Appendix A

Mathematical background

This final chapter collects various definitions and facts used in the book. Part of the material is included here to help the reader recall some less commonly used notions, other issues are mentioned just to fix notation and terminology. Of course, not everything could be included, and some choice had to be made. For instance, we omitted definitions related to Turing Machines, a part of standard curriculum, because we only refer to Turing machines on a fairly high level of abstraction. On the other hand, we included Kleene's definition of recursive functions, as we believe it may be less familar to many computer science students. In general, most of the contents of this chapter is probably well known to the reader. We recommend to consult it only in case of doubt, and this is why we placed it at the end.

A.1. Set theory

Our notation is quite standard. For instance we write $\cup, \cap, -, \varnothing$ for set-theoretic union, intersection, difference and empty set, respectively. The sum (resp. intersection) of a family $\mathcal{A}$ of sets is written as $\bigcup \mathcal{A}$ (resp. $\bigcap \mathcal{A}$), e.g. $\bigcup\{X, Y\} = X \cup Y$. The symbols $\in$ and $\subseteq$ denote set membership and set inclusion. The powerset of A (the set of all subsets of A) is denoted by $\mathcal{P}(A)$, and we write $\{a \in A \mid W(a)\}$ for the set of all elements of A satisfying $W(a)$. Also $\{a, b, c\}$ is the set of (at most three) elements a, b and c. Ordered pairs are written as $\langle a, b\rangle$ and the Cartesian product of A and B is $A \times B$. The Cartesian power $A \times \cdots \times A$ with n coordinates is abbreviated as A^n.

We often write $\vec{a}$ to denote a sequence of the form $a_1, \ldots, a_n$. The vector notation and the symbol $\in$ are used quite liberally, for instance we may write $a \in \vec{a}$ when a is one of $a_1, \ldots, a_n$, or $\vec{a} \in A$ when all $a_1, \ldots, a_n$ are in A. At various occasions we also find it convenient to confuse a sequence $a_1, \ldots, a_n$ with the set $\{a_1, \ldots, a_n\}$.

We have the following notation for the commonly used sets of numbers: $\mathbb{N}$ is the set of natural numbers, including zero, $\mathbb{Q}$ is the set of rationals and $\mathbb{R}$ is the set of reals. We write (a, b) for an open segment of $\mathbb{R}$, i.e. $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$. The set of all words (strings) over an alphabet A is denoted by A^*.

Relations and functions. Relations are identified with subsets of the appropriate Cartesian products. For instance, a binary relation r on A is a subset of A^2.

Such a relation is:

reflexive	iff	$\langle a, a\rangle \in r$, for all $a \in A$;
symmetric	iff	$\langle a, b\rangle \in r$ implies $\langle b, a\rangle \in r$;
anti-symmetric	iff	$\langle a, b\rangle, \langle b, a\rangle \in r$ implies $a = b$;
transitive	iff	$\langle a, b\rangle, \langle b, c\rangle \in r$ implies $\langle a, c\rangle \in r$.

The *transitive closure*of a relation r on A is the smallest (with respect to inclusion) transitive relation containing r. Similar terminology applies for other properties of relations. An *equivalence relation* is one which is reflexive, transitive and symmetric. An *equivalence class* of an equivalence relation r in A, determined by $a \in A$, is the set $[a]_r = \{b \in A \mid \langle a, b\rangle \in r\}$. Any element of an equivalence class can be called a *representative* of the class. The set of all equivalence classes is called the *quotient set* and denoted by $A/_r$.

A function $f : A \to B$ is defined as a relation $f \subseteq A \times B$, such that for all $a \in A$ there is exactly one $b \in B$ with $\langle a, b\rangle \in f$. The set A is then called the *domain* of f and denoted by $\mathrm{dom} f$. The *range* of f is the set $\mathrm{rg} f = \{b \in B \mid \exists a \in A(f(a) = b)\}$. If $f : A \to B$ and $C \subseteq A$ then the *image* of C under f is the set $f(C) = \{f(a) \mid a \in \mathcal{C}\}$. Iterated composition of a function $f : A \to A$ with itself is denoted by f^i. A *partial function from A to B* is an arbitrary function $f : A' \to B$, where $A' \subseteq A$. If $A' = A$ we may say that f is *total*.

