ABSTRACT

In the first main part of the thesis the descriptional complexity of minimal pumping constants—the smallest value satisfying a previously fixed pumping lemma—is studied. Therefore, we compare the minimal pumping constants of various pumping lemmata. In this way, we improve on the simultaneous regulation of minimal pumping constants and other measures, such as the nonterminal complexity and the (non-)deterministic state complexity. As a consequence of the obtained results, we derive a complete hierarchy of the studied measures for regular languages (REG). Afterwards, we study the operational complexity of minimal pumping constants w.r.t. several pumping lemmata, whereby we also differentiate the input languages according to size of their input alphabet. For each case we are able to completely classify the range of the minimal pumping constant for the operations Kleene star, reversal, complement, prefix- and suffix-closure, union, set-subtraction, concatenation, intersection, and symmetric difference. In this way, we also solve some related open problems from the literature.
The second main part will be dedicated to the computational complexity of the Pumping-Problem, that is, for a given finite automaton A (or a grammar G) and a value p, to decide whether the language $L(A)$ ($L(G)$, respectively) satisfies a previously fixed pumping lemma w.r.t. the value p. We show that the problem is decidable for all studied pumping lemmata, if the language under consideration is regular, and the problem becomes undecidable if the language is (linear) context-free. More interestingly, we prove that for two supersets of REG, namely for k-rated linear languages and for well-matched visibly pushdown languages, the Pumping-Problem remains decidable. It turns out that this problem is computationally intractable for REG, namely, it is coNP-hard for binary deterministic finite automata (DFAs) and for unary nondeterministic finite automata (NFAs). Among other complexity results, we show that the minimal pumping constant for binary n-state DFAs and unary n-state NFAs cannot be approximated within a factor dependent on n unless the Exponential Time Hypothesis fails. To that end, we describe those homomorphisms that, in a precise sense, preserve the respective pumping arguments used in two different pumping lemmata. Indeed, this concept coincides with the classic notion of star height preserving homomorphisms. Also, we gain a complete understanding of the minimal pumping constants for bideterministic finite automata and build a framework for proving that the Pumping-Problem for certain subregular languages is coNP-complete.

Part I

PREAMBLE

1

INTRODUCTION

In the beginning there were symbols. More precisely, there were finite and infinite sequences of symbols, called words, there were sets of words, named languages, and the attempt to find some kind of patterns in this chaos. In the early 20^{th} century Axel Thue studied the properties of finite and infinite words, see [97, 99]. In fact, this can be seen as the point of origin for formal language theory. A famous decision problem in this field is the WORD PROBLEM, which asks for two given expressions whether they are equivalent. Thue proved that a special case of the WORD PROBLEM for finitely presented semigroups is decidable, see [98, 100], which means there is an algorithm which, given an arbitrary input, computes the decision for that input after finitely many steps. Decades later, it was shown by Emil L. Post [84] and Andrey A. Markov [70] that the aforementioned problem is undecidable in general. The former proof reduced an undecidable problem for Turing machines to the problem under consideration. These machines are of great importance for the formal language theory and were introduced by Alan M. Turing as *automatic machines* in [101] to create a device capable of computing everything that can be computed by a machine. That they really do this, is stated by the *Church-Turing Thesis*. A *Turing machine* is a device to edit the symbols on a tape consisting of a finite state control together with a read and write head which can be moved along the tape. According to the read input the Turing machine switches its state, moves its head or writes a symbol onto the tape. The inscription of the tape at the beginning and end of the computation is often referred as the input and the output of the machine, respectively. By choosing states of the device to be accepting, the Turing machine acts as a *language acceptor*. If the machine is in an accepting state after a computation, it is then said that the input is accepted. The set of all accepted input words is denoted as the *formal language accepted by the Turing machine*.

More generally, formal languages were studied by Noam Chomsky who used *formal grammars* to provide a constructive way to describe these languages. According to their complexity, Chomsky subdivided formal grammars into the types 0, 1, 2, and 3, where higher numbers indicate more restrictions on the grammar. The corresponding language families are the sets of recursively enumerable, context-sensitive, context-free, and the regular languages, respectively.

