Chapter 1

Preliminaries

1. Basic First Order Logic

1.1. Language. A *first order language* $\mathcal{L}_\Sigma$ is specified by its *signature* Σ, which consists of three disjoint alphabets Pred, Func, and Const of *predicate*, *function*, and *constant symbols*, respectively, and a function

$$\text{arity} : \text{Pred} \cup \text{Func} \to \mathbf{N} \setminus \{0\}$$

that assigns to each predicate and function symbol its (nonzero) number of arguments. In addition, $\mathcal{L}_\Sigma$ has two fixed countable alphabets of free and bound variables

$$\begin{aligned} \text{FreeVar} &= \{a_0, a_1, a_2, \dots\}, \\ \text{BoundVar} &= \{v_0, v_1, v_2, \dots\}, \end{aligned}$$

and the following special symbols:

Boolean connectives: $\to$ (implication), $\neg$ (negation);

Universal quantifier: $\forall$ (for all);

Punctuation marks: ( ) (brackets) and , (comma).

An arbitrary finite string of symbols of the above alphabets is called an *expression*. Certain expressions are called *terms* and *formulas*. Terms are defined by the following inductive clauses:

1. Free variables and constant symbols are terms.
2. If f is a function symbol of arity n, and $t_1, \dots, t_n$ are terms, then the expression $f(t_1, \dots, t_n)$ is a term.

(In this kind of definition it is always implicitly assumed that an expression is a term *only if* it can be inductively constructed by Clauses 1 and 2.)

Formulas are defined as follows:

1. If P is a n-ary predicate symbol, and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula (called *atomic formula*).

2. If A, B are formulas, then so are $(A \to B)$ and $(\neg A)$.

3. If A is a formula, and a is a free variable occurring in A, then, for every bound variable x *not occurring* in A, the expression $(\forall x A^a_x)$ is a formula. (Here A^a_x denotes the result of replacing all occurrences of the variable a in A by x.)

Formulas without any occurrences of quantifiers are called *quantifier free* or *open*. To stress that a variable a occurs in A one usually writes A as $A(a)$ (notice that the parentheses here are not the formal symbols of our language); then A^a_x can be written as $A(x)$. More generally, for a formula $A(b_1, \ldots, b_n)$, in which free variables $b_1, \ldots, b_n$ occur, $A(t_1, \ldots, t_n)$ denotes the result of simultaneous substitution of terms $t_1, \ldots, t_n$ for all occurrences of $b_1, \ldots, b_n$, respectively, in A.

To enhance readability, on practice one adopts various notational conventions, such as omitting superfluous parentheses, using a, b, c instead of a_0, a_1, a_2, etc. Other standard boolean connectives, $\wedge$ (conjunction), $\vee$ (disjunction), $\leftrightarrow$ (equivalence), and the existential quantifier $\exists$ are treated as suitable abbreviations:

$$\begin{aligned} A \vee B &= (\neg A) \to B \\ A \wedge B &= \neg(A \to \neg B) \\ A \leftrightarrow B &= (A \to B) \wedge (B \to A) \\ \exists x A^a_x &= \neg(\forall x(\neg A^a_x)) \end{aligned}$$

1.2. Models. Models are natural semantical structures for a first order language. Let M be a nonempty set. By a *n-ary predicate* on M we mean a subset of $M^n = M \times M \times \cdots \times M$ (n times); an *n-ary function* on M is a function $M^n \to M$. For a n-ary predicate Q one often writes $Q(x_1, \ldots, x_n)$ instead of $\langle x_1, \ldots, x_n \rangle \in Q$; similarly, $f(x_1, \ldots, x_n)$ means $f(\langle x_1, \ldots, x_n \rangle)$.

A *model* of a signature Σ is a nonempty set M together with a mapping that assigns to each predicate symbol P of Σ a predicate P_M on M of the same arity, to each function symbol f a function f_M on M of the same arity, and to each constant symbol c an element $c_M \in M$. Such a mapping is called the *interpretation* of Σ in M. The set M is called the *universe* or the *domain* of the model. We shall denote the model and its universe by the same letter M and often identify the two concepts in free speach.

REMARK 1.1. Mathematical standards provide certain defaults for the choice of the signature and its interpretation in a model. For example, in the context of integers, '+' denotes the usual addition operation, '=' is the equality relation, '0' is 0. So, $(\mathbf{Z}, =, +, 0)$ denotes the model with the universe $\mathbf{Z}$, the binary predicate $=$, the binary function $+$, and the constant 0. The subscript $_{\mathbf{Z}}$ is omitted

everywhere without causing confusion. We shall also follow other standard notational conventions, such as, e.g., writing $a_1 = a_2$ instead of $=(a_1, a_2)$, and $a_1 + a_2$ instead of $+(a_1, a_2)$.

Let M be a model. By an *evaluation* of a subset $V \subseteq \mathrm{FreeVar}$ in M we mean an arbitrary function $e : V \to M$. The evaluation is uniquely extended in a natural way to a function from the set of terms, whose free variables are contained in V, to M according to the following rules:

1. $e(c) = c_M$, for any constant symbol c;
2. $e(f(t_1, \ldots, t_n)) = f_M(e(t_1), \ldots, e(t_n))$, for any n-ary function symbol f and terms $t_1, \ldots, t_n$.

For formulas A, whose free variables are in V, the relation $M \vDash_e A$, to be read "formula A is *valid* under the evaluation e *in* M," or "the evaluation e *satisfies* A in M," is defined by the following inductive clauses:

1. $M \vDash_e P(t_1, \ldots, t_n) \iff P_M(e(t_1), \ldots, e(t_n))$, if $P(t_1, \ldots, t_n)$ is an atomic formula;
2. $M \vDash_e (B \to C) \iff M \nvDash_e B$ or $M \vDash_e C$;
3. $M \vDash_e (\neg B) \iff M \nvDash_e B$;
4. $M \vDash_e (\forall x A^a_x) \iff M \vDash_{e'} A$, for every evaluation $e' : V \cup \{a\} \to M$ such that $e'(b) = e(b)$ for all $b \in V \setminus \{a\}$.

   (Notice that the free variable a does not occur in $(\forall x A^a_x)$, but may possibly be contained in V.)

Clauses 1–4 are sometimes referred to as *Tarski conditions* for satisfaction. If $b_1, \ldots, b_n$ includes all the free variables of A, and an evaluation e maps each b_i to an element $x_i \in M$, one often writes $M \vDash A[b_1/x_1, \ldots, b_n/x_n]$, or even $M \vDash A[x_1, \ldots, x_n]$, instead of $M \vDash_e A$.

Formulas without (occurrences of) free variables are called *sentences*. Satisfaction of such formulas in a model does not depend on the evaluation of free variables and is completely determined by the structure of the model itself. Hence, in any model each sentence is either *true* (valid) or *false* (not valid).

Every term t, whose (free) variables are exactly $b_1, \ldots, b_n$, defines on a model M a n-ary function $t_M : \langle x_1, \ldots, x_n \rangle \mapsto e(t)$, where e is the evaluation mapping b_i to $x_i \in M$. Such functions are called *term definable* in M. Every formula A, whose free variables are exactly $b_1, \ldots, b_n$, defines a n-ary predicate A_M on M:

$$A_M(x_1, \ldots, x_n) \iff M \vDash A[b_1/x_1, \ldots, b_n/x_n].$$

Such predicates are called *definable* in M. A function is *definable* in M, iff its graph is.

EXAMPLE 1.1. In the language of the model $(\mathbf{Z}, \leq)$ the only terms are variables, hence any term definable function is the identity. On the other hand, the relation $a_2 = a_1 + 1$ is definable by the formula

$$a_1 \leq a_2 \wedge \forall v_0\, (v_0 \leq a_2 \to (v_0 \leq a_1 \vee a_2 \leq v_0)).$$

Hence, the successor function $s(x) := x + 1$ is definable in $(\mathbf{Z}, \leq)$.

1.3. Morphisms. Let M and M' be two models of the same signature Σ. A *homomorphism* $\phi : M \to M'$ is a mapping of M to M' that preserves all the functions, predicates, and constants of Σ, i.e., for all n-ary $P \in \mathrm{Pred}$, $f \in \mathrm{Func}$, and $c \in \mathrm{Const}$

$$\begin{aligned} P_M(x_1, \ldots, x_n) &\Rightarrow P_{M'}(\phi(x_1), \ldots, \phi(x_n)) \\ \phi(f_M(x_1, \ldots, x_n)) &= f_{M'}(\phi(x_1), \ldots, \phi(x_n)) \\ \phi(c_M) &= c_{M'} \end{aligned}$$

It is immediate that composition of homomorphisms is a homomorphism. A homomorphism $\phi : M \to M'$ is an *isomorphism*, if it has an inverse, that is, a homomorphism ψ such that

$$\phi \circ \psi = id_{M'}, \qquad \psi \circ \phi = id_M,$$

where id_M is the identity homomorphism of M onto M: $id_M(x) = x$.

A *submodel* of M is any subset $N \subseteq M$ containing all constants of M and closed under all the functions of M, i.e.

1. $c_M \in N$, for all $c \in \mathrm{Const}$;
2. $x_1, \ldots, x_n \in N \Rightarrow f_M(x_1, \ldots, x_n) \in N$, for all n-ary $f \in \mathrm{Func}$.

Predicates, functions and constants of N are, by definition, those of M restricted to the domain N.

A homomorphism $\phi : M \to M'$ is an (isomorphic) *embedding*, iff $N := rng(\phi)$ is a submodel of M', and ϕ is an isomorphism of M and N. Every embedding is a one-to-one homomorphism, but the converse, in general, is not true. A one-to-one homomorphism of M into M' is an embedding iff for every n-ary predicate $P \in \mathrm{Pred}$ one has

$$P_{M'}(\phi(x_1), \ldots, \phi(x_n)) \Rightarrow P_M(x_1, \ldots, x_n),$$

for all $x_1, \ldots, x_n \in M$.

A *congruence* of a model M is an equivalence relation $\sim$ preserving all the functions and predicates of M: if $x_1 \sim y_1$, …, $x_n \sim y_n$ then

1. $P_M(x_1, \ldots, x_n) \iff P_M(y_1, \ldots, y_n)$, for all n-ary $P \in \mathrm{Pred}$;
2. $f_M(x_1, \ldots, x_n) \sim f_M(y_1, \ldots, y_n)$, for all n-ary $f \in \mathrm{Func}$.

If $\sim$ is a congruence of a model M, then all the functions and predicates of M are correctly defined on the set $M/\sim$ of equivalence classes with respect to $\sim$. The resulting structure is called the *factormodel* of M w.r.t. $\sim$. The canonical mapping that assigns to each element $x \in M$ its equivalence class $[x]$ is a homomorphism. This homomorphism is onto and *strong* in the sense that, for every predicate $P \in \mathrm{Pred}$ and every tuple $y_1, \ldots, y_n$ of elements of $M/\sim$ such that $P_{M/\sim}(y_1, \ldots, y_n)$, there exist $x_1, \ldots, x_n \in M$ such that $P_M(x_1, \ldots, x_n)$ and $[x_i] = y_i$ for $i = 1, \ldots, n$.

Vice versa, if $\phi : M \to M'$ is a homomorphism, then the binary relation $\sim$ on M defined by

$$x \sim y \iff \phi(x) = \phi(y)$$

is a congruence. The canonical mapping $\bar{\phi} : [x] \mapsto \phi(x)$ is a homomorphism of $M/\sim$ into M'. If ϕ is strong, then $\bar{\phi}$ is an embedding. If, besides, ϕ is onto, then $\bar{\phi}$ is an isomorphism of $M/\sim$ and M'.

The validity of first order formulas is preserved under isomorphisms of models: if $\phi : M \to M'$ is an isomorphism, then for every formula A, whose free variables are among $b_1, \ldots, b_n$, one has

$$M \vDash A[b_1/x_1, \ldots, b_n/x_n] \iff M' \vDash A[b_1/\phi(x_1), \ldots, b_n/\phi(x_n)],$$

for all $x_1, \ldots, x_n \in M$. The same holds, if M' is the factormodel $M/\sim$ for a congruence $\sim$ of M, and ϕ is the canonical homomorphism $x \mapsto [x]$. By transitivity, it follows that the validity of formulas is preserved under any strong onto homomorphisms of models.

1.4. Hilbert-style proof system. We fix a first order language $\mathcal{L}_\Sigma$. Hilbert-style proof system in $\mathcal{L}_\Sigma$ is given by the following logical *axiom schemata* and *inference rules*:

Axiom schemata:

A1. $A \to (B \to A)$

A2. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

A3. $(\neg A \to B) \to ((\neg A \to \neg B) \to A)$

A4. $\forall x A(x) \to A(t)$, where t is any term (and $x \in \mathrm{BoundVar}$ does not occur in $A(a)$)

Inference rules:

R1. $\dfrac{A, \quad A \to B}{B}$

R2. $\dfrac{A \to B(a)}{A \to \forall x B(x)}$ (a does not occur in A, x does not occur in $B(a)$)

Everywhere above A, B and C are arbitrary formulas of $\mathcal{L}_\Sigma$. Rule R1 is called *modus ponens*, and R2 is a version of *generalization* rule.

A first order *theory* T is any set of formulas of $\mathcal{L}_\Sigma$, called *extralogical axioms* of T. A *proof* or a *derivation* of a formula A in a theory T is a finite sequence of formulas that ends with A, and such that each formula occurring in the sequence is either a logical axiom (that is, has the form A1–A4 above), or is an extralogical axiom of T, or follows from some previous formulas by one of the inference rules. A formula A is *provable* in T, if it has a proof in T (denoted $T \vdash A$). Provable formulas are also called *theorems* of T. $\vdash A$ denotes the fact that A is provable in pure logic, that is, without any extralogical axioms. Two theories are *deductively equivalent,* if they have the same set of theorems.

A formula is *refutable* in T, if $T \vdash \neg A$. A theory T is *inconsistent,* if there is a formula, which is simultaneously provable and refutable in T; otherwise T is *consistent*. If T is inconsistent, then any formula of $\mathcal{L}_\Sigma$ is provable in T.

We say that M is a *model of a theory* T, if M is a model of the signature Σ, and every (extralogical) axiom of T is valid in M under any evaluation of its free variables; it then follows that every theorem of T must be valid in M, too. We denote this fact $M \vDash T$. The following result has fundamental importance.

THEOREM 1.2 (GÖDEL'S COMPLETENESS THEOREM). A first order theory T is consistent, if and only if there is a model M such that $M \vDash T$.

COROLLARY 1.3. A sentence A is provable in a theory T iff A is valid in every model of T, i.e. iff, for all models M, $M \vDash T$ implies $M \vDash A$. In particular, A is provable in pure logic, iff A is valid in any model of Σ.

Another important corollary is the following theorem.

THEOREM 1.4 (COMPACTNESS THEOREM). A theory T has a model iff every finite subset of the set of extralogical axioms of T does.

Models M and N of the same signature Σ are called *elementary equivalent,* iff for every sentence A in $\mathcal{L}_\Sigma$

$$M \vDash A \iff N \vDash A.$$

The *elementary theory* of a model M is axiomatized by the set of all sentences valid in M. Thus, models are elementary equivalent, iff their elementary theories coincide. Obviously, isomorphic models are elementary equivalent.

A submodel $N \subseteq M$ is called an *elementary* submodel, iff for all formulas A whose free variables are among $b_1, \dots, b_n$ and any tuple $x_1, \dots, x_n$ of elements of N

$$M \vDash A[b_1/x_1, \dots, b_n/x_n] \iff N \vDash A[b_1/x_1, \dots, b_n/x_n].$$

Clearly, in this case M and N are automatically elementary equivalent; the converse, however, need not be true in general.

Let Σ be a signature. By the *cardinality* $\mathrm{card}(\Sigma)$ of Σ we mean the cardinality of the joint alphabet $\mathrm{Pred} \cup \mathrm{Func} \cup \mathrm{Const}$ of its predicate, function, and constant symbols. Cardinality of a model is the cardinality of its universe.

THEOREM 1.5 (LÖWENHEIM-SKOLEM THEOREM). Any infinite model M of signature Σ has an elementary submodel of cardinality $\max(\aleph_0, \mathrm{card}(\Sigma))$. In particular, any infinite model of a countable signature has a countable elementary submodel.

In this book we shall exclusively deal with countable languages and theories. Löwenheim-Skolem theorem shows that in such a situation it is possible, in principle, to only deal with countable models. It should be noted, however, that it is not always mathematically fruitful to get rid of higher infinities in this way. [1]

1.5. Some helpful facts. There are a few basic facts on Hilbert-style proof system that will often be, explicitly or implicitly, referred to in this book. All of them are almost trivial consequences of Gödel's completeness theorem. On practice, however, they are usually proved directly, by purely syntactical manipulations, and are relied upon in the proofs of Gödel's theorem. The advantage of syntactical proofs is that they are constructive, and this is very essential for the questions of arithmetization of metamathematics. On the other hand, these proofs usually are more cumbersome — an expected consequence of our somewhat arbitrary choice of the proof system.

LEMMA 1.6 (DEDUCTION THEOREM). Let T be a theory, and A be a sentence in the language of T. Then, for all formulas B,

$$T \cup \{A\} \vdash B \iff T \vdash A \to B.$$

The theory $T \cup \{A\}$ is sometimes denoted $T + A$. By Compactness we obtain the following corollary.

COROLLARY 1.7. Let T be a set of sentences. Then $T \vdash B$, iff there are $A_1, \ldots, A_n \in T$ such that

$$\vdash A_1 \to (A_2 \to \ldots (A_n \to B) \ldots).$$

We notice that any theory T is deductively equivalent to a set of sentences (which is obtained by binding all the free variables occurring in the axioms of T by outer universal quantifiers). This operation is usually called the *universal closure*.

LEMMA 1.8 (RENAMING BOUND VARIABLES). Assume that bound variables x and y do not occur in a formula $A(a)$. Then

$$\vdash \forall x A(x) \leftrightarrow \forall y A(y).$$

[1] For example, uncountable cardinals naturally arise in, and are very essential for, the ordinal analysis of (relatively weak) fragments of second order arithmetic, although the whole theory can be reformulated in terms of countable hierarchies of primitive recursive functions, without any mention of cardinals.

