1.1 Meaning and Comprehension

Meaning and Information

Natural language is arguably the most important mean we have for exchanging information. It influences our everyday interactions with the external world and those with other people. It allows information to be conveyed between speakers and hearers about the world, their beliefs, intentions and so on. To make the communication possible many cognitive processes have to be engaged. First of all, a natural language expression carrying information has to be produced by a speaker. This generation process requires the ability of producing grammatical sequences meaning whatever a speaker intends to share with a hearer. Secondly, a hearer has to decode the information contained in an expression, i.e., a hearer has to understand the meaning of an expression (see e.g. Kamp and Stokhof, 2008, for elaboration on information and language).

Different Aspects of Meaning

The comprehension process can work in many different ways depending on the many ways in which natural language can be used. For example, consider the following sentences.

(1) Mt. Everest is the highest mountain in the world.

(2) K2 might be the most difficult mountain to climb.

If we consider the declarative use of language — a speaker stating facts, like in sentence (1), or raising possibilities as in (2) — then understanding may be described as the ability to imagine the world satisfying the sentence and adjust one’s beliefs accordingly.
However, there are many more possibilities. Let us consider the following sentences.

(3) Is K2 higher than Mt. Everest?

(4) Climb K2 in winter!

To understand question (3) one has to realize what are the possible answers. On the other hand, to comprehend directive (4) we have to recognize the speaker’s intentions, i.e., realize what he wants us to do.

Moreover, there are many indirect uses of language such that their comprehension depends very much on cooperation among speakers. For example consider the following:

(5) K2 has either been climbed in winter or not.

(6) I met Alice and Bob in the Everest base camp. She climbed it in one day and he failed.

(7) I am here right now.

When I am stating sentence (5) I do not only claim directly this obvious alternative. Rather I want to say that I actually do not know whether anyone has in fact done it. In some other situations meaning is crucially connected to a context. For instance, in understanding discourse and dealing with anaphora one has to recognize the references of pronouns to the entities mentioned previously in a conversation. Simply consider example (6). In other cases, a hearer has to know the context to recognize the denotation of indexical expressions, like in sentence (7).

Obviously it is impossible to exhaust all the functions of language in a short list like the one above, and this was not our aim. We only wanted to illustrate some varieties of linguistic meaning; next we will try to identify some common features of understanding.

**Basic Elements of Comprehension**

What are the building-blocks of natural language comprehension? We believe that there are at least two basic processes:

- Confronting meaning with the actual world.

- Confronting meaning with knowledge and beliefs.
The first process relates new information conveyed by the meaning of a natural language expression to the actual state of affairs. In the case of declarative sentences it allows us to decide whether the new information is true, or what the world should look like to make it true. In the case of questions we have to check the possible answers against the real world in order to be able to give a true answer. We even need to relate to the world to realize that some sentences bring indirect information. For example, to draw the intended implicatures from (5) I first have to realize that the truth-value of the statement is independent from the facts.

The second process, confronting meaning with knowledge, crucially determines our belief changes triggered by the new information carried by an utterance. When we hear a declarative sentence we check whether it is consistent with our knowledge. Sometimes we even make one step more, namely we check whether the implications of a sentence agree with our knowledge. Moreover, we search through our knowledge to answer a question. Also processing discourse can be seen in this perspective. For example, when we deal with a sequence of two sentences, like (6), we may add the first sentence to our knowledge and then simply look for referents for the pronouns among the most recently added beliefs.

1.2 Geography of Perspectives

1.2.1 Traditional Semantics and Pragmatics

Formal semantics describes functions which assign some set-theoretic object in the universe of discourse (possible worlds, models, possible interpretations of language) to every well-formed expression. We say that the object is an extension (denotation) of this expression. Such an approach — motivated by the work of Alfred Tarski (1944) and successfully applied to natural language by his student Richard Montague (1970) — is often called set-theoretic (or model-theoretic) semantics. In other words, formal semantics establishes a potential link between a well-formed natural language expression and an abstraction of the real world in the form of a model.

Some non-linguistic aspects of the reference relation, like the influence of the context or of knowledge, have been generating a lot of interest. These problematic cases were traditionally falling under the scope of pragmatics, however the distinction between semantics and pragmatics has never been clear. They were early investigated in the philosophy of language. Let us mention only the analysis given by John Langshaw Austin (1975) for performative utterances, like the following:

\[(8) \text{I name this mountain K2.}\]

Additionally, recall Paul Grice’s (1991) cooperative principle and conversational maxims explaining indirect communication, e.g., sentence (5).
Chapter 1. Algorithmic Semantics

In the model-theoretic semantics there were, from the beginning, serious attempts to incorporate context-dependence on the agenda, e.g., by studying indexicals expressions (see e.g. Bar-Hillel, 1954; Montague, 1970; Kamp, 1971). One of the first systematic accounts of context dependence within this tradition were provided by David Kaplan and Robert Stalnaker. Kaplan (1979) pointed out that circumstances intervene into the process of determining an utterance’s truth-value twice rather than only once as possible-worlds semantics appeared to suggest. He has argued that the utterance must be first confronted with its context to yield an intension, which only then can be confronted with a possible world to yield the truth-value of the utterance with respect to the actual world. Thus, to understand an utterance such as (7) we must first exploit the context to unfold the indexicals ‘T’, “here”, and “now”, to reach a proposition. Then this proposition can be evaluated in the possible worlds semantics.

Stalnaker has been one of the most influential theorists exploring the philosophical aspects of possible worlds semantics. According to his view of possible worlds, they stand for ways this world could have been, and are the maximal properties that this world could have had. Moreover, he has used the apparatus of possible worlds semantics to explore counterfactuals, conditionals, and presupposition. For example, central to his account of intentionality is the view of propositions as sets of possible worlds (see e.g. Stalnaker, 2003). Finally, his view of assertion as narrowing the conversational common ground to exclude situations in which the asserted content is false was a major impetus for the recent development of belief revision — a very rich theory which accounts for belief changes triggered by new information.

1.2.2 The Dynamic Turn in Semantics

The traditional model-theoretic approach has problems with combining the two basics aspects of meaning: relation to the external world and correspondence to knowledge. Simply put, model-theoretic semantics focuses only on the static relation between expressions and their denotations. This problem was recognized very early. For example, recall Carnap’s notion of a “meaning postulate”, trying to extend the inferential power of traditional semantic analysis by adding bits of lexical awareness (see Carnap, 2007). However, in the extensional tradition the problem has always been treated very partially and statically. Not much was done about changes in knowledge caused by new linguistic information before the so-called Dynamic Turn in the seventies. Around that time the real change in perspective can be observed. Since then a lot of work has been put into grasping dynamic aspects of knowledge change triggered by linguistic information (see e.g. Peregrin, 2003, for an overview). Below we briefly recall three important research axes.
1.2. Geography of Perspectives

Discourse Representation

Pioneering work in the dynamic framework was Discourse Representation Theory, introduced by Hans Kamp. This is a theoretical framework for dealing with issues in the semantics and pragmatics of anaphora, tense and presuppositions. Its distinctive features are that it is a mentalist and representationalist theory of interpretation, not only of individual sentences but of discourse as well. In these respects it made a clear break with classical formal semantics, but in other respects it continues the tradition, e.g., in its use of model-theoretic tools (see Kamp and Reyle, 1993).

Belief Revision

In the logic of belief revision, a belief state is represented by a set of sentences. The major operations of change are those consisting in the introduction or removal of a belief-representing sentence. In both cases, changes affecting other sentences may be needed, for instance in order to retain consistency. Rationality postulates for such operations have been proposed, and representation theorems have been obtained that characterize specific types of operations in terms of these postulates. There are many technical methods of representing these processes. The logical perspective is mostly formed around Dynamic Epistemic Logic and the algebraic perspective exploits mostly the AGM framework (see e.g. Gärdenfors, 2003; van Ditmarsch et al., 2007). Belief revision perspective can be treated as an in-depth study of the second fundamental of comprehension: ‘knowledge-checking’. However, its relationship with natural language semantics is still waiting for a development.

Games and Pragmatics

The work of Stalnaker and Kaplan put pragmatic considerations into the agenda of dynamic semantics. In parallel, David Lewis also worked on these ideas, for example formulating the analysis of counterfactual conditionals in terms of the theory of possible worlds. However, arguably his most important input into linguistics has given rise to game-theoretic considerations in that area. Lewis (2002) claimed that social conventions are solutions to game-theoretic ‘coordination problems’, but, what is more important from our perspective, he has pointed out that the use of a language in a population relies on conventions of truthfulness and trust among the members of the population. He has recast in this framework notions such as truth and analyticity, claiming that they are better understood as relations between sentences and a language, rather than as properties of sentences. This has led directly to the development of highly formal models of communication explaining Grice’s principles in terms of signaling games or evolutionary equilibria (see e.g. Benz et al., 2005). Recently language is seen as an interaction between speaker and hearer, as kind of a game in which by mutual interpretation
players come to an interpretation. This approach underlines Optimality Theory (see e.g. Kager, 1999) and it spans across semantics and pragmatics.

1.2.3 Meaning as Algorithm

Traditionally, in logic finite models were considered as a pathological case while infinite universes were in the center of attention. In parallel with the Dynamic Turn, researchers have started to be interested in finite interpretations. With this change in the perspective some new philosophical dimensions have arisen. Particularly, in natural language semantics the idea of treating meanings in terms of truth-conditions has naturally started to evolve towards algorithmic explanations. Below we briefly present this evolution and in Chapter 1.6 we take a closer look at finite interpretations in natural language semantics. This approach is particularly important for our thesis.

