1. From decidability to feasibility

1.1. Prerequisites: the language of first-order logic

We can introduce first-order languages through the concept of signature. Once the logical symbols $\neg = not$, $\wedge = and$, $\vee = or$, $\rightarrow = if...then$, $\exists = for\ some$, $\forall = for\ all$ have been established. The signature is essentially the list of non-logical symbols of the language; more formally, it is a quadruple $\langle \mathcal{F}, \mathcal{P}, \mathcal{C}, ar \rangle$, where $\mathcal{F}$ is the set of functional symbols of the language, $\mathcal{P}$ is the set of relational symbols, $\mathcal{C}$ is the set of constant symbols, and ar is a function that assigns each functional or predicative symbol its arity. For example, in the theory of ordered rings $\mathcal{F} = \{+, -, \cdot\}$, $\mathcal{P} = \{\leq\}$ and $\mathcal{C} = \{0, 1\}$; in Peano's first-order arithmetic theory, $\mathcal{F} = \{S, +, \cdot\}$, $\mathcal{R} = \{\leq\}$ and $\mathcal{C} = \{0\}$; in Zermelo-Fraenkel first order set theory, $\mathcal{R} = \{\in\}$ and no constant or function symbols occurs. With regard to a particular relation, namely identity (or equality) $=$, there are two options:

1. (More common) Include this relation among the logical symbols and always interpret $t = s$ as "t and s denote the same object", considering the identity axioms as logical rules (*theories with identity*).

2. Consider $=$ as a binary relation of the signature, interpretable as a set of pairs of elements of the domain, adding, however, axioms that specify that this relation must be an equivalence and a congruence (*theories without identity*).

Terms are builded from function symbols, constant symbols and variables. Thus, any constant c and variable x are terms, and if $t_0, ..., t_n$ are terms and f is a symbol of -arity n, then $f(t_0, ..., t_n)$ is a term. *Atomic formulas* are defined to be of the form $P(t_0, ..., t_n)$ where P is a k -ary relation symbol of the signature. In particular, for t, s terms, $t = s$ is an atomic formula. More complex formulas are inductively defined from atomic formulas and logical constants; hence any atomic formula is a formula, and if α and β are formulas, then so are $(\neg\alpha), (\alpha \wedge \beta), (\alpha \vee \beta), (\alpha \rightarrow \beta), \forall x\alpha$ and $\exists x\alpha$.

We therefore introduce Tarski's semantics for first-order logic in a somewhat pedantic manner, starting from the notion of *realisation*. A *realisation* is a pair $\mathcal{U} = \langle A, \rho_U \rangle$, where:

1. A is a non-empty domain of objects (the universe of discourse).

2. ρ_U is function that assigns a meaning to every *constant* of the language, whether individual, functional or predicative, while assigning to variables only a domain of variation, i.e. A.

Logics that admit empty domains and terms that do not denote any object are called 'free logics'. They also admit terms t that denote objects outside the (non-empty) domain for which, clearly, the principle $\forall x\phi(x) \rightarrow \phi(t)$ is not acceptable in the aforementioned domain. More precisely, a classical realisation satisfies these conditions:

1. For each individual constant c_i of the language,$\rho_U(c_i)$ is an object of A, which we will denote by c_i^U.
2. For each functional symbol f_i, $\rho_U(f_i)$ is *an operation* of the same -arity as f_i on A, which we also denote by f_i^U.

Duccio Pianigiani, University of Siena, Italy, duccio.pianigiani@unisi.it, 0000-0001-9441-7226

Referee List (DOI 10.36253/fup_referee_list)
FUP Best Practice in Scholarly Publishing (DOI 10.36253/fup_best_practice)

Book References DOI 10.36253/979-12-215-0778-2.references

3. For each predicative symbol P_i, $\rho_U(P_i)$ is a *relation* of the same -arity as P_i on A, which we denote by P_i^U. In particular, if $\rho_U(=)$ is precisely the set of pairs $< a, a >$ of elements of A, we say that it is a *normal* realisation.
4. For each variable x_i, $\rho_U(x_i) = A$.

Other presentations can be found in the scientific literature. For example, we can consider the structure (of the same signature as the language) obtained by explicitly writing, instead of ρ, the *interpretations* of the relational and functional symbols and constants according to ρ. If $\rho(R_i) = R_i^U$, $\rho(f_i) = f_i^U$, $\rho(c_i) = c_i^U$. We will denote the realisation as follows:

$$\mathcal{U} = \langle A, R_0^U, ..., R_n^U, f_0^U, ..., f_m^U, c_0^U, ..., c_k^U \rangle$$

Hence, many presentations start from the concept of an *algebraic-relational structure* of a certain signature and a language for this structure, i.e. a language of the same type of similarity. Notice that so far, having established a realisation, we are only able to interpret closed terms and statements in the strict sense. *Open* formulas are defined as those containing free variables. They are either *satisfied* or not, depending on how the free variables are interpreted in a given domain. Following the most orthodox approach, we give a simultaneous interpretation σ of all variables: by *evaluation in A* we mean a function $\sigma : Var \longrightarrow A$, i.e. an infinite sequence $\sigma = \langle \sigma(x_0), \sigma(x_1), \sigma(x_2), ... \rangle$ of objects of A. Since we do not know in advance which variables might appear in a formula, the simplest solution is in fact to require that an assignment assigns values to *all* variables. By *x_i-variant of σ* we mean a sequence:

$$\sigma(a/x_i) = \langle \sigma(x_0), \sigma(x_1), \sigma(x_2), ...\sigma(x_{i-1}), a, \sigma(x_{i+1})... \rangle$$

that is, a function that differs from σ only in the value it assigns to x_i. The evaluation of terms, induced by σ in a realisation $\mathcal{U}$ is the following:

1. $\rho_U^\sigma(c_i) = \rho_U(c_i) = c_i^U$, for every constant$c_i$.
2. $\rho_U^\sigma(x_i) = \sigma(x_i)$, for every variable$x_i$.
3. $\rho_U^\sigma(f(t_0, ..., t_n)) = \rho_U(f)(\rho_U^\sigma(t_0), ..., \rho_U^\sigma(t_n))$.

To formally define the notion of *truth*, we must first establish the meaning of the variables. At this point, however, we are able to evaluate the formulas; we arrive at the notion of truth, following Tarski, by passing through the relation of *satisfaction*. Some key notions of semantics will be the following:

1. *Satisfiability*: ϕ is satisfiable in $\mathcal{U}$ if there exists some assignment σ to the variables that satisfies ϕ in $\mathcal{U}$ (notation $\mathcal{U} \models_\sigma \phi$)
2. *Truth*: ϕ is true in $\mathcal{U}$ if and only if it is satisfied by every σ (notation $\mathcal{U} \models \phi$).
3. *Validity*: ϕ is universally valid if and only if it is true in every realisation (notation $\models \phi$).

The *statements*, which in this context are properly thoseformulas that are free of variables, will have the characteristic of being *satisfied by all assignments*, or *not satisfied by any assignment* of value to the variables, with respect toa certain *realisation*. A statement will be said to be *true in* ρ_U, or *false in* ρ_U.

We define the satisfaction relation σ satisfies ϕ in $\mathcal{U}$(notation $\rho_U^\sigma \models \phi$, or more commonly $\mathcal{U} \models_\sigma \phi$), inductively, through the following clauses:

1. $\mathcal{U} \models_\sigma P_i(t_0, ..., t_n)$ if and only if $\langle \rho_U^\sigma(t_0), ..., \rho_U^\sigma(t_n) \rangle \in \rho_U^\sigma(P_i)$
2. $\mathcal{U} \models_\sigma \neg\psi$ if and only if $\mathcal{U} \not\models_\sigma \psi$
3. $\mathcal{U} \models_\sigma \phi \wedge \psi$ if and only if $\mathcal{U} \models_\sigma \phi$ and $\mathcal{U} \models_\sigma \psi$ (similarly for $\vee$)
4. $\mathcal{U} \models_\sigma \phi \rightarrow \psi$ if and only if $\mathcal{U} \models_\sigma \neg\phi$ or $\mathcal{U} \models_\sigma \psi$
5. $\mathcal{U} \models_\sigma \exists x_j \phi$ if and only if $\mathcal{U} \models_{\sigma(a/x_j)} \phi$ for some $a \in \mathcal{U}$.

6. $\mathcal{U} \models_\sigma \forall x_j \phi$ if and only if $\mathcal{U} \models_{\sigma(a/x_j)} \phi$ for every $a \in \mathcal{U}$.

By the *Coincidence lemma*, if σ, τ are two assignments that coincide on the free variables of t. Then $\rho^\sigma_\mathcal{U}(t) = \rho^\tau_\mathcal{U}(t)$. Furthermore, if σ, τ are two assignments that coincide on the free variables of a formula ϕ, then $\mathcal{U} \models_\sigma \phi$ if and only if $\mathcal{U} \models_\tau \phi$. In particular, if ϕ is a statement, i.e. it has no free variables, then if $\mathcal{U} \models_\sigma \phi$ *for some* σ, then $\mathcal{U} \models_\sigma \phi$ *for every* σ. Note that two assignments σ, τ clearly coincide in assigning values to all variables in the set X, if X *è empty*. A statement is therefore *true in all evaluations* associated with a given realisation, or it is not true in *any* of them ("true in ρ_U", or "false in ρ_U"). A *formula* ϕ can be satisfied by *all* the evaluations associated with a given realisation (we will write $\mathcal{U} \models \phi$ and say that it is true in that realisation), by *none* of them, or only by *some* of them. A formula ϕ is universally (or logically) valid if it is true in all realisations. As regards deductive systems, in this volume we will introduce those most commonly used in *Proof Theory*, i.e. those *à la* Gentzen (many rules and few axioms). However, it is useful to recall a typical axiomatic presentation (in Frege-Hilbert style), consisting of many logical axioms and few deduction rules. In axiomatic derivations, a sequence of sentences counts as a correct derivation from a set of hypothesis, if every sentence in it either is an axiom, or is an element of the given set of hypothesis, or come from previous formulas of the sequence by a rule of the calculus. Let us give a typical example of axiomatisation:

1. $p_0 \to (p_1 \to p_0)$
2. $(p_0 \to (p_1 \to p_2)) \to ((p_0 \to p_1) \to (p_0 \to p_2))$
3. $(p_0 \wedge p_1) \to p_0$
4. $(p_0 \wedge p_1) \to p_1$
5. $p_0 \to (p_1 \to (p_0 \wedge p_1))$
6. $p_0 \to (p_0 \vee p_1)$
7. $p_1 \to (p_0 \vee p_1)$
8. $(p_0 \to p_2) \to ((p_1 \to p_2) \to ((p_0 \vee p_1) \to p_2)$
9. $(p_0 \to p_1) \to ((p_0 \to \neg p_1) \to \neg p_0)$
10. $\neg\neg p_0 \to p_0$

Propositional rules:

$$\frac{\phi \quad \phi \to \psi}{\psi} \qquad \frac{\phi}{\phi\sigma}$$

namely Modus Ponens and uniform substitution, where $\sigma$is a substitution: 1) $p\sigma = \sigma(p)$; 2) $\bot\sigma = \bot$; 3) $(\phi \wedge \psi)\sigma = \phi\sigma \wedge \psi\sigma$ (analogously for other connectives). Alternatively, due to the fact that every formula derivable from 1)-10) can be derived by applying the substitution rule only to the axioms, axioms are very often given as *axiom schemes*, i.e. replacing propositional variables with metavariables to represent infinitely many individual axioms, incorporating, so to speak, the substitution rule, which will therefore be omitted. By removing axiom 10, we obtain the *Intuitionistic Propositional Calculus* IPC.

