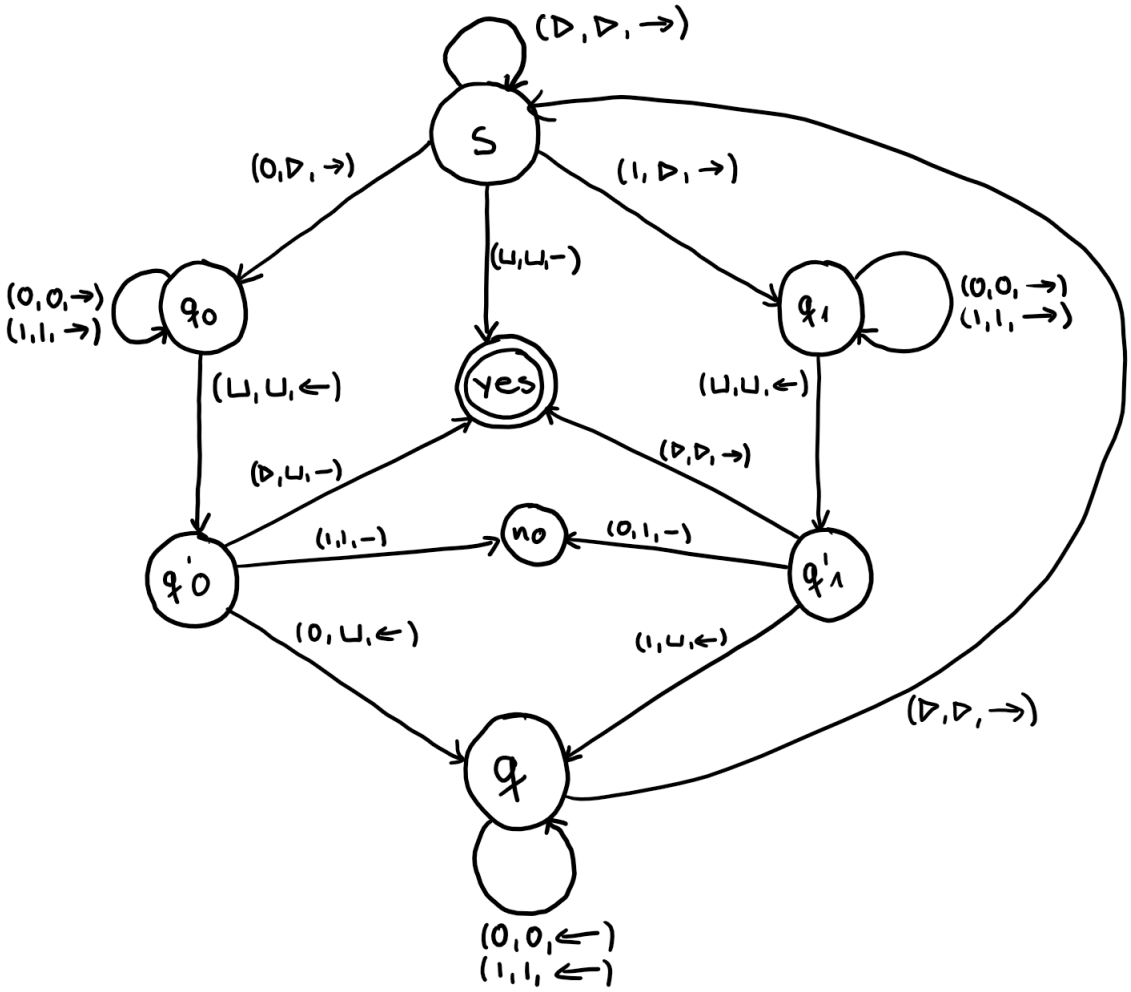

Figure 2: Binary palindrome Turing Machine M contructed from the definition in example 2.3 in the book [7]

- edges of the Turing machine:
  (input symbol σ, symbol to replace input symbol on tape with, direction in which the head moves)

$p \in K, \quad \sigma \in \Sigma$		$\delta(p, \sigma)$
s	0	$(q_0, \triangleright, \rightarrow)$
s	1	$(q_1, \triangleright, \rightarrow)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	$\sqcup$	(“yes”, $\sqcup$, $-$)
q_0	0	$(q_0, 0, \rightarrow)$
q_0	1	$(q_0, 1, \rightarrow)$
q_0	$\sqcup$	$(q_0', \sqcup, \leftarrow)$
q_1	0	$(q_1, 0, \rightarrow)$
q_1	1	$(q_1, 1, \rightarrow)$
q_1	$\sqcup$	$(q_1', \sqcup, \leftarrow)$

