Chapter 19
Applications: Overview

Abstract This chapter makes several general remarks about applications of regulated language models discussed earlier in this book. It concentrates its attention to three application areas—biology, compilers, and linguistics. The chapter consists of two sections. Section 19.1 describes applications of regulated grammars at present. Section 19.2 suggests their applications in the near future.

Key words: regulated grammars, applications, perspectives, linguistics, compilers, biology

As already stated in Chapter 1, this book is primarily and principally meant as a theoretical treatment of regulated grammars and automata. Nevertheless, to demonstrate their practical importance, we make some general remarks regarding their applications in the present two-section chapter. However, since applications of regulated grammars and automata overflow so many scientific areas, we only cover a very restricted selection of these applications. Indeed, in Section 19.1, we concentrate our attention primarily to applications related to grammar-based regulation in three application areas concerning them—linguistics, compilers, and biology. We give an insight into the current applications of regulated grammars in these three areas, which are further illustrated by quite specific case studies in Chapter 20. In Section 19.2, we make several general remarks concerning application-oriented perspectives of regulated grammars and automata in the near future, and we illustrate them by examples.

19.1 Current Applications

In the present section, we demonstrate applications of regulated grammars in linguistics, compilers, and biology. In a greater detail, we further illustrate them by case studies covered in Chapter 20.

Linguistics

In terms of English syntax, grammatical regulation can specify a number of relations between individual syntax-related elements of sentences in natural languages. For instance, relative clauses are introduced by *who* or *which* depending on the subject of the main clause. If the subject in the main clause is a person, the relative clause is introduced by *who*; otherwise, it starts by *which*. We encourage the reader to design a regulated grammar that describes this dependency (consult [8]).

In other natural languages, there exist syntax relations that can be elegantly handled by regulated grammars, too. To illustrate, in Spanish, all adjectives inflect according to gender of the noun they characterize. Both the noun and the adjective may appear at different parts of a sentence, which makes their syntactical dependency difficult to capture by classical grammars; obviously, regulated grammars can describe this dependency in a more elegant and simple way.

Apart from description, specification, and transformation of language syntax, regulated grammars can be applied to other linguistically oriented fields, such as *morphology* (see [2, 4]).

Compilers

Rather than cover a complete process of computer compilation related to grammar-based regulation, we restrict our attention primarily to *parsing* (see [1, 3, 5, 9, 11, 16]), already discussed in Sections 6.7 and 8.1 in a rather theoretical way. In addition, we briefly sketch applications in compilation phases controlled by parsing, such as syntax-directed translation.

Ordinary parsers represent crucially important components of compilers, and they are traditionally underlined by ordinary context-free grammars. As their name indicates, regulated parsers are based upon regulated context-free grammars. Considering their advantages, including properties (I) through (IV) listed next, it comes as no surprise that they become increasingly popular in modern compilers design.

(I) Regulated parsers work in a faster way than classical parsers do. Indeed, ordinary parsers control their parsing process so they consult their parsing tables during every single step. As opposed to this exhaustively busy approach, in regulated parsers, regulated grammatical mechanisms take control over the parsing process to a large extent; only during very few pre-determined steps, they consult their parsing tables to decide how to continue the parsing process under the guidance of regulating mechanism. Such a reduction of communication with the parsing tables obviously results into a significant acceleration of the parsing process as a whole.

(II) Regulated context-free grammars are much stronger than ordinary context-free grammars. Accordingly, parsers based upon regulated grammars are more powerful than their ordinary versions. As an important practical consequence, they can parse syntactical structures that cannot be parsed by ordinary parsers.

(III) Regulated parsers make use of their regulated mechanisms to perform their parsing process in a deterministic way.

(IV) Compared to ordinary parsers, regulated parsers are often written more succinctly and, therefore, readably as follows from reduction-related results concerning the number of their components, such as nonterminals and rules, achieved earlier in this book (see Sections 4.4.2, 4.6.2, 4.7.4, 6.4, 10.2.2, 10.3.2, and 10.4.2).

