Introduction

In computer science, the main task is to study the structure of computational problems, and possible algorithms to solve them. Recursion theory has provided many answers to the question which of the problems appearing in a computer scientist's every day life can be solved with an algorithm, and, more importantly, which cannot. In fact, by a simple argument comparing the number of possible algorithms and the number of possible problems, it is evident that "most" problems cannot be solved by any algorithm at all. Recursion theory also provided a suitable model for computation, which is independent of whatever kind of computer hardware might be in fashion at a given date: the Turing machine is both universal enough to serve as the general definition of a "computer," and simple enough for the researcher to prove results without too much technical reliance on the model itself.

For problems appearing in practice, the answer "there is an algorithm to solve it" is not entirely satisfying. Usually, we are interested in an algorithm solving the problem at hand using as few resources as possible. Among the most important measurements of resources are the time needed for the computation, and the memory used by an algorithm. Therefore, questions like "is there an algorithm to solve this problem in a time which is linear in the input length" become important. This is where complexity theory has provided many answers, and, maybe as important, interesting questions.

One of the most basic questions that complexity theory considered was the question how to define an efficient algorithm. The answer which is generally agreed on by people working in the field today is the following: an algorithm is efficient if the time it needs to perform its task on a Turing machine is polynomial in the length of its input. It can be shown that this class is "robust," meaning that if we study the class of problems solvable on a "real" computer in polynomial time instead, we get the same class of problems. Thus, the class P containing all problems with a polynomial-time algorithm is considered to contain all problems which can be solved "efficiently." In addition to this class, many other complexity classes have been defined, which are meant to group problems where the computational power required to solve them is similar. The question to determine for some problem, in which complexity class it belongs, is therefore the same as asking what resources we need to solve it.

It is obvious that positive results in the way of "there is an efficient algorithm to solve problem A" can be shown by simply stating an algorithm for the problem at hand, proving its correctness and analyzing its running time. But what about negative results, proving that there is *no* efficient algorithm for a given problem? Results of this kind have proven to be much more difficult to achieve. In fact, for many problems which are very relevant in practice, it is unknown if an efficient algorithm can exist.

To be able to compare the complexity of given problems, the notion of *reductions* was introduced. In this way, even if we do not know if we can solve the problems A or B efficiently, it is possible to prove statements like "either both A and B have an efficient algorithm, or none of them has." Using this concept, it was shown that many problems appearing in practice are "equivalent" in a certain way: either all of them can be solved by an efficient algorithm, or none can. These problems form the fundamental class of NP-complete problems. The question if an efficient algorithm for this class of problems exists is one way to phrase complexity theory's most important open question: the "P-NP"-problem. The first known mention of this problem is in a letter by Kurt Gödel to John von Neumann, where he asks if there is a better method to prove first-order formulas than simply testing all combinations. Despite considerable efforts by computer scientists and mathematicians for more than 35 years, this question remains open. Most researchers believe that the answer to it is "no," and therefore to say that a problem is NP-complete is now generally understood as meaning that there probably is no efficient algorithm for it.

But what about cases "in between" efficiently-solvable and NP-complete? It is very conceivable that there are problems which are "easier" than the NP-complete problems, but still do not have an efficient algorithm. Under the assumption that the NP-complete problems themselves cannot be solved efficiently, this, and in fact a much stronger result, has been proven by Richard Ladner in [Lad75b]: there are infinitely many "degrees of complexity" between P and NP. It is of interest that very few "natural" problems appear to lie in these intermediate degrees. One of the well-known candidates for such a problem is graph isomorphism, which is the problem to determine if two given graphs are mathematically the same structure.

From the very beginning of the study of these problems, in fact starting withthe above-mentioned letter by Gödel, propositional formulas lay at the heart of the discussion. One of the most important problems in complexity theory is the *satisfiability problem*, which is the following: given a propositional formula, determine if there is an assignment which makes the formula true. For example, consider the formula

$$\varphi_1 = x \wedge (y \vee \overline{x}).$$

This formula can easily seen to be *satisfiable*, by setting both variables x and y to "true." On the other hand, consider the formula

$$\varphi_2 = (\overline{x} \vee y) \wedge (\overline{y} \vee z) \wedge x \wedge \overline{z}.$$

This formula is not satisfiable: for any "true/false"-assignment to the variables x, y, and z, the formula is false. The satisfiability problem seems like a simple enough question to be solved by an algorithm: simply test all of the possible truth assignments to the variables, and check if one of them makes the formula true. While this procedure certainly is correct, it cannot be considered efficient: for a formula in which n variables appear, there are 2^n many possible truth assignments to the variables which need to be tested. Assuming that a computer can test 1.000.000.000 assignments per second, this would mean that for a formula with 1000 variables, the algorithm would roughly take $3 \cdot 10^{284}$ years to check all possible assignments. Since formulas of this length do appear

in practical settings, this obviously is not satisfactory. However, there are no known algorithms which solve the problem significantly faster. The P-NP-problem can be stated as the question if there is an algorithm which can do significantly better, i.e., perform only a polynomial number of computation steps instead of the exponentially many tests needed for the complete search algorithm described above.

