Chapter 6

Proof complexity

Logic became important when mathematicians realized that mathematics cannot be founded only on our intuition about mathematical entities and that it was necessary to introduce axiomatic systems and formalize reasoning. Moreover, it turned out that we will never have a complete set of axioms for number theory and set theory. Therefore logic has an important role to play, both in studying what is provable from particular sets of axioms and, even more, in studying what is not provable. To this end two fields of logic have been developed: proof theory and model theory. The aim of proof theory is to study provability by syntactical means, namely to actually study proofs. In model theory, on the other hand, we use models to show that a sentence is not provable. (Recall that in order to show that a sentence is not provable from a set of axioms, it suffices to find a model that satisfies the axioms, but not the sentence.) Both fields have, of course, a much wider range of applications.

When we are not able to prove a theorem from a set of axioms, the reason may be not that it is not provable, but that the proof is too difficult to find. This is the problem that mathematicians have to cope with in their everyday work. To prove a theorem with a nontrivial proof requires ingenuity, experience, patience, and often also a little bit of luck. Yet sometimes all this may not be enough. It may happen that every proof of the theorem is so complex that it cannot even be written down. In some rare cases, such as the Four Color Theorem, a computer can help us, but what if the lengths of proofs exceed even the capacity of computers? This can indeed happen. For every sufficiently strong theory, one can construct short sentences whose proofs are so long that they cannot be physically represented in our universe. Although known examples of such theorems are rather contrived, there is no reason why it cannot also happen for interesting open problems.

Thus there is yet another role for logic: to study the complexity of proofs. The basic measure of the complexity of a proof is its length. We may be unable to find a proof for various reasons, but large length is an absolute obstacle, because an extremely long proof cannot ever be written down. Naturally, most of the research in proof complexity theory focuses on studying the lengths of proofs. However, there are other reasons for studying the lengths of proofs, the main one being the relations of proof complexity to computational complexity. The starting point of both theories is essentially the same. In computational complexity theory the classical concept of computability is replaced by feasible computability, usually represented by computability in polynomial time. Similarly, in proof complexity the concept of provability is replaced by feasible provability, where proofs of polynomial lengths are the standard representation of the imprecise concept of short proofs. There are several other analogies between these two theories, which strongly suggests that computational complexity and proof complexity are just two facets of the same concept.

Therefore the term *proof complexity* is now interpreted in a much broader sense than a few decades ago. It can be briefly described as the study of the phenomenon of complexity using the means of mathematical logic. In this field we study not only the lengths of proofs, but also calculi for propositional logic and theories that formalize reasoning with concepts of a given complexity.

6.1 Proof theory

— **A** —

We have already met proof theory in this book several times. In previous parts I wrote mainly about the formalization of first order logic, and Gödel's incompleteness theorem, which is the most important result in proof theory. In this section I will describe some other classical results of proof theory and some results about the lengths of proofs in first order logic.

Direct and indirect proofs

Imagine a proof as a path going from the assumptions of a theorem to the conclusion. The path can be straight or circuitous. In logic we call a proof that goes straight from the assumptions to the conclusion a *direct proof*; a roundabout proof is called *indirect.* Thinking of proofs in such a geometric way can, however, be rather misleading. It can happen that there is a short indirect proof, while all direct proofs are long.

So, what precisely does it mean when we say that a proof is direct? A direct proof is characterized by the property that it only uses concepts which occur in the theorem. We may also allow some simple combinations of these concepts, the precise definition of which depends on the particular logical calculus used. A typical direct proof is based on splitting the problem into several cases. An indirect proof, on the other hand, contains auxiliary theorems, which are usually called *lemmas.* Lemmas may be statements that formally have little to do with the theorem to be proven.

Direct proofs are as a rule easier to find. We need only to try combinations of concepts already contained in the statement of the theorem. Direct proofs are also easier to understand and thus they are more useful when we want to understand why a theorem is true, which is in most cases the reason why we study a proof. This leads us to an important question: *can every theorem be proven using a direct proof?*

The short answer to the above question is: *yes.* The more detailed answer is: *in principle yes, but in practice not always.* We will see that direct proofs may be much longer than indirect ones and that this may prevent us from writing down such proofs.

I will describe two results that show that any proof can be transformed into a direct proof. The first one is the important *Herbrand Theorem.* This theorem is due to Jacques Herbrand (1908-1931), a talented French mathematician who died very young in a mountaineering accident.

To avoid technicalities, I will explain only a special case of Herbrand's Theorem (see more about it in part C). I will consider statements of the form *'there exists an x which satisfies ϕ',* which we represent formally by

$$\exists x\ \phi(x).$$

where I am assuming that this is a purely existential formula, namely, that there are no quantifiers inside of ϕ. Herbrand's Theorem concerns provability in pure predicate calculus.[1] It says that $\exists x\ \phi(x)$ is provable if and only if for some terms $t_1, t_2, \ldots, t_n$, the disjunction

$$\phi(t_1) \vee \phi(t_2) \vee \ldots \vee \phi(t_n) \tag{6.1}$$

is provable. The latter formula says that there are n objects, specified by terms $t_1, t_2, \ldots, t_n$, such that ϕ holds for at least one of them. Notice that this can be interpreted as a proof by case distinction. There are n possible cases such that in case 1 we prove $\exists x\ \phi(x)$ by showing $\phi(t_1)$, in case 2 we prove $\exists x\ \phi(x)$ by showing $\phi(t_2)$, and so on.

Herbrand's Theorem moreover says that disjunction (6.1) is provable in propositional logic. This shows that we only need to use subformulas of ϕ and nothing else (more precisely, subformulas with various terms substituted in them). Thus we can construct a direct proof of $\exists x\ \phi(x)$. Furthermore, once we have such a disjunction, we can determine its validity by an algorithm, say, by trying all combinations of truth values.

[1] By the completeness theorem, logical validity and provability are the same things. As we are now interested in proofs, not just mere logical validity, I will use the latter term.

Herbrand's theorem has other important applications. In particular the fact that (in a sense) it reduces provability in first order logic to provability in propositional logic makes it attractive for automated theorem proving. It should be noted that Herbrand's Theorem does not give us any bound on the number of disjuncts, if we only know the formula. If we could bound the number of disjuncts and the complexity of the terms, we would get a decision procedure for first order logic, which is impossible, as we know. Moreover, such a Herbrand proof may be much longer than the usual one, as we will see in a moment. Yet it seems easier to base theorem proving programs on searching for terms and testing propositional validity, than on generating sequences of formulas representing proofs.

How to speed-up proofs

We have seen that the lengths of proofs may depend very much on the formal systems used. It may happen that for two formal systems some theorems have much shorter proofs in one than in the other. In proof theory this phenomenon is called *speed-up*. (The choice of the word is not quite justified, as we may actually find a long proof more quickly than a short one.) Thus there is speed-up between the sequent calculus without cuts and the sequent calculus with cuts; but there are more situations in which it occurs. I will consider the two most important cases.

The first one concerns an axiomatic theory and the extension of it obtained by adding a new axiom. Let S be a theory and T an extension of S by an axiom α not provable in S. (Such a theory T is usually denoted by $S+\alpha$.) I will, of course, assume that both theories are consistent. If we add an unprovable sentence to S, there will be an infinite number of other sentences in the extension that are unprovable in S. But we cannot talk about speeding-up unprovable sentences, as they have no proofs in S. We are concerned with the sentences that are provable in both theories. If a sentence ϕ is provable in both theories S and T, then it may have a shorter proof in T because we may also use the new axiom α.

Indeed, it is not very difficult to prove that under very mild assumptions about S, there is a an extremely large speed-up in T for any α unprovable in S. It suffices to assume that one can formalize a relatively weak fragment of arithmetic in S and then the speed-up is so large, that it cannot be bounded by any computable function. This means that if we take a sentence ϕ provable in S and we only have a proof P of ϕ in T, we are not able to estimate, by any algorithm, the length of a proof P' of ϕ in S. The reason is that with increasing lengths of proofs P the lengths of proofs P' grow at a rate so fast that it is beyond all computable functions. This result is due to Mostowski.

For all practical purposes it does not matter if the speed-up is exponential, superexponential, or non-computable, because it just means that some proofs in the weaker system are infeasible. What is remarkable about the result above is that such a speed-up appears when T is extended with any unprovable sentence α, however mild this extension is.

The second result concerns a single theory, which we use to prove theorems in two different ways. Suppose a sentence ϕ is provable in a theory T, but all the proofs are extremely long. Thus we are not able to write any of them down. It may, however, happen that we can still learn that a proof of ϕ in T exists and that we can prove this fact using T. This is the case when the sentence

"sentence ϕ is provable in T"

has a short proof in T.

Let us denote the above sentence by $Pr_T(\phi)$. It is important to realize that the two sentences ϕ and $Pr_T(\phi)$ are essentially different. The first one can be an arbitrary mathematical sentence, whereas the second one is a sentence speaking about proofs in a formal system for the theory T. Furthermore, it is not possible to derive one from the other in T, in spite of the fact that they convey essentially the same information.

To be quite precise, we have to assume something about the theory T. First, it has to be able to talk about finite structures, and it has to be sufficiently strong that it can prove basic things about syntax. Second, it has to be a true theory. Recall that this means that all sentences provable in T are true sentences. (For example, we know, by the Second Gödel Incompleteness Theorem, that there are *consistent* theories T that prove that *they are inconsistent*; such a theory T is not true, since it proves a sentence that is false.)

Let us get back to ϕ and $Pr_T(\phi)$ and consider the lengths of proofs of these sentences. One can show that if ϕ has a short proof, then also $Pr_T(\phi)$ has a short proof. More exactly, given a proof of ϕ we can construct a proof of $Pr_T(\phi)$ in polynomial time. The converse is not true. There are sentences ϕ such that $Pr_T(\phi)$ has a short proof, but ϕ has only very long proofs. Hence we can substantially shorten proofs by proving sentences $Pr_T(\phi)$ instead of proving ϕ directly. The speed-up is extremely large, but not as large as in the previous case. (In particular it can be bounded by a computable function, but this makes little difference from the practical point of view.)

What happens if we repeat this construction, namely, if *we prove that it is provable that ϕ is provable?* It is better to use a bit of formalism in such cases. So the question is: do we get a speed-up if we use $Pr_T(Pr_T(\phi))$ instead of $Pr_T(\phi)$? Yes, we do, and again the speed-up is extremely large. We can continue in this manner and each time we add another provability predicate, we get such a speed up.

In more intuitive terms, this phenomenon means that sometimes when we are stuck with a problem, we can solve it by considering it *"from a higher perspective"*. The *'higher perspective'* here means considering not the sentence itself, but its provability. The ability of humans to switch to a higher level is often contrasted with the limited abilities of computer programs in automated theorem proving. Furthermore, the fact that we have an infinite hierarchy of levels creates the illusion that in principle we can solve any problem—we need only to go to a sufficiently high level. I will explain in detail why this is wrong in Chapter 7. ♠ Here I will only show that iterating provability, although it does give a very large speed-up, is not as powerful as one would expect.

My argument will be based on comparing the two methods of speeding-up proofs considered in this subsection. The question is: given a theory T, do we get shorter proofs by extending by a new axiom, or by proving $Pr_T(\phi)$ instead of ϕ? To answer this question we need an important concept, the *reflection principle for T*. Roughly, it is the principle that allows us to derive the truth of a formula from the fact that it is provable. Formally, the reflection principle for T is the set of implications of the form

$$Pr_T(\phi) \to \phi,$$

for every sentence ϕ.

This principle is not provable in T, because it implies the consistency of T (see part C). Thus there are sentences ϕ for which the implications above are not provable in T. However, for some T' stronger than T, it is possible that T' proves (all implications of) the reflection principle for T. Furthermore, in typical cases the lengths of these proofs are short, namely, they can be bounded by a polynomial depending on the size of ϕ. Let T' be an extension of T that satisfies this property and let us now compare

- L_1: the length of the shortest proof of $Pr_T(\phi)$ in T, and
- L_2: the length of the shortest proof of ϕ in T'.

Since T' is an extension of T, the T-proof of $Pr_T(\phi)$ is also a T'-proof of $Pr_T(\phi)$. So the shortest proof of $Pr_T(\phi)$ in T' has length at most L_1. Also in T' we can prove $Pr_T(\phi) \to \phi$ by a short proof, say, of length m. Then, using modus ponens, we get a proof of ϕ of length at most $L_1 + m$. Thus we have obtained a bound

$$L_2 \leq L_1 + m,$$

where m is only polynomial in the length of ϕ.

In other words, in T' we can prove ϕ almost as quickly as we can prove $Pr_T(\phi)$ in T. There is essentially no speed-up between proofs of ϕ and proofs of $Pr_T(\phi)$ if we work in the stronger theory T'. But not only that: we can just as easily obtain $Pr_T(\phi)$ from $Pr_T(Pr_T(\phi))$ and then derive ϕ as above. Clearly, we can continue in this way for any finite number of applications of the provability predicate. Thus T' eliminates speed-up for any number of iterations of Pr_T!

It is not difficult to come up with such extensions. Let us take as an example Zermelo-Fraenkel Set Theory, ZF. In order to obtain an extension of ZF in which the sentences expressing the reflection principle for ZF have short proofs, we need only to add the axiom postulating the existence of an inaccessible cardinal number, which is the weakest of all large cardinal axioms. Essentially the same can be done with any strengthening of ZF. For example, let T be ZF augmented with an axiom expressing the existence of a measurable cardinal. Then it suffices to add an axiom saying that there exists an inaccessible cardinal above a measurable cardinal.

The conclusion is that *it is much more effective to use new mathematical axioms than "shifting to a higher perspective"*. However, we must be aware of the danger of introducing inconsistencies, if we use new axioms.

I should also stress that so far we do not have truly mathematical sentences for which we are able to prove speed-up in these situations. Although there are concrete examples of natural theorems which have short proofs using some theory, but for which we only know long elementary proofs, we are not able to prove the speed-up formally. If this were achieved, it would be a spectacular result.

I will return to this subject in the next chapter ♠ , as these are central questions in the foundations of mathematics and need more detailed study.

— B —

Sequent calculus and cut-elimination

Now I will consider another way of transforming general proofs into direct ones. It was invented by Gerhard Gentzen (1909 - 1945), who incidentally also died young and tragically. He was a victim of Czech mistreatment of Germans in post-war Czechoslovakia.

Gentzen's idea was to construct a formal system in which formulas are gradually derived from smaller ones. Thus for each connective he picked a few natural rules that enabled him to derive compound formulas from their components. Let us consider a couple of examples. Natural rules for conjunction and disjunction are:

- *from A and B derive $A \wedge B$;*
- *from A derive $A \vee B$;*
- *from B derive $A \vee B$.*

Unfortunately, this idea does not work without a technical modification of the system. Recall that a proof is a sequence of formulas such that each formula is an instance of an axiom schema or is derived from previous formulas by a logical rule available in the given system. So one step of a proof is one formula. The modification that Gentzen introduced is that one step may consist of several formulas. Such a string of formulas is called a *sequent.* The meaning of a sequent is the disjunction of the formulas in it. The reason why we do not write it in the form of a disjunction is that we want to define operations on *formulas* rather than on *subformulas.* In this context it is better to view a proof as a text whose lines are sequents. Therefore we also say *proof lines* instead of *proof steps.*

In a sequent calculus we apply rules to a fixed small number of formulas in one or more sequents (usually just one) and the rest of the sequents is simply copied to the next proof line. Thus an application of the rule for conjunction looks like:

$$\begin{array}{l} \quad\vdots \\ \alpha, \gamma_1, \dots, \gamma_k \\ \quad\vdots \\ \beta, \delta_1, \dots, \delta_l \\ \quad\vdots \\ \alpha \wedge \beta, \gamma_1, \dots, \gamma_k, \delta_1, \dots, \delta_l \end{array}$$

The rules for other connectives and for quantifiers are treated in a similar fashion.

We also need some *initial sequents*, lines that can always be added. These are all sequents of the form $\alpha, \neg\alpha$, where α stands for an arbitrary formula. These sequents represent the law of excluded middle. Further, we need some *structural rules*, namely, *permutation, weakening* and *contraction.* The first one enables us to permute the formulas in a sequent. Using weakening we can add an arbitrary formula to a sequent (which makes the disjunction of the formulas weaker; thus this rule also preserves logical validity). To reduce the length of a sequent we are allowed to "contract" pairs of identical formulas, by replacing them with just one formula.

There is another structural rule, which is treated separately, because it plays a special role. This rule is called *cut* and enables us to eliminate a pair of complementary formulas. Here is how it looks:

$$\begin{array}{l} \vdots \\ \alpha, \gamma_1, \ldots, \gamma_k \\ \vdots \\ \neg\alpha, \delta_1, \ldots, \delta_l \\ \vdots \\ \gamma_1, \ldots, \gamma_k, \delta_1, \ldots, \delta_l \end{array}$$

Notice that contraction and cut are the only rules that reduce the complexity of sequents. Whereas contraction only discards repetitions of the same formula, using cut we can completely eliminate a formula from the rest of the proof. Thus it is only the presence of a cut that makes a proof indirect. We can view the first two sequents participating in an application of the cut rule as lemmas needed to derive the third sequent. The role of these sequents as lemmas is more apparent if, for example, the side formulas $\gamma_1, \ldots, \gamma_k$ in the first sequent are missing. Then α itself is a lemma. The second sequent, $\neg\alpha, \delta_1, \ldots, \delta_l$, can be interpreted as the lemma *'α implies the disjunction of $\delta_1, \ldots, \delta_l$'*, formally expressed by

$$\alpha \rightarrow \delta_1 \vee \ldots \vee \delta_l.$$

Thus, from lemma α and this implication, we get $\delta_1 \vee \ldots \vee \delta_l$.

Gentzen proved that one can already derive all true sentences in the system without the cut rule. The reason for having the cut rule in the system is that it makes the system more efficient. The cut rule is needed to simulate the rule of *modus ponens*, which is the main rule used in natural reasoning. We will see that without the cut rule some proofs must be extremely long. Moreover, Gentzen described an algorithm to transform a proof with cuts into a cut-free proof. This is called the *cut-elimination procedure.*

Recall that all propositional rules, except for cut, construct larger formulas from smaller ones. The quantifier rules do not change the propositional structure of formulas; they only change terms and add quantifiers. Hence the only parts that can possibly occur in a cut-free proof without occurring in the proven sequent are terms. Let us say that a formula α is a *quasi-subformula* of β if it is a subformula when we disregard terms and variables. For example, $\exists z(\neg P(z, f(z)))$ is a quasi-subformula of $\forall x(\exists y(\neg P(x, y)) \vee Q(x))$. Then cut-free proofs have the following *subformula property*:

Every formula in a proof is a quasi-subformula of some formula in the last sequent.

The subformula property allows us to consider cut-free proofs to be direct proofs, and Gentzen's cut-elimination procedure to be a way of transforming general proofs into direct ones.

Herbrand's Theorem can easily be derived from Gentzen's Cut-elimination Theorem. Consider a proof of a logically valid sentence $\exists x\ \phi(x)$. By Gentzen's Theorem, we can transform it into a cut-free proof. Then one can easily see that it is possible to permute the proof lines so that all instances of propositional rules precede instances of quantifier rules. Thus the resulting proof has the following form.

$$\begin{array}{cl} \vdots & \\ \vdots & \text{[propositional rules]} \\ \vdots & \\ \phi(t_1), \phi(t_2), \ldots, \phi(t_n) & \\ \vdots & \\ \vdots & \text{[instances of } \exists\text{-introduction rule applied to } \phi(t_i)\text{'s,} \\ \vdots & \text{and contraction applied to pairs } \exists x\ \phi(x), \exists x\ \phi(x)] \\ \vdots & \\ \exists x\ \phi(x) & \end{array}$$

Recall that the interpretation of a sequent is the disjunction of the formulas in it. So the sequent in the middle of the proof is a Herbrand disjunction and is derived in the propositional calculus.

Although the cut-elimination procedure is not extremely complicated, it may result in a tremendous increase in the length of the proof. The estimates of the length of the cut-free proof have the form of the superexponential function. If we denote by n the length of a proof with cuts, then one can only bound the length of the cut-free proof by

$$2^{2^{\cdot^{\cdot^{2}}}}\Big\}{\scriptstyle n\text{-times}}.$$

It would already be bad if it were a single exponential, 2^n, but this is much worse. It is not just because we do not know a better cut-elimination procedure; it has been proven that the superexponential cannot be replaced by a function that grows more slowly. Examples of sentences for which such an increase of length is unavoidable were found by Richard Statman. These bounds are quite general: they not only hold for cut-elimination in the sequent calculus, they also show that if we want to get any proof with formulas of low complexity, we have to increase the size of the proof this much. In particular, the bound also applies to Herbrand's Theorem.

In mathematics we distinguish elementary and nonelementary proofs. This is not a precisely defined concept; roughly, we say that a proof is nonelementary if it uses results from a branch of mathematics other than the one from which the theorem comes. In many cases, a nonelementary proof was discovered first and an elementary one was only found later. But in some cases we still only have nonelementary proofs. This concept is very much related to the formal concept of direct and indirect proofs. The fact that direct proofs may be much longer than indirect ones suggests that the reason why we do not have elementary proofs of some theorems is that the length of such proofs is too large.

We can draw a similar conclusion for automated theorem proving. Almost all automated theorem provers are based on Herbrand's Theorem. It is very likely that in this way we will never be able to prove some theorems that have already been proven by mathematicians. Therefore it is necessary to extend theorem provers so that they will also be able to generate lemmas. Some research into this problem has already been done, but finding useful lemmas is still more an art than a science.

Useful inconsistent theories

A logical theory classifies sentences into two categories: true and false. We know that we cannot fully formalize the true sentences in, for example, arithmetic. Thus for such theories we also have a third category—the independent sentences. However what really matters is that a consistent theory gives us *some* classification. When a theory is inconsistent, everything is provable, so there is no classification. This is because in logic we can derive any sentence from a contradiction. The Latin name of this principle is *ex falso quodlibet,* which means *from falsehood, anything you like.* Therefore the problem of consistency is the main problem in the foundations of mathematics.

Thus inconsistent theories seem to be completely useless. But if we replace the qualitative concept of provability by the quantitative concept of the lengths of proofs, the situation looks different. If T is inconsistent, then we may still distinguish between sentences that have short proofs and those that have long proofs. This is only possible when the shortest proof of contradiction is long, since otherwise every short sentence also has a short proof.

Another reason for considering only consistent theories is that we believe that reality is consistent. But often there is an intermediate phase between observation and deduction, the phase when we propose a theory explaining the observed facts. To this end, we must only use consistent theories, as any inconsistent theory is surely a wrong description. But quite surprisingly, if we take into account the lengths of proofs, we can also use inconsistent theories to prove true facts. That is because it can happen that although the theory is inconsistent, all sentences that have short proofs are true.

These ideas appeared in the beautiful seminal paper *Existence and Feasibility in Arithmetic* of Rohit Parikh. It is the same paper where he showed the speed-up using provability predicates that I explained in the previous subsection. To construct an inconsistent theory with the required properties, Parikh started with an arbitrary consistent extension T of Peano Arithmetic. Then he added a new predicate $F(x)$ (F standing for *'feasible number'*) and several axioms. For the sake of simplicity, I will only use the most important ones:

1. *$F(0)$ (0 is feasible);*

2. *if $F(x)$ then $F(x+1)$ (if x is feasible, so is $x+1$);*

3. *there exists a number x such that $\neg F(x)$ (there exists an unfeasible number).*

This theory is still consistent. Recall the subsection of Chapter 3 where we considered nonstandard analysis (see page **??**). In those theories we can distinguish between finite and infinite natural numbers. We can interpret the predicate $F(x)$ as finite numbers and the third axiom will just express that there also exist infinite numbers. Thus the above theory can be consistently interpreted in nonstandard analysis. (In fact, any nonstandard model of arithmetic will do.) The meaning of the axioms is that induction fails for the predicate F. This is all right, because the axioms of Peano Arithmetic only say that induction holds for arithmetical formulas and F is a new predicate which is not in the language of arithmetic.

Notice that the reason why the theory is consistent is that we do not specify the unfeasible number in condition 3. The key idea is to change the third axiom so that it does talk about a concrete number, but a very large one.

3'. *$\neg F(t)$ (t is unfeasible).*

Here one picks for t a suitable closed arithmetical term that represents a concrete large number m.

This theory, let us call it FPA_t, is clearly inconsistent. If we start with axiom 1 and apply axiom 2 m times we will prove that m is feasible. More precisely we prove that the number represented by

$$0 + \underbrace{1 + 1 + \ldots + 1}_{m-\text{times}}$$

is feasible. Then we need only to prove that this term represents the same number as the term t. The length of such a proof can be estimated by some small multiple of m. However, if the term t is a concise representation of m, there can be much shorter proofs. In general, to ensure that no proof of contradiction in FPA_t is shorter than n, the value of the term t must be much larger than n. Parikh showed that if we take an approximately n times iterated exponential (a stack of twos of length several times n), then one can guarantee that there is no proof of contradiction of length less than n.

In effect he showed more: if one considers sentences that do not mention the predicate F, then such short proofs prove only true sentences, although the proof may use F. Thus if we pick n to be a safe upper bound on the lengths of all proofs that can ever be produced, and m to be the n times iterated exponential, then we can only prove true sentences about numbers.

Let us observe that for this theorem to be nontrivial it is necessary to have a short term t representing the large number m. Such a term cannot be constructed in the basic language of arithmetic that uses only $0, +, \cdot$. We need to enrich the language by exponentiation *exp* (the function 2^x) and superexponentiation *supexp*. These functions can be described using axioms that correspond to their recursive definitions.

$$\begin{aligned} exp(0) &= 1 \\ exp(x+1) &= 2 \cdot exp(x) \\ supexp(0) &= 1 \\ supexp(x+1) &= exp(supexp(x)) \end{aligned}$$

Using the function symbol *supexp* we can write a very short term

$$supexp(supexp(supexp(1+1+1)))$$

whose value is large enough to ensure that no contradiction will ever be proved in FPA_t and, moreover, all arithmetical sentences that one will ever be able to prove will be true statements.

This theory is used to justify the strict finitistic point of view called *ultrafinitism*. According to this approach to the foundations of mathematics we can only say that a number n exists if it is small enough to be physically represented. Large numbers are considered to be only abstractions that do not represent existing entities. In the theory above, the predicate F is used to distinguish between small and large, or as the ultrafinitists say, between feasible and unfeasible. But this theory does not render correctly our intuition about small (feasible) on the one hand, and large (unfeasible) on the other. We consider $2^{2^{100}}$ already to be a large number, as we surely cannot represent it in decimal notation in the visible universe. But if we take $t = 2^{2^{100}}$, we will get a very short proof of contradiction in FPA_t. (I estimate it to be half a page, even if written completely formally).

I think that the significance of this result is rather in showing that there are theories that have very short axiomatic systems and are inconsistent, but where every proof of contradiction is extremely long. Thus we cannot exclude the possibility that the set theories that we are using as the foundations of mathematics are inconsistent, but we will never find a contradiction. The positive message is that if this is so, it may still be all right: if the shortest proof of contradiction is really very large, then what we are proving are true sentences anyway.