Let C be a formula in which an n-ary predicate symbol P occurs, and let $A(b_1, \ldots, b_n)$ be a formula not containing any bound variables of C. $C(P/A)$ denotes, roughly, the result of substituting A for all occurrences of P in C. More precisely, for atomic formulas we have

$$P(t_1, \ldots, t_n)(P/A) = A(b_1/t_1, \ldots, b_n/t_n),$$

the substitution leaves the other atomic formulas unchanged and distributes over boolean connectives and quantifiers.

LEMMA 1.9 (SUBSTITUTION OF EQUIVALENT FORMULAS). Assume that formulas $A(b_1, \ldots, b_n)$ and $B(b_1, \ldots, b_n)$ do not contain any bound variables occurring in C. Then the equivalence

$$\vdash A(b_1, \ldots, b_n) \leftrightarrow B(b_1, \ldots, b_n)$$

implies

$$\vdash C(P/A) \leftrightarrow C(P/B).$$

Notice that combined applications of the previous two lemmas allow one to rename bound variables bound by quantifiers that may occur quite deep within a formula.

We say that a formula A is in *prenex normal form*, iff A has the form $\mathsf{Q}x_1\mathsf{Q}x_2 \ldots \mathsf{Q}x_n A_0(x_1, \ldots, x_n)$, where Q denotes indifferently one of the quantifiers $\forall$ or $\exists$, and A_0 is quantifier-free.

LEMMA 1.10 (PRENEX NORMAL FORM THEOREM). To every formula A one can effectively associate a formula A' in prenex normal form such that $\vdash A \leftrightarrow A'$.

It has to be noted that prenex normal form of a formula is, in general, not unique. (It is, of course, unique modulo logical equivalence.)

1.6. Equality. Most of the first order theories we shall deal with in this book are theories with *equality*. First order language with equality is a language whose signature Σ contains a distinguished binary predicate symbol $=$. A *model with absolute equality* is a model M for a language with equality such that $=$ is interpreted as the equality relation $\{\langle x, x\rangle \mid x \in M\}$ on M.

Equality axioms are the following ones:

1. $a_1 = a_1$
2. $a_1 = a_2 \to a_2 = a_1$
3. $a_1 = a_2 \to (a_2 = a_3 \to a_1 = a_3)$
4. $a_1 = b_1 \wedge a_2 = b_2 \wedge \cdots \wedge a_n = b_n \to (P(a_1, \ldots, a_n) \leftrightarrow P(b_1, \ldots, b_n))$
5. $a_1 = b_1 \wedge a_2 = b_2 \wedge \cdots \wedge a_n = b_n \to (f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n))$

for all n-ary $f \in \mathrm{Func}$ and $P \in \mathrm{Pred}$. These axioms are obviously valid in every model with absolute equality (for any evaluation of their free variables).

A *theory with equality* is a first order theory T in a language with equality that contains equality axioms. In this context, the other extralogical axioms of T are usually called *mathematical axioms*.

It is immediate that in any model M satisfying the equality axioms, the interpretation of $=$ is a congruence of M. Hence, the factormodel $M/{=_M}$ interprets $=$ as the equality relation, that is, is a model with absolute equality. Since factorization preserves the validity of first order formulas, we obtain the following versions of Gödel's completeness and Compactness theorems.

COROLLARY 1.11. A first order theory with equality is consistent, iff it has a model with absolute equality. A theory with equality T has a model with absolute equality, iff every finite subset of the set of mathematical axioms of T does.

In the following, unless specified otherwise, we shall always deal with theories with equality. When speaking about models we shall omit the words "with absolute equality".

1.7. Definitional extensions. Let T and U be first order theories such that the language of U contains that of T. U is *conservative* over T, iff every theorem of U in the language of T is provable in T. The simplest conservative extensions of theories are definitional ones. They come in the following two basic types.

INTRODUCING A PREDICATE SYMBOL. Let $A(a_1, \ldots, a_n)$ be a formula in the language of T, whose free variables are exactly those shown. Consider a theory U which has, in addition to the language of T, a new n-ary predicate symbol P. Axioms of U are those of T together with the formula

$$P(a_1, \ldots, a_n) \leftrightarrow A(a_1, \ldots, a_n).$$

Then U is conservative over T.

Model-theoretically, this fact is clear: any model M of T can be expanded to a model of U by defining the interpretation of P as the predicate $A_M(x_1, \ldots, x_n)$. A constructive syntactical proof is also easy. For any formula B in the language of U, let B^- denote the formula $B(P/A')$, where A' is the result of renaming all the bound variables of A in order to make them disjoint from the variables of B. Since all the axioms and rules of our Hilbert-style proof system are schematic, it is can be easily shown that $T \vdash B^-$, whenever $U \vdash B$, by induction on the length of the proof of B in U. For formulas B in the language of T, obviously, B^- coincides with B, whence the conservativity. Moreover, for this translation we have $B \leftrightarrow B^-$ provably in U, for all formulas B in the language of U.

INTRODUCING A FUNCTION SYMBOL. We say that a formula $F(a_1, \ldots, a_n, b)$ defines a function within a theory T, iff T proves

$$\exists y\, (F(a_1, \ldots, a_n, y) \wedge \forall z\, (F(a_1, \ldots, a_n, z) \rightarrow y = z)).$$

(This formula is usually abbreviated by $\exists! y F(a_1, \ldots, a_n, y)$.) For such a formula F we introduce a new n-ary function symbol f and consider a theory U in the extended language which, in addition to the axioms of T, has the axiom

$$F(a_1, \ldots, a_n, f(a_1, \ldots, a_n)).$$

Then U is conservative over T. Indeed, any model M of T can be expanded to a model of U by defining the interpretation of f as the (unique) n-ary function on M whose graph is $F_M(x_1, \ldots, x_n, y)$.

2. Turing Machines and Computability

The general notion of computability is as basic as those of provability and truth. Even the most ancient computing device, *abacus*, which is essentially a method of moving pebbles from one hole to another according to a certain finite set of rules, provides a universal model of computation, i.e., is capable of simulating any computational procedure. Nowadays, hundreds of different computational models are known. For theoretical purposes one usually takes the simple model of a *Turing machine*.

2.8. Turing machines. Informally speaking, a Turing machine consists of a *tape*, a *head*, and a *control device*. The tape is linear and two-way infinite. It is divided into *cells*, which contain symbols of a given *alphabet* $A = \{S_0, S_1, \ldots, S_n\}$; the symbol S_0 will be interpreted as a blank cell. At any given moment the head reads the content of exactly one cell and passes this information to the control device. The control device finds itself in one of the finitely many *inner states* $Q = \{q_0, q_1, \ldots, q_m\}$. It operates the head according to its *program*, which consists of finitely many *commands* of the following three basic types:

- If the control device is in a state q_i and the head reads a symbol S_j, then the content of the cell must be replaced by S_k and the control device must change its state to q_r (this command is abbreviated by $q_i S_j S_k q_r$).
- If the control device is in a state q_i and the head reads a symbol S_j, then the head must move one cell to the right and the control device must change its state to q_r (abbreviated $q_i S_j R q_r$).
- If the control device is in a state q_i and the head reads a symbol S_j, then the head must move one cell to the left and the control device must change its state to q_r (abbreviated $q_i S_j L q_r$).

We assume that the program is consistent in the sense that it does not contain a pair of different commands with the same two initial symbols $q_i S_j$.

The machine works stepwise, starting from a configuration where all but finitely many symbols written on the tape are blank, the head reads the leftmost nonblank symbol (unless all the symbols are blank), and the control device is

in a distinguished *initial state* q_0. Steps of the computation correspond to the execution of the commands of the program. Since there are no conflicting instructions, the behaviour of the machine is deterministic, that is, at most one command can be executed at a time. In principle, there exist two possibilities:

1. The machine stops, not having a command corresponding to its current inner state and the symbol it reads. Then the content of the tape is called the *result* of the computation of the machine on a given initial configuration.

2. The machine never stops. In this situation the result of the computation is undefined.

Now we give the formal definitions. A *Turing machine* is a triple $\mathcal{T} = \langle A, Q, P\rangle$, where

- $A = \{S_0, S_1, \ldots, S_n\}$ is an alphabet of *tape symbols*.
- $Q = \{q_0, q_1, \ldots, q_m\}$ is a set of *inner states*, $A \cap Q = \emptyset$.
- P (*program*) is a finite set of quadruples (*commands*) of the form $q_iS_jS_kq_r$, $q_iS_jLq_r$, $q_iS_jRq_r$, where $S_j, S_k \in A$, $q_i, q_r \in Q$, L and R are two extra symbols not occurring in $A \cup Q$, and different commands from P begin with different pairs of symbols q_iS_j.

A *configuration* of a Turing machine $\mathcal{T}$ is a word of the form Xq_iY, where X and Y are words in the alphabet A, $Y \neq \Lambda$ (empty word), and $q_i \in Q$. XY is called the *tape word* of the configuration Xq_iY. A configuration is called *initial*, iff q_i is the initial state q_0, $X = \Lambda$, and $Y = S_0$ or begins with a tape symbol different from S_0. A machine $\mathcal{T}$ *transforms* a configuration α into a configuration β (denoted $\alpha \underset{\mathcal{T}}{\longrightarrow} \beta$), iff one of the following conditions holds:

1. α has the form Xq_iS_jY, $\beta = Xq_rS_kY$ and $q_iS_jS_kq_r \in P$,
2. α has the form $XS_kq_iS_jY$, $\beta = Xq_rS_kS_jY$ and $q_iS_jLq_r \in P$,
3. α has the form q_iS_jY, $\beta = q_rS_0S_jY$ and $q_iS_jLq_r \in P$,
4. α has the form Xq_iS_jY $(Y \neq \Lambda)$, $\beta = XS_jq_rY$ and $q_iS_jRq_r \in P$,
5. α has the form Xq_iS_j, $\beta = XS_jq_rS_0$ and $q_iS_jRq_r \in P$.

A machine $\mathcal{T}$ *stops* in a configuration α, iff there is no β such that $\alpha \underset{\mathcal{T}}{\longrightarrow} \beta$. A *computation protocol* of a machine $\mathcal{T}$ is a finite sequence of configurations $\alpha_0, \ldots, \alpha_k$ such that α_0 is an initial configuration, $\alpha_i \underset{\mathcal{T}}{\longrightarrow} \alpha_{i+1}$ for $i = 0, 1, \ldots, k-1$, and $\mathcal{T}$ stops in the configuration α_k. We let $\alpha \underset{\mathcal{T}}{\Longrightarrow} \beta$ iff there is a computation protocol $\alpha_0, \ldots, \alpha_k$ of a machine $\mathcal{T}$ such that $\alpha_0 = \alpha$ and $\alpha_k = \beta$.

Let Σ and Δ be any two finite alphabets, and let Σ^* and Δ^* denote the sets of all words in Σ and Δ, respectively. A partial function $f : \Sigma^* \to \Delta^*$ is *computed* by a Turing machine $\mathcal{T} = \langle A, Q, P\rangle$, iff $A \supset \Sigma \cup \Delta$, the blank symbol

does not belong to $\Sigma \cup \Delta$, and for all words $X \in \Sigma^*$, $Y \in \Delta^*$, $f(X) = Y$ holds exactly when for some configuration β, $q_0 X S_0 \underset{\mathcal{T}}{\Longrightarrow} \beta$ and the tape word of β has the form $W_1 Y W_2$, where the words W_1, W_2 consist entirely of blank symbols. Notice that $f(X)$ is undefined exactly for those X that either the machine $\mathcal{T}$ starting from $q_0 X S_0$ never stops, or stops in a meaningless configuration.

A (partial) function $f : \Sigma^* \to \Delta^*$ is *computable* or *partial recursive*, iff there is a Turing machine that computes it. We give a separate definition of a computable n-ary partial function $f : \mathbf{N}^n \to \mathbf{N}$.

Let $\overline{n}$ denote the usual binary expansion of a number $n \in \mathbf{N}$: $\overline{0} = 0$, $\overline{1} = 1$, $\overline{2} = 10$, $\overline{3} = 11$, etc. We consider an alphabet Σ containing the following symbols: $0, 1, \sharp$. A sequence of numbers $\langle k_1, \ldots, k_n \rangle$ is then coded in Σ^* by a string of the form $\overline{k_1}\sharp\overline{k_2}\sharp\cdots\sharp\overline{k_n}$. By definition, a (partial) function $f : \mathbf{N}^n \to \mathbf{N}$ is *computable*, iff the corresponding partial word function $\overline{f} : \Sigma^* \to \Sigma^*$ defined by

$$\overline{f} : \ \overline{k_1}\sharp\overline{k_2}\sharp\cdots\sharp\overline{k_n} \longmapsto \overline{f(k_1, \ldots, k_n)}$$

is computable.

The famous Church-Turing Thesis expresses the fact that the notion of Turing machine adequately formalizes the intuitive idea of comutability.

CHURCH-TURING THESIS. Every intuitively computable word function, in particular, every intuitively computable number-theoretic function, is Turing machine computable.

Of course, Church-Turing Thesis cannot be proved or disproved mathematically. Its validity is rather established experimentally, by modelling other formalizations of computability via Turing machines. We shall rely upon Church-Turing Thesis on a number of occasions in this book by presenting informal descriptions of algorithms instead of exhibiting the concrete Turing machines claimed to exist.

?? Space, time

2.9. Recursive and recursively enumerable sets. First, we fix some standard terminology and notation concerning partial functions. By a partial n-ary function on $\mathbf{N}$ we mean any function $f : D \to \mathbf{N}$, where $D \subseteq \mathbf{N}^n$. The set D is called the *domain* of f and denoted $dom(f)$; *range* of f is the set $rng(f) = \{f(x) \mid x \in D\}$. For a partial function f, $f(x)\uparrow$ means that $f(x)$ is *undefined* or *diverges,* that is, $x \notin dom(f)$; $f(x)\downarrow$ means that $f(x)$ *converges,* that is, $x \in dom(f)$. We write $f(x) \simeq g(x)$, if either $f(x)\uparrow$ and $g(x)\uparrow$, or $f(x)\downarrow$, $g(x)\downarrow$, and $f(x) = g(x)$. An n-ary function f is *total*, iff $dom(f) = \mathbf{N}^n$.

Cantor's standard pairing function

$$c(x, y) = \frac{(x + y)(x + y + 1)}{2} + x$$

establishes a one-to-one correspondence between $\mathbf{N} \times \mathbf{N}$ and $\mathbf{N}$. It enumerates pairs (x, y) of natural numbers in the order

$$(0,0);\ (0,1);\ (1,0);\ (0,2);\ (1,1);\ (2,0);\ (0,3); \ldots$$

$c(x, y)$ is total recursive, and so are the inverse *projection functions* π_0, π_1 uniquely defined by the equations

$$\pi_0(c(x, y)) = x, \quad \pi_1(c(x, y)) = y.$$

Using c one can define a recursive bijection $c_n : \mathbf{N}^n \to \mathbf{N}$ as follows:

$$c_n(x_1, \ldots, x_n) = c(x_1, c(x_2, \ldots c(x_{n-1}, x_n) \ldots)).$$

Notice that in this case, for $1 \leq i < n$,

$$x_i = \pi_0((\pi_1)^{i-1}(c_n(x_1, \ldots, x_n))).$$

So, the n-dimensional projection functions π_i^n can be defined by

$$\begin{array}{rcll} \pi_i^n(x) & = & \pi_0((\pi_1)^{i-1}(x)), & \text{for } 1 \leq i < n; \\ \pi_n^n(x) & = & (\pi_1)^{n-1}(x). & \end{array}$$

Let f be a partial n-ary function on $\mathbf{N}$, and let $f' : \mathbf{N} \to \mathbf{N}$ be defined by

$$f'(x) \simeq f(\pi_1^n(x), \ldots, \pi_n^n(x)).$$

Then, obviously, f is computable iff f' is and, moreover, the value of f can be recovered by

$$f(x_1, \ldots, x_n) \simeq f'(c_n(x_1, \ldots, x_n)).$$

The correspondence $f \rightsquigarrow f'$ allows in many situations to speak about unary computable functions, rather than about n-ary functions for an arbitrary n.

A subset $X \subseteq \mathbf{N}$ is called *decidable* or *recursive*, iff its characteristic function

$$\chi_X(x) = \begin{cases} 1, & \text{if } x \in X \\ 0, & \text{if } x \notin X \end{cases}$$

is computable. Informally, X is decidable, if there is an algorithm deciding in finitely many steps whether $x \in X$ or $x \notin X$, for any given x. Notice that χ_X is a total function, that is, the decision algorithm converges everywhere.

Since $\chi_{X \cap Y}(x) = \chi_X(x) \cdot \chi_Y(x)$, and $\chi_{\mathbf{N} \setminus X}(x) = 1 - \chi_X(x)$, decidable sets are closed under boolean operations. Decidable sets include all finite sets, as well as many other familiar sets, such as the set of even numbers, the set of prime numbers, etc. Since there are only countably many computable functions, there do exist undecidable subsets of $\mathbf{N}$. Decidable n-ary predicates are similarly defined: $P(x_1, \ldots, x_n)$ is decidable, iff the function

$$\chi_P(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } P(x_1, \ldots, x_n) \\ 0, & \text{if not } P(x_1, \ldots, x_n) \end{cases}$$

is computable.

A set $X \subseteq \mathbf{N}$ is *recursively enumerable* (r.e.), iff for some computable (partial) function $f : \mathbf{N} \to \mathbf{N}$, $X = rng(f)$. Intuitively, r.e. sets are those that can be generated by some effective procedure.

LEMMA 2.1. For any subset $X \subseteq \mathbf{N}$ the following properties are equivalent:

(i) X is recursively enumerable;

(ii) $X = \emptyset$ or $X = rng(f)$, for some total recursive function $f : \mathbf{N} \to \mathbf{N}$;

(iii) For some recursive binary relation $R(x, y)$,

$$X = \{x \in \mathbf{N} \mid \exists y R(x, y)\};$$

(iv) $X = dom(f)$, for some partial recursive function f.

LEMMA 2.2 (GRAPH THEOREM). A partial n-ary function f is computable, iff its graph $\{\langle x_1, \ldots, x_n, y\rangle \mid f(x_1, \ldots, x_n) = y\}$ is r.e.

LEMMA 2.3 (POST'S THEOREM). $X \subseteq \mathbf{N}$ is decidable, iff both X and $\mathbf{N} \setminus X$ are r.e.