Although we do not know how to solve this problem efficiently, for a satisfiable formula, it is easy to "prove" that it is indeed satisfiable, by simply giving a satisfying assignment, as we did above for the formula φ_1. There are many problems which share this characteristic. For another example, consider the *Traveling Salesman Problem*. Here we are given a set of cities, a table of plane ticket costs for each city-to-city connection, and a number c. Our task is to determine if there is a round-trip which costs at most c Euros. Again, given such a round-trip, it is easy to check if it satisfies the cost bound. But it seems to be difficult to answer the question if such a trip exists. This property gives a characterization of the problems in the complexity class NP: we do not know how to solve them efficiently, but there are short and easily verifiable "proofs" to show that the answer to such a question is "yes."

The satisfiability problem was the first problem proven to be NP-complete, by Stephen Cook in [Coo71], and, independently, by Levin (a partial English translation of his result can be found in [Tra84]). From that starting point on, literally thousands of problems were proven to fall into this class, and entire books are devoted to proving these kinds of results [GJ79]. The search for NP-complete problems is motivated by two main reasons. First, if there is one problem of these which can be solved in polynomial time, then this immediately gives efficient algorithms for all the NP-complete problems. Therefore, it was hoped that if enough NP-complete problems were known, then there would be discovered some problem which is both NP-complete and efficiently solvable, thus proving that P=NP, and giving efficient algorithms for a vast number of practically relevant problems. However, this has not happened, and in fact, most researchers now believe that it never will, since there probably simply are no efficient algorithms for NP-complete problems. But the search for NP-complete problems still remains interesting: when analyzing the complexity of a problem occurring in practice, in order to prove that it is NP-complete, it is useful to have a problem as "similar as possible" to it for which completeness is known. If for a practical problem we know that it is NP-complete, then we know that with known algorithms and techniques, we cannot obtain an efficient solution, and we need to consider approximation algorithms. Therefore, knowing many completeness results helps to influence decisions in practical software design.

In the above mentioned Traveling Salesman Problem, the goal was to find a strategy of visiting cities. In the satisfiability problem, we search for a strategy to assign truth values to the variables. In both examples, there was no opponent we needed to take into account. When we add possible opponents, and study problems in a game-theoretic setting, then often problems which cannot be solved in NP anymore occur. Consider the following "game:" We are given a propositional formula φ, where the occurring variables are $x_1, \ldots, x_n$. Player A starts to assign a value to the first variable, x_1. Then player B may choose a value for the variable x_2, then it is A's turn again and he determines the value for x_3, and so on, until every variable has been assigned a value. Player A (the *universal player*) wins if the formula φ is false under this assignment, and player B (the

existential player) wins if the formula is true. The question if player A or B have a winning strategy in this game does not seem to be solvable in NP, because unlike with short and easily-checkable assignments for a formula, there does not seem to be short way to encode the strategies in this more general setting. The obvious approach would be to write down every possible "reply" to the other player's moves, but it is easy to see that this results in a table with exponentially many entries in the number of rounds of the game. Problems like these therefore lie in higher complexity classes - the classes arising here are those forming the *polynomial hierarchy*, and the class PSPACE. The example just discussed can be phrased as the validity problem for a *quantified Boolean formula*, where the variables controlled by A are quantified with $\forall$, and the variables controlled by B are quantified with $\exists$.

As mentioned, the class P is considered as the class of problems which can be solved efficiently. It is obvious that there are different "degrees" of efficiency, and hence it is natural to study complexity classes below P. To this end, alternative models of computation were introduced, allowing to obtain results on questions of efficient parallel algorithms, and algorithms with low space usage. Both extensions of the Turing machine and different, circuit-based models were introduced, which allow natural definitions of various complexity classes inside P. Similarly to the NP-complete problems, the notion of completeness for these classes was introduced to describe problems which are "among the hardest" in them. Again, it turned out that satisfiability problems related to restricted classes of Boolean formulas are typical examples for complete problems of these complexity classes. Therefore, a systematic study of these restricted satisfiability problems is of interest, to gain insight into those complexity classes with deep connections to Boolean formulas.