This is, of course, not the only possible scenario of what can happen if we admit the possibility that we are using inconsistent theories. The reason why we are not able to find a contradiction (and perhaps never will) may not be the large length of the proofs of contradiction. It could happen that there is a proof of contradiction of *medium length*, but one that is, so to speak, well-hidden. Recall that deciding if a sentence has a proof of polynomial length, for some fixed polynomial, is an **NP**-complete problem. Thus if $\mathbf{P} \neq \mathbf{NP}$ (and if this manifests itself on the level of medium length inputs), there is no efficient algorithm to find a proof of contradiction, even if we know that it has medium length. Mathematicians are not algorithms, so this is only an indication that it may be hard. But if one believes that cryptography is possible, namely, that there are secure encryption schemas, then one must also admit that to hide a contradiction so that nobody will find it is possible as well.

The prevailing feeling is that Nature is not evil. She does not reveal secrets easily, but she also does not try hard to hide them from us. Therefore we believe that if there is a contradiction in set theory, or if some large cardinal is inconsistent, we will eventually discover this. Furthermore, we are not confined to simply searching the proofs in a given theory. We may look at the theory from higher perspectives and thus even be able to find a proof of contradiction whose length is beyond what can be written down.

My own view is that, most likely, Zermelo-Fraenkel Set Theory is consistent ♠ and is also consistent with most of the currently studied large-cardinal axioms. The reason I am talking about the possibility of inconsistency is that we need to have a complete picture of what can happen. We need it in order to understand the main problem of the foundations of mathematics, which is the consistency of the foundations. ♠

Ordinal analysis of theories and fast growing functions.

In mathematics we often assign parameters to structures which provide us with some basic information about them. Usually the parameter is a natural number and is called *order, rank, dimension, degree* etc. Such parameters enable us to compare the structures we are studying or help us to prove that they are different. For theories in predicate logic we do not have any useful numerical parameter (the number k_T proposed by Chaitin, which I mentioned in the last chapter, is practically impossible to determine, thus is only of theoretical interest). It is however possible to assign countable *infinite ordinals* to theories and in several cases the ordinal has also been computed. These ordinals are *constructive,* which means that they are explicitly described (see page **??**). Thus, although infinite, they can be represented by finite objects. Since ordinals are linearly ordered we can use them, for example, to compare the strength of theories.

The study of ordinals assigned to theories is an important branch of proof theory, called *ordinal analysis of theories.* This branch of proof theory started with the work of Gentzen, who proved the consistency of Peano Arithmetic using transfinite induction over ϵ_0.[2] The ordinal ϵ_0 has a more intimate relation to Peano Arithmetic: it is the least ordinal for which Peano Arithmetic is not able to prove that it is well-ordered. The fact that Peano Arithmetic is unable to prove that ϵ_0 is well ordered follows from a combination of Gentzen's result with Gödel's. Indeed, if we had a proof that ϵ_0 were well ordered in Peano Arithmetic, then we could use Gentzen's proof to prove that Peano Arithmetic was consistent, which would violate Gödel's Theorem.

In this way the ordinal ϵ_0 is uniquely determined by Peano Arithmetic. In general, the least ordinal α for which a theory T does not prove that it is well-ordered is called *the proof-theoretic ordinal of the theory T*. In logic we often need to talk about "natural" representations of concepts in theories. Therefore it is remarkable that the proof-theoretic ordinal of a theory is simply an ordinal, not a suitable representation of an ordinal. To this end one has to define it more carefully.

Definition 1 *The proof-theoretic ordinal of T is the least ordinal α such that for no representation of α, T proves that it is well-ordered.*

[2] I briefly mentioned this in Chapter 2, page **??**, but it was only in part C; now I am going to describe it in more detail.

Equivalently, α is the limit of all ordinals with representations for which T proves the well-ordering property. (So the trick that enables one to avoid talking about natural representations is to consider all possible representations.)

In order to avoid problems with formalizing the concept of well-ordering, one only considers theories in which it is possible to talk about sets. Then we can use the usual set-theoretical definitions. This excludes Peano Arithmetic as it is usually defined, but one can define an extension in which sets are present and which is essentially of the same strength.

Naturally, we would like to find proof-theoretic ordinals for as many theories as possible. The motivation is not only to determine a parameter that reflects the strength of theories, but also to get consistency proofs for theories stronger than Peano Arithmetic. This is because having the proof-theoretic ordinal we are usually able to produce a constructive proof of the consistency of the theory. Of course, such consistency proofs do not solve the consistency problem, they only replace the assumption of the consistency of the theory with the assumption that the particular ordinal notation defines a well-ordered set, but it is additional evidence for believing that the theory is consistent. In any case, the ordinal analysis usually provides us with a better understanding of the theory and its relation to other theories.

Let us have a closer look at the case of Peano Arithmetic. Peano Arithmetic is the theory that postulates the principle of induction for all arithmetical formulas. It is an easy fact that the principle of induction is equivalent to the least number principle (that *every nonempty set has the least element*). This is nothing else but the fact that the set of natural numbers, which is the least infinite ordinal ω, is well-ordered. Now, this may easily lead to the confusion: *why did Gentzen reduce the consistency of the theory that says that ω is well-ordered to the fact that ϵ_0, which is a larger ordinal, is well-ordered?* The reason is that he wanted to find a *constructive* proof of the consistency of Peano Arithmetic, a proof that would only use simple concepts, which means algorithmically decidable relations. Therefore he could not use the general least number principle he had to use the principle restricted to computable subsets. This is the reason why he had to take a larger ordinal. Thus Gentzen's result can be presented as the reduction of

(a) the least number principle for *arithmetical sets* and the ordinal ω, to

(b) the least number principle for *computable sets* (a smaller class) and ϵ_0 (a larger ordinal).

How much did Gentzen fulfill Hilbert's vision of the proofs of consistency? One may be worried by the fact that the proof uses a transfinite ordinal much larger than ω, the ordinal representing natural numbers. Indeed, in the standard set-theoretical representation ε_0 is the set of all ordinals less than ε_0, thus it is a set that contains infinite sets (which, in turn, also contain infinite sets etc.). But recall that Cantor normal form (Chapter 3, page **??**) enables us to represent all ordinals below ε_0 by finite terms (or even natural numbers, if we prefer) so that the ordering relation is algorithmically decidable. As a matter of fact, any constructive ordinal has a representation of this kind *by definition*, but the representation of ordinals below ε_0 is especially simple.

Thus the complexity of (b) is not caused by the representation of the ordinal, but by its properties. It is good to recall the application of ε_0 to proving that Goodstein sequences always end at zero (page **??**). For each Goodstein sequence we constructed a strictly decreasing sequence of ordinals below ε_0. We also noticed that for Goodstein sequences to get to zero it takes an extremely large number of steps. Therefore, also the decreasing sequences of ordinals are extremely long. They are so long that it is quite nontrivial to prove that they actually reach zero. For a linear ordering, the condition that every decreasing sequence of elements is finite is equivalent to being well-ordered. Thus it is also quite nontrivial to prove using limited means that the representation of ε_0 by the Cantor normal form defines a well-ordering.

According to the Second Incompleteness Theorem one cannot reduce the consistency of Peano Arithmetic to a very simple statement; it must be a statement unprovable in Peano Arithmetic. Therefore, Gentzen's proof is not a *reduction* of the strength of the statement, it is rather a *transformation* into a different kind of statement. No matter whether we accept it as a solution of Hilbert's problem for Peano Arithmetic, it certainly shows the consistency from a different perspective and that is always useful.

Let us now look at how one can determine the proof-theoretic ordinal. To prove that α is the proof-theoretic ordinal of T we have to show two things:

1. for every $\beta < \alpha$, T proves that β is well-ordered (a lower bound);

2. T does not prove that α is well-ordered (an upper bound).

The second part is certainly more interesting. All proofs of 2. are based on the Second Incompleteness Theorem, namely, one proves that the sentence *"α is constructively well-ordered"* implies the consistency of T. To give you some idea of these proofs, I will briefly sketch the main steps of Gentzen's proof of the consistency of Peano Arithmetic using ϵ_0, which gives the upper bound on the proof-theoretic ordinal of Peano Arithmetic.

Having in mind that we should use only constructive means, Gentzen's idea of the proof of consistency looks quite natural. Gentzen's proof, like essentially all proofs of consistency, uses induction on the length of a proof to show that every formula in the proof is true, which in particular implies that a contradiction cannot appear in the proof. To this end we need to be able to express that a formula is true. According to the Gödel-Tarski theorem on the undefinability of truth (see page **??**), we cannot define the truth of arithmetical formulas in arithmetic, since this relation is of higher complexity. But we do not need to consider all proofs, we are interested only in proofs of contradiction. The important fact is that contradiction can be expressed by a quantifier free formula, say $0 = 1$. However, in the rest of the proof we can have arbitrarily complex arithmetical formulas, which is the main problem that we have to cope with.

Now recall what we learned in the first subsection: there are means to reduce the complexity of formulas in proofs. By Gentzen's Cut-Elimination Theorem, we can transform every proof into a cut-free one. If we have a cut-free proof, we know that all formulas in the proof are quasi-subformulas of the proven sentence. Then in order to prove that all formulas in the proof are true, we only need a truth definition for quasi-subformulas of the proven sentence. In particular, if the sentence has small quantifier complexity, we only need the truth definition for formulas of that quantifier complexity.

This looks promising, but what is the formula that we want to show to be unprovable? It is not $0 = 1$; we are considering provability from axioms, hence it is an implication of the form

$$(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n) \rightarrow 0 = 1,$$

where $\alpha_1, \ldots, \alpha_n$ are some axioms of Peano Arithmetic. So we need another idea. (Of course, we did not expect that the consistency proof would be so easy—if it were, we could do it in Peano Arithmetic, which is impossible by the Second Incompleteness Theorem.)

The next idea is, basically, to eliminate also the use of induction axioms. Suppose we have proven the assumption of an induction axiom

$$\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(x+1)), \tag{6.2}$$

and we want to derive the conclusion

$$\forall x\ \phi(x). \tag{6.3}$$

The crucial observation is that if we only need a numerical instance $\phi(n)$ of the sentence $\forall x\ \phi(x)$, we can derive it from the assumption (6.2) only using *logic*, namely, $n + 1$ applications of *modus ponens*.[3] (By the way, the reason we believe in the induction principle is that it is a repetition of *modus ponens* infinitely many times.)

This is good, because we want to eliminate the use of axioms so that we can then apply cut-elimination to reduce the complexity of formulas in the proof. What is not good is that we only get numerical instances, instead of the sentence with the universal quantifier (6.3). At this point another idea steps in, which is to allow a rule with an infinite number of premises. The rule is:

From $\phi(0), \phi(1), \phi(2), \ldots,$ *derive* $\forall x\ \phi(x)$.

Combining this rule with the proofs of numerical instances makes it possible to derive the conclusion of the induction axiom for ϕ. Thus we can eliminate induction axioms. What remains are the basic axioms that determine the properties of the successor, addition and multiplication. This is a finite set of axioms and, moreover, their quantifier complexity is low. In fact, if we introduce the predecessor function, we can present them using purely universal sentences.

The elimination of the induction axioms is not for free. The price we have to pay is that we must use infinite proofs. It may look strange that we wanted to find a constructive proof and yet ended up

[3] Assume we represent n by the sum of n ones, $1 + 1 + \ldots + 1$.

using infinite proofs that use a very nonconstructive infinite rule. The point is that we are not going to study such proofs in general. We only need to describe the infinite proofs that result from finite proofs by eliminating the induction axioms. We can represent each such proof by the program that produces its parts. A program is a piece of finite text, so these infinite proofs are represented by finite entities.

The proof that we obtain by eliminating the induction axioms is infinite, but it shares an important property with finite proofs: the length from any initial sequent to the last sequent is finite. Since the proof tree is infinitely branching, due to the new rule, the latter property does not imply that there is a general bound on the length of the branches of the tree. Typically there will be arbitrarily long (finite) paths from initial sequents to the last sequent. The nice feature of such trees is that, although we cannot use numbers to bound their depths, we can measure the depth using ordinals.

The next step is to apply cut-elimination. This is done very much like for ordinary finite proofs, so I will not go into the details. Thus we finally obtain a proof all of whose formulas are of low complexity, namely, they define computable predicates. For such formulas we do have a definition of truth in Peano Arithmetic. Thus we may apply the standard way of proving consistency—proving by induction that all formulas in the proof are true. Since we are not in the standard situation, where proofs are finite, we have to use a stronger means. Thus to prove that all formulas in such a proof are true we use *transfinite induction* on the ordinals that bound the depths of such proofs.

These are the basic ideas of the proof. The technical part is to estimate the ordinal depths of the infinite proofs obtained by the elimination of induction and cuts. Once it is established that the depth is always some ordinal below ϵ_0, all the pieces fit together and the consistency of Peano Arithmetic is proven.

One can prove that the proof-theoretic ordinal exists for every theory T that extends Peano Arithmetic. But this general theorem gives us little information about specific proof-theoretic ordinals. What we need is a representation of this ordinal based on a combinatorial construction, such as the Cantor normal form of ordinals less than ϵ_0, rather than a construction based on syntactical concepts related to the theory. This is already very hard for relatively weak set theories, and ingenious tricks are needed to make even slow progress in this field. There seems to be no hope to do the ordinal analysis of theories such as Zermelo-Fraenkel Set Theory with the means presently available. Ordinal analysis fails also at the opposite end of the spectrum of theories—for weak theories. One can analyze subtheories of Peano Arithmetic where induction is restricted to Σ_n formulas, but for theories weaker than arithmetic with induction for Σ_1 formulas, we are not able to use ordinals to distinguish different theories. This is quite unfortunate, because such weak theories play a key role in proof complexity, as we will see in the next section, and we lack any means for proving that some theories are stronger than others.

*

In Chapter 4 we saw that the unprovability of some sentences may be caused by the fact that they implicitly define very fast growing functions (see page **??**). As n grows, the values of such a function $f(n)$ are so large that the theory is unable to prove that computations of the value of f always terminate. This suggests that we could compare theories according to the fast-growing functions they are able to handle.

We will say that a function f is provably total in a theory T, if there exists a definition of f in T such that it is provable in T that for every natural number n the value of $f(n)$ is defined. We consider only computable functions and every such function has a definition in any theory that contains a sufficiently strong set of arithmetical axioms. If T is finitely axiomatized or given by a computable set of axioms, we can effectively enumerate all definitions of computable functions in T and all proofs of their totality. Then we can take the "diagonal" over all provably total functions and obtain a computable function that is not provably total. Thus for every such theory there are functions that are not provably total hence the resulting hierarchy of theories is infinite.

Using functions to gauge theories is not very convenient, as there are too many functions. It is better to take a suitable subset which is linearly ordered and use it as a scale. Such a scale of functions, usually called a *hierarchy*, cannot consist of functions indexed by natural numbers

$$f_0, f_1, f_2, \ldots$$

where the program for f_n is computable from the index n. (This can be proven using the standard diagonal technique—take the diagonal function $F(n) = f_n(n)$ and obtain a function that is not bounded by any of the functions in the scale.) Therefore one uses hierarchies of functions indexed by constructive ordinals. There is no canonical way of defining such a hierarchy for all constructive ordinals; the hierarchies always depend on technical details of their definitions. (I sketched the basic idea of these constructions in Chapter 3, page **??**.)

If we want to prove that the functions f_α are provably total in T for all ordinals α below some constructive ordinal κ, we need to construct a representation of every such α in T and prove in T that it defines a well-ordered set. Then we can use transfinite induction over α to prove that all functions f_α, for $\alpha < \kappa$, are provably total. If T does not prove that α is well-ordered for some $\alpha < \kappa$, then it also does not prove that f_α is total. Thus using function hierarchies we get the same parameter as before, the proof-theoretic ordinal of the theory. So the possibility of formalizing large well-orderings is closely tied with the possibility of formalizing fast growing functions.

Let me conclude by stressing that one should always bear in mind that there is no absolute measure of the strength of a theory. Thus any parameter, such as the proof-theoretic ordinal, gives us only partial information about it. ♠

— C —

Sequent calculi. What I described above is a modification of Gentzen's sequent calculus proposed by Kurt Schütte (1909-1998). It is called *one sided sequent calculus* because the original one has sequents with two parts separated by an arrow. The meaning of a sequent

$$\phi_1, \dots, \phi_n \Rightarrow \psi_1, \dots, \psi_m$$

is

$$\phi_1 \wedge \dots \wedge \phi_n \rightarrow \psi_1 \vee \dots \vee \psi_m,$$

(the conjunction of the first sequence of formulas implies the disjunction of the second one). Gentzen's calculus is very natural and useful, but has the disadvantage of having a large number of rules. Therefore, I will define a version of Schütte's calculus.

Sequents are finite multisets of formulas using the connectives $\neg, \wedge, \vee$ and the two quantifiers $\forall, \exists$. A multiset will be represented by a string of formulas separated by commas; viewing it as a multiset means that we will ignore the order of formulas in the string. To save using the rules for negation, we will only allow negations in atomic formulas. For a compound formula ϕ, we interpret $\neg\phi$ as a formula obtained by pushing the negation to the atomic formulas while implicitly using the rules

$$\neg(A \wedge B) \equiv \neg A \vee \neg B, \quad \neg(A \vee B) \equiv \neg A \wedge \neg B, \quad \neg\forall x A \equiv \exists x \neg A, \quad \neg\exists x A \equiv \forall x \neg A, \quad \neg\neg A \equiv A.$$

The explicit rules are:

- *axiom* $A, \neg A$; (axiom)
- *from* Γ, *derive* Γ, A; (weakening)
- *from* Γ, A, A, *derive* Γ, A; (contraction)
- *from* Γ, A, B, *derive* $\Gamma, A \vee B$; ($\vee$-introduction)
- *from* Γ, A *and* Δ, B, *derive* $\Gamma, \Delta, A \wedge B$; ($\wedge$-introduction)
- *from* $\Gamma, A(y)$, *where* y *is not free in* Γ, *derive* $\Gamma, \forall x A$; ($\forall$-introduction)
- *from* $\Gamma, A(t)$, *derive* $\Gamma, \exists x A(x)$; ($\exists$-introduction)
- *from* Γ, A *and* $\Delta, \neg A$, *derive* Γ, Δ. (cut)

Here I am using A and B to represent formulas, Γ and Δ to represent strings of formulas, x and y to represent variables and t to represent a term. Proofs in sequent calculi are usually presented in a tree form with applications of rules shown by horizontal lines.

Example. Consider a proof of the propositional tautology $\neg\alpha \vee (\alpha \wedge \neg\beta) \vee (\beta \wedge \neg\gamma) \vee \gamma$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha,\neg\alpha \qquad \beta,\neg\beta}{\neg\alpha,(\alpha\wedge\neg\beta),\beta} \qquad \gamma,\neg\gamma}{\neg\alpha,(\alpha\wedge\neg\beta),(\beta\wedge\neg\gamma),\gamma}}{\neg\alpha\vee(\alpha\wedge\neg\beta),(\beta\wedge\neg\gamma),\gamma}}{\neg\alpha\vee(\alpha\wedge\neg\beta)\vee(\beta\wedge\neg\gamma),\gamma}}{\neg\alpha\vee(\alpha\wedge\neg\beta)\vee(\beta\wedge\neg\gamma)\vee\gamma}$$

The graph representing the tree structure of this proof is in Figure 6.1.

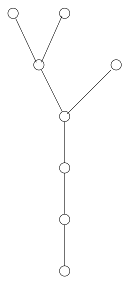

Figure 6.1: The proof-tree of the example.

Proofs in sequent calculi are quite cumbersome. These calculi are not practical for formalizing mathematical proofs, but are very convenient for proof theory. In particular, the transformation of indirect proofs into direct ones is quite natural—this is the cut-elimination procedure.

Cut-elimination. The main idea of cut-elimination is to replace cuts with complex formulas by cuts with less complex formulas. By eliminating one cut with a complex formula we may introduce a large number of cuts with simpler cut formulas, but we still make progress, because the number of complex cuts is reduced.

As an example consider the elimination of a cut in which the cut formula is a disjunction of two other formulas. Let the cut formula be $\phi\vee\psi$. In the calculus introduced above, the negation of it is identified with $\neg\phi\wedge\neg\psi$. One has to consider several cases of how these formulas were introduced in the proof. The most typical case is that $\phi\vee\psi$ was introduced by $\vee$-introduction and $\neg\phi\wedge\neg\psi$ was introduced by $\wedge$ introduction. Then instead of the cut applied to $\phi\vee\psi$ and $\neg\phi\wedge\neg\psi$, we apply two cuts to the pairs $\phi,\neg\phi$ and $\psi,\neg\psi$. This is shown schematically as the transformation of the proof

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{\vdots}{\Gamma',\phi,\psi}}{\Gamma',\phi\vee\psi}\\ \vdots\end{array}}{\Gamma,\phi\vee\psi} \qquad \dfrac{\begin{array}{c}\dfrac{\dfrac{\vdots}{\Delta',\neg\phi}\qquad\dfrac{\vdots}{\Delta'',\neg\psi}}{\Delta',\Delta'',\neg\phi\wedge\neg\psi}\\ \vdots\end{array}}{\Delta,\neg\phi\wedge\neg\psi}}{\Gamma,\Delta}\\ \vdots\end{array}$$

into the proof

$$\begin{array}{c}\dfrac{\begin{array}{c}\dfrac{\dfrac{\dfrac{\vdots}{\Gamma',\phi,\psi}\qquad\dfrac{\vdots}{\Delta',\neg\phi}}{\Gamma',\Delta',\psi}\qquad\dfrac{\vdots}{\Delta''\neg\psi}}{\Gamma',\Delta',\Delta''}\\ \vdots\end{array}}{\Gamma,\Delta}\\ \vdots\end{array}$$

In this particular type of proof the formula $\phi\vee\psi$ was introduced only once, therefore the proof after the elimination of the cut is not larger. In general, $\phi \vee \psi$ can be introduced several times. Then we have to introduce more cuts on the pairs $\phi, \neg\phi$ and $\psi, \neg\psi$ and the resulting proof is larger. Thus we have to choose a suitable strategy, i.e., an order for eliminating cuts, otherwise the procedure may not terminate.

One strategy that leads to a terminating procedure is to always eliminate the uppermost cuts. If the cut formula in such a cut is not atomic, the elimination will produce new cuts, but the complexity of the formulas in the new cuts will be smaller. Then we eliminate those, which will produce even more cuts, etc. But since elimination of cuts on atomic formulas does not produce new cuts, we will eventually eliminate all cuts.

This resembles the problem of Hercules and Hydra (see page **??**), but here Hydra behaves differently: if we cut a head, the new heads will grow on upper, not lower, nodes. Also in cut elimination one only needs a superexponential number of steps, whereas in the game of Hercules and Hydra the number is much larger. I invite the reader to design the corresponding game and to study strategies for it.

Herbrand's Theorem. There are constructive proofs that give an explicit construction of terms in the disjunction and nonconstructive proofs that only prove the existence of such terms. I will explain the idea of a nonconstructive proof. The proof is very similar to the proof of the Completeness Theorem that I sketched in Chapter 2 (page **??**).

Suppose, as before, we have a sentence with only one quantifier and it is an existential quantifier, $\exists x\ \phi(x)$. Arguing by contradiction, suppose that none of the disjunctions of the form (6.1) is a propositional tautology. This is equivalent to the fact that any finite set of term instances

$$\{\neg\phi(t_1), \neg\phi(t_2), \dots, \neg\phi(t_n)\}$$

is consistent. This means that one can assign truth values to the atomic sentences so that all sentences $\neg\phi(t_1), \neg\phi(t_2), \dots, \neg\phi(t_n)$ are true. By the compactness of propositional calculus (page **??**), the set of *all* sentences $\neg\phi(t)$ is consistent. Hence there exists a truth assignment A to atomic sentences such that *all* sentences $\neg\phi(t)$ are true. Using A, we construct a term model of the sentence $\forall x \neg\phi(x)$ in the same way as in the proof of the completeness theorem. Namely, the elements of the model are terms, a function symbol $f(x_1, \dots, x_k)$ is interpreted as the mapping $t_1, \dots, t_k \mapsto f(t_1, \dots, t_k)$ and, for a relation symbol $R(x_1, \dots, x_l)$, the corresponding relation is true for terms $s_1, \dots, s_l$, if and only if the atomic formula $R(s_1, \dots, s_l)$ is true in the assignment A.

Thus from the assumption that no disjunction of the form (6.1) is a propositional tautology, we have derived the existence of a model in which $\exists x\ \phi(x)$ is false, hence cannot be provable. This proves the theorem.

Herbrand's theorem for existential formulas with more existential quantifiers is essentially the same. The general form of Herbrand's theorem characterizes provability (which is the same as logical validity) of arbitrary sentences in prenex form (this means that all quantifiers precede the other parts of the formula). The general case is reduced to the existential case by eliminating universal quantifiers by means of new function symbols. This is the dual transformation to skolemization that I mentioned in Chapter 2. Consider as an example a sentence of the form

$$\exists x \forall y \exists z \forall u\ \phi(x, y, z, u).$$

The *herbrandization* of this formula is

$$\exists x \exists z\ \phi(x, F(x), z, G(x, z)).$$

A sentence in the prenex form is provable if and only if its herbrandization is. This is an immediate consequence of the fact that a sentence is consistent (has a model) if and only if its skolemization is consistent. Thus the same proof as above gives us that the sentence $\exists x \forall y \exists z \forall u\ \phi(x, y, z, u)$ is provable if and only if a disjunction of the form

$$\phi(t_1, F(t_1), s_1, G(t_1, s_1)) \vee \phi(t_2, F(t_2), s_2, G(t_2, s_2)) \vee \dots \phi(t_n, F(t_n), s_n, G(t_n, s_n))$$

is a propositional tautology for some terms $t_1, \dots, t_n, s_1, \dots, s_n$. Notice that the function symbols F and G may occur also inside of terms $t_1, \dots, t_n, s_1, \dots, s_n$.

One can also state a general form of Herbrand's theorem without additional function symbols. In general one has to state a rather complicated condition on terms, but for the prefix $\exists\forall\exists$ it is fairly simple: a sentence of the form

$$\forall x \exists y \forall z\ \phi(x, y, z)$$

is provable if and only if a disjunction of the form

$$\phi(x, t_1(x), y_1) \vee \phi(x, t_2(x, y_1), y_2) \vee \dots \phi(x, t_n(x, y_1, y_2, \dots, y_{n-1}), y_n)$$

is a propositional tautology for some terms $t_1, \dots, t_n$ that may only contain the variables explicitly shown.

The reflection principle implies consistency. ♠ Assume that the reflection principle is true. Apply it to a formula that expresses contradiction, say $\beta \wedge \neg\beta$. By contraposition the implication is equivalent to:

$$\neg(\beta \wedge \neg\beta) \to \neg Pr_T(\beta \wedge \neg\beta).$$

Since the antecedent is the negation of a refutable formula, it is provable. Thus we get the consequent

$$\neg Pr_T(\beta \wedge \neg\beta).$$

This formula expresses the fact that a contradiction is not provable in the theory T, which is the consistency of T.