$p \in K, \quad \sigma \in \Sigma$		$\delta(p, \sigma)$
q_0'	0	$(q, \sqcup, \leftarrow)$
q_0'	1	(“no”, 1, $-$)
q_0'	$\triangleright$	(“yes”, $\sqcup$, $\rightarrow$)
q_1'	0	(“no”, 1, $-$)
q_1'	1	$(q, \sqcup, \leftarrow)$
q_1'	$\triangleright$	(“yes”, $\triangleright$, $\rightarrow$)
q	0	$(q, 0, \leftarrow)$
q	1	$(q, 1, \leftarrow)$
q	$\triangleright$	$(s, \triangleright, \rightarrow)$

Figure 3: Definition of binary palindrome Turing Machine M in example 2.3 in the book [7]

On input $x = 0110$ it results in the following computaion table:

▷	0_s	1	1	0	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1_{q_0}	1	0	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1	1_{q_0}	0	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1	1	0_{q_0}	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1	1	0	$⊔_{q_0}$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1	1	$0_{q'_0}$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1	1_q	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1_q	1	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	$▷_q$	1	1	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	1_s	1	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	▷	1_{q_1}	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	▷	1	$⊔_{q_1}$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	▷	$1_{q'_1}$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	$▷_q$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	▷	$⊔_s$	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔
▷	▷	▷	"yes"	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔

Figure 4: Computation table of binary palindrome Turing Machine M on input $x = 0110$ in the book [7]

- σ is the current input symbol
- q is the current state
- ⊔ is the clank symbol
- ▷ is the first symbol

Questions:
Is the Turing Macheine constructed correctly? Does the *no*-state need a double circle, because it halts or does only the accepting state get a double circle?

P-completeness of CircuitValue

Problem: CircuitValue

The CircuitValue Problem is the problem of computing the output of a given Boolean circuit on a given input.

In terms of time complexity, it can be solved in linear time (topological sort).

- P-complete

Proof idea:

- CircuitValue is in P (prerequisite for being P-complete)
- show: any language $L \in$ P can be reduced to CircuitValue
- L is decided by Turing Machine M in polynomial time (n^k)
- show: there is a reduction R and an input x to M
  R puts out a circuit C without variable gates, whose value is true only if M accepts x
- computation table T of M
- if $i = 0$: value of $T_{i,j}$ is the jth symbol of x or a ⊔

- if $j = 0$: value of $T_{i,j}$ is a $\triangleright$
- if $j = |x|^k - 1$: value of $T_{i,j}$ is a $\sqcup$

$\triangleright$	0_s	1	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_{q_0}	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1_{q_0}	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	0_{q_0}	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	0	$\sqcup_{q_0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	$0_{q_0'}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1_q	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_q	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright_q$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_s	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	1_{q_1}	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	1	$\sqcup_{q_1}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	$1_{q_1'}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright_q$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	$\sqcup_s$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	"yes"	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

Figure 5: Initial rows marked on previous palindrome example table

- value of $T_{i,j}$ is the content of position j of the string on time i

$\rightarrow$ depends on the same position and neighbor position in the previous steo $i-1$: $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$

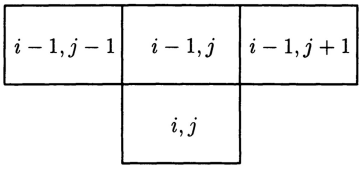

Figure 6: Value of $T_{i,j}$ depends on values of $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$

- $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1} \in \Sigma \Rightarrow$ cursor not at position $j-1, j, j+1$, this concludes to $T_{i,j} = T_{i-1,j}$

- set Γ contains all symbols that can appear on the table (symbols from Σ or symbol state combinations)
- encode each symbol $\sigma \in \Sigma$ as a vector $s = (s_1, ..., s_m)$
  - the entries of the vector $(s_1, ..., s_m)$ are either 0 or 1
  - the vector hast $m = \log |\Gamma|$ entries
- computation table can now be constructed as a table $S_{i,j,l}$ with binary entries (and $0 \leq i \leq n^k - 1$ and $0 \leq j \leq n^k - 1$ and $1 \leq l \leq m$)
- each binary entry S_{ijl} depends on previous entries $S_{i-1,j-1,l'}, S_{i-1,j,l'}, S_{i-1,j+1,l'}$, where l' ranges over 1 to m
- there are m Boolean functions $F_1, ..., F_m$ such that

$$S_{i,j,l} = F_l(S_{i-1,j-1,1}, ..., S_{i-1,j-1,m}, S_{i-1,j+1,1}, ..., S_{i-1,j+1,m})$$