The definition r.e. sets can be extended to arbitrary n-ary predicates via the effective coding of n-tuples of numbers described above. Besides, both decidable and r.e. sets of numbers can be obviously generalized to the *sets of words* in an arbitrary finite alphabet. Typical examples of r.e. sets are provided by the following lemma.

LEMMA 2.4. Let T be a first order theory formulated in a finite language. If the set of extralogical axioms of T is r.e., then so is the set of its theorems. In particular, any finitely axiomatized theory has an r.e. set of theorems.

PROOF. Any formula in the language of T is a word in a finite alphabet. More precisely, it can be identified with a word in a finite alphabet, if one suitably encodes the alphabets of free and bound variables. E.g., one can stipulate that a_i and v_i are the following words in the 3-letter alphabet $\{a, v, \sharp\}$:

$$a_i = a \underbrace{\sharp \cdots \sharp}_{i \text{ times}} a$$

$$v_i = v \underbrace{\sharp \cdots \sharp}_{i \text{ times}} v$$

An effective enumeration of the set of theorems of T can be described in the following way (using the Church-Turing thesis). Let $f(n)$ be a total recursive enumeration of the set of axioms of T. Construct finite sets of formulas T_n inductively as follows. $T_0 = \emptyset$; T_{n+1} is the union of T_n and the set of all formulas A such that A is either a logical axiom of length n, or follows from some formulas in T_n by an application of modus ponens or generalization rules, or $A = f(n)$. Clearly, for each n, $T_n \subset T$, and, moreover, any theorem of T eventually belongs to some T_n, because any T-proof uses only finitely many logical and extralogical axioms. Let $g(n, i)$ denote the i-th formula in the set T_n (say, using a fixed lexicographic ordering of formulas). Since T_n is effectively generated, $g(n, i)$ is computable as a (partial) function of n and i; hence the function $h(n) = g(\pi_0(n), \pi_1(n))$ is partial recursive and enumerates the set of all theorems of T, q.e.d.

2.10. Universal Turing Machine. One of the fundamental ideas of the theory of computation is that programs can not only be executed themselves, but also be used as initial data by the other programs. Any operational system works in a similar way: if one feeds it (the name of) an executable program P together with a suitable input for P, it would try to apply P to the given input. Of course, if anything goes wrong — P is not an executable program, or the format of P does not fit the format of the input we gave it — the system will issue an error message. But the point is that through this mechanism a *single* program (operational system) is able to model any other (*arbitrary*) algorithm. The theoretical analog of the operational system is the notion of a universal function (Turing machine) introduced below.

First, we need some effective coding of Turing machines. There is a lot of freedom in the choice of such a coding; just to be specific, we develop an approach based on binary expansions of numbers, which will also be used in other situations throughout this book.

Since the alphabets of any Turing machine $\mathcal{T} = (A, Q, P)$ are finite, we can fix a sufficiently large constant c such that all the symbols can be coded (in some way) as binary strings of length c, that is, we have an injective function

$$code : A \cup Q \cup \{R, L\} \longrightarrow \{0,1\}^c.$$

Then we let

$$\begin{aligned} code(Q) &= code(q_0) * \cdots * code(q_m), \\ code(A) &= code(S_0) * \cdots * code(S_n), \end{aligned}$$

where $*$ denotes the concatenation of binary strings. Commands are coded as particular strings of length $4c$, e.g.,

$$code(q_i S_j R q_k) = code(q_i) * code(S_j) * code(R) * code(q_k),$$

and similarly for the other types of commands. If $P = \{C_1, \ldots, C_r\}$ then we let

$$code(P) = code(C_1) * \cdots * code(C_r).$$

For a binary string $s = a_{n-1} \ldots a_0$, let $\ulcorner s \urcorner$ denote the number whose binary expansion is $1a_{n-1} \ldots a_0$, that is,

$$\ulcorner s \urcorner = 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2 + a_0.$$

A Turing machine $\mathcal{T} = (A, Q, P)$ can then be encoded, using Cantor's coding of quadruples, as follows:

$$\ulcorner \mathcal{T} \urcorner = c_4(\ulcorner code(A) \urcorner, \ulcorner code(Q) \urcorner, \ulcorner code(P) \urcorner, c).$$

Clearly, different Turing machines have different codes, and the coding is effective in the sense that one can effectively reconstruct the machine, that is, its alphabets and the program, from its code.

Let ϕ_i^n denote the partial function $\mathbf{N}^n \to \mathbf{N}$ computed by the Turing machine coded by i. For definiteness, we stipulate that ϕ_i^n is the empty function, if i does not encode any Turing machine (whose tape alphabet contains $\{0, 1, \sharp\}$).

A *universal function* for the class of all computable partial n-ary functions is an $(n+1)$-ary function Φ_n such that for all $i, x_1, \ldots, x_n \in \mathbf{N}$,

$$\Phi_n(i, x_1, \ldots, x_n) \simeq \phi_i^n(x_1, \ldots, x_n).$$

LEMMA 2.5.

1. Φ_n is computable.
2. For every partial recursive n-ary function f, there is an i (called the *index* of f) such that
$$\Phi_n(i, x_1, \ldots, x_n) \simeq f(x_1, \ldots, x_n).$$
3. There is a total recursive m-ary function S_m^n such that for every partial recursive $n+m$-ary function f there holds
$$\Phi_n(S_m^n(y_1, \ldots, y_m), x_1, \ldots, x_n) \simeq f(y_1, \ldots, y_m, x_1, \ldots, x_n).$$

THEOREM 2.6. The set $K := \{x \mid \Phi_1(x, x)\downarrow\}$ is r.e. and undecidable.

PROOF. K is r.e. being the domain of the computable function $\Phi_1(x, x)$. Assume that the characteristic function

$$\chi_K(x) = \begin{cases} 1, & \text{if } \Phi_1(x, x)\downarrow \\ 0, & \text{otherwise} \end{cases}$$

is computable. Then the function

$$h(x) = \begin{cases} 0, & \text{if } \chi_K(x) = 0 \\ \uparrow, & \text{if } \chi_K(x) = 1 \end{cases}$$

is computable, too. Let m be any index of h, that is,

$$h(x) \simeq \Phi_1(m, x).$$

Then we have

$$h(m)\downarrow \Leftrightarrow \chi_K(m) = 0 \Leftrightarrow \Phi_1(m, m)\uparrow \Leftrightarrow h(m)\uparrow,$$

which is a contradiction, q.e.d.

Chapter 2

Gödel's Incompleteness Theorems

1. Formal arithmetic: Computability and Truth

In this section we establish a fundamental relationship between the notions of Computability and Truth in formal arithmetic. Our technical goal is to show that all computable functions and r.e. relations are definable in the language of arithmetic. In fact, for a natural class of arithmetical Σ_1 formulas there is an exact match between the power of computation and the expressive power of arithmetical language, that is, r.e. predicates are exactly those definable by Σ_1 formulas in the standard model. In particular, the correspondence allows to consider the Σ_1 fragment of the arithmetical language as a kind of high-level programming language and an alternative model of computation. The proof of the arithmetical definability of r.e. predicates, thus, has the format of constructing an interpreter of Turing machines in the language of arithmetic.

At the same time, the same correspondence yields a fundamental theorem of logic: the set of all true arithmetical formulas does not have an effective (r.e.) axiomatization. This result implies a version of Gödel's famous First Incompleteness Theorem stating that there is a true arithmetical sentence, which is unprovable in the classical axiomatic system of Peano arithmetic PA. Moreover, no r.e. extension of PA, whose axioms are true, can be complete in this sense. This version of Gödel's theorem will be further improved in the next section, where the notion of Provability is studied in greater detail.

1.11. Standard model and Peano arithmetic. The language of arithmetic is a first order language containing binary predicate symbols $=$ and $\leq$; binary function symbols $+$ and $\cdot$; unary function symbols S and exp; and a constant 0. The *standard model* of arithmetic is a model with the universe

$\mathbf{N} = \{0, 1, 2, \dots\}$ such that all the symbols have their usual interpretation: $=$ is the equality relation; $\leq$ is the ordering relation; $+$ and $\cdot$ are the addition and multiplication operations; S is the successor function $S(x) = x + 1$; exp is the base 2 exponentiation function $\exp(x) = 2^x$; 0 is 0.

Formulas in the above language are called *arithmetical*. In writing them we shall follow the usual notational conventions such as omitting superfluous parentheses; using infix notation for $=$, $\leq$, $+$ and $\cdot$ (that is, writing e.g. $x \leq y$ instead of $\leq(x, y)$); writing 2^x instead of $\exp(x)$; and so forth.

Peano Arithmetic PA is a first order theory with equality formulated in the arithmetical language and having the following mathematical axioms:

P1. $\neg S(a) = 0$

P2. $S(a) = S(b) \to a = b$

P3. $a + 0 = a$

P4. $a + S(b) = S(a + b)$

P5. $a \cdot 0 = 0$

P6. $a \cdot S(b) = a \cdot b + a$

P7. $\exp(0) = S(0)$

P8. $\exp(S(a)) = \exp(a) + \exp(a)$

P9. $a \leq 0 \leftrightarrow a = 0$

P10. $a \leq S(b) \leftrightarrow (a \leq b \vee a = S(b))$

P11. $A(0) \wedge \forall x(A(x) \to A(S(x))) \to \forall x A(x)$, for all arithmetical formulas $A(a)$, possibly containing other free variables.

The last axiom is called the *induction axiom schema*. It formalizes the usual principle of mathematical induction applied to the predicates definable in the language of arithmetic. The following lemma and its corollary are immediate.

LEMMA 1.1. $\mathbf{N} \models PA$, that is, all theorems of PA are valid in the standard model.

COROLLARY 1.2. PA is consistent.

Gödel's famous First Incompleteness Theorem, in the most narrow formulation, states that the converse of Lemma 1.1 does not hold, that is, there exists a valid arithmetical sentence which is unprovable in PA. Thus, PA is properly included in the full *true arithmetic* TA — the theory axiomatized by the set of all true arithmetical sentences. We shall prove Gödel's theorem later in this chapter. For now, let us notice (following Kreisel [**?**]) that, from the point of view of the foundations of mathematics, this raises the question about the status of PA: if PA is incomplete, what makes it better than any other incomplete system of arithmetic, and why is it important to study it? We do not want to go deeply into these matters, especially at this early point, yet it would be unfair to the philosophically inclined reader, if we completely avoid them.

1.12. What is so special about PA? Peano arithmetic plays a prominent role in the investigations on proof theory and foundations of mathematics, as well as in mathematical logic as a whole. This is largely due to the following widely accepted thesis that we, for convenience, dub "Peano's."

PEANO'S THESIS. PA naturally formalizes all finitary mathematics.

The status of this statement is similar to the Church-Turing thesis in that it relates the informal concept of "finitary mathematics" to the formal concept of PA. Very roughly, finitary, or concrete, mathematics deals with finite objects, such as numbers, graphs, finite fields, ... and concrete, explicitly definable constructions with these objects. It includes, e.g., the elementary number theory, combinatorics, large parts of algebra and discrete mathematics. Although Peano's thesis cannot be proved or disproved mathematically, there is much evidence supporting it.

(a) Indeed, large portions of finitary mathematics, like those mentioned above, have either been explicitly formalized, or developed to such a degree of detalization that their formalizability in PA became evident. Nowadays automated deduction systems are used to construct formal analogues of mathematical proofs.

(b) Other approaches to the formalization of finitary mathematics, e.g., the one based on the concept of a hereditarily finite set, yield systems equivalent to PA (modulo suitable translations). For example, an alternative formulation of PA is obtained by replacing the infinity axiom, in the standard formulation of Zermelo-Fraenkel set theory, by its negation.

(c) Finally, there is proof-theorists' experience of formalization in PA. We hope, the reader of this book will eventually develop his own intuition as to what is and what is not formalizable in PA.

The notion of "finitary mathematics," which mainly refers to a certain area of conventional mathematical practice, is not to be confused with the philosophical concept of "finitism." The latter was developed in the context of the so-called *Hilbert's Programme* of justifying infinitary methods in mathematics using the means not involving, in any form, the notion of actual infinity. From the strictly finitist point of view even PA is not acceptable, essentially because the quantifiers ranging over an infinite domain (especially complex combinations of them) are not unequivocally finitistically meaningful. See the introductory part of Hilbert and Bernays [**?**] for a discussion.

Hilbert himself never developed any formal system capturing the notion of finitistic proof — actually, it was not necessary for the aims of his program, only the usual mathematics had to be formalized. Since then, a few good candidates for such systems have been proposed (see e.g. Kreisel [**?**]). We shall also mention one such system later. However, as Hilbert's Programme, in its most radical formulation, failed, these questions became less urgent than they have been, and nowadays mainly present a historical and philosophical interest.

?? Consistency of PA is infinitary. Analytic methods in number theory. PA can talk about infinite objects, provided they have some kind of finite descriptions. Many results apply to arbitrary theories.

1.13. Bounded formulas. Can we effectively decide, whether a given arithmetical formula is true or false? We shall soon prove that, in general, we cannot. The problem is with the quantifiers: one cannot check the truth of $\forall x A(x)$ simply by looking through all $x = 0, 1, 2, \ldots$. However, there are large enough subclasses of the class of arithmetical formulas for which the truth or falsity can be effectively recognized.

Let the expressions $\forall x \leq t\ A(x)$ and $\exists x \leq t\ A(x)$ abbreviate the formulas $\forall x\,(x \leq t \to A(x))$ and $\exists x\,(x \leq t \wedge A(x))$, respectively, where t is any term (not containing the variable x). Occurrences of quantifiers of this kind are called *bounded*, and bounded or *elementary formulas* are those, all of whose quantifiers are bounded. The class of elementary formulas is denoted $\Delta_0(\exp)$, to stress the fact that the exponentiation function is explicitly present in the language. Notice that, by definition, quantifier-free formulas are elementary. *Elementary predicates* are those definable by elementary formulas in the standard model.

LEMMA 1.3. Elementary predicates are decidable.

PROOF. We give an informal description of the decision algorithm associated with an arbitrary elementary formula $A(a_1, \ldots, a_n)$. The algorithm inputs a tuple of numbers $x_1, \ldots, x_n$ and outputs 1 or 0 depending on whether $A[x_1, \ldots, x_n]$ is true or false. (We call this procedure the *evaluation* of A.) The formula A is analyzed according to the following instructions.

- $t_1 = t_2$: Compute the values of the terms t_1, t_2; return 1 if the values coincide, return 0 otherwise.
- $t_1 \leq t_2$: Compute the values of t_1, t_2; return 1 if the value of t_1 is less than or equal to that of t_2, return 0 otherwise.
- $B \to C$: Evaluate separately B and C; return 1 if B returns 0 or C returns 1; return 0 otherwise.
- $\neg B$: Evaluate B; return 1 if B returns 0 and vice versa.
- $\exists v \leq t\ A(v, a_1, \ldots, a_n)$: Compute the value m of t; for $x = 0, 1, \ldots, m$ evaluate $A[x, x_1, \ldots, x_n]$; return 1, as soon as an $x \leq m$ is found such that $A[x, x_1, \ldots, x_n]$ evaluates to 1; return 0, if no such x is found.
- $\forall v \leq t\ A(v, a_1, \ldots, a_n)$: evaluated dually.

Notice that according to the above clauses an elementary formula, for any tuple of numbers, evaluates to 0 or to 1, that is, the algorithm terminates on any input. It is also clear that a formula evaluates to 1 if and only if it is valid. By Church-Turing Thesis this means that elementary formulas define decidable predicates in the standard model of arithmetic, q.e.d.

A rough estimate of the complexity of the evaluation procedure allows us to sharpen the previous result. Let the *iterated exponentiation* function 2^x_n be defined by the equations $2^x_0 = x$, $2^x_{n+1} = 2^{2^x_n}$. *Multiexponential* functions are those of the form 2^x_n for a fixed n. Since 2^x grows faster than any polynomial, we easily obtain the following lemma.

LEMMA 1.4. For every arithmetical term $t(a_1, \ldots, a_m)$ there is an n such that $t(x_1, \ldots, x_m) \leq 2_n^{x_1+\cdots+x_m}$, for all (sufficiently large) $x_1, \ldots, x_n \in \mathbf{N}$.

This lemma can be restated as saying that the rate of growth of arithmetical terms is multiexponentially bounded.

LEMMA 1.5. Elementary predicates are decidable in mutiexponential number of steps of the length of the input.

PROOF. First we need to estimate the complexity of the term evaluation procedure. It is somewhat easier to estimate the amount of space (= memory) used. Indeed, the simple "school" algorithms for addition and multiplication are obviously polytime and use, respectively, linear $O(n)$ and quadratic $O(n^2)$ amount of space (where n is the size of the input). We recommend the reader to check these claims for himself.

For exponentiation, any reasonable algorithm computing 2^x uses the amount of memory of order $2^{O(n)}$, where n is the size of x. For example, one can represent a number x with the binary expansion $a_n \ldots a_1 a_0$ in the form

$$(\ldots(a_n \cdot 2 + a_{n-1}) \cdot 2 + \cdots + a_1) \cdot 2 + a_0$$

and compute a sequence of zeros of length x starting from a single 0 corresponding to the bit a_n, and applying for $i = n, \ldots, 1$ (n times) the operation of duplicating the string, followed by adding an extra 0 in case $a_i = 1$. Then if one adds the symbol 1 at the beginning of that sequence, one obtains the binary expansion of 2^x. The complexity of this algorithm is easily estimated using the fact that the duplication operation uses linear space.

Now we notice that an upper space bound for a computation of a function $f(g(x), h(x))$ can be obtained by composition of the space bounds for f and the sum of those for g and h (assuming those bounds grow faster than the identity function). By induction, using Lemma 1.4, we conclude that the space complexity of the evaluation procedure for any arithmetical term t on input of size n can be bounded by a multiexponential function.

Now we estimate the complexity of the evaluation procedure for elementary formulas. The multiexponential bound is proved by induction. For atomic formulas the result follows from our bound for the evaluation of terms. The treatment of boolean connectives is straightforward, so let us consider the principal case of a bounded existential quantifier.

To decide whether $\exists v \leq t\, A[x_1, \ldots, x_k]$ holds we must first evaluate the term t, say to a number y, and then check for all $x \leq y$ whether $A[v/x, x_1, \ldots, x_k]$ holds. The evaluation of t consumes multiexponential space, and we can organize our computation in such a way that the validity of formulas $A[x, x_1, \ldots, x_k]$ is checked independently for different $x \leq y$. That is, the piece of tape used for the evaluation of $A[0, x_1, \ldots, x_k]$ is then erased and reused for the evaluation of $A[1, x_1, \ldots, x_k]$ and so on. Thus, it is sufficient to give a multiexponential bound on the complexity of the evaluation of $A[x, x_1, \ldots, x_k]$ in the worst case.

