Mikhail Barash

Defining Contexts in Context-Free Grammars

Turku Centre *for* Computer Science

TUCS Dissertations
No 204, September 2015

Defining Contexts in Context-Free Grammars

Kontekstien antaminen kontekstittomille kieliopeille

Mikhail Barash

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Pub3 on September 25, 2015 at 12 noon.

University of Turku
Department of Mathematics and Statistics
FI-20014 Turku – Åbo

2015

Supervisors

Alexander Okhotin
Department of Mathematics and Statistics
University of Turku
FI-20014 Turku – Åbo
Suomi – Finland

Tero Harju
Department of Mathematics and Statistics
University of Turku
FI-20014 Turku – Åbo
Suomi – Finland

Reviewers

Andreas Maletti
Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
DE-70569 Stuttgart
Deutschland – Germany

Juha Kortelainen
Department of Information Processing Science
University of Oulu
FI-90014 Oulu – Uleåborg
Suomi – Finland

Opponent

Frank Drewes
Institutionen för datavetenskap
Umeå universitet
SE-90187 Umeå – Uumaja
Sverige – Sweden

ISBN 978-952-12-3261-9
ISSN 1239-1883

Abstract

This thesis introduces an extension of Chomsky's context-free grammars equipped with operators for referring to left and right contexts of strings. The new model is called *grammar with contexts*.

The semantics of these grammars are given in two equivalent ways — by *language equations* and by *logical deduction*, where a grammar is understood as a logic for the recursive definition of syntax. The motivation for grammars with contexts comes from an extensive example that completely defines the *syntax and static semantics* of a simple typed programming language.

Grammars with contexts maintain most important practical properties of context-free grammars, including a variant of the *Chomsky normal form*. For grammars with one-sided contexts (that is, either left or right), there is a *cubic-time tabular parsing algorithm*, applicable to an arbitrary grammar. The time complexity of this algorithm can be improved to quadratic, provided that the grammar is *unambiguous*, that is, it only allows one parse for every string it defines. A tabular parsing algorithm for grammars with two-sided contexts has fourth power time complexity. For these grammars there is a recognition algorithm that uses a *linear amount of space*.

For certain subclasses of grammars with contexts there are low-degree polynomial parsing algorithms. One of them is an extension of the classical *recursive descent* for context-free grammars; the version for grammars with contexts still works in linear time like its prototype. Another algorithm, with time complexity varying from linear to cubic depending on the particular grammar, adapts *deterministic LR parsing* to the new model.

If all context operators in a grammar define regular languages, then such a grammar can be transformed to an equivalent grammar without context operators at all. This allows one to represent the syntax of languages in a more succinct way by utilizing context specifications.

Linear grammars with contexts turned out to be non-trivial already over a one-letter alphabet. This fact leads to some *undecidability results* for this family of grammars.

Tiivistelmä suomeksi

Tässä väitöskirjassa esitellään Chomskyn kontekstittomien kielioppien laajennus, jossa on vasemmille ja oikeille konteksteille viittaavia operaattoreita. Uutta kielioppiperhettä kutsutaan *konteksteilla varustetuiksi kieliopeiksi*.

Näiden kielioppien semantiikka määritellään kahdella ekvivalentilla tavalla — *kieliyhtälöin* sekä *loogisin päättelyin*, jossa kielioppi katsotaan syntaksin rekursiivisen määritelmän logiikaksi. Konteksteilla varustettujen kielioppien motivaationa on laaja esimerkki, jossa yksinkertaisen ohjelmointikielen *syntaksi ja staattinen semantiikka* määritellään täysin.

Konteksteilla varustetut kieliopit omaavat monia kontekstittomien kielioppien käytännön ominaisuuksia, kuten *Chomskyn normaalimuodon* variaation. Yksipuolisilla konteksteilla varustetut kieliopit (toisin sanoen kontekstit ovat joko vasemmanpuoleisia tai oikeanpuoleisia) voidaan jäsentää *taulukkointiin perustuvalla jäsennysalgoritmilla*, jonka aikakompleksisuus on kuutiollinen, ja joka on sovellettavissa mielivaltaisten kielioppien analyysiin. Tämän algoritmin aikakompleksisuus paranee neliölliseksi, mikäli kielioppi on *yksiselitteinen*, toisin sanoen, jos jokainen kieliopin määrittelemä merkkijono on jäsennettävissä vain yhdellä tavalla. Kaksipuolisilla konteksteilla varustettujen kielioppien jäsennysalgoritmilla on neljännen potenssin aikakompleksisuus. Kaikki nämä kielioppiperheet voidaan tunnistaa käyttäen *lineaarista määrää tiloja*.

Konteksteilla varustettujen kielioppien tietyt alaperheet voidaan jäsentää algoritmeilla, jotka toimivat alhaista astetta olevassa polynomiajassa. Yksi näistä algoritmeista on klassisen *rekursiivisen laskeutumisen* laajennus, joka toimii lineaarisessa ajassa, kuten alkuperäinen algoritmikin. Toinen algoritmi, jonka aikakompleksisuus vaihtelee lineaarisesta kuutiolliseen kieliopista riippuen, mukauttaa *deterministisen LR-jäsennysalgoritmin* uudelle kielioppiperheelle.

Mikäli jokainen konteksteilla varustetun kieliopin kontekstioperaattori määrittelee säännöllisen kielen, voidaan kyseinen kielioppi muuttaa ekvivalentiksi kontekstittomaksi kieliopiksi. Tämän avulla voidaan määritellä kielten syntaksi ytimekkäämmin hyödyntämällä kontekstioperaattorien ilmaisuvoimaa.

Lineaariset konteksteilla varustetut kieliopit osoittautuivat epätriviaaleiksi jo kun aakkosto koostuu vain yhdestä merkistä. Tämä johtaa muutaman näitä kielioppeja koskevan *ratkeamattomuustuloksen* osoittamiseen.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisors, Dr. Alexander Okhotin and Professor Tero Harju for their guidance and support. I am thankful to Alexander for fruitful discussions we had, for numerous comments on the papers and the present thesis. Thank you for *always* being available and willing to help. Not in the last instance I would like to thank you for the great sense of humor, which made our discussions so fun.

I am grateful to Dr. Andreas Maletti (University of Stuttgart) and Dr. Juha Kortelainen (University of Oulu) for agreeing to review almost the quarter of a thousand pages of my dissertation and for many constructive comments on it, that helped me improve the thesis.

I would like to thank Professor Frank Drewes (Umeå University) for agreeing to act as my opponent.

I am thankful to my research director, Professor Juhani Karhumäki, for creating a warm and creative atmosphere at the department and for his support of my studies.

I am very grateful to Professor Ion Petre (Åbo Akademi University) for his encouragement and support. I am also thankful for introducing me to the field of computational biology and for many interesting seminars we had.

I am thankful to Dr. Anssi Yli-Jyrä (University of Helsinki) for discussions we had on computational linguistics.

I am thankful to Professor Jarkko Kari for his excellent lectures on automata and formal languages and his support of my studies.

I am grateful to my office mates, Jarkko Peltomäki, Dr. Tommi Lehtinen, Reino Niskanen, Jetro Vesti, and fellow students Markus Whiteland and Michal Szabados, for all the nice discussions and meetings we had.

Special thanks are to Markus Whiteland and Professor Jarkko Kari for proofreading my translation of the abstract of this thesis into the Finnish language, and for tremendous help in correcting mistakes in the translation.

It has been a great honour for me to pursue my doctoral education at the Department of Mathematics and Statistics of University of Turku. I am very grateful to all members of the department for creating a very friendly and homely atmosphere. Special thanks are to Dr. Eija Jurvanen, Dr. Arto

Lepistö, Dr. Eeva Vehkalahti et, bien sûr, Dr. Roope Vehkalahti. (Bonjour!)

I am also thankful to the secretaries of the department, Sonja Vanto, Tuire Huuskonen, and Laura Kullas, for their encouragement, support and assistance in everyday matters.

I am very grateful to Turku Centre of Computer Science (TUCS) and University of Turku Graduate School (UTUGS) for their generous financial support of my studies. I would also like to thank the staff of Turku Centre for Computer Science, especially Irmeli Laine, Outi Tuohi, and Tomi Suovuo, for their assistance and support.

I am thankful to all members of Computational Biology Laboratory of Åbo Akademi University for interesting seminars we had. Special thanks are to Diana-Elena Gratie, Dr. Cristian Gratie, Dr. Vladimir Rogojin, Dr. Bogdan Iancu, and Charmi Panchal for creating a nice and friendly atmosphere.

I am also grateful to Dr. Yuliya Prokhorova, Dr. Sergey Ostroumov, and Inna Pereverzeva for their encouragement, especially at the early stage of my studies.

I am very grateful to Professor Nadezhda Vitkovskaya and Professor Vladimir Kobychev (Irkutsk State University) for their tremendous support and encouragement, and for their assistance in making my education in Finland a reality.

I am thankful to Dr. Viktor Kurganskii, Dr. Alexander Semenov, Dr. Irina Zakharova, Professor Vladimir Srochko, Professor Mikhail Falaleev, and Dr. Margarita Argutchintseva (Irkutsk State University) for their interesting lectures and support during my studies.

I am extremely grateful to the teacher of English Larisa Fedorova, la professeure de français Larisa Chechetkina, die Lehrerin für die deutsche Sprache Inna Zlydneva, suomen kielen opettaja Kaarina Valtanen, svensklärarna Charlotta Sandberg (Åbo Akademi University) och Ekaterina Grishkevich, for giving me most valuable knowledge—the ability to communicate in foreign languages. Thank you! Merci! Danke! Kiitos! Tack!

I would like to express my gratitude to all my friends who always supported and helped me. In particular, I am thankful to my best friend Michael Tzukanov and (in alphabetical order) Shamil Alifov, Gennady Kurushin, Adham Maharramli, Olga Mironenko, and Maria Vahtila. Special thanks are to Aleksi Vähäpassi and Maria Vahtila for explaining me all the fanciful constructions of the Finnish language.

Last but not least, I am very grateful to my family for their all-embracing support and their unconditional love.

Turku, 25th August 2015
Suomen Turussa, 25. elokuuta 2015 *Mikhail Barash*
Åbo, den 25 augusti 2015

List of original publications

1. **Mikhail Barash**, Alexander Okhotin. "Defining contexts in context-free grammars", *Language and Automata Theory and Applications* (LATA 2012, A Coruña, Spain, 5–9 March 2012), LNCS **7183** (2012), 106–118.

2. Mikhail Barash. "Recursive descent parsing for grammars with contexts", *Current Trends in Theory and Practice of Computer Science* (SOFSEM 2013, Špindlerův Mlýn, Czech Republic, 26–31 January 2013), Local Proceedings II (2013), 10–21.

3. Mikhail Barash. "Programming language specification by a grammar with contexts", *Non-classical Models of Automata and Applications* (NCMA 2013, Umeå, Sweden, 13–14 August 2013), Österreichische Computer Gesellschaft **294** (2013), 51–67.

4. **Mikhail Barash**, Alexander Okhotin. "Linear grammars with one-sided contexts and their automaton representation", *LATIN 2014: Theoretical Informatics* (Montevideo, Uruguay, 31 March – 4 April 2014), LNCS **8392** (2014), 190–201.

5. **Mikhail Barash**, Alexander Okhotin. "Grammars with two-sided contexts", *Automata and Formal Languages* (AFL 2014, Szeged, Hungary, 27–29 May 2014), EPTCS **151** (2014), 94–108.

6. **Mikhail Barash**, Alexander Okhotin. "An extension of context-free grammars with one-sided context specifications", *Information and Computation* **237** (2014), 56–94.

7. **Mikhail Barash**, Alexander Okhotin. "Generalized LR parsing for grammars with contexts", *Computer Science in Russia* (CSR 2015, Listvyanka, Lake Baikal, Russia, 13–17 July 2015), LNCS **9139** (2015), 67–79.

8. **Mikhail Barash**, Alexander Okhotin. "Two-sided context specifications in formal grammars", *Theoretical Computer Science* **591** (2015), 134–153.

9. **Mikhail Barash**, Alexander Okhotin. "Linear grammars with one-sided contexts and their automaton representation", *RAIRO Theoretical Informatics and Applications* **49**:2 (2015), 153–178.

10. **Mikhail Barash**, Alexander Okhotin. "Grammars with regular context specifications", manuscript.

Contents

Chapter 1

Introduction

Context-free grammars are best understood as a logic for defining the syntax of languages. In this logic, definitions are inductive, so that the properties of a string are determined by the properties of its substrings. This is how a rule $S \to aSb$ asserts that if a string $a^{n-1}b^{n-1}$ has the property S, then the string $a^n b^n$ has the property S as well. This is demonstrated in the following simplest context-free grammar.

$$\begin{aligned} S &\to aSb \\ S &\to \varepsilon \end{aligned}$$

The basis of the induction is given by the rule $S \to \varepsilon$ defining the empty string.

Besides the concatenation, the formalism of this logic includes a disjunction operation, represented by having multiple rules for a single symbol. One can further augment this logic with conjunction and negation operations, leading to *conjunctive grammars* [50] and *Boolean grammars* [56]. These grammars preserve the main idea of the context-free grammars (that of defining syntax inductively, as described above), maintain most of their practically important features, such as efficient parsing algorithms [56, 60, 61, 70, 53, 55], and have been a subject of diverse research [3, 16, 31, 32, 44, 64, 65, 66, 67]. Conjunctive and Boolean grammars are "context-free" in the general sense of the word, as they define properties of each substring independently of the context, in which it occurs, and thus Boolean grammars constitute a natural general case of context-free grammars. Ordinary context-free grammars can be viewed as their disjunctive fragment.

When Chomsky [9] introduced the term "context-free grammar" for an intuitively obvious model of syntax, he had a further idea of a more powerful model, in which one could define rules applicable only in some particular contexts [9, p. 142]. Chomsky's own attempt to implement this idea by string

rewriting resulted in a model equivalent to linear-space Turing machines, in which the "nonterminal symbols," meant to represent syntactic categories, could be freely manipulated as tape symbols. Even though the resulting devices are still known under the name of "context-sensitive grammars," they have nothing to do with the syntax of languages, and, in particular, they fail to implement Chomsky's original idea of a phrase-structure rule applicable in a context.

Even though Chomsky's terminology for formal grammars, such as the term "context-free," was generally accepted by the research community, the actual idea of a rule applicable in a context was never investigated again. None of the successful extensions of context-free grammars, such as tree-adjoining grammars [35] or multi-component grammars [36, 81], allow expressing any conditions on the contexts—in spite of the nickname "mildly context-sensitive" [36, 92]. It should also be noted that both tree-adjoining grammars and multi-component grammars define the properties of strings inductively on their length, and have nothing to do with Chomsky's "context-sensitive" string rewriting. Thus, the theory of formal grammars beyond ordinary context-free has developed in a different direction than initially pointed to by Chomsky. Nevertheless, the general idea of context-sensitive rules in formal grammars remains interesting and deserves investigation.

The present thesis undertakes to reconsider Chomsky's [9] idea of contexts in grammars, this time using the more appropriate tools of deduction systems and language equations. The concept of a formal grammar as a logic and its semantics as logical inference was first presented in a monograph by Kowalski [45, Ch. 3], and then elaborated by Pereira and Warren [73]. Later, Rounds [79] used first-order logic over positions in a string augmented with a fixpoint operator, FO(LFP), to represent formal grammars as formulae of this logic. What is particularly important about this representation, is that all the aforementioned successful extensions of the context-free grammars, such as tree-adjoining grammars and multi-component grammars, are not only expressible in this logic, but the way they are expressed represents these grammars exactly as they are intuitively understood.