Parikh's proof. The proof that there is no short proof of contradiction in Parikh's theory FPA_t has two main steps. Suppose a proof d is a short proof of contradiction.

In the first step we replace d by a direct proof d', which can be, for example, a Herbrand disjunction. The direct proof d' can be fairly large; we can only bound it by a superexponential function in the length of d. This is the reason why one has to take the term t so that it defines a large number.

Let m be the number that t defines (the value of t). The second step is a proof that any direct proof of contradiction must have size at least m. This is proven by showing that in a direct proof of contradiction there must be a term with value n for every n, $0 \leq n < m$. The basic idea is that one can obtain a contradiction only using term instances of the axiom

$$F(x) \to F(x+1)$$

and propositional logic. If there is no term s in the proof whose value is n, for $0 \leq n < m$, then we can interpret $F(n)$ as the formula $x \leq n$. With this interpretation (and the natural interpretation of the arithmetical operations and the inequality relation) all term instances of the axioms occurring in the proof become true. Hence the proof cannot contain a contradiction.

Since there are at least m different terms in the proof d', its length is at least m. This implies that if k is the length of d, then $supexp(k) \geq m$. This gives the required lower bound on the length of d.

Proofs, trees and ordinals. When rules with infinite numbers of premises are used, the structures of the proofs are infinite well-founded trees. A tree is called *well-founded* if it only has finite branches. This makes perfect sense, because we want to derive sentences from axioms that are only at the leaves of the tree. If there were infinite branches, there would be deductions that were not based on any axiom.

A well-founded tree has a *height*, which is a countable ordinal defined inductively as follows. A tree consisting of a single vertex, the root, has height 0. If α_i, for $i \in I$, are the heights of the subtrees attached to the root, then the height of the tree is $\sup_{i\in I}(\alpha_i + 1)$.

xxxxxxxxxxxxx Figure: A tree of height 2ω xxxxxxxxxxxxxxxxxxxxx ♠

Theorems with short nonelementary proofs. There are many examples of theorems that have shorter nonelementary proof than all known elementary proofs. In Chapter 1 I mentioned the Prime Number Theorem (page **??**). This theorem was first proved using analytic means, and only later was an elementary proof found. This is quite a typical example, in many cases first a nonelementary proof is found and later an elementary, but longer, proof is also discovered.

Another area where problems are stated using elementary concepts, but are sometimes solved using nonelementary means, is finite combinatorics. A celebrated result of László Lovász, his proof of *Kneser's Conjecture*, is such an example. The conjecture (which is now a theorem of Lovász) is the following statement:

Theorem 6.1 *Let n and k be natural numbers such that $n \geq 2k$. Let $C_1 \cup C_2 \cup, \ldots, \cup C_{n-2k+1}$ be a partition of k-element subsets of an n-element set. Then in one class C_i there are two disjoint sets.*

Lovász solved the problem using the result in topology called Borsuk's Theorem:

Theorem 6.2 *An antipodal continuous mapping from an n-dimensional sphere to an m-dimensional sphere exists only if $m \geq n$.*

An antipodal mapping is a mapping that maps pairs of opposite points to pairs of opposite points.

Kneser's Conjecture was stated in 1955 and was proven in 1978, but only in 2003 was an elementary proof found by Jiří Matoušek.

In logic there are several theorems that can be proven either using proof theory, where the proofs are longer, or model theory, where the proofs are shorter. An example is Herbrand's theorem.

Strictly speaking, it is not clear that the nonelementary proofs are shorter in the formal sense considered in this section. We would have to say explicitly what the theory is in which the nonelementary proof is formalized.

Consider, for example, Kneser's Conjecture. The nonelementary proofs are short, but refer to Borsuk's Theorem. Should we consider Borsuk's Theorem as an axiom? This would not be fair, but if we do not, then we have to take into account also the length of the proof of Borsuk's Theorem.

Ideally we would like to have an example of a theorem that has a long proof in Finite Set Theory (Zermelo-Fraenkel Set Theory without the axiom of infinity) and a short proof in Zermelo-Fraenkel Set Theory. I am not aware of any example of a mathematical result of this type. One problem is that a lot of methods that use infinite sets can be simulated in Finite Set Theory. Another is that deriving basic algebraic, geometric and topological results requires quite lengthy proofs.

But even if we had such a theorem, the most difficult problem would still be there—to prove that an elementary proof cannot be short. We are only able to prove that for contrived theorems.

6.2 Theories, complexity classes and propositional proofs

— **A** —

The relationship between proofs and computations has been studied since the time the fields of proof theory and computability theory started. A number of interesting results connecting proofs with computations have been proved. Among the logical calculi that can be used for formalizing computation the most prominent is the λ-calculus of Church. λ-calculus is a basis of functional programming languages, such as *Lisp*. I mentioned λ-calculus in Chapter 3 as a theory that was originally proposed as an alternative approach to the foundations of mathematics instead of set theory, but over the course of time, the computational nature of it turned out to be more important. Another important calculus related to computability is the calculus of intuitionistic logic. (As a matter of fact, there is a close connection between λ-calculus and intuitionistic logic, but it would take us too far afield to explain it.)

After the advent of computational complexity theory it was natural to look for logical calculi that could be related to polynomial time computations. At that time Parikh (in the same paper I mentioned some pages back) proposed *"an anthropomorphic system"* PB based on on Peano Arithmetic with induction restricted to a subclass of formulas whose validity is efficiently testable.[4] In Parikh's theory PB induction is restricted to bounded arithmetical formulas, formulas in which the range of quantification is bounded to finite intervals. For such formulas one can algorithmically decide their satisfiability in the domain of natural numbers (the algorithm runs in linear space). An important property of PB, as well as of many theories studied subsequently, is that it is not provable that for every x there exists 2^x, in other words, the exponential function is not defined on all numbers. These two properties led Parikh to the conclusion that such a system is closer to what humans can actually do, if we view it from the philosophical point of view, and closer to the research in complexity theory, if view it from the mathematical point of view.

Only a little later, in 1975, Cook introduced his system PV of *polynomially verifiable* identities. His aim was to design a system that was somehow connected with polynomial time computations. Building on these two formal systems, more theories were introduced in the work of Jeff Paris, Alex Wilkie, Samuel Buss and others. Although different, the essence of these systems is the same.

Another line of research that started at about the same time concerned proving lower bounds on the lengths of proofs in propositional calculus. This apparently unrelated subject has a very close connection to the theories mentioned above. The connection was discovered by Cook and described in the same paper in which he introduced his theory PV. It is the fact that proofs of sentences of a certain type can be translated into at most polynomially long proofs of sequences of tautologies in the propositional calculus. This in principle enables one to prove the independence of some sentences by proving lower bounds on tautologies in the propositional calculus. Thus the logical problem of proving independence is reduced to a combinatorial problem of studying the lengths of propositional proofs. It has turned out that proving lower bounds on propositional proofs is a very hard problem, so this connection has not produced many independence results. However, it helped us more fully understand the essence of the problems concerning these theories and the propositional calculus.

[4]The notation PB has never been used afterwards, instead the theory was denoted by $I\Delta_0$, where I stands for *induction* and Δ_0 for bounded arithmetical formulas.

Originally the term *'Proof Complexity'* was only used for the subfield that studies *lengths of proofs*. As the connections between theories and the propositional calculus became more apparent, the term acquired much wider meaning and it is now used for the whole field that includes not only the complexity of propositional proofs, but also the study of theories related to feasible computations.

Theories corresponding to complexity classes

One of the key ideas of Proof Complexity is that it is possible to define theories that in some sense correspond to complexity classes. The link is not mere formal similarity, it is based on certain relations between a complexity class and the corresponding theory. We are not able to construct such a theory for every complexity class, but there are a number of important complexity classes for which theories have been defined which relate closely to their corresponding classes. Here I will only outline the basic idea of this correspondence.

As we are mostly interested in complexity classes that are near the bottom of the hierarchy, we may view the corresponding theories as subtheories of Peano Arithmetic. Recall that Peano Arithmetic is axiomatized by two sets of axioms:

1. *Basic Axioms.* These axioms state a few basic properties of the constant 0 and the arithmetical operations. Specifically these are the following axioms (where, for the sake of simplicity, I am omitting the axioms for the less-than-or-equal relation $\leq$):

$$\begin{array}{lll} S(x) \neq 0 & x + 0 = x & x \times 0 = 0 \\ S(x) = S(y) \rightarrow x = y & x + S(x) = S(x + y) & x \times S(y) = S(x + y) \\ x \neq 0 \rightarrow \exists y(x = S(y)) & & \end{array}$$

2. *The Schema of Induction.* This is an infinite set of axioms, one axiom for every arithmetical formula with free variable x, that states the principle of induction for this formula. Formally it is expressed by

$$(\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(S(x)))) \rightarrow \forall x\ \phi(x),$$

where $\phi(x)$ is the arithmetical formula. In other words, we state the principle of induction in the maximal possible strength that we can achieve using only the language of arithmetical operations.

Let us also recall that Peano Arithmetic is in a well-defined sense equivalent to Finite Set Theory. The theory is fairly strong, in particular, essentially all mathematical results concerning numbers and finite structures can be formalized and proven within the framework of Peano Arithmetic and Finite Set Theory.

Now suppose we want to prove theorems about sets and functions from a certain class. Clearly, we will need the induction principle at least for sets from this class. Specifically, if we are interested in a complexity class $\mathcal{C}$ and the sets in this class are exactly those that can be defined by formulas from a class Φ, then we should postulate the induction axiom for every formula from the class Φ. The key idea is to study theories in which we state induction *only* for these formulas. In other words, we take something like the minimal theory that is able to describe the objects we are interested in and to prove general theorems about them.

Given a complexity class $\mathcal{C}$, I will use the following symbol

$$\Theta_{\mathcal{C}}$$

to denote the theory in which induction is restricted to sets of complexity $\mathcal{C}$. This is a rather informal notation for several reasons. In order to define this theory precisely, we need to specify a class of formulas Φ that represents the sets in $\mathcal{C}$. The language of basic arithmetical operations is usually not rich enough to get a natural class of such formulas, thus we must use more function and predicate symbols. Then we also have to extend the set of basic axioms in order to fix the interpretations of the new symbols. All these things are, however, just technicalities that we can ignore in our informal description of the theories. Thus the structure of the set of axioms of $\Theta_{\mathcal{C}}$ is essentially the same as for Peano Arithmetic:

1. *Basic Axioms.* A finite set of axioms concerning the primitive notions. In particular we can always include the basic axioms of Peano Arithmetic.

2. *The Schema of Induction for the class of formulas* Φ. An infinite set of axioms—one axiom for every formula of Φ with free variable x that states the principle of induction for this formula.

The theory PB introduced by Parikh was Peano Arithmetic with induction restricted to formulas with bounded quantifiers. We call a quantifier bounded if the quantified variable x occurs in a context of the form $x \leq t$ for some term t not containing x. Informally, the range of quantification is restricted to the finite domain of numbers between 0 and t. The theories introduced by Buss were also based on induction for bounded formulas, except that he used a richer language. Since complexity classes are always defined by bounding resources, the classes of formulas defining them consist of bounded formulas in some language. Therefore we use the term *Bounded Arithmetic* to denote all theories constructed in this way.

The schema described above enables us, in principle, to define the theory $\Theta_{\mathcal{C}}$ for every complexity class $\mathcal{C}$. But in order to view this theory as associated with the complexity class, we want to have closer relations between these two objects. For a number of complexity classes such relations have been shown. I will say more about these relations shortly.

Let us observe one particular similarity between complexity classes and these theories. A complexity class is defined by *restricting computational resources.* A Bounded Arithmetic theory is defined by *restricting provability resources.* We will see other similarities of this type in the following subsections.

Several reasons why studying weak theories is important

According to the Incompleteness Theorem, there are unprovable true sentences in Peano Arithmetic. We can make the theory stronger by postulating induction for a larger class of formulas, but such a theory will still be incomplete. By stating induction for a subclass of arithmetical formulas we are making things worse—the theories are even more incomplete. In fact, the theories that we will consider will be much weaker than Peano Arithmetic. Although we can never fully determine a structure such as the natural numbers with arithmetical operations, we can at least try to make the theory as strong as possible. Thus it may seem strange to do the opposite, to work with a restricted set of axioms, but there are good reasons for this. One should realize that by defining theories of Bounded Arithmetic we are not trying to propose theories that should serve as "foundations for arithmetic". We are defining new objects of study—theories of a particular kind. The rule of thumb is that the weaker the theory is the more one can prove about it. Thus considering weak theories is, in fact, an advantage.[5]

Now I will briefly review some reasons why researchers are interested in these theories. Then in the following subsections I will describe the theories in more detail.

1. It is possible to study problems that concern theories and that are analogous to the problems about complexity classes.

Let us consider an example. As sketched above, one can define theories for some complexity classes, in particular this is possible for **P** and **NP**. So let $\Theta_{\mathbf{P}}$ and $\Theta_{\mathbf{NP}}$ be the theories corresponding to the complexity classes **P** and **NP**. Then we can ask the question that corresponds to the fundamental problem **P** vs. **NP** in the realm of theories:

Do the theories $\Theta_{\mathbf{P}}$ *and* $\Theta_{\mathbf{NP}}$ *prove the same theorems?*

I will state it more succinctly as the question

$$\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}} \ ?$$

Most problems in complexity theory are stated as a question about whether or not two complexity classes are the same. Similarly, the main open problems in proof complexity ask for proving or disproving that two theories are equivalent. Although they concern different concepts, the stumbling blocks preventing us from solving these problems seem to be very similar. Thus we are led to the conclusion that logic must play an important role not only in proof complexity, but also in computational complexity. Therefore it seems conceivable that our inability to make any progress in solving problems in computational complexity may be caused not by their mathematical complexity, but by some obstacles

[5] Yet the theories are still too strong if one wants to apply the methods of main-stream model theory. Thus one has to develop methods specific for this field.

of a logical nature. If this feeling is right, the study of theories of Bounded Arithmetic should help us solve both types of problems.

2. It is possible to characterize functions from some complexity classes as functions that can be formalized in the corresponding theories.

This is an important example of a *formally provable* relation between theories and complexity classes. Let us consider an example again. $\Theta_{\mathbf{P}}$, the theory associated with the complexity class $\mathbf{P}$, has the following constructive property. Roughly, if we prove the existence of a number satisfying some polynomial time condition, then we can construct this number in polynomial time. More precisely, if for some polynomial time relation R we prove that for every x there exists y such that x is in relation R with y, then one can find an algorithm that for every x constructs such a y in polynomial time.

Notice that it is unlikely that in general such algorithms should exists for all polynomial time computable relations of this type. Thus the provability in $\Theta_{\mathbf{P}}$ guarantees us more than if we only knew that for every x such a y existed.

3. It is natural to argue about complexity classes using concepts of the same nature.

When Parikh defined his theory PB, he was motivated by such considerations. He also observed that in his theory it is not possible to prove that for every number n, there exists the number 2^n, that is, the exponential function cannot be formalized. Thus in proving a sentence about n one cannot use numbers much larger than n. We know that this may prevent us from using nonelementary arguments, such as arguments using probability and algebra, which are quite common in combinatorics and number theory. But our aim is not to use these theories to prove new theorems, rather to use them to understand what methods are needed for particular theorems.

Incidentally the fact that the exponential function is not formalizable is a common property of all theories associated with complexity classes below the exponential time class $\mathbf{EXP}$.

Another example is Cook's theory PV. Cook's idea is that as $\mathbf{P}$ renders very well the intuitive concept of efficiently computable functions, we should also have a similar concept concerning proofs. I will explain his theory in more detail shortly.

4. We want to determine the axioms needed for proving that a given algorithm does what it is expected to do.

When we are studying a computational problem from the point of view of complexity, we ask what is the least time, space etc., that an algorithm needs for solving the problem. We thus consider various computational resources needed for computation. But an algorithm is not just a piece of code in a programing language, there is also an *idea* (or several ones) on which it is based. If are to design a correct algorithm for a problem, we have to use the definition of the problem and we have to apply some *ideas, methods, tricks, principles etc.* The latter words can be formalized as the *axioms needed for proving the correctness of the algorithm.* These are not computational resources, but they can also be viewed as a kind of complexity of the algorithm. It is natural to call them the *proof complexity* of the problem.

It is possible, for example, that for some problem P, we cannot solve P efficiently using algorithms based on simple arguments, while there exists an algorithm based on deeper mathematics that solves it efficiently. Thus we may have a trade-off between the time complexity and the complexity of axioms needed for proving the correctness of an algorithm for P. This shows that there exists an intrinsic connection between computational and proof complexities. In order to formalize this connection, we need a scale of formal systems. These formal systems need not only be theories formalized in predicate logic, but we will see that one can also use the propositional calculus.

— B —

Universal-P sentences and formalization of complexity classes

Although we are interested in all sentences provable in the theories that we study, some sentences are more important than others. I have already mentioned the set of sentences Π_1, which I also call *universal finite sentences.* These are sentences of the form $\forall x\ \phi(x)$, where in ϕ all quantifiers are bounded. Hence for a given x we can algorithmically decide whether $\phi(x)$ is true or false. Such sentences are *empirically testable:* we can test the validity of $\phi(x)$ for various values x, but we will never be sure that $\forall x\ \phi(x)$ is true (unless we prove the sentence). On the other hand, a Σ_1 sentence, which is of the form $\exists x\ \phi(x)$, with ϕ a bounded formula, is *empirically verifiable*—given an element a that satisfies $\phi(a)$, we can verify the truth of $\exists x\ \phi(x)$.

Algorithmic decidability is a very rough approximation of what actually can be computed. Having the concept of polynomial time computations, it is natural to consider the sentences $\forall x\ \phi(x)$ with the predicate $\phi(x)$ being *computable in polynomial time.* Thus we get a much better approximation of the intuitive concept of empirically testable sentences. Not surprising, these sentences play an important role in proof complexity. I will call them *universal-***P** *sentences* and use the notation

$$\Pi_{\mathbf{P}}$$

for the set of these sentences.

But this concept needs a more precise definition, otherwise it may lead to nonsensical conclusions. For instance, if a sentence of the form $\forall x\ \phi(x)$, where ϕ is *arbitrary*, is true, then the satisfiability of $\phi(x)$ is trivial, in particular, it is decidable in polynomial time. Applying the above definition literally, we would conclude that every true sentence starting with a universal quantifier is universal-**P**.

The point is that we must know *a priori* that ϕ is decidable in polynomial time. More precisely, every theory T to which we apply this concept should be "aware" of the fact that ϕ is decidable in polynomial time. This means that if we formalize Turing machine computations, then for some Turing machine M, T is able to prove that M accepts exactly those strings or numbers that satisfy ϕ and M runs in polynomial time.

So the main question is how one should pick the class $\Phi_{\mathbf{P}}$ of formulas that define sets in **P**. A simple solution is to take the formulas that define Turing machines with built-in clocks that ensure polynomial bounds on their running time. This suffices for formalizing $\Pi_{\mathbf{P}}$ and some other concepts that we will need. However, if we used it to define the theory $\Theta_{\mathbf{P}}$ we would run into problems. In order to prove properties of Turing machines we would need to use induction, but the induction axioms would already be stated in terms of Turing machines. For defining $\Theta_{\mathbf{P}}$ it is better to use a class defined by simple syntactical conditions. Such a class can be defined if we are willing to use a language with infinitely many primitive symbols. This is the approach taken in the theory PV. For languages with finite sets of primitives, there is another solution. One can take a class of formulas Γ that defines only a subclass of **P**, but produces induction axioms that are strong enough to prove induction for all predicates in **P** (see part C).

All this also concerns sets of formulas $\Phi_{\mathcal{C}}$ for other complexity classes $\mathcal{C}$. For some complexity classes, for example for **NP**, it is easy to define a class of formulas that is syntactically simple, but for others it may be even more difficult than for **P**.

It is clear that I am using the symbols $\Phi_{\mathcal{C}}, \Theta_{\mathcal{C}}$ and $\Pi_{\mathbf{P}}$ as generic names for sets that can be specified in various ways. But in spite of the ambiguity of the definition of $\Phi_{\mathcal{C}}$, the concept of the associated theory $\Theta_{\mathcal{C}}$ is quite robust: the theorems that one can derive from various axiomatizations are the same. To stress the fact that I am considering theories T and S as the set of theorems provable therein, I will use the notation $T \equiv S$ to express that T and S are equivalent in that sense that they prove the same theorems.

Some relations between theories and complexity classes

It is interesting that we can formulate problems in proof complexity that are like problems in computational complexity, but one may wonder whether this similarity has deeper roots or is just a superficial analogy. I will explain, using the example of the problems **P** vs. **NP** and $\Theta_{\mathbf{P}}$ vs. $\Theta_{\mathbf{NP}}$, what we know about the relation between such pairs of problems.

Recall that the theories $\Theta_{\mathbf{P}}$ and $\Theta_{\mathbf{NP}}$ are defined by restricting the induction axiom schema to a set of formulas that define sets in **P** and to a set of formulas that define sets in **NP** respectively. Let these two sets of formulas be denoted by $\Phi_{\mathbf{P}}$ and $\Phi_{\mathbf{NP}}$. Naturally, we pick the classes so that $\Phi_{\mathbf{P}}$ is a subset of $\Phi_{\mathbf{NP}}$ and the basic axioms are the same. This ensures that $\Theta_{\mathbf{NP}}$ is an extension of, or is equal to, $\Theta_{\mathbf{P}}$, just as the class **NP** is an extension of, or is equal to, the class **P**.

Suppose that $\mathbf{P} = \mathbf{NP}$. This means that formulas of $\Phi_{\mathbf{P}}$ define all sets that are definable by formulas of $\Phi_{\mathbf{NP}}$. So for every formula $\phi(x)$ from $\Phi_{\mathbf{NP}}$ there exists an equivalent formula $\psi(x)$ in $\Phi_{\mathbf{P}}$. However, the equivalence $\phi(x) \equiv \psi(x)$ does not have to be provable in $\Theta_{\mathbf{P}}$. It may happen that to prove this equivalence one would need stronger axioms than are available in $\Theta_{\mathbf{P}}$. Therefore it is possible that $\mathbf{P} = \mathbf{NP}$, yet $\Theta_{\mathbf{P}}$ is not equivalent to $\Theta_{\mathbf{NP}}$.

Let us now consider the opposite relation. Suppose that $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$. Formally this means that for every formula $\phi(x)$ from $\Phi_{\mathbf{NP}}$, there exists a finite set of formulas $\psi_1(x), \ldots, \psi_n(x)$ from $\Phi_{\mathbf{P}}$ such that the induction axiom for $\phi(x)$ follows from the induction axioms for $\psi_1(x), \ldots, \psi_n(x)$ and the basic axioms. Notice that this does not say anything about a formula from the class $\Phi_{\mathbf{NP}}$ being equivalent to a formula from $\Phi_{\mathbf{P}}$. So again, we cannot exclude the possibility that $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$ and $\mathbf{P} \neq \mathbf{NP}$ hold true simultaneously. However, although we are not able to prove such a direct connection, the assumption $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$ does imply an inclusion relation between two complexity classes.

Theorem 6.3 *If* $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$*, then* $\mathbf{NP} \subseteq$ **nonuniform-P**.

Notice that the statement $\mathbf{P} = \mathbf{NP}$ is equivalent to $\mathbf{NP} \subseteq \mathbf{P}$, since we know that the other inclusion is true. Since also $\mathbf{P} \subseteq$ **nonuniform-P**, the conclusion of the theorem is a weakening of $\mathbf{P} = \mathbf{NP}$.

This theorem shows that there are provable relations between proof complexity and computational complexity. Further, it shows that it is reasonable to conjecture that $\Theta_{\mathbf{P}} \not\equiv \Theta_{\mathbf{NP}}$, since it follows from the generally accepted conjecture from computational complexity that $\mathbf{NP} \not\subseteq$ **nonuniform-P**.

Several more implications of this kind have been proven, but no implication in the opposite direction is known. In spite of not having consequences in complexity theory, proving $\Theta_{\mathbf{P}} \not\equiv \Theta_{\mathbf{NP}}$ would be an extremely important result in proof complexity, since it is one of the central problems in this field. But it would also be interesting from the point of view of computational complexity, as it would show that $\mathbf{P} = \mathbf{NP}$ cannot be proved using limited means.

Search problems

A search problem is given by a binary relation $R(x, y)$. The task is, for a given input x, to find y such that x is in relation R with y. We will be interested in the special kind of search problems where

1. *the relation R is computable in polynomial time,*
2. *the length of y is polynomially bounded by the length of x.*

This is a familiar situation—I explained the **P** vs. **NP** problem using search problems of this kind. However, if we are only interested in complexity classes, as we were when defining **P** and **NP**, we need only to decide whether or not the search problem has a solution for a given input x, we do not need to find a solution. So the **P** vs. **NP** problem concerns *decision problems*, whereas *search problems* concern actually finding y. The class of all problems satisfying 1. and 2. is called *Polynomial Search Problems* and I will abbreviate it by **PS**.

From the point of view of complexity, the class **PS** is not particularly interesting. One can easily prove that all **PS** problems are solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$. What is much more interesting is the subclass of *Total Polynomial Search Problems*, which will be abbreviated by **TPS**.[6] These are the **PS** problems that satisfy another condition:

3. *for every x, there exists y such that $R(x, y)$ holds true.*

They are called *total*, because an algorithm solving the problem determines a function f such that for all x, $R(x, f(x))$ holds true. In such situations we say that the function is *total* to stress that it is defined for all inputs, in contrast to *partial* functions defined only for some inputs. Notice that there are no natural decision problems associated with problems in **TPS**—since a solution always exists, there is nothing to decide.

[6]I deviate from the more common abbreviations **FNP** for **PS** and **TFNP** for **TPS**, since the I find the usual notation rather confusing.

Example. An example of a total polynomial search problem is the problem of finding a proper factor of a composite number. As stated it is not total, because prime numbers do not have proper factors. But since primality is decidable in polynomial time, we can make it total by stipulating that 0 is a solution for inputs that are primes.

Condition 3. is fundamentally different from the previous two. If we want to guarantee that an algorithm A for R runs in polynomial time, we can simply equip A with a step-counter and arrange it so that it always stops after the number of steps reaches a particular polynomial value. Similarly, we can guarantee that the length of y does not exceed some polynomial value. But if we are to produce a certificate that condition 3. is always true, we have to give a *proof* of the condition. For some **TPS** problems the proof may be easy, for some it may be hard. The particular measure of hardness that we will be interested in is the minimal theory in which condition 3. is provable.

This brings us back to the theories that are studied in proof complexity. But before talking about theories in Bounded Arithmetic, I need to recall a general phenomenon concerning incompleteness. In the section about concrete independent combinatorial sentences I mentioned that these sentences are connected with fast growing functions. The sentences were of the form

$$\forall x \exists y \ R(x, y),$$

where $R(x, y)$ was some computable relation; these are the Π_2 sentences. We can also view such sentences as total search problems, but, of course, not polynomial ones, because we do not bound the size of y. The particular sentences that I presented were not provable in Peano Arithmetic because, for increasing x, the least y satisfying the relation R was increasing so fast that Peano Arithmetic was not able to capture such a rate of growth. In general, every theory T (consistent and having a recursive set of axioms) can handle only a limited rate of growth. Thus we can always define a function that grows so fast that it is not possible to prove in T that the function is total. This enables us to use fast growing functions as a scale for measuring the strength of theories.