- every boolean function can be renderes as a boolean circuit

$\rightarrow$ there is a boolean circuit C with $3m$ inputs and m outputs that computes $T_{i,j}$ with given $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$

- reduction R from L to CircuitValue
- for each input x, $R(x)$ consists of $(|x|^k - 1) \cdot (|x|^k - 2)$ copies of the circuit C
  $\rightarrow$ one for each entry in $T_{i,j}$
- call $C_{i,j}$ the (i,j)th copy of C
- $C_{i,j}$ depends on $C_{i-1,j-1}, C_{i-1,j}, C_{i-1,j+1}$ (if $i \geq 0$)
- first row and first and last column known
- other entries based on these entries
- output circuit of $R(x)$ is $C_{|x|^k-1,1}$
  (final step and string position 1, assuming that string position 1 contains *yes* or *no*)
- value of circuit $R(x)$ is only true if $x \in L$ because of equivalence of table structure
- $R(x)$ is in $\log n$ space

CircuitValue without NOT remains P-complete:

- AND, OR, NOT gate in circuit
- monotone circuit: does not have NOT gates
- move NOT downwards with DeMorgan's law (put negation into bracktes and exchange operator) until inputs are changed

MonotoneCircuitValue is P-complete.

CircuitSat is NP-complete

Problem: CircuitSat

The circuit satisfiability problem (CircuitSat) is the decision problem of determining whether a given Boolean circuit has an assignment of its inputs that makes the output true.

Input: a Boolean circuit C

Question: Is there a truth assignment which makes C output the value true?

- cook's theorem: Sat is NP-complete

Proof idea:

- CircuitSat reduces to SAT
- Turing Machine decides circuit nondeterministically
- describe a reduction R
- a variable is added in the nondeterministic Turing Machine
- check if one of the variables is tue: use this choice (?)
- problem: can we set thiese variables such that the Turing Machine accepts?
- answer corresponds direct to *is there a choice of nd decisions such that the turing machine accepts?*

Pina Kolling
piko0011

- extremely direct reduction
- SAT is NP-complete

TODO
proof!
Questions:

Relation between complexity classes

$N \subseteq NL \subseteq NC \subseteq P \subseteq NP \subseteq PSPACE$

NP-complete problems

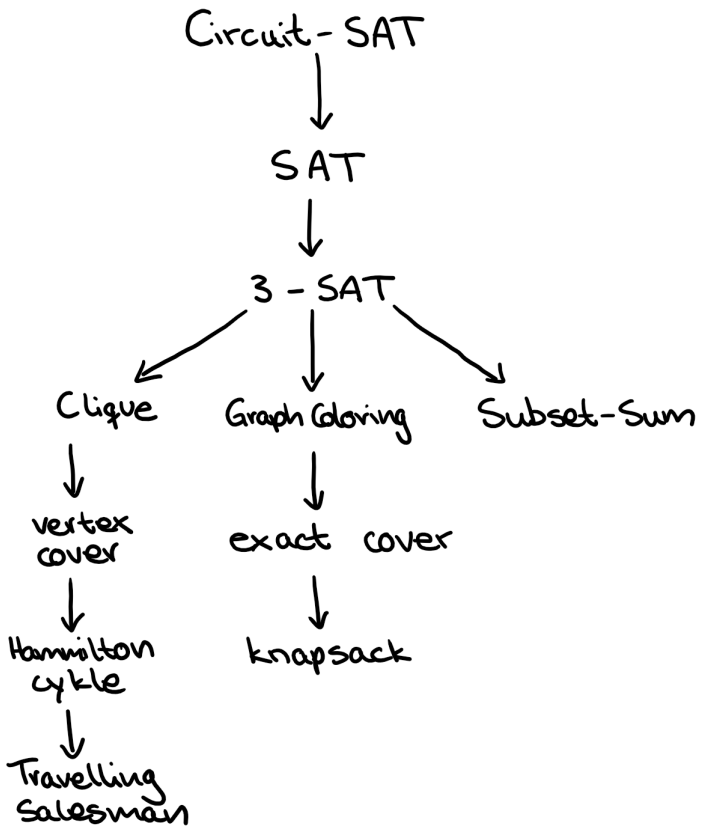

Figure 7: NP-complete problems in relation

- k-SAT for $k \geq 3$ is NP-complete

Circuit-SAT

The circuit satisfiability problem (Circuit-SAT) is the decision problem of determining whether a given Boolean circuit has an assignment of its inputs that makes the output true.