The new model proposed in this thesis are ***grammars with contexts***, which are based on conjunctive grammars and introduce four special operators for referring to the context of a substring being defined. The *left context operator* refers to the "past" of the current substring: an expression $\lhd\alpha$ defines any substring that is directly preceded by a prefix of the form α. This operator is meant to be used together with the usual specifications of the structure of the current substring, using conjunction to combine several specifications. For example, consider the following grammar.

$$
\begin{aligned}
A &\rightarrow BC \,\&\, \lhd D \\
B &\rightarrow b \\
C &\rightarrow c \\
D &\rightarrow d
\end{aligned}
$$

Its first rule represents any substring of the form BC preceded by a substring of the form D. That is, it specifies that a substring bc of a string $w = dbc\ldots$ has the property A; however, this rule will not produce the same substring occurring in the strings $w' = abc$ or $w'' = adbc$. The other *extended left context operator* $\trianglelefteqslant\alpha$ represents the form of the left context of the current substring concatenated with the substring itself, so that the rules

$$
\begin{aligned}
A &\rightarrow B \,\&\, \trianglelefteqslant E \\
B &\rightarrow b \\
E &\rightarrow ab
\end{aligned}
$$

state that the substring b occurring in the string $w = ab$ has the property A.

One can symmetrically define operators for right contexts $\rhd\alpha$ and extended right contexts $\trianglerighteqslant\alpha$ (which define the form of the current substring concatenated with its right context). Then a grammar may contain such rules as $A \rightarrow BC \,\&\, \lhd D \,\&\, \rhd E$, defining any substring of the form BC preceded by a substring of the form D and followed by a substring of the form E. Consider a grammar with the following rules.

$$
\begin{aligned}
A &\rightarrow BC \,\&\, \lhd D \,\&\, \rhd E \\
B &\rightarrow b \\
C &\rightarrow c \\
D &\rightarrow d \\
E &\rightarrow e
\end{aligned}
$$

Here the rule for A asserts that a substring bc of a string $w = dbce$ has the property A. However, this rule will not produce the same substring bc occurring in another string $w' = dbcd$, because its right context does not satisfy the condition $\rhd E$.

Figure 1.1 illustrates how context operators refer to a substring and its left and right contexts. Note that left context operators ($\lhd$, $\trianglelefteqslant$) are defined to refer to the whole left context, which begins with the first symbol of the entire string, and similarly the right context ($\trianglerighteqslant$, $\rhd$) ends at the last symbol of the entire string. Intuitively, by a context one would often mean the last few symbols preceding the substring or the next few symbols following a

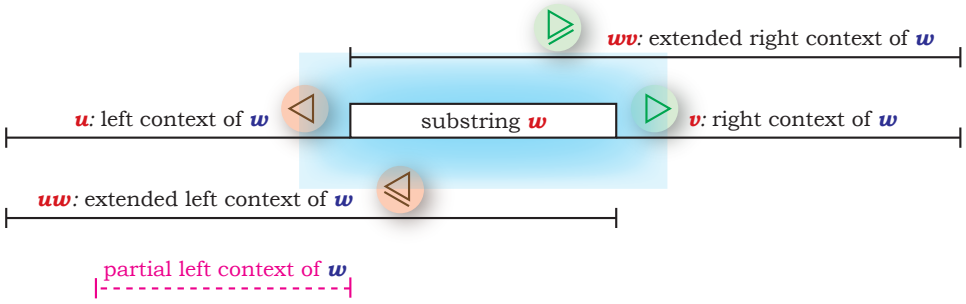


Figure 1.1: A substring w with a left context u and a right context v, and context operators applied to it.

substring, and at first glance, context operators could have been defined in that way. However, those "partial contexts" can be expressed using the proposed operators as $\lhd\Sigma^*\beta$ and $\rhd\delta\Sigma^*$, respectively. Conversely, the whole left context can be simulated by a partial left context, provided that the entire string begins with a special marker symbol: if a partial context beginning with that symbol is requested, then it can only be the whole left context. Thus, a partial left context operator would be less convenient to use, as well as not any easier to implement than the proposed operator for the whole left context. This makes the above definition as expressive as its alternative, and more convenient to handle.

In this work, the above intuitive definition of grammars with contexts is formalized by logical deduction. For grammars with left contexts, the deduction uses *elementary propositions* of the form

$$A\big(u\langle w\rangle\big),$$

where $u\langle w\rangle$ denotes a substring w in left context u (that is, occurring in a string uwv) and A is a syntactic property defined by the grammar ("nonterminal symbol" in Chomsky's terminology); this proposition asserts that w has the property A in the context u. Then, each rule of the grammar, which is of the general form $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_n$, becomes a deduction scheme for inferring elementary propositions of the form $A\big(u\langle v\rangle\big)$ from each other, and the language generated by the grammar is ultimately defined as the set of all such strings w that $S\big(\varepsilon\langle w\rangle\big)$ can be deduced. A standard proof tree of such a deduction constitutes a parse tree of the string w. This definition generalizes the representation of ordinary context-free grammars by deduction—assumed, for instance, by Pereira and Warren [73] and Sikkel [83]—as well as the extension of this representation to conjunctive grammars [59].

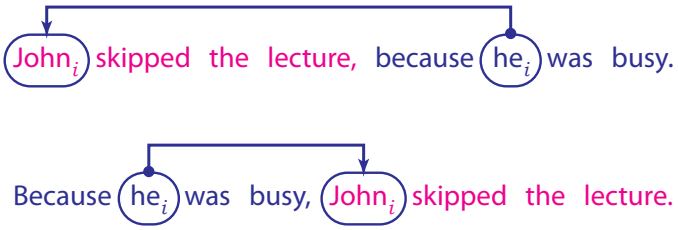


Figure 1.2: How left and right contexts can be used to check that a pronoun refers to a noun, informally illustrated.

For grammars with two-sided contexts, the definition is formalized by deductions of propositions of the form

$$A\big(u\langle w\rangle v\big),$$

which state that the substring w occurring in the context between u and v has the property A, where A is a syntactic category defined by the grammar. Again, each rule of the grammar becomes a schema for deduction rules, and a string w is generated by the grammar, if there is a proof of the proposition $S\big(\varepsilon\langle w\rangle\varepsilon\big)$.

There is an alternative, equivalent definition of grammars with contexts. It uses a generalization of language equations, in which the unknowns are *sets of strings with contexts* ($u\langle w\rangle$ and $u\langle w\rangle v$). All connectives in the rules of a grammar—that is, concatenation, disjunction, conjunction and both context operators—are then interpreted as operations on such sets, and the resulting system of equations is proved to have a least fixpoint, as in the known cases of ordinary context-free grammars [20] and conjunctive grammars [51]. This least solution defines the language generated by the grammar.

These definitions ensure that the proposed grammars with contexts define the properties of strings inductively from the properties of their substrings and of the contexts, in which these substrings occur. Nothing like bit manipulation in "sentential forms" is possible in the proposed model, and hence the model avoids falling into the same pit as Chomsky's "context-sensitive grammars," that of being able to simulate computations of space-bounded Turing machines.

The purpose of the present thesis is to settle the basic properties of grammars with contexts, and thus to justify that *these grammars make sense as a model of syntax.*

The definitions of grammars with contexts and their semantics are given in **Chapter 2**, which also presents basic examples of grammars. These examples model several types of cross-references, such as anaphoric constructions

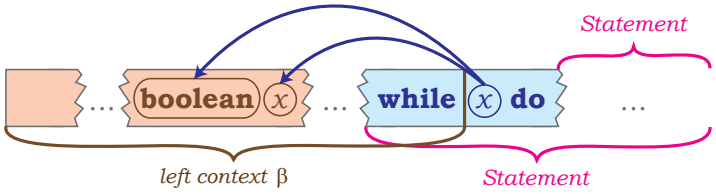


Figure 1.3: An informal illustration of how a grammar with contexts can check that an identifier is declared and has a matching type.

in natural languages [22], as shown informally in Figure 1.2, or declaration of identifiers before or after their use, informally illustrated in Figure 1.3. Ideas of these examples are then used in **Chapter 3**, where the complete syntax of a simple typed programming language is defined by a grammar with one-sided contexts.

Chapter 4 then proceeds with developing a normal form for grammars with contexts, which generalizes the Chomsky normal form for ordinary context-free grammars. In the normal form, every rule is a conjunction of one or more *base conjuncts* describing the form of the current substring (either as a concatenation of the form BC or as a single symbol a), with any context specifications ($\lhd D$, $\trianglelefteq E$, $\trianglerighteq F$, $\rhd H$). The transformation to the normal form proceeds in three steps. First, all rules generating the empty string in any contexts are eliminated. Second, all rules with an explicit empty context specification ($\lhd\varepsilon$, $\rhd\varepsilon$) are also eliminated. The final step is elimination of the rules of the form $A \to \ldots \,\&\, B$, where the dependency of A on B potentially causes cycles in the definition.

This normal form is used to extend the basic Cocke–Kasami–Younger parsing algorithm to grammars with contexts. The parsing algorithm for grammars with one-sided contexts works in time $\mathcal{O}(n^3)$, where n is the length of the input string. A variant of this algorithm works in time $\mathcal{O}(n^2)$ for unambiguous grammars with one-sided contexts; this algorithm extends a similar algorithm for unambiguous Boolean grammars [62], which is in turn based upon the algorithm of Kasami and Torii [37]. The extension of the Cocke–Kasami–Younger algorithm for grammars with two-sided contexts works in time $\mathcal{O}(n^4)$.

It is also demonstrated that every language defined by a grammar with contexts can be recognized in deterministic linear space.

In **Chapter 5**, the *Generalized LR* (GLR) parsing algorithm [47, 88, 89] is extended to handle the case of grammars with left contexts. The original GLR algorithm is an extension of one of the most well-known and widely used parsing methods, LR parsing [40]. This extension is applicable to the whole class of context-free grammars: wherever an LR(k) parser would not be able to proceed deterministically, a Generalized LR parser simulates all available

actions and stores the results in a graph-structured stack. The extension of this algorithm to the case of grammars with one-sided contexts has to check multiple conditions for a reduction operation, some of them referring to the context of the current substring. The conditions are represented as paths in a graph-structured stack, and accordingly require table-driven graph searching techniques. In spite of these complications, a direct implementation of the algorithm works in time $\mathcal{O}(n^4)$, whereas a more careful implementation leads to a $\mathcal{O}(n^3)$ upper bound on its running time. On "good" grammars, the running time can be as low as linear. This algorithm becomes the first sign of possible practical implementation of grammars with contexts.

In the literature, ideas of (extended) right contexts of the form $\triangleright\alpha\Sigma^*$ (in the terminology of this thesis) have occasionally arisen in connection with parsing; those contexts are considered as "lookahead strings" and are used to guide a deterministic parser. If α represents a regular language, these simple forms of contexts occur in LR-regular [11], LL-regular [29] and LL(*) [72] parsers. Some software tools for engineering parsers, such as those developed by Parr and Fischer [72] and by Ford [18], allow specifying contexts $\triangleright\alpha\Sigma^*$, with α defined within the grammar, and such specifications can be used by a programmer for *ad hoc* adjustment of the behaviour of a deterministic recursive descent parser. **Chapter 6** describes a variant of recursive descent parsing, extended to handle the case of grammars with contexts, additionally allowing a limited backtracking. The conjunction operation is implemented by scanning a single substring multiple times, as in the case of conjunctive grammars [61]. Possible rules are tested one by one in a given order; the applicability of the rule is determined by checking the context specifications suitable during the parsing. A direct implementation of the algorithm may use exponential time on some extreme grammars, however, using the memoization technique guarantees the linear time complexity of the parser.

A special case when all context operators in a grammar define regular languages is discussed in **Chapter 7**. A *grammar with regular contexts* is equipped with a finite automaton and the context operators in grammar rules refer to its states: if the automaton accepts from a given state, then the corresponding context operator is considered satisfied. It turns out that regular context operators can be effectively eliminated from every such grammar. This means, in particular, that every context-free grammar which additionally has regular context operators can be in fact converted to an ordinary grammar.

Chapter 8 investigates the *linear subclass* of grammars with one-sided contexts, where linearity is understood as a restriction to concatenate nonterminal symbols only to terminal strings. Linear grammars with one-sided contexts are proved to be computationally equivalent to a special kind of cellular automata. Such an automaton, processing an input string over a

one-symbol alphabet, turns out to be able to simulate a fixed Turing machine. This subsequently allows uniform undecidability proofs for linear grammars with contexts. Some further examples of linear grammars with contexts are also constructed.

Possible directions for future work on grammars with contexts are discussed in **Chapter 9**, which concludes the present thesis.

Part I

Definitions and examples

Chapter 2

Grammars with contexts

Formal grammars deal with finite strings over a finite alphabet Σ. The set of all such strings is denoted by Σ^*. The length of a string $w = a_1 \dots a_\ell$, with $a_i \in \Sigma$, is the number of symbols in it, denoted by $|w| = \ell$. The unique string of length 0 is known as the *empty string* and denoted by ε. The concatenation of two strings $u, v \in \Sigma^*$ is the string $u \cdot v = uv$. For all strings $u, v, w \in \Sigma^*$, consider their concatenation uvw: then the string v is known as a *substring* of uvw, the string u is its *left context* and w is its *right context*.

Any subset of Σ^* is a *(formal) language* over Σ. Since languages are sets, all Boolean operations on sets apply to them. In addition, for any two languages $K, L \subseteq \Sigma^*$, their concatenation is $KL = \{\, uv \mid u \in K,\ v \in L \,\}$. The concatenation of arbitrarily many strings from the same language $L \subseteq \Sigma^*$, defined by $L^* = \{\, w_1 \dots w_\ell \mid \ell \geqslant 0,\ w_i \in L \,\}$, is known as the *Kleene star* of a language, or simply the *star*.

A formal grammar defines a language, viewed as a set of syntactically correct strings, by specifying their syntactic structure. The model introduced in this chapter, *grammars with contexts*, uses concatenation, conjunction and disjunction, as well as the new *context operators* for left contexts ($\lhd$, $\trianglelefteq$) and for right contexts ($\rhd$, $\trianglerighteq$) to specify the syntax of languages.

2.1 Grammars with one-sided contexts

Let us consider *left contexts of strings*: a substring v occurring in a left context u shall be denoted by $u\langle v \rangle$. Formally, this is a pair of a string and its left context. It shall occasionally be referred to as simply a string, so that a reference to "a string $u\langle v \rangle$" assumes a substring occurring in a given context.

In this section, grammars with left contexts are formalized by a deduction of elementary propositions of the form $A\big(u\langle v \rangle\big)$, where A is a syntactic

property defined by the grammar ("nonterminal symbol" in Chomsky's terminology); this proposition asserts that v has the property A in the context u. Though left contexts are assumed below, all results symmetrically hold for grammars with right contexts.

Definition 2.1. *A grammar with left contexts is a quadruple* $G = (\Sigma, N, R, S)$*, where*

- Σ *is the alphabet of the language being defined;*
- N *is a finite set of auxiliary symbols ("nonterminal symbols" in Chomsky's terminology), disjoint with* Σ*, which denote the properties of strings defined by the grammar;*
- R *is a finite set of grammar rules, each of the form*

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteqslant\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteqslant\gamma_n, \qquad (2.1)$$

  with $A \in N$*,* $k \geqslant 1$*,* $m, n \geqslant 0$*,* $\alpha_i, \beta_i, \gamma_i \in (\Sigma \cup N)^*$*;*
- $S \in N$ *is a symbol representing syntactically well-formed sentences of the language (in the common jargon, "the start symbol").*

For each grammar rule (2.1), each term α_i, $\lhd\beta_i$ and $\trianglelefteqslant\gamma_i$ is called a *conjunct*. Each *base conjunct* α_i gives a representation of the substring being defined as a concatenation of shorter substrings. A conjunct $\lhd\beta_i$ similarly describes the form of the *left context* or the *past* of the substring being defined. Conjuncts of the form $\trianglelefteqslant\gamma_i$ refer to the form of the left context and the current substring, concatenated into a single string.