If measured by provably total functions, most of the theories of Bounded Arithmetic would be at the bottom of the scale. In a Bounded Arithmetic theory one can typically formalize only polynomial growth. More precisely, one can prove that a function is total only when the *length* of the function value grows at most as a polynomial in the length of the argument. But this does not mean that they are extremely weak if we consider sentences of low complexity. In fact, this is an advantage of studying these theories—we are naturally led to problems that concern low complexity classes. Namely, if a Π_2 sentence is provable in Bounded Arithmetic, then the length of y is polynomially bounded by the length of x, so condition 2. is automatically satisfied. It suffices then to assume condition 1. and we get a **TPS** problem.

Let us see what some particular theories can prove about **TPS** problems. Specifically, I will consider the theories $\Theta_{\mathbf{P}}$ and $\Theta_{\mathbf{NP}}$. In what follows I will always assume that R is a binary relation computable in polynomial time (condition 1). The next theorem, due to S.R. Buss, expresses formally the relation between $\Theta_{\mathbf{P}}$ and **P** that I explained in paragraph 2. of the previous subsection.

Theorem 6.4 *If a sentence $\forall x \exists y \ R(x, y)$, with $R(x, y)$ representing a polynomial time computable relation, is provable in $\Theta_{\mathbf{P}}$, then the associated search problem is solvable in polynomial time, that is, there exists a polynomial time algorithm that for every x constructs y such that $R(x, y)$.*

Theorems of this type are called *witnessing theorems*, because one can efficiently find a witness for the existential quantifier $\exists y$, given x as input.

The class of search problems solvable in polynomial time is denoted by **FP**. Thus the theorem says that the problems for which $\Theta_{\mathbf{P}}$ can verify condition 3. are from the class **FP**. One can also show the opposite: every problem in **FP** can be formalized in such a way that $\Theta_{\mathbf{P}}$ proves condition 3.

Since **FP** is a sort of a functional version of the class **P**, we get another reason for associating the theory $\Theta_{\mathbf{P}}$ with the complexity class **P**.

In order to state the corresponding result for $\Theta_{\mathbf{NP}}$, we need to define another class of search problems, *Polynomial Local Search*, abbreviated by **PLS**. This class was introduced by D.S. Johnson, C.H. Papadimitriou and M. Yannakakis. A **PLS** problem S is defined as follows. Given an input string a we have

1. a set U_a;

2. a subset $F_a \subseteq U_a$ whose members are called *feasible solutions* (but they are not necessarily solutions of S); we assume that a fixed element, say 0, is always in F_a;

3. a *neighborhood function* h_a that maps U_a into itself;

4. and a *cost function* c that assigns a rational number to every $x \in U_a$ (we may assume that c is the same function for all inputs a).

To motivate the definition of solutions of S, I will first give an interpretation of these notions. We think of feasible solutions as potential solutions and we are interested in their cost. We do not insist on finding a solution with a maximal cost. Instead, we are satisfied with a solution that cannot be improved using the neighborhood function. Thus the neighborhood function should be thought of as a heuristic that helps improve the cost of a feasible solution. We always have some feasible solution, namely 0, and we can try to improve the cost of a feasible solution that we have by applying the function h to it. Since the set of feasible solutions is finite, we cannot go on improving the cost for ever, the process has to terminate. There may be two reasons for stopping at some feasible solution x: (1) $h_a(x)$ is not a feasible solution, or (2) the cost does not increase, that is, $c(h_a(x)) \leq c(x)$. Such an x is defined to be a solution of the search problem S.

Now let me state it formally. An x is a *solution* of S for input value a if

1. *x is a feasible solution, that is, $x \in F_a$, and*

2. *$h_a(x)$ is not a feasible solution, or $c(h_a(x)) \leq c(x)$.*

It should be stressed that we do not require x to be obtained by the process described above. I used the process only to motivate the definition. Also notice that a solution x does not have to be a feasible solution with maximal cost; the definition only states that it cannot be improved using the heuristic h.

As we are defining a class of **TPS** problems, we require that the primitives by which a **PLS** problem is defined are computable in polynomial time. In particular, the set U_a is a set of all strings of length n, where n is polynomial in the length of the input a, one can decide in polynomial time if $x \in F_a$ and one can compute in polynomial time the functions h_a and c.

Example. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ♠

Furthermore, we want **PLS** to be a true complexity class. As defined so far, it is an interesting class of search problems, but it is not closed under polynomial reductions. Let us, therefore, stipulate that **PLS** is the class of all total search problems that can be polynomially reduced to a search problem defined above.

The concept of polynomial reducibility of search problems is similar to the polynomial reducibility of sets mentioned in Chapter 5. Suppose we have two polynomial search problems P and Q, defined by relations $r(x, y)$ and $s(x, y)$ respectively. We say that *P is polynomially reducible to Q,* if there exists a polynomial time algorithm that for a given a computes b such that $r(a, b)$ *using queries to the problem Q*. This means that during the computation the algorithm may compute some c and request d such that $s(c, d)$. It may ask several such queries, and if they are answered correctly, it must produce a correct answer (that is, b such that $r(a, b)$).

The natural algorithm to compute a solution of a **PLS** problem which I used to motivate the definition does not help us to compute solutions efficiently. The size of U_a can be exponentially large and the number of values of the cost function can be exponential. Thus the natural algorithm typically runs in exponential time. It is a generally accepted conjecture that there are **PLS** problems that cannot be solved in polynomial time. On the other hand we also believe that **PLS** does not exhaust all total polynomial search problems, in fact, it seems to be just a small subclass.

After a rather lengthy description, we can return to our theory $\Theta_{\mathbf{NP}}$ and at last state a characterization of the total search problems that can be formalized in it. I will say that a search problem P defined by a relation $r(x, y)$ is *'properly formalized'* in a theory T if

1. the relation $r(x, y)$ is formalized by a formula from $\Phi_{\mathbf{P}}$ and

2. T proves that the search problem is total (condition 3. of the definition of **TPS**).

Theorem 6.5 *If a sentence $\forall x \exists y\ R(x,y)$, with $R(x,y) \in \Phi_{\mathbf{P}}$, is provable in $\Theta_{\mathbf{NP}}$, then the associated search problem is in* **PLS***.*

Less formally:

> *Every* **PLS** *problem that can be properly formalized in $\Theta_{\mathbf{NP}}$ is in* **PLS***.*

Again, one can also prove that every **PLS** problem can be properly formalized in $\Theta_{\mathbf{NP}}$. Thus **PLS** characterizes the search problems properly formalizable in $\Theta_{\mathbf{NP}}$.

The two characterizations give us another relation between the theories and the complexity classes.

Theorem 6.6 *If $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$, then* $\mathbf{FP} = \mathbf{PLS}$*.*

While the assumption is the same, the conclusion of the theorem is different from the conclusion of Theorem 6.3 stated in the previous subsection. We do not know if one theorem follows from the other.[7]

The theorem above gives us some provable connection between the theory $\Theta_{\mathbf{NP}}$ and a complexity class of search problems, but does not relate the theory $\Theta_{\mathbf{NP}}$ with complexity the class **NP**. In order to obtain such a connection we have to generalize search problems. Noticing that polynomial time is mentioned in condition 1. of the definition, we would like to replace it by a condition in which **NP** is used instead of **P**. Changing the definition in the most straightforward way does not work, so we have to introduce another concept.

We will consider computations that run in polynomial time and have free access to an **NP** set. One can imagine such algorithms as defined by enriching the set of elementary instructions with a nonelementary test "$x \in A$?" for a set $A \in \mathbf{NP}$ and assuming that to perform the test we only need one unit of time. Since testing $x \in A$? is an operation that an ordinary human cannot perform, we call it an *oracle*. Using such computations we define the class $\mathbf{P}^{\mathbf{NP}}$ as the class of decision problems computable in polynomial time with free access to **NP**.

Now we can also define the class of *Total* $\mathbf{P}^{\mathbf{NP}}$ *Search Problems*, denoted by **TNPS**, by replacing condition 1. in the definition of **TPS** by

1′. *the relation R is in* $\mathbf{P}^{\mathbf{NP}}$*.*

So everything is the same as in **TPS**, except that we consider a more general class of computations, computations that have free access to **NP** sets. Now I can state the version of Theorem 6.4 for $\Theta_{\mathbf{NP}}$, which is also due to Buss.

Theorem 6.7 *If a sentence $\forall x \exists y\ R(x,y)$, with $R(x,y)$ representing a $\mathbf{P}^{\mathbf{NP}}$ relation, is provable in $\Theta_{\mathbf{NP}}$, then there exists a polynomial time algorithm that uses a set in* **NP** *as an oracle such that, for every x, it constructs y such that $R(x,y)$.*

This theorem differs from Theorem 6.4 only in allowing the more general class of computations.

As in Theorem 6.4, one also gets the opposite—the theory $\Theta_{\mathbf{NP}}$ is able to formalize polynomial time algorithms with **NP** oracles. Let me also introduce $\mathbf{FP}^{\mathbf{NP}}$, the functional version of $\mathbf{P}^{\mathbf{NP}}$. Then we can restate the theorem as follows: the problems of **TNPS** that are provably total in $\Theta_{\mathbf{NP}}$ are exactly those in the class $\mathbf{FP}^{\mathbf{NP}}$.

In this way we get an almost direct connection between the theory $\Theta_{\mathbf{NP}}$ and the class **NP**. This relation does not determine $\Theta_{\mathbf{NP}}$ uniquely from **NP**, but it is a good indication that our choice of theory is good.

The two examples, Theorems 6.4 and 6.7 suggest a paradigm that one can use for finding more pairs of complexity classes and associated theories. Indeed, associated theories have been constructed for many important complexity classes.

Bounded Arithmetic is a natural set up for studying search problems, but we can ask the same question for arbitrary theories, in particular for theories that are not based on postulating induction

[7] If we have the correspondence $\mathbf{P} \leftrightarrow \Theta_{\mathbf{P}} \leftrightarrow \mathbf{FP}$, and $\mathbf{NP} \leftrightarrow \Theta_{\mathbf{NP}} \leftrightarrow \mathbf{PLS}$, the curious reader may wonder what we can get from it about the relations between the complexity classes. Namely, we have the following correspondences $\mathbf{P} \leftrightarrow \mathbf{FP}$, and $\mathbf{NP} \leftrightarrow \mathbf{PLS}$. The first one is easily explicable: one is a class of decision problem, the other is a function class with the same bounds on the computational resources. In the case of $\mathbf{NP} \leftrightarrow \mathbf{PLS}$, however, I do not have any explanation that does not mention $\Theta_{\mathbf{NP}}$.

for a certain class of formulas. We only need the theory to be strong enough to formalize polynomial time computations. We can even take strong theories such as Zermelo-Fraenkel Set Theory. Each such theory defines a subclass of search problems of **TPS** thus it seems that we could measure the strength of theories by these search-problem-complexity classes. But there are two obstacles: first, we are not able to prove that the hierarchy of search problems increases; second, we do not have explicit descriptions of these classes, except for the theories at the very bottom. Still, the mere fact that there is such a possibility seems very interesting. Let me, therefore, state this fundamental fact explicitly.

> *It is possible that one could measure the strength of theories by complexity classes contained in* **TPS**.

I will say more about it in the last section of this chapter.

Feasibly constructive proofs

Let us now return to the beginnings of Proof Complexity in the 1970's. After defining the classes **P** and **NP**, Cook wondered whether there is something in logic that would correspond to the class **P**—once we accept polynomial time computations as a natural formalization of feasible computations, it seems that there should be some concept representing feasible proofs in a similar way. Polynomial time computations are computations that perform a polynomial number of elementary steps in order to produce the result. Thus the corresponding concept in logic should be proofs consisting of a polynomial number of elementary steps, so we need only to define elementary proof steps. The question, however, is not as easy as it looks at first glance.

In 1975 Cook introduced a system intended to formalize feasible proofs. It was in the seminal paper *Feasibly constructive proofs and the propositional calculus,* which I consider to be the founding paper of proof complexity, for all the fundamental ideas of this field are more or less explicitly present in it. Cook called his formal system PV, for *Polynomial Verifiability.* I am not going to describe the details of the system PV, I will rather focus on explaining the motivation behind some particular concepts used in it.

Let us put ourselves in Cook's position back in the 1970s and try to devise such a formal system. The first thing we have to do is decide what formulas such a system should use. It is natural to require the following properties from the system:

1. *the sentences should have low logical complexity;*

2. *the system should be able to formalize basic statements about feasible computations.*

With hindsight, we guess that these should be universal-**P** sentences. But we have to be more specific about how we represent polynomial time computable sets. Since we want to formalize polynomial time algorithms, a natural thing is to have a name for every such algorithm. Therefore in PV there is a term $t(x_1, \dots, x_k)$ for every polynomial time algorithm that has k numbers as inputs. The converse is also true: every term defines a function computable in polynomial time. Although PV formally talks about natural numbers, it treats them, in fact, as binary strings. Thus, in fact, there is a term for every polynomial time algorithm whose input and output values are binary strings.

In PV all formulas are equations of the form

$$s(x_1, \dots, x_k) = t(x_1, \dots, x_k)$$

with s and t terms. The meaning is that the equation holds for all natural numbers $x_1, \dots, x_k$, which is written in predicate logic as $\forall x_1 \dots \forall x_k \; s(x_1, \dots, x_k) = t(x_1, \dots, x_k)$ but, as usual in equational theories, we omit the universal quantifiers in PV. Using such formulas, we are able to express that two algorithms define the same function. Thus one obtains a particular formalization of $\Pi_{\mathbf{P}}$ sentences.

Proceeding further to the axioms and rules of the system, our next goal is to satisfy the following requirement:

3. *the nature of proofs should also be feasible.*

It is much harder to give a more precise meaning to this vague statement, although intuitively it seems reasonable. Here comes another useful idea—*polynomial verifiability.* The idea behind this concept is:

Given an equality with a feasible proof, one should be able to verify the equality in a polynomial number of steps.

Using this concept we can specify the system that we are trying to design as the strongest system in which all provable equalities are polynomially verifiable.

So what is needed now is a definition of the polynomial verifiability of PV equations. A naive definition of the polynomial verifiability of $s(x) = t(x)$ would be to say that we can verify in polynomial time that $s(N)$ represents the same number as $t(N)$ for every given number N. But this does not say anything about provability, since every true equality of this form is verifiable in this sense (we can simply run the polynomial time algorithms corresponding to s and t on the input N and test if they produce the same number). In order to obtain a nontrivial concept one has to add a condition about the uniformity of such verifications. Specifically, for a given input length n the verification of $s(N) = t(N)$ for *all* numbers N of length n should have the same form. When formalized, it means that for every input length n there exists *one* verification that works for all N of length n. In such a verification the number is represented by a *variable* x, so technically it is a proof of a general statement, except that the domain of the variable is finite (the set of numbers of length n). What we gain by restricting x to numbers of length n is that, essentially, we do not need any general axioms in such proofs.

These ideas cannot be fully understood without seeing at least some details of the formalism. So let us start with the language of PV. It contains basic arithmetical operations and constants, but on top of these also several less common primitives. What will only be important for us are the two successor functions $s_0(x)$ and $s_1(x)$ and the trimming function $tr(x)$. The intended interpretations of them are:

$$s_0(x) = 2x, \quad s_1(x) = 2x + 1, \quad tr(x) = \lfloor x/2 \rfloor,$$

where $\lfloor y \rfloor$ is the largest integer less than or equal to y. Notice that these are bit operations on the binary representations of the number x: s_0 adds 0 to the binary representation of x, s_0 adds 1 to the binary representation of x, and $tr(x)$ trims off the last bit (except for 0, where $tr(0) = 0$). As usual, one can form compound terms from the primitive function symbols and constants.

Furthermore, one can introduce new function symbols that represent functions defined by *recursion on notation*. Given functions $f(y), g_0(x, y)$ and $g_1(x, y)$, we say that $h(x, y)$ is defined by recursion on notation, if it satisfies:

$$\begin{array}{rcl} h(0, y) & = & f(y) \\ h(s_0(x), y) & = & g_0(h(x, y), y) \\ h(s_1(x), y) & = & g_1(h(x, y), y). \end{array} \tag{6.4}$$

(The variable y plays the role of a parameter; for the sake of simplicity I am using only one parameter.) The advantage of this kind of recursion is that for a number N of length n and any number M, we need only $n + 1$ applications of the functions f, g_0, g_1 in order to compute the value $h(N, M)$. Indeed, if $a_n a_{n-1} \ldots a_1$ is the binary representation of N, then

$$N = s_{a_n}(s_{a_{n-1}}(\ldots s_{a_1}(0)\ldots)),$$

thus

$$h(N, M) = g_{a_n}(g_{a_{n-1}}(\ldots g_{a_1}(f(M), M)\ldots, M), M). \tag{6.5}$$

This suggests that if the initial functions f, g_0, g_1 are computable in polynomial time, then so should h be. In general this is not true, because it can happen that the numbers grow exponentially with the length of the first parameter. Therefore it is necessary to add a bound on the size of the values of h. This is just an annoying complication, so I will skip the technical details of how the growth of h is controlled in PV.

Let us now proceed to the axioms and rules. PV contains basic axioms about the primitive constants and functions. For example, we have the following axioms connecting the two successors with the trimming function:

$$tr(s_0(x)) = x, \quad tr(s_1(x)) = x.$$

When we introduce a new symbol h for a function defined by recursion on notation, we also add the corresponding equalities (6.4). The key axioms are the axioms of induction. In PV we are only allowed to use equalities, thus one cannot state induction in the usual form. There is a way to simulate induction using only equations of PV terms, but since I want to avoid technicalities, I will switch to an extension $PV1$ of PV in which one can use propositions made of equations.

The schema of induction in $PV1$ is motivated by recursion on notation, therefore it is called *Induction on Notation* (sometimes *'Polynomial Induction'* is also used). Translating recursion on notation literally, we obtain the following form of induction on notation:[8]

$$\phi(0,y) \wedge \forall x(\phi(x,y) \rightarrow \phi(s_0(x),y) \wedge \phi(s_1(x),y)) \rightarrow \forall x\ \phi(x,y),$$

where the formula $\phi(x,y)$ is an equality $s(x,y) = t(x,y)$ for some PV terms s and t. (Again I am simplifying it by only using one parameter y.) Instead of this, people prefer the more concise equivalent version:

$$\phi(0,y) \wedge \forall x(\phi(tr(x),y) \rightarrow \phi(x,y)) \rightarrow \forall x\ \phi(x,y).$$

If we state induction as an axiom (in either way) we cannot avoid the use of the universal quantifier in the antecedent. Therefore, in $PV1$ we represent the induction principle by a deduction rule:

from $\phi(0,y)$ and $\phi(tr(z),y) \rightarrow \phi(z,y)$, it is possible to derive $\phi(x,y)$.

Notice that the induction rule is the only general principle used in $PV1$; otherwise we only have axioms of equality, a finite set of axioms about the primitive concepts and the equations that determine the functions introduced by recursion on notation.

Now we can return to the concept of polynomial verifiability and explain it in more detail. Suppose we prove an equality $s(x) = t(x)$ and we want to verify it for numbers of length at most n. To this end we have to formally express that x has length at most n. This can be done by the following equation:

$$tr(tr(\ldots tr(x)\ldots)) = 0, \tag{6.6}$$

where the function symbol tr is applied n-times. Let me abbreviate the long term at the left hand side by $tr^n(x)$. As I said, verifying the equality $s(x) = t(x)$ for numbers of length at most n is nothing else but proving it for such numbers using only elementary means. This means that we should use an assumption saying that the length of x is at most n and then only elementary rules. The assumption about the length will be represented by equation (6.6) above. Since $PV1$ is based on elementary axioms and rules and one general principle, we can define:

Polynomial verification is a proof that uses the axioms of $PV1$, but not the induction rule.

Such a definition may not seem quite satisfactory. Why do we say that some axioms are elementary and others not? A more detailed analysis of the axioms is clearly needed. But I am afraid that this would not help very much if one just wants to understand the essence of the problem. It is more instructive to watch how the induction rule can be eliminated. If we accept the definition of the polynomial verifiability as proposed above, elimination of the induction rule is all we need in order to prove that the equalities provable in $PV1$ are polynomially verifiable.

So let us look at how induction can be eliminated. Let us consider an equality $s(x) = t(x)$ provable in $PV1$; I will abbreviate it by $\phi(x)$. Let a proof P of the equality $\phi(x)$ be given and assume we need a polynomial verification for numbers of length n. The idea is to use our assumption that x has length at most n expressed by the equation (6.6) to transform P into a proof that does not use the rule of Induction on Notation. Suppose P ends with an application of the induction rule, namely, that $\phi(x)$ was obtained from assumptions $\phi(0)$ and $\phi(tr(z)) \rightarrow \phi(z)$. Since we have $tr^n(x) = 0$, we get

$$\phi(tr^n(x))$$

by substitution from $\phi(0)$. Now we will use the second assumption of the induction rule. We will successively substitute terms $tr^{n-1}(x), tr^{n-2}(x), \ldots, tr(x)$ and x for the variable z.[9] Thus we obtain

$$\begin{array}{rcl} \phi(tr^n(x)) & \rightarrow & \phi(tr^{n-1}(x)), \\ \phi(tr^{n-1}(x)) & \rightarrow & \phi(tr^{n-2}(x)), \\ & \ldots & \\ \phi(tr(x)) & \rightarrow & \phi(x). \end{array}$$

[8]It turns out that one can also use the standard form of induction without increasing the power of PV. Nevertheless, for showing the connection with polynomial verifiability induction on notation is more convenient.

[9]Strictly speaking, we cannot use substitution to obtain the following formulas. Instead we have to reproduce the proof of $\phi(tr(z)) \rightarrow \phi(z)$ for z replaced by each of the terms $tr^{n-1}(x), tr^{n-2}(x), \ldots, tr(x)$.

Now we apply modus ponens n times and obtain $\phi(x)$. Thus we have eliminated one application of the induction rule and the length of the resulting proof is bounded by a polynomial in n. Repeating essentially the same procedure for all occurrences of the induction rule we get a proof that does not use this rule and has polynomial size. This proof is a polynomial verification of the equation $s(x) = t(x)$.

The fact that universal sentences provable in PV (or $PV1$) can be polynomially verified is one good reason for associating PV with the complexity class **P**. But, of course, any theory that is weaker than PV also has this property. Therefore we need another property to impose a lower bound on the strength a theory that we want to associate with **P**. The ability of PV to prove that polynomial verifiability is sound is such a property. More precisely, PV proves the formalization of the following sentence:

> *If x has length n and y (a code of a proof) is a polynomial verification of $\phi(z)$ for numbers of length n, then $\phi(x)$ is indeed true.*[10]

This property pretty much determines PV: essentially, PV is the weakest system that proves this reflection principle.

The concept of polynomial verifiability is important and deserves more detailed study. There is, however, an alternative approach which is, in some sense, much cleaner and therefore is the choice that is usually preferred over polynomial verifiability. It is based on showing a similar relation of proofs in predicate logic to proofs in the propositional calculus. This means that we replace equational proofs of equalities $s(N) = t(N)$ by propositional proofs of tautologies that express the same fact as $s(N) = t(N)$.

Down to the propositional calculus

When I was a student I wondered why some authors of logic books pay so much attention to the propositional calculus. Back then I thought that there was nothing interesting about the propositional calculus. The two main reasons for me were: first, I thought one could express only elementary facts using propositional logic; second, the calculus is decidable. Both are true, but on a closer look, one can still find very interesting problems. As a matter of fact, my current perception of the propositional calculus is quite the opposite—I believe that some of the most fundamental problems are in this area.

I will start with some examples of what can be formalized in propositional logic. The prevailing wrong impression that only trivialities can be expressed in propositional logic is caused by the fact that introductory logic texts mention only very simple formulas. Once we are willing to work with long formulas, we can express fairly complex statements.

Our first set of examples are arithmetical statements. With n propositional variables we can encode n-bit numbers. Let number x be represented by propositional variables $p_1, p_2, \dots, p_n$ and number y by $q_1, q_2, \dots, q_n$. Then we can express the equality $x = y$ by

$$(p_1 \equiv q_1) \wedge (p_2 \equiv q_2) \wedge \ldots \wedge (p_n \equiv q_n).$$

This is not a tautology, it is a formula talking about two unknown n-bit numbers, which is true for some assignments and false for others.

A more interesting example is a formula that expresses $x \leq y$ for two n-bit numbers. Let the two numbers be represented as above with p_1 and q_1 being the most significant bits, p_2 and q_2 the second most significant etc. Then the formula reads:

$$\begin{array}{l}
((p_1 \equiv q_1) \wedge \ldots \wedge (p_{n-2} \equiv q_{n-2}) \wedge (p_{n-1} \equiv q_{n-1}) \wedge (p_n \equiv q_n))\vee \\
((p_1 \equiv q_1) \wedge \ldots \wedge (p_{n-2} \equiv q_{n-2}) \wedge (p_{n-1} \equiv q_{n-1}) \wedge \neg p_n \wedge q_n)\vee \\
((p_1 \equiv q_1) \wedge \ldots \wedge (p_{n-2} \equiv q_{n-2}) \wedge \neg p_{n-1} \wedge q_{n-1})\vee \\
\ldots \\
(\neg p_1 \wedge q_1).
\end{array}$$

Using this formula (and the one obtained by interchanging p_i's with q_i's), we can express the true arithmetical fact

$$(x \leq y) \vee (y \leq x),$$

[10] Sentences expressing that, for every sentence ϕ from some class of sentences Φ, if ϕ is provable in a theory T, then ϕ is true, are called reflection principles. If the class of sentences Φ contains all sentences, the reflection principle is stronger than consistency, hence not provable in the theory T. In the case considered here the class of sentences is very restricted, so the sentence is actually provable.

for n-bit numbers, by a propositional *tautology.* We cannot express the above dichotomy by a single tautology for all numbers. If we want to talk about all numbers, we have to use a sequence of tautologies parameterized by the lengths of the numbers.

Let us now consider the formula $x + y = z$ where x and y are (at most) n-bit numbers and z is an (at most) $n+1$-bit number. Similarly as we did for $x = y$, we want to say that each bit of $x + y$ is equal to the corresponding bit of z. Therefore, it suffices, for every i, to construct a formula which is true if the i-th bit of $x + y$ is 1 and false if it is 0. To construct such a formula, the standard algorithm based on computing the carry bits is used. Thus one obtains a formula whose size is polynomial in n.

For $x \cdot y = z$, we can also construct a polynomial size formula, although a it is little more complicated. Then we can combine these formulas to construct tautologies that express various true arithmetical sentences. For example, we can construct a tautology that expresses the distributivity law for n-bit numbers. All these tautologies have polynomial sizes, when measured by the length of the numbers about which they talk.