Sinn und Bedeutung

This is a tradition, going back to Gottlob Frege (1892) (see also Church, 1973, 1974; Dummett, 1978), of thinking about the meaning of a sentence as the mode of presenting its truth-value. In modern terms we can try to explicate the Fregean Art des Gegebenseins of a referent (the way the referent is given) by saying that the meaning of an expression is a procedure for finding its extension in a model. Accordingly, a sentence meaning is a procedure for finding the truth-value. Quoting Frege:

It is the striving for truth that drives us always to advance from the sense to the reference. (Frege, 1892, p. 63)

Similar ideas can be found around the same time also in the writings of other philosophers.
Let us quote Ludwig Wittgenstein:

To understand a proposition means to know what is the case if it is true. (Wittgenstein, 1922, 4.024)

Also Kazimierz Ajdukiewicz has claimed:

1 This way of interpreting Frege’s “Sinn” is not a standard view in the philosophy of language. Although, it could help to solve some notorious puzzles of the Fregean theory of meaning, e.g., those related to the meaning of indirect speech and discussed in the recent paper of Saul Kripke (2008).

2 An even more procedural account is proposed in “Philosophical Investigations” where Wittgenstein asks the reader to think of language and its uses as a multiplicity of language-games. The parts of language have meaning only within these games. (Wittgenstein, 1953, §23).
For two different persons an expression brings the same meaning, whenever it gives them the same method of deciding whether this expression can be applied to a given object or not.³

(Ajdukiewicz, 1931)

In model theory such procedures are often called model-checking algorithms. This approach has been adopted by many theoreticians, to different degrees of explicitness, and we can trace it back to Fregean ideas. For obvious reasons, Frege himself could not speak about procedures or algorithms directly.

The First Explicit Formulation

Historically — as far as we are aware — the Fregean idea was for the first time explicitly formulated in procedural terms by the Czech logician, philosopher and mathematician Pavel Tichý (1969). In the paper, which can be best summarized by its title “Intension in terms of Turing machines”, he identified the meaning of an expression with a Turing machine computing its denotation. The main technical objective of the paper is to account for the distinction between analytic and logical truths by treating concepts as procedures.⁴ However, the author also recognized the broader application of the algorithmic idea. He noticed that:

[... ] the fundamental relationship between sentence and procedure is obviously of a semantic nature; namely, the purpose of sentences is to record the outcome of various procedures. Thus e.g. the sentence “The liquid X is an acid” serves to record that the outcome of a definite chemical testing procedure applied to X is positive. The present paper is an attempt to make this simple and straightforward idea the basis for an exact semantic theory. (Tichý, 1969, p. 7)

Moreover, he directly argues for identifying meaning with an algorithm, a dynamic procedure of searching for the denotation instead of a static model-theoretic entity:

For what does it mean to understand, i.e. to know the sense of an expression? It does not mean actually to know its denotation but to know how the denotation can be found, how to pinpoint the denotation of the expression among all the objects of the same type. E.g. to know the sense of “taller” does not mean actually to know who is

³=”Dwaj ludzie rozumieją pewne wyrażenie w tym samym znaczeniu, gdy rozumienie to uzbraja ich obu w tę samą metodę rozstrzygania, czy wyrażenie to zastosować do jakiegoś przedmiotu, czy też nie.”

⁴=“The idea was for the first time presented at the Third International Congress for Logic, Methodology and Philosophy of Science in Amsterdam (1967) under the title “Analyticity in terms of Turing Machines”.”
taller than who, but rather to know what to do whenever you want to
decide whether a given individual is taller than another one. In other
words, it does not mean to know which of the binary relations on the
universe is the one conceived by the sense of “taller”, but to know a
method or procedure by means of which the relation can be identified.
Thus it seems natural to conceive of concepts as procedures.

(Tichý, 1969, p. 9)

Later Tichý developed a system of intensional logic with an extensive philo-
osophical justification of the Fregean idea (see Tichý, 1988).

Other Appearances of the Same Idea

Let us trace the idea of a meaning as an algorithm a little bit more as it has been
gradually formulated in more recent terminology. All approaches that we briefly
outline below try to account for the very same idea. However, none of them refers
to the work of Tichý.

Patrick Suppes (1982) has investigated an algebraic semantics for natural
language and finally came to the conclusion that the meaning of a sentence is
a procedure or a collection of procedures. His motivation seems to be mainly
psychological; let us quote the author:

The basic and fundamental psychological point is that, with rare ex-
ceptions, in applying a predicate to an object or judging that a relation
holds between two or more objects, we do not consider properties or
relations as sets. We do not even consider them as somehow sim-
ply intensional properties, but we have procedures that compute their
values for the object in question. Thus, if someone tells me that an
object in the distance is a cow, I have a perceptual and conceptual
procedure for making computations on the input data that reach my
peripheral sensory system [. . .] Fregean and other accounts scarcely
touch this psychological aspect of actually determining application of
a specific algorithmic procedure. (Suppes, 1982, p. 29)

He also has made a point that meaning can be treated not only in terms of
single procedures but as collections of those:

I have defended the thesis that the meaning of a sentence is a proce-
dure or a collection of procedures and that this meaning in its most
concrete representation is wholly private and idiosyncratic to each in-
dividual. (Suppes, 1982, p. 33)

Another approach — similar to Tichý’s way of thinking in its direct use of au-
tomata — was formulated by Johan van Benthem (1986) to describe the meanings
of quantifiers in natural language. Van Benthem’s semantic automata recognize
the truth-value of a generalized quantifier expression on a structure. We will
study this approach in more detail in Chapter 7, where we even show that it
is psychologically plausible. In particular, it correctly predicts some aspects of
the processing of natural language quantifiers, both on a cognitive and a neuro-
logical level. Here let us only note that the intuition behind introducing the
automata-theoretic perspective were not only technical:

An attractive, but never very central idea in modern semantics has
been to regard linguistic expressions as denoting certain “procedures”
performed within models for the language.

(van Benthem, 1986, p. 151)

The algorithmic theory of meaning found its strongest linguistic formulation
in the works of Yiannis Moschovakis (1990). He has analyzed the Fregean notions
of sense and denotation as algorithm and value, and then developed a rigorous
logical calculus of meaning and synonymy (Moschovakis, 2006). He also succeeded
in popularizing this idea. Reinhard Muskens (2005) has provided a similar theory
built on a considerably lighter formalization. All these works are mainly of a
linguistic character and try to present the Fregean distinction between meaning
and denotation in a strict mathematical framework, throwing some light on the
classical problems studied in the philosophy of language.

This line of research is also followed by Michiel van Lambalgen and Fritz
Hamm (2004). They have proposed to study the meaning-as-algorithm idea in
the paradigm of logic programming. The idea was taken further in the book “The
Proper Treatment of Events” (Lambalgen and Hamm, 2005). There the combi-
nation of the event calculus (as developed in Artificial Intelligence) with type free
theory and logic programming techniques is used to formulate an axiomatized
semantic theory for a broad range of linguistic applications. The authors argue
that the proposed architecture, which sees the heart of the semantics of tense
and aspect in the notion of planning, has cognitive plausibility. The argument
proceeds via an examination of the role of time in cognitive processes.

Theories Similar in Spirit

Other advances in natural language semantics might also be viewed as incorpo-
rating some procedural ideas. First of all, Montague (1970) committed himself
to the idea of intensional semantics, where the meaning of an expression can be
identified with a function choosing its denotation in every possible world. This
function can be interpreted as corresponding to some model-checking procedure.
Even if Montague’s approach cannot be directly labeled “procedural”, to some
extent he has motivated most of the works exploring algorithmic ideas, notably
that of Moschovakis and his followers. Moreover, the procedural spirit can be
found in the works creating the dynamic turn in linguistics, as all of them were strongly influenced by Montague. E.g., they adopted possible world semantics, compositionality and other tools developed by Montague.

Two other examples of procedural theories are dynamic semantics and game-theoretic semantics. Dynamic semantics (see Groenendijk and Stokhof, 1991; van den Berg, 1996) formalizes meaning in terms of transformations between states. The idea is very simple; quoting Paul Dekker’s (2008) guide to dynamic semantics:

People use language, they have cognitive states, and what language does is change these states. ’Natural languages are programming languages for mind’, it has been said. (Dekker, 2008, p. 1)

Game-theoretic semantics (see Lorenzen, 1955; Hintikka and Sandu, 1997) sees the meaning of a sentence as a winning strategy in a game leading to its verification. The game-theoretic semantics have initiated many advances in logic and linguistics. The game-theoretic metaphor was extended even further by Merlijn Sevenster (2006) who has proposed a strategic framework for evaluating some fragments of language. However, it is not completely clear how these relate to more traditional model-checking approaches.

Considering language in a strategic paradigm as a goal-directed process rather than a recursive, rule-governed system (see Hintikka, 1997) may help to understand aspects of meaning different than model-checking, like inferential properties, confronting new information with knowledge and so on. For example, so-called model construction games which build models for a set of formulas (see van Benthem, 2003; Hodges, 1997) can help to understand how language users integrate new information into a consistent information state. And, at least from a logical point of view, model-building games are related to Lorenzen dialogical games for proofs.