As languages can be described in multiple ways it is indispensable to have tools to classify a given language into one of these language families. A common approach to do so, is to form language classes based on different automata that accept them. To this end, certain automata are of particular importance. Due to the Church-Turing Thesis, Turing machines provide a means to formally define

different complexity classes. Pushdown automata are not as powerful as these machines but can be handled better. Nevertheless, to enable a bigger number of decidable decision problems, visibly pushdown automata are considered. However, the class of deterministic finite automata allows efficient computations. In the following, we will outline these automata and illustrate the languages they accept, which consequently defines a set of language classes.

The languages accepted by Turing machines are called the *recursively enumerable languages*. Indeed, this language class is exactly generated by Chomsky's grammars of type 0. Over the years, a lot of variants of Turing machines emerged, e.g., allowing the machine to make nondeterministic choices for its next computation step. In favor of better understandability and handiness, many restrictions evolved for these machines. Most naturally one may request the number of computational steps to be functionally dependent from the length $|w|$ of the input word w, i.e., it is bounded from above by a function $f(|w|)$. Analogously, one may demand the Turing machine to only use a bounded number $g(|w|)$ of tape cells. By requesting simple properties on the functions f and g, one obtains classes of languages accepted by Turing machines with the aforementioned restricted capabilities. The most famous classes lead to the following complexity hierarchy

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP},$$

where the prefix N indicates that the associated Turing machines are allowed to make nondeterministic computation choices. Here, L, P, and EXP are the classes of languages arising from Turing machines, where the function f is linear, polynomial, and exponential, respectively. Additionally, for the languages in the set PSPACE the respective function g is polynomial. The algorithm that is processed by the Turing machine (its instructions) is then said to be in the according class. More than a hundred[1] attempts have been made to either shrink the above hierarchy by showing two classes to be equal as well as to show that classes are unequal, e.g., that P is not equal to NP. There are many problems also of applied science, for which only an algorithm in one of these classes is known. In case of NP, there are also many examples for such problems such as process scheduling, cf. [102], or the prediction of protein folding, cf. [13, 103]. The Exponential Time Hypothesis (ETH), as well as related assumptions which are stronger than $\mathsf{P} \neq \mathsf{NP}$, emerged as a swiss-army knife, which allows for a fine-grained analysis of NP-hard problems. To name a few, it helps to gauge the inherent limits of approximability beyond polynomial time, as well as those of parameterized and exact exponential algorithms [27]. In recent years, numerous algorithmic impossibility results based on the ETH and related hypotheses have been derived [67, 105], more recently also for problems related to finite automata and regular expressions, see, e.g., [28, 33, 45]. We call a Turing machine, that uses

1 By 2016, Gerhard Woeginger collected about 116 published proof attempts for the relationship between P and NP on his website *https://wscor.win.tue.nl/woeginger/P-versus-NP.htm* .

only a linearly bounded number of cells of the tape, a *linearly bounded automaton*. These automata accept exactly the set of context-sensitive languages.

In addition, context-free languages are accepted by nondeterministic pushdown automata, which are not allowed to change the content of the tape but consist of a finite state control with a stack that can store an infinite number of symbols. The deterministic variant of this model is often used for parser design, cf. [1], and efforts have been made to use the model in DNA-computing, cf. [22, 64]. Unfortunately, many decision problems are undecidable for context-free grammars. To overcome this problem, input-driven languages, also known as visibly languages, emerged, cf. [20, 72]. The central property of these languages is the fact that each input symbol uniquely determines the type of operation performed by an according language acceptor. Hence, for the class of *visibly pushdown automata* (VPAs), which accept exactly the visibly pushdown languages, the height of their stack is solely determined by the input word. This property enables algorithms that produce VPAs accepting unions, intersections, or complements of visibly pushdown languages. Utilizing these operations, many decision problems, such as that of inclusion, equivalence, or universality, have been proven decidable. Because of these advantages, VPAs have been widely applied to tasks ranging from verification [2, 24] over XML processing [65] to learning algorithms using VPAs as surrogate models for recurrent neural networks (RNNs) [12], and they have also been studied for grammar inference [74]. The last two points are likely of particular interest as the field of machine learning is developing rapidly, and for example *surrogate models* for RNNs can help to obtain a better understanding of the properties of a RNN.