We often write $f(a \mapsto b)$ for a function defined as follows:

$$f(a \mapsto b)(x) = \begin{cases} b, & \text{if } x = a; \\ f(x), & \text{otherwise.} \end{cases}$$

The notion of a finite Cartesian product is generalized as follows. Given a family of sets A_t for $t \in T$ we define $\Pi_{t\in T} A_t$ as the set of all functions f such that $\mathrm{dom} f = T$ and $f(t) \in A_t$ for all $t \in T$. If all A_t are equal to some A then $\Pi_{t\in T} A_t$ becomes the set of all functions from T to A, often denoted by A^T. Observe that $A^{\varnothing}$ (which is identified with A^0) is a one-element set. Thus a nullary function $f : A^0 \to B$ is naturally identified with an element of B (a constant).

As we said, relations are typically understood as sets of pairs. But it is sometimes convenient to identify a relation r with a function ranging over $\{0, 1\}$ so that we may write $r(\vec{a}) = 1$ iff $\vec{a} \in r$. Observe that this interpretation of relations suggests the following identification: a nullary relation on an arbitrary set is simply a (classical) truth value.

MULTISETS. A fundamental property of sets is *extensionality*: If sets A and B have the same elements then they are equal (A and B are names of the same object). But sometimes it is convenient to consider collections where a member can occur many times. For this purpose one introduces the notion of a *multiset*, which is like a set with repetitions of elements. Formally, a multiset of elements of a set Ψ is defined as a function $\Gamma : \Psi \to \mathbb{N}$. We say that $\psi \in \Psi$ is an *element* of Γ, when $\Gamma(\psi) > 0$. If $\Gamma(\psi) = n$ then we say that ψ occurs n times in Γ. The notation $\varnothing$, $\{\psi, \psi\}$, etc. applies to multisets with the obvious meaning. The union of two multisets Γ' and Γ'' is defined by the equation

$$(\Gamma' \sqcup \Gamma'')(\psi) = \Gamma'(\psi) + \Gamma''(\psi).$$

More about multisets can be found e.g. in Chapter 2 of [21].

ORDERED SETS. A binary relation $\leq$ on A is a *partial order* iff it is transitive, reflexive and anti-symmetric. The pair $\langle A, \leq\rangle$ is then called a *partially ordered set*.

This terminology is used quite liberally, e.g. one can say that A itself is a partially ordered set or a partial order. A partial order is called *total* or *linear* if each two elements a, b are *comparable*, i.e. either $a \leq b$ or $b \leq a$ always holds.

Observe that each family of sets $\mathcal{F}$ is partially ordered by inclusion. We adopt the convention that whenever we talk about a family of sets, and we mention any order-related notions (e.g. lower and upper bounds defined below) we always mean the partial order given by set inclusion.

An element $a \in A$ is *maximal* (resp. *minimal*) in A iff $a \leq b$ (resp. $a \geq b$) implies $a = b$, for all $b \in A$. Observe that the top (resp. bottom) element is always maximal (resp. minimal), but not conversely.

If A is a partial order then a subset $B \subseteq A$ is *upward-closed* (resp. *downward-closed*) iff $a \geq b \in B$ (resp. $a \leq b \in B$) implies $a \in B$. An element a of a partially ordered set A is an *upper* (resp. *lower*) *bound* of $B \subseteq A$ iff $a \geq b$ (resp. $a \leq b$) for all $b \in B$. We say that a is the *supremum* of B (and we write $a = \sup_A B$), when it is the *least upper bound* of B, i.e.

- a is an upper bound of B;
- $a \leq b$, for any other upper bound b of B.

Observe that there is at most one element satisfying these conditions. Dually, we define the *infimum* of a subset B (written $\inf_A B$) as the *greatest lower bound* of B. If A is known from the context, we may omit the subscript $_A$ in the notation $\sup_A B$ and $\inf_A B$. But in some cases the subscript is important. Observe for instance that if $C \subseteq B \subseteq A$ then $\sup_B C \geq \sup_A C$ (assuming both sides exist) but not necessarily $\sup_B C = \sup_A C$.

The reader can easily verify that $\sup_A \varnothing$ (resp. $\inf_A \varnothing$), if it exists, must be the bottom (resp. top) element of A. Note also that if $\mathcal{A} \subseteq \mathcal{P}(X)$ then $\sup_{\mathcal{P}(X)} \mathcal{A} = \bigcup A$ and $\inf_{\mathcal{P}(X)} \mathcal{A} = \bigcap A$.

The following result is often useful, but the proof of it (based on the axiom of choice, see e.g. [212, 290]) is highly non-constructive.