A grammar with left contexts degenerates to a conjunctive grammar [50] if no context operators are ever used, that is, if $m = n = 0$ for every rule (2.1); and further to an ordinary context-free grammar if conjunction is never used, that is, if $k = 1$ in every rule.

Intuitively, a rule (2.1) can be read as follows: a substring v occurring in a left context u has the property A, if

- for each base conjunct $\alpha_i = X_1 \ldots X_\ell$ in the rule, with $X_1, \ldots, X_\ell \in \Sigma \cup N$, the string v is representable as a concatenation $X_1 \ldots X_\ell$, that is, $v = v_1 \ldots v_\ell$, where each substring v_i has the property X_i, *and*
- for each conjunct $\lhd\beta_i$, the left context u is similarly representable as β_i, *and*
- for each conjunct $\trianglelefteqslant\gamma_i$, the string uv is representable as γ_i.

As in an ordinary context-free grammar, multiple rules $A \to \Phi_1, \ldots, A \to \Phi_\ell$ for a single symbol A represent logical disjunction of conditions, and are denoted by $A \to \Phi_1 \mid \ldots \mid \Phi_\ell$.

The semantics of grammars with contexts are formally defined by a deduction system, which deals with elementary propositions of the form "a string $v \in \Sigma^*$ written in a left context $u \in \Sigma^*$ has the property $X \in \Sigma \cup N$", denoted by $X\big(u\langle v\rangle\big)$. The deduction begins with axioms, such as that any symbol $a \in \Sigma$ written in any context has the property a, which is denoted by $a\big(x\langle a\rangle\big)$ for all $x \in \Sigma^*$. Each rule in R is then regarded as a schema for deduction rules. For example, a rule $A \to BC$ allows making deductions of the form

$$B\big(u\langle v\rangle\big), C\big(uv\langle w\rangle\big) \vdash_G A\big(u\langle vw\rangle\big) \quad \text{(for all } u, v, w \in \Sigma^*\text{)},$$

which is essentially a concatenation of v and w that respects the left contexts. If the rule is of the form $A \to BC \,\&\, \lhd D$, then this deduction requires an extra premise ensuring that the left context u has the property D.

$$B\big(u\langle v\rangle\big), C\big(uv\langle w\rangle\big), D\big(\varepsilon\langle u\rangle\big) \vdash_G A\big(u\langle vw\rangle\big) \quad \text{(for all } u, v, w \in \Sigma^*\text{)}$$

And if the rule is $A \to BC \,\&\, \lhd\!\!\!=\, E$, the deduction proceeds as follows.

$$B\big(u\langle v\rangle\big), C\big(uv\langle w\rangle\big), E\big(\varepsilon\langle uvw\rangle\big) \vdash_G A\big(u\langle vw\rangle\big) \quad \text{(for all } u, v, w \in \Sigma^*\text{)}$$

The general form of the deduction schemata induced by rules in R is defined below.

Definition 2.2. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, and define the following deduction system on propositions of the form $X\big(u\langle v\rangle\big)$, with $X \in \Sigma \cup N$ and $u, v \in \Sigma^*$. There is a single axiom scheme:*

$$\vdash_G a\big(x\langle a\rangle\big) \quad \text{(for all } a \in \Sigma \text{ and } x \in \Sigma^*\text{)}.$$

Each rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \lhd\!\!\!=\gamma_1 \,\&\, \ldots \,\&\, \lhd\!\!\!=\gamma_n$ in the grammar defines the following scheme for deduction rules:

$$I \vdash_G A\big(u\langle v\rangle\big),$$

for all $u, v \in \Sigma^*$ and for every set I of propositions satisfying the properties below:

- *for every base conjunct $\alpha_i = X_1 \ldots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $v = v_1 \ldots v_\ell$ with $X_j\big(uv_1 \ldots v_{j-1}\langle v_j\rangle\big) \in I$ for all $j \in \{1, \ldots, \ell\}$;*

- *for every conjunct* $\lhd\beta_i = \lhd X_1 \dots X_\ell$ *with* $\ell \geqslant 0$ *and* $X_j \in \Sigma \cup N$*, there should be such a partition* $u = u_1 \dots u_\ell$*, that* $X_j\big(u_1 \dots u_{j-1}\langle u_j\rangle\big) \in I$ *for all* $j \in \{1, \dots, \ell\}$*;*

- *every conjunct* $\trianglelefteqslant\gamma_i = \trianglelefteqslant X_1 \dots X_\ell$ *with* $\ell \geqslant 0$ *and* $X_j \in \Sigma \cup N$ *should have a corresponding partition* $uv = w_1 \dots w_\ell$ *with* $X_j\big(w_1 \dots w_{j-1}\langle w_j\rangle\big) \in I$ *for all* $j \in \{1, \dots, \ell\}$*.*

Note that if $\alpha_i = \varepsilon$ *in the first case, that is, if* $\ell = 0$*, then the given condition implies* $v = \varepsilon$*, and deduction is possible only if* $v = \varepsilon$*; similarly,* $\beta_i = \varepsilon$ *implies* $u = \varepsilon$*, and* $\gamma_i = \varepsilon$ *implies* $u = v = \varepsilon$*.*

Then the language generated by a symbol $A \in N$ *is defined as the set of all strings with contexts* $u\langle v\rangle$*, for which the proposition* $A\big(u\langle v\rangle\big)$ *can be deduced from the axioms in one or more steps:*

$$L_G(A) = \{\, u\langle v\rangle \mid u, v \in \Sigma^*, \vdash_G A\big(u\langle v\rangle\big) \,\}.$$

The language generated by the grammar G *is the set of all strings in left context* ε *generated by* S*:*

$$L(G) = \{\, w \mid w \in \Sigma^*, \vdash_G S\big(\varepsilon\langle w\rangle\big) \,\}.$$

Using both kinds of left context operators (proper $\lhd$ and extended $\trianglelefteqslant$) is actually redundant, since each of them can be expressed through the other as follows. Let Σ^* define all strings in any contexts, then:

- a proper left context $\lhd D$ is equivalent to a concatenation $D'\Sigma^*$, for a new nonterminal symbol D' with a single rule $D' \to \varepsilon \,\&\, \trianglelefteqslant D$;

- an extended left context $\trianglelefteqslant E$ can be replaced by a concatenation $\Sigma^* E'$, where the new nonterminal E' has a single rule $E' \to \varepsilon \,\&\, \lhd E$.

Using both context operators shall become essential later in Section 4.4, when transforming the grammar to a normal form, in which the empty string is prohibited.

The following sample grammar with contexts defines a rather simple language, which is an intersection of two ordinary context-free languages, and hence could be defined by a conjunctive grammar without contexts. The value of this example is in demonstrating the machinery of contexts in action.

Example 2.1. The following grammar generates the language $\{\, a^n b^n c^n d^n \mid n \geqslant 0 \,\}$.

$$
\begin{array}{lcl}
S & \to & aSd \mid bSc \mid \varepsilon \,\&\, \lhd A \\
A & \to & aAb \mid \varepsilon
\end{array}
$$

The symbol A generates all strings $a^n b^n$, with $n \geqslant 0$, in any context. Without the context specification $\lhd A$, the symbol S would define all strings of the form $w \cdot h(w^R)$, where $w \in \{a, b\}^*$, w^R denotes the mirror image of w, and h is a mapping a to d and b to c. However, the rule $S \to \varepsilon \,\&\, \lhd A$ ensures that the first half of the string is of the form $a^n b^n$ for some $n \geqslant 0$, and therefore S generates only strings of the form $a^n b^n c^n d^n$ with $n \geqslant 0$. Consider the following logical derivation of the fact that the string $abcd$ in left context ε is defined by S.

$$
\begin{array}{rll}
 & \vdash a\big(\varepsilon\langle a\rangle\big) & (axiom) \\
 & \vdash b\big(a\langle b\rangle\big) & (axiom) \\
 & \vdash c\big(ab\langle c\rangle\big) & (axiom) \\
 & \vdash d\big(abc\langle d\rangle\big) & (axiom) \\
 & \vdash A\big(a\langle\varepsilon\rangle\big) & (A \to \varepsilon) \\
a\big(\varepsilon\langle a\rangle\big), A\big(a\langle\varepsilon\rangle\big), b\big(a\langle b\rangle\big) & \vdash A\big(\varepsilon\langle ab\rangle\big) & (A \to aAb) \\
A\big(\varepsilon\langle ab\rangle\big) & \vdash S\big(ab\langle\varepsilon\rangle\big) & (S \to \varepsilon \,\&\, \lhd A) \\
b\big(a\langle b\rangle\big), S\big(ab\langle\varepsilon\rangle\big), c\big(ab\langle c\rangle\big) & \vdash S\big(a\langle bc\rangle\big) & (S \to bSc) \\
a\big(\varepsilon\langle a\rangle\big), S\big(a\langle bc\rangle\big), d\big(abc\langle d\rangle\big) & \vdash S\big(\varepsilon\langle abcd\rangle\big) & (S \to aSd)
\end{array}
$$

The tree corresponding to this deduction is given in Figure 2.1, where the reference to a context is marked by a dotted arrow. This tree represents a parse of the string according to the grammar.

In the next example a similar language is defined by a grammar that uses an extended context operator.

Example 2.2. Consider a grammar defining the language $\{\, a^n b^n c^n \mid n \geqslant 0 \,\}$.

$$
\begin{array}{lcl}
S & \to & aSc \mid B \,\&\, \trianglelefteq E \\
B & \to & bB \mid \varepsilon \\
E & \to & aEb \mid \varepsilon
\end{array}
$$

The nonterminal E generates all strings $a^n b^n$, with $n \geqslant 0$, in any contexts. Without the context operator $\trianglelefteq E$, the symbol S would define the language $\{\, a^n b^m c^n \mid n, m \geqslant 0 \,\}$. However, the rule $S \to B \,\&\, \trianglelefteq E$ ensures that the prefix of the string is of the form $a^k b^k$, for some $k \geqslant 0$, and therefore S only defines strings of the form $a^n b^n c^n$, with $n \geqslant 0$.

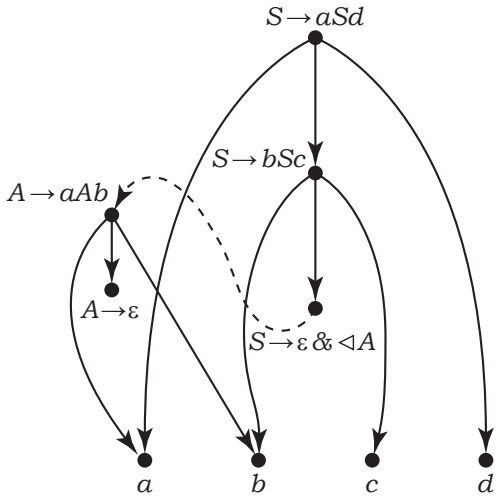


Figure 2.1: A parse tree of the string *abcd* according to the grammar in Example 2.1.

Consider the following logical deduction of the fact that the string *abc* is defined by this grammar.

$$\begin{aligned}
&\vdash a\big(\varepsilon\langle a\rangle\big) && (axiom)\\
&\vdash b\big(a\langle b\rangle\big) && (axiom)\\
&\vdash c\big(ab\langle c\rangle\big) && (axiom)\\
&\vdash E\big(a\langle\varepsilon\rangle\big) && (E \to \varepsilon)\\
a\big(\varepsilon\langle a\rangle\big), E\big(a\langle\varepsilon\rangle\big), b\big(a\langle b\rangle\big) &\vdash E\big(\varepsilon\langle ab\rangle\big) && (E \to aEb)\\
&\vdash B\big(ab\langle\varepsilon\rangle\big) && (B \to \varepsilon)\\
b\big(a\langle b\rangle\big), B\big(ab\langle\varepsilon\rangle\big) &\vdash B\big(a\langle b\rangle\big) && (B \to bB)\\
B\big(a\langle b\rangle\big), E\big(\varepsilon\langle ab\rangle\big) &\vdash S\big(a\langle b\rangle\big) && (S \to B \,\&\, \trianglelefteq E)\\
a\big(\varepsilon\langle a\rangle\big), S\big(a\langle b\rangle\big), c\big(ab\langle c\rangle\big) &\vdash S\big(\varepsilon\langle abc\rangle\big) && (S \to aSc)
\end{aligned}$$

The deductions according to Definition 2.2 are just a formalization of such parse trees as the one in Figure 2.1, in the same way as Chomsky's string rewriting formalizes context-free parse trees. Even though deductions look bulkier than string rewriting, they ultimately represent the same logical content, which is as simple as a parse tree.

2.2 Grammars with two-sided contexts

Let us now consider grammars which allow operators for expressing the form of the left ($\lhd$, $\trianglelefteq$) and the right contexts ($\rhd$, $\trianglerighteq$) of a substring being defined.

Definition 2.3. *A grammar with two-sided contexts is a quadruple* $G = (\Sigma, N, R, S)$*, where*

- Σ *is the alphabet of the language being defined;*
- N *is a finite set of auxiliary symbols ("nonterminal symbols" in Chomsky's terminology), which denote the properties of strings defined in the grammar;*
- R *is a finite set of grammar rules, each of the form*

$$\begin{aligned} A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_n \,\&\\ \&\, \trianglerighteq\kappa_1 \,\&\, \ldots \,\&\, \trianglerighteq\kappa_{m'} \,\&\, \rhd\delta_1 \,\&\, \ldots \,\&\, \rhd\delta_{n'}, \end{aligned} \tag{2.2}$$

  with $A \in N$*,* $k \geqslant 1$*,* $m, n, m', n' \geqslant 0$ *and* $\alpha_i, \beta_i, \gamma_i, \kappa_i, \delta_i \in (\Sigma \cup N)^*$*;*
- $S \in N$ *is a symbol representing well-formed sentences of the language.*

If all rules in a grammar have only left contexts (that is, if $m' = n' = 0$), then this is a grammar with one-sided contexts. If no context operators are ever used ($m = n = m' = n' = 0$), this is a conjunctive grammar, and if the conjunction is also never used ($k = 1$), this is an ordinary context-free grammar.

For each rule (2.2), each term α_i, $\lhd\beta_i$, $\trianglelefteq\gamma_i$, $\trianglerighteq\kappa_i$ and $\rhd\delta_i$ is called a *conjunct*. Denote by $u\langle w\rangle v$ a substring $w \in \Sigma^*$, which is preceded by $u \in \Sigma^*$ and followed by $v \in \Sigma^*$. Intuitively, such a substring is generated by the rule (2.2), if

- each *base conjunct* $\alpha_i = X_1 \ldots X_\ell$ gives a representation of w as a concatenation of shorter substrings described by $X_1, \ldots, X_\ell$, as in context-free grammars;
- each conjunct $\lhd\beta_i$ similarly describes the form of the *left context* u;
- each conjunct $\trianglelefteq\gamma_i$ describes the form of the *extended left context* uw;
- each conjunct $\trianglerighteq\kappa_i$ describes the *extended right context* wv;
- each conjunct $\rhd\delta_i$ describes the *right context* v.

