Appendix A

Glossary of Complexity Classes

Summary: This glossary includes self-contained definitions of most complexity classes mentioned in the book. Needless to say, the glossary offers a very minimal discussion of these classes and the reader is referred to the main text for further discussion. The items are organized by topics rather than by alphabetic order. Specifically, the glossary is partitioned into two parts, dealing separately with complexity classes that are defined in terms of algorithms and their resources (i.e., time and space complexity of Turing machines) and complexity classes defined in terms of non-uniform circuits (and referring to their size and depth). The algorithmic classes include time-complexity based classes (such as $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$) and the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$. The non-uniform classes include the circuit classes $\mathcal{P}/\mathrm{poly}$ as well as $\mathcal{NC}^k$ and $\mathcal{AC}^k$.

Definitions (and basic results) regarding many other complexity classes are available at the constantly evolving *Complexity Zoo* [1].

A.1 Preliminaries

Complexity classes are sets of computational problems, where each class contains problems that can be solved with specific computational resources. To define a complexity class one specifies a model of computation, a complexity measure (like time or space), which is always measured as a function of the input length, and a bound on the complexity (of problems in the class).

We follow the tradition of focusing on decision problems, but refer to these problems using the terminology of promise problems (see Section 2.4.1). That is, we will refer to the problem of distinguishing inputs in Π_{yes} from inputs in Π_{no},

and denote the corresponding decision problem by $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$. Standard decision problems are viewed as a special case in which $\Pi_{\text{yes}} \cup \Pi_{\text{no}} = \{0,1\}^*$, and the standard formulation of complexity classes is obtained by postulating that this is the case. We refer to this case as the case of a trivial promise.

The prevailing model of computation is that of Turing machines. This model captures the notion of (uniform) algorithms (see Section 1.2.3). Another important model is the one of non-uniform circuits (see Section 1.2.4). The term uniformity refers to whether the algorithm is the same one for every input length or whether a different "algorithm" (or rather a "circuit") is considered for each input length.

We focus on natural complexity classes, obtained by considering natural complexity measures and bounds. Typically, these classes contain natural computational problems (which are defined in Appendix G). Furthermore, almost all of these classes can be "characterized" by natural problems, which capture every problem in the class. Such problems are called complete for the class, which means that they are in the class and every problem in the class can be "easily" reduced to them, where "easily" means that the reduction takes less resources than whatever seems to be requires for solving each individual problem in the class. Efficient algorithm for a complete problem implies an algorithm of similar efficiency for *all* problems in the class.

Organization: The glossary is organized by topics (rather than by alphabetic order of the various items). Specifically, we partition the glossary to classes defined in terms of algorithmic resources (i.e., time and space complexity of Turing machines) and classes defined in terms of circuit (size and depth). The former include the time-complexity classes $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$ as well as the space-complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$, which are all reviewed in Section A.2. The latter include the circuit classes $\mathcal{P}$/poly as well as $\mathcal{NC}^k$ and $\mathcal{AC}^k$, which are reviewed in Section A.3.

A.2 Algorithm-based classes

The two main complexity measures considered in the context of (uniform) algorithms are the number of steps taken by the algorithm (i.e., its time complexity) and the amount of "memory" or "work-space" consumed by the computation (i.e., its space complexity). We review the time complexity based classes $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{ZPP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$ as well as the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$.

By prepending the name of a complexity class (of decision problems) with the prefix "co" we mean the class of complement problems; that is, the problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in co$\mathcal{C}$ if and only if $(\Pi_{\text{no}}, \Pi_{\text{yes}})$ is in $\mathcal{C}$. Specifically, deciding membership in the set S is in the class co$\mathcal{C}$ if and only if deciding membership in the set $\{0,1\}^* \setminus S$ is in the class $\mathcal{C}$. Thus, the definition of co$\mathcal{NP}$ and co$\mathcal{RP}$ can be easily derived from the definitions of $\mathcal{NP}$ and $\mathcal{RP}$, respectively. Complexity classes defined in terms of symmetric acceptance criteria (e.g., deterministic and two-sided error randomized classes) are trivially closed under complementation (e.g., co$\mathcal{P} = \mathcal{P}$

and co$\mathcal{BPP} = \mathcal{BPP}$) and so we do not present their "co"-classes. In other cases (most notably $\mathcal{NL}$ and $\mathcal{SZK}$), the closure property is highly non-trivial and we comment about it.