A.1.1. LEMMA (Kuratowski, Zorn). *Let A be partially ordered. If every chain (linearly ordered subset) has an upper bound in A then A has a maximal element.* □

If A and B are partially ordered then a function $f : A \to B$ is *monotone* iff $a_1 \leq a_2$ implies $f(a_1) \leq f(a_2)$. The following result is sometimes called *Knaster-Tarski fixed point theorem:*

A.1.2. THEOREM. *Let A be a* complete lattice (*all subsets have suprema and infima*) *and let $f : A \to A$ be monotone. Define $a_0 = \inf\{a \in A \mid f(a) \leq a\}$. Then a_0 is the least fixed point of f. That is, $f(a_0) = a_0$ and $a_0 \leq b$, whenever $f(b) = b$.*

PROOF. Let $B = \{a \in A \mid f(a) \leq a\}$. We have $f(a_0) \leq f(b) \leq b$, for $b \in B$ and thus $f(a_0) \leq a_0$, i.e. a_0 itself is in B. But then $f(a_0) \in B$, because of monotonicity, so that also $a_0 \leq f(a_0)$. □

A.1.3. REMARK. Define $a_1 = \inf\{a \in A \mid \forall b \in A(b \leq a \to f(b) \leq a)\}$, under the assumptions of Theorem A.1.2. Then $a_1 = a_0$, i.e. we have an alternative definition of the least fixed point. Note however that monotonicity is essential. For instance, let $f : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ be such that $f(A) = A \cup \{\min(\mathbb{N} - A)\}$ for finite A and $f(A) = A - \{\min A\}$ for infinite A. Then $a_0 = \varnothing$, $a_1 = \mathbb{N}$, and f has no fixed point. But observe that $f(a_1) \leq a_1$ holds for arbitrary f.

INDUCTION. A partially ordered set A is *well-founded* iff every non-empty subset of A has a minimal element. Observe that if A and B are well-founded then the

lexicographic order on $A \times B$, given by $(a,b) \leq (a',b')$ iff $a < a'$, or $a = a'$ and $b \leq b'$, is also well-founded. If A is a well-founded set then properties of elements of A can be proved by induction with respect to the ordering.

A.1.4. LEMMA (Principle of induction). *Let $B \subseteq A$, where A is a well-founded set. If $\forall a \in A(\forall b \in A(b < a \to b \in B) \to a \in B)$ then $B = A$.*

PROOF. Otherwise the set $B' = \{a \in A \mid a \notin B\}$ has a minimal element a' satisfying $\forall b \in A(b < a' \to b \in B)$. □

Many proofs by "structural induction" are based on the principle above. To show a statement of the form $\forall x \in X\,(W_x)$ one assigns an element $f(x)$ of a well-founded set A to every $x \in X$. Then the problem is reduced to showing that $B = A$ where $B = \{a \in A \mid \forall x \in X(f(x) = a \to W_x)\}$.

In particular it may happen that a set is well-founded because it has been defined by induction. For instance, the transitive and reflexive closure r^* of a relation r is the least set such that $r \subseteq r^*$ and for $\langle a,b\rangle, \langle b,c\rangle \in r^*$ also $\langle a,c\rangle \in r^*$. This induces a stratification of r^* as the union of an infinite sequence of sets: $r_0 = r$, $r_{n+1} = r_n \cup \{\langle a,c\rangle \mid \langle a,b\rangle, \langle b,c\rangle \in r_n\}$, and we have a well-founded order on r^*:

$$\langle a,b\rangle \leq \langle c,d\rangle \quad \text{iff} \quad \forall n(\langle c,d\rangle \in r_n \to \langle a,b\rangle \in r_n).$$

That is, $\langle a,b\rangle \leq \langle c,d\rangle$ holds when $\langle a,b\rangle$ falls into the union r^* at an earlier (or the same) stage than $\langle c,d\rangle$.

The above justifies a proof method called *induction with respect to the definition of* r^*. To show that a property $P(a,b)$ holds for each pair $\langle a,b\rangle \in r^*$ one first proves $P(a,b)$ for all $\langle a,b\rangle \in r$ and then makes the induction step by showing that $P(a,b)$ and $P(b,c)$ implies $P(a,c)$. Similar machinery occurs whenever one talks about induction with respect to various inductive definitions.