First-order extension. In this case, we add some additional axioms and rules to the axiomatic basis of propositional calculus:

$$\forall x \phi(x) \to \phi(t) \qquad \phi(t) \to \exists x \phi(x)$$

and rules:

$$\frac{\phi \to \psi(z)}{\phi \to \forall x \psi}$$

universal generalization where where z does not appear among the free variables of ϕ.

$$\frac{\phi(z) \to \psi}{\exists x \phi \to \psi}$$

existential generalization where z does not appear among the free variables of ψ. In fact, there are many (obviously equivalent) ways to axiomatise classical first-order logic. Often, only axioms

for implication and negation are given, plus those for the universal quantifier, since connectives, like quantifiers, are interdefinable. This is not the case for intuitionistic logic, where the above-mentioned logical operators are all independent and that is why we have reported an axiomatisation for the full language. We use the symbology "$\Gamma \vdash \phi$" to indicate derivability from hypothesis in the set Γ (perhaps empty) in a given deductive system, of the formula ϕ. For systems that satisfy axioms 1. and 2., the deduction theorem holds:

$$\Gamma \cup \{\gamma\} \vdash \psi \Rightarrow \Gamma \vdash \gamma \to \psi$$

In this volume, we will also discuss classical modal logic systems, i.e., extensions of this logic with operators $\Box$ = *necessarily* and $\Diamond$ = *possibly*. The semantics for this logic are those of Kripke models. The prerequisites will be provided at the time.

1.2. The concept of formal system

The discovery of antinomies in *naïve* set theory led many matematicians in the early 20th century to abandon Cantor's idea of the actual infinite and set-theory. On the contrary, the eminent logician and mathematician David Hilbert, one of the most remarkable figures in mathematics of the entire century, believed that "Cantorian paradise" (i.e. set theory) had to be defended, and considered it his personal duty to defend mathematics against skeptics, clarifying and justifying the mathematician's use of the infinite. Hilbert's reaction to the antinomies concerning the existence of certain sets, was to claim that a set of mathematical objects (e.g. set of all real numbers) exists only if we can prove that the corresponding axiom system for them is free of contradictions, and a proof of this has to be done combinatorially or constructively. Infinitary mathematics must be based on safe *finitary* mathematics, but what is the finitary, in some sense (essentially kantian) intuitive mathematics? Skolem's *Primitive Recursive Arithmetic* PRA, for example, was a candidate to embody the finist methods. Actually there are many different versions of this theory. Among the most common is this one: the first-order language of PRA contains symbols for any primitive recursive function and the relative recursive defining axioms. In this extended language, the theory allows for induction over open formulas. We will see also that we may think of it as the theory $I\Sigma_1$ in the more restricted language of *Peano Arithmetic*. In Tait (1968), the author argues that finitist arithmetic coincides with PRA, even though this identification is largely conventional and disputable. However, after the discovery of Gödel's incompleteness results, wanting to keep the core of this programme, as we will see, it was necessary to expand Hilbert's finististic standpoint. Many issues we will deal with have arisen during the controversy around the *legitimacy of abstract objects* as the infinite sets, which led on the other hand to the clarification of the concept of *finite*, *effective*, *algorithmic* procedures. Hilbert never formulated his program exactly, however, in a nutshell, we can say that it consisted of the following steps:

1. Formalize all infinitistic mathematics in an higher-order formal system of mathematical logic.

2. Show the consistency of this system with finitist tools, which have purely combinatorial character. We will see that this implies the elimination, in principle, of the ideal objects (e.g. uncountable sets) from the proof of concrete, "real" statements (certain universal sentences $\forall x \psi(x)$, where $\psi(x)$ is a decidable predicate): the justification for the infinitary mathematics lies in its conservativity over real, finitisitically meaningful mathematics.

Gödel's second incompleteness theorem (1931) showed that this program was unattainable: no consistent, recursively axiomatized system that contains sufficient mathematics proves its own consistency. The first incompleteness theorem, on the other hand, showed that for a wide class of formal theories, there are true sentence that can be expressed in the language of these theories, but that are neither provable, nor refutable in them. Lastly, Hilbert and Ackermann in 1928 also formulated the *Entscheidungsproblem* (thet means "decision problem"), i.e. the problem of devising an algorithm for deciding the validity of statements of first-order logic, but in 1936, Alonzo Church and Alan Turing independently showed that a general solution to this problem is impossible. To show this, they gave a deep formal treatment of the intuitive notion of "effectively computable

method" which gave rise to the modern abstract models of computation, which we will account for in the book. This section is devoted to recalling some fundamental metatheoretical concepts related to formalised mathematical theories.

Formal mathematical theories. A formal mathematical (first order) theory is given by the following ingredients:

1. A first order language L.
2. A set of logical axioms (formulas of L).
3. A set of inference rules.
4. A set T of formulas containg the logical axioms and closed under the inference rules, i.e. if $\Gamma \subseteq \mathsf{T}$ and ϕ is a formula of the language derivable from Γ by means of the logical axioms and the inference rules (write $\Gamma \vdash \phi$), then $\phi \in \mathsf{T}$.
5. If a set Γ exists such that $\mathsf{T} = \{\phi | \Gamma \vdash \phi\}$, then Γ is a set of *non-logical axioms* for T and T is the set of *theorems*. As usual, we will write $\mathsf{T} \vdash \phi$ to mean that ϕ is derived from the axioms of T by means of logical axioms and inference rules of the theory.

In this book we will actually favour formal systems in the style of Gentzen, i.e. many rules and few axioms (see on p. 174). For an axiomatic theory T, let $Thm(\mathsf{T}) = \{\phi | \mathsf{T} \vdash \phi\}$ the set of theorems and $Ref(\mathsf{T}) = \{\phi | \mathsf{T} \vdash \neg\phi\}$ the set of refutable sentences. We say that the theory T is *consistent* iff $Thm(\mathsf{T}) \cap Ref(\mathsf{T}) = \emptyset$. We say that T is *syntactically complete* if for each sentence ϕ of the language of T, either $\phi \in Thm(\mathsf{T})$, or $\phi \in Ref(\mathsf{T})$. We say that T is *decidable*, if the set $Thm(\mathsf{T})$ is algorithmically decidable. Lastly, we say that S is an *extension* of T iff the language of T is a subset of the language of S and all theorems of T are also theorems of S.

Among the various meanings in logic of the word "completeness" it is worth remembering the following:

1. *Syntactic completeness.* An axiomatic theory is complete in this sense, if for all sentence of its language ϕ, either ϕ, or $\neg\phi$ is a theorem of it.
2. *Completeness with respect to the truth.* If for instance $\mathsf{T} = \mathsf{PA}$, then, if ϕ is true in the standard (intendend) model $\mathbb{N}$ of natural numbers, then it is a theorem of T.
3. *Semantic completeness.* A sentence ϕ is a theorem of a certain theory, if and only if it is true in *all* models $\mathfrak{M}$ of that theory. First order theories are complete in this third sense.

Gödel's first theorem states that for many theories, even weaker than PA, properties 1. and 2. do not apply.

Decidability. A theory is called *decidable* if the set of its theorems, i.e. the sentences derivable in it, is *decidable*, that is, there is a mechanical procedure which enables one to decide whether an arbitrary given sentence of the language of the theory is a theorem or not. A theory is *axiomatizable* if there is a decidable set of axioms for it. If a theory is axiomatizable and *syntactically complete*, then it is *decidable*, although the converse does not always hold: there are *incomplete* theories which are *decidable* (for instance, the theory ACF of algebraically closed fields is decidable, but not complete: it becomes complete when we fix the characteristic and we get an ACF_p). All theories which contain Robinson arithmetic Q (see below) or that interpret it are both incomplete and undecidable. Contrasting with the undecidability of Q we have Tarski's decidability result of the theory of ordered *Real Closed Fields*.

Axiomatizability. The theory Q is *finitely axiomatized*. However, the set of axioms of our formal systems will be often *infinite*, but we will be able to determine by an effective (algorithmic) procedure whether a certain formula is an axiom or not, namely, the theory will be *recursively axiomatizable*. Or, at least, we will be able to enumerate all and only the axioms: we say in this case that the theory has a *recursively enumerable set of axioms*. Craig's theorem (see p. 69) clarifies the connection among these alternatives.

Theorem 1. *If a theory T is recursively axiomatizable (often called simply "axiomatizable") and syntactically complete, then is decidable.*

Proof. Recall that if a theory is axiomatizable, then the set of its theorems is recursively (or "computably") enumerable (see e.g. Enderton (2001) 156-57). Suppose now T is complete and has

a computable set of axioms. If T is *inconsistent*, it is clearly computable (Algorithm: "just say yes"). If T is *consistent*, to decide whether or not a sentence ϕ is in T, simultaneously search for a derivation of ϕ from T and a derivation of $\neg\phi$. Since T is complete, you are bound to find one or the other; and since T is consistent, if you find a derivation of $\neg\phi$, there is no derivation of ϕ.
QED

The complete theory of a model.

As we have already mentioned in the prerequisites, a structure (or model) $\mathcal{U}$ for a first-order language consists of a non-empty domain equiped with distinguished relations, functions and designed individuals on it, that interpret respectively the relational, functional and constant symbols of the language. We denote $Th(\mathcal{U})$ the so-called *complete theory* of the model $\mathcal{U}$, namely the set of true sentences defined on it. Clearly, if L is the language of the model, then for any $\phi \in L$, either $\phi \in Th(\mathcal{U})$, or $\neg\phi \in Th(\mathcal{U})$. It is typical in this regard to contrast these two historical results:

1. First order theory of *Real Closed Ordered Fields* RCOF is recursively (or effectively) axiomatizable and complete in the first sense. For a celebrated theorem due to Tarski:
$$\mathsf{RCOF} = Th(\langle \mathbb{R}, +, -, \cdot, 0, 1, < \rangle)$$
2. The first Gödel theorem, states at the opposite that no consistent, effectively axiomatizable and sufficiently powerful theory of arithmetic is complete in the first two senses. Hence $Th(\langle \mathbb{N}, +, \cdot, S, 0 \rangle)$, although complete, *is not effectively axiomatizable.*

The First Incompleteness Theorem can be actually formulated as follows.

Theorem 2. (Gödel 1931) *There is no decidable set of axioms for the complete theory of the standard model.*

Proof. If a theory is *consistent* and *represents (or binumerates) all computable functions*[1], then is *undecidable*. If a theory is *axiomatizable* and *complete* then is *decidable*. Then if a theory is *consistent*, *axiomatizable* and *represents all recursive functions*, it *cannot be complete*. Now consider that the complete theory of the standard model is complete and represents all computable functions: hence cannot be axiomatizable.
QED

For "sufficiently powerful theories", we will mean extensions of the theory of formalized arithmetic theory Q (Robinson Arithmetic), a finitely axiomatized theory in the language $L = \{\overline{0}, S, +, \cdot\}$ whose axioms are[2]:

1. $Sx \neq \overline{0}$
2. $Sx = Sy \to x = y$
3. $x + \overline{0} = x$
4. $x + Sy = S(x + y)$
5. $x \cdot \overline{0} = \overline{0}$
6. $x \cdot Sy = x \cdot y + x$
7. $x \neq \overline{0} \to \exists y(Sy = x)$.

In we replace axiom 7. with the axiom scheme of induction:

$$\phi(\overline{0}) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x)$$

we get the theory (not finitely axiomatizable) called "first-order Peano arithmetic", denoted by PA.

Strong fragments of Peano Arithmetic. Recall this classification of formulas (here of the language of PA), called *Arithmetical Hierarchy.* A formula is called a *bounded formula* if it contains only bounded quantifiers $\forall x \leq t$ or $\exists x \leq t$. The set of bounded formulas is denoted Δ_0. For $n \geq 0$ the classes Σ_n and Π_n of first-order formulas are inductively defined by:

[1] For instance, Odifreddi (1989-1999), 40-1 uses the terminology "represents, weakly represents" , while in Hájek and Pudlák (1993), 155 is used the terminology "binumeratess, numerates".

[2] Hoping not to cause confusion, and to avoid further burdening the notation, we will use in all the book the symbol "=" for equality, for conversion and to mean equal by definition. We hope the reader will grasp the different meanings intuitively.

1. $\Sigma_0 = \Pi_0 = \Delta_0$,
2. Σ_{n+1} is the set of formulas obtained by prepending an arbitrary block of existential quantifiers and bounded universal quantifiers to Π_n-formulas.
3. Π_{n+1} is the set of formulas obtained by prepending an arbitrary block of universal quantifiers and bounded existential quantifiers to Σ_n-formulas.

We want to point out that the correct notation would be for example Π_n^0 and Σ_n^0 where the superscript "0" means that we are talking about first-order formulas (while the superscript "1" would mean that we are talking about second-order formulas). If we do not write anything it will be implicit that we are talking about first-order formulas. Recall that in classical logic it is possible to rewrite formulas to bring all quantifiers to the front (*prenex normal form*), hence each formula is in one of the above form. A formula is Δ_n in the considered theory (e.g. PA) if it is provably equivalent in the theory to both a Σ_n formula and a Π_n formula. Note that if a formula is in Π_n, then is also in Σ_{n+1}. A corresponding hierarchy of subtheories of PA is defined by restricting the induction axiom to a fixed level of the arithmetic hierarchy. We denote $I\Sigma_n$ Peano Arithmetic with induction restricted to the class Σ_n.