A more interesting example of an arithmetical tautology is the following. Let P be a fixed prime number. The fact that P is prime can be expressed in predicate logic by

$$x \cdot y = P \to (x = 1 \vee y = 1).$$

As we have formulas for equality and multiplication, we can write a formula, in fact a tautology, expressing the primality of P. In this tautology the bits of x and y will be represented by propositional variables, whereas the bits of P will be the corresponding truth constants. If P has n bits, we need only to consider numbers x and y with at most n bits, so we can write it as a single tautology.

Arithmetical tautologies are an ample source of interesting tautologies, but there are others. Perhaps the most heavily researched set of tautologies are the tautologies expressing the *Pigeonhole Principle.* The principle says that if there are $n+1$ pigeons in n holes, then there is at least one hole in which there are at least two pigeons. Let us consider propositional variables p_{ij} indexed by two indices $i = 1, 2, \ldots, n+1$ and $j = 1, 2, \ldots, n$. One should think of the variable p_{ij} as being true if and only if pigeon i sits in hole j. I will use $\bigwedge$ and $\bigvee$ to denote conjunctions and disjunctions of several terms. The pigeonhole tautology PHP_n then reads:

$$\bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} p_{ij} \to \bigvee_{j=1}^{n} \bigvee_{i=1}^{n+1} \bigvee_{1 \le i' \le n+1;\ i' \ne i} (p_{ij} \wedge p_{i'j}).$$

Notice that this formula closely follows the English description of the principle, except for the last part, where instead of saying that *there are two* pigeons, we say that *there exist a pigeon i and there exists a pigeon i' different from i.*

Let us now express the Pigeonhole Principle in predicate logic. Let $P(x, y)$ be a symbol for a binary relation, where x is a number from 1 to $n+1$ and y is a number from 1 to n. The first order sentence reads:

$$\forall x \exists y P(x, y) \to \exists y \exists x \exists x' (x' \ne x \wedge P(x, y) \wedge P(x', y)).$$

The similarity of the two expressions above is quite conspicuous. The quantifiers correspond to the big conjunctions and disjunctions and the binary relation is represented by propositional variables indexed by pairs. This suggests that we can translate any first order sentence true in some finite relational structure into a propositional tautology, which is indeed possible.

So what can actually be expressed by propositional formulas? First recall that *every* boolean function of n variables can be defined using a propositional formula (we could also say *boolean formula,* meaning the same thing) with n variables. Such representations are, for example, the well-known disjunctive and conjunctive normal forms. These forms have exponential size, except for some very special boolean functions. We cannot use them when we want to express, say, that a medium size number P is a prime; the form would be too large.

Thus the right question is: which functions can be defined by medium size boolean formulas? In precise mathematical terms the question is: *which sets of 0–1 strings can be defined by sequences of formulas of polynomial size?* We know that if we use boolean circuits, instead of formulas, we can define every set in **P** by a sequence of polynomial size circuits. Formulas can be viewed as a special kind of circuits, circuits whose underlying graph is a tree. It seems that this is a weaker computational model. Using sequences of polynomial size boolean formulas, or equivalently using sequences of polynomial size tree-like boolean circuits, one can define the sets in a class called $\mathbf{NC}^1$, a subclass of $\mathbf{P}$. Whether the two classes are equal or not, is one of the wide-open problems. The commonly accepted conjecture is that $\mathbf{NC}^1$ is a proper class of $\mathbf{P}$.

In this way the propositional calculus is closely connected with computational complexity, but it is important to realize that there are also fundamental differences between the two fields. In complexity theory we study the circuit complexity of boolean functions and we also study the formula complexity of boolean functions. In the propositional calculus we are primarily interested in tautologies. A tautology is also represented by a boolean formula, but the formula is true for every assignment of the propositional variables. Hence the formula defines a very simple boolean function—the function identically equal to 1 (where 1 represents the truth). The relation to the complexity of boolean functions manifests itself only when we look at the components from which tautologies are constructed.

Example. Let $\phi(p_1, \ldots, p_n)$ and $\psi(p_1, \ldots, p_n)$ be two propositional formulas. Suppose that

$$\phi(p_1, \ldots, p_n) \equiv \psi(p_1, \ldots, p_n)$$

is a tautology. Since it is a tautology, it defines the boolean function constantly equal to 1. But the function defined by ϕ, which is the same as the function defined by ψ, may be an arbitrary one.

In particular, for proof complexity it is more important to know which *sentences* can be represented by sequences of boolean formulas. Since we can define $\mathbf{NC}^1$ sets, we can also represent *universal-*$\mathbf{NC}^1$ *sentences.* The reason is that in logic we are interested in tautologies. A tautology, although represented by boolean formula, is interpreted as if there were universal quantifiers in front of it. Saying that the formula is satisfied by all truth assignments is the same as if we had these universal quantifiers and said that that the quantified propositional formula is true. Since these implicit quantifiers range only over finite domains, we need infinite sequences of boolean formulas to represent a sentence in predicate logic. The relation between the sentence and the sequence of propositional formulas is that the sentence is true if and only if all propositional formulas are tautologies.

Example. Above I have sketched how to represent the true arithmetical sentence $\forall x \forall y (x \leq y \vee y \leq x)$.

Given a sentence ϕ of predicate logic, we call the sequence of boolean formulas representing ϕ a *translation of ϕ into propositional logic.* The word 'translation' is justified by the fact that we often want to preserve at least part of the syntactical structure of ϕ in the translation, which enables us to prove interesting relations between proofs in predicate logic and proofs in the propositional calculus.

It is not difficult to show that in fact one can translate universal-$\mathbf{P}$ sentences into sequences of propositional formulas. Roughly, the universal quantifier enables us to bootstrap the $\mathbf{NC}^1$ predicate into a $\mathbf{P}$ predicate. Thus we can represent our fundamental type of sentences in propositional logic.

*

Let us turn to the most important part of the propositional calculus, the proofs. We have a variety of interesting tautologies and we wonder how difficult it is to prove them. There are various axiomatizations of propositional logic. Essentially in all these axiomatizations one can prove every tautology of size n by a proof of at most exponential size in n, but exponentially large proofs are not very useful. The crucial question is which tautologies have polynomial size proofs. Of course, this makes sense only for infinite sequences of tautologies.

For example, we may ask if the Pigeon Hole tautologies PHP_n have polynomial size proofs. The bound should be polynomial in the lengths of the tautologies, but since PHP_n has polynomial size in n, this is the same as being polynomial in n. Although these are fairly easy tautologies, we know that in some weak proof systems they have only exponential proofs.

In order to be able to answer such questions we must first specify what a propositional proof is. The presence of different axiomatizations suggests that the answer may depend on how we define proofs. Therefore it is useful to introduce the concept of a *propositional proof system.* A typical proof system is based on axiom schemas and derivation rules. A proof in such a system is a sequence of formulas that follow either from the schemas or from the previous formulas by applying a rule. The choice of the schemas and rules influences the minimal length of the proof of a tautology. In order to get more efficient proof systems we can extend these basic systems in various ways. We may add infinite families of tautologies, such as the Pigeon Hole tautologies. Or we can add some special kinds of rules. One of the strong rules is the *substitution rule.* According to this rule, if we derive a tautology ϕ, we may pick any variable p in ϕ and any proposition ψ and substitute ψ for p in ϕ. Another important rule is the *extension rule.* This rule allows us to abbreviate derived tautologies by symbols in order to avoid the use of extremely large formulas.

One can also consider proof systems that differ more radically from the standard concept. To give you a glimpse on how such proof systems might look, I will describe a classical result in algebra called *Hilbert's Nullstellensatz.* Suppose a system of m algebraic equations with n complex variables is given by

$$\begin{gathered} p_1(x_1,\dots,x_n)=0 \\ p_2(x_1,\dots,x_n)=0 \\ \dots \\ p_m(x_1,\dots,x_n)=0 \end{gathered} \tag{6.7}$$

where $p_1, p_2, \dots, p_m$ are polynomials. Suppose we can find polynomials

$$q_1(x_1,\dots,x_n), q_2(x_1,\dots,x_n), \dots, q_m(x_1,\dots,x_n)$$

such that

$$\sum_{i=1}^{m} q_i(x_1,\dots,x_n)p_i(x_1,\dots,x_n) \tag{6.8}$$

is the constant 1 polynomial. Then we know that the system of equations (6.7) does not have a solution. Indeed, if a string of numbers $a_1, \dots, a_n$ were a solution, then after substituting $a_1, \dots, a_n$ into the sum (6.8) we would get a sum of zeros, which is zero. But the sum is identically equal to 1, hence a solution cannot exist.

Thus using such strings of polynomials we can prove that a system of equations does not have a solution. (We can check that the polynomial (6.8) is 1 by expanding it into a sum of monomials.) Hilbert's Nullstellensatz says that this method always works, namely, *if the system does not have a solution, then such a string of polynomials $q_1, q_2, \dots, q_m$ exists.* So we can view it as a proof system for unsatisfiable systems of equations. The strings of polynomials $p_1, p_2, \dots, p_m$ play the role of tautologies, whereas strings of polynomials $q_1, q_2, \dots, q_m$ that satisfy the condition above play the role of proofs. In this system we are assuming that the polynomials are represented as sums of monomials. The system is formally not a propositional proof system, because it concerns complex numbers, but a modification of this system also works for the two element field.[11] Interpreting 1 as truth and 0 as falsehood, as usual, we get a natural propositional proof system.

What is remarkable about this system is that it is not based on sequential derivations of tautologies. A string of polynomials $q_1, q_2, \dots, q_m$, which is a proof in this system, in general does not have parts that are proofs of the unsatisfiability of other systems of equations. The proof makes sense only as a whole.

Another noteworthy fact is the use of a mathematical theorem for designing a proof system. Hilbert's Nullstellensatz is a typical characterization theorem in which a *universal* property is characterized by an *existential* property. The universal property that is characterized is: '*no* string of number is a solution', the existential property is: 'there *exists* a string of polynomials such that the sum (6.8) reduces to 1'. In general, this is exactly what proofs should do. If we want to prove a general statement, we need to characterize the fact that something holds in all structures (of some given type), by the existence of a string of symbols (with some properties).

With these examples in mind we can now isolate the basic properties of propositional proof systems:

1. *Soundness and Completeness.*

2. *Verifiability.*

Soundness means that only tautologies are provable, completeness means that all tautologies are provable. Thus the first condition says that the set of provable formulas is the set of tautologies. This seems to be in contrast to the situation in predicate logic, where different theories may prove different sets of

[11] For the two-element field, we must always include the equation $x_j^2 - x_j = 0$ for all variables, which expresses that variables represent zeros and ones.

theorems. However, what we are actually interested in are the sets of tautologies that have short proofs; these sets are in general different for different systems.

The second condition talks about the nature of proofs. We would like a proof to be undeniable evidence. Whether or not we accept it as such depends also on whether or not we believe in the soundness of the system, but if we believe in it, then we need only to know for sure that a given text is a proof in the system. Put otherwise, we should be able to verify the correctness of a proof. The verification should be a mechanical process, it should not require any ingenuity of the verifier. We formalize it by the requirement that

$2'$. *the correctness of proofs can be verified by an algorithm running in polynomial time.*

The general definition of a propositional proof system is based on these two conditions. Formally, a proof system is given by a binary relation R defined on strings in a finite alphabet such that

$1''$. *for any x there exists y such that $R(x, y)$ if and only if x represents a propositional tautology;*

$2''$. *$R(x, y)$ is decidable in polynomial time.*

Since we are interested in proofs that have at most polynomial length, the central problem about propositional proof systems is:

Problem 1 *Does there exist a propositional proof system P such that for some polynomial $p(x)$, every tautology ϕ has a proof in the proof system P of length at most $p(|\phi|)$, where $|\phi|$ denotes the length of ϕ.*

It is not difficult to see that this problem is equivalent to the following problem about complexity classes:

Problem 2 *Is* $\mathbf{NP} = \mathbf{coNP}$*?*

We conjecture that the answer to these problems is negative—there is no such proof system and, equivalently, $\mathbf{NP} \neq \mathbf{coNP}$. The inequality $\mathbf{NP} \neq \mathbf{coNP}$ implies $\mathbf{P} \neq \mathbf{NP}$, hence proving this conjecture could be even more difficult than proving $\mathbf{P} \neq \mathbf{NP}$.

Let us call a proof system *polynomially bounded* when it satisfies the condition from Problem 1. So the conjecture is that no propositional proof system is polynomially bounded. Can we prove at least that some specific proof systems are not polynomially bounded? Such results have been obtained and researchers have been working on proving them for more proof systems. It is probably naive to think that by gradually extending the set of systems that are not polynomially bounded we will eventually be able to prove the conjecture that none is polynomially bounded. But there are other reasons for proving lower bounds on the lengths of proofs in concrete proof systems. I will explain them in the next subsection.

Theories and propositional proof systems

On our journey through formal systems we arrived at the bottom level—propositional proof systems. However, we will see that the bottom and the top are not as far apart as it may seem on the first sight. In plain words, propositional proof systems are closely connected with theories.

In order to demonstrate these connections I have to identify certain types of proof system. The most common proof systems are based on axioms, or axiom schemas, and derivation rules. One such system was described on page **??**. The proof systems of this type are called *Frege systems*. Frege was the first to introduce a formal system for predicate logic. The propositional part of his proof system was essentially what we call today a Frege system. ♠ All Frege systems are equivalent in the sense I will explain shortly.

Let us say that a propositional proof system P *polynomially simulates* a propositional proof system Q if, given a tautology ϕ and a proof of d of ϕ in Q, one can construct in polynomial time a proof d' of ϕ in P. When P polynomially simulates Q we think of P as *being at least as strong as* Q.

One can prove that every two Frege systems P and Q polynomially simulate each other. The proof is very easy in case the two systems use the same set of connectives. Then in order to simulate a proof d in Q by a proof in P, we need only to simulate each step in the proof d. To simulate one step in d we need a constant number of steps in P. Thus the proof d' in P is larger at most by a constant factor. Since any two Frege systems simulate each other, Frege systems form a natural class of proof systems.

The most important class of proof systems is called *Extended Frege* proof systems. An Extended Frege proof system is a Frege System augmented with the Extension Rule. As mentioned before, the rule is used to abbreviate long propositions and is formalized as follows. For every proposition ψ we may introduce the proposition

$$q \equiv \psi,$$

where q is a variable that does not occur in the previous part of the proof. Furthermore, q must not occur in the proposition that we are proving. The reason for using this rule is that we may have proofs that have a small (polynomial) number of steps, but some of the formulas occurring in the proof may be (exponentially) large. In particular, there are proofs that are exponentially large, but have only polynomial number of steps. Repeated applications of the extension rule enable us to transform such proofs into polynomial size proofs.

However, we are not able to prove that it is necessary to use the Extension Rule in order to avoid the exponential blowup of formulas, although it seems very likely to be the case. On the positive side, we know that every two Extended Frege systems polynomially simulate each other. Thus Extended Frege proof systems form another natural class of proof systems, which I will denote by **EF**.

The reason why we consider Extended Frege systems important is their relation to the theory $\Theta_{\mathbf{P}}$. The main relation is the content of the following theorem.

Theorem 6.8 *1. In $\Theta_{\mathbf{P}}$ it is provable that Extended Frege systems are sound.*

2. If it is provable in $\Theta_{\mathbf{P}}$ that a propositional proof system P is sound, then Extended Frege proof systems polynomially simulate P.

Thus the Extended Frege proof systems are determined by the condition of *being the strongest systems whose soundness is provable in $\Theta_{\mathbf{P}}$.* We have obtained the triad of closely related concepts

P	$\Theta_{\mathbf{P}}$	**EF**

one in computational complexity, one in predicate logic and one in propositional logic. There are more such triads, though the propositional proof systems of them are not as natural as **EF**.

There is another important connection between $\Theta_{\mathbf{P}}$ and **EF**. Let $\forall x\ \phi(x)$ be a true $\Pi_{\mathbf{P}}$ sentence. Let a sequence of tautologies τ_n, $n = 1, 2, \ldots$, be the propositional translations of $\forall x\ \phi(x)$. Thus τ_n expresses in propositional logic that $\phi(x)$ holds true for all numbers x of length n. Now we can ask whether the tautologies have feasible proofs. The feasibility can be interpreted in a weaker sense as the existence of proofs of polynomial lengths, and in the stronger sense, as the existence of an algorithm that for a given n constructs in polynomial time a proof of τ_n. According to our intuition, we expect that if $\forall x\ \phi(x)$ is an "easy" theorem, then the tautologies τ_n should have short proofs (that are also easy to find). The intuition is indeed true; we just have to define easiness as provability in a weak theory and to consider a suitable propositional proof system. The prime example is the pair $\Theta_{\mathbf{P}}$ and **EF**.

Theorem 6.9 *1. Suppose a $\Pi_{\mathbf{P}}$ sentence $\forall x\ \phi(x)$ is a theorem of $\Theta_{\mathbf{P}}$. Then there exists a polynomial time algorithm that constructs proofs of the propositional translations τ_n of this sentence in an Extended Frege proof system. In particular, the propositional translations have polynomial size proofs.*

2. If a proof system P has the property stated in 1. for Extended Frege systems, then P polynomially simulates Extended Frege systems. In plain words, Extended Frege systems are the weakest systems for which 1. holds true.

The opposite of 1. is not true, there are sentences $\forall x\ \phi(x)$ that are not provable in $\Theta_{\mathbf{P}}$, but whose propositional translations have polynomial size proofs in **EF** proof systems (moreover the proofs are constructible in polynomial time).

In order to prove 1., one has to transform a proof in predicate logic into a proof in propositional logic. This is similar to what we have already seen: starting with a general proof in predicate logic, we want to reduce the complexity of formulas in the proof. In the case of Theorem 6.9, we want to reduce the complexity all the way down to propositional logic. To this end we can use polynomially verifiable proofs. Recall that such proofs, although they are in predicate logic, use only elementary axioms. Thus one only needs to simulate the elementary axioms and rules. In this way we can translate proofs of theorems of $PV1$ into proofs in an Extended Frege system. Since $PV1$ and $\Theta_{\mathbf{P}}$ prove the same $\Pi_{\mathbf{P}}$ sentences, we obtain the same for $\Theta_{\mathbf{P}}$.

The relation of $\Theta_{\mathbf{P}}$ to **EF** is similar to the relation of **P** to **nonuniform-P**. Recall that $\mathbf{P} \subseteq$ **nonuniform-P** and a language A in the alphabet $\{0,1\}$ is in **nonuniform-P** is if and only if there exists a sequence of boolean circuits C_n such that for every n the section of A consisting of inputs of length n is the set of inputs accepted by the circuit C_n. So we have a polynomial time Turing machine on the one hand and a sequence of circuits on the other. Similarly, we have a proof of a sentence ψ on the one hand, and a sequence of propositional proofs of the translations of ψ on the other. The similarity of these relations suggests that we should view **EF**, the class of proof systems associated with theory $\Theta_{\mathbf{P}}$, as the nonuniform version of the theory. It is therefore natural to extend the triad to the quadruple:

P	**nonuniform-P**
$\Theta_{\mathbf{P}}$	**EF**

The two rows represent complexity classes and formal systems; the two columns represent uniform and nonuniform concepts.

I have mentioned that there are other triads of complexity classes, theories and propositional proof systems. In particular, there exists a propositional proof system G_1 associated with $\Theta_{\mathbf{NP}}$, which means that G_1 is the strongest propositional proof system whose soundness is provable in $\Theta_{\mathbf{NP}}$. Having the associated proof systems for $\Theta_{\mathbf{P}}$ and $\Theta_{\mathbf{NP}}$ gives us a means that potentially can be used to show that these theories are different. Here is the idea of such a proof. Suppose that $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$. Since $\Theta_{\mathbf{NP}}$ proves the soundness of G_1 and we are assuming $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$, $\Theta_{\mathbf{P}}$ also proves the soundness of G_1. Since Extended Frege systems are the strongest systems whose soundness $\Theta_{\mathbf{P}}$ proves, they must polynomially simulate G_1. In particular, if P is an Extended Frege system, then every tautology has at most a polynomially longer proof in P than in G_1. Hence a possible strategy for proving $\Theta_{\mathbf{P}} \not\equiv \Theta_{\mathbf{NP}}$ is to find a sequence of tautologies τ_n, $n = 1, 2, \ldots$ such that for some polynomial $p(n)$ the tautologies have proofs of lengths at most $p(n)$ in G_1, but do not have proofs of polynomial length in P.

In this way the problem of separating $\Theta_{\mathbf{P}}$ from $\Theta_{\mathbf{NP}}$ has been reduced to a combinatorial problem about proving lower bounds on the lengths of proofs in **EF**. Unfortunately, we are not able to use this reduction, because we are not able to prove superpolynomial lower bounds on **EF** proofs for any sequence of tautologies. In other words, we are not able to disprove that **EF** proof systems are polynomially bounded. If, contrary to our expectation, **EF** proof systems were polynomially bounded, then in particular we would have **NP** = **coNP**. Thus not only are we not able to prove that **NP** = **coNP**, which means that no propositional proof system is polynomially bounded, but we are also not able to prove the weaker statement that Extended Frege proof systems are not polynomially bounded.

As it is one of the central problems in the area, I will state this explicitly.

Problem 3 *Are* **EF** *proof systems polynomially bounded?*

We are not able to prove superpolynomial lower bounds even for the weaker class of Frege proof systems. These are the most natural propositional proof systems, yet we do not know how to prove such bounds. Although this problem does not seem to have any direct connection with complexity, it seems as difficult as the open problems about complexity classes. In particular, it looks very much like the problem of proving superpolynomial lower bounds on boolean circuits computing boolean functions defined by **NP** sets. A boolean circuit can be represented by a sequence of elementary boolean operations, likewise a Frege proof is a sequence of applications of elementary logical rules; an Extended Frege proof is only slightly more complex. We do not have any methods for proving more than linear lower bounds for any concrete boolean function likewise we lack methods for proving more than linear lower bounds on Extended Frege proofs of any tautology.

Proving lower bounds can not only help us prove separation of theories, it can also be used to prove independence results. According to Theorem 6.9, to prove that a $\Pi_{\mathbf{P}}$ sentence ϕ is unprovable in $\Theta_{\mathbf{P}}$, it suffices to prove superpolynomial lower bounds on the lengths of proofs in **EF** of the propositional translations of ϕ. The same is true for any theory that has an associated proof system (in fact, as we will see shortly, for all theories). Such independence results have actually been proven for some weaker systems, but it would take us too far afield to explain these results.

Proving that some sentence is unprovable in $\Theta_{\mathbf{P}}$ seems to be a very weak result. Indeed, $\Theta_{\mathbf{P}}$ is much weaker than Peano Arithmetic, Peano Arithmetic is much weaker than Zermelo-Fraenkel set theory and

Zermelo-Fraenkel set theory is much weaker than the strongest set theories that have been considered (obtained by adding large cardinal axioms to Zermelo-Fraenkel set theory). But our goal is not to show independence result for the strongest possible theories. We want to know what is provable in some particular theories. We have a good reason for asking what is provable in $\Theta_{\mathbf{P}}$, namely, we want to prove that it is weaker than $\Theta_{\mathbf{NP}}$. We also hope that once we are able to prove independence results for $\Theta_{\mathbf{P}}$ we will have a method that will enable us to prove such results for other theories.

There is another important reason for trying to prove independence results in this way, which is the fact that we know only one method of proving the independence of $\Pi_{\mathbf{P}}$ sentences. The only such method is Gödel's theorem. This method gives some specific sentences which are not useful for proving independence of concrete mathematical theorems. Therefore we are looking for other methods and proving lower bounds on propositional proofs could be an alternative.

Proof complexity and computational complexity

When comparing proof complexity with computational complexity the most striking fact is that many concepts in one field have a counterpart in the other. Here are some pairs of such concepts.

computational complexity	*proof complexity*
complexity classes	theories
elementary operations	basic axioms
computations with bounded resources	induction restricted to some class of formulas
the **P** vs. **NP** problem	the $\Theta_{\mathbf{P}}$ vs. $\Theta_{\mathbf{NP}}$ problem
circuits	propositional proofs

Although in most cases the correspondence is not buttressed by a formal definition, the similarities are very conspicuous. This suggests that the two fields study the same general phenomenon from different perspectives. Yet the two fields are essentially different. They have different goals and, in spite of some relations, the open problems in proof complexity are not equivalent to open problems in computational complexity.

In particular researchers working in proof complexity are not trying to solve **P** vs. **NP** or some other problem about separating complexity classes. Proof complexity studies how difficult is to prove certain theorems. Thus in the case of **P** vs. **NP**, the questions relevant for proof complexity are: *how strong a theory do we need to prove* $\mathbf{P} = \mathbf{NP}$ (in case it is true), and *how strong a theory do we need to prove* $\mathbf{P} \neq \mathbf{NP}$ (in case it is true)? When a proof of $\mathbf{P} = \mathbf{NP}$ or of $\mathbf{P} \neq \mathbf{NP}$ is found, we will be able to give an upper estimate on the theory in which it is provable. Meanwhile we can only try to show the unprovability of such statements in weak theories.

Concerning the unprovability of such statements, very little has been achieved so far. In fact, it seems that to prove such results is as hard as to solve the corresponding problems. For example, proving that $\Theta_{\mathbf{P}} \not\equiv \Theta_{\mathbf{NP}}$, which would show that $\mathbf{P} = \mathbf{NP}$ is not provable in $\Theta_{\mathbf{P}}$, seems as difficult as actually proving $\mathbf{P} \neq \mathbf{NP}$. But there are some results in this direction. One of the early results, due to Buss, is the following.

Theorem 6.10 $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$ *is not provable in* $\Theta_{\mathbf{P}}$.

This is a corollary of Theorem 6.4 also due to Buss (see part C for the proof). The conjecture $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$ is important in cryptography because if it were true that $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, then the protocols currently used would probably be insecure.[12] Specifically, the existence of one-way bijections implies $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$.

Another interesting result is due to Razborov. He proved that it is impossible to prove lower bounds on circuit size in certain theories. The precise statement of his result is quite complicated. One can fairly easily state a weaker version of the result which concerns a certain extension of theory $\Theta_{\mathbf{NP}}$, but this is not interesting, because this formalization is very weak; one cannot prove even very elementary lower

[12] The conjecture talks about polynomial time algorithms and for practical purposes it is important what the polynomial bound is in these algorithms. The protocols would be insecure if the polynomial bounds were reasonably small.

bounds in it. Therefore Razborov had to introduce a more complicated theory, to my mind also quite unnatural, in which it is possible to prove some well-known lower bounds for certain type of circuits, but which still does not prove lower bounds for general circuits. Furthermore, some of these results are only proved using conjectures such as the PRG-Conjecture (which we considered in the previous chapter, see page **??**). Although complicated and unnatural, these results confirm the conjecture that there is a qualitative difference between proving lower bounds for restricted type of circuits and for general ones. I will say a little more about this in part C.