Synonymy

One of the most famous philosophical problems with meaning is concerned with the synonymy of linguistic expressions. It is widely claimed that we do not understand the meaning of “meaning” as long as we cannot define synonymy (see Quine, 1964). According to the algorithmic proposal the problem is equivalent to the question about the identity relation between algorithms. In other words, having a set $A$ of algorithms we search for an equivalence relation $\approx$ on $A$ such that for every $f, g \in A$:

$$f \approx g \iff f \text{ and } g \text{ realize the same algorithm.}$$

We will see some consequences of this reformulation of the synonymy problem when discussing computability issues later in this chapter.
1.3 Our Procedural Perspective on Meaning

1.3.1 Questions

In this thesis we are mainly concerned with the first-mentioned logical element of natural language comprehension. Namely, we study how meaning relates to the external world. However, as the model-theoretic perspective cannot be torn apart from knowledge representation we also sometimes (possibly between the lines) refer to the problem of the relation between meaning and knowledge, e.g., we will speak later about referential and inferential meaning. To understand the processes lying behind these aspects of comprehension we need to consider, among others, the following questions:

- What is the meaning of a given expression?
- How do people recognize the denotations of linguistic expressions?
- Why do some sentences seem to be more difficult to understand than others?

There are many connections between these questions and this is why in trying to answer one of them we have to take a position about the rest.

1.3.2 Methodology

Above we have discussed a certain computational approach to these problems which we adopt throughout the whole thesis. Therefore, we align with the dynamic way of thinking by treating comprehension not from the static extensional perspective but as a dynamic procedure.

The algorithmic model-checking approach to meaning seems reasonable for a big fragment of natural language in many different contexts. In the dissertation we follow this philosophical line and identify the meaning of an expression with an algorithm that recognizes its extension in a given finite model. We sometimes refer to the meaning understood in such a way as referential meaning to stress that this is just one of the possible aspects of meaning — as discussed earlier in this chapter. Even though we see this line of thinking as another manifestation of the dynamic turn we have to restrict ourselves to studying a basic element of comprehension, model-checking, and not pursue its more context-dependent aspects.

In other words, we say that the referential meaning of a sentence $\varphi$ is given by a method of establishing the truth-value of $\varphi$ in possible situations. Such procedures can be described by investigating how language users look for the truth-value of a sentence in various situations.\(^5\)

\(^5\)See Chapter 6 for an example of research which tries to describe referential meaning for some class of natural language sentences.
In the thesis we do not study the algorithmic theory of meaning in itself. However, our technical work is motivated mostly by these ideas. Particularly, we are in debt to the formulations of procedural semantics which appear in the research of Tichý (1969) and van Benthem (1986) as we will also work mainly with machines recognizing quantifiers. Moreover, we follow Suppes (1982) and claim that for one semantic construction there are many essentially different algorithms. Their usefulness depends on the situation. To understand this idea better let us consider the following sentence:

(9) The majority of people at the party were women.

The referential meaning of sentence (9) can be expressed by a simple counting procedure. However, in the case when there are over five hundred people at the party the counting procedure is not very efficient. But it may happen that guests perform some traditional dance in pairs. Then we could apply a different algorithm, for instance simply check if some woman remains without a man. In this case the second method would be much more efficient than the first one. Actually, in Chapter 7 we present empirical research indicating that the same linguistic construction can be understood by people using different procedures which are triggered by various contexts.

1.3.1. Example. Let us give one more related example capturing some of the intuitions. Imagine two people: John, who is a mathematician, and Tom, who is a geologist. They both speak English, using such expressions as: “being higher than” and “prime number”. Both of them understand these expressions. However, only John knows the simple ancient algorithm, the Sieve of Eratosthenes, for finding all prime numbers up to a specified integer. On the other hand, only Tom understands the principles of using engineering instruments for measuring levels. Therefore, there are expressions such that their truth-value in certain situations can be decided only by John or only by Tom. For example, they can both easily decide whether the following sentences are true:

(10) 17 is a prime number.

(11) John is taller than Tom.

However, if we pick a big number or two objects whose lengths cannot be compared directly or via simple measurements, then it is very likely that only one of them can decide the truth-value of the sentence. For instance, John but not Tom can decide the truth-value of the following sentence:

We talk about “constructions” and not “expressions” to avoid misunderstandings as the same expression can be used for several different semantic constructions. For example, the expression “and” can serve as a propositional connective or as an operation between noun phrases.
1.3. Our Procedural Perspective on Meaning

(12) 123456789 is a prime number.

Analogously, only Tom knows how to decide the truth-value of the following statement:

(13) Gasherbrum I is 50 meters higher than Gasherbrum II.

The point of this example is to stress that when dealing with a natural language sentence we very often want to know whether it is true or false and we may need to use different meanings of it in various situations to find out. This linguistic ability is based on the fact that we are continuously using various techniques for recognizing the extensions of natural language constructions. These techniques can be identified with meanings. Moreover, learning natural language constructions consists essentially of collecting procedures for finding denotations (see Gierasimczuk, 2007). This way of thinking is in line with the algorithmic view of meaning.\footnote{Notice that this way of thinking can also contribute to the philosophical idea of the division of linguistic labor (see Putnam, 1985). Simply, experts know more relevant meaning-algorithms and hence understand the meaning of an expression belonging to the domain of their expertise in a more sophisticated way.}

Summing up, what follows is the main methodological assumption of the thesis:

\textbf{Main Assumption} The referential meaning of an expression $\chi$ is a collection of algorithms computing the extension of $\chi$ in a given finite model.

However, having model-checking algorithms for all sentences is not the only possible way to understand language. As we discussed, the other basic component of language comprehension is the ability to relate it to knowledge. For example, we can also recognize some inferential relations between sentences establishing so-called \textit{inferential meaning}. For instance, knowing that a sentence $\psi$ is true and $\varphi$ follows from $\psi$ we know that $\varphi$ is true. Already Jean Piaget (2001) has noticed that this is a mechanism used very often to evaluate sentences. We also study such situations in the thesis. One of the interesting problems — which we leave open — is how inferential meaning relates to belief revision.

1.3.3 Psychological Motivations

Our main aim in applying a dynamic, procedural framework is motivated by our interest in psycholinguistics. We share this motivation with Suppes (1982) and Lambalgen and Hann (2005). We believe that a good linguistic theory should give direct input for empirical studies on language processing. Unfortunately, most of the dynamic frameworks do not make a single step in this direction.
We study the computational properties of natural language to shed some light on the cognitive processes behind language comprehension. However, as at this stage of research we are far from formulating a satisfactory, detailed model of comprehension, we rather focus on abstract computational properties which do not depend on any specific implementation. Therefore, we are not studying concrete procedures formalizing meaning but properties common to all possible procedures.

In the rest of this chapter we give an intuitive description of computations (mathematical details are presented in Section 2.3). Then we discuss the links between computability theory and cognitive science as we treat comprehension as a cognitive task. Our aim is to put computational semantics in a broader context of cognitive science.

1.4 Algorithms and Computations

1.4.1 What is an “Algorithm”?

We have decided to view some semantic issues from the computational perspective. In this section we will then try to shed some light on what these procedures or algorithms we are talking about really are. Actually, the issue is highly non-trivial. Even though the intuitions behind the notion of “algorithm” are probably more precise than those standing behind “meaning” they are still far from being completely clear. On the other hand, this is not a serious obstacle because we are interested rather in the inherent properties of computational problems than in any concrete algorithmic implementations.

Euclid’s method for finding the greatest common divisor for any two positive integers (developed around 4 BC) is commonly believed to be the first non-trivial algorithm. As far as etymology is concerned the word “algorithm” is connected to the name of the Persian astronomer and mathematician Al-Khwarizmi. He wrote a treatise in Arabic in 825 AD “On Calculation with Hindu Numerals”. It was translated into Latin in the 12th century as “Algoritmi de numero Indorum”, which title was likely intended to mean “Algoritmi on the numbers of the Indians”, where “Algoritmi” was the translator’s rendition of the author’s name. However, some people misunderstood the title, and treated Algoritmi as a Latin plural. This led to the word “algorithm” coming to mean “calculation method”. Coincidentally, in his treatise he introduced the decimal positional number system to the Western world and many simple algorithms for dealing with addition, subtraction, multiplication and division of decimal numbers. We all learn these algorithms in the beginning of our education.

There is no generally accepted formal definition of “algorithm” yet. An informal definition could be “an algorithm is a mechanical procedure that calculates something”. It is an interesting problem in the philosophical foundations of com-
1.4. Algorithms and Computations

computer science to come up with a reasonable and widely acceptable formal definition. Actually, there is some research going in this direction and the proposed methods for dealing with this problem are very similar to those used for solving foundational issues of mathematics in the beginnings of the 20th century. For example, people have suggested formalizing algorithms in set theory, rejecting their existence (there are no algorithms just programs) or axiomatizing their theories (see e.g. Moschovakis, 2001, for a discussion).

1.4.2 Turing Computability

Following tradition, we have decided to view computability in terms of Turing machines (see Section 2.3.2). This decision may be justified by the following widely believed philosophical claim:

**Church-Turing Thesis** *A problem has an algorithmic solution if and only if it can be computed by a Turing machine.*

The Church-Turing Thesis (Turing, 1936; Church, 1936) states that everything that ever might be mechanically calculated can be computed by a Turing machine.

Let us briefly try to justify the Church-Turing Thesis here.\(^8\) Obviously, if a problem is computable by some Turing machine then we can easily calculate it by following the steps of this machine. But why should we believe that there are no computable problems beyond Turing machine computability? First of all, we do not know of any counterexample. Every function which we know how to compute can be computed by a Turing machine. Moreover, all the known methods of constructing computable functions from computable functions lead from Turing computable functions to other Turing computable functions. Additionally, the class of all Turing computable functions is very natural in the following sense. All attempts so far to explicate computability (many of them entirely independent) have turned out to define exactly the same class of problems. For instance, definitions via abstract machines (Random Access Machines, quantum computers, cellular automata and genetic algorithms), formal systems (the lambda calculus, Post rewriting systems) and particular classes of function (recursive functions) are all equivalent to the definition of Turing machine computability.

