Pina Kolling
piko0011

Summary: Lecture 2

Summary for the chapters *7.1 Complexity classes* and *7.2 Hierarchy problem.* [3]

Complexity classes

Background knowledge:

A complexity class is a set which contains problems with similar complexities. The complexities are examined in regards of a specific ressource, for example time or space. For the problems the most efficient solution/algorithm is analysed.

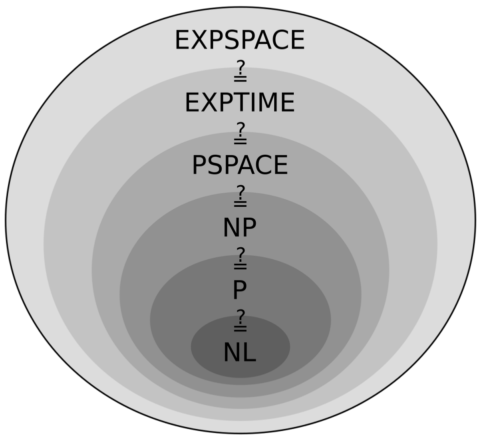

Figure 1: Complexity classes [2]

Usually the complexity depends on the input size. With the asymptotic complexity, classes are build, which are the complexity classes. [4]

Summary:

Parameters of complexity classes: [1, 3]

- **Model of computation:**
  here: multistring Turing Machine
- **Mode of computation:**
  for example: deterministic or non-deterministic (deterministic: the computer will always produce the same output for a given input while going through the same states, non-deterministic: can show different behaviors for the same input)
- **Resources:**
  something *expensive* that the machine uses up, for example: time or space
- **Restrictions/Bound:**
  for example: upper bound, lower bound as a function $f : \mathbb{N} \to \mathbb{N}$

Definition of complexity classes: [3]
The complexity class is the set of all languages which are decided by a Turing Mashine M that is operating in the defined mode and for any input x, M uses at most $f(|x|)$ units of the defined resource.

Definition proper complexity function: [3, 1, 4]

- $f : \mathbb{N} \to \mathbb{N}$
- $\forall n \in \mathbb{N}\ f(n+1) >= f(n)$ (f is non-decreasing)
- It exists a multistring Turing Machine M that fullfills the following conditions with an input of size n:
  - M halts after $O(n + f(n))$ steps (runs in time $O(n + f(n))$)
  - M uses $O(f(n))$ space
  - M maps 1^n to $1^{f(n)}$

Examples of proper functions: [3]

$$\begin{aligned} f(x) &= \log n^2 \\ f(x) &= n \log n \\ f(x) &= n^2 \\ f(x) &= n^3 + 3n \\ f(x) &= 2^n \\ f(x) &= \sqrt{n} \\ f(x) &= n! \end{aligned}$$

If the function f and g are proper, $f + g$, $f \cdot g$ and 2^g are proper, too.

Definition precise Turing Machine: [3, 1]
A multistring Turing Machine M is called a precise Turing Machine, if there are functions f and g such that, for every input x of length n, M stops after exactly $f(n)$ steps with exactly $g(n)$ blanks on strings $2, ..., k$.

If M is a precise Turing Machine and f is a proper complexity functon such that, M decides a language in $f(n)$, then there exists a precise Turing Machine M' of the same type as M which decides the same language in $O(f(n))$.

Complexity classes: [3, 1]

Class name	Description
TIME(f)	deterministic time
SPACE(f)	deterministic space
NTIME(f)	non-deterministic time
NSPACE(f)	non-deterministic space

(f is a proper complexity function)

Sometimes f is not a particular function but a family of function which are parametrized by an integer $k >= 0$.

Class	Function	Description
P	$\bigcup_{k>=0}$TIME(n^k)	Polynomial time
NP	$\bigcup_{k>=0}$NTIME(n^k)	Non-deterministic polynomial time
EXP	$\bigcup_{k>=0}$TIME(2^{n^k})	Exponential time
L	SPACE$\log n$	Logarithmic space
NL	NSPACE($\log n$)	Non-deterministic logarithmic space
PSPACE	$\bigcup_{k>=0}$SPACE(n^k)	Polynomial space
NPSPACE	$\bigcup_{k>=0}$NSPACE(n^k)	Non-deterministic polynomial space

Complement classes: [3, 1]

For a string that is part of a language, one *yes* input needs to be found. For a string to be not part of a lanugage, all the paths must be a *no*. The complement of a language $L \subseteq \Sigma^*$ is the set of all valid inputs that do not belong to L. It is denoted as $\bar{L}$ with $\bar{L} = \Sigma^* - L$. This can be extended to decision problems. The complement of a decision problem A is called A COMPLEMENT. The *yes* and *no* answers on an Turing Machine can be switched to solve the complement problems.

The complement of a complexity class C, the class of all the complements is denoded as coC. The deterministic classes are closed under complement, for examlpe $coP = P$. That does not cound for non-deterministic classes.

Hierarchy problem

References

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

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

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

[4] 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`.