The semantics of grammars with two-sided contexts, similarly to the one-sided case, are defined by a deduction system of elementary propositions of the form "a string $w \in \Sigma^*$ written in a left context $u \in \Sigma^*$ and in a right context $v \in \Sigma^*$ has the property $X \in \Sigma \cup N$", denoted by $X\big(u\langle w\rangle v\big)$. The deduction begins with axioms: any symbol $a \in \Sigma$ written in any context has the property a, denoted by $a\big(u\langle a\rangle v\big)$ for all $u, v \in \Sigma^*$. Each rule in R is

then regarded as a schema for deduction rules. For example, a rule $A \to BC$ allows making deductions of the form

$$B\big(u\langle w\rangle w'v\big), C\big(uw\langle w'\rangle v\big) \vdash_G A\big(u\langle ww'\rangle v\big) \quad \text{(for all } u, w, w', v \in \Sigma^*\text{)},$$

which, similarly to the case of grammars with one-sided contexts, is a concatenation of w and w' respecting the contexts. If the rule is of the form $A \to BC \,\&\, {\lhd}D$, this deduction requires an extra premise for the context operator ${\lhd}D$.

$$B\big(u\langle w\rangle w'v\big), C\big(uw\langle w'\rangle v\big), D\big(\varepsilon\langle u\rangle ww'v\big) \vdash_G A\big(u\langle ww'\rangle v\big)$$

And if the rule is $A \to BC \,\&\, {\rhd}F$, the deduction proceeds as follows.

$$B\big(u\langle w\rangle w'v\big), C\big(uw\langle w'\rangle v\big), F\big(u\langle ww'v\rangle\varepsilon\big) \vdash_G A\big(u\langle ww'\rangle v\big)$$

Now the general form of the deduction schema induced by a rule in R shall be defined.

Definition 2.4. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts. Define the following deduction system of propositions of the form $X\big(u\langle w\rangle v\big)$, with $X \in \Sigma \cup N$ and $u, w, v \in \Sigma^*$. There is a single axiom scheme $\vdash_G a\big(u\langle a\rangle v\big)$, for all $a \in \Sigma$ and $u, v \in \Sigma^*$. Each rule (2.2) in R defines the following scheme for deduction rules:*

$$I \vdash_G A\big(u\langle w\rangle v\big),$$

for all $u, w, v \in \Sigma^*$ and for every set of propositions I satisfying the properties below.

- *For every base conjunct $\alpha_i = X_1 \dots X_\ell$, with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $w = w_1 \dots w_\ell$ with the proposition $X_j\big(uw_1 \dots w_{j-1}\langle w_j\rangle w_{j+1} \dots w_\ell v\big)$ in I for all $j \in \{1, \dots, \ell\}$.*
- *For every conjunct ${\lhd}\beta_i = {\lhd}X_1 \dots X_\ell$ there should be such a partition $u = u_1 \dots u_\ell$, that $X_j\big(u_1 \dots u_{j-1}\langle u_j\rangle u_{j+1} \dots u_\ell wv\big) \in I$ for all $j \in \{1, \dots, \ell\}$.*
- *Every conjunct ${\lhdeq}\gamma_i = {\lhdeq}X_1 \dots X_\ell$ should have a corresponding partition $uw = x_1 \dots x_\ell$ with $X_j\big(x_1 \dots x_{j-1}\langle x_j\rangle x_{j+1} \dots x_\ell v\big) \in I$ for all $j \in \{1, \dots, \ell\}$.*
- *For every conjunct ${\rhd}\delta_i$ and ${\rhdeq}\kappa_i$, the conditions are defined symmetrically.*

The language defined by a symbol $A \in N$ is defined as

$$L_G(A) = \{\, u\langle w\rangle v \mid u, w, v \in \Sigma^*,\ \vdash_G A\big(u\langle w\rangle v\big) \,\}.$$

The language generated by the grammar G is the set of all strings with empty left and right contexts generated by S.

$$L(G) = \{\, w \mid w \in \Sigma^*,\ \vdash_G S\big(\varepsilon\langle w\rangle\varepsilon\big) \,\}$$

The following grammar with two-sided contexts defines the language of all strings of the form $a^n b^n c^n d^n$ $(n \geqslant 0)$, possibly with a symbol e inserted anywhere in d^n.

Example 2.3. The following grammar generates the language $\{\, a^n b^n c^n d^n \mid n \geqslant 0 \,\} \cup \{\, a^n b^n c^n d^\ell e d^{n-\ell} \mid n \geqslant \ell \geqslant 0 \,\}$.

$$\begin{aligned} S &\to aSd \mid bSc \mid \varepsilon \mathbin{\&} \lhd A \mid Se \mathbin{\&} \rhd D \\ A &\to aAb \mid \varepsilon \\ D &\to Dd \mid \varepsilon \end{aligned}$$

The rules $S \to aSd$ and $S \to bSc$ match each symbol a or b in the first part of the string to the corresponding d or c in its second part. In the middle of the string, the rule $S \to \varepsilon \mathbin{\&} \lhd A$ ensures that the first half of the string is $a^n b^n$. These symbols must have matching symbols $c^n d^n$ in the second half. Furthermore, the rule $S \to Se \mathbin{\&} \rhd D$ allows inserting the symbol e only in a right context of the form d^*.

The following deduction proves that the string *abced* has the property S.

$$\begin{aligned} &\vdash a\big(\varepsilon\langle a\rangle bced\big) && (axiom) \\ &\vdash b\big(a\langle b\rangle ced\big) && (axiom) \\ &\vdash c\big(ab\langle c\rangle ed\big) && (axiom) \\ &\vdash e\big(abc\langle e\rangle d\big) && (axiom) \\ &\vdash d\big(abce\langle d\rangle\varepsilon\big) && (axiom) \\ &\vdash A\big(a\langle\varepsilon\rangle bced\big) && (A \to \varepsilon) \\ a\big(\varepsilon\langle a\rangle bced\big), A\big(a\langle\varepsilon\rangle bced\big), b\big(a\langle b\rangle ced\big) &\vdash A\big(\varepsilon\langle ab\rangle ced\big) && (A \to aAb) \\ A\big(\varepsilon\langle ab\rangle ced\big) &\vdash S\big(ab\langle\varepsilon\rangle ced\big) && (S \to \varepsilon \mathbin{\&} \lhd A) \\ b\big(a\langle b\rangle ced\big), S\big(ab\langle\varepsilon\rangle ced\big), c\big(ab\langle c\rangle ed\big) &\vdash S\big(a\langle bc\rangle ed\big) && (S \to bSc) \\ &\vdash D\big(abce\langle\varepsilon\rangle d\big) && (D \to \varepsilon) \\ D\big(abce\langle\varepsilon\rangle d\big), d\big(abce\langle d\rangle\varepsilon\big) &\vdash D\big(abce\langle d\rangle\varepsilon\big) && (D \to Dd) \\ S\big(a\langle bc\rangle ed\big), e\big(abc\langle e\rangle d\big), D\big(abce\langle d\rangle\varepsilon\big) &\vdash S\big(a\langle bce\rangle d\big) && (S \to Se \mathbin{\&} \rhd D) \\ a\big(\varepsilon\langle a\rangle bced\big), S\big(a\langle bce\rangle d\big), d\big(abce\langle d\rangle\varepsilon\big) &\vdash S\big(\varepsilon\langle abced\rangle\varepsilon\big) && (S \to aSd) \end{aligned}$$

This deduction can be represented as the tree in Figure 2.2.

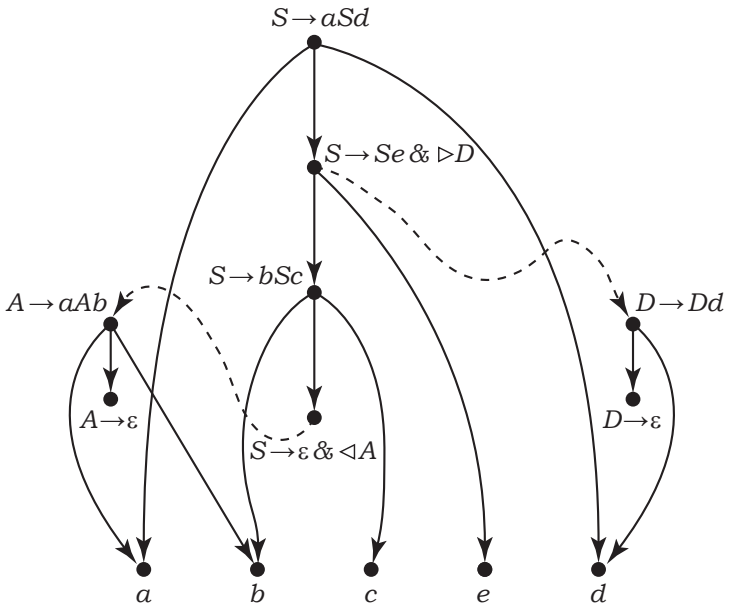


Figure 2.2: A parse tree of the string *abced* according to the grammar in Example 2.3.

Note, that in fact, using both kinds of left context operators and both kinds of right context operators is redundant, since they can be expressed through each other as follows:

- $\lhd D$ is equivalent to $\widetilde{D}\Sigma^*$, where the new nonterminal symbol $\widetilde{D}$ has the sole rule $\widetilde{D} \to \varepsilon \,\&\, \trianglelefteq D$;
- $\trianglelefteq E$ can be replaced by $\Sigma^*\widetilde{E}$, where the new nonterminal $\widetilde{E}$ has the unique rule $\widetilde{E} \to \varepsilon \,\&\, \lhd E$;
- $\trianglerighteq F$ can be expressed as $\widetilde{F}\Sigma^*$, where the new nonterminal $\widetilde{F}$ has the only rule $\widetilde{F} \to \varepsilon \,\&\, \rhd F$;
- $\rhd H$ is representable as $\Sigma^*\widetilde{H}$, where the new nonterminal $\widetilde{H}$ has the unique rule $\widetilde{H} \to \varepsilon \,\&\, \trianglerighteq H$.

Again, having all four kinds of context operators shall become essential when a grammar is transformed to a normal form (Section 4.5), where the empty string is prohibited.

Context operators can also be used to refer to the form of the entire string, as follows. Consider a rule $B \to \Phi \,\&\, \lhd X_A$, with Φ being any conjuncts and let nonterminal X_A have rules $X_A \to X_A a$ (for all $a \in \Sigma$) and $X_A \to \varepsilon \,\&\, \rhd A$.

Then the context specification $\lhd X_A$ in the rule for B allows referring to the form of the entire string.

2.3 Examples of grammars

Several examples of grammars with contexts generating important syntactic constructs are given below. All examples use ordinary context-free elements, such as a grammar for $\{\, a^n b^n \mid n \geqslant 0 \,\}$, and combine these elements using the new context operators. This leads to natural specifications of languages in the style of classical formal grammars.

The ideas of the examples given in this section have been implemented in an extensive example of a grammar with left contexts, which defines the complete syntax of a simple typed programming language. That grammar is discussed in the next chapter.

2.3.1 Declarations of objects before use

The following example defines an abstract language representing declaration of identifiers before their use. Though the same language can be defined by a conjunctive grammar [68, Ex. 3], the grammar with contexts given below provides a more natural definition of this construct.

Example 2.4. The following grammar defines the language

$$\begin{aligned}\big\{ u_1 \dots u_n \;\mid\; & \text{for every } u_i, \\ & \textbf{either } u_i \in d^* a, \\ & \textbf{or } u_i = c^k a \text{ and there exists } j < i, \text{ for which } u_j = d^k a \big\}.\end{aligned} \tag{2.3}$$

$$\begin{array}{lcl} S & \to & DS \mid US \mid \varepsilon \\ D & \to & dD \mid a \\ C & \to & cC \mid a \\ U & \to & C \,\&\, \lhd EFa \\ E & \to & CE \mid DE \mid \varepsilon \\ F & \to & dFc \mid aE \end{array}$$

Substrings of the form $d^k a$ stand for declarations, while every substring of the form $c^k a$ is a reference to a declaration of the form $d^k a$. The claim is that S generates a string $u_1 \dots u_\ell \langle u_{\ell+1} \dots u_n \rangle$, with $u_i \in c^* a \cup d^* a$, if and only if every reference u_i in the suffix $u_{\ell+1} \dots u_n$ has a corresponding earlier declaration in the prefix $u_1 \dots u_{i-1}$. The grammar defines all strings satisfying this condition inductively on ℓ. The base case is given by the rule $S \to \varepsilon$: the string $u_1 \dots u_n \langle \varepsilon \rangle$ has the desired property. The rule $S \to DS$

appends any declaration (D), while the rules $S \to US$ and $U \to C \,\&\, \triangleleft EFa$ append a reference (C), which has a matching earlier declaration ($\triangleleft EFa$). The latter context specification checks the declaration as follows: the symbol E represents the prefix of the string up to that declaration, while F matches the symbols d in the declaration to the symbols c in the reference.

In the examples given in this chapter, identifiers are given in unary notation, and are matched by rules of the same kind as the rules defining the language $\{\, a^n b^n \mid n \geqslant 0 \,\}$. In fact, it is possible to extend the examples so that identifiers over an arbitrary alphabet Σ can be used. This is due to the fact that there is a conjunctive grammar generating the language $\{\, w\#w \mid w \in \Sigma^* \,\}$, for some separator $\# \notin \Sigma$ [50, 68].

2.3.2 Declarations of objects before or after use

Consider a variant of the problem of checking that every identifier is declared before use, where the identifiers may be declared *before or after* their use. This is fairly common: consider, for instance, the declaration of classes in C++, where an earlier defined method can refer to a class member defined later. However, no conjunctive or Boolean grammar expressing this construct is known.

Example 2.5. Consider the language

$$\begin{aligned}\big\{ u_1 \dots u_n \;\mid\; & \text{for every } u_i, \\ & \textbf{either } u_i \in d^* a, \\ & \textbf{or } u_i = c^k a \text{ and there exists } j \in \{1, \dots, n\} \text{ with } u_j = d^k a \big\}.\end{aligned} \tag{2.4}$$

This is an abstract language representing declaration of identifiers *before or after* their use. As in Example 2.4, substrings $d^k a$ are declarations and substrings $c^k a$ are references. This language is generated by the following grammar.

$$\begin{array}{lcl} S & \to & DS \mid US \mid CS \,\&\, HaE \mid \varepsilon \\ D & \to & dD \mid a \\ C & \to & cC \mid a \\ U & \to & C \,\&\, \triangleleft EFa \\ E & \to & CE \mid DE \mid \varepsilon \\ F & \to & dFc \mid aE \\ H & \to & cHd \mid aE \end{array}$$

As compared with the grammar in Example 2.4, this grammar allows matching a reference to a declaration located *later* in the string, which is

done by the rule $S \to CS \,\&\, HaE$. Here the first conjunct CS appends a reference, while the other conjunct HaE uses H to match the cs forming this reference to the ds in the later declaration.

The grammar in Example 2.5 uses context specifications ($\lhd EFa$) along with iterated conjunction ($S \to CS \,\&\, HaE$) to express what would be more naturally expressed in terms of two-sided contexts. The following grammar defines the language in a much more natural way.