By induction hypothesis, $A[x, x_1, \ldots, x_k]$ can be checked using the amount of space bounded by 2_c^{m+n} for some constant c, where m is the size of x, and n is the size of $x_1, \ldots, x_k$. At worst, the bound is $2_c^{|y|+n}$, and for the size $|y|$ of y we have a bound of a similar form 2_d^n, because the term t has multiexponential rate of growth. So, the worst case estimate for the complexity of the evaluation of $A[x, x_1, \ldots, x_k]$ for $x \leq y$ is

$2_c^{2_d^n+n} \leq 2_{c+d+1}^n$ for sufficiently large n. This proves a multiexponential bound on the space complexity. At the cost of iterating exponentiation one more time we obtain a similar bound for the time complexity, q.e.d.

1.14. Arithmetical hierarchy. Elementary formulas are the simplest kind of formulas we shall deal with, and now we are going to classify arbitrary arithmetical formulas according to their logical complexity. The most common measure of logical complexity of a formula is the depth of quantifierchanges, which leads to the classical *arithmetical hierarchy*.

For $n \geq 0$ the classes of Σ_n and Π_n formulas are inductively defined as follows. Σ_0 and Π_0 formulas are elementary formulas. Σ_{n+1} formulas are those of the form $\exists x_1 \ldots \exists x_m A(x_1, \ldots, x_m)$, where A is a Π_n formula. Π_{n+1} formulas are those of the form $\forall x_1 \ldots \forall x_m A(x_1, \ldots, x_m)$, where A is a Σ_n formula. In other words, Σ_n formulas are bounded formulas prefixed by n alternating blocks of similar quantifiers, starting from $\exists$, and Π_n formulas are defined dually. The classes of Σ_n and Π_n formulas are denoted $\boldsymbol{\Sigma}_n$ and $\boldsymbol{\Pi}_n$, respectively.

LEMMA 1.6. Every arithmetical formula is logically equivalent to a Σ_n formula, for some n.

PROOF. Add a dummy existential quantifier in front of the given formula (taking, e.g., $\exists x\,(x = x \wedge A)$ instead of A) and transform it to a prenex normal form by the usual algorithm, q.e.d.

For obvious reasons, the above lemma also holds for the class Π_n instead of Σ_n. The same trick of introducing dummy quantifiers also yields the following lemma.

LEMMA 1.7. For all n, each of the classes Σ_n and Π_n is contained both in Σ_{n+1} and Π_{n+1}.

By extension of terminology, we shall often call Σ_n any formula logically equivalent to a Σ_n formula in the sense of our official definition. The next lemma obviously follows from the rules of predicate logic.

LEMMA 1.8. Modulo logical equivalence:

1. The classes Σ_n and Π_n are closed under $\vee, \wedge$.
2. $A \in \Sigma_n \iff \neg A \in \Pi_n$, and dually.
3. The class Π_n is closed under the universal quantification,
   the class Σ_n is closed under the existential quantification.

From the computational point of view, the most interesting class of formulas is Σ_1.

LEMMA 1.9. Any predicate definable by a Σ_1 formula is recursively enumerable.

PROOF. We describe an effective evaluation procedure associated with a given Σ_1 formula B of the form $\exists x_1 \ldots \exists x_n A(x_1, \ldots, x_n)$, with A elementary.

Choose some effective enumeration of sequences of natural numbers $\langle x_1, \ldots, x_n \rangle$ of length n. Evaluate $A[x_1, \ldots, x_n]$ for all such tuples in the order of this enumeration using the evaluation procedure for elementary formulas (Lemma 1.3). Terminate and return 1 as soon as a tuple $\langle x_1, \ldots, x_n \rangle$ is found such that $A[x_1, \ldots, x_n]$ evaluates to 1; loop forever, if there is no such a tuple.

Clearly, the above procedure terminates if and only if B is true. Thus, the predicate defined by B is the domain of a computable function, q.e.d.

1.15. Arithmetic as a programming language. The evaluation procedure gives a Σ_1 formula $A(a_1, \ldots, a_n)$ the interpretation of a program that on input $x_1, \ldots, x_n$ outputs 1 if and only if $A[x_1, \ldots, x_n]$ is true. Thus, arithmetic can be considered as a specific programming language. The analogy goes deeper than it might seem at the first sight, so let us briefly discuss some of the apparent differences and similarities between arithmetic and the usual programming languages.

First of all, standard programming languages, such as FORTRAN or C^{++}, are *operational* in the sense that their basic constructions are operators telling the computer what we want it to *do*, rather than the statements asserting the truth or falsity of particular propositions. In contrast, arithmetical language, by its nature, is *declarative*. The operational interpretation is provided by the evaluation procedure for Σ_1 formulas. There is a good deal of freedom in the choice of this procedure. For example, one may search for an $x \leq n$ such that $A[x]$ holds by looking through $x = n, n-1, \ldots, 0$ rather than in the opposite order — the result will be the same. This is parallel to the situation with compiling high-level programming languages: specification usually allows for numerous different realizations of the compilers. In fact, our evaluation procedure, when formalized, becomes a compiler from the high-level arithmetical language to the low-level language of Turing machines.

Second, the arithmetical "programs" evaluate predicates rather than the functions. It has long been understood in recursion theory and programming that this is not a serious restriction: computable functions can be defined on the basis of the notion of r.e. set. The correspondence is provided by the Graph theorem **??**. On practice this means that we can always interpret the process of computation of a function as the (unbounded) search for its value, which falls within the formalism of the "arithmetic programming."

With this understanding, the operational reading of Σ_1 formulas reveals many other common features with the usual high-level programming languages. Thus, boolean logic provides for a version of the **if-then-else** operator, and bounded quantifiers, formalizing bounded search, are the analogs of loops, or **for** operators. Finally, the unbounded existential quantifier is naturally interpreted as the unbounded search or, if you like, as a **while** operator.

Since the arithmetical language has built-in constructs for the main programming operators, it is not surprising that it is as powerful as all other universal languages, that is, *all computable functions and relations can be represented in arithmetic*. Moreover, to program within the formalism of arithmetic is generally easier than to operate the low-level Turing machines, especially because the language of first order logic is relatively well attuned to the ordinary human language.

Another difference between arithmetic programming and languages such as PASCAL is that in arithmetic there is only one type of data — nonnegative integers —

and correspondingly not too many basic functions. However, the ordinary types, such as binary strings, lists, arrays, records, etc. can always be introduced via encoding — after all, within the real computers all these objects also look more or less like strings of zeros and ones! To develop some such coding is our objective in the next two subsections.

1.16. Bounded definitional extensions. In order to enable arithmetic to talk about objects other than the numbers and formulate complex mathematical facts or constructions, one uses the standard method of definitional extensions. However, it is not difficult to see that, in general, the quantifier complexity of the formulas in the extended language differs from that of their translations in the original language. For us it will be important that the complexity of the translated bounded formulas is preserved. For this reason we introduce *bounded definitional extensions.* Under the programming interpretation of arithmetical language, the role of defined predicates and functions is similar to that of *subroutines.*

Let $\mathcal{L}$ be a first order language containing that of PA. Bounded formulas of $\mathcal{L}$ are defined similarly to the elementary formulas and denoted $\Delta_0(\mathcal{L})$. It is understood that any function symbols of $\mathcal{L}$ may be used in bounding terms.

Let T be a theory formulated in $\mathcal{L}$, and let $A(b_1, \dots, b_n)$ be a bounded formula with all the free variables shown. We add to the signature of $\mathcal{L}$ a new n-ary predicate symbol P, and let a theory U be obtained from T by adding the axiom

$$P(b_1, \dots, b_n) \leftrightarrow A(b_1, \dots, b_n).$$

U is called a bounded definitional extention of T by the predicate symbol P, and the language of U is denoted $\mathcal{L}(P)$.

Being a particular kind of definitional extension, the theory U is conservative over T and admits the canonical translation $(\cdot)^-$ of $\mathcal{L}(P)$ formulas into $\mathcal{L}$ formulas. Essentially, B^- is the result of replacing all occurrences of the subformula P in B by A, possibly with some bound variables of A renamed. In addition to all the usual properties of the canonical translation (see Section 1.1.7) we have the following one.

LEMMA 1.10. $B \in \Delta_0(\mathcal{L}(P))$ implies $B^- \in \Delta_0(\mathcal{L})$.

PROOF. Substituting a bounded formula in a bounded formula we obtain a bounded formula, q.e.d.

Now we consider bounded definitional extensions of T by new function symbols. Let $F(a_1, \dots, a_n, b)$ be a bounded formula of $\mathcal{L}$ such that T proves

1. $\exists!\, y\, F(a_1, \dots, a_n, y)$;
2. $F(a_1, \dots, a_n, b) \to b \leq t(a_1, \dots, a_n)$, for some $\mathcal{L}$ term t.

Then we can introduce an n-ary function symbol f into the language of T and add the axiom

$$F(a_1, \dots, a_n, f(a_1, \dots, a_n)),$$

thus obtaining a conservative definitional extension U of T formulated in the language $\mathcal{L}(f)$. Notice that U proves that the new function is bounded by a suitable $\mathcal{L}$ term:

$$f(a_1, \ldots, a_n) \leq t(a_1, \ldots, a_n).$$

General properties the canonical translation $(\cdot)^-$ obviously hold for U. In addition we have the following lemma.

LEMMA 1.11. If B is a bounded formula in $\mathcal{L}(f)$, then B^- is provably equivalent to a bounded $\mathcal{L}$ formula within T.

PROOF. It will be enough to prove this lemma for atomic formulas B. The latter are of the form $P(t_1, \ldots, t_k)$, for some $\mathcal{L}(f)$ terms $t_1, \ldots, t_k$ and a predicate symbol P of $\mathcal{L}$. As usual, we argue by induction on the total number of occurrences of the symbol f in $t_1, \ldots, t_k$.

Assume, for example, that f occurs in t_1. Choose an innermost such occurrence; then t_1 can be rewritten in the form $t_1'(f(s_1, \ldots, s_n))$, where the terms s_i do not contain the symbol f, and t_1' has one occurrence of f less than t_1. Since f is provably majorized by a certain $\mathcal{L}$ term t, the canonical translation of $P(t_1, \ldots, t_k)$ is provably equivalent to the formula

$$\exists x \leq t(s_1, \ldots, s_n)\, [(P(t_1'(x), t_2, \ldots, t_k))^- \wedge F(s_1, \ldots, s_n, x)],$$

where $(P(t_1'(x), t_2, \ldots, t_k))^-$ is provably equivalent to a bounded formula by the induction hypothesis, q.e.d.

Of course, bounded definitional extensions can be applied repeatedly. In this way several function and predicate symbols can be introduced into a given theory.

Notice that by Lemmas 1.10 and 1.11, the arithmetical hierarchy is invariant w.r.t. bounded definitional extensions of the arithmetical language. That is, we can identify Σ_n and Π_n formulas in the original and in the extended language (via the canonical translation). Also notice that any term in a bounded definitional extension of the language of arithmetic is majorized by a (monotonically increasing) arithmetical term; hence it has mutiexponential rate of growth and any *term definable* function in this language can also be introduced in a suitable bounded definitional extension.

1.17. Coding binary strings, words, and sequences. Now we are ready to develop some basic coding machinery. The particular details of the coding are inessential. For now, it will only be essential for us that all the functions and predicates defined below are introduced in a bounded definitional extension of TA, i.e., in the standard model. We start with the coding of binary strings.

BINARY STRINGS. Recall that each positive number x can be uniquely represented in the form

$$x = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2 + a_0,$$

where $a_0, \ldots, a_n \in \{0,1\}$ and $a_n \neq 0$. The string of digits $a_n a_{n-1} \ldots a_0$ is called the binary expansion of x; a_i is the $i+1$ st bit of x (denoted $bit(x,i)$). We also denote $|x| = n$; the binary length of x equals $|x|+1$.

Thus, binary expansion provides a one-to-one correspondence between binary strings and positive natural numbers: a string $a_{n-1} \ldots a_0$ is coded by the number whose binary expansion is $1a_{n-1} \ldots a_0$; in particular, the empty string Λ is coded by 1. Notice that $|x|$ equals to the length of the string coded by x.

We start with some obvious abbreviations (confusing the alphabets of free and bound variables):

$$\begin{aligned} x \neq y &:\leftrightarrow \neg\, x = y \\ x < y &:\leftrightarrow x \leq y \wedge x \neq y \\ x - y = z &:\leftrightarrow (y \leq x \wedge x = z + y) \vee (\neg\, y \leq x \wedge z = 0) \end{aligned}$$

The term $S(S(\ldots S(0)\ldots))$ (n times) that names the number n will be denoted $\overline{n}$ or just n, when there is no danger of confusing it with a variable. Such terms will be called *numerals*.

Now we define the set of binary strings, the length of the string function, the concatenation function, and the $bit(x,i)$ function.

$$\begin{aligned} String(x) &:\leftrightarrow x \neq 0 \\ |x| = y &:\leftrightarrow (x = 0 \wedge y = 0) \vee (2^y \leq x \wedge x < 2^{y+1}) \\ x * y = z &:\leftrightarrow z = x \cdot 2^{|y|} + (y - 2^{|y|}) \\ bit(x,i) = y &:\leftrightarrow y \leq 1 \wedge \exists u \leq x\, \exists v < 2^i\ x = 2^{i+1} \cdot u + 2^i \cdot y + v \end{aligned}$$

Notice that the function $x * y$ returns a meaningless result, if x or y do not code any strings (i.e. equal 0). Also notice that the functions $|\cdot|$, $*$, and bit satisfy all the requirements of bounded definitional extensions; the rate of growth bounds are obvious, in each case.

Apart from binary strings, we also have to be able to work with the *words* in an arbitrary finite alphabet Σ, and finite *sequences* of such words. Elements of the alphabet Σ will be called *characters*.

Since Σ is finite, we can choose in advance a certain constant $c \in \mathbf{N}$ and code all the characters as different binary strings of length c (in an arbitrary way). We shall call such strings *bytes*. *Character words*, i.e., words in Σ, will be coded as binary strings by concatenating the bytes corresponding to their characters. Sequences of words will be coded as the words in the alphabet Σ extended by a special separator symbol (which will belong to the bytes but not to the characters). Finally, everything will be coded into natural numbers via the standard binary coding decribed in the previous sections. The numerical code, or the *Gödel number*, of an object a will be denoted $\ulcorner a \urcorner$. We do not distinguish between Gödel numbers and the corresponding numerals. Now we turn to the formal definitions.

CHARACTERS AND BYTES. Let $\Sigma = \{C_0, \ldots, C_n\}$. Pick a number c such that $2^c \geq n + 2$. To the symbols of Σ we associate, e.g., the following codes: $\ulcorner C_i \urcorner = 2^c + i$, for $0 \leq i \leq n$, and $\ulcorner ; \urcorner = 2^c + n + 1$ (an extra separator symbol). For whatever comes below, the particular assignment of these codes is not essential, except that all of them must correspond to different binary strings of length c. Now we $\Delta_0(\exp)$ define the sets of characters and bytes as follows.

$$\begin{aligned} Char(x) &:\leftrightarrow x = \ulcorner C_0 \urcorner \vee \cdots \vee x = \ulcorner C_n \urcorner \\ Byte(x) &:\leftrightarrow String(x) \wedge |x| = \overline{c} \end{aligned}$$

Finite sets will often be defined similarly to the predicate $Char(x)$ above, that is, by simply enumerating all of their elements. We shall not always write such formulas out explicitly.

WORDS. We define the set of words, that is, sequences of bytes, length of word function $\|\cdot\|$, subword relation, and a function $byte(x, i)$ that recovers the $(i+1)$-st byte of a word x. Notice that the concatenation of words, in our coding, happens to be the same operation as the concatenation of the corresponding binary strings. The empty word Λ is coded by 1.

$$\begin{aligned} Word(x) &:\leftrightarrow String(x) \wedge \exists k \leq x\, |x| = \overline{c} \cdot k \\ \|x\| = y &:\leftrightarrow (Word(x) \wedge \overline{c} \cdot y = |x|) \vee (\neg Word(x) \wedge y = 0) \\ x \subseteq_w y &:\leftrightarrow Word(x) \wedge Word(y) \wedge \\ &\quad \exists v, w \leq y\, (Word(v) \wedge y = v * x * w) \\ byte(x, i) = y &:\leftrightarrow [Word(x) \wedge Byte(y) \wedge \\ &\quad \exists v, w \leq x\, (Word(v) \wedge \|v\| = i \wedge x = v * y * w)] \vee \\ &\quad [\neg(Word(x) \wedge i < \|x\|) \wedge y = 0] \\ Empty(x) &:\leftrightarrow x = 1 \end{aligned}$$

Character words are then defined as follows:

$$ChWord(x) :\leftrightarrow Word(x) \wedge \forall i < \|x\|\, Char(byte(x, i))$$

SEQUENCES. Sequences $\langle w_1, \ldots, w_s \rangle$ of character words are coded as the words of the form $w_1; w_2; \ldots; w_s$ where ; is the extra separator symbol. The empty sequence $\langle\,\rangle$, unlike the empty word, will be coded by 0. Also notice that for any character word w, $\ulcorner \langle w \rangle \urcorner = \ulcorner w \urcorner$, in particular, $\ulcorner \langle \Lambda \rangle \urcorner = 1$.

Thus, the predicate expressing that x codes a sequence, the concatenation function (denoted $x; y$), the subsequence relation, and the predicate "the char-

acter word x occurs in the sequence y" can be defined as follows.

$$\begin{aligned}
Seq(x) &:\leftrightarrow x = 0 \vee [Word(x) \wedge \\
&\quad \forall i < \|x\| \, (Char(byte(x,i)) \vee byte(x,i) = \ulcorner;\urcorner)] \\
x;y = z &:\leftrightarrow (x = 0 \wedge z = y) \vee (y = 0 \wedge z = x) \vee \\
&\quad (x \neq 0 \wedge y \neq 0 \wedge z = x * \ulcorner;\urcorner * y) \\
x \subseteq_s y &:\leftrightarrow Seq(x) \wedge Seq(y) \wedge \\
&\quad \exists u, v \leq y \, (Seq(u) \wedge Seq(v) \wedge y = u;x;v) \\
x \in_s y &:\leftrightarrow ChWord(x) \wedge x \subseteq_s y
\end{aligned}$$

1.18. Simulating Turing machines in arithmetic. Our goal in this section is the following fundamental theorem.

THEOREM 1.12. Every r.e. predicate is definable in the standard model of arithmetic by a Σ_1 formula.