The so called *strong fragments* of Peano Arithmetic, denoted by $I\Sigma_n$, or $I\Pi_n$, for $n > 1$, are obtained by restricting the induction schema to the classes Σ_n, resp. Π_n. These fragments are *finitely axiomatizable* (see Hájek and Pudlák (1993), 77-81) because there exists a *universal formula* for them (this is not the case for the full PA). A universal formula for Σ_n is a Σ_n-formula $\Psi(x, y)$ such that for each Σ_n- formula $\theta(x)$ there exists a number e such that:

$$I\Sigma_1 \vdash \theta(x) \leftrightarrow \Psi(x, \overline{e})$$

where $\overline{e} = \overbrace{SSS...S\overline{0}}^{e-times}$. Hence we can collapse infinitely many instances of induction in a *unique* axiom:

$$\forall y(\Psi(\overline{0}, y) \wedge \forall x(\Psi(x, y) \rightarrow \Psi(x + 1, y)) \rightarrow \forall x\Psi(x, y))$$

Why the theory Q? This theory was introduced by Raphael Robinson in 1950 and in fact, it is not adequate to the formalization of arithmetic, due to the absence of the induction principle. However we will see what is the tecnical reason of interest: it *binumerates* the recursive functions (see 1). We will see that Q, although very weak, still is *essentially undecidable* (all consistent extensions of it are also undecidable).

It should be remarked that we can go much weaker in complexity, e.g. by considering a theory named R, introduced in Tarski, Mostowski and Robinson (1953) which is strictly weaker than Q, is still essentially undecidable, binumerates the recursive functions, but is not finitely axiomatizable. In some way Q is a minimal finitely axiomatizable theory in which every recursive function is "binumerable" or ("representable"). The importance of this theory is also due to foundational reasons in the current debate on constructive mathematics, where some go beyond the traditional finitist or intuitionist positions. For them, it represents a sort of paradigm. In this respect, it is worth mentioning the influential Nelson (1986) where the *interpretability in* Q is synonymous with *safety*. The author's radical finitism poses a barrier in the *totality of exponentiation*, a fact he didn't believe in. His strictly finitist program implies the denial of the existence of certain large natural numbers as 2^{65536} and claims that we should try to develop mathematics under the assumption of the denial of the totality of exponentiation. We will see (Parikh's theorem 111) that the totality of exponentiation cannot be proved in very weak theories.

A theory is *essentially incomplete* if all its recursively axiomatizable extensions are incomplete. Gödel (Rosser) theorem in fact says that a certain weak base theory Q is essentially incomplete (i.e also incompletable) in the first two sense. Robinson proved that Q is also *essentially undecidable*, i.e. that any consistent theory that extends (or interprets) Q is undecidable. A proof of incompleteness that uses these properties of recursively enumerable and recursive sets rather than self-reference can be called *structural*.

No axiomatic subtheory of Q obtained by removing any one of its axioms remains essentially undecidable (see Tarski, Mostowski and Robinson (1953)). Let us remove for example the axiom

$S(x) = S(y) \to x = y$. Call Q^- the theory thus obtained and consider the structure:

$$\mathcal{M} = \langle \{0, 1\}, 0, S, +, \cdot \rangle$$

where 0 and $\cdot$ have their usual meaning, but $S(\overline{0}) = S(\overline{1}) = 1$; $0 + 0 = 0$ and

$$x + y = \begin{cases} 0 & if\ x = y = 0 \\ 1 & if\ x = 1\ or\ y = 1 \end{cases} \tag{1}$$

Note that all the other axioms are true in this structure. Let $Th(\mathcal{M})$ the set of sentences true in this structure, observing that it is an extension of Q^- and it is *decidable*: indeed it proves

$$\forall x \phi(x) \leftrightarrow \phi(\overline{0}) \wedge \phi(\overline{1}), \quad \exists x \phi(x) \leftrightarrow \phi(\overline{0}) \vee \phi(\overline{1})$$

The statement that first Incompleteness theorem by Gödel holds for the consistent axiomatizable theory T is actually equivalent to the each of the statements: "T is essentially incomplete" and "T is essentially undecidable". The equivalence depends in particular on the facts that every *incomplete decidable* theory has a consistent, *decidable complete* extension in the same language, and that it is well known that every consistent *axiomatizable* and *complete* theory is *decidable* (if the theory is not consistent, the algorithm tha decides it is the one that says always YES; otherwise, being the theory axiomatizable, we can effectively make a list of its correct derivations: if a sentence does not appear in the list, since the theory is complete, it will appear its negation) .

In more recent years (see, among others, Visser (2009)), attention has also been paid to the "other" Robinson arithmetic R, axiomatized by the following schemes (where the language is $0, S, +, \cdot, \leq$ and the numerals $\overline{n}$ are defined by means of the successor S):

1. $\overline{m} + \overline{n} = \overline{m + n}$
2. $\overline{m} \cdot \overline{n} = \overline{m \cdot n}$
3. $\overline{m} \neq \overline{n}$, if $m \neq n$.
4. $\forall x(x \leq \overline{n} \to x = \overline{0} \vee ... \vee x = \overline{n})$
5. $\forall x(x \leq \overline{n} \vee \overline{n} \leq x)$

If we remove the axiom 5. the remaining theory R_0 is no longer essentially undecidable. However, if $\leq$ is not assumend as primitive and $x \leq y$ is defined as $\exists z(z + x = y)$, then the theory R_0 is still essentially undecidable. If in axiom 4. we put $\leftrightarrow$ in place of the implication, the theory is essentially undecidable, but not minimal essentially undecidable (for a more detailed discussion, see Jones (1983)).

The theory R is *intepretable* (in the sense of the definition on p. 112) in Q, but not viceversa: if Q were interpretable in R, then it would be intepretable in a finite fragment of R, but finite fragments of R have finite models ("locally finitely satisfable") where, on the contrary, Q has only infinite models.

Theorem 3. (Ryll-Nardzewski 1952) *There is no finite set of axioms* S *of the language of* PA*, such that* $\mathsf{S} \vdash \phi$ *if and only if* $\mathsf{PA} \vdash \phi$.

Proof. A direct argument appeals to Gödel's second result: reasoning by contradiction if PA were finitely axiomatizable, and therefore had finite instances of theinduction axiom, then would be equivalent to some $I\Sigma_k^0$, i.e. PA, with the induction restricted to formulas of the form $\exists x_0 \forall x_1 \exists x_2 ... Q x_{k-1} \theta$ for some fixed k. However it is provable that $\mathsf{PA} \vdash Con(I\Sigma_k^0)$, for all k. Hence we would have $I\Sigma_k^0 \vdash Con(I\Sigma_k^0)$, in other words $\mathsf{PA} \vdash Con(\mathsf{PA})$, against Gödel's second theorem. QED

The first order logic has earned only with time the fame of "standard logic". From a famous theorem in Lindström (1969), we now know that countable compactness and downwards Löwenheim-Skolem characterize first-order logic, in the sense that it is the strongest logic to fulfils both, where:

1. *Compactness* (K. Gödel 1930, A. I. Malčev 1937)): "Γ has a model iff each of its finite subset $\Sigma \subseteq \Gamma$ has a model", and

2. *Löwenheim-Skolem theorems* (L. Löwenheim 1915, T. Skolem 1920):

   (a) (Löwenheim-Skolem downwards). Let Σ a set of sentencese of the first order language $\mathcal{L}$ and let its cardinality $|\mathcal{L}| = \kappa$; let $\kappa < \lambda$. If Σ has a model of cardinality λ, then Σ has a model of cardinality $\kappa \leq \alpha < \lambda$.

   (b) (Löwenheim-Skolem upwards). Let Σ a set of sentences of the first order language $\mathcal{L}$ and let its cardinality $|\mathcal{L}| = \kappa$. If Σ has a model of cardinality $\lambda \geq \kappa$ then Σ has also a model of cardinality μ, for all $\mu \geq \lambda$.

It is worth making a comparison with the *second order* Peano arithmetic PA_2 (in the language with the only functional symbol S, the successor, were $Sn = n + 1$, and a constant 0), a finitely axiomatizable theory with axioms:

1. $\forall x \neg (Sx = 0)$

2. $\forall x \forall y (Sx = Sy \rightarrow x = y)$

3. $\forall X ((X(0) \wedge \forall x (X(x) \rightarrow X(Sx)) \rightarrow \forall y X(y)))$

The privileged interpretation consists of the second-order structure:

$$\mathcal{N} = \langle \mathbb{N}, \mathcal{P}(\mathbb{N}), \overline{0}, \overline{S} \rangle$$

From a theorem due to Dedekind (1887), it follows that all *principal* models $\langle U, F, a, g \rangle$ of this theory (i.e. those models in which the second-order variables vary over *all* the power-set of the domain U, namely $F = \mathcal{P}(U)$) are isomorphic: that is to say, with respect to this semantics, PA_2 is *categorical*. It is well known that (because of compactness theorem and Löwenheim-Skolem theorems) categoricity *does not hold* if we formulate Peano arithmetic in the First Order Logic, and that there exist non standard models, besides the standard one. But the categoricity of $\mathsf{PA}_2$is strictly dependent from the choice of the above semantic of *principal* models. A more *general* interpretation of the second-order language is a structure:

$$\mathcal{U} = \langle \{\mathcal{F}_i | i \in \mathbb{N}\}, P_0, ..., P_{i_j}, \{a_i | i \in I\} \rangle$$

where, for all i, $\mathcal{F}_i$ is a family of $i - ary$ relations on $A^i = A \times ... \times A$, where are interpreted the relational variables. If for all i, $\mathcal{F}_i$ coincides with the whole-power set $\mathcal{P}(A^i)$, then we say that the interpretation is *principal*. For *general* interpretations, Leon Henkin (1950), showed the semantic completeness (i.e. the compactness) and Löwenheim-Skolem and thus the non-categoricity. So, what is second order logic with general semantic? The "nothing but" thesis says: Second-order logic with the general semantics is nothing but first-order logic (many-sorted) together with the *comprehension axioms*. Thus a sentence is valid in the general semantics iff it is logically implied (in first-order logic) by the set of comprehension axioms:

$$\exists P \forall u_0 ... \forall u_n (P(u_0, ..., u_n) \leftrightarrow \phi)$$

where ϕ is a second order formula not containing P. These tell us what sets exist. The second order language is therefore treated often as a two-sorted first order language and in this case a model consists of two sets, where the elements of the first domain are interpreted as numbers and the elements of the second domain are interpreted as sets.

1.3. Models of computation: Turing machines

The computational model proposed by Alan Turing has some highly intuitive features. Comparing the various approaches that emerged at the time, Gödel stated the following:

I was completely convinced only by Turing's paper (K. Gödel, 1968).

In Turing (1936) a computation model is introduced, obtained by analyzing the activity of an idealized human agent ("computator") that performs computations with pencil and paper writing symbols on a squared paper, running operations "so elementary, which is not easy to imagine that they can be further broken down". It is assumed (*finiteness conditions*) that the number of symbols, as well as the number of squares observed at any one moment, and the number of "mental states" of computator are *finite*. The operations which the idealized human runs are change the symbol in the observed square and move to a different set of squares; operations depend only on the internal state of computator and the symbol scanned. Turing suggested that any computable function, in an informal way, was "computorable" (*Turing Thesis*). In fact he proved with a rigorous argument (although not in a formal system) that every function "computorable" is computable by a Turing machine (*Turing theorem*). Important matematicians immediately accepted Turing's analysis. Within a very short time, the various models of computation proposed at the time (Turing machines, recursive functions, lambda calculus...) were shown to be equivalent (see Soare (2014) for an historical reconstruction), but in an interview with W. Aspray, in 1985, Kleene declared:

> Turing's definition of computability was intrinsically plausible, whereas with the other two [recursive functions and λ-definability], a person became convinced only after he investigated and found, much by surprise, how much could be done with the definition.