A.2.1 Time complexity classes

We start with classes that are closely related to polynomial-time computations (i.e., $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$ and $\mathcal{ZPP}$), and latter consider the classes $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$.

A.2.1.1 Classes closely related to polynomial time

The most prominent complexity classes are $\mathcal{P}$ and $\mathcal{NP}$, which are extensively discussed in Section 2.1. We also consider classes related to randomized polynomial-time, which are discussed in Section 6.1.

P and NP. The class $\mathcal{P}$ consists of all decision problem that can be solved in (deterministic) polynomial-time. A decision problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{NP}$ if there exists a polynomial p and a (deterministic) polynomial-time algorithm V such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ there exists $y \in \{0,1\}^{p(|x|)}$ such that $V(x,y) = 1$.
2. For every $x \in \Pi_{\text{no}}$ and every $y \in \{0,1\}^*$ it holds that $V(x,y) = 0$.

A string y satisfying Condition 1 is called an NP-witness (for x). Clearly, $\mathcal{P} \subseteq \mathcal{NP}$.

Reductions and NP-completeness (NPC). A problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ is polynomial-time reducible to it, where polynomial-time reducibility is defined and discussed in Section 2.2. Loosely speaking, a polynomial-time reduction of problem Π to problem Π' is a polynomial-time algorithm that solves Π by making queries to a subroutine that solves problem Π', where the running-time of the subroutine is not counted in the algorithm's time complexity. Typically, NP-completeness is defined while restricting the reduction to make a single query and output its answer. Such a reduction, called a Karp-reduction, is represented by a polynomial-time computable mapping that maps yes-instances of Π to yes-instances of Π' (and no-instances of Π to no-instances of Π'). Hundreds of NP-complete problems are listed in [81].

Probabilistic polynomial-time (BPP, RP and ZPP). A decision problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{BPP}$ if there exists a probabilistic polynomial-time algorithm A such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ it holds that $\Pr[A(x)=1] \geq 2/3$.
2. For every $x \in \Pi_{\text{no}}$ it holds that $\Pr[A(x)=0] \geq 2/3$.

That is, the algorithm has two-sided error probability (of 1/3), which can be further reduced by repetitions. We stress that due to the two-sided error probability of $\mathcal{BPP}$, it is not known whether or not $\mathcal{BPP}$ is contained in $\mathcal{NP}$. In contrast to $\mathcal{BPP}$, there are one-sided and zero-error classes, denoted $\mathcal{RP}$ and $\mathcal{ZPP}$, respectively. A problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{RP}$ if there exists a probabilistic polynomial-time algorithm A such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ it holds that $\mathsf{Pr}[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\text{no}}$ it holds that $\mathsf{Pr}[A(x)=0] = 1$.

Again, the error probability can be reduced by repetitions, and thus $\mathcal{RP} \subseteq \mathcal{BPP} \cap \mathcal{NP}$. A problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{ZPP}$ if there exists a probabilistic polynomial-time algorithm A, which may output a special ("don't know") symbol $\perp$, such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ it holds that $\mathsf{Pr}[A(x) \in \{1, \perp\}] = 1$ and $\mathsf{Pr}[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\text{no}}$ it holds that $\mathsf{Pr}[A(x) \in \{0, \perp\}] = 1$ and $\mathsf{Pr}[A(x)=0] \geq 1/2$.

Note that $\mathcal{P} \subseteq \mathcal{ZPP} = \mathcal{RP} \cap \text{co}\mathcal{RP}$. When defined in terms of promise problems, these classes have complete problems (w.r.t Karp-reductions), but the same is not known when considering only standard decision problems (with trivial promise).

The counting class $\#\mathcal{P}$. Functions in $\#\mathcal{P}$ count the number of solutions to an NP-type search problem (or, equivalently, the number of NP-witnesses for a yes-instance of a decision problem in $\mathcal{NP}$). Formally, a function f is in $\#\mathcal{P}$ if there exists a polynomial p and a (deterministic) polynomial-time algorithm V such that $f(x) = |\{y \in \{0,1\}^{p(|x|)} : V(x,y)=1\}|$. Indeed, p and V are as in the definition of $\mathcal{NP}$, and it follows that deciding membership in the set $\{x : f(x) \geq 1\}$ is in $\mathcal{NP}$. Clearly, $\#\mathcal{P}$ problems are solvable in polynomial space. Surprisingly, the permanent of positive integer matrices is $\#\mathcal{P}$-complete (i.e., it is in $\#\mathcal{P}$ and any function in $\#\mathcal{P}$ is polynomial-time reducible to it).