Also learning algorithms for the more simple family of regular languages (REG) have been the subject of research [8]. The simpler structure of these languages facilitates their use and analysis. For instance, in [62] Stephen C. Kleene provided regular expressions which were shown to be representations of regular languages. Regular expressions are an established and widely used tool in applied computer science. They can be found in a variety of software tools such as `awk`, `ed`, `emacs`, `grep`, `lex`, `sed`, `vi`, and many others as well. A common task in computer science, especially in bioinformatics, is to find all occurrences of the words from X in S for a given set of words X and a sequence S of symbols. This is is called *pattern matching*, and in the case where X is described by a regular expression, it is often performed by deterministic finite automata (DFAs). As already indicated, this automata model accepts exactly REG. In addition, also the subclasses of REG, namely subregular languages, were the subject of research, e.g., for finite languages L learning algorithms of deterministic cover automata were developed [58], which accept a superset of the language L.

As we have seen so far, we can classify languages by the automata which accept them. Another widely used tool for showing that particular languages are *not*, e.g., regular or context-free, is the application of pumping lemmata, which are also called iteration, intercalation, or $uvwxy$-theorems. Since the beginning

of automata and formal language theory, researchers have studied pumping and iteration properties of formal languages to gain better insights into the computational complexity and expressive power of various types of language accepting or generating mechanisms. It is well known that not all formal language families obey pumping properties as, e.g., context-sensitive or Type-0 languages. Hence, satisfying a particular pumping property provides information about the structure of the language family, and is very often used to show that a particular language does not belong to a language family in question. The first usage of pumping arguments dates back to 1960, where Stephen Scheinberg showed that the language $\{ a^n b^n c^n \mid n \geqslant 0 \}$ is not context-free [92]. Indeed, this can also be obtained by applying Bar-Hillel's lemma [11]. In fact, the literature on pumping properties is far-reaching, with very different applications beside of language classification, see, e.g., [55], where language families are defined *via* pumping properties, [94] with a focus on pumping with the additional requirement that repeating a sub-word, i.e., a consecutive part of the word, is allowed only if it is done a minimal number of times, [61], which investigates regular pumping of Turing machine languages and learnability, or [76, 82], where the relations between restarting automata, pumping patterns, and formal neural networks are studied, just to mention a few. For an annotated bibliography on pumping lemmata we refer to [78].

One variant of the pumping lemma states that for any regular language L, there exists a constant p (depending on L) such that any word w in the language of length at least p can be split into three parts $w = xyz$, where y is non-empty, and $xy^t z$ is also in the language, for every $t \geqslant 0$—see Lemma 4. By the contrapositive one can prove that certain languages are not regular. Since the aforementioned pumping lemma is only a necessary condition, it may happen that such a proof fails for a particular language such as

$$\{ a^m b^n c^n \mid m \geqslant 1 \text{ and } n \geqslant 0 \} \cup \{ b^m c^n \mid m, n \geqslant 0 \}.$$

The application of pumping lemmata is not limited to prove non-regularity. For instance, they also imply an algorithm that decides whether a regular language is finite or not. A regular language L is *infinite* if and only if there is a word of length ℓ with $p < \ell \leqslant 2p$, where p is the aforementioned constant of the pumping lemma. Here a small p narrows the interval for a witness on infiniteness. Thus, for instance, the question arises on how to determine a small or smallest value for p such that the pumping lemma is still satisfied.

For a regular language L the value of p in the aforementioned pumping lemma can always be chosen to be the number of states of a finite automaton, regardless whether it is deterministic or nondeterministic, accepting L. Consider the unary language $a^n a^*$, where all values p with $0 \leqslant p \leqslant n$ do *not* satisfy the property of the pumping lemma, but $p = n + 1$ does. A closer look on some example

languages reveals that sometimes a much smaller value suffices. For instance, consider the language

$$L = a^* + a^*bb^* + a^*bb^*aa^* + a^*bb^*aa^*bb^*,$$

given as a regular expression, where the concatenation symbol is omitted, $+$ denotes the possibility of choice between its operands, and $*$ is used for the Kleene star operation. For a set of words X the set X^* denotes the smallest superset of X which contains the empty word and is closed under concatenation. Indeed, the language L is accepted by a (minimal) deterministic finite automaton with five states, the sink state included—see Figure 1. Already for $p = 1$ the statement of