The Turing machine was shortly thereafter taken as a model for many things: in Gödel's opinion, Turing's definition fulfils all defining properties of a formal mathematical system (see Feferman (2006)). According to some exponents of the computationalist theory of mind, starting from McCulloch and Pitts (1943), the Turing machine might even provide a model for the mind. Other, starting from Lucas (1961), on the contrary, claim that Gödel's first theorem refutes the mechanistic thesis that the human mind is, or can be accurately modeled as a Turing machine. However, as regards this discussion, Gödel himself was cautious, and wrote that from his theorem actually simply followed this dichotomy:

> *Either* mathematics is incompletable ... that is to say, the human mind (even within the realm of pure mathematics) infinitely surpasses the powers of any finite machine, *or* else there exist absolutely unsolvable diophantine problems (Gödel (1951), 310).

The Turing model has been used also to study the computational complexity of problems and to define complexity classes. This, we will see, among other things, provides a formal definition to the intuitive notion of "feasible" computation as *polynomial time* computation. We consider first a very simple example of one-tape machines. The machine's tape is potentially *infinite*. This corresponds to an assumption that the *memory* of the machine is (potentially) *infinite*. The cells are all marked with a symbol. In each instant the machine is in a state q_i and after writing something, the head can slide to the right ($= R$) or left ($= L$).

Formally, a deterministic Turing machine, in the simplest version presented here, is a set of instructions based on this alphabet:

1. *Tape symbols* $\{\alpha_0, ..., \alpha_n, *\}$. The symbols which are different from $*$ ("the blank") are called *the alphabet of the machine* (or *input alphabet*).

2. *Internal states* $\{q_0, ..., q_n\}$. We distinguish an *initial state* q_0.

3. *Action symbols* L, R, standing for "left" and "right".

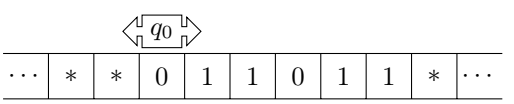

Figure 1. Tape of a Turing machine.

The tape is divided into *cells*, and each cell contains one symbol from the tape alphabet. There is a special blank symbol $*$. At any instant, all but finitely many cells hold $*$. The tape alphabet contains the blank symbol in addition to the alphabet of inputs, but may contain other symbols as well. However, the tape alphabet *cannot be the same as the input alphabet.* The tape alphabet always contains the blank symbol, but the input alphabet cannot contain this symbol. Indeed, if the blank symbol were part of the input alphabet, the Turing machine would never know when the input actually ends. At the beginning the head is positioned on the first symbol on the left different from $*$ and the machine is in the state q_0. Instructions are often given by quadruples. Quadruple Turing machines have three kinds of instructions, namely $q_i\alpha\beta q_j$ ("if You are in the state q_i reading α, print β in its place and go in state q_j") or $q_i\alpha A q_j$ ("if You are in state q_i reading α, go in direction A ($= L, R$) and change the state in q_j"), where α and β are tape symbols. Pure erasing is a special kind of printing $*$. A set of instructions $\Im$ is *consistent*, if for all quadruples we have that if $q_i\alpha A q_j \in \Im$ and $q_i\alpha B q_k \in \Im$, then $A = B$ and $q_j = q_k$. If at a given instant the machine is in a state q_j reading α, but there is no instruction on how to proceed, the computation ends. We can distinguish states with a special role called *final states*. Turing's original formalism of instructions actually used *quintuple*, whereas the quadruple approach is due to Post: the quintuple variation prints and moves at each step, while the quadruple variation does one or the other but not both. Post's version of Turing machine, when in a given state, either prints or moves and so its rules are more elementary. However a quadruple program can be converted into an equivalent quintuple program. There are just two kinds *quintuple* instructions and the sequence $q_i\alpha\beta A q_j$ must be read: "in state q_i read α, write in its place β, go in direction A and change the state in q_j", where A can be $R =$"go right", $L =$"go left"(sometimes also the istruction $I =$"do not move" is added, that does not add power). A more general definition of a single-tape Turing machine $\mathcal{M}$ can therefore be that it is a sequence $\mathcal{M} = \langle \Gamma, \Sigma, Q, *, F, q_o, \delta \rangle$ where Γ is the tape alphabet, $\Sigma \subset \Gamma$ is the *inputs alphabet* (a subset of tape symbols), $* \notin \Sigma$ is a special symbol ("blank") in Γ, Q is a finite set of states, $F \subseteq Q$ is a set of *final states* (e.g. $F = \{q_{f_0}, ..., q_{f_n}\}$), δ is the trasition function. and q_0 is the initial state. An *instantaneous configuration* of the machine is a string of the form $a_0a_1...a_iqa_{i+1}a_{i+2}...a_n$ where $a_0a_1...a_ia_{i+1}a_{i+2}...a_n$ is the portion of the tape between the leftmost and rightmost *nonblanks* (the remaing cells are *blank*), q is the state of the machine and a_{i+1} is the symbol scanned by the tape head. If C_j, C_i are configurations, then $C_j \longrightarrow C_i$ denotes the transition from the first to the second configuration (one step of the computation). Instead we denote with $C_j \longrightarrow^* C_i$ a finite sequence of configurations that begins with C_j and ends with C_i. A computation is just a finite sequence of such configurations. Turing machines can be used either to calculate functions (*transducers*), or to decide "YES" or "NO"problems. In the latter case, when at some point we reach a configuration C_n such that we can not go any further, we have these possible cases:

1. $C_n = ...\alpha q_f \beta...$, where q_f is a final state, then the computation *accepts*.

2. $C_n = ...\alpha q_j \beta...$, where q_j is *not* a final state, then the computation *rejects*.

Recall that, in general, the computation may also *not terminate* and go on forever; therefore it is necessary to distinguish between *recognition* and *acceptance*[3] Let Σ^* be the set of all finite strings (*words*) of an alphabet Σ. We call *language* (or *problem*) a subset of Σ^*:

1. A machine $\mathcal{M}$ *recognizes* (i.e. decides) a language $L \subseteq \Sigma^*$, if for all $\sigma \in \Sigma^*$ there is a *maximal computation* (i.e. such that we can no longer go on) $q_0\sigma \longrightarrow^*_{\mathcal{M}} wq_jz$ such that $q_j = q_f$ iff $\sigma \in L$.
2. A machine *accepts* a language $L \subseteq \Sigma^*$ if for every $\sigma \in L$ it is able to establish this membership; but if $\sigma \notin L$, although it do not says incorrectly that $\sigma \in L$, it can give a negative answer, or go on to calculate forever. In other words, this machine accepts L iff $L = \{\sigma \in \Sigma^* | q_0\sigma \longrightarrow^*_{\mathcal{M}} wq_fz\}$, where q_f is a final state.

[3] The the terminology of the literature is not uniform and the meaning of these terms often does not coincide with ours.

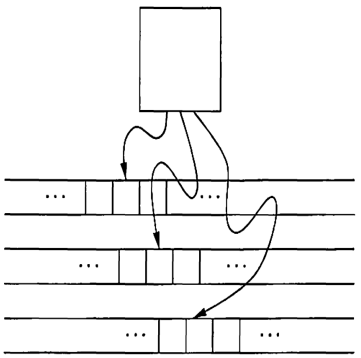

Figure 2. A multitape Turing machine

A language is called *decidable*, if there is a machine that recognizes it; is called *semidecidabile* (or computably enumerable) if there is a machine that accepts it. We say the *partial function* $f : \Sigma^* \longrightarrow \Sigma^*$ is *computable*, if there exists a machine $\mathcal{M}$ such that if $\sigma \in \Sigma^*$ and $f(\sigma) = y$, then the machine halts on σ and the final configuration has the form $\varepsilon q_f y$, where q_f is a final state and ε is the empty string and the machine does not halt on any string not in the domain of f.

Alternative definitions of Turing machines abound, all having the same power. For example, a *multitape* Turing machine is like an ordinary Turing machine, but with several tapes. Each tape has its own head for reading and writing. Every multitape Turing machine has an equivalent single-tape Turing machine (see Ausiello, Gambosi and d'Amore (2002) 181-92). A multitape machine (see fig.2) working in time $t(n)$ has an equivalent one-tape machine working in time $O(t(n)^2)$.

Definition 1. *With writing $f(n) \in O(g(n))$ we mean that there are n_0, c such that for all $n \geq n_0, f(n) \leq c \cdot g(n)$ (f is asymptotically bounded by g). If the other way round does not hold, we say that g grows faster than f. The set $O(g(n))$ is the order of growth of g.*

The Turing machine model perhaps may seem unrealistic. In 1971, for example, Hartmanis, Cook and Reckhow defined the model of random-access register machine (RAM model), for the purpose of introducing a theoretical model much closer in spirit to the design of Von Neumann architecture of modern computers. Actually all *reasonable* models define the same set of computable functions. We will show that also deterministic and *indeterministic* machines are equivalent *about what they can calculate*, but in this case *not for the amount of resources used.* However, if we consider *deterministic* machines, it is widely believed that all effective classical deterministic "sequential" (i.e. the instructions are executed in a sequence) models are polynomially- related with regard to the resources required to compute:

> *Invariance Thesis*. There exists a standard class of (deterministic) machine models, which includes among others all variants of Turing Machines and all variants of RAM machine models that simulate each other with Polynomially bounded overhead in time, and constant factor overhead in space (see Van Emde Boas (1990)).

More precisely, let us define the worst-case time complexity $t_M(n)$ of a machine (see 6 for the definition), which essentially means that for any w of length n the machine M take at most $t_M(n)$ steps to halt and return an output, and the worst space complexity $s_M(n)$, which means that for any w of length n the machine M needs to visit at most $s_M(n)$ cells to halt and return an output. In particular a machine $\mathcal{M}_1$ *run faster* than a machine $\mathcal{M}_2$, if $t_{\mathcal{M}_1}(n) \in O(t_{\mathcal{M}_2}(n))$ but not conversely.

A model of computation $\mathcal{C}$ simulates a model of computation $\mathcal{D}$ with *time overhead* $\alpha_t(x)$ and *space overhead* $\alpha_s(x)$, if for every machine $M_i \in \mathcal{D}$ there exists a machine $M_{i^*} \in \mathcal{C}$ such that

M_{i^*} simulates M_i and when a function $f(x)$ is computed by M_i in time $t(n)$ and space $s(n)$ then M_{i^*} requires $\alpha_t(t(n))$ time and $\alpha_s(s(n))$ space for computing it. It can be shown, for instance, that there exists a simulation of the *RAM* model by the Turing machine model with cubic time overhead and constant space overhead: $\alpha_t(t(n)) \in O(t(n)^3)$ and $\alpha_s(s(n)) \in O(s(n))$ (Slot and Emde Boas 1988). Hence these models can be simulated each other with a limited (polynomial) use of space and time. *Without taking into account the cost of mutual simulation*, many other models of computation are equivalent in power to Turing Machines. We deepen here the case of *non-deterministic* machines. Although simulable, the non-deterministic model is not simulable *efficiently*. Other models, as Randomized computation, Parallel computation, Analog computers, DNA computers, Quantum computers are excluded as well.

Deterministic Turing machines cannot include two or more quintuples (or quadruples) with the same first two elements: now let's drop this restriction. In his seminal 1936 paper, Turing also defined an extension of his "automatic machines" that he called "choice machines", which are now more commonly known as *nondeterministic Turing machines*. The execution of a nondeterministic Turing machine is not determined entirely by its input and its transition function; rather, at each step of its execution, the machine can (so to speak) *choice* the next move. Hence, a nondeterministic Turing machine allows for the possibility of *more than one next move from a given configuration*. In other words the *non-deterministic* machines are obtained by dropping the requirement of *coherence*. A non-deterministic algorithm can be interpreted as an algorithm that has the additional ability to *guess* a continuation in the scope of a finite set of possible continuations. This is the *nondeterministic* move. Hence every configuration may yield *more than one* next configuration. In contrast to the deterministic Turing machines, for which a computation is a sequence of configurations, a computation of a nondeterministic TM is a *tree of configurations* that can be reached from the initial configuration. The *root* of the tree is the initial configuration of the machine. The *children* of a node are the configurations that can follow in one move. The highest number of quintuples of the machine having the same first two elements is called the *degree of nondeterminism*. A node is a *leaf* if there is no next move. A *path* from the root to a leaf is an *accepting* computation if and only if the leaf is an *accepting configuration*. A *deterministic* computation corresponds to a particular sequence of choices. If *some* branch is an accepting computation (i.e. leads to a configuration of the form xqy with $q \in F$), then the machine accepts its input. It *rejects* its input, if *all branches* (i.e. for *every* choice of steps) are rejecting computations (i.e lead to a final configuration xqy with $q \notin F$).

