1 Computational theory

Computational task: A problem that needs to be solved.
Computational process: A sequence of actions capable of solving a computational task. In the Theory of Computation is taken to be an algorithm (a finite description of a series of elementary computation steps, where the way the next step is determined must be deterministic).
A computational task can have 0..N solving processes. A task with no solving processes is an unsolved task. Distinct processes can solve the same task in different ways and some of them can be unacceptable (i.e. requiring too much time or space).

1.1 Asymptotic notation

$f : \mathbb{N} \to \mathbb{N}$, $g : \mathbb{N} \to \mathbb{N}$

- f is $\mathbf{O}(\mathbf{g})$ if it grows asymptotically less or as much as g ($\exists c \in \mathbb{R}^+ : f(n) \leq c * g(n)$ for sufficiently large n)
- f is $\mathbf{\Omega}(\mathbf{g})$ if it grows asymptotically more or as much as g ($\exists c \in \mathbb{R}^+ : f(n) \geq c * g(n)$ for sufficiently large n)
- f is $\theta(\mathbf{g})$ if it grows asymptotically in the same way (f is both $O(g)$ and $\Omega(g)$)

2 Turing Machine

Turing Machine (TM) $\mathcal{M} = (\Gamma, Q, \delta)$ with k tapes ($k-1$ of them R/W).
Alphabet $\Gamma = \{\square, \triangleright, 0, 1, \dots\}$, a finite set of symbols that can be found in the tapes.
Finite set of **states** $Q = \{Q_{init}, Q_{halt}, \dots\}$.
Transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}$.

Universal Turing Machine (UTM) $\mathcal{U}$: $\forall x, \alpha \in \{0,1\}^*$ (where α represents the TM $\mathcal{M}_\alpha$):

- $\mathcal{U}(x, \alpha) = \mathcal{M}_\alpha(x)$
- $\mathcal{M}_\alpha(x)$ halts within T steps $\Rightarrow \mathcal{U}(x, \alpha)$ halts within $cT\log(T)$ steps (c depends only on $\mathcal{M}_\alpha$, not on x)

Given $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$, these are equivalent:

- f is **computable in time** T.
- A TM $\mathcal{M}$ computes f in time T.
- $\mathcal{M}$ returns $f(x)$ on input x in a number of steps smaller or equal to $T(|x|) \forall x \in \{0,1\}^*$.

A **language** $\mathcal{L}_f \subseteq \{0,1\}^*$ is decidable in time T if and only if f is computable in time T.
A function $T : \mathbb{N} \to \mathbb{N}$ is **time-constructible** if the function itself can be computed on a Turing machine (i.e. $\forall x \in \{0,1\}^*$ $x \to \lfloor T(|x|) \rfloor$ is computable).

3 Rice's theorem

All non-trivial extensional properties of a program (a Turing Machine) are undecidable. To demonstrate that a property P is undecidable we have to:

1. Show that P is non-trivial. This is accomplished by (both the steps are needed):
   (a) demonstrating that $P \neq \emptyset$. To do this is sufficient to describe a TM $\mathcal{M}$ with the property P
   (b) finding a TM $\mathcal{M}$ which has not the property P
2. Show that P is extensional: Assume $L(\mathcal{M}) = L(\mathcal{N})$ and $\mathcal{M} \in P$. Then demonstrate that $\mathcal{M} \in P$.

4 Complexity classes

$T : \mathbb{N} \to \mathbb{N}$; c constant.
$\mathcal{L} \in \mathbf{DTIME}(T(n)) \iff \exists \mathcal{M}$ deterministic TM deciding the language and running in time $n \to c \cdot T(n)$.
$\mathcal{L} \in \mathbf{NDTIME}(T(n)) \iff \exists \mathcal{M}$ non-deterministic TM deciding the language and running in time $n \to c \cdot T(n)$.

4.1 Polynomial time computable problems

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(\mathbf{n^c})$$

4.2 Exponential time computable problems

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(\mathbf{2^{n^c}})$$

4.3 Between feasible and unfeasible

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NDTIME}(n^c) = \bigcup \mathcal{L} : \mathcal{L} = \left\{ x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)} . \mathcal{M}(\llcorner x, y \lrcorner) = 1 \right\}$$

$p : \mathbb{N} \to \mathbb{N}$ must be a polynomial function. y can be said **certificate** for $\mathcal{L}$. $\mathcal{M}$ can be said **verifier** for $\mathcal{L}$.
A language $\mathcal{H} \subseteq \{0,1\}^*$ is said to be:

- **NP-hard** if $\forall \mathcal{L} \in \mathbf{NP}, \mathcal{L} \leq_p \mathcal{H}$. This means that it is at least as hard as any language in **NP**. Simplifying it means that it cannot be too easy. It could also be un-computable, **NP-complete** or outside **NP**.
- **NP-complete** if $\mathcal{H} \in \mathbf{NP}$ is **NP-hard**. Note that **NP-hard**ness does not imply **NP-complete**ness as a **NP-hard** language may be un-computable or outside **NP**.

Note that:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \qquad \mathbf{NP\text{-}complete} \subseteq \mathbf{EXP} \qquad \mathbf{P} \subsetneq \mathbf{EXP}$$

- $\mathcal{L} \in \mathbf{P}$ is **NP-hard** $\Rightarrow \mathbf{P} = \mathbf{NP}$.
- $\mathcal{L} \in \mathbf{P}$ is **NP-complete** $\iff \mathbf{P} = \mathbf{NP}$.

No such language has been found and $\mathbf{P} = \mathbf{NP}$ has not yet been proven (famous **P vs NP** problem).

4.3.1 SAT

A **kCNF** (k-Conjunctive Normal Form) is a propositional formula which is a conjunction of disjunctions ("clauses") which contain at most $k \in \mathbb{N}$ literals.
Cook-Levin Theorem: The following languages are **NP-complete**:

- $SAT = \{\lfloor F \rfloor | F \text{ is a satisfiable CNF}\}$
- $3SAT = \{\lfloor F \rfloor | F \text{ is a satisfiable 3CNF}\}$

A problem can be deduced to be **NP-complete** by demonstrating that it is NP and reducing it to an existing **NP-complete** problem. For example Maximum Independent Set (INDSET) and ILP can be reduced to SAT and are **NP-complete**.