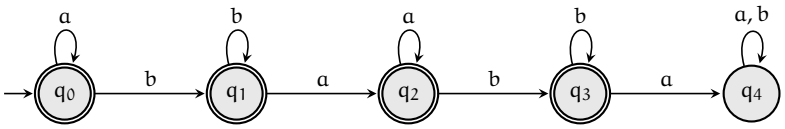

Figure 1: The minimal deterministic finite automaton A which accepts the language $L = a^* + a^*bb^* + a^*bb^*aa^* + a^*bb^*aa^*bb^*$.

the pumping lemma is satisfied. The reader may verify that regardless whether the considered word starts with a or b, this letter can be readily pumped. Thus, the minimal pumping constant satisfying the statement of the pumping lemma for the language L is 1, because the case $p = 0$ is equivalent to $L = \emptyset$ [29]. This leads to the notation of a minimal pumping constant for a language L w.r.t. a particular pumping lemma, which is the smallest number p such that the pumping lemma under consideration for the language L is satisfied.

Recently, the above-mentioned minimal pumping constants have also been investigated for teaching purposes, i.e., an instructional tool was developed which is able to calculate those constants, cf. [87]. As we have seen, pumping is a well-studied and elaborated concept in computer science, which makes it very surprising that the literature is almost completely lacking a study of the complexity of pumping, i.e., there has yet to be any research conducted on the computational complexity of pumping. A classical task of computational complexity theory is to answer the following questions for a given decision problem P:

- Is P decidable?
- If P is undecidable, are there constraints for P such that its constrained variant is decidable?
- If P is decidable, how hard is it to compute a decision for P for an arbitrary input?

We will show that the problem of determining the minimal pumping constant is rather complicated, i.e., already for linear context-free languages the problem

is undecidable, while there are even subregular language classes for which the problem is only solvable in exponential time.

Lately, Jürgen Dassow and Ismaël Jecker have made a first step towards an understanding of the complexity of pumping by investigating minimal pumping constants from the perspective of descriptional complexity in [29]. Besides basic facts on these constants for two specific pumping lemmata [19, 53, 63, 85] their relation to each other and their behavior under regularity preserving operations was studied in detail. In fact, it was proven that for three natural numbers p_1, p_2, and p_3 with $1 \leqslant p_1 \leqslant p_2 \leqslant p_3$, there is a regular language L over a growing size alphabet such that $\mathtt{mpc}(L) = p_1$, $\mathtt{mpl}(L) = p_2$, and $\mathtt{sc}(L) = p_3$, where mpc (mpl, respectively) refers to the minimal pumping constant induced by the pumping lemma from [63] (from [19, 53, 85], respectively) and sc is the abbreviation of the deterministic state complexity. This simultaneous regulation of three measures is novel in descriptional complexity theory. For the exact statements of the mentioned pumping lemmata mentioned above we refer to Lemma 3 and Lemma 4. In the case of real applications, we have to realize that these do not have growing alphabets. We will therefore improve the previously presented result by proving it for binary languages. The *operational complexity of pumping* or *pumping lemmata* for an n-ary regularity preserving operation $\circ$ undertaken in [29] is in line with other studies on the operational complexity of other measures for regular languages such as the state complexity or the accepting state complexity to mention a few. The operational complexity of pumping is the study of the set $g_\circ(k_1, k_2, \ldots, k_n)$ of all numbers k such that there are regular languages $L_1, L_2, \ldots, L_n$ with minimal pumping complexity $k_1, k_2, \ldots, k_n$, respectively, and the language $L_1 \circ L_2 \circ \cdots \circ L_n$ has minimal pumping complexity k. In [29] a complete picture for the operational complexity w.r.t. the pumping lemma from [63] (measure mpc) for the operations Kleene closure, complement, reversal, prefix and suffix-closure, circular shift, union, intersection, set-subtraction, symmetric difference, and concatenation was given—see Table 4 on page 83. However, for the pumping lemma from [19, 53, 85] (measure mpl) some results from [29] are only partial (set-subtraction and symmetric difference) and others even remained open (circular shift and intersection); for comparison see the table mentioned above. The behavior of these measures differ with respect to finiteness/infinity of ranges, due to the fact that for the pumping lemma from [19, 53, 85] the pumping has to be done within a prefix of bounded length. We address the aforementioned problems through a broader analysis of the operational complexity of minimal pumping constant, i.e., by taking more pumping lemmata and subregular language classes into account.