TOPOLOGICAL SPACES. A *topological space* is a pair $\mathcal{T} = \langle T, \mathcal{O}\rangle$, consisting of a set T together with a family $\mathcal{O} \subseteq \mathcal{P}(T)$ of sets, called *open sets* of T, which is required to satisfy the following conditions:

- $\varnothing, T \in \mathcal{O}$;
- $A, B \in \mathcal{O}$ implies $A \cap B \in \mathcal{O}$;
- $\mathcal{R} \subseteq \mathcal{O}$ implies $\bigcup \mathcal{R} \in \mathcal{O}$.

That is, arbitrary unions and finite intersections of open sets must be open. The *interior* of a set $A \subseteq T$, denoted by $\mathrm{Int}(A)$ is the largest open set contained in A (i.e. the union of all open subsets of A). The complement of an open set is called a *closed set*. The *closure* of A is the smallest closed set containing A. For $\mathcal{T} = \langle T, \mathcal{O}\rangle$ we use the notation $\mathcal{O} = \mathcal{O}(\mathcal{T})$, and we often make no distinction between $\mathcal{T}$ and T.

Many topological spaces can be defined using a measure of *distance* between points. Such spaces are called *metric spaces*. Examples of metric spaces are the set $\mathbb{Q}$ of rationals, the set $\mathbb{R}$ of reals, as well as all Euclidean spaces $\mathbb{R}^k$. In all these cases the topology is defined according to a common pattern. Let $\varrho(a,b)$ denote the distance between points a and b. A set A is *open* iff for every $a \in A$ there is an $r > 0$ with $\{b \mid \varrho(a,b) < r\} \subseteq A$. In the space of real numbers, the distance between x and y is $|x - y|$. We say that $a \in \mathbb{R}$ is an *accumulation point* of $A \subseteq \mathbb{R}$ when a is a limit of a sequence x_n of elements of A, and all x_n are different from a.

A.2. Algebra and unification

A *signature* is a family of function and relation symbols, each with a fixed arity (possibly zero). Nullary symbols are called *constants*. If not stated otherwise, signatures are assumed to be finite.

Assume a fixed infinite set $\{a_0, a_1, \ldots\}$ of *individual variables*. An *algebraic term*[1] over a signature Σ, or just *term*, is either an *individual variable*, or an expression of the form $(ft_1 \ldots t_n)$, where f is an n-ary function symbol, and $t_1, \ldots, t_n$ are terms. We usually omit outermost parentheses when writing terms. The symbol $\mathrm{FV}(t)$ denotes the set of all individual variables occurring in a term t. A term is *closed* iff $\mathrm{FV}(t) = \varnothing$.

The formal definition of an algebraic term involves a prefix application of function symbols. Of course, there is a tradition to write some binary function symbols in the infix style, and we normally do it this way. Our most beloved signature is the one that has the (infix) arrow as the only symbol. It is not difficult to see that algebraic terms over this signature can be identified with simple types, or with implicational formulas if you prefer.

In this section we consider only *algebraic signatures*, i.e., signatures without relation symbols. An *algebra* over such a signature $\Sigma = \{f_1, \ldots, f_n\}$, where the arity of each f_i is r_i, is a set A together with functions $f_i^A : A^{r_i} \to A$, i.e. a system $\mathcal{A} = \langle A, f_1^A, \ldots, f_n^A \rangle$. (We often forget about the notational distinction between $\mathcal{A}$ and its domain A.) A subset B of A is a *subalgebra* of A iff B is closed under all operations. That is, whenever $\vec{b} \in B$ then also $f_i^A(\vec{b}) \in B$, for all $i = 1, \ldots, n$ (assuming the appropriate length of $\vec{b}$). If $\langle A, f_1^A, \ldots, f_n^A \rangle$ and $\langle B, f_1^B, \ldots, f_n^B \rangle$ are two algebras of the same signature then a function $h : A \to B$ satisfying the condition $h(f_i^A(\vec{a})) = f_i^B(h(\vec{a}))$ is a *homomorphism*. If, in addition, h is a bijection, and the converse function is also a homomorphism then h is called an *isomorphism* and the two algebras are *isomorphic* to each other.

Unification. A *substitution* is a function from variables to terms which is identity almost everywhere. Such a function S is extended to a function from terms to terms by $S(ft_1 \ldots t_n) = fS(t_1) \cdots S(t_n)$ (i.e. $S(f) = f$, when f is a constant). In case $S(a) = s$ and $S(b) = b$, for all other variables b, one may write $t[a := s]$ for $S(t)$.