Moreover, the Church-Turing Thesis is interesting because it allows us to conclude that a problem is not decidable from the proof that it is not Turing computable. Take the class of all functions from natural numbers to natural numbers. There are uncountably many such functions, but there are only countably many Turing machines (this follows easily from the definition, see Section 2.3.2). Hence, some natural number functions must not be computable. The

\(^8\)In a recent paper Mostowski (2008) gives a proof of the thesis based on the assumption that a finite but potentially infinite world is a good mathematical model of our reality.
most famous non-computable problems are the Halting Problem (decide whether Turing machine $M$ will halt on input $x$), the decision problem for first-order logic (i.e., the question whether a given formula $\varphi$ is a theorem), and the Tenth Hilbert Problem (the algorithmic solvability in integers of Diophantine equations).

Identity of meanings-as-algorithms

Let us relate the above arguments directly to the semantic considerations. We have noticed, discussing procedural approaches to meaning, that the synonymy problem can be stated in terms of an identity criterion for algorithms.

The minimal demand on the identity relation is to identify notational variants of the same algorithm. The widest reasonable definition will identify algorithms computing the same partial functions, i.e., terminating on the same inputs and giving the same output on identical input.

Let us observe that this criterion of identity is not computable as otherwise we could easily compute the Halting Problem. In this case, we can not find an algorithm deciding whether two expressions have the same meaning or not.\footnote{Notice that this can cause some serious real life problems, e.g., for copyright legislation in the domain of software.} It is possible only in some simple cases, for instance, when meanings can be identified with finite automata as in the case of Van Benthem’s approach studied in Chapter 7. This fact nicely matches the status of the long-standing and unsolved synonymy problem in the philosophy of language.\footnote{However, see Moschovakis (1990) for a sophisticated algorithmic theory of meaning with a computable synonymy problem.}

Moreover, notice that two in the above sense identical algorithms do not have to be equally good in every sense. For instance, if we take a sentence containing $n$ words it is possible that one algorithm will need $n^2$ steps to compute its referential meaning, while the other needs $2^{2^n}$ steps. Thus, the identity definition should probably be enriched by a condition saying that in order for two algorithms to be identical they not only have to compute the same partial function but also must be comparable with respect to their efficiency. But how can we say that two different algorithms are similarly effective?

Computational Complexity

Inherent Complexity  With the development of programming practice it has been observed that there are computable problems for which we do not know any effective algorithms. Some problems need too much of our computational resources, like time or memory, to get an answer. Computational complexity theory, described briefly in Section 2.3, investigates the amount of resources required for the execution of algorithms and the inherent difficulty of computational problems. This means that the theory does not deal directly with concrete algorithmic
1.4. Algorithms and Computations

procedures, but instead studies the abstract computational properties of queries. These properties determine in a precise mathematical sense some properties of all possible algorithms which can be used to solve problems. As a result, the theory explains why for some computable questions we cannot come up with useful algorithms.

**Tractability and Intractability** An important aspect of computational complexity theory is to categorize computational problems via complexity classes. In particular, we want to identify efficiently solvable problems and draw a line between tractability and intractability. From our perspective the most important distinction is that between problems which can be computed in polynomial time with respect to the size of the problem, i.e., relatively quickly, and those which are believed to have only exponential time algorithmic solutions. The class of problems of the first type is called PTIME (P for short). Problems belonging to the second are referred to as NP-hard problems (see Section 2.3.3 for mathematical details). Intuitively, a problem is NP-hard if there is no “clever” algorithm for solving it. The only way to deal with it is by using brute-force methods: searching throughout all possible combinations of elements over a universe. In other words, NP-hard problems lead to combinatorial explosion.

Notice that all complexity claims reaching out to empirical reality make sense only under the assumption that the complexity classes defined in the theory are essentially different. These inequalities are sometimes extremely difficult to prove. We will discuss these, mainly technical, issues in Section 2.3.3, where we give formal definitions. Now, let us only mention the most famous complexity problem. As we said above PTIME is the class of problems which can be computed by deterministic Turing machines in polynomial time. Moreover, speaking precisely, NP-hard problems are problems being at least as difficult as problems belonging to the NPTIME (NP) class. This is the class of problems which can be computed by nondeterministic Turing machines in polynomial time. NP-complete problems are NP-hard problems belonging to NPTIME, hence they are intuitively the most difficult problems among the NPTIME problems. In particular, it is known that P=NP if any NPTIME-complete problem is PTIME computable. Unfortunately, we do not know whether P=NP. It is the famous question worth at least the prize of $1,000,000 offered by the Clay Institute of Mathematics for solving one of the seven greatest open mathematical problems of our time. However, the experience and practice of computational complexity theory allows us to safely assume that these two classes are different. This is what almost all computer scientists believe and we also take it for granted.

**Complexity Assumption** $P \neq NP$. 
Before we move to more general considerations let us give some examples. Many natural problems are computable in polynomial time, for instance calculating the greatest common divisor of two numbers or looking something up in a dictionary. However, we will focus here on a very important NP-complete problem, the satisfiability problem for propositional formulae.

The problem is to decide whether a given propositional formula is not a contradiction. Let \( \varphi \) be a propositional formula with \( p_1, \ldots, p_n \) distinct variables. Let us use the well-known algorithm based on truth-tables to decide whether \( \varphi \) has a satisfying valuation. How big is the truth-table for \( \varphi \)? The formula has \( n \) distinct variables occurring in it and therefore the truth-table has \( 2^n \) rows. If \( n = 10 \) there are 1,024 rows, for \( n = 20 \) there are already 1,048,576 rows and so on. In the worst case, to decide whether \( \varphi \) is satisfiable we have to check all rows. Hence, in such a case, the time needed to find a solution is exponential with respect to the number of different propositional letter of the formula. A seminal result of computational complexity theory states that this is not a property of the truth-table method but of the inherent complexity of the satisfiability problem. We have the following:

1.4.1. **Theorem (Cook 1971).** sat is NP-complete.

**What is Tractable?** To answer this question the following thesis was formulated, for the first time by Jack Edmonds (1965):

**Edmonds’ Thesis** The class of practically computable problems is identical to PTIME class, that is the class of problems which can be computed by a deterministic Turing machine in a number of steps bounded by a polynomial function of the length of a query.

The thesis is accepted by most computer scientists. For example, Garey and Johnson (1979) claim:

Most exponential time algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only through the gain of some deeper insight into the nature of the problem. There is wide agreement that a problem has not been “well-solved” until a polynomial time algorithm is known for it. Hence, we shall refer to a problem as intractable, if it is so hard that no polynomial time algorithm can possibly solve it.

(Garey and Johnson, 1979, p. 8)

This also justifies the following definition of identity between algorithms (synonymy), where we do not distinguish between algorithms which differ in complexity up to some polynomial. According to it, we say that algorithms \( f \) and \( g \)

---

**sat-problem** Before we move to more general considerations let us give some examples. Many natural problems are computable in polynomial time, for instance calculating the greatest common divisor of two numbers or looking something up in a dictionary. However, we will focus here on a very important NP-complete problem, the satisfiability problem for propositional formulae.

The problem is to decide whether a given propositional formula is not a contradiction. Let \( \varphi \) be a propositional formula with \( p_1, \ldots, p_n \) distinct variables. Let us use the well-known algorithm based on truth-tables to decide whether \( \varphi \) has a satisfying valuation. How big is the truth-table for \( \varphi \)? The formula has \( n \) distinct variables occurring in it and therefore the truth-table has \( 2^n \) rows. If \( n = 10 \) there are 1,024 rows, for \( n = 20 \) there are already 1,048,576 rows and so on. In the worst case, to decide whether \( \varphi \) is satisfiable we have to check all rows. Hence, in such a case, the time needed to find a solution is exponential with respect to the number of different propositional letter of the formula. A seminal result of computational complexity theory states that this is not a property of the truth-table method but of the inherent complexity of the satisfiability problem. We have the following:

1.4.1. **Theorem (Cook 1971).** sat is NP-complete.

**What is Tractable?** To answer this question the following thesis was formulated, for the first time by Jack Edmonds (1965):

**Edmonds’ Thesis** The class of practically computable problems is identical to PTIME class, that is the class of problems which can be computed by a deterministic Turing machine in a number of steps bounded by a polynomial function of the length of a query.

The thesis is accepted by most computer scientists. For example, Garey and Johnson (1979) claim:

Most exponential time algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only through the gain of some deeper insight into the nature of the problem. There is wide agreement that a problem has not been “well-solved” until a polynomial time algorithm is known for it. Hence, we shall refer to a problem as intractable, if it is so hard that no polynomial time algorithm can possibly solve it.

(Garey and Johnson, 1979, p. 8)

This also justifies the following definition of identity between algorithms (synonymy), where we do not distinguish between algorithms which differ in complexity up to some polynomial. According to it, we say that algorithms \( f \) and \( g \)
are identical if and only if they compute the same partial functions and moreover their working time on the same inputs differ at most by a polynomial function of that input. Notice that with such a definition we get the whole hierarchy of algorithms for a given expression generated by possible model-checking algorithms. This will be important for our considerations in Section 1.8.

