One-Week Tutorial ECI'2007 Course

Modern Trends in the Formal Language Theory

Outline

Professor Alexander Meduna, *PhD*
Brno University of Technology
Czech Republic, Europe

Prerequisites: discrete mathematics and formal language theory at an undergraduate level

I. Introduction.

We start the course by reviewing the basic material concerning the classical formal language theory, including context-free grammars and pushdown automata.

II. Modifications of Automata.

Regulated Automata. First, this class discusses *regulated automata*. Specifically, it investigates pushdown automata that regulate the use of their rules by control languages. It proves that this regulation has no effect on the power of pushdown automata if the control languages are regular. However, the pushdown automata regulated by linear control languages characterize the family of recursively enumerable languages. All these results are established in terms of acceptance by final state, acceptance by empty pushdown, and acceptance by final state and empty pushdown. In its conclusion, this lecture formulates several open problems.

Deep Pushdown Automata. Indisputably, the context-free grammars and pushdown automata, which represent their fundamental automaton counterpart, fulfill a crucial role in the formal language theory. Over its history, this theory has modified the context-free grammars in many ways, including various regulated versions of these grammars. Many of these modified context-free grammars define a language family lying between the families of context-free and context-sensitive languages. To give a specific example, an infinite hierarchy of language families between the families of context-free and context-sensitive languages was established based on n-limited state grammars, which represent regulated grammars underlain by context-free grammars. As a matter of fact, most regulated context-free grammars without erasing productions are stronger than context-free grammars but no more powerful than context-sensitive grammars. Compared to the number of grammatical modifications, there exist significantly fewer modifications of pushdown automata although the automata theory has constantly paid some attention to their investigation. Some of these modifications, such as finite-turn pushdown automata, define a proper subfamily of the family of context-free languages. On the other hand, some other modifications, such as two-pushdown automata, are as powerful as the Turing machines. As opposed to the language families generated by regulated context-free grammars without erasing productions, there are hardly any modifications of pushdown automata that define a language family between the families of context-free and context-sensitive languages. It thus comes as no surprise that most of these

modified context-free grammars, including the *n*-limited state grammars, lack any automaton counterpart.

During this lecture, we introduce deep pushdown automata, which represent a new modification of ordinary pushdown automata. However, as opposed of the previous modifications, the power of the deep pushdown automata is similar to the generative power of regulated context-free grammar without erasing productions because they are stronger than ordinary pushdown automata but less powerful than context-sensitive grammars. More precisely, these automata give rise to an infinite hierarchy of language families coinciding with the hierarchy resulting from the *n*-limited state grammars. In this sense, the deep pushdown automata represent the automaton counterpart to the state grammars and, in this sense, fill this gap.

The introduction of deep pushdown automata is inspired by the standard conversion of a context-free grammar to an equivalent pushdown automaton, *M*, frequently referred to as general top-down parser. Recall that during every move, *M* either *pops* or *expands* its pushdown depending on the symbol occurring on the pushdown top. If an input symbol, *a*, occurs on the pushdown top, *M* compares the pushdown top symbol with the current input symbol, and if they coincide, *M* pops the topmost symbol from the pushdown and proceeds to the next input symbol on the input tape. If a nonterminal occurs on the pushdown top, the parser expands its pushdown so it replaces the top nonterminal with a string. *M* accepts an input string, *x*, if it makes a sequence of moves so it completely reads *x*, empties its pushdown, and enters a final state; the latter requirement of entering a final state is dropped in some books.

A deep pushdown automaton, ${}_{deep}M$, represents a slight generalization of *M*. Indeed, ${}_{deep}M$ works exactly as *M* except that it can make *expansions of depth m* so ${}_{deep}M$ replaces the *m*th topmost pushdown symbol with a string, for some $m \geq 1$. We demonstrate that the deep pushdown automata that make expansions of depth *m* or less, where $m \geq 1$, are equivalent to the *m*-limited state grammars, so these automata accept a proper language subfamily of the language accepted by the deep pushdown automata that make expansions of depth $m + 1$ or less. The resulting infinite hierarchy of language families obtained in this way occurs between the family of context-free and context-sensitive languages. For every positive number *n*, however, there exist some context-sensitive languages that cannot be accepted by any deep pushdown automata that make expansions of depth *n* or less.

In the conclusion of this lecture, we formulate some open problem areas concerning the deep pushdown automata. Specifically, it suggests some deterministic and generalized versions of these automata to study.

III. Parallel Grammars

Indisputably, the parallel computation fulfills a crucial role in the modern computer science as a whole. Whenever investigating this computation, we face the problem of choosing its most appropriate model in order to grasp it as rigorously as possible. In the formal language theory, it is more than natural to base this model upon a suitable type of grammars.

To have the grammatical model of parallel computation simple, we surely prefer grammars based on context-free productions to those based on context-dependent productions. However, sequential grammars, such as an ordinary context-free grammars, can hardly serve as a model of this kind because they rewrite only a single symbol during a derivation step. Although purely parallel grammars, such as *L* systems, reflect the parallel computation more appropriately, this reflection is still not quite adequate from a realistic point of view. Indeed, these parallel grammars work in a completely parallel way since

they rewrite all symbols of the sentential form during a derivation step. In reality, however, parallel computation is usually performed in a partially parallel way: some parts of information are processed in parallel while the rest remains unchanged. Of course, this partially parallel computation is most appropriately formalized by partially parallel grammars, which represent a compromise between purely sequential and purely parallel grammars. That is, these grammars work in a semi-parallel context-free way so that they simultaneously rewrite some symbols during a single derivation step while leaving the other symbols unchanged. Partially parallel grammars of this kind are discussed in the lecture.