Example 2.6. The following grammar with two-sided contexts defines language (2.4).

$$\begin{array}{lcl} S & \to & DS \mid US \mid VS \mid \varepsilon \\ D & \to & dD \mid a \\ C & \to & cC \mid a \\ U & \to & C \,\&\, \lhd EFa \\ V & \to & C \,\&\, \rhd HaE \\ E & \to & CE \mid DE \mid \varepsilon \\ F & \to & dFc \mid aE \\ H & \to & cHd \mid aE \end{array}$$

Declarations of the form d^*a are added by the rule $S \to DS$. The rule $S \to US$ appends a reference of the form c^*a, restricted by an extended left context $\lhd EFa$, which ensures that this reference has a matching *earlier* declaration; here E represents the prefix of the string up to that earlier declaration, while F matches the symbols d in the declaration to the symbols c in the reference. The possibility of a *later* declaration is checked by another rule $S \to VS$, which adds a reference of the form c^*a with an extended right context $\rhd HaE$, where H is used to match the cs forming this reference to the ds in the later declaration. Using these rules, S generates substrings of the form $u_1 \dots u_\ell \langle u_{\ell+1} \dots u_n \rangle \varepsilon$, with $0 \leqslant \ell \leqslant n$ and $u_i \in c^*a \cup d^*a$, such that every reference in $u_{\ell+1} \dots u_n$ has a corresponding declaration somewhere in the whole string $u_1 \dots u_n$.

2.3.3 Forward declarations of objects

The next example abstracts the syntactic mechanism of *function prototypes*, found in the C programming language and, under the name of *forward declarations*, in the programming language Pascal [30]. The grammar for this language, that is given below, is based on the grammar from the previous example.

Example 2.7. Consider the language

$$\{u_1 \dots u_n \mid \text{for every } u_i, \textbf{either } u_i = d^k a \text{ and there exists } j > i, \text{ such that } u_j = b^k a, \textbf{ or } u_i = c^k a \text{ and there exists } j < i, \text{ for which } u_j = d^k a, \textbf{ or } u_i \in b^* a\}.$$

A substring of the form $d^k a$ represents a function prototype and a substring $b^k a$ represents its body. Calls to functions are expressed as substrings $c^k a$. The first condition in the definition of the language means that every prototype must have a body somewhere later, while the second restriction requires that a call to a function must be preceded by its prototype. It is also possible to further extend the language so that a call could refer directly to a preceding function body, without using a prototype.

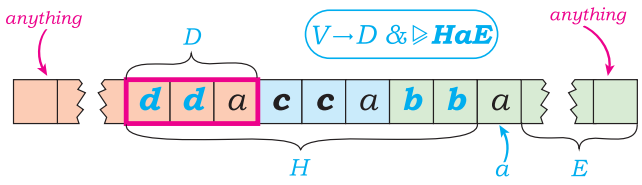


Figure 2.3: The grammar from Example 2.7: how a *following* body is found for a declaration.

This language can be generated by the following grammar with two-sided contexts.

$$\begin{aligned}
S &\to US \mid VS \mid BS \mid \varepsilon \\
B &\to bB \mid a \\
C &\to cC \mid a \\
D &\to dD \mid a \\
U &\to C \,\&\, \triangleleft EFa \\
V &\to D \,\&\, \triangleright HaE \\
E &\to BE \mid CE \mid DE \mid \varepsilon \\
F &\to dFc \mid aE \\
H &\to dHb \mid aE
\end{aligned}$$

The idea of the grammar is that nonterminal S should generate a substring $u_1 \dots u_\ell \langle u_{\ell+1} \dots u_n \rangle \varepsilon$, with $0 \leqslant \ell \leqslant n$ and $u_i \in b^*a \cup c^*a \cup d^*a$ if and only if every prototype $u_i = d^k a$ in $u_{\ell+1} \dots u_n$ has a corresponding body $b^k a$ in $u_{i+1} \dots u_n$ and every call $u_i = c^k a$ in $u_{\ell+1} \dots u_n$ has a corresponding prototype $d^k a$ in $u_1 \dots u_{i-1}$. The rules for S define all substrings satisfying these conditions inductively on the length of those substrings, until the entire string $\varepsilon \langle u_1 \dots u_n \rangle \varepsilon$ is defined. The rule $S \to \varepsilon$ defines the base case: the string $u_1 \dots u_n \langle \varepsilon \rangle \varepsilon$ has the desired property.

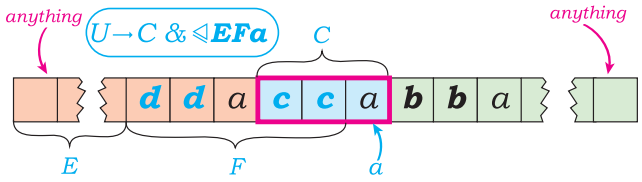


Figure 2.4: The grammar from Example 2.7: how a *preceding* declaration is found for a call.

The rules $S \to VS$ and $V \to D \,\&\, \rhd HaE$ append a prototype $d^k a$ and the extended right context of the form $d^k a \dots b^k a \dots$ ensures that this prototype has a matching body somewhere *later* within the string. The rules $S \to US$ and $U \to C \,\&\, \lhd EFa$ append a call $c^k a$, and the context specification $\dots d^k a \dots c^k a$ checks that it has a matching prototype *earlier* in the string. Function bodies $b^k a$ are added by the rule $S \to BS$.

2.3.4 References in natural languages

The next example uses the ideas of Example 2.6 to represent a linguistic phenomenon of *anaphora* [22].

Example 2.8. Consider the following two English sentences.

(1) John$_i$ skipped the lecture, because he$_i$ was busy.

(2) Because he$_i$ was busy, John$_i$ skipped the lecture.

Both sentences have a pronoun in one of the clauses, which refers to a noun in the other clause. In sentence (1), the pronoun he points to its left towards its antecedent John, while the pronoun in sentence (2) refers to its right towards its postcedent (such a reference to the right is sometimes called a *cataphora*).

The following grammar with two-sided contexts defines complex sentences, where a pronoun always refers to a contextual entity within the same sentence. For the sake of simplicity, the grammar does not allow for agreement of gender or grammatical number.

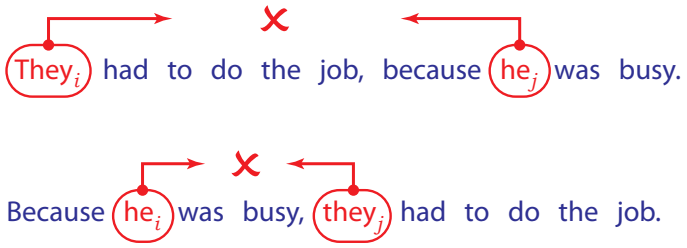


Figure 2.5: Examples of sentences not defined by the grammar from Example 2.8.

$$
\begin{aligned}
&\mathit{ComplexSentence} \to \mathit{Clause}\ ,\ \mathit{SubordinatingConjunction}\ \mathit{Clause}\ \mid \\
&\qquad\qquad \mathit{SubordinatingConjunction}\ \mathit{Clause}\ ,\ \mathit{Clause} \\
&\mathit{SubordinatingConjunction} \to \mathsf{because} \mid \ldots \\
&\mathit{Clause} \to \mathit{NounPhrase}\ \mathit{VerbPhrase} \\
&\mathit{NounPhrase} \to \mathit{Noun} \mid \mathit{ValidPronoun} \\
&\mathit{Noun} \to \ldots \\
&\mathit{VerbPhrase} \to \ldots \\
&\mathit{Pronoun} \to \mathsf{he} \mid \ldots \\
&\mathit{ValidPronoun} \to \mathit{Pronoun}\ \&\ \triangleleft\ \mathit{anything}\ \mathit{Noun}\ \mathit{anything}\ \mid \\
&\qquad\qquad \mathit{Pronoun}\ \&\ \triangleright\ \mathit{anything}\ \mathit{Noun}\ \mathit{anything} \\
&\mathit{anything} \to \mathsf{a}\ \mathit{anything} \mid \mathsf{b}\ \mathit{anything} \mid \ldots
\end{aligned}
$$

The sentences defined by this grammar may only have a single main clause and a single subordinary clause, both of which follow the subject–verb–object (SVO) positioning. The rules for the nonterminal *ValidPronoun* define a pronoun if and only if there is a noun located either to the left of it (antecedent) or to the right of it (postcedent).

Another example deals with *reflexive possessive pronouns*, met, for instance, in Swedish [26]. Such a pronoun reflects that the subject in the phrase owns the object.

Example 2.9. Consider the following Swedish sentence.

(3) Jonas$_i$ är i Oslo och Erik$_j$ går på bio med hans$_i$ fru.
Jonas is in Oslo and Erik goes to cinema with his wife.

In this sentence, possessive pronoun hans ("his") refers to Jonas, but not to the subject of the clause (Erik). In order to make a reference to Erik in the second clause, the reflexive pronoun sin ("his own") should be used, as in the following sentence.

(4) Jonas$_i$ är i Oslo och Erik$_j$ går på bio med sin$_j$ fru.
Jonas is in Oslo and Erik goes to cinema with his own wife.

The following fragment of a grammar with contexts defines sentences where a possessive pronoun can only be used if the sentence is complex, and a reflexive possessive pronoun is only allowed when it refers to a noun to its left and the sentence has only one clause. This means that the use of reflexive possessive pronouns is preferred to the use of possessive pronouns.

$$
\begin{aligned}
&\mathit{ComplexSentence} \to \mathit{Clause}\ \mathit{CoordinatingConjunction}\ \mathit{Clause} \mid \\
&\qquad\qquad \mathit{Clause} \\
&\mathit{CoordinatingConjunction} \to \mathsf{och} \mid \ldots \\
&\mathit{Clause} \to \mathit{NounPhrase}\ \mathit{VerbPhrase} \\
&\mathit{VerbPhrase} \to \mathit{Verb}\ \mathit{NounPhrase} \\
&\mathit{PrepositionalPhrase} \to \mathit{Preposition}\ \mathit{NounPhrase} \\
&\mathit{Preposition} \to \mathsf{med} \mid \ldots \\
&\mathit{NounPhrase} \to \mathit{Noun} \mid \mathit{NounPhrase}\ \mathit{PrepositionalPhrase} \mid \\
&\qquad\qquad \mathit{ValidPossessivePronoun}\ \mathit{Noun} \\
&\mathit{Noun} \to \ldots \\
&\mathit{PossessivePronoun} \to \mathsf{hans} \mid \ldots \\
&\mathit{ReflexivePossessivePronoun} \to \mathsf{sin} \mid \ldots \\
&\mathit{ValidPossessivePronoun} \to \\
&\quad \mathit{PossessivePronoun} \\
&\qquad \mathbin{\&} \lhd \mathit{Noun}\ \mathit{anything}\ \mathit{CoordinatingConjunction}\ \mathit{anything} \mid \\
&\quad \mathit{ReflexivePossessivePronoun} \mathbin{\&} \lhd \mathit{anything}\ \mathit{Noun}\ \mathit{anything}
\end{aligned}
$$

As in the previous example, for the sake of simplicity, the grammar does not allow for agreement of grammatical gender and number.

2.3.5 Defining arbitrary cross-references

The next example gives a grammar with contexts that defines reachability on graphs. Sudborough [85] defined a linear context-free grammar for a special encoding of the graph reachability problem on acyclic graphs, in which every arc goes from a lower-numbered vertex to a higher-numbered vertex. The grammar presented below allows any graph and uses a direct encoding. This example illustrates the ability of grammars with contexts to define arbitrary cross-references.

Example 2.10. Consider encodings of directed graphs as strings of the form $b^s\, a^{i_1} b^{j_1}\, a^{i_2} b^{j_2}\, \ldots\, a^{i_n} b^{j_n}\, a^t$, with $s, t \geqslant 1$, $n \geqslant 0$, $i_k, j_k \geqslant 1$, where each block $a^i b^j$ denotes an arc from vertex number i to vertex number j, while the prefix b^s and the suffix a^t mark s as the source vertex and t as the target.

Formally, the language of graph reachability is defined as follows.

$$\big\{b^s\, a^{i_1} b^{j_1}\, a^{i_2} b^{j_2}\, \ldots\, a^{i_n} b^{j_n}\, a^t \;\big|\; s,t \geqslant 1,\, n > 0,\, i_k, j_k \geqslant 1,\\ \exists \ell_1, \ldots, \ell_m :\, s = i_{\ell_1},\, j_{\ell_1} = i_{\ell_2},\, j_{\ell_2} = i_{\ell_3},\, \ldots,\, j_{\ell_{m-1}} = i_{\ell_m},\, j_{\ell_m} = t\big\}$$

Then the following grammar defines all graphs with a path from s to t.

$$\begin{array}{lcl} S & \to & FDCA \mid F \\ A & \to & aA \mid a \\ B & \to & bB \mid b \\ C & \to & ABC \mid \varepsilon \\ D & \to & B \mathbin{\&} \lhd BCE \mid B \mathbin{\&} \rhd FDCA \mid B \mathbin{\&} \rhd F \\ E & \to & aEb \mid DCA \\ F & \to & bFa \mid bCa \end{array}$$

The grammar is centered around the nonterminal D, which generates all substrings $b^s a^{i_1} b^{j_1} \ldots a^{i_k} \langle b^{j_k} \rangle a^{i_{k+1}} b^{j_{k+1}} \ldots a^{i_n} b^{j_n} a^t$, such that there is a path from j_k to t in the graph. If this path is empty, then $j_k = t$. Otherwise, the first arc in the path can be listed either to the left or to the right of b^{j_k}. These three cases are handled by the three rules for D. Each of these rules generates b^{j_k} by the base conjunct B, and then uses an extended left or right context operator to match b^{j_k} to the tail of the next arc or to a^t.

The rule $D \to B \mathbin{\&} \lhd BCE$ considers the case when the next arc in the path is located to the left of b^{j_k}. Let this arc be $a^{i_\ell} b^{j_\ell}$, for some $\ell < k$. Then the extended left context BCE covers the substring $b^s a^{i_1} b^{j_1} \ldots a^{i_\ell} b^{j_\ell} \ldots a^{i_k} b^{j_k}$. The concatenation BC skips the prefix $b^s a^{i_1} b^{j_1} \ldots a^{i_{\ell-1}} b^{j_{\ell-1}}$, and then the nonterminal E matches a^{i_ℓ} to b^{j_k}, verifying that $i_\ell = j_k$. After this, the rule $E \to DCA$ ensures that the substring b^{j_ℓ} is generated by D, that is, that there is a path from j_ℓ to t. The concatenation CA skips the inner substring $a^{i_{\ell+1}} b^{j_{\ell+1}} \ldots a^{i_k}$.

The second rule $D \to B \mathbin{\&} \rhd FDCA$ searches for the next arc to the right of b^{j_k}. Let this be the ℓ-th arc in the list, with $\ell > k$. The extended right context $FDCA$ should generate the suffix $b^{j_k} \ldots a^{i_\ell} b^{j_\ell} \ldots a^{i_n} b^{j_n} a^t$. The symbol F covers the substring $b^{j_k} \ldots a^{i_\ell}$, matching b^{j_k} to a^{i_ℓ}. Then, D generates the substring b^{j_ℓ}, checking that there is a path from j_ℓ to t. The concatenation CA skips the rest of the suffix.

Finally, if the path is of length zero, that is, $j_k = t$, then the rule $D \to B \mathbin{\&} \rhd F$ uses F to match b^{j_k} to the suffix a^t in the end of the string.