A.2.1.2 Other time complexity classes

The classes $\mathcal{E}$ and $\mathcal{EXP}$ corresponding to problems that can be solved (by a deterministic algorithm) in time $2^{O(n)}$ and $2^{\text{poly}(n)}$, respectively, for n-bit long inputs. Clearly, $\mathcal{NP} \subseteq \mathcal{EXP}$.

In general, one may define a complexity class for every time bound and every type of machine (i.e., deterministic, probabilistic and non-deterministic), but polynomial and exponential bounds seem most natural and very robust. Another robust type of time bounds that is sometimes used is quasi-polynomial time (i.e., $\widetilde{\mathcal{P}}$ denotes the class of problems solvable by deterministic machines of time complexity $\exp(\text{poly}(\log n))$). Another class that occurs is $\mathcal{NEXP}$ the class of problems that can be solved by a non-deterministic machine in $2^{\text{poly}(n)}$ steps.[1]

[1] Alternatively, analogously to the definition of $\mathcal{NP}$, a problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{NEXP}$

The Polynomial-time hierarchy, $\mathcal{PH}$. For any natural number k, the k^{th} level of the polynomial-time hierarchy consists of problems $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ such that there a polynomial p and a polynomial-time algorithm V that satisfies the following two conditions:

1. For every $x \in \Pi_{\text{yes}}$ there exists $y_1 \in \{0,1\}^{p(|x|)}$ such that for every $y_2 \in \{0,1\}^{p(|x|)}$ there exists $y_3 \in \{0,1\}^{p(|x|)}$ such that for every $y_4 \in \{0,1\}^{p(|x|)}$ ... (going for k quantifier alternations) ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 1$.

2. For every $x \in \Pi_{\text{no}}$, the above condition does not hold. That is, for every $y_1 \in \{0,1\}^{p(|x|)}$ there exists $y_2 \in \{0,1\}^{p(|x|)}$ such that for every $y_3 \in \{0,1\}^{p(|x|)}$ there exists $y_4 \in \{0,1\}^{p(|x|)}$ ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 0$.

Such a problem Π is said to be in Σ_k (and $\Pi_k \stackrel{\text{def}}{=} \text{co}\Sigma_k$). Indeed, $\mathcal{NP} = \Sigma_1$ corresponds to the special case where $k = 1$. Interestingly, $\mathcal{PH}$ is polynomial-time reducible to $\#\mathcal{P}$.

A.2.2 Space complexity

When defining space-complexity classes, one counts *only* the space consumed by the actual computation, and *not* the space occupied by the input and output. This is formalized by postulating that the input is read from a read-only device (resp., the output is written on a write-only device). Four important classes of decision problems are defined below.

- The class $\mathcal{L}$ consists of problems solvable in logarithmic space. That is, a problem Π is in $\mathcal{L}$ if there exists a standard (i.e., deterministic) algorithm of logarithmic space-complexity for solving Π. This class contains some simple computational problems (e.g., matrix multiplication), and arguably captures the most space-efficient computations. Interestingly, $\mathcal{L}$ contains the problem of deciding connectivity of (undirected) graphs.

- Classes of problems solvable by randomized algorithms of logarithmic space-complexity include $\mathcal{RL}$ and $\mathcal{BPL}$, which are defined analogously to $\mathcal{RP}$ and $\mathcal{BPP}$. That is, $\mathcal{RL}$ corresponds to algorithms with one-sided error probability, whereas $\mathcal{BPL}$ allows two-sided error.

- The class $\mathcal{NL}$ is the non-deterministic analogue of $\mathcal{L}$, and is traditionally defined in terms of non-deterministic machines of logarithmic space-complexity.[2] The class $\mathcal{NL}$ contains the problem of deciding whether there exists a directed

---

if there exists a polynomial p and a polynomial-time algorithm V such that the two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ there exists $y \in \{0,1\}^{2^{p(|x|)}}$ such that $V(x,y) = 1$.
2. For every $x \in \Pi_{\text{no}}$ and every $y \in \{0,1\}^*$ it holds that $V(x,y) = 0$.