**Invariance** As we said before, computational complexity theory is interested in the inherent complexity of problems independent of particular algorithmic solutions and their implementations. The most common model of computation used in the theory is the Turing machine. However, to justify computational complexity distinctions, e.g., between tractable and intractable problems, one has to give some argument that those distinctions are in fact independent of the particular implementation. The situation here is very similar to assuming the Church-Turing Thesis and the analogous arguments suggest another commonly believed assumption, the so-called Invariance Thesis (see e.g. Garey and Johnson, 1979):

**Invariance Thesis** *Given a “reasonable encoding” of the input and two “reasonable machines”, the complexity of computation of these machines on that input will differ by at most a polynomial amount.*

By “reasonable machine” any type of deterministic Turing machine or any other realistic computing machine (including neural networks, often suggested as a plausible model of brain computational architecture) are meant. Notice, however, that non-deterministic Turing machines (as well as quantum computers) are not realistic in this sense (see the argument for Theorem 2.3.13).

Assuming the Invariance Thesis we get that a task is difficult if it corresponds to a function of a high computational complexity, independent of the computational devices we are working with, at least as long as they are reasonable.

**Are PTIME Algorithms Always Tractable?** The common belief in Edmonds’ Thesis stems from the practice of programmers. NP-hard problems often lead to algorithms which are not practically implementable even for inputs of not very large size. Assuming the Church-Turing Thesis, $P \neq NP$, and the Invariance Thesis one comes to the conclusion that this has to be due to some internal properties of these problems and not to the restrictions of the current computing technology. However, even with these assumptions there are still some doubts. They lead to a better understanding of the nature of computational complexity claims and that is why we briefly discuss some of them below.

First of all, in some cases the polynomial algorithm is essentially better just for very large data. For example, not so long ago the best deterministic procedure checking whether a given number $n$ is prime was bounded by $n^{0.5 \log \log n}$. This
algorithm was practically used even though it is not polynomial. The reason is very simple: $n^{\log \log n} > n^2$ for only $n > e^{e^{10}}$, where $e \approx 2.718281$ (Blass and Gurevich, 2003). In other words, the polynomial algorithm is essentially better only for very big input.

Secondly, a polynomial algorithm might be also practically intractable. $n^{98466506514687}$ is a polynomial but even for small $n$ an algorithm of that working time would not be practical. However, many theorists believe that if we come up with polynomial algorithms of a high degree then it is only a matter of time before they will be simplified. In practice, programmers implement mainly algorithms with time complexity not greater than $n^3$. On the other hand, exponential procedures are sometimes used if they are supposed to work on small input data. Related additional difficulty comes from the fact that the computational complexity measures do not take constants into considerations (see Definition 2.3.14) as when $n$ increases their influence on the complexity decreases. Then, an algorithm working in time $n^3 + 2^{630}$ is still taken as a reasonable polynomial bound of degree 3.

Finally, let us consider an even more drastic example (see Gurevich, 1995). Let $g$ be an uncomputable function and define $f$ as follows:

$$f(x) = \begin{cases} 1 & \text{if length of } x < 2^{9999999999999999999999999999999} \\ g(x) & \text{otherwise.} \end{cases}$$

Is $f$ computable? For all inputs of reasonable size $f$ is computable and has value 1. It is uncomputable only for extremely big inputs. Therefore, can we intuitively claim that $f$ is computable?

We will discuss again the pros and coins of setting the tractability border along PTIME computability limits in the following chapter. However, now we turn to the cognitive perspective which is more relevant for understanding the background of our technical work.

1.5 Computability and Cognition

Accepting Edmonds’ Thesis we have to agree that problems beyond PTIME are computationally intractable. Our practical experience suggests that this is a reasonable assumption even though we raised some theoretic doubts in the previous section. The question appears how this belief influences cognitive science and, particularly, psycholinguistics. Below we briefly review this issue.

\[\text{In 2002, Indian scientists at IIT Kanpur discovered a new deterministic algorithm known as the AKS algorithm. The amount of time that this algorithm takes to check whether a number } n \text{ is prime depends on a polynomial function of the logarithm of } n \text{ (Agrawal et al., 2004).}\]
1.5.1 Computational Explanation of Cognitive Tasks

Cognitive Tasks  What is a cognitive task? Taking a very abstract perspective we can say that a cognitive task is an information-processing or computational task. Namely, the aim of a cognitive task is to transform the initial given state of the world into some desired final state. Therefore, cognitive tasks can be identified with functions from possible initial states of the world into possible final states of the world. Notice that this understanding of cognitive tasks is very closely related to psychological practice (see e.g. Sternberg, 2008). First of all, experimental psychology is naturally task oriented, because subjects are typically studied in the context of specific experimental tasks. Furthermore, the dominant approach in cognitive psychology is to view human cognition as a form of information processing (see e.g. van Rooij, 2004).

Marr’s Levels  One of the primary objectives of behavioral psychology is to explain human cognitive tasks understood in the very abstract way outlined above. David Marr (1983) was the first to propose a commonly accepted general framework for analyzing levels of explanation in cognitive sciences. In order to focus on the understanding of specific problems, he identified three levels (ordered according to decreasing abstraction):

- computational level (problems that a cognitive ability has to overcome);
- algorithmic level (the algorithms that may be used to achieve a solution);
- implementation level (how this is actually done in neural activity).

Marr argues that the best way to achieve progress in cognitive science is by studying descriptions at the computational level in psychological theories. He claims:

An algorithm is likely to be understood more readily by understanding the nature of the problem being solved than by examining the mechanism (and the hardware) in which it is embodied.

(Marr, 1983, p. 27)

Marr’s ideas in the context of computational complexity were nicely summarized by Frixione (2001):

The aim of a computational theory is to single out a function that models the cognitive phenomenon to be studied. Within the framework of a computational approach, such a function must be effectively computable. However, at the level of the computational theory, no assumption is made about the nature of the algorithms and their implementation.

(Frixione, 2001, p. 381)
1.5.2 Computational Bounds on Cognition

How can we apply all the above considerations to cognitive science? First of all, we want to talk about some very general properties of cognitive tasks which can be studied from the perspective of Marr’s computational levels. As we have noticed before, one of the abstract computational properties of functions, independent from any particular implementation, is their complexity. Viewing cognitive tasks as functions we can therefore pose questions about their computational complexity. Notice that as we do not know the details of our cognitive hardware or the precise algorithms implemented in the brain, the inherent perspective of computational complexity theory is very well-suited for the purposes of cognitive science.

The most common computational claim about cognition is a so-called psychological version of the Church-Turing thesis.

Psychological Version of the Church-Turing Thesis *The human mind can only deal with computable problems.*

In other words, cognitive tasks consist of computable functions.

This claim is commonly believed. However, despite its wide acceptance, the psychological version of the Church-Turing Thesis has its critics. The first opposition comes from researchers who believe that cognitive systems can do more than Turing machines. In fact, we agree that there are some uncomputable problems in the research scope of cognitive science as well. For example, the general framework of learning (identifiability in the limit) is not computable (see Gold, 1967). Presumably, there are more cognitive tasks lying beyond Turing computability (see Kugel, 1986, for an extensive discussion). Researchers from that camp often argue for the possibility of hyper-computations, i.e., the realizability of super-Turing machines, like Zeno-machines (Accelerated Turing machines) that allow a countably infinite number of algorithmic steps to be performed in finite time (see e.g. Syropoulos, 2008).

Some of them even misuse Gödel’s theorems to claim that the human mind cannot have an algorithmic nature. J.R. Lucas (1961) has claimed:

> Goedel’s theorem seems to me to prove that Mechanism is false, that is, that minds cannot be explained as machines. So also has it seemed to many other people: almost every mathematical logician I have put the matter to has confessed to similar thoughts, but has felt reluctant to commit himself definitely until he could see the whole argument set out, with all objections fully stated and properly met. This I attempt to do. (Lucas, 1961, p. 112)
Then he gives the following argument. A computer behaves according to a program, hence we can view it as a formal system. Applying Gödel's theorem to this system we get a true sentence which is unprovable in the system. Thus, the machine does not know that the sentence is true while we can see that it is true. Hence, we cannot be a machine.

Lucas' argument was revived by Roger Penrose (1996) who additionally supported it by claiming that the human mind can solve uncomputable problems thanks to a specific quantum feature of the brain's neural system. Let us only mention that Lucas' argument has been strongly criticized by logicians and philosophers (see e.g. Benacerraf, 1967; Pudlak, 1999).

Another stream of opposition is related to practical computability. It is believed that cognitive systems, being physical systems, perform their tasks under computational resource constraints. Therefore, the functions computed by cognitive systems need to be computable in realistic time and with the use of a realistic amount of memory. We agree with this objection, and will consider it in more detail in the next section.

**Tractable Cognition**

What is tractable cognition? We do not know what kind of a device the human cognitive system is. Hence, it would be particularly useful to define tractable cognition in terms of computational complexity constraints on cognitive functions, taking into account the Invariance Thesis.

As far as we are aware the version of Edmonds' Thesis for cognitive science was for the first time formulated explicitly in print\(^\text{12}\) by Frixione (2001) (see e.g. van Rooij, 2008, for a discussion):

**P-cognition Thesis** *Human cognitive (linguistic) capacities are constrained by polynomial time computability.*

In other words, the P-cognition Thesis states that a cognitive task is (hard) easy if it corresponds to a(n) (in)tractable problem. In our psycholinguistic setting this means that the intractable natural language constructions are those for which the model-checking computational complexity goes beyond polynomial time computability.