Once I was asked why we study proof complexity when it seems that the problems in proof complexity are as difficult as in those computational complexity. If, for example, we are interested in the **P** vs. **NP** problem, we should address the problem directly. My answer is, to put it shortly, as follows. Firstly, as I explained above, our aim in proof complexity is not to solve problems from computational complexity (but, of course, if we discover a possible way to do this using proof complexity we will go after it). Secondly, I am a logician and am interested in the foundations of mathematics. Proof complexity is directly connected with the foundations of mathematics, whereas the connection with computational complexity is not so tight. Furthermore, I believe that there may be hurdles of a fundamental nature that prevent us from solving the basic problems of computational complexity. If this is the case, we are more likely to find them using logic.

Nevertheless, there are a few cases in which proof complexity has indeed helped solve a problem in computational complexity. In the rest of this subsection I will describe such a result. It concerns an important problem called *Integer Linear Programing.* An instance of Integer Linear Programing is given by a finite set of linear inequalities with rational coefficients:

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \ \dots \ + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ \dots \ + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq b_m, \end{array} \tag{6.9}$$

where a_{ij} are rational numbers and x_j are unknowns. The aim is to find an integral solution, which means to find integers $c_1, c_2, \dots, c_n$ that satisfy the inequalities.

This problem is closely related to the *Linear Programing* problem, which is defined in the same way, with the difference that what we are looking for now are any *rational* solutions. Furthermore, if a solution exists, then we want to find a solution that optimizes some linear function. For Linear Programing, polynomial time algorithms are known, whereas Integer Linear Programing is not solvable in polynomial time unless $\mathbf{P} = \mathbf{NP}$. Deciding whether the set of inequalities (6.9) has an integral solution is an **NP**-complete problem. Linear Programing is a potent tool in designing efficient algorithms. Integer Linear Programing could play an even more important role, if it were solvable in polynomial time. Still, it makes sense to design algorithms for Integer Linear Programing and study how they perform on particular types of data, even if we believe that $\mathbf{P} \neq \mathbf{NP}$. On the other hand, if somebody believes that $\mathbf{P} = \mathbf{NP}$ and wants to prove it, then it would be a good strategy to try to find a polynomial time algorithm for Integer Linear Programing.

Let us now focus on decision problems, since they determine the complexity of these two problems. From high school, we know two operations that preserve solutions of linear inequalities:

1. *adding two equations;*

2. *multiplying an equation by a positive constant.*

According to a well-known result, the system of inequalities (6.9) does not have a rational solution if and only if we can derive from it the contradictory inequality

$$0 \geq 1$$

using these two deduction rules. The proof is always short, since it suffices to multiply the equations (6.9) by nonnegative constants and then add them. The tricky thing is how to find the constants.

It can happen that a system of linear inequalities has a rational solution, but does not have any integral solutions. Thus we need something else to characterize the solvability of (6.9) in the domain of integers. In 1975, building on work of R. E. Gomory, V. Chvátal proved that it suffices to add another simple rule in order to characterize unsolvability of (6.9) over the integers.

3. *If the constants* $c_1, c_2, \ldots, c_n$ *are integers, then from*

$$c_1x_1 + c_2x_2 + \ldots c_nx_n \geq d$$

we can derive

$$c_1x_1 + c_2x_2 + \ldots c_nx_n \geq \lceil d \rceil,$$

where $\lceil d \rceil$ *denotes the least integer such that* $\lceil d \rceil \geq d$.

In other words, if the coefficients on the left hand side are integral, we can round up the number on the right hand side. This rule is clearly true in the domain of integers, because if the coefficients on the left hand side are integral, then the sum on the left hand side is an integer. The method based on 1., 2. and 3. is called the method of *cutting-planes*.[13]

Knowing that 1. and 2. suffice for designing a polynomial time algorithm for Linear Programing, it is natural to ask whether 1., 2. and 3. suffice for designing a polynomial time algorithm for Integer Linear Programing. Since we believe that $\mathbf{P} \neq \mathbf{NP}$, we expect the answer to be negative. So the question is: *can we prove that there is no polynomial time algorithm for Integer Linear Programing based on on the method of cutting-planes?* Proving $\mathbf{P} \neq \mathbf{NP}$ may be extremely difficult, but what we are asking for is much weaker we only want to prove that a certain type of algorithms cannot run in polynomial time.

Such a result was proven in 1996 and the proof uses methods developed in proof complexity. It is quite natural to view the three rules of the method of cutting-planes as the derivation rules of a proof system. Since the variables $x_1, x_2, \ldots, x_n$ range over integers, adding the inequalities $0 \leq x_1 \leq 1$, $0 \leq x_2 \leq 1, \ldots, 0 \leq x_n \leq 1$ further restricts the solutions to zeros and ones. Thus we can think of inequalities as a kind of boolean formulas. In particular, we can express disjunctions of variables and negated variables by inequalities (for example, the disjunction $x \vee y \vee \neg z$ is equivalent to $x + y + (1 - z) \geq 1$). This enables us to represent various interesting tautologies by unsatisfiable sets of inequalities. ♠

Let τ be a propositional tautology and let S be the system of unsatisfiable inequalities representing τ. If d is a derivation of the contradictory inequality $0 \geq 1$ from S, we say that d is a *cutting-planes* proof of τ. This defines the *cutting-planes* proof system. It is one of the few proof systems for which we are able to prove exponential lower bounds on the lengths of proofs. Translated back to Integer Linear Programing it means that there are systems of inequalities that are not satisfied by integers, but for which any derivation of a contradiction based on the rules 1., 2. and 3. is of exponential length. In particular whatever strategy we use in an algorithm that is based only on these three rules, the algorithm must run in exponential time.

Similar results have been obtained for some other methods for solving Integer Linear Programing.

— C —

Bounded arithmetical formulas. Bounded quantifiers are of the form $\forall x \leq t$ and $\exists x \leq t$, where x is the quantified variable and t is a term. The variables of t must be different from x and must either be free or quantified before x in the formula in which the bounded quantifier is used. Bounded quantifiers may be treated as primitive notions or as abbreviations:

$$\forall x \leq t\ \phi \quad \text{as an abbreviation of} \quad \forall x (x \leq t \to \phi),$$

$$\exists x \leq t\ \phi \quad \text{as an abbreviation of} \quad \exists x (x \leq t \wedge \phi).$$

Formulas that contain no quantifiers or only bounded quantifiers are called *bounded formulas.*

Example. The following bounded formula defines that y is the least proper divisor of x:

$$1 < y < x \wedge \exists z \leq x (y \cdot z = x) \wedge \forall w \leq y (\exists z \leq x (w \cdot z = x) \to (w = 1 \vee w = y)).$$

Since the ranges of quantified variables in a bounded formula are finite intervals, satisfiability of bounded formulas is algorithmically decidable.

[13] I am only talking about characterizing unsolvability, for the sake of simplicity, but Gomory and Chvátal proved the more general result that the method of cutting-planes suffices to derive all inequalities that are consequences of the initial ones.

Definition of $\Theta_{\mathbf{P}}$. According to the general paradigm we should define a set of formulas $\Phi_{\mathbf{P}}$ that define the sets in $\mathbf{P}$ and define $\Theta_{\mathbf{P}}$ as some set of basic axioms plus induction axioms for all formulas in $\Phi_{\mathbf{P}}$. We have met such a class in the subsection about the equational theory PV. So one possibility is to define $\Theta_{\mathbf{P}}$ as an extension of PV, or better of $PV1$, to predicate logic. A rather unpleasant feature of such a theory is that the set of function symbols is infinite. Because of this, such a theory has an infinite number of axioms on top of the induction axioms.

Apparently, there is no natural set of formulas based on a finite set of predicates and functions that defines $\mathbf{P}$. Nevertheless, one can define $\Theta_{\mathbf{P}}$ using a language with only a finite number of primitives. The trick is that one does not have to postulate induction for formulas defining all sets in $\mathbf{P}$. It suffices to postulate induction for a set of formulas that only define some sets in $\mathbf{P}$ and derive induction for the others from these axioms. An elegant theory of this kind, which I am going to describe below, was proposed by Emil Jeřábek.

The primitives of the language of the theory consist of the usual constant 0, the successor function $S(x)$, the arithmetical operations $+$ and $\cdot$, the binary relation $\leq$ and three more functions $|x|$, $x\#y$ and $\lfloor \frac{x}{2^y} \rfloor$. The intended interpretation of $|x|$ is the length of x in binary representation, which is in mathematical symbols $\lceil \log(x+1) \rceil$. The intended interpretation of $x\#y$ is $2^{|x|\cdot|y|}$. The intended interpretation of the third function is clear from the symbols representing it—it is x with y least significant bits truncated; it can also be defined as the y-times iterated truncation function $tr(x)$. (Recall that $\lceil \dots \rceil$ and $\lfloor \dots \rfloor$ denote rounding up to an integer and respectively rounding down to an integer.)

The binary operation $\#$ plays a special role. Let us observe that for a natural number n, the length of $n\#n$ is approximately quadratic in the length of n. Similarly, the length of $n\#(n\#n)$ is approximately cubic in the length of n, etc. Viewing numbers as binary strings, the presence of this operation (with the axioms that determine its properties) ensures the possibility of constructing strings of polynomial length for every polynomial. This is a necessary condition if we want to be able to formalize polynomial time computations, but it is also sufficient if it is part of a suitable axiomatization. One can also show that the terms of this language do not extend the length more than polynomially. Due to this fact and the special form of the axioms, one can show that the theory is only able to prove that the length of a string can be extended polynomially, but not more than that.

Here are the basic axioms that determine the meaning of the primitive symbols.

$$
\begin{array}{ll}
x+0=x, & x+S(y)=S(x+y),\\[1ex]
x\cdot 0=0, & x\cdot S(y)=x\cdot y+x,\\[1ex]
\lfloor \frac{x}{2^0} \rfloor = x, & \lfloor \frac{x}{2^y} \rfloor = 2\cdot\lfloor \frac{x}{2^{S(y)}} \rfloor \ \vee\ \lfloor \frac{x}{2^y} \rfloor = S(2\cdot\lfloor \frac{x}{2^{S(y)}} \rfloor),\\[1ex]
|0|=0, & x\neq 0 \to |x| = S(|\lfloor \frac{x}{2^1} \rfloor|),\\[1ex]
0\#1=1, & x\neq 0 \to x\#1 = 2\cdot(\lfloor \frac{x}{2^1} \rfloor\#1),\\[1ex]
 & x\neq 0 \to (y\#x) = (y\#1)\cdot(y\#\lfloor \frac{x}{2^1} \rfloor).
\end{array}
$$

As usual, 1 and 2 are abbreviations for $S(0)$ and $SS(0)$.

A *sharply bounded quantifier* is a bounded quantifier in which the bounding term t has the form $|s|$, i.e., the outermost function is the length function. A *sharply bounded formula* is a formula with all quantifiers (if there are any) sharply bounded. The class of sharply bounded formulas is denoted by Σ_0^b. One can easily see that the range of the quantified variables in a sharply bounded formulas is bounded by a polynomial in the lengths of the free variables. Therefore, satisfiability of sharply bounded formulas can be computed in polynomial time. To obtain $\Theta_{\mathbf{P}}$ we now add an infinite set of axioms consisting of:

The induction schema for sharply bounded formulas.

The class of sets definable by sharply bounded formulas is most likely smaller than $\mathbf{P}$, yet this schema of induction suffices to prove induction for all sets in $\mathbf{P}$. In $\Theta_{\mathbf{P}}$ we define the sets of $\mathbf{P}$ using bounded formulas that are not sharply bounded. The set of these formulas can be defined syntactically, hence it would be possible to have the induction schema stated for all such formulas. It is only for reasons of simplicity and elegance that induction is postulated only for sharply bounded formulas.

Definition of $\Theta_{\mathbf{NP}}$. To define $\Theta_{\mathbf{NP}}$ is a simpler task, because there is a natural class of formulas that define all sets in $\mathbf{NP}$. Also the theory is stronger, hence formalizing the necessary concepts in it is easier.

The class of formulas that we need is called Σ_1^b. It consists of formulas of the following form: a prefix of bounded existential quantifiers followed by a sharply bounded formula.[14] The prefix can be empty, so $\Sigma_0^b \subseteq \Sigma_1^b$.

[14] Σ_1^b is often defined as a more general class where in the prefix sharply bounded universal quantifiers are also allowed.

Example. The **NP** set of numbers x that have a proper divisor less than $\sqrt[3]{x}$ can be defined by the following Σ_1^b formula:

$$\exists y \leq x \exists z \leq x(y \neq 1 \wedge y \cdot y \cdot y \leq x \wedge y \cdot z = x).$$

We define $\Theta_{\mathbf{NP}}$ as an extension of $\Theta_{\mathbf{P}}$ in which the induction schema is postulated for all Σ_1^b-formulas.

Alternative definitions of theories associated with P. Some authors associate different theories with **P**. In this book I associate a theory T with a complexity class $\mathcal{C}$ if T has induction axioms for formulas defining the complexity class $\mathcal{C}$ and not more. In other words my criterion is induction axioms. Others consider witnessing theorems (such as Theorem 6.4) to be the criterion. This means that they associate with a theory the smallest complexity class of functions for which the witnessing theorem holds, and then they take the corresponding complexity class of sets. If we use witnessing functions to determine theories corresponding to complexity classes, we may postulate a somewhat stronger form of induction. In particular, in the case of the class **P**, we may define the theory using Polynomial Induction

$$\phi(0, y) \wedge \forall x(\phi(tr(x), y) \rightarrow \phi(x, y)) \rightarrow \forall x\ \phi(x, y)$$

for the set of Σ_1^b-formulas. This theory, denoted by S_2^1, is stronger than $\Theta_{\mathbf{P}}$, assuming a plausible conjecture about complexity classes. Yet the witnessing functions for S_2^1 are still only **FP**.

Nevertheless, this discrepancy is not significant. The most important theorems are those that are $\Pi_{\mathbf{P}}$ sentences and these theorems are the same in all proposed theories for **P**.

The main idea of the proof of Theorem 6.3. The idea of the proof is first to derive nontrivial information about computations from the provability of Σ_1^b-induction from $\Theta_{\mathbf{P}}$ and then reduce this information to a relation between two complexity classes.

Let $\gamma(x, y)$ be a formula defining a relation computable in polynomial time; think of the relation as a kind of search problem, in which x is a parameter and y is a solution to x. Suppose that 0 is always a solution and no solution is greater than x, i.e., $\forall x(\gamma(x, 0) \wedge \forall y(\gamma(x, y) \rightarrow y \leq x))$ is true. I will, moreover, assume that this is provable in $\Theta_{\mathbf{P}}$. Then it is provable in $\Theta_{\mathbf{NP}}$ that for every x there exists a largest solution. Expressed formally, the following sentence is provable in $\Theta_{\mathbf{NP}}$:

$$\forall x \exists y \forall z(\gamma(x, y) \wedge (\gamma(x, z) \rightarrow z \leq y)). \tag{6.10}$$

Hence if $\Theta_{\mathbf{P}} \equiv \Theta_{\mathbf{NP}}$ were true, then this sentence would also be provable in $\Theta_{\mathbf{P}}$. To derive a computational consequence of such an assumption, I will use the following general result.

Lemma 6.11 *Let $\phi(x, y, z)$ be a formula defining a polynomial time relation. Suppose that $\forall x \exists y \forall z\ \phi(x, y, z)$ is provable in $\Theta_{\mathbf{P}}$. Then there exists a number k and polynomial time computable functions $f_0, \ldots, f_{k-1}$ such that*

$$\phi(n, f_0(n), m_1) \vee \phi(n, f_1(n, m_1), m_2) \vee \phi(n, f_2(n, m_1, m_2), m_3) \vee \ldots \vee \phi(n, f_{k-1}(n, m_1, \ldots, m_{k-1}), m_k)$$

is true for all numbers $n, m_1, m_2, \ldots, m_k$.

This lemma was proved using Herbrand's theorem (see the version without additional function symbols on page 20). The best way to understand the complicated formula is to interpret it as interactive computation with two players. The two players are called Teacher and Student. Student is limited to polynomial time computations and his goal is, for a given n, to compute a solution s, which is a number satisfying $\forall z\ \phi(n, s, z)$. Student may make several attempts and each time Teacher helps him by providing a counterexample to his s. This means that Teacher gives Student some t such that $\phi(n, s, t)$ is false, unless Student has already found the right s.

The meaning of the formula is that Student, represented by the functions $f_0, \ldots, f_{k-1}$, succeeds in at most k steps. Given n, Student first computes $f_0(n)$. If it is not a solution, he gets m_1 from Teacher and computes $f_2(n, m_1)$, and so on. Teacher cannot give counterexamples in all k rounds, because that would make the formula false. Hence Student must succeed.

Now, let us look at what would happen if sentence (6.10) were provable in $\Theta_{\mathbf{P}}$. Under this assumption we would be able to compute greatest solutions for every search problem defined by a polynomial time relation, using only polynomial time and a constant number of counterexamples. Without counterexamples, this would imply $\mathbf{P} = \mathbf{NP}$. It seems likely that a constant number of counterexamples cannot be of much help, because the number of possible solutions can be exponential. This intuition is confirmed by proving that in such a case we would have $\mathbf{NP} \subseteq$ **nonuniform-P**. Roughly speaking, Teacher's advice can be simulated by nonuniformity.

$\mathbf{P} = \mathbf{NP}$ and collapsing theories. We are not able to use the assumption $\mathbf{P} = \mathbf{NP}$ to prove that two theories are equal. But if we moreover know that $\mathbf{P} = \mathbf{NP}$ is provable in a specific theory, then we can prove such a consequence.

Let us consider an example. We know that if $\mathbf{P} = \mathbf{NP}$, then $\mathbf{EXP} = \mathbf{NEXP}$ (the latter class is Exponential Nondeterministic Time). The proof of this implication can be formalized in $\Theta_{\mathbf{EXP}}$, hence if $\Theta_{\mathbf{EXP}}$ proves $\mathbf{P} = \mathbf{NP}$, then it also proves $\mathbf{EXP} = \mathbf{NEXP}$. But if it proves that $\mathbf{EXP} = \mathbf{NEXP}$, then we can derive induction for $\mathbf{NEXP}$ formulas from induction for $\mathbf{EXP}$ formulas. Putting things together we get:

> *If $\Theta_{\mathbf{EXP}}$ proves $\mathbf{P} = \mathbf{NP}$, then $\Theta_{\mathbf{EXP}} = \Theta_{\mathbf{NEXP}}$.*

Thus the provability of $\mathbf{P} = \mathbf{NP}$ in some theory implies a collapse of two theories.

But, if $\mathbf{P} = \mathbf{NP}$ is true, how would we find it out? Only by proving it in some theory. The theory could be so strong that it does not correspond to any natural complexity class; however, it is more likely that such a theory would not be too strong (in particular, it is conceivable that it would be $\Theta_{\mathbf{EXP}}$). Hence it is likely that we would get a collapse of theories.

Sentences unprovable in $\Theta_{\mathbf{P}}$. Although we lack methods for proving the independence of arithmetical sentences from $\Theta_{\mathbf{P}}$, we can at least prove some partial results.

The first kind of results like this are proofs of unprovability based on conjectures from computational complexity. An example of such an independence is the following theorem (a consequence of the Euler-Fermat theorem):

Theorem 6.12 *If the cryptographic protocol RSA (see page **??**) is secure, then $\Theta_{\mathbf{P}}$ does not prove the following sentence*

$$\forall x \forall y (x > 1 \wedge (x, y) = 1 \rightarrow \exists z (y^z \equiv 1 \bmod x))$$

(The meaning of the expression $(x, y) = 1$ is that x and y are coprime.)

The second kind of results involve sentences that are likely to be false. An example of such a sentence was given in Theorem 6.10. It was stated rather informally as '$\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$ is not provable in $\Theta_{\mathbf{P}}$'. The precise statement is as follows.

> *If $\Theta_{\mathbf{P}}$ proves that $\forall x\ \phi(x) \equiv \psi(x)$, where ϕ and ψ are Σ_1^b and Π_1^b sentences respectively, then the sentences define a set in $\mathbf{P}$.*

The proof is simple. Let $\phi(x)$ have the form $\exists y\ \alpha(x, y)$ where α is sharply bounded. Let $\psi(x)$ have the form $\forall z\ \beta(x, z)$ where β is sharply bounded. I am assuming that the bounds on the quantifiers $\exists x$ and $\forall y$ are implicit in the formulas. The assumption about the provability of $\forall x\ \phi(x) \equiv \psi(x)$ in $\Theta_{\mathbf{P}}$ implies that

$$\forall x \exists y \exists z (\alpha(x, y) \vee \neg \beta(x, z))$$

is provable in $\Theta_{\mathbf{P}}$. According to Theorem 6.4, there exist polynomial time computable functions f and g such that the following is true:

$$\forall x (\alpha(x, f(x)) \vee \neg \beta(x, g(x))).$$

Whence,

$$\phi(x) \equiv \alpha(x, f(x)).$$

Since $\alpha(x, y)$ is sharply bounded and f computable in polynomial time, $\alpha(x, f(x))$ defines a set in $\mathbf{P}$. This finishes the proof.

The Polynomial Hierarchy. We have already met the Arithmetical Hierarchy (see page **??**). Let us recall that it is a hierarchy of arithmetically defined subsets of natural numbers defined as follows. The level Σ_n are the sets definable by formulas with n alternations between existential and universal quantifiers, where the formula starts with an existential quantifier. The level Π_n is defined in the same way, except that we require the defining formulas to start with a universal quantifier.

Now, the Polynomial Hierarchy is a feasible version of the arithmetical hierarchy where we only allow quantifications limited to finite domains. The finite domains consist of numbers, or of strings of polynomial length. The levels of the Polynomial Hierarchy are denoted by Σ_n^p and Π_n^p (the superscript p standing for 'polynomial').

Let us consider a couple of examples.

1. The class Σ_1^p is the familiar class $\mathbf{NP}$. We know that a set $A \in \mathbf{NP}$ can be defined by a formula of the following form.

> $x \in A$ *if and only if* $\exists y (|y| \leq p(|x|) \wedge B(x, y))$.

In this formula, $|\dots|$ denotes the length and p denotes some polynomial, so the first part says that the length of y is polynomially bounded by the length of x. The binary relation $B(x, y)$ is assumed to be computable in polynomial time.

An example of a class in Σ_1^p is the set of all pairs (G, k) such that G is a graph that contains a clique of size k, thus has a clique number of *at least* k.

2. The class Π_1^b is defined by formulas of the form

$x \in A$ *if and only if* $\forall y(|y| \leq p(|x|) \to B(x, y))$.

This class is dual to the class **NP**, in the sense that it is the class of complements of sets in **NP** and vice versa. (This is due to the fact that the negation of the existential quantifier is the universal quantifier.) Therefore it is usually denoted by **coNP**.

3. The class Σ_2^p is the class of all sets A that can be defined as follows.

$x \in A$ *if and only if* $(\exists y_1, |y_1| \leq p_1(|x|))(\forall y_2, |y_2| \leq |p_2(x)|)C(x, y_1, y_2)$,

where p_1 and p_2 are polynomials, and C is a ternary relation computable in polynomial time. Here I am using a shorter notation for bounded quantifiers.

An example of a class in Σ_2^p is the set of all pairs (G, k) such that the clique number of G *equals to* k. To define such pairs we must say that *there exists* a clique of size k and *for every* subset X of vertices of size $k+1$, X is not a clique in G. (Since we can switch the quantifiers in this example, the set is also in Π_2^p.)

If we view **NP** as an extension of **P** by adding one existential quantifier, the Polynomial Hierarchy is the hierarchy obtained by further extensions of these classes by adding more quantifiers. It has been conjectured that all the classes $\mathbf{P}, \Sigma_n^p$ and Π_n^p, for $n = 1, 2, , \dots, ,$ are distinct, that is, the Polynomial Hierarchy is a proper hierarchy. One can show that if $\Sigma_n^p = \Pi_n^p$, then all the classes with higher indices collapse to Σ_n^p.

One reason for believing that the Polynomial Hierarchy does not collapse is that the analogous relations do in fact hold for the class of recursive sets **Rec** and the levels Σ_n and Π_n of the Arithmetical Hierarchy. This is a rather weak argument, because some analogies apparently fail. While $\mathbf{Rec} = \Sigma_1 \cap \Pi_1$, we conjecture that **P** is a proper subclass of $\mathbf{NP} \cap \mathbf{coNP}$. If $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$ were true, we would be able to invert any polynomial time computable function and thus one-way functions would not exist.

Several conjectures can be reduced to the conjecture that the Polynomial Hierarchy does not collapse. In particular, if $\mathbf{NP} \subseteq$ **nonuniform-P**, then $\Sigma_2^p = \Pi_2^p$. Hence, by Theorem 6.3, the conjecture about the noncollapsing of the Polynomial Hierarchy implies that $\Theta_{\mathbf{NP}}$ is stronger than $\Theta_{\mathbf{P}}$.

The Bounded Arithmetic Hierarchy and relativizations. One can define a hierarchy of theories that corresponds to the Polynomial Hierarchy. For each class of sets Σ_n^p define the class of bounded formulas Σ_n^b that define the sets in Σ_n^p. Then extend $\Theta_{\mathbf{P}}$ by postulating induction for such formulas. Thus we obtain a series of theories $\Theta_{\Sigma_n^p}$. The standard notation for these theories is T_2^n and we call the series of theories *the Bounded Arithmetic Hierarchy.* The union of all theories T_2^n is denoted T_2. Many results proved for $\Theta_{\mathbf{P}}$ and $\Theta_{\mathbf{NP}}$ can be extended to these theories. In particular it has been proven that if the Polynomial Hierarchy does not collapse, then also the Bounded Arithmetic Hierarchy does not.

A natural way to extend theories in Bounded Arithmetic is to enrich the language so that one can talk about sets. The simplest such extensions are obtained by adding a new predicate R to the language and extending the induction axioms to the corresponding class of formulas in the extended language. Given a theory T, we denote such an extension by $T[R]$.

For example, to obtain $\Theta_{\mathbf{P}}[R]$, consider the formalization of $\Theta_{\mathbf{P}}$ described above. First we define sharply bounded formulas in the extended language in the same way as in the original one, the only difference being that we can now also use R. Then we extend the induction scheme to these formulas.

The interpretation of a sentence involving R is that the sentence is true *for all* subsets R. The computational interpretation of R is as an "oracle", a means to obtain additional information for free. Indeed, theories augmented with R behave very much like complexity classes with oracles. The difference is that whereas in complexity theory an oracle is an explicit set, in logic it is just a variable. We follow the complexity-theoretic terminology by calling the extended theories *relativized.*

As in complexity theory, we can prove relativized separations. In particular, we can prove *without any unproven assumptions* that

$$\Theta_{\mathbf{P}}[R] \neq \Theta_{\mathbf{NP}}[R] \neq T_2^2[R] \neq T_3^3[R] \dots$$

These results are based on proving lower bounds for proof systems corresponding to the relativized theories (which are different from the proof systems for the unrelativized theories).

The method of feasible interpolation. Feasible interpolation, invented by Jan Krajíček, is a useful way of proving lower bounds on the lengths of proofs in propositional calculus. It is based on the classical result called *Craig's Interpolation Theorem,* due William Craig. That theorem is normally stated for predicate logic, but has also a version for propositional logic, which is what we are interested in now.