Once the symbol D checks the path from any vertex to the vertex t, for the initial symbol S, it is sufficient to match b^s in the beginning of the string to any arc $a^{j_k} b^{j_k}$, with $j_k = s$. This is done by the rule $S \to FDCA$, which

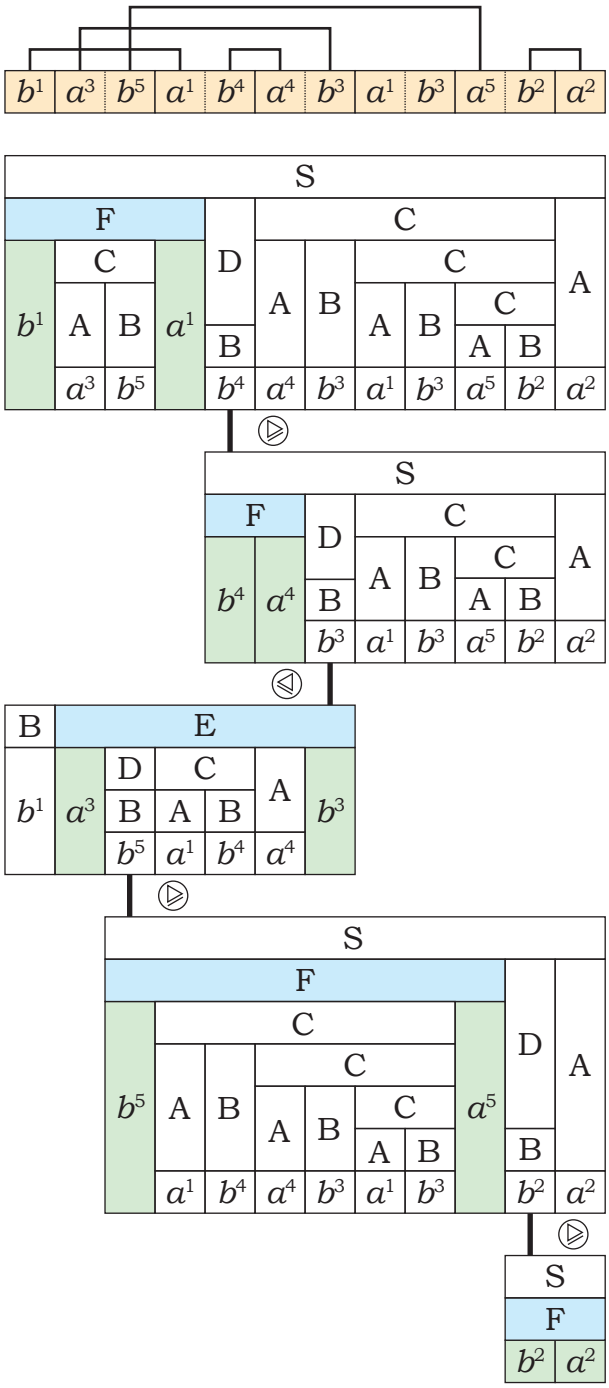
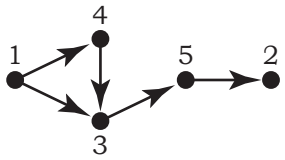


Figure 2.6: An informal diagram representing a parse of the string $b^1\ a^3b^5\ a^1b^4\ a^4b^3\ a^1b^3\ a^5b^2\ a^2$ according to the grammar in Example 2.10. Highlighted blocks correspond to the substrings constituting the path 1–4–3–5–2.

operates in the same way as the second rule for D. The case of s and t being the same node is handled by the rule $S \to F$.

Note, that for succinctness, one could replace the rules $D \to B \,\&\, \triangleright FDCA$ and $D \to B \,\&\, \triangleright F$ with a single rule $D \to B \,\&\, \triangleright S$.

The following example gives an instance of a correct graph encoding and its representation as a parse tree.



Consider a directed graph with five vertices, numbered from 1 to 5, and the arcs 3—5, 1—4, 4—3, 1—3, and 5—2, where the path from vertex 1 to vertex 2 is sought, and an encoding for this instance of the problem as a string $b^1\ a^3b^5\ a^1b^4\ a^4b^3\ a^1b^3\ a^5b^2\ a^2$. One of the paths from 1 to 2 is 1, 4, 3, 5, 2, and Figure 2.6 informally illustrates the parse of this string which follows that path according to the grammar.

2.4 Alternative characterizations of grammars with contexts

In this section, the semantics of grammars with contexts are formalized in two more ways, both of which are equivalent to the definition by deduction. One of these definitions directly expresses grammars in first-order logic over positions in a string [79], while the other one uses a generalization of *language equations*, in which the unknowns are *sets of strings with contexts* $u\langle v\rangle$. All connectives in the rules of a grammar—that is, concatenation, disjunction, conjunction and both context operators—are then interpreted as operations on such sets, and the resulting system of equations is proved to have a least fixpoint, as in the known cases of ordinary context-free grammars [20] and conjunctive grammars [51]. This least solution defines the language generated by the grammar.

2.4.1 Definition in the first-order logic over positions in the input

In the semantics defined in this subsection, nonterminal symbols of the grammar become *binary predicates*, with the arguments referring to positions in the string.

Definition 2.5. *Let $w \in \Sigma^*$ be a string and consider a set of variables, representing positions in w, ranging from 0 to $|w|$. Then the set of* terms *over these variables is defined inductively as follows:*

- *a variable is a term;*
- *the symbols* $\underline{\text{begin}}$ *and* $\underline{\text{end}}$ *are constant terms, which refer to the first and the last positions in the string;*
- *if* t *and* t' *are terms, then* $t < t'$ *and* $t = t'$ *are terms;*
- *if* t *is a term, then so are* $t + 1$ *and* $t - 1$*, representing the operations of increment and decrement.*

Every predicate has two arguments, corresponding to positions in the string: the beginning of the current substring and its ending. There are built-in unary predicates $a(t)$, for each $a \in \Sigma$, which assert that the symbol in position t in the string is a, and binary predicates $t < t'$ and $t = t'$ for comparing positions. Arguments to predicates are given as terms, which are either variables ($t = x$) or constants referring to the first and the last positions ($t = \underline{\text{begin}}$, $t = \underline{\text{end}}$), and which may be incremented ($t + 1$) or decremented ($t - 1$). Each formula is constructed from predicates using conjunction, disjunction and first-order existential quantification.

Definition 2.6. *Let* Σ *be an alphabet, let* N *be a finite set of predicate symbols, and let* $A \in N$*. Then:*

- *if* t *and* t' *are terms, then* $A(t, t')$ *is a formula;*
- *if* $a \in \Sigma$ *and* t *is a term, then* $a(t)$ *is a formula;*
- *if* t *and* t' *are terms, then* $t < t'$ *and* $t = t'$ *are formulae;*
- *if* φ *and* ψ *are formulae, then so are* $\varphi \wedge \psi$ *and* $\varphi \vee \psi$*;*
- *if* φ *is a formula and* x *is a free variable in* φ*, then so is* $\exists x(\varphi)$*.*

Each predicate $A(x, y)$ is defined by a formula $\varphi_A(x, y)$ that states the condition of a substring delimilited by positions x and y having the property A.

Definition 2.7. *A* grammar over positions in the input *is a quadruple* $G = (\Sigma, N, \langle \varphi_A \rangle_{A \in N}, \sigma)$*, where*

- Σ *is the finite alphabet of the language being defined;*
- N *is a finite set of predicate symbols, each of rank 2;*
- *each* $\varphi_A(x, y)$ *is a formula with two free variables* x *and* y*, which defines the predicate* $A(x, y)$*;*
- σ *is a formula with no free variables, which defines the condition of being a syntactically well-formed sentence.*

The membership of a string w in a language is expressed by the statement of the form σ, which may be true of false.

The following example demonstrates the definition.

Example 2.11. Consider the grammar with two-sided contexts that defines the singleton language $\{abca\}$.

$$\begin{array}{lcl} S & \to & aS \mid Sa \mid BC \\ A & \to & a \\ B & \to & b \,\&\, \lhd A \\ C & \to & c \,\&\, \rhd A \end{array}$$

This grammar is expressed by the following formulae defining predicates $S(x,y)$, $B(x,y)$, $A(x,y)$ and $C(x,y)$, respectively.

$$\begin{array}{lcl} S(x,y) & = & (a(x+1) \wedge S(x+1,y)) \vee (S(x,y-1) \wedge a(y)) \\ & & \vee\, (\exists z\, (x < z \wedge z < y \wedge B(x,z) \wedge C(z,y))) \\ A(x,y) & = & a(x+1) \wedge x+1 = y \\ B(x,y) & = & b(x+1) \wedge x+1 = y \wedge A(\underline{\text{begin}}, x) \\ C(x,y) & = & c(x+1) \wedge x+1 = y \wedge A(y, \underline{\text{end}}) \end{array}$$

In general, the language defined by some nonterminal $A \in N$ of a grammar with contexts can be represented in this logic as follows. First, assume that all rules of the grammar are of the following form.

$$\begin{aligned} A \to {} & B_1 C_1 \,\&\, \ldots \,\&\, B_k C_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n \,\&\, \\ & \rhdeq F_1 \,\&\, \ldots \,\&\, \rhdeq F_{n'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m'} \end{aligned} \tag{2.5}$$

$$\begin{aligned} A \to {} & a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n \,\&\, \\ & \rhdeq F_1 \,\&\, \ldots \,\&\, \rhdeq F_{n'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m'}, \end{aligned} \tag{2.6}$$

($k \geqslant 1$, $m, n, n', m' \geqslant 0$, $B_i, C_i, D_i, E_i, F_i, H_i \in N$, $a \in \Sigma$)

As will be shown later in Section 4.5, there is no loss of generality in this assumption.

The formula corresponding to a nonterminal $A \in N$ is a disjunction of two subformulae, one representing all rules for A of the form (2.6) and another all rules of the form (2.5). For a set $\mathcal{X} = \{X_1, \ldots, X_\ell\}$ with $X_i \in N$, and

for a context operator $Q \in \{\lhd, \trianglelefteq, \rhd, \trianglerighteq\}$, denote $QX := QX_1 \,\&\, \ldots \,\&\, QX_\ell$.

$$\varphi_A(x,y) = \bigvee_{\substack{A \to a \,\&\, \lhd\mathcal{D} \,\&\, \trianglelefteq\mathcal{E} \,\&\, \\ \rhd\mathcal{F} \,\&\, \trianglerighteq\mathcal{H} \in R}} \Bigg(a(x+1) \wedge x+1 = y \wedge$$

$$\bigwedge_{D\in\mathcal{D}} D(\underline{\text{begin}}, x) \wedge \bigwedge_{E\in\mathcal{E}} E(\underline{\text{begin}}, y) \wedge \bigwedge_{F\in\mathcal{F}} F(x, \underline{\text{end}}) \wedge \bigwedge_{H\in\mathcal{H}} H(y, \underline{\text{end}}) \Bigg) \vee$$

$$\bigvee_{\substack{A \to B_1C_1 \,\&\ldots\&\, B_kC_k \,\& \\ \lhd\mathcal{D} \,\&\, \trianglelefteq\mathcal{E} \,\&\, \rhd\mathcal{F} \,\&\, \trianglerighteq\mathcal{H} \in R}} \Bigg(\bigwedge_{i=1}^{k} \exists z (x < z \wedge z < y \wedge B_i(x,z) \wedge C_i(z,y)) \wedge$$

$$\bigwedge_{D\in\mathcal{D}} D(\underline{\text{begin}}, x) \wedge \bigwedge_{E\in\mathcal{E}} E(\underline{\text{begin}}, y) \wedge \bigwedge_{F\in\mathcal{F}} F(x, \underline{\text{end}}) \wedge \bigwedge_{H\in\mathcal{H}} H(y, \underline{\text{end}}) \Bigg)$$

The first subformula is a disjunction over all rules of the form (2.6), which has A on its left-hand side. The subformula corresponding to such a rule, is, in its turn, a conjunction of all conjuncts in it. The base conjunct a is expressed as a term $a(x+1)$, meaning that the $(x+1)$th symbol of the string is the terminal a. Every context specification $\lhd D$ is expressed as $D(\underline{\text{begin}}, x)$, asserting that D defines exactly the fragment of the string, beginning from its first position and ending at position x, which is essentially the left context with respect to the position x. Similarly, each conjunct $\trianglelefteq E$ is represented as $E(\underline{\text{begin}}, y)$, that is, this time the fragment again starts from the very first position of the string, but ends at position y. That is, it begins where the string itself begins and includes the fragment between the positions x and y, which is exactly the definition of the extended left context. Conjuncts of the form $\trianglerighteq F$ and $\rhd H$ are expressed in the formula in a symmetric way.

The second subformula corresponds to rules of the form (2.5). It is again a conjunction corresponding to all conjuncts of the rule. Every base conjunct of the form BC, which is a concatenation of two symbols B and C, is represented as a conjunction of two conditions: the fragment between the positions x and z should be defined by B and the fragment between z and y should be defined by C. The existence of a "middle" position z is given by the existential quantifier.

2.4.2 Definition by language equations

The representation of ordinary context-free grammars by language equations, introduced by Ginsburg and Rice [20], is one of the several ways of

defining their semantics. For example, a grammar $S \to aSb \mid \varepsilon$ is regarded as an equation $S = \big(\{a\} \cdot S \cdot \{b\}\big) \cup \{\varepsilon\}$, which has the unique solution $S = \{\, a^n b^n \mid n \geqslant 0 \,\}$. Conjunctive grammars inherit the same definition by equations [51], with the conjunction represented by the intersection operation. This important representation shall now be extended to grammars with contexts.

In order to represent contexts in the equations, the whole model has to be extended from ordinary formal languages to languages of strings with contexts. In the case of grammars with one-sided contexts, these strings are of the form $u\langle v\rangle$, and a language of strings with contexts is a set of pairs of strings $L \subseteq \Sigma^* \times \Sigma^*$. In what follows, only grammars with left contexts are considered; the results can be extended to the case of grammars with two-sided contexts in a straightforward way.

All usual operations on languages used in equations will have to be extended to languages of strings with contexts. For all $K, L \subseteq \Sigma^* \times \Sigma^*$, consider their:

- *union* $K \cup L = \{\, u\langle v\rangle \mid u\langle v\rangle \in K \text{ or } u\langle v\rangle \in L \,\}$;
- *intersection* $K \cap L = \{\, u\langle v\rangle \mid u\langle v\rangle \in K,\ u\langle v\rangle \in L \,\}$;
- *concatenation* $K \cdot L = \{\, u\langle vw\rangle \mid u\langle v\rangle \in K,\ uv\langle w\rangle \in L \,\}$;
- *(proper) left context* $\lhd L = \{\, u\langle v\rangle \mid \varepsilon\langle u\rangle \in L,\ v \in \Sigma^* \,\}$;
- *extended left context* $\trianglelefteq L = \{\, u\langle v\rangle \mid \varepsilon\langle uv\rangle \in L \,\}$.

A language of strings with contexts shall be occasionally called *a language with contexts* or simply *a language*. Given a grammar with contexts, the plan is to define a system of equations, in which the unknowns are languages (of strings with contexts) defined by the nonterminal symbols of the grammar. A solution of the system is any assignment of languages to the variables that turns each equation into an equality. The system will be defined such that it always has solutions, and among them there is a *least solution* with respect to the partial order $\sqsubseteq$ of *componentwise inclusion*, and this solution will be used to define the language generated by the grammar. In a similar way one can define the notion of *componentwise intersection* $\sqcup$.

The order of componentwise inclusion is defined on the set $(2^{\Sigma^* \times \Sigma^*})^n$ of n-tuples of languages, where n is the number of nonterminal symbols in the grammar. For any two such n-tuples, let $(K_1, \ldots, K_n) \sqsubseteq (L_1, \ldots, L_n)$ if and only if $K_1 \subseteq L_1, \ldots, K_n \subseteq L_n$. The least element under this order is the vector of empty sets $\bot = (\varnothing, \ldots, \varnothing)$.