Let us give an argument in favor of the P-cognition Thesis. We know that a brain contains roughly \(10^{15}\) synapses operating at about 10 impulses per second, giving more or less \(10^{16}\) synapse operations per second (see e.g. Kandel et al., 2000). Obviously, not every cognitive capacity can claim all of the processing power of the entire brain as there are many parallel cognitive tasks going on\(^\text{12}\)Independently, exactly the same idea was formulated in the talk of Marcin Mostowski at Alfred Tarski Centenary Conference in Warsaw on 1 June 2001 and appeared in print 3 years later (see Mostowski and Wojtyniak, 2004).
every moment. Now, assume that we are dealing with two cognitive problems: the polynomial problem $\Phi$ and the exponential problem $\Psi$. For illustrative purposes assume that for $\Phi$ the brain needs to make $n^2$ steps if the instance of the problem is of size $n$, while the problem $\Psi$ requires $2^n$ steps. Table 1.1 shows how much time a brain would need to compute these problems for different sizes of input (if it can work with the speed of 10,000 steps per second on each of them).

<table>
<thead>
<tr>
<th>$n$</th>
<th>Time for $\Phi$</th>
<th>Time for $\Psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.01 sec</td>
<td>0.10 sec</td>
</tr>
<tr>
<td>20</td>
<td>0.04 sec</td>
<td>1.75 min</td>
</tr>
<tr>
<td>30</td>
<td>0.09 sec</td>
<td>1.2 days</td>
</tr>
<tr>
<td>50</td>
<td>0.2 sec</td>
<td>8.4 centuries</td>
</tr>
<tr>
<td>100</td>
<td>1 sec</td>
<td>$9 \times 10^{17}$ years</td>
</tr>
</tbody>
</table>

Table 1.1: The table compares the running time of a brain working with power 10,000 steps per second to solve two cognitive tasks $\Phi$ and $\Psi$. $\Phi$ is computable in $n^2$ and $\Psi$ requires $2^n$ steps.

It is clearly visible from the table that there is an essential difference between polynomial and exponential problems. Even for not very big inputs exponential problems can lie beyond brain power. The P-cognition Thesis emphasizes this difference in the context of cognitive science. In the dissertation we accept the P-cognition Thesis.

**Psychological Practice**

Complexity claims are used in cognitive science, among other functions, to evaluate the feasibility of computational theories of cognition. For example, Levesque (1988) has recognized the computational complexity of general logic problems (like SAT) and has concluded that we need to adjust logic in order to obtain psychologically realistic models of human reasoning. Similarly, Tsotsos (1990) has noticed that visual search in its general (bottom-up) form is NP-complete. As a result he has suggested adjusting the visual model by assuming that top-down information helps to constrain the visual search space. Moreover, in the study of categorization and subset choice computational complexity serves as a good evaluation of psychological models (see e.g. van Rooij et al., 2005). Additionally, complexity restrictions on cognitive tasks have been noted in philosophy of language and of mind. For instance, Cherniak (1981), Chalmers (1994) and Hofstadter (2007) have argued that a philosophical theory of mind should take computational restrictions seriously. Iris van Rooij (2004) in her dissertation gives many examples of the influence computational restrictions have on cognitive science.
1.5.3 Common Doubts

Our experience shows that many researchers are skeptical about applicability of computational complexity in cognitive science. Below we answer some common doubts.

Limit Behavior

Computational complexity is defined in terms of limit behavior (see Section 2.3.3). In other words, a typical question of computational complexity theory is of the form:

As the size of the input increases, how do the running time and memory requirements of the algorithm change?

Therefore, computational complexity theory, among other things, investigates the scalability of computational problems and algorithms, i.e., it measures the rate of increase in computational resources required as a problem grows. The implicit assumption here is that the size of the problem is unbounded. For example, models can be of any finite size, formulae can contain any number of distinct variables and so on.

It is often claimed that as complexity theory deals only with relative computational difficulty of problems it has not much to offer for cognitive considerations. Simply put, the inputs we are dealing with in our everyday life are not of arbitrary size. In typical situations they are even relatively small. In fact, critics claim, computational complexity does not say how difficult it is to solve a given problem for a fixed size of the input.

In general, even though computational complexity is formally defined in terms of limit behavior it can still be reasonably interpreted as saying something about problem difficulty on a fixed model. Namely, if the computational complexity of the problem is high, then it means that there are no “clever” algorithms for solving it, e.g., we have to do a complete search throughout the whole universe. Therefore, it is very likely that on a given fixed model we also have to use a brute-force method and this will be again difficult (even for not so big $n$). For example, checking SAT for a formula containing 5 variables is already quite challenging.

Moreover, even if in typical cognitive situations we are dealing with reasonably small inputs we have no bounds on their size. Potentially, the inputs can grow without limit. Therefore, computational complexity makes perfect sense as a difficulty measure.

Additionally, in experimental practice subjects usually do not realize the size of the universe in which they are supposed to solve some given problem. Therefore, it seems that they develop strategies (algorithms) for all possible universes (sizes).

However, if the size of a problem is too big subjects often start to approximate their solution, using heuristic methods instead of “precise” algorithms. But even
then we can still measure the rate of increase of the difficulty (e.g. by changes in
reaction time) on the interval where subjects realize their algorithmic procedures. For
example, it is known that reaction time increases linearly when subjects are
asked to count objects between 3 and 15. Up to 3 or 4 objects the answer is imme-
diate, so-called subitizing. For tasks containing more than 15 elements subjects
start to approximate: reaction time is constant and the number of incorrect an-
swers increases dramatically (see e.g. Sternberg, 2008). Analogously, we can ask
how reaction time increases when subjects are solving PTIME problems compared
to a situation when they are dealing with an NP-hard task.

There are also other experimental possibilities to observe computational com-
plexity restrictions. For example, one can manipulate the universe to get a drop
in the complexity and see what happens. In Chapter 7 we study the push-down
recognizable quantifier “most” over an appropriately ordered universe, where it
could be recognized by finite automata, and we show a decrease in reaction time
to the level fitting finite-automata problems. Similar methods can be used in case
of NP-hard problems by giving various certificates simplifying tasks and observing
changes in subjects’ behavior.

Finally, theoretically speaking, it may be more beneficial to assume that tasks
are finite but that we do not know upper bounds than to try to develop a com-
plexity measure up to fixed size of a problem. Such a general approach works
very well for syntax, where the assumption that language sentences can be of any
length has led to the development of generative grammar and mathematical
linguistics (see Section 1.7.1 for a short discussion).

Summing up, various computational complexity measures seem to be a very
attractive tool for task-oriented cognitive science. At that point they are the
best established proposition for measuring the difficulty of problems in precise
mathematical terms. Moreover, this perspective seems promising as it was suc-
cessfully applied in a few domains of psychological study which we mentioned
above. However, there is still not enough work in this direction to draw any
strong conclusions.

Last but not least, the biggest quest in applying computational complexity to
cognitive science might be in discovering the right complexity measures. For sure,
such measures we are using now just give a rough approximation of upper-bound
difficulty.

Why Worst-case Complexity?

In our opinion the most serious criticism about the status of the P-cognition
Thesis is that attacking worst-case computational complexity as a measure of
difficulty for cognitive processes. The point here is whether we really need to
consider the worst-case performance of cognitive algorithms on arbitrary inputs.
It might be the case that inputs generated by “nature” have some special proper-
ties which make problems tractable on those inputs even though in principle they
might be NP-hard. It is also possible that our cognitive (linguistic) perception is simplifying things, pre-computing possible inputs. For example, it might be the case that intractability of some problems comes from a parameter which is usually very small no matter how big is the whole input. This way of thinking leads to the consideration of so-called parametrized complexity theory (see Downey and Fellows, 1998; Flum and Grohe, 2006) as a measure for the complexity of cognitive processes. Iris van Rooij (2008) dwells on this subject proposing the Fixed-parameter Tractability Thesis as a refinement of the P-cognition Thesis. Another natural way is to turn toward so-called average-case complexity theory (see e.g. Impagliazzo, 1995; Bogdanov and Trevisan, 2006). It studies the computational complexity of problems on randomly generated inputs. The theory is motivated by the fact that many NP-hard problems are in fact easily computable on “most” of the inputs. One potential problem with applications of this measure to cognitive science is that we are unable to come up with a probability distribution on the space of all possible inputs generated by the “nature”.

In principle, we agree that worst-case computational complexity might not be the measure best tailored for describing the hardness of cognitive tasks. However we also think that an asymptotic analysis of cognitive capacities is the necessary first step in the quest for understanding their complexity. In the book we have tried to initiate this first step in the domain of natural language semantics and hope that it will be the beginning of a journey finally leading to a better understanding of natural language processing.

1.5.4 Explaining the Comprehension Task

Let us summarize what we have said so far by answering to the following question:

How does computational cognition relate to our psycholinguistic perspective?

We want to explain one of the basic components of comprehension. Using the algorithmic theory of meaning we can think about that component in terms of model-checking procedures. Model-checking takes a situation and a natural language expression as an input and then transforms them to the output, i.e., a denotation of the expression in the world. In the paradigmatic case of sentences we get the truth-value. Therefore, model-checking might be treated as an example of a cognitive task.

Now, from the most abstract, computational level of the explanation, we are interested in a mathematical specification of the properties of model-checking. In particular, we will study the computational complexity with respect to logical definability of a given expression. That is why we are interested in the combination of complexity and definability. The theory which pursues the relation between them is called descriptive complexity theory (see Section 2.4). Using it we will investigate the logical properties of natural language quantifiers postulated by
some linguistic formalisms to track down the distinction between quantifiers with tractable (PTIME) and intractable (NP-hard) referential meaning (the model-checking problem).

Another aspect which should be taken into account is the algorithmic level of explanation. Even though it is not the perspective we have explicitly taken in the thesis, algorithmic considerations are a part of our study. We give explicit algorithms which can be used to solve model-checking problems for example as parts of our proofs in Chapter 3. We consider concrete procedures for simple quantifiers in Chapter 7 and investigate what kind of model-checking subjects are realizing for some special sorts of sentences in Chapter 6. However, it might be the case that humans are actually using completely different algorithms. In other words, we only claim that complexity restricts agents. Nevertheless, we do not think that humans are always using the simplest possible procedures.