A frequent and more intuitive presentation of the "physical model" of these machines, is indeed the one that adds the unit of reading and writing of the deterministic machine, a peripherial called *guessing-module* having it own write-only head. Suppose that the tape cells are numbered with integers $... -3, -2, -1, 0, 1, 2, 3...$ at the beginning the input is written starting from cell 1 to the right, the read-write head is scanning cell 1, the write-only head is scanning cell -1, and the finite state control is inactive. The *guessing module* writes, from right to left, a symbol at a time starting from the cell -1, then halts (*guessing stage*, actually the process may or may not terminate). At this point the control unit enters into play (*checking stage*) and the guessed string can be examined during the (purely deterministic) checking stage. The machine adopts the initial state q_0 and works as ordinary machine with the combination of the "guessed word" and the original input word, now as its input. During this second stage the guessing module and its guessing head are passive. An alternative way to consider a non-deterministic machines is therefore to say that a machine has the task of *guessing* the right solution to a problem.

The intuitive idea behind this approach to non-deterministic model is to provide a *solution-verifier* model, rather than a *problem-solver*: given a *guessed* solution to an instance of the problem, the algorithm can deterministically *verify* that it is a correct solution to the problem. Formally, a *verifier* for a language L is a *deterministic* Turing machine $\mathcal{V}$ such that $L = \{\sigma | \exists c(\mathcal{V} \text{ accepts } \langle \sigma, c \rangle)\}$. In other words, a verifier for L is a deterministic Turing machine that takes in an input x and an *evidence* c, and checks if c witnesses that x is in L. The additional information provided by c is also called *proof* or *certificate*. A nondeterministic computation can be therefore considered as divided into two phases: guess+verify. The *non-deterministic* moves guess a solution; then there is a *deterministic* subroutine which checks if there is a computation that accepts input. We call

verifier the above model,

Theorem 4. *The following are quivalent:*

1. *L is recursively enumerable (i.e. accepted by a deterministic Turing machine).*
2. *L is accepted by a non-deterministic Turing machine.*
3. *there exists a verifier for L.*

Proof. $1 \Rightarrow 2$ because a deterministic machine is a fortiori an indeterministic one. $2 \Rightarrow 3$: let us consider an accepting sequence of configurations of a nondeterministic machine; we encode this sequence and let c be a code of it. Let V be the verifier that uses c as a certificate. It imitates the nondeterministic machine by consulting c to know the next move to do. Clearly there exists c such that V accepts $\langle x, c\rangle$ if there exists a computation of the nondeterministic machine that accepts x. $3 \Rightarrow 1$: let V be a verifier for a language $L \subseteq \Sigma^*$ and let us consider the algorithm A that, on input $x \in \Sigma^*$, for each $k = 1, 2, 3...$ simulates V on the pair $\langle x, c\rangle$ for all $c \in \Sigma^*$ of length less or equal to k, for k steps and accepts if V accepts within this time. Now, if if the algorithm A on input x accepts, then this means that V itself accepts $\langle x, c\rangle$ and therefore $x \in L$. On the other hand, if $x \in L$, then V, being a verifier for L, accepts $\langle x, c\rangle$ for some c within t steps, for some t. Then the above program A accepts x at stage $k = max\{|c|, t\}$. Hence L is the language accepted by a deterministic algorithm A. QED

Actually we can simulate the nondeterministic model by the deterministic model, but *with an exponential cost.* If this loss of efficiency is inherent in some way, or due to the limits of our current knowledge, is a very important and unresolved issue. Essentially, it is the famous problem $\mathsf{P} = \mathsf{NP}$ we will discuss below.

Remark 1. *We remark that complexity classes that we will see, as* P *and* NP*, are defined with reference to the model Turing machines. However this does not imply a restriction, as the the most important complexity classes are invariant with respect to the computation model considered.*

We now show how to simulate a non deterministic computation with a deterministic machine. Let d be the degree of indeterminism of the machine. If for instance $d = 3$, then a computation is a subtree of the complete ternary tree as the following:

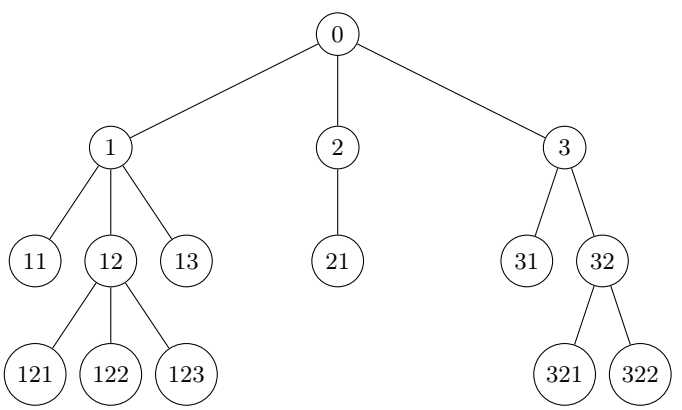

Figure 3. A nondeterministic computation

Each node is labelled with a string denoting a branch of nondeterministic computation. To prove the equivalence between nondeterministic and deterministic machine, we use a deterministic machine with three tapes. Suppose that in the nondeterministic machine each node in the computation tree has at most d children: in *tape 1* of the deterministic machine we copy the initial configuration of the nondeterministic machine, in *tape 2* we simulate the computation tree, typing in lexicographic order all finite strings of numbers from 1 to d that code nodes of the complete

d-ary tree, recalling that the computation tree is actually a *subtree* of this tree, and in *tape 3* we perform the computation. The algorithm works in phases, one phase for each string of numbers $j_1, ..., j_k$:

1. generates $j_1, ..., j_k$ on the second tape.
2. Copy the input from the first to the third tape,
3. Set $n = 1$ and let q the actual state of the nondeterministic machine and a the symbol scanned on tape 3:
   (a) if $\delta(q, a) = \emptyset$ or the number of elements of the set $\delta(q, a)$ is less than j_n, STOP (negative outcome).
   (b) Otherwise, take the j_n--th triple and apply the transition to the third tape.

Now, if there is a path of length k in the nondeterministic computation leading to a final configuration, that is to say a configuration $...xqy...$ where $q \in F$, then there is surely a phase of the simulation associated with a sequence of length k on the second tape that identifies this path and the simulation of this sequence will lead to a final state. If, on the other hand, such a path does not exist, then the associated deterministic machine can never reach a final state. But if the nondeterministic machine accepts a string in k steps then the deterministic machine requires $\leq k \cdot d^k$ steps. Actually, if there is a branch of the nondeterministic machine that accept in k steps, and d is the degree of indeterminacy, then the deterministic machine has to process at most

$$\sum_{j=0}^{k} d^j = 1 + d + d^2 + ... + d^k = \frac{d^{k+1} - 1}{d - 1} \leq k \cdot d^k$$

strings for simulating the computation.

Universality. A key property of Turing's model is universality: *there is a machine capable of simulating any other machine*. The possibility of a "universal" machine, was once considerd counterintuitive, because the parameters of the universal machine are fixed (alphabet size, number of states, and number of tapes), while the corresponding parameters for the machine being simulated could be much larger. The method by which the problem has been overcome is that of *coding*. We start here by considering a very simple deterministic one-tape kind of machines, with input alphabet $\{0, 1\}$ and tape alphabet $\{0, 1, *\}$. Since each machine $\mathcal{M}$ with a finite alphabet can be simulated by a machine $\mathcal{M}$ with tape alphabet $\{0, 1, *\}$, nothing is lost if therefore we consider, in our simple example, only machines whose input language is a binary language (see e.g. Du, Ko (2014) pp. 25-32 or Hopcroft, Motwani and Ullman (2000) pp. 377-81) and this argument may be extended to any fixed alphabet to get a binary encoding of a $k - tapes$ machine over this alphabet and also to nondeterministic machines. As we will see in the following chapters, the method of assigning a unique, unambiguously decodable numerical code to each symbol of a formal language, assigning natural numbers to symbols and then combining them into a single number to encode a string of symbols, was devised by Kurt Gödel in view of his incompleteness theorems, and has had wide applications. In our case, we can assign a numerical code to each state, tape symbol and direction and simply write each quintuple or quadruple as a sequence of five or four codes separated by some symbol between one code and another and between one string representing an instruction and another. Even better, identifying binary strings with natural numbers, such encoding can actually be done by means of binary strings[4]. Hence *Turing machines can ultimately be coded as binary strings*. Let therefore $\mathcal{M} =< \Pi, \Sigma, Q, q_0, \delta, \{q_f\} >$, where $\Pi = \{0, 1, *\}$, $Q = \{q_0, ..., q_{k+1}\}$, $q_f = q_{k+1}$. Let us consider this encoding:

$$\ulcorner 0 \urcorner = 0, \ulcorner 1 \urcorner = 00, \ulcorner * \urcorner = 000, \ulcorner q_i \urcorner = 0^{i+1}, \ulcorner L \urcorner = 0, \ulcorner R \urcorner = 00$$

A quintuple $q_i 01 R q_k$ will be coded by the binary string $0^{i+1}1010010010^{k+1}$. If the instructions of $\mathcal{M}$ are $\alpha_0, ..., \alpha_n$, then, if $\ulcorner \alpha_j \urcorner$ is the code of α_j, we set

$$\ulcorner \mathcal{M} \urcorner = 111 \ulcorner \alpha_0 \urcorner 11 \ulcorner \alpha_1 \urcorner 11 ... 11 \ulcorner \alpha_n \urcorner 111$$

[4] We identify a binary string σ with the natural number n s.t. the base-two representation of $n + 1$ is 1σ.

Each string encodes at most one machine. However, note that there can be many encodings of this kind for the same Turing machine: actually, since different orderings of the istructions give different codes, there are $m!$ equivalent codes for a machine of m istructions. We have coded all Turing machines in binary, but not all finite binary strings have the shape of a code. However, we can associate a Turing machine to *every* binary string, simply by associating the *trivial machine that rejects all inputs*, to the binary strings not having the shape of a code. In this way we can enumerate all Turing machines according to the lexicographic order of their index $\mathcal{M}_\varepsilon, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_{00}, \mathcal{M}_{01}, \mathcal{M}_{000}$.... We have an *enumeration* of the one tapes machine, i.e. that every binary string codes a Turing machine and each machine is coded at least by one string.

We describe now a three-tapes machine $\mathcal{U}$ that simulates $\mathcal{M}$, where the first tape[5] contains the input code $\ulcorner\mathcal{M}\urcorner^\frown\sigma$, the second tape is used to simulate the tape of $\mathcal{M}$ and is initialized to contain the binary string σ and the third tape contains the current state code, say 0^i of $\mathcal{M}$. The machine $\mathcal{U}$ works as follows:

1. preliminary check the first tape content, to see e.g. if it has a prefix of the form of a legal code of a deterministic machine. If no, STOP, reject. Otherwise:
2. *Initialize the first, the second and the third tapes.* Copy σ in the second tape $\ulcorner q_0\urcorner$ in the third tape. The first tape head is positioned on the first symbol of $\ulcorner\mathcal{M}\urcorner$, that of the second tape on the first symbol of σ and the third on the $\ulcorner q_0\urcorner$'s leftmost symbol as well.
3. If the third tape contains $\ulcorner q_f\urcorner$, STOP and accept. Otherwise:
   (a) if the symbol currently scanned in second tape is for example 1 and the content of the third tape is $\ulcorner q_i\urcorner$, then the first tape head scans $\ulcorner\mathcal{M}\urcorner^\frown\sigma$ starting from the left up to the second block 111, searching for a substring starting with $110^{i+1}1001$; if it is not found STOP, reject. Otherwise:
   (b) let e.g. that substring be $110^{i+1}1001010010^{k+1}$: then write $\ulcorner q_k\urcorner$ in the third tape, replace the content of the second tape with $\ulcorner 0\urcorner$, move this head to the right and go back to step 3.