If P and R are substitutions then $P \circ R$ is defined by $(P \circ R)(x) = P(R(x))$. A substitution S is an *instance* of another substitution R (written $R \leq S$) iff $S = P \circ R$, for some substitution P.

An *equation* is a pair of terms, written "$t = u$". A *system of equations* is a finite set of equations. Variables occurring in a system of equations are called *unknowns*. A substitution S is a solution of an equation "$t = u$" iff $S(t) = S(u)$ (meaning that $S(t)$ and $S(u)$ is the same term). It is a solution of a system E of equations iff it is a solution of all equations in E. A solution R of a system E is *principal* iff the following equivalence holds for all substitutions S:

$$S \text{ is a solution of } E \quad \text{iff} \quad R \leq S.$$

Thus, for instance, the equation $f(gab)a = fc(fbb)$ has a principal solution S, such that $S(a) = fbb$, $S(b) = b$ and $S(c) = g(fbb)b$ (and many other solutions too), while the equation $f(gab)h = fc(fbb)$, where h is a constant, has no solution. This

[1] Do not confuse algebraic terms with lambda-terms, and do not confuse individual variables with propositional or lambda-variables.

is because no substitution can turn fbb into h, or vice versa. Another example with no solution is $f(gab)a = fa(fbb)$, but this time the reason is different: if S were a solution then $S(a)$ would be a proper subterm of itself.

The problem of determining whether a given system of equations has a solution is called the *unification problem*. It is not difficult to see that there is no loss of generality in considering single equations rather than systems of equations and just one binary function symbol rather than arbitrary signatures (Exercise 3.7). The unification algorithm of Robinson [418] can be found in various textbooks, for instance [21, 349, 474]. We have:

A.2.1. LEMMA. *If a system of equations has a solution then it has a principal one, of size at most exponential in the size of the input.*

A.3. Partial recursive functions

A.3.1. DEFINITION. Let $g : \mathbb{N}^k \to \mathbb{N}$ and $h_1, \ldots, h_k : \mathbb{N}^m \to \mathbb{N}$ be partial functions. The *composition*

$$f(n_1, \ldots, n_m) = g(h_1(n_1, \ldots, n_m), \ldots, h_k(n_1, \ldots, n_m))$$

is defined iff:

- $h_i(n_1, \ldots, n_m)$ is defined, for all $i = 1, \ldots, k$;
- $g(a_1, \ldots, a_k)$ is defined, where $a_i = h_i(n_1, \ldots, n_m)$, for $i = 1, \ldots, k$.

A.3.2. DEFINITION. Let $g : \mathbb{N}^m \to \mathbb{N}$ and $h : \mathbb{N}^{m+2} \to \mathbb{N}$ be partial functions. We say that a partial function $f : \mathbb{N}^{m+1} \to \mathbb{N}$ is defined from g and h by *primitive recursion* iff for all $n, n_1, \ldots, n_m \in \mathbb{N}$,

$$\begin{array}{lcl} f(0, n_1, \ldots, n_m) & = & g(n_1, \ldots, n_m); \\ f(n+1, n_1, \ldots, n_m) & = & h(f(n, n_1, \ldots, n_m), n, n_1, \ldots, n_m). \end{array}$$

The equations above are understood so that the left-hand-side is defined if and only if the right-hand-side is defined.

A.3.3. DEFINITION. If $g : \mathbb{N}^{m+1} \to \mathbb{N}$ is a partial function, then the notation $\mu n[g(n_1, \ldots, n_m, n) = 0]$ stands for the least number n such that

- $g(n_1, \ldots, n_m, n) = 0$.
- $g(n_1, \ldots, n_m, n')$ is defined and not equal to 0, for all $n' \in \{0, \ldots, n-1\}$;

If there is no such n then $\mu n[g(n_1, \ldots, n_m, n) = 0]$ is undefined. The partial function $f : \mathbb{N}^m \to \mathbb{N}$ given by

$$f(n_1, \ldots, n_m) = \mu n[g(n_1, \ldots, n_m, n) = 0]$$

is said to be defined from g by *minimization*.

The following introduces an important subclass of recursive functions.

A.3.4. DEFINITION. The class of *primitive recursive functions* is the smallest class of total numeric functions containing the *initial functions*

(i) *projections:* $p_i^m(n_1, \ldots, n_m) = n_i$ for all $1 \leq i \leq m$;
(ii) *successor:* $s(n) = n + 1$;
(iii) *zero:* $z(n) = 0$.

and closed under composition and primitive recursion. A predicate (set) $P \subseteq \mathbb{N}^k$ is *primitive recursive* iff its characteristic function is primitive recursive.