Definition 2.8. *For every grammar with contexts* $G = (\Sigma, N, R, S)$*, the associated system of language equations is a system of equations in variables*

N, in which every variable assumes a value of a language with contexts $L \subseteq \Sigma^* \times \Sigma^*$, and which contains the following equation for every variable A.

$$A = \bigcup_{\substack{A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\& \\ \&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\& \\ \&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_{m'} \in R}} \Big[\bigcap_{i=1}^{k} \alpha_i \;\cap\; \bigcap_{i=1}^{m} \lhd\beta_i \;\cap\; \bigcap_{i=1}^{m'} \trianglelefteq\gamma_i \Big]. \tag{2.7}$$

Each string α_i, β_i or γ_i in such a system denotes a concatenation of variables and symbols, where each instance of a symbol $a \in \Sigma$ defines a constant language $\{\, x\langle a\rangle \mid x \in \Sigma^* \,\}$. If any of the α_i, β_i or γ_i is an empty string, it denotes the language $\{\, x\langle\varepsilon\rangle \mid x \in \Sigma^* \,\}$.

Let $N = \{A_1, \ldots, A_n\}$ and let $(L_{A_1}, \ldots, L_{A_n})$, with $L_{A_i} \subseteq \Sigma^* \times \Sigma^*$, be the least solution of the system (2.7). Define the language generated by each nonterminal symbol $A \in N$ as the corresponding component of this solution: $L_G(A) = L_A$. Let $L(G) = \{\, w \mid \varepsilon\langle w\rangle \in L_S \,\}$.

For this definition to be correct, it should be proved that any such system has a *least solution* with respect to the partial order $\sqsubseteq$ of componentwise inclusion. Let $N = \{A_1, \ldots, A_n\}$ be the set of variables. Then the system of language equations (2.7) can be denoted by

$$\begin{cases} A_1 &= \varphi_1(A_1, \ldots, A_n) \\ &\vdots \\ A_n &= \varphi_n(A_1, \ldots, A_n) \end{cases} \tag{2.8}$$

where the functions $\varphi_i \colon (2^{\Sigma^* \times \Sigma^*})^n \to 2^{\Sigma^* \times \Sigma^*}$ in the right-hand sides are defined as a composition of the operations of concatenation, union, intersection, left contexts ($\lhd$) and extended left contexts ($\trianglelefteq$). The right-hand sides of such a system can be represented as a vector function $\varphi = (\varphi_1, \ldots, \varphi_n)$, which has the following properties.

Lemma 2.1. *For every grammar with contexts, the vector function $\varphi = (\varphi_1, \ldots, \varphi_n)$ in the associated system of language equations is monotone, in the sense that for any two vectors K and L, the inequality $K \sqsubseteq L$ implies $\varphi(K) \sqsubseteq \varphi(L)$.*

The proof of Lemma 2.1 follows directly from the definitions of the operations, like in the case of conjunctive grammars [51].

Another property of functions in the right-hand sides of these equations is their *continuity*: whenever a sequence of arguments $\{L^{(i)}\}_{i=1}^{\infty}$ converges to L, the sequence formed by applying φ to all elements of the sequence converges to $\varphi(L)$. It is actually sufficient to prove the weaker property of $\sqcup$-*continuity*, defined with respect to *ascending sequences* of the form $L^{(1)} \sqsubseteq L^{(2)} \sqsubseteq \ldots \sqsubseteq L^{(k)} \sqsubseteq \ldots$

Lemma 2.2. *For every grammar with contexts, the vector function* $\varphi = (\varphi_1, \ldots, \varphi_n)$ *in the associated system of language equations is* $\sqcup$*-continuous, in the sense that for every ascending sequence* $\{L^{(i)}\}_{i=1}^{\infty}$ *of vectors of languages with contexts, it holds that*

$$\bigsqcup_{i=1}^{\infty} \varphi(L^{(i)}) = \varphi\Big(\bigsqcup_{i=1}^{\infty} L^{(i)}\Big).$$

The proof of this lemma is based on the fact that the membership of any element $u\langle v\rangle$ in an n-tuple of sets $\varphi(L^{(i)})$ depends on finitely many elements in the argument $L^{(i)}$.

The next result follows by the standard method of fixpoint iteration, as explained, for instance, in the handbook chapter by Autebert et al. [4]. According to this method, the vector function φ on the right-hand side of the system is iteratively applied to the vector of empty sets, and after countably many iterations, the process converges to the least solution of the system. The method applies to every system of equations (2.7) corresponding to a grammar with contexts, due to monotonicity and continuity of its right-hand sides (Lemmata 2.1 and 2.2).

Lemma 2.3. *Every system of language equations (2.8), where* φ *is monotone and* $\sqcup$*-continuous, has a least solution, which is given by*

$$\bigsqcup_{k=0}^{\infty} \varphi^k(\bot).$$

Example 2.12. The following system of equations represents the grammar in Example 2.1.

$$\begin{aligned} S &= \big(\Sigma^*\langle a\rangle \cdot S \cdot \Sigma^*\langle d\rangle\big) \cup \big(\Sigma^*\langle b\rangle \cdot S \cdot \Sigma^*\langle c\rangle\big) \cup \big(\Sigma^*\langle\varepsilon\rangle \cap \lhd A\big) \\ A &= \big(\Sigma^*\langle a\rangle \cdot A \cdot \Sigma^*\langle b\rangle\big) \cup \Sigma^*\langle\varepsilon\rangle \end{aligned}$$

Consider the sequence of iterations leading to the least solution of the system.

$$
\begin{aligned}
\varphi^0(\bot) &= \begin{pmatrix} \varnothing \\ \varnothing \end{pmatrix} \\
\varphi^1(\bot) &= \begin{pmatrix} \varnothing \\ \Sigma^*\langle\varepsilon\rangle \end{pmatrix} \\
\varphi^2(\bot) &= \begin{pmatrix} \varepsilon\langle\varepsilon\rangle \\ \Sigma^*\langle\varepsilon\rangle \cup \Sigma^*\langle ab\rangle \end{pmatrix} \\
\varphi^3(\bot) &= \begin{pmatrix} \{\varepsilon\langle\varepsilon\rangle,\ ab\langle\varepsilon\rangle\} \\ \Sigma^*\langle\varepsilon\rangle \cup \Sigma^*\langle ab\rangle \cup \Sigma^*\langle a^2b^2\rangle \end{pmatrix} \\
\varphi^4(\bot) &= \begin{pmatrix} \{\varepsilon\langle\varepsilon\rangle,\ ab\langle\varepsilon\rangle,\ a\langle bc\rangle,\ a^2b^2\langle\varepsilon\rangle\} \\ \Sigma^*\langle\varepsilon\rangle \cup \Sigma^*\langle ab\rangle \cup \Sigma^*\langle a^2b^2\rangle \cup \Sigma^*\langle a^3b^3\rangle \end{pmatrix} \\
\varphi^5(\bot) &= \begin{pmatrix} \{\varepsilon\langle\varepsilon\rangle,\ ab\langle\varepsilon\rangle,\ a^2b^2\langle\varepsilon\rangle,\ a^3b^3\langle\varepsilon\rangle,\ a\langle bc\rangle,\ aab\langle bc\rangle,\ \varepsilon\langle abcd\rangle\} \\ \Sigma^*\langle\varepsilon\rangle \cup \Sigma^*\langle ab\rangle \cup \Sigma^*\langle a^2b^2\rangle \cup \Sigma^*\langle a^3b^3\rangle \cup \Sigma^*\langle a^4b^4\rangle \end{pmatrix}, \quad \text{etc.}
\end{aligned}
$$

This sequence tends to a pair $S = \{\, a^{n-i}\langle a^i b^n c^n d^i\rangle \mid 0 \leqslant i \leqslant n,\ n \in \mathbb{N}\,\} \cup \{\, a^n b^{n-i}\langle b^i c^i\rangle \mid 0 \leqslant i \leqslant n,\ n \in \mathbb{N}\,\}$, $A = \{\, x\langle a^n b^n\rangle \mid x \in \Sigma^*\,\}$, which is therefore the least solution of the system.

This proves that every system of equations corresponding to a grammar with contexts has a least solution, and hence Definition 2.8 is correct. It remains to prove that Definitions 2.2 and 2.8 are equivalent, that is, they define the same languages $L_G(A)$, for all $A \in N$.

Let N be the set of nonterminal symbols, let $|N| = n$. For every n-tuple of languages with contexts $L \in (2^{\Sigma^* \times \Sigma^*})^n$ and for every symbol $A \in N$, denote the A-component of this tuple by $[L]_A$.

Theorem 2.1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, where $N = \{A_1, \ldots, A_n\}$, and let $A_i = \varphi_i(A_1, \ldots, A_n)$ $(1 \leqslant i \leqslant n)$ be the associated system of language equations. Let $u\langle v\rangle \in \Sigma^* \times \Sigma^*$ be a string with a context. Then, for every $A \in N$,*

$$
u\langle v\rangle \in \Big[\bigsqcup_{t \geqslant 0} \varphi^t(\varnothing, \ldots, \varnothing)\Big]_A \quad \textit{if and only if} \quad \vdash_G A\big(u\langle v\rangle\big).
$$

Proof. The argument uses the notation $\alpha(L)$ for the substitution of an n-tuple of languages with contexts L into a concatenation $\alpha = X_1 \ldots X_\ell$, with $X_i \in \Sigma \cup N$. Its value is a language with contexts, defined as $\alpha(L) = [L]_{X_1} \ldots [L]_{X_\ell}$, where $[L]_A$ with $A \in N$ is the A-component of L, while $[L]_a = \{\, x\langle a\rangle \mid x \in \Sigma^*\,\}$ for all $a \in \Sigma$.

$\Leftarrow$ The proof is by induction on t, the number of iterations made until the string $u\langle v\rangle$ appears in $\big[\varphi^t(\varnothing, \ldots, \varnothing)\big]_A$.

Basis. For $t = 0$, there are no $A \in N$ satisfying the assumptions.

Induction step. Let the string $u\langle v\rangle$ be obtained in t iterations, that is, $u\langle v\rangle \in [\varphi^t(\bot)]_A = A(\varphi^{t-1}(\bot))$. Then, substituting $\varphi^{t-1}(\bot)$ into the equation for A yields

$$u\langle v\rangle \in \bigcup_{\substack{A\to\alpha_1\,\&\ldots\&\,\alpha_k\,\&\\ \&\,\lhd\beta_1\,\&\cdots\&\,\lhd\beta_m\,\&\\ \&\,\trianglelefteq\gamma_1\,\&\cdots\&\,\trianglelefteq\gamma_{m'}\in R}} \Big(\bigcap_{i=1}^{k}\alpha_i(\varphi^{t-1}(\bot))\cap \cap\bigcap_{i=1}^{m}\lhd\beta_i(\varphi^{t-1}(\bot))\cap\bigcap_{i=1}^{m'}\trianglelefteq\gamma_i(\varphi^{t-1}(\bot))\Big).$$

Hence, there should exist such a rule

$$A \to \alpha_1\,\&\ldots\&\,\alpha_k\,\&\lhd\beta_1\,\&\ldots\lhd\beta_m\,\&\trianglelefteq\gamma_1\,\&\ldots\&\trianglelefteq\gamma_{m'}, \tag{2.9}$$

that a substitution of the languages $\varphi^{t-1}(\bot)$ into each conjunct of this rule (α_i, $\lhd\beta_i$ or $\trianglelefteq\gamma_i$) evaluates to a language containing $u\langle v\rangle$.

Consider each base conjunct α_i, with $i \in \{1,\ldots,k\}$. Let $\alpha_i = X_{i,1}\ldots X_{i,\ell_i}$, where $\ell_i \geqslant 0$ and $X_{i,j} \in \Sigma \cup N$ for all $j \in \{1,\ldots,\ell_i\}$. It is known that $u\langle v\rangle \in \alpha_i(\varphi^{t-1}(\bot))$. Then, by the definition of concatenation with contexts, there is such a partition $v = v_1\ldots v_{\ell_i}$, that

$$uv_1\ldots v_{j-1}\langle v_j\rangle \in X_{i,j}(\varphi^{t-1}(\bot)), \tag{2.10}$$

for all $j \in \{1,\ldots,\ell_i\}$. For each $X_{i,j}$, if $X_{i,j}$ is a terminal symbol $X_{i,j} = a \in \Sigma$, then the substitution gives a constant language $X_{i,j}(\varphi^{t-1}(\bot)) = \{\, x\langle a\rangle \mid x \in \Sigma^* \,\}$, and the above condition (2.10) implies that $v_j = a$; hence, the proposition $X_{i,j}\big(uv_1\ldots v_{j-1}\langle v_j\rangle\big)$ is deduced by an axiom. If $X_{i,j}$ is a nonterminal symbol $X_{i,j} = B \in N$, then the string $uv_1\ldots v_{j-1}\langle v_j\rangle$ should be in $X_{i,j}(\varphi^{t-1}(\bot)) = \big[\varphi^{t-1}(\bot)\big]_B$; by the induction hypothesis, $\vdash_G X_{i,j}\big(uv_1\ldots v_{j-1}\langle v_j\rangle\big)$.

The cases of conjuncts $\lhd\beta_i$ and $\trianglelefteq\gamma_i$ are analogous. Consider, for instance, $\lhd\beta_i$, and let $\beta_i = Y_{i,1}\ldots Y_{i,\ell'_i}$, where $Y_{i,j} \in \Sigma\cup N$. If $u\langle v\rangle \in \beta_i(\varphi^{t-1}(\bot))$, then there is a partition $u = u_1\ldots u_{\ell'_i}$, for which $u_1\ldots u_{j-1}\langle u_j\rangle \in Y_{i,j}(\varphi^{t-1}(\bot))$, for all $j \in \{1,\ldots,\ell'_i\}$. For every $Y_{i,j} = B \in N$, the induction hypothesis can be used to show that $\vdash_G Y_{i,j}\big(u_1\ldots u_{j-1}\langle u_j\rangle\big)$; for a terminal symbol $Y_{i,j} = a \in \Sigma$, this proposition is an axiom.

For each extended context $\trianglelefteq\gamma_i$, let $\gamma_i = Z_{i,1}\ldots Z_{i,\ell''_i}$. Here $u\langle v\rangle \in \gamma_i(\varphi^{t-1}(\bot))$ implies that there is a partition $uv = w_1\ldots w_{\ell''_i}$, such that $w_1\ldots w_{j-1}\langle w_j\rangle \in Z_{i,j}(\varphi^{t-1}(\bot))$, for all $j \in \{1,\ldots,\ell''_i\}$. Again, it is proved that $\vdash_G Z_{i,j}\big(w_1\ldots w_{j-1}\langle w_j\rangle\big)$, using the induction hypothesis for nonterminal symbols and the axiom for terminal symbols.

Thus all the premises for deducing the proposition $A\big(u\langle v\rangle\big)$ according to the rule (2.9) have been obtained, and one can make the desired step of deduction:

$$\big\{X_{i,j}\big(uv_1\dots v_{j-1}\langle v_j\rangle\big)\big\}_{i,j}, \big\{Y_{i,j}\big(u_1\dots u_{j-1}\langle u_j\rangle\big)\big\}_{i,j},$$
$$\big\{Z_{i,j}\big(w_1\dots w_{j-1}\langle w_j\rangle\big)\big\}_{i,j} \vdash_G A\big(u\langle v\rangle\big).$$

$\Leftarrow$ Assume that $A\big(u\langle v\rangle\big)$ is deducible in the grammar, and let p be the number of steps in its deduction. It is claimed that $u\langle v\rangle \in \big[\varphi^t(\bot)\big]_A$ for some t depending on p. The proof is by induction on p.