SAT

The SAT (satisfiability) problem is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. [9]

3-SAT

Like the SAT problem, 3-SAT is determining the satisfiability of a formula in CNF where each clause is limited to at most three literals.

Clique

The CLIQUE problem is the problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph.

VertexCover

In graph theory, a VERTEXCOVER (sometimes NODECOVER) of a graph is a set of vertices that includes at least one endpoint of every edge of the grap

HamiltonCycle

A HAMILTONCYCLE is a graph cycle (i.e., closed loop) through a graph that visits each node exactly once.

TravellingSalesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

GraphColoring

In graph theory, graph coloring is a special case of graph labeling. It is an assignment of colors to elements of a graph subject to certain constraints.

ExactCover

Given a collection S of subsets of set X, an exact cover is the subset S^* of S such that each element of X is contained is exactly one subset of S^*.

Knapsack

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

SubsetSum

The SUBSETSUM problem involves determining whether or not a subset from a list of integers can sum to a target value. For example, consider the list of $nums = [1, 2, 3, 4]$. If the target is 7 , there are two subsets that achieve this sum: $\{3, 4\}$ and $\{1, 2, 4\}$.

P-complete problems

- CIRCUITVALUE
- LINEARPROGRAMMING
- HORNSAT

CircuitValue

The CircuitValue Problem is the problem of computing the output of a given Boolean circuit on a given input.
In terms of time complexity, it can be solved in linear time (topological sort).
The problem is closely related to the Sat (Boolean Satisfiability) problem which is complete for NP and its complement, which is complete for co-NP.

LinearProgramming

LinearProgramming is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships.

HornSAT

HornSAT is the problem of deciding whether a given set of propositional Horn clauses is satisfiable or not.
A Horn clause is a clause (a disjunction of literals) with at most one positive literal.

NL problems

- 2-Sat
- Reachability

2-Sat

Like the SAT and 3-SAT problem, 2-SAT is determining the satisfiability of a formula in CNF where each clause is limited to at most two literals.

Reachability

Given a graph G and two nodes $n_1, n_2 \in V$, is there path from n_1 to n_2?
A graph $G = (V, E)$ is a finite set V of nodes and a set E of edges as node pairs.

Reachability can be nondeterministically solved in space $\log n$.

L problems

- 1-Sat

1-Sat

Like the SAT and 3-SAT problem, 2-SAT is determining the satisfiability of a formula in CNF where each clause is limited to at most one literal.

References

[1] Martin Berglund. *Lecture notes in Computational Complexity.*

[2] *Chomsky's Normal Form (CNF).* Website. https://www.javatpoint.com/automata-chomskys-normal-form, opened on 26.09.2022.

[3] *Circuit satisfiability problem – Proof of NP-Completeness.* Website. https://en.wikipedia.org/wiki/Circuit_satisfiability_problem, opened on 25.11.2022.

[4] *Complexity classes diagram image source.* https://en.wikipedia.org/wiki/Complexity_class.

[5] *Image source: P-NP sets.* https://www.techno-science.net/actualite/np-conjecture-000-000-partie-denouee-N21607.html.

[6] klaus-joern Lange. "The Boolean Formula Value Problem as Formal Language". In: (Jan. 2012). DOI: 10.1007/978-3-642-31644-9_9.

[7] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[8] A.J. Kfoury Robert N. Moll Michael A. Arbib. *An Introduction to Formal Language Theory.* Springer-Verlag, 1988.

[9] Prof. Dr. Thomas Schwentick. *Lecture notes in Grundbegriffe der theoretischen Informatik.* https://www.cs.tu-dortmund.de/nps/de/Studium/Ordnungen_Handbuecher_Beschluesse/Modulhandbuecher/Archiv/Bachelor_LA_GyGe_Inf_Modellv/_Module/INF-BfP-GTI/index.html.