Most computable functions of everyday use are in fact primitive recursive. (An exception is for instance the *Ackermann function* of Exercise 10.5.) In particular, the following bijective *pairing function*

$$p(m,n) = \frac{(m+n)(m+n+1)}{2} + m,$$

is primitive recursive, as well as the left and right converse functions ℓ and r, satisfying $n = p(\ell(n), r(n))$ for all n. There are more sophisticated examples of primitive recursive functions and predicates. One is as follows. Consider a fixed enumeration M_k of all Turing Machines that are capable of processing an input $\vec{n} \in \mathbb{N}$ and returning an output $m \in \mathbb{N}$. (If a partial function f is computed this way by a Turing machine, we call it *Turing computable* and we say that k is a *Gödel number* of f.) Define *Kleene's T predicate:*

$$T(k,\vec{n},m) \quad \text{iff} \quad M_k \text{ halts on input } \vec{n} \text{ in } r(m) \text{ steps with output } \ell(m).$$

The proof of the following can be found in [109, 269, 337].

A.3.5. Lemma. *The predicate $T(k,\vec{n},m)$ is primitive recursive.*

A.3.6. Definition.

(i) The class of *partial recursive functions* is the smallest class of partial functions containing the initial functions and closed under composition, primitive recursion and minimization.

(ii) A set $A \subseteq \mathbb{N}^k$ is *recursively enumerable* iff it is the domain of some partial recursive function.

(iii) A *recursive function* is a partial recursive function that is total.

(iv) A set $A \subseteq \mathbb{N}^k$ is *recursive* iff its characteristic function is recursive.

It is well-known that (partial) recursive functions coincide with (partial) Turing-computable functions, see e.g. [337].

When we talk about a recursive or partially recursive *function* we often mean a particular algorithm (Turing machine) computing this function. The following notation follows this convention.

A.3.7. Notation. Let f be a partial recursive function computed by the machine M_k. Then the characteristic function of the binary predicate $T(k,\cdot,\cdot)$ is denoted by t_f, that is,

$$t_f(\vec{n},m) = 0 \quad \text{iff} \quad M_k \text{ halts on input } \vec{n} \text{ in } r(m) \text{ steps with output } \ell(m).$$

Clearly, t_f is primitive reccursive, and we have the following result:

A.3.8. Theorem (Kleene's normal form). *Every partial recursive function f can be written as*

$$f(\vec{n}) = \ell(\mu y[t_f(\vec{n},y) = 0]),$$

where t_f and ℓ are primitive recursive functions.

A.3.9. Corollary. *A set $B \subseteq \mathbb{N}$ is recursively enumerable if and only if there is a primitive recursive set $A \subseteq \mathbb{N}^2$ such that*

$$n \in B \quad \textit{iff} \quad \textit{there exists } m \textit{ with } (n,m) \in A.$$

A.4. Decision problems

Of course not every recursively enumerable set is recursive. For instance, the set $\{(k,n) \in \mathbb{N} \mid M_k \text{ halts on input } n\}$ is not, and we usually express this fact by saying that "the halting problem for Turing machines is undecidable". Formally, a *decision problem* is defined as an arbitrary set of numbers, or more generally (and often more conveniently) an arbitrary set of words over a fixed alphabet. Observe that a numerical input is always represented as a word, and a word can easily be encoded as a number. A problem P is *decidable* iff the set P is recursive. Otherwise it is *undecidable*. In practice, the notion of a "decision problem" is used in a more relaxed way. For instance, each of the following questions is a decision problem:

Does a given Turing machine halt on a given input?

Is a given first-order formula a classical tautology?

Here, the input data is not an arbitrary number or an arbitrary word, but a certain finite object (a machine and an input, a formula). This is not a problem, because we can encode Turing machines as words or numbers. A formula is a word itself, so we do not even need to encode it. But not every word is a formula, and a more adequate formulation of the second problem is

Is a given word a well-formed formula that is a classical tautology?

It does not really matter which question we actually ask, because there is a simple algorithm to decide whether a given word is a formula. But consider this one:

Is a given first-order classical tautology an intuitionistic tautology?

The question whether an input word is a correct instance of this problem is no longer trivial. In fact, it is undecidable. There is no effective encoding of first-order classical tautologies as all natural numbers. Thus the "decision problem" above must be classified as ill-formed according to our definition.