There are two different systematic ways of phrasing the restrictions of propositional formulas that we consider in this thesis. A propositional formula is usually defined to be built of propositional variables, constants, and the operators $\wedge, \vee$, and $\neg$, representing conjunction, disjunction, and negation. What happens if we remove one of them? It is obvious that removing either $\wedge$ or $\vee$ does not reduce the expressive power of the formulas, since we can simulate one of them using the other and negation: $x \wedge y$ is equivalent to $\overline{(\overline{x} \vee \overline{y})}$, and analogously $x \vee y$ can be expressed as $\overline{(\overline{x} \wedge \overline{y})}$. But what if we forbid negation? It is easy to see that the satisfiability problem for negation-free formulas is much simpler than the one for arbitrary formulas: We can simply set every variable occurring in a given formula to 1, and if this assignment does not satisfy the formula, then no assignment will. This problem is not only solvable by an efficient, i.e., a polynomial-time algorithm, but there are efficient parallel algorithms for this problem, as we will see as an easy corollary from the results in Chapter 2. But what about other possible operators, like implication? Or the binary exclusive-or? In fact, we can introduce any Boolean function as an operator allowed in propositional formulas. For each possible set of Boolean functions, this gives a restriction of formulas: the class of formulas built using variables and these connectors. Therefore, we can define an infinite number of possible restrictions in this way, and for each of these restrictions, we obtain a new version of the satisfiability problem, each with a potentially different complexity.

To consider an infinite set of problems, we need some structure on this set. One of the most important results in the classification of the expressive power of these restricted

formulas was Emil Post's work regarding certain "closed classes of Boolean functions." He proved his results already in the 1920s, but his work was not published until 1941, in [Pos41]. His results identify all the "classes of expressiveness" which can be generated by Boolean formulas restricted in this way, and therefore allow for a systematic study of restrictions of propositional formulas by limiting, or extending, the possible operators in the way suggested above. His classification is now known as *Post's lattice*.

One of the first known results applying Post's work to complexity theory and the study of satisfiability problems was achieved by Harry Lewis in [Lew79], where he examined the question which operators make the satisfiability problem NP-complete, and which combinations give efficient algorithms. In particular, he showed that this problem is "dichotomic:" the complexity degrees between NP-completeness and solvable in polynomial time mentioned above do not appear here. Dichotomy results are very interesting in complexity theory: for one, they show that the infinite class of problems in question breaks down to finitely many, if we are only interested in their "complexity behaviors." In this way, it is shown that an infinite class of problems can be considered "the same" from a computational point of view. Also, in many cases a dichotomy theorem demonstrates the exact point where the problem gets difficult, and can therefore give a precise description of the features which make the problems in question hard. Lewis' work gives a precise answer what kind of operators used in formulas make the problem "easy," and which make them NP-complete.

Another restriction is to remove one of the most important features from Boolean formulas, which is nesting. A usual Boolean formula can be nested to any degree. By only considering formulas in *conjunctive normal form*, the nesting degree is reduced to a constant. These are formulas of the form $C_1 \wedge \cdots \wedge C_n$, where the "clauses" C_i must be of a very simple and regular form. It turns out that if we allow arbitrary clauses with up to three variables, the satisfiability problem for these formulas is still NP-complete. If we restrict the number of variables appearing in each clause to 2, then the problem is solvable in nondeterministic logarithmic space, which is a subclass of P. But there are other possible restrictions on these clauses than just limiting the number of variables allowed to occur. A systematic study of these restrictions is known as the *constraint satisfaction problem*. In its non-uniform version, this problem studies so-called Γ-formulas, where the appearing clauses must take the form of some "templates" defined in a set Γ. For the Boolean case, Thomas Schaefer showed in [Sch78], that again, the problem is dichotomic: such a problem either can be solved in polynomial time, or is NP-complete. Surprisingly, this result can be proven by again applying Post's lattice mentioned above. Post's classification is used indirectly here, with an interesting "*Galois connection*" between Boolean functions and closure properties of the clauses allowed in the language Γ. It can be shown that both of these restrictions can be phrased in an algebraic context, and the lattices of closed sets that appear in both cases are dually isomorphic. This means that Post's analysis of the closed classes of Boolean functions also gives us a complete list of cases to study in the constraint satisfaction setting. However, this isomorphism does not seem to allow the direct transfer of complexity results from one of the restrictions to the other.

Constraint satisfaction problems have very interesting theoretical properties, as their dichotomic complexity behavior and the connections to universal algebra. But there also

is a vast number of practical applications. Constraint satisfaction problems generalize not only many well-studied cases of the satisfiability problem, but can be used to express almost any combinatorial problem which can be phrased as a set of local conditions. For example, constraint satisfaction problems play a role in database theory, electronic design automation, scheduling problems, and many other computational settings. On the theoretical side, they generalize problems like graph colorings, graph search, various flavors of satisfiability problems, and many more. Therefore, constraint satisfaction problems can be seen as the "combinatorial core of complexity theory" [CKS01], and hence learning about constraint formulas gives us better insight into many of complexity theory's questions.