PROOF. The idea of the proof is quite natural: for a given Turing machine $\mathcal{T}$ we want to express the predicate $Comp_{\mathcal{T}}(x,z)$ saying, roughly, that "x codes an input of $\mathcal{T}$, and z codes the protocol of a (terminating) computation of $\mathcal{T}$ on input x." Since the protocol z explicitly contains all necessary information about the computation, we may expect $Comp_{\mathcal{T}}(x,z)$ to be $\Delta_0(\exp)$ definable (all quantifiers will be bounded by multiexponential functions of z). And then, the predicate expressing the termination of the machine $\mathcal{T}$ on input x could be defined by $\exists z \, Comp_{\mathcal{T}}(x,z)$. To implement this idea, we have to spell out in the language of arithmetic all the definitions of Section **??**.

Consider a Turing machine $\mathcal{T} = \langle A, Q, P\rangle$, with $A = \{S_0, \ldots, S_n\}$ and $Q = \{q_0, \ldots, q_m\}$. The alphabet of characters Σ associated with $\mathcal{T}$ consists of tape symbols, inner state symbols, and extra symbols L and R, that is, $\Sigma = A \cup Q \cup \{L, R\}$. We work with the natural Gödel numbering of Σ described in the previous section. The predicates $A(x)$, $Q(x)$ defining the finite sets of tape and inner state symbols are introduced by enumerating their elements, in the same way as the predicate $Char(x)$ defining the set of characters is. The set of tape words can then be defined by

$$TpWord(x) \quad :\leftrightarrow \quad Word(x) \wedge \forall i < \|x\| \; A(byte(x,i))$$

COMMANDS. Any command w of a Turing machine is a particular character word of length 4. It is obtained by concatenation of the corresponding characters, that is, for $w = q_i S_j L q_r$, one has $\ulcorner w \urcorner = \ulcorner q_i \urcorner * \ulcorner S_j \urcorner * \ulcorner L \urcorner * \ulcorner q_r \urcorner$, and similarly for the other types of commands. Since the program of the Turing machine consists of only finitely many commands, say $w_0, \ldots, w_k$, we can also define the set of commands by enumeration:

$$P(x) \quad :\leftrightarrow \quad x = \ulcorner w_0 \urcorner \vee \cdots \vee x = \ulcorner w_k \urcorner$$

CONFIGURATIONS AND TRANSITIONS. To better understand the following definitions the reader is invited to look up Section **??**. The set of configurations of a Turing machine is easily defined as follows.

$$\begin{aligned} Config(z) \quad :\leftrightarrow \quad & Word(z) \wedge \exists u, v, q \subseteq_w z \\ & (TpWord(u) \wedge TpWord(v) \wedge \\ & Q(q) \wedge \neg Empty(v) \wedge z = u * q * v) \end{aligned}$$

The transition relation $x \underset{\mathcal{T}}{\longrightarrow} y$ asserting that the machine $\mathcal{T}$ transforms a configuration x into a configuration y (in one step) can be spelled out by the following formula:

$$\begin{aligned} & Config(x) \wedge Config(y) \wedge \\ & \exists u, v, p, q, a, b \subseteq_w x * y \\ & \quad [TpWord(u) \wedge TpWord(v) \wedge Q(p) \wedge Q(q) \wedge A(a) \wedge A(b) \wedge \\ & \qquad [(x = u * p * a * v \wedge y = u * q * b * v \wedge P(p * a * b * q)) \\ & \qquad \vee (x = u * b * p * a * v \wedge y = u * q * b * a * v \wedge P(p * a * \ulcorner L \urcorner * q)) \\ & \qquad \vee (x = p * a * v \wedge y = q * \ulcorner S_0 \urcorner * a * v \wedge P(p * a * \ulcorner L \urcorner * q)) \\ & \qquad \vee (x = u * p * a * v \wedge \neg Empty(v) \wedge y = u * a * q * v \wedge P(p * a * \ulcorner R \urcorner * q)) \\ & \qquad \vee (x = u * p * a \wedge y = u * a * q * \ulcorner S_0 \urcorner \wedge P(p * a * \ulcorner R \urcorner * q)) \\ & \qquad] \\ & \quad] \end{aligned}$$

This formula directly mirrors the definition of the $\underset{\mathcal{T}}{\longrightarrow}$ relation in Section **??**.

COMPUTATIONS. Now we define a formula $Init(x, z)$ expressing the predicate "z is an initial configuration with an input word x," a formula $Stop(z)$ "z is a stopping configuration," and a formula $Comp(x, z)$ expressing that z is the protocol of a terminating computation on input x.

$$\begin{aligned} Init(x, z) \quad :\leftrightarrow \quad & Config(z) \wedge z = \ulcorner q_0 \urcorner * x \\ Stop(z) \quad :\leftrightarrow \quad & Config(z) \wedge \neg \exists q, a, p\, (Q(q) \wedge A(a) \wedge P(p) \wedge \\ & q * a \subseteq_w z \wedge \exists v \subseteq_w p\ p = q * a * v) \\ Comp(x, z) \quad :\leftrightarrow \quad & Seq(z) \wedge \exists v \in_s z\ Stop(v) \wedge \forall u, v, w \leq z \\ & (z = u; v; w \wedge ChWord(v) \rightarrow \\ & \qquad (Init(x, v) \vee \exists y \in_s u\ y \underset{\mathcal{T}}{\longrightarrow} v)) \end{aligned}$$

Now we complete the proof of Theorem 1.12. Let $R(x_1, \ldots, x_n)$ be an n-ary r.e. predicate. Then for some Turing machine $\mathcal{T}$, whose alphabet contains at least the symbols $\{0, 1, \sharp\}$, $\mathcal{T}$ terminates on input

$$code(k_1) \sharp \cdots \sharp code(k_n)$$

iff $R(k_1, \ldots, k_n)$ holds. (Here $code(k_i)$ denotes the binary expansion of the number k_i understood as a string of symbols 0 and 1 of the alphabet of $\mathcal{T}$.) The function $code(x)$ can be introduced by

$$\begin{aligned} code(x) = y \quad :\leftrightarrow \quad & ChWord(y) \wedge \|y\| = |x| + 1 \wedge \\ & \forall i \leq |x| \, [(bit(x,i) = 1 \to byte(y,i) = \ulcorner 1 \urcorner) \\ & \qquad \wedge (bit(x,i) = 0 \to byte(y,i) = \ulcorner 0 \urcorner)] \end{aligned}$$

Therefore we have

$$R(k_1, \ldots, k_n) \iff \mathbf{N} \vDash Termin[k_1, \ldots, k_n],$$

where the termination predicate $Termin(a_1, \ldots, a_n)$ is defined by the Σ_1 formula

$$\exists z \, Comp(code(a_1) * \ulcorner \sharp \urcorner * \cdots * \ulcorner \sharp \urcorner * code(a_n), z),$$

q.e.d.

Theorem 1.12 gives an important computational characterization of arithmetical Σ_1-predicates. Post's theorem immediately yields the following corollary.

COROLLARY 1.13. A predicate is decidable, iff it is definable in the standard model both by a Σ_1 and by a Π_1 formula.

The following theorem, together with Lemma 1.5, gives a computational characterization of elementary predicates.

THEOREM 1.14. Every predicate decidable in multiexponential number of steps of the length of the input is elementary.

PROOF. To simplify notations, we only give a proof for unary predicates. Let $R(x)$ be a predicate decidable in multiexponential number of steps of the length of x, i.e., of $|x| + 1$. Consider a Turing machine $\mathcal{T}$ such that, depending on whether $R(x)$ is true or false, on input coding the binary expansion of x, $\mathcal{T}$ terminates in a configuration of the form Xq_i1Y or Xq_i0Y after $\leq 2_m^{|x|+1}$ steps of the computation. $R(x)$ can be defined by the formula

$$\begin{aligned} & \exists z \, (Comp(code(x), z) \wedge \\ & \qquad \exists q, u, v \subseteq_w z \, (Q(q) \wedge Stop(u * q * \ulcorner 1 \urcorner * v) \wedge u * q * \ulcorner 1 \urcorner * v \in_s z). \end{aligned}$$

Clearly, for the proof it is sufficient to give a multiexponential bound on z.

Let N be the number of steps in a computation z, thus $N \leq 2_m^{|x|+1}$. Notice that, for all $i < N$, $\|(z)_{i+1}\| \leq \|(z)_i\| + 1$, because the length of a configuration of a Turing machine cannot increase more than by 1 in one step. Hence, $\|(z)_i\| \leq \|(z)_0\| + i$ for all i, and

$$\|z\| \leq N + \sum_{i=0}^{N} (\|(z)_0\| + i) \leq \|(z)_0\| \cdot (N+1) + O(N^2).$$

We may also assume that $\|(z)_0\| = |x| + 3$ for our particular way of coding initial configurations, so

$$\|z\| \leq (|x| + 3) \cdot (2_m^{|x|+1} + 1) + O((2_m^{|x|+1})^2) \leq 2_{m+1}^{|x|+1}$$

for sufficiently large x. Now we recall that $|z| = \|z\| \cdot c$, for some constant c depending on our coding of characters, and $z \leq 2^{|z|+1}$. Hence z can be bounded by a term of the form

$$2^{c \cdot 2_{m+1}^{|x|+1} + d}$$

for some constants c, d, and m, which has the required rate of growth, q.e.d.

1.19. Gödel's First Incompleteness Theorem. Having done all the necessary technical work in the previous section, now we are ready to obtain several forms of Gödel's First Incompleteness Theorem.

THEOREM 1.15. The set of all true arithmetical sentences is not r.e.

PROOF. Let K be an r.e. nonrecursive set, and let $K(a)$ be its Σ_1 definition in the standard model. For any $n \in \mathbf{N}$ we have

$$n \notin K \iff \mathbf{N} \vDash \neg K(\overline{n}).$$

If the set of true arithmetical sentences were r.e., so would be the set of all n such that $\mathbf{N} \vDash \neg K(\overline{n})$, and hence, the complement of K. Then, by Post's Theorem, K would be decidable, a contradiction, q.e.d.

COROLLARY 1.16. The full true arithmetic TA is neither r.e., nor has an r.e. axiomatization.

PROOF. An r.e. axiomatization of TA would generate a recursive enumeration of the set of all true arithmetical sentences, q.e.d.

Let T be a consistent arithmetical theory. T is *semantically complete*, if every true arithmetical sentence is provable in T. T is *syntactically complete*, if for every sentence A, either $T \vdash A$ or $T \vdash \neg A$. A theory T is *sound*, if $\mathbf{N} \vDash T$, that is, if all theorems of T are true. Obviously, soundness implies consistency. We also have the following lemma.

LEMMA 1.17.

1. Semantically complete consistent theories are syntactically complete.
2. If T is syntactically complete and sound, then T is semantically complete.
3. Any semantically complete consistent theory is deductively equivalent to TA.

Hence, for sound theories the notions of syntactical and semantical completeness are equivalent (and often simply called *completeness*).

COROLLARY 1.18.

1. Consistent r.e. theories are semantically incomplete.
2. Sound r.e. theories are (syntactically and semantically) incomplete.
3. PA is incomplete.

REMARK 1.19. The previous corollary is sometimes called a weak form of Gödel's First Incompleteness Theorem. It can be improved in several ways. First, the assumptions of soundness and recursive enumerability of theories can be weakened. Second, concrete meaningful examples of true unprovable sentences can be exhibited. These improvements will be obtained in the next section using slightly more advanced methods.

2. Provability and Computability

In the previous section we arrived at the fundamental relationship between the notions of Computability and Truth in formal arithmetic: a predicate is r.e., iff it is definable in the standard model by a Σ_1-formula. In this section we study two classical models of computation related to the concept of Provability: the first one is the *numerability* of r.e. sets, and the second one is the *dual numerability* of pairs of disjoint r.e. sets (or, equivalently, partial 0–1-valued functions). Whereas in the case of Σ_1-definability the computation mechanism corresponds to the evaluation procedure for Σ_1-formulas, here it can be understood as a *proof-search* procedure. We shall see that, under some minimal assumptions on the class of theories under consideration, this mechanism provides adequate models of computation. As a payoff we shall obtain the classical results in the theory of formal systems: Gödel-Rosser incompleteness theorem, undecidability and recursive inseparability results.

2.20. Σ_1-completeness. Let Γ be a class of arithmetical sentences. A theory T is called Γ-*complete*, if $T \vdash A$, for all $A \in \Gamma$ such that $\mathbf{N} \vDash A$. Thus, Γ-completeness is a restricted version of semantical completeness introduced in Section 2.1.19. T is Γ-*sound*, if for all $A \in \Gamma$, $T \vdash A$ implies $\mathbf{N} \vDash A$. Obviously, since all axioms of PA are true, it is Γ-sound for any Γ.

Let BA (*Basic Arithmetic*) be the arithmetical theory obtained from PA by replacing the induction axom schema IA by the single axiom

P11. $a \leq b \vee b \leq a$.

By induction on b it is not difficult to show that P11 is provable in PA. Therefore we obtain the following lemma.

LEMMA 2.1. BA is a finitely axiomatized subtheory of PA.

Basic arithmetic only plays a technical role in this book. It is very weak as far as the provability of universal statements is concerned. For example, it does not prove the commutativity of addition and multiplication (see Excercise **??**). Nonetheless, the following important theorem shows that BA, and all the more so PA, has the full power of reasoning about the existential statements (Σ_1-sentences).

THEOREM 2.2. *BA is Σ_1-complete.*

PROOF. The idea of the proof is simple: truth of a Σ_1-sentence can be effectively verified by means of the evaluation procedure described in Lemma 1.9. Such a verification essentially is a BA-proof of the sentence. Below we give a rather pedantic elaboration of this idea, because later we would like to formalize the very proof of Theorem 2.2 inside (a sufficiently weak fragment of) PA.

LEMMA 2.3. *For any $m, n \in \mathbf{N}$, BA proves*

(i) $\overline{m} + \overline{n} = \overline{m+n}$

(ii) $\overline{m} \cdot \overline{n} = \overline{m \cdot n}$

(iii) $\exp(\overline{n}) = \overline{\exp(n)}$

PROOF. Each of the statements is proved by external induction on n. Recall that $\overline{0}$ is 0 and $\overline{n+1}$ is $S(\overline{n})$.

(i) Basis: $\overline{m} + 0 = \overline{m}$, by P3.

Induction step. Assume we have a proof of $\overline{m} + \overline{n} = \overline{m+n}$. Consider the following sequence of formulas:

1. $\overline{m} + \overline{n} = \overline{m+n}$ (assumption)
2. $S(\overline{m} + \overline{n}) = S(\overline{m+n})$ (1, equality)
3. $\overline{m} + S(\overline{n}) = S(\overline{m} + \overline{n})$ (P4)
4. $\overline{m} + S(\overline{n}) = S(\overline{m+n})$ (2, 3, equality)

This sequence essentially is a BA-proof of $\overline{m} + \overline{n+1} = \overline{m+n+1}$ from the assumption $\overline{m} + \overline{n} = \overline{m+n}$. Appending it to the proof of $\overline{m} + \overline{n} = \overline{m+n}$ given by the induction hypothesis (and inserting a few obvious intermediate steps) we obtain a BA-proof of $\overline{m} + \overline{n+1} = \overline{m+n+1}$.

Similar proofs of (ii) and (iii) are left as an easy excercise for the reader, q.e.d.

LEMMA 2.4. *For any arithmetical term $t(b_1, \ldots, b_m)$ and any $k_1, \ldots, k_m, l \in \mathbf{N}$,*

$$\mathbf{N} \vDash t(k_1, \ldots, k_m) = l \quad \Longrightarrow \quad BA \vdash t(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}.$$

PROOF. External induction on the build-up of t. If t is a variable or a constant 0, the claim is obvious. For compound terms the claim follows by induction hypothesis from Lemma 2.3. For example, if t has the form $t_1 + t_2$, then for

some $l_1, l_2 \in \mathbf{N}$ the formulas $t_1(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}_1$ and $t_2(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}_2$ are provable. Now we can construct the following derivation:

1. $t_1(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}_1$ (assumption)
2. $t_2(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}_2$ (assumption)
3. $t_1(\overline{k}_1, \ldots, \overline{k}_m) + t_2(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l}_1 + \overline{l}_2$ (1, 2, equality)
4. $\overline{l}_1 + \overline{l}_2 = \overline{l_1 + l_2}$ (Lemma 2.3)
5. $t_1(\overline{k}_1, \ldots, \overline{k}_m) + t_2(\overline{k}_1, \ldots, \overline{k}_m) = \overline{l_1 + l_2}$ (3, 4, equality)

Successor, multiplication, and exponentialtion functions are treated similarly, q.e.d.

LEMMA 2.5. For all $m, n \in \mathbf{N}$,

(i) If $m \leq n$, then $BA \vdash \overline{m} \leq \overline{n}$;

(ii) If $m \neq n$, then $BA \vdash \neg\overline{m} = \overline{n}$;

(iii) If $m < n$, then $BA \vdash \neg\overline{n} \leq \overline{m}$.

PROOF. (i) By external induction on n.

Basis. For $n = m$ the claim amounts to $BA \vdash \overline{m} \leq \overline{m}$. This is straightforwardly proved by a separate (external) induction on m, using P9, P10, or can be directly inferred from the axiom $a \leq b \vee b \leq a$ substituting $\overline{m}$ for a and b.

Induction step. Assume $m \leq n + 1$, then either $m \leq n$ or $m = n + 1$. If $m \leq n$, then $BA \vdash \overline{m} \leq \overline{n}$ by the induction hypothesis. If $m = n + 1$, then $\overline{m}$ is graphically the same as $S(\overline{n})$, so $BA \vdash \overline{m} = S(\overline{n})$ by the equality axioms. In any case,

$$BA \vdash \overline{m} \leq \overline{n} \vee \overline{m} = S(\overline{n}).$$

Hence $BA \vdash \overline{m} \leq S(\overline{n})$ by P10, q.e.d.