As far as the implementation level is concerned we do not study neurological activity when natural language is processed. However, our work was to some extent motivated by the neurological research discussed in Chapter 7. Therefore, even though we formally restrict ourselves to classical cognitive science we see a possible direct connection with neurocognitive reality.

Below we discuss more specific impacts of computational complexity on the domain of our interest.

1.6 Finite Interpretations

The above discussion leads us to another restriction of our approach which is partially motivated by the computational nature of our considerations.

Most of the authors considering the semantics of natural language are interested only in finite universes. This is also common in papers devoted to natural language quantifiers; let us cite Dag Westerståhl (1984):

> In general these cardinals can be infinite. However, we now lay down the following constraint:

(FIN) Only finite universes are considered.

This is a drastic restriction, no doubt. It is partly motivated by the fact that a great deal of the interest of the present theory of determiners comes from applications to natural language, where this restriction is reasonable. (Westerståhl, 1984, p. 154)

In typical communication situations we indeed refer to finite sets of objects. For example, the intended interpretations of the following sentences are finite sets:
1.6. **Finite Interpretations**

(14) Exactly five of my children went to the cinema.

(15) Everyone from my family has read *Alice’s Adventures in Wonderland*.

In many cases the restriction to finite interpretations essentially simplifies our theoretic considerations.

First of all, there is a conceptual problem with computational complexity theory for an infinite universe. Even though from the mathematical point of view we can work with infinite computations, the classical study of resource bounds in such a setting does not make a lot of sense as we need infinite time and infinite memory resources. On the other hand, maybe we need only finitely many states or loops. Hence, after all it is a matter of proposing reasonable definitions and finding interesting applications of infinite computations in cognitive modeling. For example, they can be useful as a framework for some cognitive processes which at least in theory are supposed to continue working indefinitely, like equilibrioception.

From the semantic perspective there is, however, an additional problem with infinite universes. Namely, defining an intuitive semantics for natural language in an arbitrary universe is a very difficult task. As an example of potential trouble-makers we can give quantifiers, like “more” or “most”. We usually reduce the problem of their meaning to a question about the relations between sets of elements (see Section 2.2). In finite universes there is an easy and commonly accepted solution, which is to compare the cardinal numbers of these sets. But extending this solution to the infinite case seems to be very counterintuitive. Let us have a look at the following sentence.

(16) There are more non-primes than primes among integers.

The sentence is false if we interpret “more” in terms of cardinal numbers of sets. However, intuitively we agree that the sentence is meaningful and even true.

One possible solution to this problem is to treat such quantifiers as measure quantifiers. In infinite universes we can compare quantities by a proper measure functions, which are non-logical and context dependent concepts (see Krynicki and Mostowski, 1999, for more details).

Another approach to the problem can be formulated in terms of the so-called weak semantics for second-order definable quantifiers. The main idea of the weak semantics for second-order logic is to consider structures of the form \((\mathbb{M}, \mathcal{K})\), where \(\mathbb{M}\) is a model and \(\mathcal{K}\) is a class of relations over the universe closed under definability with parameters in a given language. The class \(\mathcal{K}\) is used to interpret second-order quantification. Phrases like: “for every relation \(R\)”, “there is a relation \(R^*\)” are interpreted in \((\mathbb{M}, \mathcal{K})\) as “for every relation \(R\) belonging to \(\mathcal{K}\)” and “there is a relation \(R\) belonging to \(\mathcal{K}\)”. This way of interpreting second-order quantification essentially modifies the standard semantics for second-order logic.
and gives an intuitive semantics for second-order definable quantifiers in arbitrary universes from the natural language point of view (see e.g. Mostowski, 1995).

In this dissertation we work with finite universes as we are mainly interested in the algorithmic aspects of the semantics. However, we find a development toward covering arbitrary universes by adopting the methods outlined above and a proper complexity measure a fascinating challenge for later work.

1.7 Complexity in Linguistics

The topic of language complexity has surfaced in many different contexts and can be measured in many different ways. We are talking only about the computational and descriptive complexity of language, but there are many other aspects of complexity, like lexical, information-theoretic (Kolmogorov complexity), structural or functional. These are usually studied in less logical manner (see e.g. Miestamo et al., 2008) and we do not compare these approaches directly.

1.7.1 Syntax

Noam Chomsky has opened the door to a view of language from the computational perspective (see e.g. Chomsky, 2002, 1969). In the early 1950s he captured language’s recursive nature, or ability to make “infinite use of finite means”, in Wilhelm von Humboldt’s famous words, inventing formal language theory. Chomsky’s complexity hierarchy of finite-state, context-free, and context-sensitive languages associated with their automata counterparts (see Section 2.3.1) changed linguistics and opened it to many interactions with computer science and cognitive science.

Chomsky’s insight was really of the nature of computational complexity theory. First of all, even though the sentences we encounter are all of bounded length — certainly less than 100 words long — Chomsky has assumed that they might be arbitrarily long. Notice that this is an assumption of exactly the same sort as discussed in Section 1.5.3 on the limit behavior of computational claims, i.e., considering inputs of arbitrary size. This assumption directly lead to the discovery of the computational model of language generation and the complexity hierarchy.

Next, using his complexity hierarchy Chomsky asked in which complexity class natural language lies. He has early demonstrated the inadequacy of finite-state description of natural languages and has claimed that natural languages are context-free (see Chomsky, 2002). This was a pathbreaking way of thinking. First of all, it has shown how one can mathematically measure some complexity aspects of natural language. Then, it has given a methodological constraint on linguistic theories of syntax; they have to be able to account at least for the context-free languages. Actually, it has also started a long standing debate whether natural
1.7. Complexity in Linguistics

language is context-free or not (see e.g. Pullum and Gazdar, 1982; Shieber, 1985; Culy, 1985; Manaster-Ramer, 1987).

Chomsky himself noticed very quickly that studying whether a grammar can generate all strings from a given language, so-called weak generative capacity, can serve only as a kind of “stress test” that doesn’t tell us much unless a grammar fails the test. He has claimed:

The study of generative capacity is of rather marginal linguistic interest. It is important only in those cases where some proposed theory fails even in weak generative capacity — that is, where there is some natural language even the sentences of which cannot be enumerated by any grammar permitted by this theory. [...] It is important to note, however that the fundamental defect of many systems is not their limitation in weak generative capacity but rather their many inadequacies in strong generative capacity.\(^{13}\) [...] Presumably, discussion of weak generative capacity marks only a very early and primitive stage of the study of generative grammar. (Chomsky, 2002:60f)

In subsequent years this belief has led to deeper study of generative formalisms. In particular, using computational complexity one could answer the question whether some proposed generative formalism is plausible in the sense of being “easy enough to process”. To be more precise, computational complexity of parsing and recognition has become a major topic along with the development of computational linguistics. The early results achieved are summarized in a book by Barton et al. (1987). A more recent survey is due to Pratt-Hartmann (2008). In general, the results show that even for relatively simple grammatical frameworks some problems might be intractable. For example, regular and context-free languages have tractable parsing and recognition problems. However, Lambek grammars, Tree-Adjoining Grammars, Head-Driven Phrase Structure Grammar, and context-sensitive grammars give raise to intractable problems.

1.7.2 Between Syntax and Semantics

Model-theoretic Syntax

One can specify the complexity of grammars not only in terms of generative mechanisms but also via a set of general constraints to which sentences generated by these grammars have to conform. On this view, a string (or a tree) is grammatical if it satisfies these constraints. How can we define complexity from this perspective? The idea is to identify the complexity of a grammar with the logical complexity of the formulae which express the constraints. In other words, we ask

\(^{13}\)Chomsky uses the term “strong generative capacity” referring to the set of structures that can be generated by a grammar.
here about descriptive complexity of grammars in a similar way to how we will be asking about descriptive complexity of model-checking problems in the following chapters.

We measure the complexity of the sentences that define constraints by saying how strong a logic we need to formulate them. Particularly, we refer to fragments of second-order logic (see e.g. Rogers, 1983) or various extensions of modal logic (see e.g. Kracht, 2003). For illustration, the two best known results of this approach are as follows. In his seminal paper Büchi (1960) showed that a language is definable by the so-called monadic fragment of second-order logic if and only if it is regular. McNaughton and Papert (1971) have proven that a set of strings is first-order definable if and only if it is star-free. These two results have their counterpart in modal logic: the temporal logic of strings captures star-free languages and propositional dynamic logic captures regular languages (see e.g. Moss and Tiede, 2006).

Notice that it is often possible to draw conclusions about computational complexity from such descriptive results. It is enough to know the complexity of the corresponding logics. For instance, we know that monadic second order logic on trees is decidable although intractable and therefore many linguistic questions about parse trees which are formulated in this logic might be “difficult” to answer. Our approach to studying the complexity of semantics goes along the same descriptive line.

**Discourse Complexity**

Some of the earliest research trying to combine computational complexity with semantics can be found in Sven Ristad’s book “The Language Complexity Game” (1993). The author carefully analyzes the comprehension of anaphoric dependencies in discourse. He considers a few approaches to describing the meaning of anaphora and proves their complexity. Finally, he concludes that the problem is inherently NP-complete and that all good formalisms accounting for it should be exactly as strong.