This lecture concentrates its investigation on the descriptional complexity of partially parallel grammars. Specifically, it reduces the number of some of their components, such as nonterminals or productions. It studies how to achieve this reduction without any decrease in this generative power, which coincides with the power of the Turing machines. By achieving this reduction, it actually makes the partially parallel rewriting more succinct and economical, and this economization is obviously highly appreciated both from a practical and theoretical standpoint.

More specifically, a special types of partially parallel context-free rewriting is central to this lecture—*scattered rewriting*. During a derivation step, scattered context grammars rewrite some symbols of the sentential form while leaving the others unrewritten. This lecture gives an overview of the main results concerning the descriptional complexity of these grammars with respect to the number of nonterminals or productions. In the conclusion, some open problems are pointed out.

IV. Context Grammars

In the classical formal language theory, we can divide grammatical productions into context-dependent and context-independent productions, and based on this division, we can naturally distinct context-dependent grammars, such as phrase-structure grammars, from context-independent grammars, such as context-free grammars. Making a derivation step according to context-dependent productions depends on rather strict conditions satisfied by the context surrounding the rewritten symbol while making a step according to context-independent productions does not, so from this point of view, we obviously always prefer using context-independent grammars to the others. Unfortunately, compared to context-dependent grammars, context-independent grammars are significantly less powerful; in fact, most of them are incapable to grasp some aspects of quite common programming languages. On the other hand, most context-dependent grammars are equivalent to the Turing machines, and this remarkable power represents their indisputable advantage. These pros and cons inspired the modern language theory to introducing some new grammars that simultaneously satisfy these properties—(1) they are based on context-independent productions, (2) their context conditions are significantly simpler than the strict conditions of classical context-dependent productions, and (3) they are as powerful as classical context-dependent grammars.

In this lecture, we overview the most essential types of these grammars, whose alternative context conditions can be classified into these three categories—(A) context conditions placed on derivation domains, (B) context conditions placed on the use of productions, (C) context conditions placed on the neighborhood of the rewritten symbols.
As already pointed out, we want the context conditions as small as possible. Therefore, we concentrate the investigation on the reduction of context conditions. Specifically, it reduces the number of some of their components, such as nonterminals or productions. It studies how to achieve this reduction without any decrease in this generative power, which coincides with the power of the Turing machines. By achieving this reduction, it actually makes the partially

parallel rewriting more succinct and economical, and this economization is obviously highly appreciated both from a practical and theoretical standpoint. Regarding each of the discussed grammars, we introduce and study their parallel and sequential versions, which represent two basic approaches to grammatical rewriting in today's formal language theory. That is, during a sequential derivation step, a grammar rewrites a single symbol in the current sentential form while during a parallel derivation step, a grammar rewrites all symbols. As context-free and E0L grammars represent perhaps the most fundamental sequential and parallel grammars, respectively, we usually base the discussion of sequential and parallel rewriting upon them.

V. Multigenerative Grammar Systems

The formal language theory has recently intensively investigated various grammar systems, which consist of several cooperating components, usually represented by context-free grammars. Although this variety is extremely broad, all these grammar systems always make a derivation that generates a single string. In this lecture, however, we introduce grammar systems that simultaneously generate several strings, which are subsequently composed into a single string by some common string operation, such as concatenation.

More precisely, for a positive integer n, an *n-generative grammar system* discussed in this lecture works with n context-free grammatical components in a leftmost way—that is, in every derivation step, each of these components rewrites the leftmost nonterminal occurring in its current sentential form. These n leftmost derivations are controlled by n-tuples of nonterminals or rules. Under a control like this, the grammar system generates n strings, out of which the strings that belong to the generated language are made by some basic operations. Specifically, these operations include union, concatenation and a selection of the string generated by the first component.

In this lecture, we prove that all the multigenerative grammar systems under discussion characterize the family of recursively enumerable languages. Besides this fundamental result, we give several transformation algorithms of these multigenerative grammar systems.

VI. A Combination of Grammars and Automata

In the formal language theory, the overwhelming majority of language-defining devices is based on rewriting systems that represent either grammars or automata. Grammars generate their languages while automata accept them. Consider, for instance, a context-free grammar G. G contains an alphabet of terminal symbols and an alphabet of nonterminal symbols, one of which represents the start symbol. Starting from this symbol, G rewrites nonterminal symbols in the sentential forms by its rules until it generates a string of terminals. The set of all terminal strings generated in this way is the language that G defines. To illustrate automata, consider a finite-state automaton M. M has a finite set of states, one of which is defined as the start state. In addition, some states are specified as final states. M works by making moves. During a move, it changes its current state and reads an input symbol. If with an input string, M makes a sequence of moves according to its rules so it starts from the start state, reads the input string, and reaches a final state, then M accepts the input string. The set of all strings accepted in this way represents the language that M defines. Although it is obviously quite natural to design language-defining devices based on a combination of grammars and automata and, thereby, make their scale much broader, only a tiny minority of these devices is designed in this combined way. To support this combined design, the present lecture introduces and discusses new rewriting systems, called *#-rewriting systems*, having features of both grammars and automata. Indeed, like

grammars, they are generative devices. However, like automata, they use finitely many states without any nonterminals. As its main result, this lecture characterizes the well-known infinite hierarchy of language families resulting from programmed grammars of finite index by the #-rewriting systems.

From a broader perspective, this result thus demonstrates that rewriting systems based on a combination of grammars and automata are naturally related to some classical topics and results concerning formal languages, on which they can shed light in an alternative way.

VII. Conclusion

This concluding part of the course summarizes all the material covered in the previous parts. In addition, it gives an overview of some other trends as well as expected future topics of the modern formal language theory.