It is immediate to verify that $\mathcal{M}$ accepts σ, then $\mathcal{U}$ accepts $\ulcorner\mathcal{M}\urcorner^\frown\sigma$. If $\mathcal{M}$ does not converge on σ (i.e. goes on forever), then $\mathcal{U}$ does not converge on $\ulcorner\mathcal{M}\urcorner\sigma$. Lastly, if $\mathcal{M}$ converges on σ but does not accept, the same does $\mathcal{U}$ on $\ulcorner\mathcal{M}\urcorner^\frown\sigma$.

Halting problem. It would be useful to have an algorithm able to decide, given a program and given a particular input, if the program terminates on that input. We have seen that an algorithm $\mathcal{U}$ exists that halts (accepting or rejecting) on $\ulcorner\mathcal{M}\urcorner^\frown\sigma$, when $\mathcal{M}$ halts (accepting or rejecting) on σ. But $\mathcal{M}$ could also loop: is there an algorithm U that also *halts* and *reject* $\ulcorner\mathcal{M}\urcorner^\frown\sigma$, when $\mathcal{M}$ *loops* on σ? Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist (it also follows the unsolvability of *Enscheidungsproblem.* Many problems indeed are proved to be unsolvable *by reducing* to them the *Halting Problem*). Indeed, suppose that there is a machine V that decides the set:

$$HALT = \{\ulcorner\mathcal{M}\urcorner^\frown\sigma | \mathcal{M} \textit{ halts on } \sigma\}$$

i.e. that on input $\ulcorner\mathcal{M}\urcorner^\frown\sigma$, halts and *accepts*, if $\mathcal{M}$ halts on σ, or halts and *rejects*, if $\mathcal{M}$ does not halt (loops) on σ.

Build another machine D such that on input σ write $\sigma^\frown\sigma$ on its tape and runs V on $\ulcorner\mathcal{M}_\sigma\urcorner^\frown\sigma$, and accepts, if V rejects, or loops, if V accepts. Hence:

$$D \text{ halts on } \sigma \text{ iff } V \text{ rejects } \sigma^\frown\sigma \text{ iff } \mathcal{M}_\sigma \text{ loops on } \sigma$$

Hence D behaves differently from every machine on at least one input. However, being in the above enumeration, D is in turn some $\mathcal{M}_{\sigma^*}$, so that $D(\sigma^*) = \mathcal{M}_{\sigma^*}(\sigma^*)$ halts iff $\mathcal{M}_{\sigma^*}(\sigma^*)$ does not halt (contradiction). Hence, the machine V cannot exist.

1.4. Refinements of the Church-Turing thesis

In 1934, Church stated an early version of the "Church thesis":

[5] We use here the symbol $^\frown$ for concatenation of strings.

> A function is effectively calculable if and only if it is λ-definable.

Influenced by Gödel's lectures on recursive functions, Church later reformulated his thesis, replacing the $\lambda-$definable functions with general recursive functions. Actually Church proposes his thesis as a "correct *definition* of effective computability". On this line were also Turing and Gödel, that is to provide a *correct definition* of effective computability. In this regard Post says that Church initially proposed the following:

> A function of positive integers is effectively computable only if recursive.

as a "working hypothesis", but criticize Church for having later transformed it in a *definition*, because "to mask this identification under a definition ... blinds us to the need of its continual verification" (Post (1936), p. 105). Only Kleene in 1943 started to use the expression "Church thesis", that is, as a hypothesis that needs further evidence to ascertain the truth:

> Thesis I. Every computable function (effectively decidable predicate) is general recursive.

In Kleene (1967), p. 232 he summarized:

> ...this claim we call Turing's thesis. It was shortly shown by Turing 1937 that his computable functions are the same as the λ-definable functions, and hence the same as the general recursive functions. So Turing's and Church's theses are equivalent. We shall usually refer to them both as Church's thesis, or in connection with that one of its three versions which deals with "Turing machines" as the Church-Turing thesis.

As for the use of the thesis of Church-Turing, a distinction can be made (see Odifreddi (1989-1999) vol. I, p. 103) between:

1. *essential use in metamathematics*: for example, if I show that a function is not recursive, then the Church-Turing thesis says it is not computable by any means. To demonstrate the unsolvability of a problem, I can therefore translate into a function, demonstrating that this is not recursive (from the recursive unsolvability is deduced the unsolvability in an absolute sense).

2. *Non essential use*. For example when, as a shortcut, we give an algorithm and then we resort to the thesis of Church-Turing to say that it corresponds to a recursive function ("proof by Church thesis"). Instead of producing a rigorous recursive definition, we prefer an informal argument, when the possibility of its formalization is evident.

Although this thesis is highly corroborated today, having withstood numerous attacks, it is worth seeing the development of its interpretation. Today, have been proposed some variants of Church's thesis. Kreisel (1971) drew a distinction between the Church's thesis and a stronger thesis:

> He called Church's *Superthesis* a stronger version of Church's Thesis, in which one not only claims that certain mathematical tasks are *equivalent* to recursive ones, but rather that each such *task* is *equal* to some program for an idealized computer: to each mechanical rule or algorithm is assigned a more or less specific programme, modulo trivial conversions, which can be seen to define the same computation process as the rule (Odifreddi (1996), p. 403).

In Kreisel's opinion, Turing established a version of the superthesis for the notion of mechanically computable function. Circumscribing the analysis to deterministic discrete mechanical devices, Gandy (1980) recommended to distinguish the thesis of Church and Turing from what he called 'Thesis M':

> *Thesis* M: Whatever can be calculated by a machine is Turing-machine-computable.

The Church-Turing thesis does not imply the thesis M. Turing in fact started from the analysis of a computation performed by a human being, but a thesis concerning procedures executable by a human has no implication concerning the extent of the procedures that machines are capable of carrying out "since, for example, there might be, among a machine's repertoire of atomic operations, operations that no human being who is working effectively is able to perform" (Copeland (2000)). To justify the thesis M, in order to specify what is meant by 'mechanically computable', Gandy considered only discrete machines with restrictions specified by a set of axioms (the most important are *local causation* and *unique assembly*) motivated on the basis of considerations drawn from special relativity and quantum mechanics.

> *Gandy's thesis.* Each computable function from a machine that satisfies the axioms of Gandy is Turing-computable.

A stronger version of the Church-Turing thesis states that any algorithmic process is simulated *efficiently* by a Turing machine. In the 1970s Robert Solovay and Volker Strassen worked out, however, a probabilistic primality test using an efficient algorithm, which was the first of its kind. They used *randomness* as an essential part of the algorithm. No efficient deterministic test for primality was known in that time, thus:

> it seemed as though computers with access to a random number generator would be able to efficiently perform computational tasks with no efficient solution on a conventional deterministic Turing machine (Nielsen, Chuang (2010) p. 6).

The probabilistic Turing machine began to look to many as the reference model. It is in particular provable that if it is possible to calculate a function with k elementary operations in a model of computation, then it is possible to calculate the function with a probabilistic machine by $p(k)$ elementary operations, for some polynomial $p(x)$. From which this modification of the strong version of the Church Turing thesis (see Nielsen, Chuang (2010) p. 140):

> *Strong Church–Turing thesis*: Any model of computation can be simulated on a probabilistic Turing machine with at most a polynomial increase in the number of elementary operations required.

In the early 80s of the 20th century Feynman highlights some essential difficulties in simulating quantum systems on classical computers and proposes as a program to develop computers based on the principles of quantum physics. In the 80's David Deutsch poses the question of whether a quantum computer can efficiently solve computational problems that have no efficient solution even with probabilistic Turing machines. In 1994 Peter Shor gave a significant response to this problem, creating a quantum algorithm that can factor a compound integer N in polynomial time in the number of digits $O(\log N)$ of N. Whereas the computations are also physical processes implemented by a physical system Deutsch poses the question of whether the laws of physics can be used to prove a version of the thesis of Church-Turing and proposed this variant (although very different from the original thesis, which speaks of effective methods, rather than finitely realisable physical systems):

> *CDT Thesis (Church, Turing, Deutsch).* Every finitely realisable physical system can be simulated by a universal model computing machine operating by finite means.

where Deutsch suggests that the reference computation model is, however, the Quantum Computer, but according to Nielsen, Chuang (2010) it is not clear whether Deutsch's notion of a *Universal Quantum Computer* is sufficient to efficiently simulate an arbitrary physical system. In the following chapters, we will be mainly interested in this thesis, known as *Edmonds-Cobham-Cook-Karp thesis*:

> A function is feasibly computable if and only if it is computed by some deterministic machine in polynomial time. A problem is feasibly decidable in case it is decidable by a deterministic Turing machine using a polynomial amount of computation time.

The seminal work Hartmanis and Stearns (1965) provided for the first time a precise definition of the time and space complexity of an algorithm, defining the concept of a complexity class. This statement is similar in form to Church's Thesis[6], however, it must be emphasised that effective computability it is not practical or feasible computability.

1.5. Turing on incomputability and the undecidability

In these two sections, we wish to establish a comparison between the proof of the *undecidability* theorem given in Turing (1936) on the one hand, and Cook (1971) and Levin (1973) independently proved *unfeasibility* theorem, on the other hand, as a paradigmatic example of the transition between two phases in the development of logic and at the same time emphasise a *trait-d'union* in the idea of logical formalisation of the computation of a Turing machine, taking a cue from a famous letter from Gödel to Von Neumann, in which research into feasibility is presented as a natural continuation of research into decidability, after the negative results of Church and Turing. In order to do this, without loss of generality, in both cases we adopt the simplest deterministic machine model, i.e. the one-tape, in which instructions are given *à la Post* by quadruples following Davis, Sigal and Weyuker (1994) pp. 451-57. For other proposals see e.g. Sipser (2006) pp. 276-82 or, for a more detailed version of it Ausiello, Gambosi and d'Amore (2002) pp. 329-35 and with regard to the Turing theorem, the Open Logic Project (2015) pp. 501-09. To show that the decision problem (*Entscheidungsproblem*) is unsolvable, Turing applied its famous result on the halting problem in this form:

> *Unsolvability of the halting problem.* There is no algorithm that, given a description of an arbitrary program Φ that computes a partial numerical function and a number n, decides whether Φ with input n halts in finitely many steps.

This result has a wide range of applications and we will show that many problems are algorithmically unsolvable by "coding" the halting problem into these problems. A problem A is reduced to a problem B, when a solution to B can be used to solve A; we prove that the solution to the *Entscheidungsproblem* could be used to find a solution for the halting problem (that is unsolvable!).

Strategy: given a Turing machine and a natural number n we want to build a set of first-order formulas Γ and a first-order formula ψ such that $\Gamma \vdash \psi$ if and only if the machine halts on input n. If it were possible to decide whether $\Gamma \vdash \psi$, we could also decide the halting problem. Since the latter is unsolvable, then there are Γ and ψ for which we are not able to decide whether ψ follows from Γ.

Preliminary will be an approproate description of the machine:

1. consider the boxes of the tape as numbered with integers:

$$... - 2, -1, 0, 1, 2, ...$$

2. instants of discrete time will be denoted by natural numbers: at 0 instant, the machine scans the square 0.

3. the alphabet of the machine will have symbols $S_0, S_1, ..., S_r$, where S_0 is the blank.

4. The predicative language chosen to describe the machine will contain predicative symbols $Q_i(t, x) =$" at time t, in the state q_i, it is scanned the square x"; $S_j(t, y); =$" at the instant t, the contents of the square y is the symbol S_j". In particular there will be the successor function symbol $S(x) = x + 1$, the constant $\overline{0}$ and the relational symbols $=$ and $<$ and relative axioms. In particular axioms for $<$ say that this relation is transitive, antireflexive and that S is total, injective and that $x < S(x)$.

[6] Beware that in some texts the Edmonds-Cobham thesis means another statement, namely: *the time complexities in any two reasonable and general models are polynomially related*, and it is also known as *extended Church-Turing thesis*.

We denote from $\overline{n}$ the numeral $\overbrace{S(S(S(...(S(0))...)))}^{n-times}$. Since we dont have subraction, the negative integers may be formalized as follows:

$$Q_i(t,-p) \leftrightarrow \exists z(Q_i(t,z) \wedge \overbrace{S(S(S...S(z)...)))}^{p-times} = 0)$$

Representation of the initial conditions. in the initial state q_0 the machine observes a sequence of n symbols S_1 (so we represent the input n); the rest of the tape is white:

$$Q_0(\overline{0},\overline{0}) \wedge \bigwedge_{0 \leq i < n} S_1(\overline{0},\overline{i}) \wedge \forall y(\overline{n} < y \rightarrow S_0(\overline{0},y))$$

Representation of the final conditions.