From a more general point of view, some fundamental parts of compilers, such as syntax-directed translators, run within the compilation process under the parser-based regulation. Furthermore, through their symbol tables, parsers also regulate exchanging various pieces of information between their components, further divided into several subcomponents. Indeed, some parts of modern compilers may be further divided into various subparts, which are run in a regulated way, and within these subparts, a similar regulation can be applied again, and so on. As a matter of fact, syntax-directed translation is frequently divided into two parts, which work concurrently. One part is guided by a precedence parser that works with expressions and conditions while the other part is guided by a predictive parser that processes the general program flow. In addition, both parts are sometimes further divided into several subprocesses or threads. Of course, this two-parser design of syntax-directed translation requires an appropriate regulation of compilation as a whole. Indeed, prior to this syntax-directed translation, a pre-parsing decomposition of the tokenized source program separates the syntax constructs for both parsers. On the other hand, after the syntax-directed translation based upon the two parsers is successfully completed, all the produced fragments of the intermediate code are carefully composed together so the resulting intermediate code is functionally equivalent to the source program. Of course, handling compilation like this requires a proper regulation of all these compilation subphases.

To give one more example in terms of modern compiler design, various optimization methods are frequently applied to the generation of the resulting target code to speed the code up as much as possible. This way of code generation may result from an explicit requirement in the source program. More often, however, modern compilers themselves recognize that a generation like this is appropriate within the given computer framework, so they generate the effective target code to speed up its subsequent execution. Whatever they do, however, they always have to guarantee that the generated target code is functionally equivalent to the source program. Clearly, this design of compilers necessitates an extremely careful control over all the optimization routines involved, and this complicated control has to be based upon a well developed theory of computational regulation. Within formal language theory, which has always provided compilation techniques with their formal models, this control can be accomplished by regulated grammars, which naturally and elegantly formalize computational regulation.

Biology

As the grammatical regulation of information processing fulfills a crucially important role in biology as a whole, it is literally impossible to cover all these applications in this scientific filed. Therefore, we restrict our attention only to microbiology, which also makes use of the systematically developed knowledge concerning these grammars significantly. Even more specifically, we narrow our attention to *molecular genetics* (see [14, 17, 18]). A regulation of information processing is central to this scientific field although it approaches this processing in a specific way. Indeed, genetically oriented studies usually investigate how to regulate the modification of several symbols within strings that represent a molecular organism. To illustrate a modification like this, consider a typical molecular organism consisting of several groups of molecules; for instance, take any organism consisting of several parts that slightly differ in behavior of DNA molecules made by specific sets of enzymes. During their development, these groups of molecules communicate with each other, and this communication usually influences the future behavior of the whole organism. A simulation of such an organism might be formally based upon regulated grammars, which can control these changes at various places. Consequently, genetic dependencies of this kind represent another challenging application area of regulated grammars in the future.

To sketch the applicability of regulated grammars in this scientific area in a greater detail, consider one-sided forbidding grammars, studied earlier in Section 6.2. These grammars can formally and elegantly simulate processing information in molecular genetics, including information concerning macromolecules, such as DNA, RNA, and polypeptides. For instance, consider an organism consisting of DNA molecules made by enzymes. It is a common phenomenon that a molecule m made by a specific enzyme can be modified unless molecules made by some other enzymes occur either to the left or to the right of m in the organism. Consider a string w that formalizes this organism so every molecule is represented by a symbol. As obvious, to simulate a change of the symbol a that represents m requires forbidding occurrences of some symbols that either precede or follow a in w. As obvious, one-sided forbidding grammars can provide a string-changing formalism that can capture this forbidding requirement in a very succinct and elegant way. To put it more generally, one-sided forbidding grammars can simulate the behavior of molecular organisms in a rigorous and uniform way. Application-oriented topics like this obviously represent a future investigation area concerning one-sided forbidding grammars.

As already stated, we give several in-depth case studies concerning linguistics, compilers, and biology in Chapter 20.

19.2 Perspectives

In the near future, highly regulated information processing is expected to intensify rapidly and significantly. Indeed, to take advantage of highly effective parallel and mutually connected computers as much as possible, a modern software product simultaneously run several processes, each of which gather, analyze and modify various elements occurring within information of an enormous size, largely spread and constantly growing across the virtually endless and limitless computer environment. During a single computational step, a particular running process selects a finite set of mutually related information elements, from which it produces new information as a whole and, thereby, completes the step. In many respects, the newly created information affects the way the process performs the next computational step, and from a more broadly perspective, it may also significantly change the way by which the other processes work as well. Clearly, a product conceptualized in this modern way requires a very sophisticated regulation of its computation performed within a single process as well as across all the processes involved.