In this thesis, we study various forms and generalizations of the satisfiability problem, which using the systematic restrictions explained above. In addition to the satisfiability problem itself, we also consider the closely related problems of model checking, enumeration, counting, and equivalence. The structure of the work is as follows: After recalling prerequisites from the literature and proving some initial results about formulas and relations of our own in Chapter 1, we start with considering formula restrictions in the Post sense. One of the simplest possible questions which can be asked in this context is the problem to determine if a given formula in which no variable appears is true. This problem, called the *formula value problem*, can be seen as the most basic satisfiability problem, where no assignment to the variables has to be considered, but a formula simply has to be evaluated. This task is one of the most important ones arising in algorithms dealing with propositional formulas. It turns out that this problem has efficient parallel algorithms for all types of formulas that we consider, and again we show that if we restrict the propositional operators appearing in the formula, the complexity of the problem decreases even further. Using Post's lattice, we show that there is a finite number of complexity classes such that for any choice of propositional operators, the formula value problem is complete for one of these classes. While this is not a "dichotomy" in the strictest sense, since there are more than two complexity cases arising here, it still shares the properties of dichotomy results which make them so interesting: the complexity of an infinite set of problems can be shown to only give complexities from a finite list.

In practice, knowing the actual solutions to a problem is often more interesting than simply knowing whether at least one solution exists. In Chapter 3, we therefore turn our attention to the problem of computing the set of satisfying assignments for a given propositional formula. This is not a decision problem like the ones mentioned up to now, where the answer to the question is simply "yes" or "no," but a problem where the task is to generate a set of assignments for a given formula. Hence, these problem cannot be grouped into the usual complexity classes of decision problems, like P or NP. Instead, we consider several notions of "efficient enumeration" suggested by David Johnson, Christos Papadimitriou, and Mihalis Yannakakis in [JPY88]. Assuming $P \neq NP$, for each possible restriction of propositional formulas, and each of the efficiency notions considered, we answer the question if such an algorithm exists.

In the remainder of the thesis, we study problems for formulas restricted in the constraint satisfaction context. In Chapter 4, we refine Schaefer's dichotomy theorem for formulas in conjunctive normal form, and consider the subclasses of polynomial time. It turns out that the Galois connection mentioned before has its limitations here: there are

cases which have the same algebraic behavior, but lead to different degrees of complexity. Hence we need to go beyond the classification provided by the algebraic properties, and perform a finer analysis of the cases. It turns out that the problem still is dichotomic in nature, revealing that each of these problems is equivalent to the standard "complete" problems of standard complexity classes inside P. Finally, in Chapter 5, we consider quantified constraint formulas. These are generalizations of the usual constraint formulas, where additionally the quantifiers $\exists$ and $\forall$ are allowed to occur. As hinted above, such formulas can be used to describe settings where two opponents are working against each other. It is well-known that adding these quantifiers to the formulas raises the complexity of the involved decision problems significantly: the problems we consider in this chapter are prototypical for the classes of the polynomial hierarchy, and for the class PSPACE, containing all computational problems which can be solved in polynomial space. We study various problems for these formulas: first, we consider the formula evaluation problem in this context, and the closely related model checking problem. Another decision problem which is very interesting is the equivalence problem, where we ask if two formulas have the same set of satisfying assignments. This question is very important in practice, since it can be used to decide whether two given database queries are equivalent, if a program behaves as its specification demands, or if two games have the same winning strategies.

Finally, we consider the *counting problem* for these formulas, which is the task to determine the number of satisfying assignments for a given formula. This problem arises in practice when we want to determine the number of elements in a database which match a given query. To study the complexity of these problems, counting complexity classes have been introduced, which have a close relationship to the classes of decision problems.

In all of the problems considered in this thesis, we show dichotomy-like results, showing that for an infinite set of problems, only a finite set of complexity classes arises, and the problems turn out to be complete for these classes. Hence, among adding to the list of complete problems for all kinds of classes, we show that all of these infinite classes of problems break down into finitely many complexity cases. Therefore, from a computational point of view, there are only finitely many different problems in this context. For the problems considered in Chapter 4, this can be made even stricter, as in fact we can show that the problems only give rise to finitely many equivalence classes under the much stronger notion of isomorphism.

Publications

The material about bases for co-clones, i.e., the results presented in Table 1.2, the surrounding discussion, and Lemma 1.5.6 previously appeared in [BRSV05]. Lemma 1.4.5 appeared in [Sch05], on which Chapter 2 is based. Chapter 4 previously appeared as [ABI+05], and the results on counting and the $\mathsf{QCSP}_k(\Gamma)$-problem from Chapter 5 appeared in the technical report [BBC+05]. Further results in that chapter contain unpublished work with Michael Bauland, Nadia Creignou, and Heribert Vollmer. The results from Chapter 3 are new.