[2] See further discussion of this definition in Section 5.3. In particular, note that we refrained here from presenting a definition analogous to the definition of $\mathcal{NP}$.

path between two given vertexes in a given directed graph. In fact, the latter problem is complete for the class (under logarithmic-space reductions). Interestingly, co$\mathcal{NL}$ equals $\mathcal{NL}$.

- The class $\mathcal{PSPACE}$ consists of problems solvable in polynomial space. This class contains very difficult problems, including the computation of winning strategies for any "efficient 2-party games" (see Section 5.4).

Clearly, $\mathcal{L} \subseteq \mathcal{RL} \subseteq \mathcal{NL} \subseteq \mathcal{P}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP}$.

A.3 Circuit-based classes

We refer the reader to Section 1.2.4 for a definition of Boolean circuits as computing devices. The two main complexity measures considered in the context of (non-uniform) circuits are the number of gates (or wires) in the circuit (i.e., the circuit's size) and the length of the longest directed path from an input to an output (i.e., the circuit's depth).

Throughout this section, when we talk of circuits, we actually refer to families of circuits containing a circuit for each instance length, where the n-bit long instances of the computational problem are handled by the n^{th} circuit in the family. Similarly, the size and depth of the circuit actually refers to the family of circuits and to (the dependence on n of) the size and depth of the n^{th} circuit in the family.

General polynomial-size circuits (P/poly). The main motivation for the introduction of complexity classes based on (non-uniform) circuits is the development of lower-bounds. For example, the class of problems solvable by polynomial-size circuits, denoted $\mathcal{P}$/poly, is a super-set of $\mathcal{P}$ (because it clearly contains $\mathcal{P}$ as well as deciding membership in any subset of $\{1\}^*$, whereas there exists such sets that represents decision problems that are not solvable (i.e., by any uniform algorithm)). Thus, showing that $\mathcal{NP}$ is not contained in $\mathcal{P}$/poly would imply $\mathcal{P} \neq \mathcal{NP}$. For further discussion see Appendix B.2. An alternative definition of $\mathcal{P}$/poly in terms of "machines that take advice" is provided in Section 3.1.2. We also mention that if $\mathcal{NP} \subset \mathcal{P}$/poly then $\mathcal{PH} = \Sigma_2$.

The subclasses AC0 and TC0. The class $\mathcal{AC}^0$, discussed in Appendix B.2.3, consists of problems solvable by constant-depth polynomial-size circuits of *unbounded fan-in*. The analogue class that allows also (unbounded fan-in) majority-gates (or, equivalently, threshold-gates) is denoted $\mathcal{TC}^0$.

The subclasses AC and NC. Turning back to the standard basis (of $\neg$, $\vee$ and $\wedge$), for any non-negative integer k, we denote by $\mathcal{NC}^k$ (resp., $\mathcal{AC}^k$) the class of problems solvable by polynomial-size circuits of *bounded fan-in* (resp., unbounded fan-in) having depth $O(\log^k n)$, where n is the input length. Clearly, $\mathcal{NC}^k \subseteq \mathcal{AC}^k \subseteq \mathcal{NC}^{k+1}$. A commonly referred class is $\mathcal{NC} \stackrel{\text{def}}{=} \cup_{k \in \mathbb{N}} \mathcal{NC}^k$.

We mention that the class $\mathcal{NC}^2 \supseteq \mathcal{NL}$ is the habitat of most natural computational problems of Linear Algebra: solving a linear system of equations as well as computing the rank, inverse and determinant of a matrix. The class $\mathcal{NC}^1$ contains all symmetric functions, regular languages as well as word problems for finite groups and monoids. The class $\mathcal{AC}^0$ contains all properties of finite objects expressible by first-order logic.

Uniformity. The above classes make no reference to the complexity of constructing the adequate circuits, and it is plausible that there is no effective way of constructing these circuits (e.g., as in case of circuits that trivially solve undecidable problem regarding unary instances). A minimal notion of constructibility of such (polynomial-size) circuits is the existence of a polynomial time algorithm that given 1^n produces the n^{th} relevant circuit (i.e., the circuit that solves the problem on instances of length n). Such a notion of constructibility means that the family of circuits is "uniform" in some sense (rather than consisting of circuits that have no relation between one another). Stronger notions of uniformity (e.g., log-space constructibility) are more adequate for subclasses such as AC and NC. We mention that log-space uniform NC circuits correspond to parallel algorithms that use polynomially many processors and run in polylogarithmic time.