As already explained in Chapter 1, computer science urgently needs to express regulated computation by appropriate mathematical models in order to express its fundamentals rigorously. Traditionally, formal language theory provides computer science with various automata and grammars as formal models of this kind. However, classical automata and grammars, such as ordinary finite automata or context-free grammars, represent unregulated formal models because they were introduced several decades ago when hardly any highly regulated computation based upon parallelism and distribution occurred in computer science. As an inescapable consequence, these automata and grammars fail to adequately formalize highly regulated computation. Consequently, so far, most theoretically oriented computer science areas whose investigation involve this computation simplify their investigation so they reduce their study to quite specific areas in which they work with various ad-hoc simplified models without any attempt to formally describe highly regulated computation generally and systematically. In this sense, theoretical computer science based upon unregulated formal models is endangered by approaching computation in an improper way, which does not reflect the expected regulated computation in the future at all. Simply put, rather than shed some light on fundamental ideas of this processing, this approach produces little or no relevant results concerning future computation.

Taking into account this unsatisfactory and dangerous situation occurring in the very heart of computational theory, the present book has paid an explicit attention to modifying automata and grammars so they work in a regulated way. As a result of this modification, the resulting regulated versions of grammars and automata can properly and adequately underlie a systematized theory concerning general ideas behind future regulated information processing. Out of all these regulated grammars and automata, we next select three types and demonstrate the way they can appropriately act as formal models of regulated computation. Namely, we choose

(1) scattered context grammars (see Section 4.7);

(2) regulated grammar systems (see Chapter 13);
(3) regulated pushdown automata (see Section 16.2).

(1) In general, the heart of every grammar consists of a finite set of rules, according to which the grammar derives sentences. The collection of all sentences derived by these rules forms the language generated by the grammar. Most classical grammars perform their derivation steps in a strictly sequential way. To illustrate, context-free grammars work in this way because they rewrite a single symbol of the sentential form during every single derivation step (see [7, 10, 12, 15, 19]).

As opposed to strictly sequential grammars, the notion of a scattered context grammar is based upon finitely many sequences of context-free rules that are simultaneously applied during a single derivation step. Beginning from its start symbol, the derivation process, consisting of a sequence of derivation steps, successfully ends when the derived strings contain only terminal symbols. A terminal word derived in this successful way is included into the language of this grammar, which contains all strings derived in this way. As obvious, this way of rewriting makes scattered context grammars relevant to regulated information processing as illustrated next in terms of computational linguistics.

Consider several texts such that (a) they all are written in different natural languages, but (b) they correspond to the same syntactical structure, such as the structure of basic clauses. With respect to (b), these texts are obviously closely related, yet we do not tend to compose them into a single piece of information because of (a). Suppose that a multilingual processor simultaneously modifies all these texts in their own languages so all the modified texts again correspond to the same syntactical structure, such as a modification of basic clauses to the corresponding interrogative clauses; for instance, *I said that* would be changed to *Did I say that?* in English. At this point, a processor like this needs to regulate its computation across all these modified texts in mutually different languages. As obvious, taking advantage of their simultaneous way of rewriting, scattered context grammars can handle changes of this kind while ordinary unregulated context-free grammars cannot.

(2) Classical grammar systems combine several grammars (see [6]). All the involved grammars cooperate according to some protocol during their derivations. Admittedly, compared to isolated grammars, these grammar systems show several significant advantages, including an increase of the generative power and, simultaneously, a decrease of their descriptional complexity. In essence, the classical grammar systems can be classified into cooperating distributed (CD) and parallel communicating (PC) grammar systems (see [6]). CD grammar systems work in a sequential way. Indeed, all the grammars that form components of these systems have a common sentential form, and every derivation step is performed by one of these grammars. A cooperation protocol dictates the way by which the grammars cooperate. For instance, one grammar performs precisely k derivation steps, then another grammar works in this way, and so on, for a positive integer k. In addition, some stop conditions are given to determine when the grammar systems become inactive and produce their sentences. For example, a stop condition of this kind says that no grammar of the system can make another derivation step. Many other cooperating protocols and stop conditions are considered in the literature (see [6] and

Chapter 4 in [13] for an overview). As opposed to a CD grammar system, a PC grammar system works in parallel. The PC grammatical components have their own sentential forms, and every derivation step is performed by each of the components with its sentential form. A cooperation protocol is based on a communication between the components through query symbols. More precisely, by generating these query symbols, a component specifies where to insert the sentential form produced by another component. Nevertheless, even PC grammar systems cannot control their computation across all their grammatical components simultaneously and globally.

Regulated grammar systems, discussed in Chapter 13, are based upon classical grammar systems, sketched above, because they also involve several grammatical components. However, these regulated versions can regulate their computation across all these components by finitely many sequences of nonterminals or rules while their unregulated counterparts cannot. As illustrated next, since the unregulated grammar systems fail to capture regulated information processing across all the grammatical components, they may be inapplicable under some circumstances while regulated grammar systems are applicable.