(ii) Without loss of generality we may assume that $m < n$. The argument goes by induction on m. If $m = 0$, then $\overline{n}$ has the form $S(\overline{n-1})$, and the result follows by P1. For $m > 0$, by the induction hypothesis, there is a BA-proof of $\neg\overline{m-1} = \overline{n-1}$. Appending to that proof the following sequence of formulas

1. $\neg\overline{m-1} = \overline{n-1}$ (assumption)
2. $\overline{m} = \overline{n} \to \overline{m-1} = \overline{n-1}$ (P2)
3. $\neg\overline{m} = \overline{n}$ (1, 2)

we obtain a BA-proof of $\neg\overline{m} = \overline{n}$, q.e.d.

(iii) The argument goes by external induction on m.

Basis. Assume $0 = m < n$. Then $\overline{n} \leq 0$ implies $\overline{n} = 0$ by P9, which implies a contradiction by (ii).

Induction step. Assume $m + 1 < n$. Then $\overline{n} \leq S(\overline{m})$ implies $\overline{n} \leq \overline{m} \vee \overline{n} = S(\overline{m})$ by P10. However, $\overline{n} \leq \overline{m}$ yields a contradiction by the induction hypothesis, and $\overline{n} = S(\overline{m})$ implies a contradiction by (ii). Hence, $BA \vdash \neg\overline{n} \leq S(\overline{m})$, q.e.d.

LEMMA 2.6. For each $m \in \mathbf{N}$, BA proves

$$a \leq \overline{m} \leftrightarrow (a = 0 \vee \cdots \vee a = \overline{m}).$$

PROOF. From P9, P10 by external induction on m, q.e.d.

LEMMA 2.7. For any bounded formula $A(b_1, \ldots, b_m)$ and any $k_1, \ldots, k_m \in \mathbf{N}$,

(i) $\mathbf{N} \vDash A(k_1, \ldots, k_m) \;\Rightarrow\; BA \vdash A(\overline{k_1}, \ldots, \overline{k_m})$;

(ii) $\mathbf{N} \nvDash A(k_1, \ldots, k_m) \;\Rightarrow\; BA \vdash \neg A(\overline{k_1}, \ldots, \overline{k_m})$.

PROOF. (i) and (ii) are proved simultaneously by induction on the build-up of A. We consider the following cases.

1. A is an atomic formula of the form $t_1(b_1, \ldots, b_m) = t_2(b_1, \ldots, b_m)$.

If $\mathbf{N} \vDash A(k_1, \ldots, k_m)$, then for some $l \in \mathbf{N}$, by Lemma 2.4, we have BA-proofs of $t_1(\overline{k_1}, \ldots, \overline{k_m}) = \overline{l}$ and $t_2(\overline{k_1}, \ldots, \overline{k_m}) = \overline{l}$. From these proofs we obtain a proof of $t_1(\overline{k_1}, \ldots, \overline{k_m}) = t_2(\overline{k_1}, \ldots, \overline{k_m})$ by the equality axioms.

If $\mathbf{N} \nvDash A(k_1, \ldots, k_m)$, then for some $l_1 \neq l_2$ we have BA-proofs of $t_1(\overline{k_1}, \ldots, \overline{k_m}) = \overline{l_1}$ and $t_2(\overline{k_1}, \ldots, \overline{k_m}) = \overline{l_2}$, by 2.4. Lemma 2.5(ii) yields a proof of $\neg \overline{l_1} = \overline{l_2}$, hence we obtain a proof of

$$\neg\, t_1(\overline{k_1}, \ldots, \overline{k_m}) = t_2(\overline{k_1}, \ldots, \overline{k_m})$$

using the equality axioms, q.e.d.

2. A is an atomic formula of the form $t_1(b_1, \ldots, b_m) \leq t_2(b_1, \ldots, b_m)$.

In this case, a similar argument, using Lemma 2.5 (i) and (iii), works.

3. A has the form $B \to C$ or $\neg B$.

In this case, the claim immediately follows from the induction hypothesis for B and C.

4. A has the form $\forall v \leq t\, B(v, b_1, \ldots, b_m)$.

(i) Assume $\mathbf{N} \vDash A(k_1, \ldots, k_m)$. By Lemma 2.4, for some $l \in \mathbf{N}$, BA proves $t(\overline{k_1}, \ldots, \overline{k_m}) = \overline{l}$. Hence, for all $k \leq l$, $\mathbf{N} \vDash B(k, k_1, \ldots, k_m)$, and by induction hypothesis we have BA-proofs of the formulas $B(\overline{k}, \overline{k_1}, \ldots, \overline{k_m})$, for all $k \leq l$. Now we append to these proofs the following formulas:

1.	$(a = 0 \vee \cdots \vee a = \overline{l}) \to B(a, \overline{k_1}, \ldots, \overline{k_m})$	(ind. hyp., equality)
2.	$a \leq \overline{l} \to B(a, \overline{k_1}, \ldots, \overline{k_m})$	(1, 2.6)
3.	$a \leq t(\overline{k_1}, \ldots, \overline{k_m}) \to B(a, \overline{k_1}, \ldots, \overline{k_m})$	(2, equality)
4.	$\forall v\, (v \leq t(\overline{k_1}, \ldots, \overline{k_m}) \to B(v, \overline{k_1}, \ldots, \overline{k_m}))$	(3)

The resulting sequence can be transformed into a proof of $A(\overline{k_1}, \ldots, \overline{k_m})$ by inserting some obvious logical steps.

(ii) Assume $\mathbf{N} \nvDash A(k_1, \ldots, k_m)$. Then for some $k \leq l = t(k_1, \ldots, k_m)$ one has $\mathbf{N} \nvDash B(k, k_1, \ldots, k_m)$, and hence

$$BA \vdash \neg B(\overline{k}, \overline{k_1}, \ldots, \overline{k_m}),$$

by the induction hypothesis. To that proof we append the following formulas:

1. $\neg B(\overline{k}, \overline{k_1}, \ldots, \overline{k_m})$ (assumption)
2. $\overline{k} \leq \overline{l} \wedge \neg B(\overline{k}, \overline{k_1}, \ldots, \overline{k_m})$ (1, 2.5(i))
3. $\overline{k} \leq t(\overline{k_1}, \ldots, \overline{k_m}) \wedge \neg B(\overline{k}, \overline{k_1}, \ldots, \overline{k_m})$ (2, 2.4, equality)
4. $\exists v\, (v \leq t(\overline{k_1}, \ldots, \overline{k_m}) \wedge \neg B(v, \overline{k_1}, \ldots, \overline{k_m}))$ (3)

This essentially yields a BA-proof of $\neg A(\overline{k_1}, \ldots, \overline{k_m})$, q.e.d.

Now we complete the proof of Theorem 2.2. The argument goes by induction on the build-up of a Σ_1-formula $A(b_1, \ldots, b_m)$. If A is elementary, the result follows by Lemma 2.7 (i). If A has the form $\exists v A_0(v, b_1, \ldots, b_m)$ and $\mathbf{N} \vDash A(k_1, \ldots, k_m)$, then for some k we have $\mathbf{N} \vDash A_0(k, k_1, \ldots, k_m)$. By the induction hypothesis it follows that BA proves $A_0(\overline{k}, \overline{k_1}, \ldots, \overline{k_m})$. From this we can logically infer $\exists v A_0(v, \overline{k_1}, \ldots, \overline{k_m})$, q.e.d.

COROLLARY 2.8.

1. Any arithmetical theory containing BA is Σ_1-complete.
2. PA is Σ_1-complete.

2.21. Numerating r.e. sets. Let T be an arithmetical theory. A relation $R(x_1, \ldots, x_n)$ is said to be *numerated in* T, iff there exists a formula $A(x_1, \ldots, x_n)$ such that for all $k_1, \ldots, k_n \in \mathbf{N}$

$$R(k_1, \ldots, k_n) \iff T \vdash A(\overline{k_1}, \ldots, \overline{k_n}).$$

The formula A is then called a *numeration* of R. Obviously, if T is an r.e. theory, any predicate numerated in T is also r.e. On the other hand, the predicates numerated in TA coincide with those definable in the standard model and thus are not necessarily r.e.

If T is r.e., numerability provides a model of computation: in order to check whether $R(k_1, \ldots, k_n)$ holds, one can enumerate all the T-proofs and search for a proof of the corresponding numerical instance of A. Thus, A can be interpreted as a program for computing R.

LEMMA 2.9. For any r.e. predicate R, there is a Σ_1-formula A such that A numerates R in any Σ_1-sound theory T containing BA.

PROOF. Let A be a Σ_1-definintion of R in the standard model. By Σ_1-soundness, $T \vdash A(\overline{k_1}, \ldots, \overline{k_n})$ implies $\mathbf{N} \vDash A(\overline{k_1}, \ldots, \overline{k_n})$ and $R(k_1, \ldots, k_n)$. The converse follows by Σ_1-completeness of T (Corollary 2.8), q.e.d.

The following important corollary shows that numeration of r.e. predicates actually provides a universal model of computation for any sufficiently strong, Σ_1-sound theory T. In particular, as such a T one can take BA or PA.

THEOREM 2.10. For any Σ_1-sound r.e. theory T containing BA, the following statements are equivalent:

(i) R has a numeration in T;

(ii) R has a Σ_1-numeration in T;

(iii) R is Σ_1-definable (in the standard model);

(iv) R is r.e.

Notice that later we shall be able to replace the assumption of Σ_1-soundness here by the mere consistency of T (Ehrenfeucht-Feferman). The same applies to the following classical theorem.

THEOREM 2.11. The set of theorems of any Σ_1-sound theory T containing BA is undecidable.

PROOF. Take a Σ_1-numeration $K(a)$ of an undecidable r.e. set K. Observe that

$$n \in K \iff T \vdash K(\overline{n}).$$

Hence, if T were decidable, so would be the set K, q.e.d.

COROLLARY 2.12 (CHURCH). Pure first order logic (in the language of arithmetic) is undecidable.

PROOF. Let $\bigwedge BA$ denote the conjunction of the universal closures of all extralogical axioms of BA (including the finitely many equality axioms). Then, by Deduction theorem,

$$BA \vdash A \iff \vdash \bigwedge BA \to A,$$

for any formula A. Hence, if pure logic were decidable, so would be BA, contradicting Theorem 2.11, q.e.d.

The following statement improves a version of Gödel's First Incompleteness Theorem given in the previous section.

THEOREM 2.13. Any Σ_1-sound arithmetical r.e. theory T is (syntactically and semantically) incomplete.

PROOF. Without loss of generality we may assume that T contains BA (adding finitely many true Π_1-axioms of BA to T preserves its Σ_1-soundness). Let $K(a)$ be a Σ_1-numeration of an undecidable r.e. set in T. If T were complete, $\neg K(a)$ would numerate the complement of K:

$$n \notin K \iff T \nvdash K(\overline{n}) \iff T \vdash \neg K(\overline{n}),$$

which is impossible, because $\mathbf{N} \setminus K$ is not r.e., q.e.d.

2.22. Representing computable functions. The graph of any computable function, being r.e., can be numerated in any Σ_1-sound theory containing BA. For technical purposes a different notion of *representability* of functions is traditionally considered. It will lead us to another natural model of computation associated with r.e. theories.

Let f be an n-ary (partial) recursive function. We say that a formula $F(x_1, \dots, x_n, y)$ *represents* f in a theory T, iff for all $k_1, \dots, k_n \in dom(f)$,

$$T \vdash F(\overline{k_1}, \dots, \overline{k_n}, y) \leftrightarrow y = \overline{l}, \tag{2.1}$$

where $l = f(k_1, \dots, k_n)$. Obviously, if F represents f in T, then the same holds in any extension of T.

LEMMA 2.14. Let T be a consistent theory containing BA, and f be a total function. If a formula F represents f in T, then F numerates the graph of f in T.

PROOF. Let $l = f(k_1, \dots, k_n)$, then

$$T \vdash y = \overline{l} \to F(\overline{k_1}, \dots, \overline{k_n}, y)$$

implies $T \vdash F(\overline{k_1}, \dots, \overline{k_n}, \overline{l})$ by the equality axioms. On the other hand, assuming $T \vdash F(\overline{k_1}, \dots, \overline{k_n}, \overline{m})$, from (2.1) we obtain $T \vdash \overline{m} = \overline{l}$. If $m \neq l$, then by Lemma 2.5 (ii) we would have $BA \vdash \neg\, \overline{m} = \overline{l}$. Hence, T would be inconsistent, contrary to our assumption, q.e.d.

LEMMA 2.15. For any partial recursive function f, there is a Σ_1-formula representing f in (any extension of) BA.

PROOF. For the sake of readability we only give a proof for unary functions f. Pick a Σ_1-formula $F(x, y)$ defining the graph of f in the standard model. We may assume that $F(x, y)$ has the form

$$\exists z_1 \dots \exists z_n\, F_0(x, y, z_1, \dots, z_n),$$

where F_0 is bounded. We define

$$\begin{aligned} F_1(x, y, z) &:\leftrightarrow y \leq z \wedge \exists z_1, \dots, z_n \leq z\, F_0(x, y, z_1, \dots, z_n) \\ F_2(x, y, z) &:\leftrightarrow F_1(x, y, z) \wedge \forall y', z' \leq z\, (F_1(x, y', z') \to y' = y) \end{aligned}$$

We show that the formula $\exists z\, F_2(x, y, z)$ represents f.

First of all, if $f(m) = n$, then for a sufficiently large k, $F_1(m, n, k)$ is true, whence, by functionality of f, $F_2(m, n, k)$ also holds. By Σ_1-completeness, the formula $\exists z\, F_2(\overline{m}, \overline{n}, z)$ is provable in BA.

For the opposite direction we need an auxiliary lemma.

LEMMA 2.16. For any $m, n \in \mathbf{N}$ such that $m \leq n$,

(i) $BA \vdash a \leq \overline{m} \to a \leq \overline{n}$,

(ii) $BA \vdash \overline{n} \le a \to \overline{m} \le a$.

PROOF. (i) Follows from Lemma 2.6.

(ii) W.l.o.g. we may assume $m < n$. By Lemma 2.5 (iii), $\overline{n} \le a$ implies $\neg a = \overline{i}$, for each $i < n$. Therefore, by 2.6, we have $\neg a \le \overline{m}$. Axiom P11 then yields $\overline{m} \le a$, q.e.d.

Now we complete the proof of Lemma 2.15 by showing

$$BA \vdash F_2(\overline{m}, y, z) \to y = \overline{n},$$

where $n = f(m)$. Assume

$$F_1(\overline{m}, y, z) \wedge \forall y', z' \le z\, (F_1(\overline{m}, y', z') \to y' = y), \tag{2.2}$$

and let n, k satisfy

$$F_1(\overline{m}, \overline{n}, \overline{k}). \tag{2.3}$$

Notice that $n \le k$, by the definition of F_1, and by Σ_1-completeness (2.3) is provable. By P11 two cases are possible: $\overline{k} \le z$ or $z \le \overline{k}$.

Suppose $\overline{k} \le z$. Then $\overline{n} \le z$ by 2.16 (ii), and by (2.3) $\exists z' \le z\, F_1(\overline{m}, \overline{n}, z')$. By (2.2) this implies $\overline{n} = y$.

If $z \le \overline{k}$, we notice that for any i, j such that $i \ne n = f(m)$, $\mathbf{N} \nvDash F_1(m, i, j)$. Hence,

$$\forall y', z' \le \overline{k}\, (F_1(\overline{m}, y', z') \to y' = \overline{n}) \tag{2.4}$$

is true and provable in BA. By (2.2) we have $F_1(\overline{m}, y, z)$, therefore $y \le z$. Since $z \le \overline{k}$, by Lemma 2.6 we obtain $\bigvee_{i=0}^{k} z = \overline{i}$, and thus $\bigvee_{i=0}^{k} y \le \overline{i}$. Lemma 2.16 (i) then yields $y \le \overline{k}$, so we can infer $y = \overline{n}$ from $F_1(\overline{m}, y, z)$ and (2.4), q.e.d.

2.23. Dually numerating pairs of disjoint r.e. predicates. Numeration $A(x_1, \dots, x_n)$ of an r.e. predicate $R(x_1, \dots, x_n)$ can be considered as a program for computing the function

$$\nu_A(x_1, \dots, x_n) := \begin{cases} 1, & \text{if } T \vdash A(\overline{x}_1, \dots, \overline{x}_n) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

In a sense, this model of computation is weak, because it neglects all negative information that may appear in the process of computation. For example, if one encounters a proof of $\neg A(\overline{k})$ (and a priori knows that T is consistent), one can conclude that k can never belong to R and stop the process of computation at that point. Thus, with the same formula A one can associate a more informative function:

$$\delta_A(x_1, \dots, x_n) := \begin{cases} 1, & \text{if } T \vdash A(\overline{x}_1, \dots, \overline{x}_n) \\ 0, & \text{if } T \vdash \neg A(\overline{x}_1, \dots, \overline{x}_n) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Partial computable 0–1-valued functions are identified with pairs of disjoint r.e. predicates (see Section **??**). This leads us to the following important definition.

Let T be a consistent theory. A formula $A(x_1, \dots, x_n)$ is called a *dual numeration* of a pair of disjoint r.e. predicates $R(x_1, \dots, x_n)$ and $S(x_1, \dots, x_n)$ in T, iff A numerates R and $\neg A$ numerates S in T. Equivalently, A dually numerates the pair (R, S), iff the function $\delta_A \simeq \chi_{R,S}$, where

$$\chi_{R,S}(x_1, \dots, x_n) := \begin{cases} 1, & \text{if } R(x_1, \dots, x_n) \\ 0, & \text{if } S(x_1, \dots, x_n) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

We say that a formula A *separates* a pair of disjoint r.e. predicates R and S in T, iff for all $k_1, \dots, k_n \in \mathbf{N}$,

(i) $R(k_1, \dots, k_n) \implies T \vdash A(\overline{k}_1, \dots, \overline{k}_n)$;

(ii) $S(k_1, \dots, k_n) \implies T \vdash \neg A(\overline{k}_1, \dots, \overline{k}_n)$.

Obviously, A separates R and S, iff the function δ_A extends $\chi_{R,S}$.

In the next section we shall prove that, for any sufficiently strong consistent theory T, any pair of disjoint r.e. predicates can be dually numerated in T (Putnam-Smullyan). Here we shall only prove a weaker fact.