A.4.1. EXAMPLE. Consider the following "decision problem": Given a Turing machine which halts on input zero, is the result of the computation also equal to zero? There is a natural algorithm to solve it. Just run the machine and check the output. This algorithm is guaranteed to terminate because... the difficulty is hidden in the definition of legal inputs. To see that the "termination property" of our algorithm is of no actual value, observe the following: *For every computable function f, every solution of the problem above requires more than $f(n)$ steps on some inputs of size n.* In other words, our "decidable" problem has no time complexity!

Indeed, otherwise consider a language L which is computable, but requires time greater than $2^{f(2n)}$, and let L be accepted by a machine M_L. For a given word w, let M^w be a machine which takes an integer input, runs M_L on w, and returns 0 if $w \in L$ and otherwise returns 1. Observe that M^w halts for every input, in particular for zero. To decide if $w \in L$ we now construct M^w (which takes time equal to the length of w plus a constant) and then we ask if the output of M^w is 0. By our assumption, this can be done in time at most proportional to $f(2n)$, contradicting the definition of L.

To show that a particular decision problem P is undecidable, we usually find another problem, which is already known to be undecidable, and we *reduce* it to P.

A.4.2. Definition. A problem P_1 is *reducible* to P_2, written $P_1 \leq P_2$, iff there is a computable function f, such that

$$w \in P_1 \quad \text{iff} \quad f(w) \in P_2,$$

for all inputs w. Clearly, if $P_1 \leq P_2$ and P_1 is undecidable then so is P_2.

Post Correspondence Problem. *Post Correspondence Problem* (PCP) is the following decision problem. Given pairs of words $(w_1, v_1), \ldots, (w_n, v_n)$, find a non-empty sequence of indices $i_1, \ldots, i_k$, such that

$$w_{i_1} w_{i_2} \ldots w_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}.$$

The sequence $i_1, i_2, \ldots, i_k$ is then called a *solution* of the problem. Also the word $w_{i_1} w_{i_2} \ldots w_{i_k}$ is often called a solution. An instance $\{(w_i, v_i) \mid i = 1, \ldots, n\}$ of Post Correspondence Problem is often displayed as:

w_1	w_2	$\ldots$	w_n
v_1	v_2	$\ldots$	v_n

For instance, 1213 is a solution of

a^2	b^2	ab^2
a^2b	ba	b

because $a^2 \cdot b^2 \cdot a^2 \cdot ab^2 = a^2b^2a^3b^2 = a^2b \cdot ba \cdot a^2b \cdot b$, and the system

a^2b	a
a^2	ba^2

has no solution. It is known that PCP is undecidable, see [245, 256].

Two-counter automata. A *two-counter automaton* $\mathcal{A} = \langle Q, q_0, q_f, I \rangle$ consists of a finite set of *states* Q, the distinguished initial state q_0 and final state q_f, and a set of instructions, each of one of the following forms:

(u_i) $(q : c_i := c_i + 1;$ `goto` $p)$;

(d_i) $(q : c_i := c_i - 1;$ `goto` $p)$;

(z_i) $(q :$ `if` $c_i = 0$ `then goto` p `else goto` $r)$;

where $i = 1, 2$ and $q, p, r \in Q$. A *configuration* of $\mathcal{A}$ is a triple $\langle q, m, n \rangle$ consisting of a state $q \in Q$ and numbers $m, n \in \mathbb{N}$, understood as current values of two counters c_1 and c_2. There is one *initial configuration*, namely $\langle q_0, 0, 0 \rangle$, and configurations $\langle q_f, m, n \rangle$ are called *final*.

The meaning of the instructions should be obvious. For configurations $\mathcal{C}_1, \mathcal{C}_2$, we write $\mathcal{C}_1 \to_{\mathcal{A}} \mathcal{C}_2$ when $\mathcal{C}_2$ is obtained from $\mathcal{C}_1$ in one step. (We assume that no (d_i) step is possible when $c_i = 0$, i.e. that the machine gets stuck when trying to subtract 1 from 0.) The notation $\twoheadrightarrow_{\mathcal{A}}$ stands for the transitive and reflexive closure of $\to_{\mathcal{A}}$, and if $\mathcal{C}_1 \twoheadrightarrow_{\mathcal{A}} \mathcal{C}_2$ then $\mathcal{C}_2$ is *reachable* from $\mathcal{C}_1$. The *halting problem* for two-counter automata is to decide if a given automaton *halts*, that is, if a final configuration is reachable from the initial one. This problem is well-known to be undecidable, see e.g. [245, 276].