Theorem 6.13 *Let $\phi(\bar{p}, \bar{q})$ denote a proposition with propositional variables $p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_m$ and let $\psi(\bar{p}, \bar{r})$ denote a proposition with propositional variables $p_1, p_2, \ldots, p_n, r_1, r_2, \ldots, r_k$. Suppose that*

$$\phi(\bar{p}, \bar{q}) \to \psi(\bar{p}, \bar{r})$$

is a tautology. Then there exists a proposition $\iota(\bar{p})$ that contains only the common variables $p_1, p_2, \ldots, p_n$ such that the two implications

$$\phi(\bar{p}, \bar{q}) \to \iota(\bar{p}), \quad \iota(\bar{p}) \to \psi(\bar{p}, \bar{r})$$

are tautologies.

Proposition $\iota(\bar{p})$ is called an *interpolant* of $\phi(\bar{p}, \bar{q}) \to \psi(\bar{p}, \bar{r})$. The intuition behind this theorem is that the logical connection between two formulas must be mediated by common primitive parts of the formulas. The proof of this theorem is not difficult if one uses semantic considerations.

In general the size of the interpolant can be exponentially larger than the two propositions. Krajíček's crucial idea was that one may be able to bound the size of the interpolant by the length of the *proof* of $\phi(\bar{p}, \bar{q}) \to \psi(\bar{p}, \bar{r})$. This is indeed true for some propositional proof systems with one proviso: one has to consider the size of the interpolant not as a formula, but as a *circuit*. Using circuits we can define boolean functions in a more concise way. As this property is important, I will give a precise definition.

Definition 2 *A proof system P has the* feasible interpolation property*, if there is a polynomial time algorithm such that given an implication of the form $\phi(\bar{p}, \bar{q}) \to \psi(\bar{p}, \bar{r})$ and its proof d in the proof system P, the algorithm constructs a circuit $C(\bar{p})$ that interpolates the implication. In particular, the size of the circuit C is bounded by a polynomial in the length of the proof d.*

If a proof system P has the feasible interpolation property, then we can reduce the problem of proving lower bounds on proofs in P to proving lower bounds on boolean circuits. To this end we need to find a suitable implication which is a tautology and prove a lower bound on the size of any circuit that interpolates it. This reduces one difficult problem to another, probably even more difficult one. Yet this method is useful. There are plausible conjectures about the sizes of circuits from which one can prove that some concrete tautologies must be hard for P. The most remarkable fact is, however, that in many cases we can prove unconditional results (results that are not based on any unproved conjectures). That is because quite often we can prove that the interpolating circuits must have special properties.

In particular, for some proof systems one can prove that if the common variables occur in the two formulas only positively, then one can construct a *monotone* circuit. For monotone circuits exponential lower bounds have been proven. The fact that one can prove exponential lower bounds on the size of monotone circuits that compute certain function does not automatically give us lower bounds on the proofs, we need more. We need to show that small monotone circuits cannot interpolate certain implications. It is quite remarkable that the lower bounds on monotone circuits that had been proved ten years before feasible interpolation was discovered give us exactly what is needed. We do not know whether this is only a fluke or if there is a deeper reason behind it.

Lower bounds based on monotone feasible interpolation have been proven for the resolution calculus, the cutting-planes proof system, the propositional intuitionistic calculus and some other systems. Unfortunately the application of this method is limited to relatively weak proof systems. It has been shown that Frege systems do not have the feasible interpolation property (assuming that factoring is hard, a conjecture widely accepted in computational complexity).

Unprovability of circuit lower bounds. Why we are not able to prove nontrivial lower bounds on the size of circuits computing concrete boolean functions is an important problem. One explanation was given by the natural proofs of Razborov and Rudich (see page **??**). In proof complexity we would like to prove that the reason is because one needs to use strong axioms for such lower bounds, but the results obtained so far in this direction are rather weak. The following one is, however, interesting because it is connected with natural proofs.

Theorem 6.14 *Assuming the PRG-Conjecture (see page **??**) the following is true. If P is a propositional proof system with the feasible interpolation property, then for every boolean function f it is hard to prove in P any exponential lower bound on the circuit complexity of f.*

This theorem is stated very informally and requires more specification. Let n and m be numbers. We represent boolean functions of n variables by 0-1 strings of length 2^n that list the values of the function. We call these strings *truth tables.* Then we construct a formula $\alpha(\bar{x}, \bar{y})$ where $\bar{x}$ is a string of 2^n variables and $\bar{y}$ is some string of auxiliary variables whose length depends on the parameter m. In α the variables $\bar{x}$ encode a truth table of a boolean function of n variables and the variables y are used to encode a circuit of size m. The formula expresses that $\bar{x}$ is a boolean function that does not have a circuit of size m.

Let P be a propositional proof system with the feasible interpolation property. Let f be an n-variable boolean function whose complexity is strictly larger than m and let $\bar{a}$ be the truth table of the function. Then $\alpha(\bar{a}, \bar{y})$ is a tautology. The content of the theorem is that, assuming the PRG-Conjecture and $m \geq 2^{n^{\varepsilon}}$, for some $\varepsilon > 0$, then $\alpha(\bar{a}, \bar{y})$ does not have a polynomial size proof in P.

(Here we are comparing the sizes of proofs with the sizes of formulas. The size of the formula $\alpha(\bar{a}, \bar{y})$ is 2^{cn} for some constant c. Thus a proof of *polynomial size* means one of size 2^{dn}, for some constant d.)

It would be more interesting to prove a corresponding result for theories formalized in predicate logic, namely, to prove the unprovability of exponential lower bounds on concrete boolean functions in such theories. A result of this type was proved by Razborov, but, as I said before, the theories are rather weak and unnatural.

The weaker version of Razborov's result is stated for a theory $S_2^2[R]$, which is slightly stronger than $\Theta_{\mathbf{NP}}[R]$. In order to assess an incompleteness result, one has to see how the unprovable sentence is formalized. In the non-relativized theory S_2^2 circuits can only be formalized by numbers (we use the bit string defined by the binary expansion of the number to encode a circuit). Proving such a result for this formalization seems impossible with the currently available means. In $S_2^2[R]$ we can code circuits by R. The unary predicate enables one to encode bit strings that are exponentially longer than those coded by numbers, but one can prove much less about the objects formalized in this way. In particular, it is consistent with $S_2^2[R]$ that R encodes a one-to-one mapping from a domain to an exponentially smaller range. Thus it is not possible to prove superpolynomial lower bounds even on the representations by disjunctive forms.

Therefore, Razborov stated his lower bound for the theory that uses two oracles R and S. The theory can be represented by the expression $T_2[R] \cup T_2[S] \cup S_2^2[R, S]$. This means that the theory has induction for

1. all bounded formulas in the language of arithmetic extended by R and
2. all bounded formulas in the language of arithmetic extended by S.
3. For formulas containing both R and S, the induction schema is restricted as in S_2^2.

Razborov considered circuits that have two parts, joined by the exclusive-or connective. One part is represented by R and the other is represented by S. Thus, roughly speaking, one can argue about each part separately using arbitrary bounded formulas, but when speaking about the whole circuit one has to use only Σ_2^b formulas. Assuming the existence of strong pseudorandom number generators, Razborov proved that the theory is unable to prove superpolynomial lower bound on such boolean circuits computing the satisfiability of boolean formulas. On the other hand the theory is able to formalize the random restriction method and thus to prove the well-known exponential lower bounds on bounded depth circuits computing the parity function.

6.3 Feasible incompleteness

— **A** —

In the last section, we saw some interesting connections between complexity theory and logic. However, the fact that complexity theory theory is connected with logic does not imply that there must also be such a connection to the foundations of mathematics. Therefore in this section I will present more concrete connections between complexity theory and the foundations. I will be mostly talking about conjectures, so the arguments are rather speculative. Whether or not these argument do give such connections depends very much on whether or not such conjectures are true. I believe that they are true, but they seem even more elusive than the main problems in complexity theory. In any case the conjectures are concrete mathematical statements, hence they can be proven or refuted (at least in principle), so those who do not like speculations can view the conjectures simply as interesting open problems.

Before considering specific conjectures I want to state a general thesis. The thesis, which is not a mathematical statement, is intended to encompass various related conjectures that researchers working in proof complexity have contemplated, although most of them have never been stated explicitly. It is possible that we will eventually find a single conjecture that will imply all the others, which will allow us to proclaim this conjecture to be the formalization of the thesis.

The Feasible Incompleteness Thesis. *The phenomenon of incompleteness manifests itself at the level of polynomial time computations.*

There are two types of consideration that lead to the thesis. The first one concerns the problem of consistency. Due to Gödel's Incompleteness Theorem the program of proving the consistency of the foundations of mathematics by finitistic means, as proposed by Hilbert, is not realizable. The foundations of all contemporary mathematics are based on Zermelo-Fraenkel Set Theory. To prove the consistency of Zermelo-Fraenkel Set Theory, not only do finitistic means not suffice, but we need an even stronger theory than the theory itself. Being unable to prove consistency, we may ask if we can prove at least that there are no *short* proofs of contradiction. If, in particular, we could show that the smallest proof of contradiction has to be very large, say of length at least 10^9, we would be sure that we will never encounter such a proof in practice. We could view this as a partial realization of Hilbert's program. Also recall that, as we have seen in Section 1, inconsistent theories may be useful in the sense that their theorems with short proofs are true. So for practical purposes we do not need absolute consistency.

Notice that given a bound n we can always find a finitistic proof of at most exponential length of the statement expressing that a given consistent theory T does not prove contradiction by proofs of size n. This is because we can list all strings of length n and for each of them prove that it is not a proof of contradiction "by inspection". The question is whether there are shorter proofs than these exponential size "brute force" proofs. So we again encounter the problem of exponential expansion. It should not then come as a surprise that the problem is connected with open problems in complexity theory.

Incidentally, having a proof that some formal system for set theory is consistent up to proofs of large length would not be dissimilar from some open problems in number theory. There are several conjectures there, including the famous Riemann Hypothesis, that have been verified up to very large numbers. In most cases this has been done with the help of computers.

The second source of conjectures is proof complexity, namely, the theories that we call Bounded Arithmetic. The main open problems concern the separation of pairs of theories associated with the pairs of complexity classes that are conjectured to be different. We not only want to show that the sets of theorems provable in these theories are different, but moreover we would like to show that the theories differ already for low complexity theorems. Again, we are not able to accomplish this goal, thus we can only make conjectures.

It seems that there is nothing special about theories corresponding to low complexity classes, thus it is natural to state conjectures about general theories. Some conjectures are quite specific, with little relation to classical incompleteness results, but some have the distinctive flavor of the incompleteness phenomenon. They say that every theory has only a limited ability to formalize complex functions and complex sets, which means that if the complexity of sets and functions is too high, the theory is unable to prove their properties. In other words, the conjectures say that *a possible cause of incompleteness is computational complexity.*

When we do not have proofs we argue by analogy. The analogy here is that we know that the unprovability of certain sentences can be caused by the fact that they describe a function that grows very fast, so fast that the theory is unable to prove that such large numbers exist. If the extremely fast growth of a function can cause unprovability, then why not the extreme complexity of a function?

These two kinds of reasoning that lead to the Feasible Incompleteness Thesis also produce two different types of conjectures. The first type concerns the constructibility of proofs and the length of proofs of certain sentences. The second type concerns the provability of sentences connected with problems in computational complexity. However, I will show that there is a conjecture that can be interpreted in both ways, which supports my feeling that there should only be one thesis.

All but one of the conjectures that I am going to state have equivalent versions stated using only concepts from computational complexity. So we can also view them as conjectures from complexity theory that have important consequences in proof complexity. This fact again confirms the close ties between the two fields. The form of these versions is typically a statement saying that some class of problems does not have a complete member, that is, one to which all others can be efficiently reduced. The equivalences and the implications between conjectures are fairly easy and I will sketch the proofs of most of them.

The reason I state the Feasible Incompleteness conjectures is that it is unlikely that the related problems are going to be solved soon. It may even happen that the set-theoretical axioms that we use, however strong they are, are not suited for problems in the theory of computational and proof complexity. If that is the case, then we need two things:

1. Some arguments (which do not have to be completely formal) showing that this is indeed the case.

2. New axioms that will enable us to decide the problems.

I imagine that this could be achieved by a system of conjectures that would do both—explain why the problems are not provable in the current formal systems and also decide them. Therefore we should look for general principles that could be accepted as such new postulates. Whether or not the Feasible Incompleteness Thesis points in the right direction is yet to be seen.

— B —

The feasible consistency problem

I will now consider the first kind of problem in more detail. As we have seen before, the standard way of formalizing the intuitive concept of feasibility is by considering arbitrarily large instances and using polynomial bounds. Instead of bounds for concrete numbers we shall rather study how the lengths of proofs of consistency grow as we increase the bounds on the lengths of proofs. Further, it is also natural not to limit ourselves to specific theories, but rather to consider a general situation. So I will assume that we have two theories S and T, both consistent and both formalized by a finite number of axioms. The latter condition excludes some important theories, such as Peano Arithmetic and Zermelo-Fraenkel Set Theory, but all that I am going to say will also hold true for them. It is simply a convenient assumption to avoid the complications caused by infinite sets of axioms. T is the theory whose consistency we are studying and S is the theory in which we are proving facts about T. You can think of T as being a strong theory that we might need for the foundation of mathematics (say a finite part of Zermelo-Fraenkel Set Theory) and S as being a fairly weak theory about whose soundness we do not have any doubts (a theory that formalizes *"finite means"*, such as a finite part of Peano Arithmetic), but the conjectures will be quite general. I will need to distinguish between proofs in S and T using a short expression, so I will call the former *S-proofs* and the latter *T-proofs*.

Given such theories S and T we want to know how difficult is to prove the sentences:

> *"there is no proof of contradiction derivable from the axioms of T whose length is at most n,"*

for specific numbers $n = 1, 2, \ldots$. I will abbreviate these sentences by saying *"T is consistent up to length n"* and I will use the formulas

$$Con_T(n)$$

to denote them. I will call *'the feasible consistency problem'* for these theories the following question:

> *If one is allowed to argue only in the theory S, how difficult is it to prove the sentence expressing that T is consistent up to length n, where n is a concrete number?*

The most interesting values of n are those that estimate proofs that can actually be written down, but we prefer to study the more mathematical question of how the length of such proofs increases with increasing n. This problem is stated informally, so let us consider how it can be formalized. One possibility is:

Problem 4 *Do the sentences $Con_T(n)$ have S-proofs of length at most $p(n)$ for some polynomial p?*

We can measure the difficulty of proving the sentences $Con_T(n)$ not only by the length of S-proofs, but also by the difficulty of finding them. Thus we have another formalization of the problem above:

Problem 5 *Is there a polynomial time algorithm for finding S-proofs of $Con_T(n)$?*

Clearly, a positive answer to Problem 5 implies a positive answer to Problem 4, since an S-proof constructed by a polynomial time algorithm can only be of polynomial length. I conjecture that if S is essentially weaker than T, then the answer to Problem 4 is negative, which implies a negative answer to Problem 5.

These problems are most interesting when T is strong and S is weak, but let us also see what happens in general.

1. **Suppose that T is weak and S is strong** (which is the opposite of what we are actually interested in). The consistency of T is clearly equivalent to the statement that $Con_T(n)$ is true for all n. Formally this is the sentence $\forall x\ Con_T(x)$. Suppose that S is so strong that it proves the consistency of T,

thus it proves $\forall x\, Con_T(x)$. Then, to get $Con_T(n)$, an instance of the general statement $\forall x\, Con_T(x)$, we need only to apply a single logical rule, the rule of deriving a specific instance from a general statement. Thus in such a case the proofs are always short and easy to construct. In fact, if n is represented in binary, the lengths of such proofs will only be logarithmic in n, which is much shorter than polynomial n.

2. **Suppose that $T = S$.** This is an important special case, since it corresponds to Gödel's Second Incompleteness Theorem. This similarity suggests that Problems 5 and 4 should have negative answers, but the opposite is true. There are polynomial size proofs of $Con_T(n)$ in T and they can be constructed in polynomial time. (In fact there are proofs of length $c \cdot n$, see part C.) Although the analogy with the Second Incompleteness Theorem fails, there is still a difference between this and the previous case. In the previous case (when S proved the consistency of T) we had logarithmic size proofs, assuming n was represented in binary. In the present case ($T = S$) this is not true anymore. No matter how concisely we represent n, the lengths of the proofs of $Con_T(n)$ are still close to n. So, in a sense, we do get an exponential difference between these two cases.

This lower bound was first proved by Harvey Friedman in 1979. Since it is an important result I state it explicitly.

Theorem 6.15 *For every consistent and finitely axiomatized theory T, there exists $\varepsilon > 0$ such that, for every n, every T-proof of $Con_T(n)$ has size at least n^{ε}.*

In other words, the length of the smallest T-proof of $Con_T(n)$ grows asymptotically at least as fast n^{ε}. There is still a gap between the lower and upper bounds; the true value seems to be very close to n.

3. **Suppose that T is essentially stronger than S.** To ensure this relation we may suppose, for example, that T proves the consistency of S. The feasible consistency problem in this case is wide open. Since T is at least as strong as S, the sentence $Con_T(n)$ implies $Con_S(n')$, for n' at most polynomially smaller than n. Thus Friedman's Theorem (applied to S) tells us that the shortest S-proofs of $Con_T(n)$ have lengths at least $n^{\varepsilon'}$, for some $\varepsilon' > 0$. But for all we know, the shortest S-proofs of $Con_T(n)$ may be of exponential length. It would indeed be interesting to prove this, but again it is the kind of statement which we do not have the slightest idea how to prove.

The relation to the P vs. NP problem

In fact, it may be as difficult as proving $\mathbf{P} \neq \mathbf{NP}$. To see the relation between the feasible incompleteness problem and problems in complexity theory, I will present an argument that in several steps naturally leads to a conjecture about the feasible incompleteness problem.

(i) *Assume* $\mathbf{P} \neq \mathbf{NP}$. *Then there is no polynomial time algorithm for deciding if a sentence has a proof of length at most n in the predicate calculus.*

This is because this problem (properly formalized) is an **NP**-complete problem. Thus, in fact, the latter statement is equivalent to $\mathbf{P} \neq \mathbf{NP}$.

(ii) Hence *if* $\mathbf{P} \neq \mathbf{NP}$, *there do not exist a consistent finitely axiomatized theory S and an algorithm A with the following property: given a sentence ϕ and a number n, such that ϕ does not have a proof of length at most n in predicate calculus, A constructs an S-proof of this fact in polynomial time ('polynomial' meaning 'polynomial in n').*

This is because constructing a proof of some sentence is at least as hard as deciding if the sentence is true. Naturally, the reason why there is no such algorithm for a theory S may be that there are no such proofs. But since the sentences in question talk about finite things, every sufficiently strong S can prove them. So the reason may be rather that the proofs are more than polynomially long. If we assume a little bit more, then this is indeed what happens. I shall state this explicitly.

(iii) *Assume* $\mathbf{coNP} \neq \mathbf{NP}$. *Then there do not exist a consistent finitely axiomatized theory S and a polynomial p with the following property: for every number n and every sentence ϕ, if ϕ does not have a proof of length n in the predicate calculus, then there is an S-proof of length at most $p(n)$ of (the formalization of) the statement that ϕ does not have a proof of size n.*

Again, the assumption and the conclusion are in fact equivalent. Let us reformulate the nonexistence of S and p as follows:

For every consistent finitely axiomatized theory S there exists a sequence of sentences $\{\phi_1, \phi_2, \phi_3, \ldots\}$, where ϕ_n does not have a proof of length n, such that the sentences

'ϕ_n does not have a proof of length at most n'

do not have polynomially bounded S-proofs.

We would surely like to know what the sentences ϕ_n are, but the mere assumption that **coNP** $\neq$ **NP** does not give us any hint. We may also wonder whether the sequence is in some sense uniform. The most uniform sequence is a sequence in which all elements are the same. If all sentences ϕ_n are one fixed sentence ϕ, then the condition about the nonexistence of proofs of length n is simply the condition that ϕ is unprovable in predicate calculus. Let T be the theory axiomatized by $\neg\phi$ then the condition that ϕ is unprovable is equivalent to T being consistent. Similarly, the conclusion of the statement above is that the sentences expressing the consistency of T up to proofs of length n do not have polynomially bounded S-proofs. Thus we have arrived at the main conjecture associated with the Feasible Incompleteness Thesis.

Conjecture 1 *For every consistent finitely axiomatized theory S, there exists a consistent finitely axiomatized theory T such that the sentences $Con_T(n)$ do not have polynomially bounded S-proofs.*

Let us analyze why the conjecture seems probable. To this end I shall state a simple consequence of Gödel's Second Incompleteness Theorem.

For every consistent finitely axiomatized theory S, there exists a consistent finitely axiomatized theory T such that S does not prove the consistency of T.

We know that one can take $T = S$, which is Gödel's theorem, but suppose we did not know it. Then we could explain the theorem as follows. If one randomly produces a finite set of axioms (which is the same as producing a single sentence), then it is difficult to decide if the set is consistent or not. If there were a theory S that, contrary to the statement above, could prove the consistency of every consistent theory, then it would help us to decide this difficult question; in fact, it would give us an algorithm. But since there is no algorithm for deciding consistency, there cannot be such a theory.

The Feasible Incompleteness Thesis says that if we limit ourselves to polynomial time computations and polynomial size proofs, we should see essentially the same picture as in the unlimited case. Thus it seems unlikely that we can prove the consistency of T up to n by a short S-proof when T is much stronger than S.

Let us return to the connections with problems in complexity theory. I started with a statement equivalent to **P** $\neq$ **NP**, then I made it stronger. The stronger version was equivalent to **coNP** $\neq$ **NP**. Then I made it more specific, thus again stronger. So the resulting Conjecture 1 is at least as strong as **coNP** $\neq$ **NP**, which in turn implies **P** $\neq$ **NP**. Thus Conjecture 1 not only talks about an important problem in proof complexity and the foundations of mathematics, it also decides the most important problem in complexity theory.

In general the fact that some conjecture implies **P** $\neq$ **NP** is not surprising; one can easily produce lots of such conjectures. What is remarkable, however, is that Conjecture 1 talks about consistency and provability. Thus we can support our belief in **P** $\neq$ **NP** by an essentially different type of reasoning.

Conjecture 1 can be equivalently expressed using only concepts from computational complexity theory; I will show this in part C. Here I will mention another equivalent formulation that is based on the concept of the propositional proof system.

Call a propositional proof system P *length-optimal* if, for every propositional proof system Q, there exists a polynomial p such that for every tautology τ, if there is a proof of length n of τ in Q, then there is a proof of length at most $p(n)$ of τ in P. In plain words, if there exists a short proof of a tautology in any proof system, then there also exists a short proof of the tautology in P. Conjecture 1 is equivalent to the following statement:

Conjecture 1$'$ *There exists no length-optimal proof system.*

The proof of the equivalence of conjectures 1 and 1$'$ is based on two constructions:

1. a construction of a propositional proof system for an arbitrary (sufficiently strong) consistent theory;

2. a construction of a theory for an arbitrary propositional proof system.

We already know that some theories can be naturally associated with propositional proof systems; but this can be done only for some special theories. Fortunately, for proving the equivalence, we do not need theories and propositional proof system to be associated so tightly. The nature of the statements that we want to prove to be equivalent is, so to speak, asymptotic. Thus we only need

1. for every theory a propositional proof system that is strong enough to simulate all provable $\Pi_{\mathbf{P}}$-sentences;

2. for every propositional proof system a theory that is strong enough to prove its soundness.

Such calculi and theories can be constructed quite easily. Given a theory T we define a propositional proof system P_T as follows. Let $taut(x)$ be a formula expressing that x is a tautology. Then a proof of a tautology ϕ in P_T is simply a proof of $taut(\phi)$ in T. As concerns the second construction, given a propositional proof system P, we simply take a sufficiently strong base theory and extend it by the axiom saying that P is a sound proof system.

This enables us to switch between theories and proof systems. If S were a theory that could prove $Con_T(n)$ shortly for all finite consistent theories, then the proof system P_S would be length-optimal. Vice versa, if a proof system P were length-optimal, then the theory S with the axiom that P is a sound proof system would prove the sentences $Con_T(n)$ shortly. (For more details see the proof of the equivalence of related Conjectures 4 and $4'$ in part C.)

Conjecture 1 is not specific about how strong a theory T must be, given the theory S, and how big the lower bound is. The most common way of making a theory stronger is to add its consistency (which is unprovable in it according to the Second Incompleteness Theorem). Concerning the lower bound, we can conjecture that it is close to the best upper bound that we have, which is exponential. Thus we arrive at the following stronger conjecture (proposed by Jan Mycielski when we were discussing Conjecture 1).

Conjecture 2 *Let S and T be finitely axiomatized consistent theories. If T proves the consistency of S, then the shortest S-proofs of $Con_T(n)$ have exponential lengths.*

Can computational complexity cause incompleteness?

Now I will consider a different kind of conjecture. It also concerns unprovability, therefore I view it as another specific instance of the Feasible Incompleteness Thesis.

The conjecture is about the provability of sentences of the form

$$\forall x \exists y \ \phi(x, y), \tag{6.11}$$

where ϕ represents a binary relation computable in polynomial time. In Chapter 4 (page **??**) we saw concrete examples of independent sentences of this form. These sentences had the property that they described very fast growing functions and the independence of these sentences was caused by the fact that the theories were not able to prove the existence of such rapidly growing functions. In this chapter we saw (page 16) that one can define hierarchies of functions indexed by constructive ordinals, by which one can measure the strength of such sentences.

We met sentences of this form again in Section 6.2 of this chapter. There they were used to define **TPS**, the class of total polynomial search problems. Recall that *total* means that the sentence (6.11) is true (thus the sentence implicitly defines a function defined for all x) and *polynomial* means that the relation ϕ is decidable in polynomial time and, for a given x, the length of every y satisfying $\phi(x, y)$ is polynomially bounded by the length of x. I briefly mentioned the possibility that with increasing strength, theories can formalize more and more search problems. This is essentially the conjecture I am going to describe in more detail now.

The basic idea is as follows. We know that the unprovability of a sentence of the form $\forall x \exists y \ \phi(x, y)$ can be caused by the fact that the function it implicitly defines grows very fast. If the length of y is polynomially bounded by the length of x, then the unprovability cannot be caused by the growth of the implicitly defined function, but it is possible that it is caused by the *computational complexity* of the function. Let x be a fixed number of length n, and suppose the bound on the length of y is $p(n)$ for some polynomial p. Although the length of y is polynomially bounded, we only know that it is somewhere in an exponentially large set (namely, a set of size $2^{p(n)}$, if we consider binary strings). If one can only use polynomial time algorithms, it does not help that this set is finite; it is like exploring an infinite set. Therefore it is conceivable that there is an infinite hierarchy of total polynomial search problems, like there is an infinite hierarchy of fast growing functions, and that every theory can formalize search problems only up to certain level. If this is indeed the case, we can say that the incompleteness concerning such sentences is caused by computational complexity.