THEOREM 2.17. Any pair of disjoint r.e. predicates can be separated in (any extension of) BA.

PROOF. Consider any pair of disjoint r.e. predicates (R, S), and let $F(x_1, \dots, x_n, y)$ represent the partial recursive function $\chi_{R,S}(x_1, \dots, x_n)$ in BA. We claim that the formula $F(x_1, \dots, x_n, 1)$ separates R and S.

Indeed, by the definition of F we have

$$BA \vdash F(\overline{k}_1, \dots, \overline{k}_n, y) \leftrightarrow y = 1, \qquad \text{if } R(k_1, \dots, k_n); \tag{2.5}$$

$$BA \vdash F(\overline{k}_1, \dots, \overline{k}_n, y) \leftrightarrow y = 0, \qquad \text{if } S(k_1, \dots, k_n). \tag{2.6}$$

Therefore, if $R(k_1, \dots, k_n)$, then $BA \vdash F(\overline{k}_1, \dots, \overline{k}_n, 1)$ by (2.5). If $S(k_1, \dots, k_n)$, then $F(\overline{k}_1, \dots, \overline{k}_n, 1)$ implies $0 = 1$ and a contradiction in BA by (2.6), q.e.d.

REMARK 2.18. The formula separating the predicates R and S is Σ_1 and works for any extension of BA.

A formula $A(x_1, \dots, x_n)$ *binumerates* a relation $R(x_1, \dots, x_n)$ in T, iff A separates R and the complement of R in T. Notice that, if the theory T is consistent, A binumerates R just in case A dually numerates the pair $(R, \mathbf{N}^n \backslash R)$.

By Post's theorem, any predicate binumerated in a consistent r.e. theory is decidable. From Theorem 2.17 we immediately obtain the following corollary.

COROLLARY 2.19. Any recursive predicate is binumerated by a suitable Σ_1-formula in (any extension of) BA.

2.24. Undecidability and Gödel-Rosser incompleteness theorems. As a straightforward application of the ideas and techniques of the previous subsection we now obtain strong undecidability and incompleteness results.

THEOREM 2.20. The set of theorems of any consistent arithmetical theory T containing BA is undecidable. Moreover, the sets of provable and refutable sentences of T are recursively inseparable.

PROOF. It will be sufficient to prove the second claim of the theorem. Let P and R denote the sets of (Gödel numbers of) provable and refutable sentences of T, respectively.

Consider a pair of recursively inseparable, disjoint r.e. sets (X, Y), and let a formula A separate this pair in T. (Such a formula exists by Theorem 2.17.) Now assume, for a contradiction, that P and R are separated by a recursive set C. Then the set $\{m \mid A(\overline{m}) \in C\}$ is recursive and separates X and Y:

$$\begin{array}{lllllll} m \in X & \Rightarrow & T \vdash A(\overline{m}) & \Rightarrow & A(\overline{m}) \in P & \Rightarrow & A(\overline{m}) \in C, \\ m \in Y & \Rightarrow & T \vdash \neg A(\overline{m}) & \Rightarrow & A(\overline{m}) \in R & \Rightarrow & A(\overline{m}) \notin C, \end{array}$$

a contradiction.

Notice that the second claim of the theorem implies the first, because for any consistent decidable extension T' of T, the (recursive) set of theorems of T' would separate P from R, q.e.d.

THEOREM 2.21 (GÖDEL-ROSSER). Let T be a consistent, r.e., arithmetical theory containing BA. Then T is (syntactically and semantically) incomplete.

PROOF. By Post's theorem, the set of theorems of any syntactically complete r.e. theory T is decidable, which contradicts the (first claim of the) previous theorem, q.e.d.

3. Self-reference

Peano arithmetic was devised as a formal system sufficiently rich to encompass all finitary mathematics. As the formal proofs are finite objects, in particular, this means that PA has to be able to "reason about" itself. For example, it makes sense to ask whether PA proves its own consistency, for the consistency assertion is nothing but a statement about the existence of a particular finite string of symbols with a rather simple syntactical description, and hence, PA must "understand" what its own consistency means.

It was Gödel who first put the idea of self-reference to a fruitful mathematical use, thus obtaining his celebrated Incompleteness Theorems. Since then, the method was developed and brought to life a great number of results in logic and recursion theory. Self-reference is also one of the main technical tools used in this book. It is interesting that, despite the ever growing range of applications of self-reference in arithmetic, the key "fixed point lemma" remains essentially unchanged in its form since Gödel's time. A proof of this lemma and some

of its prominent applications, including Tarski undefinability theorem and the original proofs of Gödel and Rosser incompleteness theorems, is the goal of this Section. We compare different proofs of incompleteness theorems from the point of view of their effectiveness and recursion-theoretic content. Finally, using a self-referential construction due to Sheperdson, we establish the important, but less well-known, theorems of Ehrenfeucht-Feferman and Putnam-Smullyan on the universality of the computation models presented in the previous section.

3.25. Fixed point lemma. Consider a first order language $\mathcal{L}_\Sigma$ of a finite signature Σ. Recall that various syntactic objects of $\mathcal{L}_\Sigma$ such as variables, terms, formulas, proofs, ... can be identified with certain expressions (words) in the alphabet of predicate and function symbols of Σ together with the following special symbols:

$$\rightarrow \ \neg \ (\ ) \ , \ \forall \ a \ v \ \sharp$$

(see ??). Words, in turn, can be effectively encoded by numbers, e.g., via the binary coding described in Section ??. We shall explicitly develop a suitable coding in the next section, but for now it will only be essential for us that the correspondence between words and their numerical codes (*Gödel numbers*) is one-to-one and both ways computable. As usual, the Gödel number of a word s will be denoted $\ulcorner s \urcorner$.

LEMMA 3.1. Assume $\mathcal{L}$ contains the language of PA. Then BA represents the following "substitution function:"

$$sub_a(x,y) := \begin{cases} \ulcorner A(a/\overline{y}) \urcorner, & \text{if } x = \ulcorner A(a) \urcorner,\ A(a) \text{ is a formula,} \\ & a \text{ is a fixed free variable;} \\ 0, & \text{otherwise.} \end{cases}$$

PROOF. By Church-Turing thesis, since the Gödel numbering is effective, the function sub_a is computable. Indeed, having x we can effectively recover the formula $A(a)$ such that $\ulcorner A(a) \urcorner = x$ (if such exists), effectively find all occurences of a in A, replace them by $\overline{y}$, and calculate the Gödel number of the resulting expression. Thus, representability of sub_a in BA follows from Lemma 2.15, q.e.d.

For the sake of readability we denote the formula representing sub_a in BA by $sub_a(x,y) = z$.

LEMMA 3.2 (FIXED POINT LEMMA). For every arithmetical formula $A(a)$, there is a formula F such that

$$BA \vdash F \leftrightarrow A(\overline{\ulcorner F \urcorner}).$$

PROOF. Consider the formula

$$C(a) :\leftrightarrow \exists y \, (sub_a(a,a) = y \wedge A(y)).$$

Let $m = \ulcorner C \urcorner$, then obviously $sub_a(m,m) = \ulcorner C(a/\overline{m}) \urcorner$. We define

$$F :\leftrightarrow C(\overline{m}),$$

and so $sub_a(m,m) = \ulcorner F \urcorner$. Since sub_a is represented in BA, we obtain

$$BA \vdash \forall y\, (sub_a(\overline{m},\overline{m}) = y \leftrightarrow y = \overline{\ulcorner F \urcorner}).$$

It follows that F, that is, the formula $\exists y\, (sub_a(\overline{m},\overline{m}) = y \wedge A(y))$, is provably equivalent to $\exists y\, (y = \overline{\ulcorner F \urcorner} \wedge A(y))$ and $A(\overline{\ulcorner F \urcorner})$, q.e.d.

Lemma 3.2 can also be generalized to arbitrary (finite) languages $\mathcal{L}$ containing that of arithmetic. In this situation the theory formulated in $\mathcal{L}$, whose only nonlogical axioms are those of BA, plays the role of BA.

The formula F is usually called the *fixed point* of A. Notice that the proof of Lemma 3.2 not only shows the existence of a fixed point for every formula $A(a)$, but also provides an explicit example of such a fixed point. Fixed points of a formula need not, in general, be unique, either graphically, or up to provable equivalence.

EXAMPLE 3.1. Let $A(a)$ be the formula $a = 0$, and suppose the a priori Gödel numbering is such that $\ulcorner 0 = 0 \urcorner$ equals 0. Then, trivially, both formulas $0 = 0$ and $0 = 1$ are the fixed points of A.

Fixed point lemma can be generalized to formulas containing additional parameters. In order to formulate it we first introduce some useful notation. The multiple substitution function is defined as follows:

$$sub_{b_1,\ldots,b_n}(x, y_1, \ldots, y_n) \quad := \quad sub_{b_1}(\cdots(sub_{b_n}(x, y_n), y_{n-1}), \cdots, y_1).$$

Clearly, this function is total recursive and thus can be represented in BA. Let $B(a)$ and $A(b_1, \ldots, b_n)$ be any formulas. Then the formula

$$\exists y\, (sub_{b_1,\ldots,b_n}(\overline{\ulcorner A \urcorner}, x_1, \ldots, x_n) = y \wedge B(y))$$

is usually abbreviated by $B(\ulcorner A(\dot{x}_1, \ldots, \dot{x}_n) \urcorner)$. Properly speaking, the expression $\ulcorner A(\dot{x}_1, \ldots, \dot{x}_n) \urcorner$ is not an arithmetical term, nor can such a term be introduced in a definitional extension of such a weak theory as BA.[1] Yet, it is often convenient to think of it as of a term denoting the function that computes the Gödel number of $A(\overline{k}_1, \ldots, \overline{k}_n)$ from the numbers $k_1, \ldots, k_n$. Notice that the variables $x_1, \ldots, x_n$ are free in $B(\ulcorner A(\dot{x}_1, \ldots, \dot{x}_n) \urcorner)$.

LEMMA 3.3 (PARAMETRIC FIXED POINT LEMMA).

(i) For every $A(a, b_1, \ldots, b_n)$, there is a formula $F(b_1, \ldots, b_n)$ such that

$$BA \vdash F(b_1, \ldots, b_n) \leftrightarrow A(\overline{\ulcorner F \urcorner}, b_1, \ldots, b_n).$$

(ii) For every $A(a, b_1, \ldots, b_n)$, there is a formula $F(b_1, \ldots, b_n)$ such that

$$BA \vdash F(b_1, \ldots, b_n) \leftrightarrow A(\ulcorner F(\dot{b}_1, \ldots, \dot{b}_n) \urcorner, b_1, \ldots, b_n).$$

[1] Although this can be done in PA, cf. Section ??.

PROOF. (i) The proof of Lemma 3.2 remains unchanged, with the understanding that now the formula A may implicitly contain the free variables $b_1, \ldots, b_n$.

(ii) This follows from (i) applied to the formula

$$\exists y\, (sub_{b_1,\ldots,b_n}(a, b_1, \ldots, b_n) = y \wedge A(y, b_1, \ldots, b_n)),$$

q.e.d.

The following *multiple fixed point* lemma is a corollary of Lemma 3.3 (i).

LEMMA 3.4. Suppose, for $i = 1, \ldots, n$, we are given arithmetical formulas $A_i(a_1, \ldots, a_n)$. Then there are formulas $F_1, \ldots, F_n$ such that, for all $i = 1, \ldots, n$,

$$BA \vdash F_i \leftrightarrow A_i(\overline{\ulcorner F_1 \urcorner}, \ldots, \overline{\ulcorner F_n \urcorner}).$$

(As before, the formulas A_i, and hence F_i, may contain additional parameters, which we do not explicitly indicate.)

PROOF. Consider the formula

$$A(a_1, \ldots, a_n, b) :\leftrightarrow \bigvee_{i=1}^{n} (A_i(a_1, \ldots, a_n) \wedge b = \overline{i}).$$

We obviously have

$$BA \vdash A(a_1, \ldots, a_n, \overline{i}) \leftrightarrow A_i(a_1, \ldots, a_n),$$

for each $1 \leq i \leq n$. Now, using lemma 3.2, let $F(b)$ satisfy the fixed point equation

$$BA \vdash F(b) \leftrightarrow \exists y_1 \ldots \exists y_n\, (A(y_1, \ldots, y_n, b) \wedge \bigwedge_{i=1}^{n} sub_b(\overline{\ulcorner F \urcorner}, \overline{i}) = y_i).$$

For each $1 \leq i \leq n$ we have $\ulcorner F(\overline{i}) \urcorner = sub_b(\ulcorner F \urcorner, i)$, and this fact is verifiable in BA, that is,

$$BA \vdash \forall y\, (sub_b(\overline{\ulcorner F \urcorner}, \overline{i}) = y \leftrightarrow y = \overline{\ulcorner F(\overline{i}) \urcorner}).$$

It follows that, for any $1 \leq k \leq n$, $F(\overline{k})$ is provably equivalent to

$$\exists y_1 \ldots \exists y_n\, (A(y_1, \ldots, y_n, \overline{k}) \wedge \bigwedge_{i=1}^{n} y_i = \overline{\ulcorner F(\overline{i}) \urcorner}),$$

and hence to

$$A_k(\overline{\ulcorner F(1) \urcorner}, \ldots, \overline{\ulcorner F(\overline{n}) \urcorner}).$$

This means that one can take the formulas $F(\overline{k})$ as F_k, q.e.d.

3.26. Tarski, Gödel and Rosser theorems. As immediate applications of the method of self-reference we derive Tarski undefinability theorem and give alternative (or rather, original) proofs of Gödel and Rosser incompleteness theorems. The following theorem shows that the set of (Gödel numbers of) true arithmetical sentences not only is not r.e., but also cannot be defined by any arithmetical formula.

THEOREM 3.5 (TARSKI). There is no arithmetical formula $T(a)$ such that, for every arithmetical sentence A,

$$\mathbf{N} \vDash A \leftrightarrow T(\overline{\ulcorner A \urcorner}).$$

PROOF. Suppose, on the contrary, that there is such a formula. Let A be a solution of the fixed point equation

$$BA \vdash A \leftrightarrow \neg T(\overline{\ulcorner A \urcorner}).$$

(Observe the similarity with the Liar paradox. The sentence A asserts: "I am false.") Then, by our assumption on $T(a)$, $\mathbf{N} \vDash A \leftrightarrow \neg A$, which is a contradiction, q.e.d.

The proof of Tarski theorem applies the fixed point lemma to a formula only to show that the formula does not exist. In contrast, Gödel and Rosser applied the same kind of argument to physically existing formulas thus obtaining celebrated incompleteness results. The following statement can be called *an abstract version of Gödel and Rosser theorems*.

THEOREM 3.6. Let T be a consistent arithmetical theory, $P(a)$ a formula, and A a fixed point of $\neg P(a)$ in T:

$$T \vdash A \leftrightarrow \neg P(\overline{\ulcorner A \urcorner}).$$

Then

(i) (GÖDEL) If $P(a)$ numerates the set of (Gödel numbers of) theorems of T in T, then $T \nvdash A$ and $T \nvdash \neg A$.

(ii) (ROSSER) If $P(a)$ separates the sets of (Gödel numbers of) provable and refutable sentences of T in T, then $T \nvdash A$ and $T \nvdash \neg A$.

PROOF. (i) Obviously, if $P(a)$ numerates the set of theorems of T in T, then

$$T \vdash A \Leftrightarrow T \vdash P(\overline{\ulcorner A \urcorner}) \Leftrightarrow T \vdash \neg A,$$

which contradicts the consistency of T.

(ii) If $P(a)$ separates provable and refutable sentences of T in T, then

$$\begin{array}{lcccl} T \vdash A & \Rightarrow & T \vdash P(\overline{\ulcorner A \urcorner}) & \Rightarrow & T \vdash \neg A, \\ T \vdash \neg A & \Rightarrow & T \vdash \neg P(\overline{\ulcorner A \urcorner}) & \Rightarrow & T \vdash A, \end{array}$$

and again, T is inconsistent in each case, q.e.d.

Notice that the statement of Theorem 3.6 is very general: it does not even require that T is sufficiently strong and r.e. (Of course, one would need some strength of T in order to ensure the existence of the fixed point A, which was simply postulated above. However, the restriction that T is r.e. in various situations can, indeed, be considerably weakened.)

We call a formula $P(a)$ numerating the set of theorems of T in T a *provability predicate for T in T*. The words "in T" are necessary, because, even if $P(a)$ expresses provability in T within T itself, it may fail to do so in the standard model, if T is not sound. Recall that, so far, we were only able to construct (Σ_1) numerations of r.e. sets in Σ_1-sound extensions of BA. In this case Theorem 3.6 (i) takes the form which is very close to Gödel's original statement of his First Incompleteness Theorem (see Notes below).

THEOREM 3.7 (GÖDEL). Let T be an r.e. arithmetical theory containing BA, $P(a)$ a Σ_1 provability predicate for T in BA, and

$$T \vdash A \leftrightarrow \neg P(\overline{\ulcorner A \urcorner}).$$

If T is consistent, then $T \nvdash A$. If T is Σ_1-sound, then also $T \nvdash \neg A$.

PROOF. The first part of the statement holds because

$$T \vdash A \;\Rightarrow\; BA \vdash P(\overline{\ulcorner A \urcorner}) \;\Rightarrow\; T \vdash \neg A.$$

The second part holds because, if T is Σ_1-sound, the Σ_1-formula $P(a)$ numerates the set of theorems of T in T, q.e.d.

A formula $P(a)$ separating provable and refutable sentences of T in T is called *Rosser-type* provability predicate for T in T. Theorem 2.17 shows that a Σ_1 Rosser-type provability predicate (in any extension of BA) does exist for every consistent r.e. arithmetical theory T containing BA, without any additional assumptions on soundness. Also notice that, if T is a Σ_1-sound theory, and $P(a)$ a Σ_1 Rosser-type provability predicate for T, then $P(a)$ numerates the theorems of T in T, and thus is, indeed, a provability predicate. Later we shall see that the converse, in general, does not hold and, in fact, the canonical "Gödel's" provability predicate is not Rosser-type.