Consider regulated information processing concerning digital images. Suppose that the processor composes and transforms several fragments of these images into a single image according to its translation rules. For instance, from several digital images that specify various parts of a face, the processor produces a complete digital image of the face. Alternatively, from a huge collection of files containing various image data, the translator selects a set of images satisfying some prescribed criteria and composes them into a single image-data file. Of course, the processor makes a multi-composition like this according to some compositional rules. As obvious, a proper composition-producing process like this necessities a careful regulation of all the simultaneously applied rules, which can be elegantly accomplished by regulated grammar systems that control their computation by sequences of rules. On the other hand, a regulation like this is hardly realizable based upon unregulated grammar systems, which lack any rule-controlling mechanism.

(3) Classical pushdown automata work by making moves during which they change states (see [7, 10, 12, 15, 19]). As a result, this state mechanism is the only way by which they can control their computation. In practice, however, their applications may require a more sophisticated regulation, which cannot be accomplished by state control. Frequently, however, the regulated versions of pushdown automata (see Chapter 16) can handle computational tasks like this by their control languages, so under these circumstance, they can act as appropriate computational models while their unregulated counterparts cannot as illustrated next in terms of parsing.

Consider a collection of files, each of which contains a portion of a source program that should by parsed as a whole by a syntax analyzer, underlain by a pushdown automaton. By using a simple control language, we can prescribe the order in which the syntax analyzer should properly compose all these fragmented pieces of code stored in several different files, after which the entire code composed in this way is parsed. As obvious, we cannot prescribe any global composition like this over the collection of files by using any classical pushdown automata, which does not regulate its computation by any control language.

To summarize this section, regulated grammars and automata represent appropriate formal models of highly regulated computation, which is likely to fulfill a central role in computer science as a whole in the near future. As such, from a theoretical perspective, they will allow us to express the theoretical fundamentals of this computation rigorously and systematically. From a more pragmatic perspective, based upon them, computer science can create a well-designed methodology concerning regulated information processing. Simply put, as their main perspective in near future, regulated grammars and automata allow us to create (a) a systematized body of knowledge representing an in-depth theory of highly regulated computation and (b) a sophisticated methodology concerning regulated information processing, based upon this computation.

References

[1] Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Addison-Wesley, Boston (2006)

[2] Aronoff, M., Fudeman, K.: What is Morphology (Fundamentals of Linguistics). Wiley-Blackwell, New Jersey (2004)

[3] Bal, H., Grune, D., Jacobs, C., Langendoen, K.: Modern Compiler Design. John Wiley & Sons, Hoboken (2000)

[4] Bauer, L.: Introducing Linguistic Morphology, 2nd edn. Georgetown University Press, Washington, DC (2003)

[5] Cooper, K.D., Torczon, L.: Engineering a Compiler. Morgan Kaufmann Publishers, San Francisco (2004)

[6] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1994)

[7] Harrison, M.: Introduction to Formal Language Theory. Addison-Wesley, Boston (1978)

[8] Huddleston, R., Pullum, G.: A Student's Introduction to English Grammar. Cambridge University Press, New York (2005)

[9] Kennedy, K., Allen, J.R.: Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. Morgan Kaufmann Publishers, San Francisco (2002)

[10] Meduna, A.: Automata and Languages: Theory and Applications. Springer, London (2000)

[11] Meduna, A.: Elements of Compiler Design. Auerbach Publications, Boston (2007)

[12] Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 1: Word, Language, Grammar. Springer, New York (1997)

[13] Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 2: Linear Modeling: Background and Application. Springer, New York (1997)

[14] Russel, P.J.: iGenetics: A Molecular Approach, 3rd edn. Benjamin Cummings Publishing, San Francisco (2009)

[15] Salomaa, A.: Formal Languages. Academic Press, London (1973)

[16] Srikant, Y.N., Shankar, P.: The Compiler Design Handbook. CRC Press, London (2002)

[17] Strachan, T., Read, A.: Human Molecular Genetics, 4th edn. Garland Science, New York (2010)

[18] Watson, J.D., Baker, T.A., Bell, S.P., Gann, A., Levine, M., Losick, R.: Molecular Biology of the Gene, 6th edn. Benjamin Cummings Publishing, San Francisco (2007)

[19] Wood, D.: Theory of Computation: A Primer. Addison-Wesley, Boston (1987)