I want a formula ψ such that ψ is true in the moment in which the machine halts. Note that the machine halts if there is a time t in which the machine, lying in state q_i, reads S_j, but no instruction begins with q_iS_j. Since both the number of states and symbols is finite, we can form the disjunction:

$$\bigvee_{i,j} \exists t \exists x(Q_i(t,x) \wedge S_j(t,x))$$

of the q_i, S_j such that no instruction starts with q_iS_j. This will be our ψ.

Representation of instructions.

The instruction $q_iS_jS_kq_m$ is formalized as follows:

$$\forall t \forall x(Q_i(t,x) \wedge S_j(t,x) \rightarrow (Q_m(t+1,x) \wedge S_k(t+1,x) \wedge$$

$$\wedge \forall y(y \neq x \rightarrow \bigwedge_{i \leq r}(S_i(t,y) \rightarrow S_i(t+1,y))$$

The instruction $q_iS_jRq_m$ is formalized as follows:

$$\forall t \forall x(Q_i(t,x) \wedge S_j(t,x) \rightarrow (Q_m(t+1,x+1) \wedge \forall y \bigwedge_{i \leq r}(S_i(t,y) \rightarrow S_i(t+1,y))$$

We leave it as an exercise to the reader to formalise the instruction $q_iS_jLq_m$. The set Γ will contain all these axioms relative to the instructions and to the initial conditions, moreover the axioms for successor, and preorder and identity. Note that according to the intended interpretation $\Gamma \vdash \psi$ implies that the machine halts on input n. The reverse will be demonstrated by induction. It will follow the equivalence between (a) $\Gamma \vdash \psi$ and (b) " The machine halts on input n ", but being (b) undecidable, it will also undecidable (a). First of all we represent the state of the machine at instant s as the conjunction of these three formulas:

1. $Q_i(\overline{s},\overline{p})$
2. $S_{j_0}(\overline{s},\overline{p_0}) \wedge ... \wedge S_j(\overline{s},\overline{p}) \wedge ... \wedge S_{j_v}(\overline{s},\overline{p_v})$
3. $\forall y((y \neq \overline{p_0} \wedge ... \wedge y \neq \overline{p} \wedge ...y \neq \overline{p_v}) \rightarrow S_0(\overline{s},y))$

Suppose the machine halts at instant s reading S_j in square p and lying in state q_i; the description of this situation is $Q_i(\overline{s},\overline{p}) \wedge S_j(\overline{s},\overline{p})$, that implies ψ. In essence, to demonstrate that from Γ follows ψ is enough therefore to show that for every $s \geq 0$, if the machine does not halts before s, then from Γ follows a description of the machine at stage s.

Induction If $s = 0$, then it is obvious, since Γ contains the initial state of the machine. For the inductive step, suppose that at s the machine has not stopped; for (IH) we are able to derive from Γ a description of the machine at instant s. Suppose, then, that at s, the machine scans the square p containing the symbol S_j, and is in the state q_i. Because the machine does not stop, the possible continuations will be as follows:

1. if the instruction is $q_iS_jS_kq_m$, then Γ will contain the formula:

$$\forall t \forall x(Q_i(t,x) \wedge S_j(t,x) \rightarrow (Q_m(t+1,x) \wedge S_k(t+1,x) \wedge$$

$$\wedge \forall y(y \neq x \rightarrow \bigwedge_{i \leq r}(S_i(t,y) \rightarrow S_i(t+1,y))$$

But this, together with the inductive hypothesis, that is, that at s the machine is described by 1., 2., 3., and the axioms for $<, =$ and successor, implies:

(a) $Q_m(\overline{s+1}, \overline{p}) \wedge S_{j_0}(\overline{s+1}, \overline{p_0}) \wedge ... \wedge S_k(\overline{s+1}, \overline{p}) \wedge ... \wedge S_{j_v}(\overline{s+1}, \overline{p_v})$

(b) $\forall y(y \neq \overline{p_0} \wedge ...y \neq \overline{p} \wedge ...y \neq \overline{p_v}) \rightarrow S_0(\overline{s+1}, y)$

That is, a description of the machine at stage $s+1$.

2. If the instruction is $q_iS_jRq_m$, then Γ will contain the axiom:

$$\forall t \forall x(Q_i(t,x) \wedge S_j(t,x) \rightarrow (Q_m(t+1,x+1) \wedge \forall y \bigwedge_{i \leq r}(S_i(t,y) \rightarrow S_i(t+1,y))$$

similarly to the previous case, we obtain:

(a) $Q_m(\overline{s+1}, \overline{p+1}) \wedge S_{j_0}(\overline{s+1}, \overline{p_0}) \wedge ... \wedge S_k(\overline{s+1}, \overline{p+1}) \wedge ... \wedge S_{j_v}(\overline{s+1}, \overline{p_v})$

(b) $\forall y(y \neq \overline{p_0} \wedge ...y \neq \overline{p+1} \wedge ...y \neq \overline{p_v}) \rightarrow S_0(\overline{s+1}, y)$

That is, a description of the machine at $s+1$.

3. If the instruction is $q_iS_jLq_m$, we leave this case as an exercise for the reader.

In any case, Γ implies a description of the machine at $s+1$. Thereby ends the inductive step. We conclude by induction that for all non-negative number s, if the machine does not stop before s, then Γ implies a description of it at instant s. Therefore, if the machine halts at s, then its description will contain the expression $Q_i(s,p) \wedge S_j(s,p)$, that implies ψ. That is, if the machine halts on input n, then from Γ follows ψ.

1.6. Cook and Levin's theorem and the unfeasibility

Computability in polynomial time has become synonymous with feasibly decidability. Church's Thesis concerns the concept of effective computability. Analogously, the thesis put forward by Cobham, Edmonds and Cook concerns the feasible computations, by identifying efficient computablility with *computable in polynomial time*. Note that, like the Church–Turing thesis, this is not a theorem. For the continuation of the discussion on this topic, it is necessary to specify the notion of computational efficiency and the complexity measures we will adopt to quantify it. We consider one-tape machines and we define:

1. the *running time* of a deterministic machine $\mathcal{M}$ on an input x is the number of steps $time_M(x)$ made by the machine until it halts (note that it can also be infinite[7]).
2. The *working space* of a deterministic machine $\mathcal{M}$ on an input x is the number of cells $space_M(x)$ visited at least once by the head during the computation (note that it can also be infinite and that can be finite although the machine runs forever).
3. the *time complexity* of the machine is:

$$t_M(n) = max\{time_M(x) || x| = n\}$$

i.e. the maximum number of steps that M makes for inputs of lengths n, before halting.

[7] A single step of computation is composed by the following actions: read the input symbol from the active cell, look up the transition rule associated with the current state and input symbol, overwrite the input symbol with the new symbol and change the current state according to the transition rule.

4. the *space complexity* of the machine is:

$$s_M(n) = max\{space_M(x)||x| = n\}$$

As for nondeterministic Turing machines, $time_M(x)$ and $space_M(x)$ are defined as the minima among the accepting paths. Moreover:

(a) $t_M(n) = max\{time_M(x)||x| = n \text{ and } M \text{ accepts } x\}$ and

(b) $s_M(n) = max\{space_M(x)||x| = n \text{ and } M \text{ accepts } x\}$

In case no x of length n is accepted by M, some convention is added to these measures. In case of time complexity, add $n + 1$, the time needed just for reading the input. In case of space, if no x of length n is accepted by M, put $s_M(n) = 1$. If no accepting path exist, these functions are undefined (we follow Du, Ko (2014) pp. 19-21).

1. $\mathsf{DTIME}(f(n))$=set of languages decided by a deterministic Turing machine $\mathcal{M}$ in time $t_M(n) \leq f(n)$,
2. $\mathsf{NTIME}(f(n))$= set of languages accepted by a nondeterministic Turing machine in time $t_M(n) \leq f(n)$,
3. $\mathsf{DSPACE}(f(n))$=set of languages decided by a deterministic Turing machine $\mathcal{M}$ in space $s_M(n) \leq f(n)$,
4. $\mathsf{NSPACE}(f(n))$= set of languages accepted by a nondeterministic Turing machine in space $s_M(n) \leq f(n)$,

Theorem 5. *The following relations hold:*

$$\mathsf{DTIME}(f(n)) \subseteq \mathsf{NTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{c \cdot f(n)})$$

Proof. See e.g. Ausiello, Gambosi and d'Amore (2002), 300-03. QED

Now we define the complexity classes:

1. $\mathsf{P} = \cup_{k=0}^{\infty}\mathsf{DTIME}(n^k)$ (or $\mathsf{DTIME}(Poly)$) is the set of languages that can be decided by a deterministic Turing machine in polynomial time in the dimension of the input,
2. $\mathsf{NP} = \cup_{k=0}^{\infty}\mathsf{NTIME}(n^k)$ (or $\mathsf{NTIME}(Poly)$) is set of languages that can be accepted by a nondeterministic machine in polynomial time in t he dimension of the input.
3. (PSPACE and $\mathsf{NPSPACE}$ are defined analogously).

By definition, any language in P or NP is decidable. However some problems are *hard to solve*, but *easy to check*. We consider P the set of decision problems solvable quickly and NP the set of problems *verifiable* quickly: however, even today, *it is not known whether* $\mathsf{P} \neq \mathsf{NP}$.

Definition 2. *A total function f is said to be polynomial-time computable if there is a Turing machine $\mathcal{M}$ that computes f, and a polynomial $p(n)$, such that the number of steps in the computation by the machine with input x is bounded by $p(|x|)$.*

We say that Q is polynomial-time reducible to L (in symbols $Q \leq_p L$) if and only if there exists a function f computable in polynomial time such that for each x, $x \in Q$ iff $f(x) \in L$. Furthermore L is said $\mathsf{NP} - hard$ if for all $Q \in \mathsf{NP}$, $Q \leq_p L$. If also $L \in \mathsf{NP}$, then we say that L is NP-complete.

A popular NP-complete problem is the so-called *traveling salesman problem*. Remember that a Hamiltonian cycle is a cycle through a graph that visits all the nodes of the graph once and only once. Let us suppose we have n cities and their distances: we are asked to find the shortest tour that touches all cities exactly once (with a final return to the starting point). In another version, given a number B, the question is whether there is a tour that touches all cities at most once, shorter than or equal to B. In a more mathematical terms, given a complete, undirected and weighted graph (*complete* means that each pair of graph vertices is connected by an edge and *weighted* means that a weight is associated with each edge), it is asked to find a *Hamiltonian circuit* with the least weight (or, in the second version, less weight or equal to B). The problem is

solvable by brute force by considering $\sim n!$ possible permutations, but are not known algorithms that work in polynomial time. Little hope to find: the traveling salesman problem is NP -complete. On the other hand, however, it is obvious that the problem can be verified in polynomial time: if you give me a tour, I reckon its cost by adding up the cost of all their edges; then I check if it is less than or equal to B.

Intuitively, the NP-complete problems are the *most difficult* in the class NP. Clearly $\mathsf{P} \subseteq \mathsf{NP}$, but the problem $\mathsf{P} = \mathsf{NP}$ is still unsolved. However if there is a language L that is NP-complete and belong to P, then $\mathsf{P} = \mathsf{NP}$. Schematically, we prove this second statement by showing that, under the hypothesis that there exist a language L that is NP-complete and also belong to P, it follows that for any Q, if $Q \in \mathsf{NP}$, then $Q \in \mathsf{P}$.

It actually suffices that the number of steps be bounded by $c \cdot |x|^r$ for some r (see Davis, Sigal and Weyuker (1994), 441-43). Now, being L an NP-complete set, then if $Q \in \mathsf{NP}$, also we have $Q \leq_p L$, namely, $Q = \{x | f(x) \in L\}$ for some polynomial-time computable function $f(x)$. Still by hypothesis $L \in \mathsf{P}$. Then we show that $Q = \{x \in \Sigma^* | f(x) \in L\}$ is also in P. Let $\mathcal{M}$ be a machine that decides L in time (say) bounded by $c \cdot |x|^r$ (where $|x|$ denotes the length of x) and let $\mathcal{V}$ a machine that computes $f(x)$ in time bounded by $a \cdot |x|^s$. Let us consider the composition $\mathcal{W}$ of these machines: first run $\mathcal{V}$ on x to compute $f(x)$, then run $\mathcal{M}$ to decide whether $f(x) \in L$.