A.5. Hard and complete

This section recalls a few basic notions from complexity theory. The reader is referred to standard textbooks, like [245, 256], for a more comprehensive discussion.

A.5.1. DEFINITION. The notation LOGSPACE and PTIME refers to the classes of languages (decision problems) solvable by deterministic Turing Machines in logarithmic space[2] and polynomial time, respectively. The class PSPACE consists of problems solvable by Turing Machines (deterministic or not) in polynomial space, and EXPTIME refers to deterministic time $2^{p(n)}$, where $p(n)$ can be any polynomial. We have the following inclusions: LOGSPACE $\subseteq$ PTIME $\subseteq$ PSPACE $\subseteq$ EXPTIME.

A.5.2. DEFINITION. We say that a language L_1 is *reducible* to a language L_2 *in logarithmic space* (or LOGSPACE-*reducible*) when there is a LOGSPACE-computable function f, such that

$$w \in L_1 \quad \text{iff} \quad f(w) \in L_2,$$

for all inputs w. Two languages are LOGSPACE-*equivalent* iff there are LOGSPACE-reductions each way.

That is, to decide if $w \in L_1$ one can ask if $f(w) \in L_2$, and the cost of the translation is only shipping and handling. (Observe that $\log n$ space is exactly what is needed to store the position of the machine head on an input of length n.) Note that a logarithmic space reduction is also a polynomial time reduction.

A.5.3. DEFINITION. We say that a language L is *hard* for a complexity class $\mathcal{C}$, iff every language $L' \in \mathcal{C}$ is reducible to L in logarithmic space. If we have $L \in \mathcal{C}$ in addition, then we say that L is *complete* in the class $\mathcal{C}$, or simply $\mathcal{C}$-*complete*.

UNIFICATION. As mentioned in Section A.2, the unification problem is decidable, and the solution can be of exponential size. In fact, Robinson's algorithm can be optimized to work in polynomial time, provided we only need to check whether a solution exists, and we do not need to *write it down* explicitly, cf. Exercise 3.6. Also the verification whether a *given* solution is correct can be done in polynomial time. The following is from Dwork *et al* [138].

A.5.4. THEOREM. *The unification problem is* PTIME-*complete.*

QUANTIFIED BOOLEAN FORMULAS. A standard example of a problem which is PSPACE-complete is the *classical* validity problem for *second-order propositional formulas* (see Definition 11.1.1), called also *quantified Boolean formulas* (QBF) in this context. A value of a second-order propositional formula under a binary valuation v is defined in a similar way as for ordinary propositional formulas, with the following clauses in addition:

$$[\![\forall p\, \alpha]\!]_v = \max\{[\![\alpha]\!]_{v(p\mapsto 1)}, [\![\alpha]\!]_{v(p\mapsto 0)}\};$$
$$[\![\exists p\, \alpha]\!]_v = \min\{[\![\alpha]\!]_{v(p\mapsto 1)}, [\![\alpha]\!]_{v(p\mapsto 0)}\}.$$

A second-order classical propositional *tautology* is a formula true under all valuations. The *QBF problem* is as follows:

Is a given second-order propositional formula a tautology?

In fact it suffices to only consider *closed* formulas, where each variable is bound by a quantifier. In this case the *validity* problem above is the same as the satisfiability

[2] One counts only the work tapes, not the input or output tapes.

problem for QBF. A closed formula is either a tautology, or it is not satisfiable. Proofs of the following fact can be found in [245, 256].

A.5.5. Theorem. *The QBF problem is* Pspace*-complete.*

Further reading. For more background information we recommend:

- Set theory, Boolean algebras: Davey and Priestley [119], Halmos [212, 213], Kuratowski and Mostowski [290], Monk [353].
- Turing machines, theory of computation and complexity: Boolos, Burgess and Jeffrey [53], Cutland [109], Hopcroft, Motwani and Ullman [245], Jones [256], Kozen [276], Mendelson [337].
- Classical logic: Adamowicz and Zbierski [6], Bell and Machover [39], Huth and Ryan [252], Mendelson [337], Monk [352], van Dalen [113].
- Rewriting, unification (and *all that*): Baader and Nipkow [21], Terese [474].
- Programming languages: Mitchell [349, 350], Pierce [393].