Pratt-Hartmann (2004) has shown that different fragments of natural language capture various complexity classes. More precisely, he has studied the computational complexity of satisfiability problems for various fragments of natural language. He has proven that the satisfiability problem of the syllogistic fragment is in PTIME as opposed to the fragment containing relative clauses which is NP-complete. He has also described fragments of language such that their computational complexity is even harder (with non-copula verbs or restricted anaphora) and finally he provides the reader with an undecidable fragment containing unrestricted anaphora.

It is also worth to mention very recent paper of Pagin (2009). The author tries to explain compositionality principle in terms of computational complexity, cognitive difficulty as experienced by language users during communication, and
1.8. Classifying Meaning by Intractability

We have argued that meaning is an algorithm. This allows us to apply computability considerations to linguistics. Moreover, according to what we said above linguistic capacities, like all cognitive processes, are bounded by computational restrictions. In particular, comprehension of natural language constructions is linked to their computational complexity. We provide some more arguments in favor of this connection throughout the dissertation. We give an example for this dependence in Section 4.5 where we discuss the interpretation of NP-complete reciprocal sentences and suggest that their meaning might be in most cases only comprehended by recognizing inferential properties. Then in Chapter 5 we have a look at collective quantification in natural language, where among others we can observe how computational complexity restricts expressibility of everyday language learnability. He argues that compositionality simplifies complexity of language communication.

Unfortunately — as far as we are aware — not much is known about the computational complexity of dynamic formalisms. This is surprising because complexity questions match the dynamic spirit perfectly and moreover can give deeper insights into the plausibility of proposed formalisms. Some complexity results are known for belief revision — mostly showing that the logic lying behind revision mechanisms is highly intractable (see e.g. van Benthem and Pacuit, 2006) — and for game-theoretic semantics (see e.g. Sevenster, 2006). But there are some very interesting questions untouched, for example: What is the complexity of a belief revision (game) when an intractable sentence is the input? Does model-checking complexity relate to the complexity of belief revision or the corresponding game in any way? But first of all, there has not been much done about the complexity of this model from the agent’s perspective. For example, how difficult is belief revision for a single agent?

The worst news is that complexity seems to be a totally open problem for formal pragmatics, even though the question of how difficult it is to solve coordination problems in linguistic games seems to be very basic. Probably, some results might be achieved via translation from complexity considerations in algorithmic game theory (see e.g. Nisan et al., 2007). We believe that computational complexity issues should be incorporated in these models to better understand their plausibility.

In this book we study the computational complexity of model-checking for a quantifier fragment of natural language. We often evoke the descriptive perspective to achieve our aims. In other words, we try to understand some aspects of meaning from the computational perspective. Below we say a few words on the underlying assumptions in this enterprise, summing up the previous considerations on semantics, cognitive science and complexity.

1.8 Classifying Meaning by Intractability
Chapter 1. Algorithmic Semantics

language. In Chapters 6 and 7 we present empirical evidence supporting the statement that computational complexity influences the comprehension of natural language. In the first of these chapters we show that people tend to choose computationally simpler interpretations of so-called Hintikka-like sentences. In the following chapter we give empirical evidence directly linking the reaction time needed to recognize the meaning of sentences with simple quantifiers to the complexity of automata computing this meaning. Below we try to capture our experience with studying the complexity of natural language into the form of a methodological statement about the descriptive power of the linguistic formalism which we needed.

1.8.1 Ristad’s Theses

The book of Ristad (1993) contains not only a technical contribution to the semantic theory of anaphora but also some methodological claims on the role of computational complexity in linguistics.

First of all, Ristad claims that natural language contains constructions which semantics is essentially NP-complete:

The second major result of this monograph is the thesis that language computations are NP-complete. This complexity thesis is a substantive, falsifiable claim about human language that is directly supported by the quiescent state of the language complexity game.

(Ristad, 1993, p. 14)

Secondly, Ristad suggests that a good linguistic theory cannot be too strong:

The central consequence of this complexity thesis for human language is that empirically adequate models (and theories) of language will give rise to NP-completeness, under an appropriate idealization to unbounded inputs. If a language model is more complex than NP, say PSPACE-hard, then our complexity thesis predicts that the system is unnaturally powerful, perhaps because it overgeneralizes from the empirical evidence or misanalyses some linguistic phenomena.

(Ristad, 1993, p. 15)

In other words, Ristad claims that every semantic model has to be at least NP-hard to be able to capture complex linguistic phenomena. On the other hand, in his opinion every computationally stronger formalism is overgeneralizing linguistic capacities. This second methodological claim can be treated as a kind of semantic Ockham’s razor. Summing up, Ristad proposes NP-completeness as a kind of methodological test for plausibility of linguistic theories.
1.8.2 Descriptive Bounds on Everyday Language

We do not completely agree with Ristad but we share his intuitions. However, we prefer to restrict claims of that sort to some fragment of language as the whole of natural language contains expressions whose complexity go beyond practical computability. Nevertheless, we can see a natural intuition supporting the use of the concept of natural language excluding “technical” expressions. However, in this narrow sense we prefer to use the term everyday language. In a sense we understand “everyday language” here as a pre-theoretic part of natural language.

In this dissertation we study the complexity of natural language quantifiers via their definability properties. In other words, we use descriptive complexity theory, i.e., we draw complexity conclusions on the basis of logical descriptions of semantics. From this perspective, our experience allows us to state the following claim which resembles Ristad’s thesis and was proposed by Mostowski and Szymanik (2005):

\[ \Sigma_1 \text{-thesis} \] Our everyday language is semantically bounded by the properties expressible in the existential fragment of second-order logic.

In other words, we claim that everyday language can be described in the existential fragment of second-order logic (see Definition 2.1.1). If some property is not definable by any \( \Sigma_1 \)-formula, then it falls outside the scope of everyday language. For example, quantifiers “there exists”, “all”, “exactly two”, “at least four”, “every other” and “most” belong to everyday language. The counterexample is the notion “there exist at most countably many” which is not definable by any \( \Sigma_1 \)-formula. Moreover, we know from Fagin’s theorem (2.4.4) that \( \Sigma_1 \)-properties correspond to NPTIME. Hence, our thesis basically restricts the methodological claim of Ristad to the realm of everyday language.

Let us give one argument in favor of accepting this methodological statement. The other one, more domain specific, will be formulated in Section 6.3.4. The intuition is that the core sentences of everyday language are sentences which can be more or less effectively verifiable. We argued that in the case of small finite interpretations this means that their truth-value can be practically computed, directly or indirectly. Direct practical computability means that there is an algorithm which for a given finite interpretation computes the truth-value in a reasonable time. From the P-cognition Thesis it follows that the referential meaning of an expression which can be directly computed is in PTIME. However, as we mentioned in Section 1.3 we frequently understand sentences indirectly, evoking their inferential dependencies with other sentences. Let us consider the following three sentences:

(17) There were more boys than girls at the party.

(18) At the party every girl was paired with a boy.
Chapter 1. Algorithmic Semantics

(19) Peter came alone to the party.

We know that sentence (17) can be inferred from sentences (18) and (19). Then we can establish the truth-value of sentence (17) indirectly, knowing that sentences (18) and (19) are true.

The question arises: What do we mean by a tractable inferential meaning? First notice that in the case of NPTIME problems the non-deterministic behavior of an algorithm can be described as follows:

Firstly, choose a certificate of a size polynomially depending on the size of input. Then apply a PTIME algorithm for finding the answer. The nondeterministic algorithm answers YES exactly when there is a certificate for which we get a positive answer.

(Garey and Johnson, 1979, p. 28)

Let us observe that in a sense such certificates are proofs. When we have a proof of a statement then we can easily check whether the sentence is true. Therefore NPTIME problems ($\Sigma_1^p$-properties) are practically justifiable. Let us consider an example.

Suppose that we know that the following are true statements:

(20) Most villagers are communists.

(21) Most townsmen are capitalists.

(22) All communists and all capitalists hate each other.

From these sentences we can easily infer the NP-complete branching interpretation of the following sentence (for a discussion on the computational complexity of sentences like this see Section 3.2 and Chapter 6):

(23) Most villagers and most townsmen hate each other.

Sentences (20), (21), and (22) have to be given or guessed. They are in a sense certificates or proofs of the truth of sentence (23).

In this sense sentences with NPTIME meaning — or by Fagin's theorem (see Theorem 2.4.4) $\Sigma_1^p$-expressible sentences — are indirectly verifiable. Moreover, NPTIME seems to capture exactly indirect verifiability. This observation gives an argument for our claim — everyday sentences have practically computable inferential meaning.

We assume the $\Sigma_1^p$-thesis — as a methodological principle — throughout the dissertation. We are using it to argue in favor of or against some linguistic formalizations. For instance, in Chapter 5 we study a common way of modeling collective quantification in natural language, the so-called type-shifting approach. By using the formalism of second-order generalized quantifiers, we observe that
1.9 Summary

In this chapter we have discussed some motivations and philosophical assumptions lying in the background of the research presented in the next chapters. These assumptions do not have a direct influence on our technical results. However, we believe that they give an additional argument for studying the computational complexity of natural language expressions.

Below we summarize our assumptions:

- We mostly study a basic element of comprehension: model-checking, to which we sometimes refer as "referential meaning".

- We identify (referential) meanings (algorithms) with Turing machines (computable functions).

- We assume that problems are tractable if they are computable in polynomial time. Otherwise, they are intractable.
• We assume that human cognitive (linguistic) capacities are constrained by polynomial time computability. Therefore, if a meaning corresponds to intractable functions we believe it is too hard for direct comprehension.

• We claim that the semantics of everyday language can be adequately described in the existential fragment of second order-logic ($\Sigma^1_1$-thesis).

• We restrict ourselves to finite universes.

We hope that our technical insights will give additional arguments in favor of our assumptions.