Note that the machine $\mathcal{V}$ cannot print more symbols that steps in the computation; hence $|f(x)| \leq |x| + a \cdot |x|^s$ and then W needs less than $b \cdot |x|^{rs}$ steps (for some b) to decide Q. It follows that $Q \in \mathsf{P}$.

Let us consider now the set SAT of *satisfable* propositional formulas (in conjunctive normal form). Although to analize the full truth-table has an exponential cost (in the number of propositional variables), it is not hard, thinking to the check+guess method, to realise that if somebody gives me a line of the truth table, I can rapidly check whether or not this line satisfies a formula. Actually $\mathsf{SAT} \in \mathsf{NP}$, because a nondeterministic algorithm that accept a formula ϕ can be defined, that works in time $a \cdot |\phi|^2$ *Cook's and Levin's theorem* (1971) says that actually SAT is NP-complete. From the previous result it follows therefore that

$$\mathsf{P} = \mathsf{NP}\ iff\ \mathsf{SAT} \in \mathsf{P}$$

Therefore, if someone found an efficient method to establish whether a formula of propositional calculus is satisfable, she would have solved this famous problem.

The first "formulation" of the problem if $\mathsf{P} = \mathsf{NP}$, a key issue in computational complexity, albeit implicitly, dates back to Kurt Gödel, in a letter to Von Neumann in 1956, although the importance of the problem raised was clear only with the work of Stephen Cook in the 1970s. In short, this is Gödel's argument: the unsolvability of the *Entscheidungsproblem* implies that the problem whether a formula ϕ of first order language has a proof is undecidable: but what, if we consider the problem whether ϕ has a proof of at most n symbols? This in principle is decidable, but at what cost? Actually Gödel considered that a proof of his conjecture would constitute a refinement of Hilbert's *Entscheidungsproblem*:

> Despite the unsolvability of the *Entscheidungsproblem*, the mental effort of the mathematician in the case of yes/no questions could be completely replaced by machines. One would indeed here to simply select an n so large that, if the machine yields no result, there would then also be no reason to think further about the problem. (Gödel's Collected Works (1986-2005) vol. V, 135).

Let us write $\vdash^n \phi$ for "the formula ϕ is provable in the predicate calculus with at most n *symbols*". Gödel asked what is the complexity of the set:

$$\mathsf{T} = \{\langle \phi, n \rangle | \vdash^n \phi\}$$

In particular, whether it is decidable by a Turing machine in time n^2. According to a widespread opinion, the polynomial-time computation is a good formalization of practical, realistic, feasible (by humans or computers) computability, a narrower requirement than the decidability in principle.

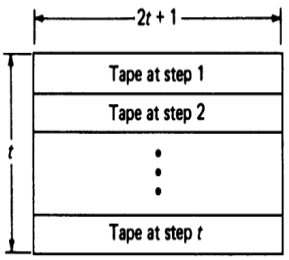

Figure 4. Let $t = p(|u|)$.

However Gödel was thinking of *linear* and *quadratic* time as corresponding to *feasibility*. Note that if our language contains k-symbols the problem is solvable brutally making a list of all proofs with at most n symbols, then see which is right for us; but this procedure has obviously a complexity of order k^n. T is in NP since a nondeterministic algorithm for verifying it consists of guessing a proof of $\leq n$ symbols and then verifying in polynomial time if the guessed proof is valid (see Buss (1995)). Today we know that the problem is NP-complete: hence if the hypothesis of Gödel was right, therefore, it would also be in P and as a result we would have $\mathsf{P} = \mathsf{NP}$. Indeed, Cook showed that the satisfability problem SAT can be reduced to it: let un consider a boolean formula $\Phi(p_0, .., p_n)$ and replace each propositional variable p_i with $x_i = y$. Call $\Phi^*(x_0, ..., x_n, y)$ this new formula. It is provable that the boolean formula is satisfable if and only if the formula:

$$\exists y(\exists z(z \neq y) \rightarrow \exists x_0, ..., \exists x_n \Phi^*(x_0, ..., x_n, y))$$

is provable with a proof of length $|\Phi|^c$, for some constant c.

But what is the length of proofs? There are infinitely many proofs of fixed *number of lines*, but finitely many proofs with *fixed number of occurrences of symbols*; in the above argument we considered the total amount of occurrences of symbols in a proof (its *size*). For instance, a variable x_n can be considered as a symbol x followed by $n-$marks and therefore of size $n + 1$. If we adopt the *number of formulas* as a measure of the length of proofs, it is undecidable for sequent calculus (see Buss (1991)). In general, many results in the study of complexity of proofs are established only for the case of symbol-length and it is not known whether those results can be transferred to the case of step-length.

We now come to the proof of the result discovered independently by Cook and Levin. The time required for a precise proof that the problem SAT is in NP is $O(|x|^2)$. This is shown for instance in Davis, Sigal and Weyuker (1994), 448, which we also follow in the precise exposition of hardness.

Theorem 6. (Cook and Levin 1971) SAT *is* NP *hard.*

Proof. Let $L \in \mathsf{NP}$ and M be a nondeterministic Turing machine accepting L in time $p(|x|)$ on input $x \in L$ (for some polynomial $p(n)$). We show that there is a *polynomial time computable function* that translates each input u into a propositional formula in conjunctive normal form δ_u such that:

$$M \text{ accepts } u \text{ iff } \delta_u \in \mathsf{SAT}$$

This δ_u will be a simulation of the computation of M on input u. To simplify matter, let us assume that an accepting computation halts *exactly* after $P(|u|)$ steps. To do this, we allow repetitions of the same configuration $C \Rightarrow C$. After $p(|u|)$ steps the scanned squares are at most $p(|u|)$ (going left or right). In the following we consider therefore a matrix (or *array*) of $2 \cdot p(|u|) + 1 \times p(|u|)$ cells, sufficient to exibit all information needed (see fig.4).

The first row of this array represents the initial configuration of the machine, in state q_1, scanning the symbol blank immediately to the left of the first symbol of u:

$$\overbrace{*...q_1*}^{t+1-times} \; s_{u_1}....s_{u_z} \; \overbrace{*...*}^{t-|u|-times}$$

where $z = |u|$ and $u = s_1....s_z$ and $t = p(|u|)$. Let the set of states of L be $Q = \{q_1, ..., q_m\}$ where q_1 is the initial state and q_m an accepting state and let the set of tape symbols be $S = \{s_0, s_1, ..., s_r\}$, where $* = s_0$. We use a set of propositional atoms $p_{h,j,k}$ and $w_{i,j,k}$, where $1 \leq h \leq m$, $0 \leq i \leq r$, $1 \leq j \leq 2 \cdot p(|u|)$ and $1 \leq k \leq p(|u|)$ and we consider an assignement of values such that:

1. $p_{h,j,k}$ is true iff M is in state q_h scanning the jth cell at step k.
2. $w_{i,j,k}$ is true iff the symbol s_i is in the jth cell at step k.

Let us say that the length of each of these atoms is of the order of complexity[8] $O(t)$. Lastly, it will be useful this abbreviation:

$$\Theta\{x_i | 1 \leq i \leq e\} = \bigwedge_{1 \leq l < f \leq e} (\neg x_l \vee \neg x_f) \wedge \bigvee_{1 \leq l \leq e} x_l$$

which is true iff exctly one of $x_1, ..., x_e$ is true. Since there are $e(e-1)/2$ pairs (l, f) with $1 \leq l < f \leq e$ and the second conjunct contains e literals, the whole formula has $O(e^2 t)$ symbols.

We formalize now the computation (where for "length"we mean the number of symbols):

1. The machine in state q_1 scans s_0 immediately to the left of u (the first row of the array):

$$\bigwedge_j w_{0,j,1} \wedge \bigwedge_j w_{u_j, p(|u|)+j+1,1} \wedge \bigwedge w_{0,p(|u|)+|u|+j+1,1} \wedge p_{1,p(|u|)+1,1}$$

   The length of this formula is $O(t^2)$.

2. At each step k there is a unique state and a unique scanned symbol:

$$\Theta\{p_{h,j,k} | 1 \leq h \leq m, 1 \leq j \leq 2 \cdot p(|u|) + 1\}$$

   The length is $O(t^4)$.

3. Each entry on the array contains exactly one symbol:

$$\bigwedge_k \bigwedge_j \Theta\{w_{i,j,k} | 0 \leq i \leq r\}$$

   This formula is of length $O(t^3)$.

4. Each configuration after the first is obtained by applying the istructions. Let us assume that the program is made of the following istructions:
   (a) $q_{i_a} s_{j_a} s_{k_a} q_{e_a} \quad (a = 1, 2, 3...r_a)$
   (b) $q_{i_b} s_{j_b} R q_{e_b} \quad (b = 1, 2, 3...r_b)$
   (c) $q_{i_c} s_{j_c} L q_{e_c} \quad (c = 1, 2, 3...r_c)$

[8] This bound depends on how we encode the index sequences and therefore the variables. In later editions the authors modified it, but with an efficient (binary) encoding, the length of these variables can be kept of order $O(t)$. However, here this constitutes a minor point.

Let moreover:

$$NOTHEAD(j,k) = \bigvee_i (w_{i,j,k} \wedge w_{i,j,k+1}) \wedge \bigwedge_h \neg p_{h,j,k}$$

(M is not scanning the jth cell at the k step) and let:

$$IDENT(j,k) = \bigvee_h \bigvee_i (p_{h,j,k} \wedge p_{h,j,k+1} \wedge w_{i,j,k} \wedge w_{i,j,k+1})$$

(M scans the jth cell at kth and $k+1th$ steps; the state and the symbol scanned is the same in both these successive configurations). Moreover let:

$$\alpha(j,k) = \bigvee_a (p_{i_a,j,k} \wedge w_{j_a,j,k} \wedge w_{k_a,j,k+1} \wedge p_{e_a,j,k+1})$$

(The $k+1th$ step comes from the k step applying one istruction from the group (a)). Analogously define $\beta(j,k)$ and $\gamma(j,k)$ relative to the groups (b) and (c)). Thus (4.) is defined as:

$$\bigwedge_k \bigwedge_j (NOTHEAD(j,k) \vee IDENT(j,k) \vee \alpha(j,k) \vee \beta(j,k) \vee \gamma(j,k))$$

This formula has length $O(t^3)$.

5. The $p(|u|)$-th configuration is terminal (recall that q_m is an accepting state):

$$\bigvee_j p_{m,j,p(|u|)}$$

This formula has length $O(t^2)$.

Now take δ_u as the conjunction of (1.)-(5.) and note that if M accepts u, then $\delta_u \in \mathsf{SAT}$, because the above assignement of truth values to the propositional atoms makes δ_u true. Conversely, if there is an interpretation that makes δ_u true, then we can reconstruct our array: by (3.) for each j,k there is a unique i such that $w_{h,j,k}$ is true: from this we can *uniquely* reconstruct the array in figure. By (2.) there are unique q_h and j such that $p_{h,j,k}$ is true. Hence each row can be made into a configuration of the machine. By (1.), the configuration corresponding to the first row of the array is an initial configuration. By (4.), for each row of the array after the first, the corresponding configuration is identical to it or results from it using one of the quadruples of the program. Finally, by (5.) the entire sequence of configurations constitutes an accepting computation. Thus, u is accepted by M.

We now shown that there is a polynomial-time computable function that maps each string u onto the corresponding CNF formula δ_u. Observe that the CNF formulas of (2.)-(5.) do not depend on u: a Turing machine can be constructed to write these on a tape in a number of steps proportional to the length of the expression, which is $O(t^4)$, and hence polynomial in $|u|$. As for (1.) some of its atoms do not depend directly on u; producing this part simply involves writing $O(t^2)$ symbols. The remaining atoms correspond in a one-one manner to the symbols making up u and can be produced in a number of steps proportional to $|u|$. This completes the proof of the Cook-Levin theorem as presented in Davis, Sigal and Weyuker (1994). QED

The Cook-Levin theorem allows one to prove that many other important problems are NP-complete, by showing that SAT can be efficiently reduced to them. A proof that an NP problem is NP-complete is a proof that the problem is far from feasible, unless every NP problem is in P, which is not known, but which is unlikely and which, if proved, would constitute a shocking novelty for computer science.