This thesis consists of four parts. The rest of the preamble consists of the introduction of the basic notions and the presentation of first results w.r.t. minimal pumping constants. Afterwards, in the second part we will study the descriptional complexity of minimal pumping constants. Hence, the third chapter

captures the relation between those constants and other measures, such as, the non-terminal complexity and the (non-)deterministic state complexity. This includes for example to show incomparabilities between some of the measures by providing families of languages having certain values for these measures. On the other hand we prove that for regular languages the value of specific measures has to be at least equal to values w.r.t. other properties. Thereby, we also improve on results w.r.t. the simultaneous regulation of these measures, e.g., we enhance the main result of [106] by showing that for a given alphabet Σ of size at least two and any value ρ between a specific minimal pumping constant p and $\sum_{i=0}^{p-1} |\Sigma|^i$ there is a regular language over the alphabet Σ with deterministic state complexity equal to ρ. In the fourth chapter the operational complexity of minimal pumping constants is studied. As mentioned before, this study started in [29] where an almost complete picture of the operational complexity of minimal pumping constants for two different variants of pumping lemmata from the literature was given. We continue this research by considering pumping lemmata for regular, (linear) context-free languages, and by restricting the size of the input alphabet, i.e., we study unary languages.

The third part of the thesis is dedicated to the computational complexity of the Pumping-Problem, that is, for a given finite automaton A (or a grammar G) and a value p, to decide whether the language $L(A)$ ($L(G)$, respectively) satisfies a previously fixed pumping lemma w.r.t. the value p. We approach this topic in several steps. First, in the fifth chapter of this thesis, we investigate, whether it is even possible to decide the Pumping-Problem. Therefore, we take advantage of the syntactic monoid for regular languages, which allows us to decide the problem under consideration. Afterwards, we tackle the problem for (linear) context-free languages. Here, we make use of *valid computations* of Turing machines, which can be encoded in linear context-free grammars. Building on this, we reduce the emptiness problem for Turing machines, which is undecidable to the problem in question. In addition, we determine the hardness of the aforementioned undecidable problem. Indeed, the decidability result for regular languages and the undecidability result for linear context-free languages lead to the quest of finding the biggest superset of the regular languages for which the Pumping-Problem is decidable. Hence, in the sixth chapter we study two incomparable supersets, namely the class of k-rated linear languages and the class of well-matched visibly pushdown languages. For both language families we show that the Pumping-Problem is decidable; for the first one we use an equivalence relation of finite index and for the latter the framework of Ext-algebras as introduced in [37]. The seventh chapter completes the study initiated in Chapter 5, namely we determine the hardness of the Pumping-Problem for the set of regular languages, for which the problem under consideration is decidable. To do so, we reduce two coNP-hard problems to the Pumping-Problem—one problem for each type of input automata. By assuming that the Exponential Time Hypothesis holds, these reductions imply that the Pumping-Problem cannot be

approximated within a certain factor. Since the problem under consideration is intractable for DFAs, the question arises whether there are non-trivial language classes for which it is easier to solve the Pumping-Problem. In the eighth chapter we develop a framework which can be used to show that for a subregular language class the Pumping-Problem remains coNP-complete. As an application of the framework, we prove that the Pumping-Problem is coNP-complete for pure-group languages and the set of languages which can be accepted by a planar DFA. Here, a language is called a *pure-group* language if it is accepted by a DFA for which each input symbol induces a permutation on its states. Additionally, a DFA is *planar* if its associated graph can be embedded in the plane. The fourth part shortly summarizes the findings of this thesis and gives directions for future research.