The original version of Rosser theorem dealt with a particular Rosser-type provability predicate that we call *Rosser's*. Let T be a consistent r.e. arithmetical theory and Σ_1-formulas $P(a)$ and $R(a)$ define the sets of Gödel numbers of provable and refutable formulas of T, respectively. Without loss of generalty we may assume that $P(a)$ has the form $\exists y P_0(a, y)$, where P_0 is an elementary formula, and, similarly, $R(a)$ has the form $\exists y R_0(a, y)$. (Under a natural elementary Gödel numbering, e.g. the one described in the next section, and under the assumption that the set of axioms of T is elementary, the formula $P_0(a, b)$ can be chosen to express the predicate "b is the Gödel number of a proof of

the formula a in T," and $R_0(a,b)$ as "b is the Gödel number of a proof of the negation of a in T.")

Rosser's provability predicate $P^R(a)$ is then defined as follows:

$$P^R(a) \quad :\leftrightarrow \quad \exists y\,(P_0(a,y) \wedge \forall x \leq y\, \neg R_0(a,x)).$$

Informally, $P^R(a)$ says that there is a T-proof of a formula A, with $\ulcorner A \urcorner = a$, such that there is no T-proof of $\neg A$ having a smaller Gödel number. Clearly, $P^R(a)$ is a Σ_1-formula.

LEMMA 3.8. *$P^R(a)$ separates the sets of provable and refutable sentences of T in (any extension of) BA.*

PROOF. The proof is very similar to that of Lemma 2.15 and, in fact, easier. To show that BA proves $P^R(\overline{\ulcorner A \urcorner})$ for every provable sentence A, we notice that, since T is consistent,

$$BA \vdash \exists y\,(P_0(\overline{\ulcorner A \urcorner}, y) \wedge \forall x \leq y\, \neg R_0(\overline{\ulcorner A \urcorner}, x)),$$

by Σ_1-completeness.

To show that BA proves $\neg P^R(\overline{\ulcorner A \urcorner})$ for every refutable sentence A we reason as follows.

Since $T \vdash \neg A$, for some n the Σ_1-formula $R_0(\overline{\ulcorner A \urcorner}, \overline{n})$ is true and provable in BA. Reasoning within BA assume $P_0(\overline{\ulcorner A \urcorner}, y)$. There are two cases: $y \leq \overline{n}$ or $\overline{n} \leq y$. In the first case, since T is consistent and $R_0(\overline{\ulcorner A \urcorner}, \overline{n})$ holds, the formula $\forall z \leq \overline{n}\, \neg P_0(\overline{\ulcorner A \urcorner}, z)$ is true and provable in BA by Σ_1-completeness. But this contradicts the assumption $P_0(\overline{\ulcorner A \urcorner}, y)$.

In the second case we obtain $\exists x \leq y\, R_0(\overline{\ulcorner A \urcorner}, x)$ taking $\overline{n}$ for x. So, we have shown that BA proves

$$\forall y\,(P_0(\overline{\ulcorner A \urcorner}, y) \rightarrow \exists x \leq y\, R_0(\overline{\ulcorner A \urcorner}, x)),$$

which implies $\neg P^R(\overline{\ulcorner A \urcorner})$, q.e.d.

THEOREM 3.9 (ROSSER). *Let T be a consistent r.e. arithmetical theory containing BA, and a sentence A satisfy*

$$T \vdash A \leftrightarrow \neg P^R(\overline{\ulcorner A \urcorner}).$$

Then $T \nvdash A$ and $T \nvdash \neg A$.

3.27. Comparing incompleteness proofs. We round up the discussion of Gödel's First Incompleteness Theorem by comparing various proofs of incompleteness for the simplest and the most important case of PA. We have five proofs by now: 1.18, 2.13, 2.21, 3.7, and 3.9. They all use increasingly more and more involved techniques, and apply to increasingly wider classes of theories. Yet, are all of them really different in the situation where the simplest idea — simulating Turing machines in arithmetic — already works?

An important feature of the incompleteness proofs of Theorems 3.7 and 3.9 is their constructive character, that is, they allow to construct independent sentences for a theory T explicitly and effectively.

Recall that a Σ_1-formula numerating an r.e. set in BA is just the definition of this set in the standard model (explicitly constructed in Theorem 1.12, given an appropriate Turing machine program). Similarly, a formula separating a given pair of disjoint r.e. sets is constructed in a more tricky way, but nonetheless explicitly, in Theorem 2.17. Alternatively, one can take the above Rosser's provability predicate as a formula separating provable and refutable sentences of a given (Σ_1-sound, sufficiently strong, r.e.) theory T. Furthermore, the proof of the fixed point lemma also provides an explicit construction of a fixed point. Thus, Theorems 3.7 and 3.9 show two different ways to exhibit independent sentences for T. In fact, these sentences can be effectively constructed from a Turing machine enumerating the set of axioms of T. These sentences will be fairly long, if one indeed proceeds via Theorem 1.12 — mainly because the Turing machine enumerating T has a long program. Yet, for example, we can estimate their arithmetical complexity without actually writing them out. (In the next section, for a more delicate analysis, we will have to write them out, but we shall do it by directly encoding the syntax of arithmetic in itself, thus economizing our programming efforts.)

COROLLARY 3.10. Under the assumptions of Gödel-Rosser theorem 2.21, there is an independent sentence for T of complexity Π_1. (Its negation is also independent and Σ_1.)

PROOF. Let

$$T \vdash A \leftrightarrow \neg P(\overline{\ulcorner A \urcorner}),$$

where P is a Σ_1 Rosser-type provability predicate for T. We know that (under the relevant assumptions) A is independent from T. Hence, so is the equivalent Π_1-sentence $\neg P(\overline{\ulcorner A \urcorner})$, and its Σ_1 negation $P(\overline{\ulcorner A \urcorner})$, q.e.d.

Let us show that the recursion-theoretic proofs 1.18, 2.13, and 2.21, although non-constructive in their present form, can be modified to produce examples of independent sentences. Recall that the r.e. undecidable set K is creative, that is, for any r.e. set W_m such that $K \cap W_m = \emptyset$, one can (effectively in m) find a point x such that $x \notin K \cup W_m$.

Let $K(a)$ be a Σ_1-formula numerating K in PA (or in the standard model). As the proofs of 1.18, 2.13 go, under the assumption of completeness of PA, $\neg K(a)$ numerates in PA the complement of K, which contradicts undecidability of K by Post's theorem. But in reality — we know that PA is actually incomplete — $\neg K(a)$ only numerates a certain r.e. subset K' of the complement of K. An index of this subset can be found effectively from the Gödel number of $\neg K(a)$ and the index of an enumeration of the set of theorems of PA. Using creativeness of K we obtain a number m such that $m \notin K \cup K'$, which implies that neither $K(\overline{m})$, nor $\neg K(\overline{m})$ can be provable. So, creativeness of K yields a constructive version of the first two proofs of Gödel's theorem. Notice that

the example of an independent sentence thus obtained also has arithmetical complexity Π_1.

Now let us examine the recursion-theoretic content of the proof of 3.7, which is based on the fixed point lemma and numerability of r.e. sets in PA. Recall that provability in PA provides a universal model of computation: r.e. sets X can be indexed by Gödel numbers of formulas $A(a)$ such that, for all $n \in \mathbf{N}$,

$$n \in X \iff PA \vdash A(\overline{n}). \tag{2.7}$$

For the provability model of computation one can, of course, reformulate all the results valid for the Turing machine model. Recall that the creative set K was defined as the diagonal set $\{x \in \mathbf{N} \mid x \in W_x\}$. By (2.7), the role of W_x now plays the set

$$\{n \in \mathbf{N} \mid PA \vdash A(\overline{n})\},$$

where $\ulcorner A \urcorner = x$, or equivalently,

$$\{n \in \mathbf{N} \mid \mathbf{N} \vDash P(sub_a(x, n))\},$$

where $P(a)$ defines the set of theorems of PA in the standard model. The analog of K is then defined (numerated) by the formula $P(sub_a(a, a))$.[2]

Repeating the constructive version of the proof of 2.13 above, we consider the set numerated by $\neg P(sub_a(a, a))$ and calculate its index (Gödel number) m. According to that proof, the sentence $\neg P(sub_a(\overline{m}, \overline{m}))$ has to be independent. No doubt, the attentive reader has already noticed that we have just written out the solution of the fixed point equation

$$\mathbf{N} \vDash A \leftrightarrow \neg P(\ulcorner A \urcorner)$$

given by the proof of the fixed point lemma. Thus, 3.7 boils down to essentially the same proof as (the constructive versions of) 1.18 and 2.13; the only difference is in the chosen model of computation.

For the analysis of Rosser's theorem we use an effective version of the notion of recursively inseparable pair of r.e. sets. Disjoint pairs of r.e. sets can be identified with computable partial 0–1-valued functions (see Section ??). The standard example of a recursively inseparable pair of r.e. sets is just the function $\Phi(x, x)$ that cannot be extended to a total recursive one. It has the stronger property that, for any computable 0–1-valued function $f(x)$ extending $\Phi(x, x)$, one can effectively (in the index of f) find an element m such that $f(m)\uparrow$. A pair of disjoint r.e. sets (R, S) or the corresponding characteristic function $\chi_{R,S}$ having this property is called *effectively inseparable*.

The dual numerability model of computation indexes $\chi_{R,S}$ by the Gödel number of a formula A such that A numerates R, and $\neg A$ numerates S in PA. The diagonalized universal function $\Phi(x, x)$ is then indexed by the Gödel number of the formula $Q(sub_a(a, a))$, where $Q(a)$ dually numerates provable and

[2] In the discussion below we sloppily write $P(sub_a(a, a))$ instead of a more complex formula $\exists y\, (P(y) \wedge sub_a(a, a) = y)$.

refutable sentences in PA (we will see in the next subsection that such a formula exists, which also shows that our representation of 0–1-valued computable functions is adequate). If R and S are effectively inseparable, from any formula $B(a)$ such that δ_B extends $\chi_{R,S}$, that is, if B separates R and S, we can effectively find a number m such that $\delta_B(m)\uparrow$. This means that for the case of $\Phi(x,x)$ neither $B(\overline{m})$, nor $\neg B(\overline{m})$ is provable in PA, and yields an effective version of Gödel-Rosser theorem 2.21.

In order to compare it with the classical proof of Rosser's theorem 3.9, we notice that by the construction of $\Phi(x,x)$ (see Section ??) the number m that effects the inseparability of $\Phi(x,x)$ is chosen as an index of the function

$$g(x) := \begin{cases} 0, & \text{if } \delta_B(x) = 1, \\ 1, & \text{if } \delta_B(x) = 0, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

This function coincides with $\delta_{\neg B}$. Now, if $P(a)$ separates the sets of provable and refutable sentences of PA, the function corresponding to the formula $P(sub_a(a,a))$ extends the one for $Q(sub_a(a,a))$. Hence, for $B(a)$ one takes $P(sub_a(a,a))$, $m := \ulcorner \neg P(sub_a(a,a)) \urcorner$, and the independent sentence $\neg P(sub_a(\overline{m},\overline{m}))$ coincides with the standard solution of the fixed point equation

$$\mathbf{N} \vDash A \leftrightarrow \neg P(\ulcorner A \urcorner).$$

This means that the (effective version of the) second proof of 2.21 is essentially the same as the proof of Rosser's theorem 3.9 modulo the choice of the computation model.

Thus, we have encountered essentially only two different incompleteness proofs — roughly, Gödel's and Rosser's. In the next section we shall see that the canonical provability predicate for PA is not PA-provably equivalent to any Rosser-type provability predicate. Moreover, the diagonal sentences produced by Gödel's and Rosser's proofs are inequivalent. All this can be interpreted as an evidence of the fact that the two proofs are essentially different.

3.28. Ehrenfeucht-Feferman and Putnam-Smullyan theorems. In the previous chapter we have dealt with the two important models of computation related to provability in formal theories: numerability of r.e. sets and dual numerability of pairs of disjoint r.e. sets. Here we are concerned with the question under what conditions these models are universal.

Let T be an r.e. theory containing BA. Theorem 2.10 shows that, if T is Σ_1-sound, a set is numerated in T iff it is r.e. Ehrenfeucht and Feferman showed that the same is true under the mere assumption of the consistency of T. Putnam and Smullyan strengthened this by showing that any pair of disjoint r.e. sets is dually numerated in T, provided T is consistent. This means that under the minimal assumtions on T dual numerability in T also provides a universal model of computation. We present a proof of these results due to Shepherdson.

First, we introduce some useful notation. Let $\exists x A_0(x)$ and $\exists x B_0(x)$ be any two Σ_1-formulas (possibly containing parameters), with A_0 and B_0 elementary. We write $\exists x A_0(x) \prec \exists x B_0(x)$ to abbreviate the formula

$$\exists x\, (A_0(x) \wedge \forall y \leq x\, \neg B_0(y)).$$

Notice that, within BA, any Σ_1-formula is provably equivalent to the one with a single leading existential quantifier, e.g.,

$$BA \vdash \exists x_1 \exists x_2 A_0(x_1, x_2) \leftrightarrow \exists x \exists x_1, x_2 \leq x\, A_0(x_1, x_2).$$

Thus we shall sloppily use the $\prec$ notation for arbitrary Σ_1-formulas (and formulas logically equivalent to them).[3] On the proof-theoreticians' slang the $\prec$ symbol is usually called *witness comparison,* and one calls the numbers x satisfying $A_0(x)$ and $B_0(x)$ *witnesses* of the formulas $\exists x A_0(x)$ and $\exists x B_0(x)$, respectively.

THEOREM 3.11 (PUTNAM-SMULLYAN). Let T be a consistent, r.e., arithmetical theory containing BA. Then every pair of disjoint r.e. predicates has a Σ_1 dual numeration in T.

PROOF. To simplify notations, we shall only prove this theorem for the case of unary predicates (sets). Let $A(x) :\leftrightarrow \exists y A_0(x, y)$ and $B(x) :\leftrightarrow \exists y B_0(x, y)$ be the Σ_1-formulas defining the given sets in the standard model, and let $P(x) :\leftrightarrow \exists y P_0(x, y)$ define the set of theorems of T, where A_0, B_0, P_0 are elementary. Consider a solution of the following parametric fixed point equation (*Sheperdson's fixed point*):

$$BA \vdash F(x) \leftrightarrow (P(neg(\ulcorner F(\dot{x}) \urcorner)) \vee A(x)) \prec (P(\ulcorner F(\dot{x}) \urcorner) \vee B(x))\,. \qquad (*)$$

Here $neg(z)$ represents in BA the function mapping the Gödel number of a formula z to the Gödel number of its negation:

$$neg(z) := \begin{cases} \ulcorner \neg C \urcorner, & \text{if } z = \ulcorner C \urcorner,\ C \text{ a formula,} \\ 0, & \text{if no such } C \text{ exists.} \end{cases}$$

Obviously, formulas on both sides of $\prec$ can be translated as proper Σ_1-formulas. More carefully, $P(neg(\ulcorner F(\dot{x}) \urcorner))$ is written as

$$\exists y, z\, (\ulcorner F(\dot{x}) \urcorner = y \wedge neg(y) = z \wedge P(z)).$$

By Lemma 3.3 (ii) a solution $F(x)$ exists. We show that the formula $F(x)$ dually numerates (A, B) in T. There are four implications to prove.

(a) Assume $T \vdash F(\overline{n})$. Let m be the Gödel number of a proof of $F(\overline{n})$ in T. Since m witnesses the right hand side of $\prec$ and $T \vdash F(\overline{n})$, provably in T there should be an earlier witness of the left hand side:

$$T \vdash \exists x \leq \overline{m}\, P_0(\ulcorner \neg F(\overline{n}) \urcorner, x) \vee \exists x \leq \overline{m}\, A_0(\overline{n}, x). \qquad (2.8)$$

[3] The formulas $A_1 \prec B$ and $A_2 \prec B$, for graphically different logically equivalent A_1 and A_2, in general, need not be BA-equivalent. Yet all the uses of $\prec$ notation in this book work whichever of the equivalent variants the reader would take.

Here we used the fact that, for the given numeral $\overline{n}$, $neg(\ulcorner F(\overline{n})\urcorner) = \ulcorner\neg F(\overline{n})\urcorner$ holds provably in BA.

Since T is consistent, the elementary formula

$$\forall x \leq \overline{m}\, \neg P_0(\ulcorner\neg F(\overline{n})\urcorner, x) \tag{2.9}$$

is true and provable in BA. Combining (2.8) and (2.9) together we obtain $T \vdash \exists x \leq \overline{m}\, A_0(\overline{n}, x)$. Therefore, $\mathbf{N} \vDash \exists x \leq \overline{m}\, A_0(\overline{n}, x)$, for otherwise by Σ_1-completeness T would be inconsistent. Hence $\mathbf{N} \vDash A(\overline{n})$.

(b) Assume $\mathbf{N} \vDash A(\overline{n})$. Since A and B define disjoint sets, we have $\mathbf{N} \nvDash B(\overline{n})$, and hence for some m witnessing $A(\overline{n})$,

$$BA \vdash A_0(\overline{n}, \overline{m}) \wedge \forall x \leq \overline{m}\, \neg B_0(\overline{n}, x). \tag{2.10}$$

Now we consider two cases. If $\forall x \leq \overline{m}\, \neg P_0(\ulcorner F(\overline{n})\urcorner, x)$ is true, a witness of $A(\overline{n})$ appears earlier than any witness of $P(\ulcorner F(\overline{n})\urcorner) \vee B(\overline{n})$. Hence the right hand side of the fixed point equation $(*)$ is true and provable in BA by Σ_1-completeness. It follows that $F(\overline{n})$ is provable in BA and T.

If $\forall x \leq \overline{m}\, \neg P_0(\ulcorner F(\overline{n})\urcorner, x)$ is false, then, by the meaning of this formula, again $F(\overline{n})$ is provable in T. Thus we have shown that in each case $T \vdash F(\overline{n})$. Notice that (a) and (b) just proved show that $F(x)$ numerates A in T. The proof of the fact that $\neg F(x)$ numerates B in T is roughly symmetrical (with the roles of A and B, F and $\neg F$ interchanged). We leave it as an excercise for the reader, q.e.d.

COROLLARY 3.12 (EHRENFEUCHT-FEFERMAN). Let T be a consistent, r.e., arithmetical theory containing BA. Then every r.e. predicate has a Σ_1-numeration in T.

PROOF. For a given r.e. predicate A consider a formula dually numerating the pair $(A, \emptyset)$ in T, q.e.d.

Notice that, unlike for the case of Σ_1-sound theories, the numeration constructed in Corollary 3.12 essentially depends on the theory T. Excercise ?? shows that this cannot be avoided.

3.29. Notes.