Recall that a theory T properly formalizes a search problem defined by a binary relation r if $r(x,y)$ is defined by a formula $\phi(x,y)$ from the class $\Phi_{\mathbf{P}}$ and T proves $\forall x \exists y\ \phi(x,y)$. I will say that a search problem R_1 is *strictly stronger* than a search problem R_2, if R_2 is polynomially reducible to R_1, but R_1 is not polynomially reducible to R_2. Now I am ready to state the conjecture.

Conjecture 3 *For every finitely axiomatized consistent theory T, there exists a total polynomial search problem Q which is strictly stronger than all search problems that T properly formalizes.*

To see that the conjecture is about incompleteness, consider T and Q with the properties stated in the conjecture. Then for any formula $\psi(x,y)$ that defines Q, the theory T does not prove $\forall x \exists y\ \psi(x,y)$.

But there is a caveat here: the fact that there exist unprovable sentences of this form is a trivial consequence of Gödel's Incompleteness Theorem. In fact, for every total polynomial search problem R we can find a formula $\phi(x,y)$ which defines R such that $\forall x \exists y\ \phi(x,y)$ is not provable in T by incorporating the consistency of T into $\phi(x,y)$ (see part C). The essence of the conjecture is that *the unprovability of $\forall x \exists y\ \psi(x,y)$ is caused by the complexity of Q,* which is expressed by saying that the sentence is unprovable for representations of Q by *any $\Phi_{\mathbf{P}}$-formula.*

Again, this conjecture has several interesting consequences in computational complexity theory. An immediate consequence of the conjecture is that the hierarchy of total polynomial search problems is infinite in the following sense: there is no total polynomial search problem to which one could reduce every total polynomial search problem. The latter statement implies $\mathbf{P} \neq \mathbf{NP}$. Furthermore, it has an equivalent stated only using the concepts of search problems and reducibilities.

Conjecture 3′ *There exists no complete problem among total polynomial search problems, that is, no problem to which total polynomial search problems are reducible.*

The equivalence of Conjectures 3 and 3′ is a consequence of the existence of a complete problem among those for which the condition of totality is provable in the theory T (see part C).

The above conjecture concerns sentences with two quantifiers and a polynomial time computable relation. The natural question is whether one can state a conjecture of this kind also for sentences with only one quantifier, namely for $\Pi_{\mathbf{P}}$-sentences. I will present such a conjecture in the next subsection and give another example in part C.

A connection between the two parts of the thesis

We have seen two types of conjectures that are instances of the Feasible Incompleteness Thesis. At first they may seem very different. The first type of conjectures concern the lengths of proofs, whereas the second type is about unprovability. Yet they are not so different. I will give an example of a conjecture that is slightly weaker than Conjecture 1, but can also be stated in the form very similar to Conjecture 3.

Conjecture 4 *For every consistent finitely axiomatized theory S there exists a consistent finitely axiomatized theory T such that S-proofs of sentences $Con_T(n)$ cannot be produced by an algorithm in time $p(n)$, for p a polynomial.*

Notice that Conjecture 1 gives a stronger reason why sentences $Con_T(n)$ are hard for S, namely, that their proofs are more than polynomially long. The above conjecture only says that it is difficult to find these S-proofs. Like Conjecture 1, it can also be stated in terms of propositional proof systems. To this end, define that a *propositional proof system P is optimal* if it polynomially simulates every propositional proof system. This property is, clearly, stronger than being length-optimal.

Conjecture 4′ *Optimal propositional proof systems do not exist.*

The fact that Conjectures 1 and 4 can be stated in terms of propositional proof systems (as Conjectures 1′ and 4′) shows that in a sense the hardness of sentences $Con_T(n)$ is caused by complexity. More precisely, it is caused by the existence of infinite hierarchies of propositional proof systems. But we can get a statement whose form is much closer to Conjecture 3 by blending the two versions of Conjecture 4 into one. Loosely speaking, the conjecture says that the complexity of propositional proof systems causes the unprovability of their soundness.

Conjecture 4″ *For every finitely axiomatized consistent theory T, there exists a propositional proof system P such that T does not prove the soundness of any formalization of P by a $\Phi_{\mathbf{P}}$-formula.*

Recall that soundness means that all propositional formulas provable in the proof system are tautologies. The soundness of P can be expressed by the $\Pi_{\mathbf{P}}$-sentence:

> *For every P-proof d, every propositional formula ϕ and every truth assignment a to the variables of ϕ, if d is a proof of ϕ, then ϕ is satisfied by a.*

Thus we have a natural conjecture about the unprovability of $\Pi_{\mathbf{P}}$-sentences. (For the proof of the equivalence of these conjectures, see part C.)

*

The subject of feasible incompleteness has not yet received the attention it deserves yet. Most researchers are convinced that solving the big open problems in complexity theory, such as proving $\mathbf{P} \neq \mathbf{NP}$, can be achieved by developing better combinatorial methods or using other parts of mathematics which have deeper theories, such as algebraic geometry, group representation theory etc. But we have seen in this chapter that computational complexity has many connection with logic and in particular with proof complexity. The connections with feasible incompleteness show a link to the foundations of mathematics. In light of this evidence it is conceivable that the open problems in complexity theory may require a fundamentally different approach. If one wants to embark on this project, the natural starting point is the study of feasible incompleteness conjectures.

— C —

Bounds on the proof complexity of $Con_T(n)$. First I will outline the *proof of Theorem 6.15.* I will assume that T is finitely axiomatized, consistent and sufficiently strong for formalizing syntax. The following notation comes in handy for this proof. For a sentence ϕ, I will denote by $||\phi||_T$ the length of the shortest proof of ϕ in T, if there is one, and otherwise $||\phi||_T = \infty$. Further, I will denote by $\bar{n}$ the binary numeral representing a natural number n. The binary numeral is the term of length approximately $\log n$ defined as follows. If we have the successor functions s_0 and s_1, then we use the representation given by the formula (6.5), page 36. Otherwise we replace the successor functions by the terms that define them and use the same formula, or use a formula that defines this expression, in case there are no such terms. I will also assume that the rule of *modus ponens* is in the proof system for predicate logic considered here.

Now the theorem I want to prove is stated simply as the following inequality for all n:

$$||Con_T(\bar{n})||_T > n^{\varepsilon}, \tag{6.12}$$

where $\varepsilon > 0$ is a constant. However, instead of this, I will only prove a closely related inequality

$$||Con_T(\bar{n}^c)||_T > n - c' \log n, \tag{6.13}$$

for some constants c and c'. Intuitively this is the same as (6.12) with ε smaller than $\frac{1}{c}$, but formally it needs a little argument which I am omitting.

The proof is based on the Diagonal Lemma **??** (page **??**) in a slightly more general form. According to that lemma, one can construct a formula $\delta(x)$ such that the sentence

$$\forall x\ (\delta(x) \equiv ||\delta(\bar{x})||_T > x) \tag{6.14}$$

is provable in T.

The first step is to prove that $\delta(\bar{n})$ is true for every n which is, by (6.14),

$$||\delta(\bar{n})||_T > n. \tag{6.15}$$

Indeed, suppose the contrary $||\delta(\bar{n})||_T \leq n$. Then in particular, $\delta(\bar{n})$ is provable in T. By Σ completeness, $||\delta(\bar{n})||_T \leq n$ is provable in T. On the other hand, since T proves $\delta(\bar{n})$, it also proves $||\delta(\bar{n})||_T > n$ according to (6.14). Thus T is inconsistent. Since we are assuming that T is consistent, we have obtained a contradiction, hence (6.15) must be true.

The part of the proof that we have just done corresponds to the proof of the First Incompleteness Theorem. As in the proof of the Second Incompleteness Theorem, we will now formalize the above proof in T. However we need more—we need also to estimate the lengths of proofs of the formulas occurring in the proof.

When proving (6.15), we have shown that the assumption $||\delta(\bar{n})||_T \leq n$ implies that T is inconsistent. One can estimate the lengths of proofs and show that if $\delta(\bar{n})$ has a proof of length at most n in T, then a contradiction has length at most n^c, for some constant, written formally

$$||\delta(\bar{n})||_T \leq n \ \rightarrow \ ||0 = 1||_T \leq n^c.$$

Let us look at the contrapositive implication where the negation of the consequent is replaced by consistency up to n^c

$$Con_T(\bar{n}^c) \ \rightarrow \ ||\delta(\bar{n})||_T > n.$$

Using (6.14) we can replace the consequent by $\delta(\bar{n})$. Thus we get

$$Con_T(\bar{n}^c) \ \rightarrow \ \delta(\bar{n}). \tag{6.16}$$

Since $\delta(\bar{n})$ follows from $Con_T(\bar{n}^c)$ and the sentence above by *modus ponens*, we have

$$||\delta(\bar{n})||_T \leq ||Con_T(\bar{n}^c)||_T + ||Con_T(\bar{n}^c) \rightarrow \delta(\bar{n})||_T.$$

We can also write it as

$$||\delta(\bar{n})||_T - ||Con_T(\bar{n}^c) \rightarrow \delta(\bar{n})||_T \leq ||Con_T(\bar{n}^c)||_T.$$

We already have a lower bound on $||\delta(\bar{n})||_T$, (6.15), so in order to prove a lower bound on $||Con_T(\bar{n}^c)||_T$, it remains to show an upper bound on $||Con_T(\bar{n}^c) \rightarrow \delta(\bar{n})||_T$. The reason why it has a proof of logarithmic length is that (6.16) is provable in T as the universal sentence

$$\forall x(Con_T(\bar{x}^c) \ \rightarrow \ \delta(\bar{x})).$$

Thus to prove a numeric instance of it, we only need to substitute the numeral representing n, which has logarithmic length. This finishes the proof of (6.13)

For stating the upper bound on the lengths of proofs of finite consistencies we need the concept of a sequential theory. Essentially, a theory T is sequential if there are formulas for coding and decoding finite sequences of arbitrary objects of the theory. The basic axioms these formulas must satisfy are, stated informally:

1. an empty sequence has a code,
2. given a sequence x and an object y, one can code a sequence that is the prolongation of x by object y.

More important than the details of the definition is the fact that it is a natural property satisfied by many theories.

Theorem 6.16 *For every sequential theory T, there exists a constant c such that for all n*

$$||Con_T(\bar{n})||_T \leq cn.$$

The proof of the theorem uses the fact that in sequential theories it is possible to define *partial truth predicates.* A partial truth predicate is a formula that defines truth for a subset of all formulas. (Recall that by the Gödel-Tarski Theorem **??** it is not possible to define truth for all formulas inside a theory.) One can show that

(i) a proof of size n can only use formulas of quantifier complexity at most $\sqrt{n}$,

(ii) one can define truth for formulas of quantifier complexity $\sqrt{n}$ by a formula of linear size.

Having such a partial truth predicate, the proof of consistency up to n is quite standard—we prove that the axioms are true and then by induction that all sentences in a proof of length n are true. Since a contradiction is a false sentence, it cannot occur in the proof. This proof has linear size, because the truth predicate has linear size.

Theories with infinite sets of axioms. All the results stated in this section can be generalized to theories with an infinite number of axioms. The only reason for using finitely axiomatized theories was that I needed the proof system to be defined by a polynomial time decidable relation. In order for the relation *'d is a proof of sentence ϕ'* to be decidable in polynomial time, we do not need a finite set of axioms, it suffices to have a set of axioms such that there exists a polynomial time algorithm for deciding if a sentence is an axiom. If this condition is satisfied and the proof relation is formalized by $\Phi_{\mathbf{P}}$-formulas, then all proofs go through. Theories such as Peano Arithmetic and Zermelo-Fraenkel Set Theory have sets of axioms defined by simple syntactical conditions, hence decidable in polynomial time.

Conjecture 1 in purely computational terms. I have mentioned that Conjecture 1 implies some separations of pairs of complexity classes. In particular we know the following implications

$$\text{Conjecture 1} \rightarrow \mathbf{NEXP} \neq \mathbf{coNEXP} \rightarrow \mathbf{NP} \neq \mathbf{coNP} \rightarrow \mathbf{P} \neq \mathbf{NP}.$$

Thus $\mathbf{NEXP} \neq \mathbf{coNEXP}$ is the strongest separation among the three; yet it does not seem to be equivalent to the conjecture. To state the computational version of the conjecture we need the following concept. We say that a set X of strings in a finite alphabet is *polynomially sparse* if there exists a polynomial $p(x)$ such that for every number n, the number of strings of length n is at most $p(n)$.

The following statement is equivalent to Conjecture 1:

Let A be a **coNP***-complete set. Then there is no nondeterministic Turing machine M with the following properties:*

1. *M accepts X (i.e., X is the set of strings accepted by M);*
2. *for every polynomially sparse set $X \subseteq A$, $X \in \mathbf{P}$, there exists a polynomial $q(x)$ such that M accepts every string $w \in X$ in time $q(|w|)$.*

The inequality $\mathbf{NP} \neq \mathbf{coNP}$ is equivalent to the statement that a **coNP**-complete set cannot be accepted by a nondeterministic Turing machine in polynomial time. The above statement strengthens it by saying that even sparse subsets that are in **P** cannot be accepted by such a machine in polynomial time.

Although rather complicated, this version of the conjecture has an advantage—since it is stated purely in terms of Turing machines and computations, it can be relativized. An oracle has been constructed with respect to which the conjecture is true, thus we have some evidence supporting Conjecture 1.

xxxxxxxxxxxxxxxxxxxx citation ♠

Making formulas artificially stronger. Suppose a total polynomial search problem P is defined by a formula $\forall x \exists y\ \phi(x, y)$ and let T be a consistent finitely axiomatized theory. We can make a formula that defines the same search problem and that is unprovable in T by forming the conjunction of $\forall x \exists y\ \phi(x, y)$ with Con_T. If we write this as $\forall x \exists y\ \phi(x, y) \wedge Con_T$, it does not look like a definition of a search problem, but we can express it equivalently by:

$$\forall x \exists y\ (x \text{ is not a proof of contradiction in } T \wedge \phi(x, y)).$$

This shows that it is very important which formula we use to represent a search problem in a theory T.

"Syntactical" and "semantical" complexity classes. Complexity theorists distinguish complexity classes that are defined syntactically and those that are defined semantically. A class $\mathcal{C}$ is syntactically defined if, for a Turing machine M, we can decide whether or not it defines an element of $\mathcal{C}$ "simply by looking at it", whereas if $\mathcal{C}$ is only defined semantically, we have to "test M for what it actually does". I propose the following definition instead of this vague description.

Definition 3 *A class $\mathcal{C}$ is* syntactically *defined if there exists an algorithmically decidable set of Turing machines $\mathcal{M}$, such that for every X, $X \in \mathcal{C}$ if and only if there exists an $M \in \mathcal{M}$ such that M defines X (i.e., X is the set of inputs accepted by M). Otherwise the class is defined* semantically.

Examples of syntactical classes are **P** and **NP**. The standard definition of **P** is not syntactical, since it is algorithmically undecidable if a Turing machine runs in polynomial time. Therefore, for the class **P**, we take $\mathcal{M}$ to be Turing machines *equipped with a clock running in polynomial time*. This means that the program for the Turing machine contains a simple routine that counts the number of steps and stops the machine after polynomial time, disregarding whether or not the other computations are finished. The set of Turing machines for **NP** is defined in the same way except that we use nondeterministic machines. A class that is not known to be syntactical is **BPP**. (But if the conjecture $\mathbf{BPP} = \mathbf{P}$ is true, then it is syntactical.)

The rule of thumb is that syntactical classes possess complete sets, while semantical classes do not. Thus there are **NP**-complete sets but no **BPP**-complete sets are known.

One can easily show that syntactically defined classes have a property that links this concept with provability in theories.

Proposition 6.17 *If $\mathcal{C}$ is syntactically defined, then there exists a true theory T such that for every $X \in \mathcal{C}$, there exists a Turing machine M such that M defines X and T proves the sentence* 'the set of inputs accepted by M is in $\mathcal{C}$'.

If $\mathcal{C}$ is not syntactically definable, then it is possible that there there is no such theory. If that is the case, it still does not imply that the sets in $\mathcal{C}$ form an infinite hierarchy with respect to the quasiordering by polynomial reductions; there may exist a complete problem in $\mathcal{C}$. The conjecture that they do form an infinite hierarchy can be viewed as an instance of the Feasible Incompleteness Thesis. Conjectures $1'$ and $3'$ are statements of this form, except that they talk about search problems (objects determined by binary relations) rather than sets. Next is another such example.

A conjecture about $\Pi_{\mathbf{P}}$-sentences. An interesting concept related both to computational complexity and proof complexity is the concept of pairs of disjoint **NP** sets. The words define this concept quite precisely—we consider pairs (A, B), where $A \cap B = \emptyset$ and $A \in \mathbf{NP}$, $B \in \mathbf{NP}$. We say that a pair (A, B) is *polynomially reducible* to a pair (A', B'), if there exists a polynomial time computable function f which maps A into A' and B into B'. We say that a pair (A, B) is *polynomially separable*, if it is polynomially reducible to the pair $(\{0\}, \{1\})$.

In complexity theory such problems are called *'promise problems'*. The "promise" is that we only get inputs from A or from B. The problem is to decide from which set the input comes. The promise is what makes the definition of the class of problems "semantical".

Clearly, if (A, B) is polynomially reducible to (A', B') and (A', B') is polynomially separable, then so is (A, B). We conjecture that there are pairs that are not polynomially separable. This conjecture is a consequence of the conjecture $\mathbf{NP} \cap \mathbf{coNP} \neq \mathbf{P}$. The latter conjecture, in turn, follows from some cryptographic conjectures, such as the conjecture that there exist one-way permutations.

The following is a stronger conjecture than the conjecture about the existence of polynomially inseparable pairs.

Conjecture 5 *There is no complete disjoint* **NP** *pair. In more detail, there is no pair of disjoint* **NP** *sets (C, D) such that every pair of disjoint* **NP** *sets is polynomially reducible to (C, D).*

This conjecture is supported by a result of C. Glaßer, A. Selman, S. Sengupta and L. Zhang that there exists an oracle with respect to which the conjecture is true. An equivalent conjecture concerning provability in theories is the following:

Conjecture 5$'$ *For every finitely axiomatized consistent theory T, there exists a pair of disjoint* **NP** *sets (A, B) such that for no formal definitions of the sets A and B is it provable in T that A is disjoint from B.*

This is similar to Conjecture 3, but the sentences it concerns are $\Pi_{\mathbf{P}}$. Indeed, let $\exists y\ \phi(x, y)$ be a definition of A and $\exists y\ \psi(x, y)$ be a definition of B. Then the disjointness of A and B can be expressed by:

$$\forall x \forall y \forall z\ (\neg\phi(x, y) \vee \neg\psi(x, y)).$$

Since $\phi(x, y)$ and $\psi(x, y)$ define polynomial time computable relations, this sentence is $\Pi_{\mathbf{P}}$.

One can show that Conjecture 5 implies Conjecture 4.

The equivalence of Conjectures 3 and 3$'$. (3)$\Rightarrow$(3$'$). This direction is easy. Suppose Conjecture 3$'$ is false. Let S be a complete **TSP**. Let T be a sufficiently strong finitely axiomatized fragment of arithmetic or set theory augmented with the axiom expressing that R is total. Then such a T shows that Conjecture 3 is false.

(3$'$)$\Rightarrow$(3). This implication requires us to be more specific about how we formalize **TPS** problems. Suppose a search problem S is defined using a binary relation $R(x, y)$, where x's are inputs and y's are potential solutions. Further we have a polynomial bound p on the lengths of solutions y, namely, we require $|y| \leq p(|x|)$, where $|\dots|$ denotes the length. There is one more implicit polynomial bound, which is the bound on the running time of the Turing machine computing the relation R. When S is formalized in some theory, we also need that the theory is able to prove the inequality $|y| \leq p(|x|)$ for the solutions and the polynomial upper bound on the running time of the Turing machine.

For **TPS** problems S_1 and S_2, I will use the inequality $S_1 \leq S_2$ to denote that S_1 is polynomially reducible to S_2. An easy fact is that for every S_1 and S_2 there exists an S such that $S_1 \leq S$ and $S_2 \leq S$. Simply take the product of the relations defining S_1 and S_2. Slightly less trivial is the following lemma.

Lemma 6.18 *For every finitely axiomatized consistent theory T, there exists a* **TPS** *problem S_T, such that every* **TPS** *problem properly formalized in T is polynomially reducible to it.*

Proof. I will define the binary relation $R(x, y)$ for S_T. Let us view x as consisting of three parts. The first part will be an input z to a search problem; the second part will be a description of a search problem S and a T-proof that it is total. The third part will be an arbitrary string whose purpose is to pad the input to a sufficient size so that its length bounds the length of the solutions of S and the time needed to compute the binary relation of S. If x has such a form, then we define $R(x, y)$ to be true if y is a solution for x in S. If x is not of this form, then we define $R(x, y)$ to be true for all y. I leave to the reader to verify all the conditions that R has to satisfy.

With this lemma in hand, the rest of the proof is easy. Suppose Conjecture $3'$ is true. Let T be a finitely axiomatized consistent theory. According to our assumption, there exists a **TPS** problem Q' which is not reducible to S_T. Let Q be a **TPS** problem such that both $Q' \leq Q$ and $S_T \leq Q$. Then Q is also not polynomially reducible to S_T, due to the transitivity of $\leq$. Again, by transitivity, if P is properly formalized in T, then $Q \not\leq P$. So Q has all the properties required by Conjecture 3.

The equivalence of Conjectures 4, $4'$ and $4''$. I will consider finitely axiomatized consistent theories. Furthermore, I will assume that they are sufficiently strong. Consistency and the latter condition imply that such theories prove only true $\Pi_{\mathbf{P}}$ sentences.

Recall that, given a theory T satisfying the conditions above, one can construct a propositional proof system P_T which simulates T for $\Pi_{\mathbf{P}}$ sentences. This means that for a $\Pi_{\mathbf{P}}$ sentence ϕ, one can define a sequence of tautologies τ_n, $n = 1, 2, \ldots$ expressing the truth of the finite instances of ϕ such that these sentences have polynomial size proofs in P_T.

Given a propositional proof system P, one can easily construct a theory T_P that proves the soundness of the system. This means that for a formula $pr(x, y)$ formalizing *'y is a P-proof of x'* and a formula $taut(x)$ formalizing *'x is a tautology'*, T proves the sentence $\forall x \forall y\ (pr(x, y) \to taut(x))$.

I will now outline the proofs of implications between these conjectures.

$(4') \Rightarrow (4)$. Suppose (4) is false and let S be the theory that witnesses this fact. I will show that the propositional proof system P_S is optimal.

Let P be an arbitrary propositional proof system. Let d be a P-proof of a tautology ϕ. According to the definition of P_S, it suffices to construct in polynomial time an S-proof of the sentence $taut(\bar{\phi})$.

I will now argue in T. *"Suppose ϕ were not a tautology; let a be a falsifying assignment of ϕ. Then T_P, the theory associated to P, would prove that ϕ is falsified by a (since T_P is sufficiently strong, it satisfies Σ-completeness). The length of this proof is polynomial in the length of ϕ, which is at most the length of d. Also by Σ-completeness, T_P proves that d is a P-proof of ϕ. This proof of polynomial length in the length of d. But T_P proves the soundness of P, thus we get that T_P proves a contradiction by a proof whose length is polynomial in the length of d."*

The above proof can actually be constructed in polynomial time. Since we also assume that we can construct in polynomial time S-proofs of $Con_{T_P}(\bar{n})$, we can construct an S-proof of $taut(\bar{\phi})$, hence a P_S-proof of ϕ. This algorithm runs in time that is polynomial in the length of d. Thus it is a polynomial simulation of the proof system P by the proof system P_S. This finishes the proof that P_S is optimal.

$(4'') \Rightarrow (4')$. Suppose $(4')$ is false. Let Q be an optimal propositional proof system. I will show that T_Q, the theory that contains an axiom that Q is sound, falsifies (7). The proof system Q is defined by a polynomial time relation $R(x, y)$ that defines that y is a proof of x. I am assuming that R is $\Phi_{\mathbf{P}}$-formalized in T_Q.

Let P be an arbitrary propositional proof system. Since Q polynomially simulates all propositional proof systems, there exists a polynomial time computable function f such that d is a P-proof of ϕ if and only if $R(\phi, f(d))$. Hence we can define the proof system P using the polynomial time computable relation $R(x, f(y))$. To obtain a $\Phi_{\mathbf{P}}$-formalization of P in T_Q, use the formalization of $R(x, y)$ and a $\Phi_{\mathbf{P}}$-formalization of the function f. Since T_Q proves the soundness for the formalization of $R(x, y)$, it also proves soundness for the formalization of $R(x, f(y))$.

$(4) \Rightarrow (4'')$. Suppose $(4'')$ is false. Let S be a theory that falsifies $(4'')$. I will show that it also falsifies (4).

Let T be a consistent finitely axiomatized theory. Let τ_n be the propositional translations of $Con_T(\bar{n})$. We can define these translations so that S proves that $Con_T(\bar{n})$ is equivalent to $taut(\tau_n)$ by proofs that can be constructed in polynomial time. Let P be an arbitrary propositional proof system (say, a Frege system). Extend P by postulating that τ_n is a proof of itself. Let P' be the extended propositional proof system. According to our assumption, S proves the soundness of P'. Hence to prove that τ_n is a tautology in S, we only need to verify that it has the right form. This only requires polynomial size proofs that can easily be constructed. Hence proofs of $Con_T(\bar{n})$ can also be constructed in polynomial time.

Main points:

- *Every proof in predicate logic can be transformed into a direct proof, but it may expand superexponentially.*

- *Theorems may have much shorter proofs if we*
    1. *either add a new axiom to the axiomatic system that we use,*
    2. *or instead of proving the theorem, we only prove that the theorem is provable.*

  The second way is an instance of what is called "considering from a higher perspective", but the first one is more effective.
- *A theory can be useful even if it is inconsistent, because it can happen that contradiction has a very long proof and all sentences provable by short proofs are true.*
- *Countable ordinals can be used to gauge the strength of theories.*
- *There are many analogies between computational complexity and proof complexity. In particular, there are theories that we view as counterparts of particular complexity classes, such as* $\mathbf{P}$ *and* $\mathbf{NP}$*.*
- *However, proof complexity, although closely related to computational complexity, is a different theory. In particular, the open problems of proof complexity are not mere reformulations of problems from computational complexity.*
- *The main problems in proof complexity are whether or not certain pairs of theories prove the same theorems. In some cases it is possible to derive separations of pairs of theories using conjectures about separations of pairs of complexity classes.*
- *The problem of separating theories can often be reduced to proving lower bounds on the lengths of proofs in the propositional calculus. Therefore, problems about the lengths of proofs in proof systems for the propositional calculus also belong among the central problems in proof complexity.*
- *According to the Feasible Incompleteness Thesis, the phenomenon of incompleteness should already manifest itself at the level of polynomial length proofs and polynomial time computations.*
- *Several conjectures that can be viewed as instances of the Feasible Incompleteness Thesis have been put forward. They have various implications in computational complexity; in particular* $\mathbf{P} \neq \mathbf{NP}$ *would be a consequence of some of these conjectures.*
- *This connection hints that the essence of the big open problems in complexity theory could be logical, rather than combinatorial.*