Basis. Let $p = 1$, that is, the proposition $A\big(u\langle v\rangle\big)$ is deduced in a single step by a rule $A \to v$. Then $u\langle v\rangle \in \big[\varphi^1(\varnothing,\dots,\varnothing)\big]_A$.

Induction step. Assume that a proposition $A\big(u\langle v\rangle\big)$ is deduced in p steps, and consider the rule used at the last step of deduction:

$$A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \dots \,\&\, \lhd\beta_m \,\&\, \lhdeq\gamma_1 \,\&\, \dots \,\&\, \lhdeq\gamma_{m'}.$$

Then,

- for each conjunct $\alpha_i = X_{i,1}\dots X_{i,\ell_i}$ in the rule, there is a partition $v = v_1\dots v_{\ell_i}$, so that $\vdash_G X_{i,j}\big(uv_1\dots v_{j-1}\langle v_j\rangle\big)$ for all $j \in \{1,\dots,\ell_i\}$;
- for each conjunct $\lhd\beta_i = \lhd Y_{i,1}\dots Y_{i,\ell'_i}$, there is a partition $u = u_1\dots u_{\ell'_i}$, for which $\vdash_G Y_{i,j}\big(u_1\dots u_{j-1}\langle u_j\rangle\big)$ for all $j \in \{1,\dots,\ell'_i\}$;
- for each conjunct $\lhdeq\gamma_i = \lhdeq Z_{i,1}\dots Z_{i,\ell''_i}$ there is a partition $uv = w_1\dots w_{\ell''_i}$, satisfying $\vdash_G Z_{i,j}\big(w_1\dots w_{j-1}\langle w_j\rangle\big)$ for all $j \in \{1,\dots,\ell''_i\}$.

Consider any base conjunct $\alpha_i = X_{i,1}\dots X_{i,\ell_i}$, with $X_{i,j} \in \Sigma \cup N$, and its j-th symbol $X_{i,j}$. If $X_{i,j} = B \in N$, then the deduction of the proposition $X_{i,j}\big(uv_1\dots v_{j-1}\langle v_j\rangle\big)$ takes fewer than p steps, and therefore, by the induction hypothesis, there exists such a number $t_{i,j}$, that $uv_1\dots v_{j-1}\langle v_j\rangle \in B(\varphi^{t_{i,j}}(\bot))$. If $X_{i,j} = a \in \Sigma$, then $v_j = a$ and $uv_1\dots v_{j-1}\langle v_j\rangle = uv_1\dots v_{j-1}\langle a\rangle \in \Sigma^*\langle a\rangle$. The concatenation of these strings with contexts $u\langle v_1\rangle$, $uv_1\langle v_2\rangle$, …, up to $uv_1\dots v_{j-1}\langle v_j\rangle$, is $u\langle v\rangle$, and therefore $u\langle v\rangle \in \alpha_i(\varphi^{t_i}(\bot))$, where $t_i = \max_j t_{i,j}$.

Applying the same procedure to each conjunct $\lhd\beta_i$ or $\lhdeq\gamma_i$ similarly yields $u\langle v\rangle \in \lhd\beta_i(\varphi^{t'_i}(\bot))$ or $u\langle v\rangle \in \lhdeq\gamma_i(\varphi^{t''_i}(\bot))$, for some appropriate numbers t'_i and t''_i. Let t be the maximum of all these numbers. Then,

$$u\langle v\rangle \in \bigcap_{i=1}^{k} \alpha_i(\varphi^t(\bot)) \cap \bigcap_{i=1}^{m} \lhd\beta_i(\varphi^t(\bot)) \cap \bigcap_{i=1}^{m'} \lhdeq\gamma_i(\varphi^t(\bot)) \subseteq \big[\varphi^{t+1}(\bot)\big]_A,$$

as desired. $\square$

2.4.3 Definition of grammars with two-sided contexts by language equations

For grammars with two-sided contexts, the model uses another generalization of language equations, in which the unknowns are accordingly *sets of triples* $u\langle w\rangle v$.

Definition 2.9. *Let $K, L \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ be languages of triples and define the following operations on them:*

- *union* $K \cup L = \{\, u\langle w\rangle v \mid u\langle w\rangle v \in K \text{ or } u\langle w\rangle v \in L \,\}$;
- *intersection* $K \cap L = \{\, u\langle w\rangle v \mid u\langle w\rangle v \in K,\ u\langle w\rangle v \in L \,\}$;
- *concatenation* $K \cdot L = \{\, u\langle ww'\rangle v \mid u\langle w\rangle w'v \in K,\ uw\langle w'\rangle v \in L \,\}$;
- *left context* $\lhd L = \{\, u\langle w\rangle v \mid \varepsilon\langle u\rangle wv \in L \,\}$;
- *extended left context* $\trianglelefteq L = \{\, u\langle w\rangle v \mid \varepsilon\langle uw\rangle v \in L \,\}$;
- *right context* $\rhd L = \{\, u\langle w\rangle v \mid uw\langle v\rangle\varepsilon \in L \,\}$;
- *extended right context* $\trianglerighteq L = \{\, u\langle w\rangle v \mid u\langle wv\rangle\varepsilon \in L \,\}$.

For every grammar with two-sided contexts $G = (\Sigma, N, R, S)$, is a system of equations in variables N, in which each variable assumes a value of a language of triples $L \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$, and which contains the following equation for every variable A.

$$A = \bigcup_{\substack{A\to\alpha_1 \,\&\ldots\&\, \alpha_k \,\&\\ \&\, \lhd\beta_1 \,\&\ldots\&\, \lhd\beta_m \,\&\\ \&\, \trianglelefteq\gamma_1 \,\&\ldots\&\, \trianglelefteq\gamma_n \,\&\\ \&\, \trianglerighteq\kappa_1 \,\&\ldots\&\, \trianglerighteq\kappa_{m'} \,\&\\ \&\, \rhd\delta_1 \,\&\ldots\&\, \rhd\delta_{n'} \in R}} \Bigg(\bigcap_{i=1}^{k} \alpha_i \;\cap\; \bigcap_{i=1}^{m} \lhd\beta_i \;\cap\; \bigcap_{i=1}^{n} \trianglelefteq\gamma_i \;\cap\; \bigcap_{i=1}^{m'} \trianglerighteq\kappa_i \;\cap\; \bigcap_{i=1}^{n'} \rhd\delta_i \Bigg).$$

Similarly to the case of grammars with one-sided contexts, each instance of a symbol $a \in \Sigma$ in such a system denotes the language $\{\, x\langle a\rangle y \mid x, y \in \Sigma^* \,\}$, and each empty string denotes the language $\{\, x\langle\varepsilon\rangle y \mid x, y \in \Sigma^* \,\}$. A solution of such a system is a vector of languages $(\ldots, L_A, \ldots)_{A\in N}$, such that the substitution of L_A for A, for all $A \in N$, turns each equation into an equality.

Again, similarly to the one-sided case, such a system has a least solution and the following result holds.

Theorem 2.2. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts, where $N = \{A_1, \ldots, A_n\}$, and let $A_i = \varphi_i(A_1, \ldots, A_n)$ $(1 \leqslant i \leqslant n)$ be the associated system of language equations. Let $u\langle w\rangle v \in \Sigma^* \times \Sigma^* \times \Sigma^*$ be a string with a context. Then, for every $A \in N$,*

$$u\langle w\rangle v \in \Big[\bigsqcup_{t\geqslant 0} \varphi^t(\varnothing, \ldots, \varnothing)\Big]_A \quad \textit{if and only if} \quad \vdash_G A\big(u\langle w\rangle v\big).$$

Chapter 3

Raison d'être: specifying a programming language

Since the era of Algol 60, the specification of syntax of programming languages has been based on different grammatical models, mainly on context-free grammars and so called Backus-Naur Form [49]. For educational purposes Wirth, the author of the Pascal programming language, has used syntactic diagrams [30] in descriptions of languages.

These models allow one to define *syntactic structure* of a language, including, for example, correctness of parentheses in nested expressions, or of every `if` statement having at most one `else` branch, and so on. However, such natural restrictions as declaration of every variable before use or agreement of the number of formal parameters and actual arguments in a function call, not to mention the agreement of their types, can not be described by context-free grammars.

In 1962, Floyd has shown [17] that Algol is not a context-free language by considering the following valid program.

$$\texttt{begin real } \underbrace{\texttt{x}\ldots\texttt{x}}_{n} \texttt{ ; } \underbrace{\texttt{x}\ldots\texttt{x}}_{n} \texttt{ := } \underbrace{\texttt{x}\ldots\texttt{x}}_{n} \texttt{ end}$$

Under a certain encoding, one can represent such programs as strings of the form $a^n b^n c^n$, with $n \geqslant 1$, which is a well-known non-context-free language.

As a matter of fact, the requirement that every identifier has to be declared before use, can be abstracted by the language $\{\, wcw \mid w \in \Sigma^*, c \notin \Sigma \,\}$, called sometimes the copy language with central marker. In its turn, the language $\{\, a^n b^m c^n d^m \mid n, m \geqslant 1 \,\}$ abstracts the agreement of the number of formal parameters and actual arguments in function calls (but not the agreement of their types, though). Both languages are known to be non-context-free.

Context-sensitive grammars, initially proposed by Chomsky to describe the syntax of natural languages, turned out to be powerful enough to define

the mentioned abstract languages; however, they are equivalent to nondeterministic linear-bounded Turing machines, which essentially means that no efficient parsing algorithm exists for such "grammars".

These issues have inspired a subsequent research to develop a reasonable (that is, efficiently parsable) grammatical model, which would be able to specify the mentioned abstract languages. Of many such attempts, one could emphasize indexed grammars by Aho [1] and two-level grammars by van Wijngaarden [93, 10]. Unfortunately, one has never found a polynomial parsing algorithm for the former model, while the latter has been shown computationally universal, making it practically irrelevant. Nonetheless, two-level grammars have been used in specification of Algol 68 programming language [93], with all "context conditions" being defined formally.

One can notice that the language $\{ a^n b^m c^n d^m \mid n, m \geqslant 1 \}$ can be represented as an intersection of two context-free languages, which means, in addition, that it can be parsed in polynomial time by invoking the appropriate context-free grammar parsing algorithm. A generalization of this idea led to the *Boolean closure of context-free languages*, for which polynomial-time parsing algorithms have been developed (see, for example, the paper by Heilbrunner and Schmitz [25]). However, as it has been later shown by Wotschke [95], the above-mentioned copy language with central marker is not representable as a finite intersection of context-free languages, and thus some other apt formalisms had to be discovered.

Meanwhile, the development of fast polynomial time parsing algorithms for subclasses of context-free grammars [40, 41], as well as the need to specify the syntax of programming languages by models as natural as context-free grammars, led to research on attribute grammars [42] and syntax-directed translation [2]. Every rule is associated with semantic actions, which, for example, may use symbol tables to look-up whether a certain variable has been previously declared. Such essentially *ad hoc* techniques are used even until now.

Apparently the first specification of a programming language entirely by a (polynomially parsable) formal grammar has been made by Okhotin [58], who constructed a Boolean grammar for a certain simple programming language. That language, however, had only one data type and hence no approach of how to implement type checking has been suggested.

This chapter specifies a very similar language named *Kieli* (Finnish for "language"), augments it with two data types (*integer* and *Boolean*), and uses grammars with contexts to specify the language in a formal way.

The grammar takes its ideas from Example 2.4, which abstracts declaration of identifiers (represented in unary notation) before their use. To generalize this example to identifiers over a many-letter alphabet, the technique for matching the symbols in a declaration and a reference (see nonterminal F) shall be much more involved than just checking the number of the

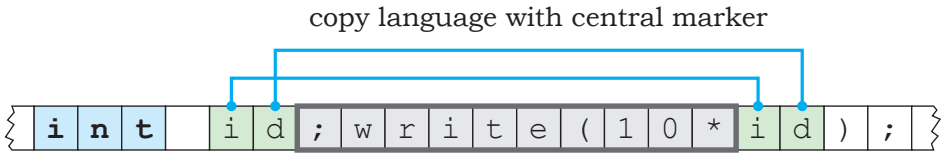


Figure 3.1: How the language $\{ wcw \mid w \in \Sigma^*, c \notin \Sigma \}$ abstracts the problem of matching identifiers.

appropriate a's and b's.

Example 3.1 ([50])**.** The following conjunctive grammar generates the language $\{ wcw \mid w \in \{a, b\}^* \}$.

$$
\begin{aligned}
S &\to C \,\&\, D \\
A &\to XAX \mid cEa \\
B &\to XBX \mid cEb \\
X &\to a \mid b \\
C &\to XCX \mid c \\
D &\to aA \,\&\, aD \mid bB \,\&\, bD \mid cE \\
E &\to XE \mid \varepsilon
\end{aligned}
$$

The nonterminal C defines the strings consisting of two parts of equal length separated by a central marker, that is, $L(C) = \{ w_1 c w_2 \mid w_1, w_2 \in \{a, b\}^*, |w_1| = |w_2| \}$. The actual comparison of symbols in corresponding positions is made by nonterminal D, which defines the language $L(D) = \{ uczu \mid u, z \in \{a, b\}^* \}$. The conjunction of C and D ensures that the strings generated by the grammar are of the desired form.

The language $L(D)$ is defined inductively as follows. The base case gives strings of the form $c\{a, b\}^*$, that is, with $u = \varepsilon$. The induction ensures that a string is in $L(D)$ if and only if its first symbol is the same as the corresponding symbol on the other side, and the string without this first symbol is also in $L(D)$.

A similar language shall be used later in Section 3.2, when a grammar for a programming language is defined.

3.1 Definition of the language *Kieli*

The complete definition of the programming language *Kieli* is presented below.

[I] Lexical conventions.

[I.1] *Alphabet of the language.* A program is a string over the following alphabet of: *letters* (`a`, ..., `z`), *digits* (`0`, ..., `9`), *punctuators* (`(`, `)`, {, }, `+`, `-`, `*`, `/`, `&`, `|`, `!`, `=`, `<`, `>`, `,`, `;`), and a *whitespace character* (␣).

[I.2] *Lexemes*, the set of the following strings over the alphabet:

[I.2.1] *Keywords*: `int`, `bool`, `if`, `else`, `while`, `return`, `void`, `true`, `false`.

[I.2.2] *Identifiers*:

[I.2.2.1] Identifier is a nonempty sequence of letters and digits, starting from a letter.

[I.2.2.2] Identifier should not coincide with any of the keywords.

[I.2.3] *Number* is a nonempty sequence of digits.

[II] Syntactic structure.

[II.1] A *program* is a finite sequence of function declarations, containing exactly one main function.

[II.2] *Function:*

[II.2.1] For every identifier x, not declared before as a name of a parameter of the current function, `int` x and `bool` x are *declarations of a function parameter.*

[II.2.2] For every identifier f, not used as a name of a function before, for every keyword $t \in \{\texttt{int}, \texttt{bool}, \texttt{void}\}$, and for every set a_1, ..., a_n (with $n \geqslant 0$) of declarations of a function parameter, t f `(` a_1 `,` ... `,` a_n `)` is a *function header.* The identifier f is the *name* of the function, a_1, ..., a_n are *parameters* of the function, and t is the *type* of the function.

[II.2.3] For every identifier x, `int main (` `int` x `)` is a *main function header.*

[II.2.4] For every function header F and for every properly returning statement s, $F\{s\}$ is a *function description.*

[II.3] *Expression:*

[II.3.1] *Integer expression:*

[II.3.1.1] A number is an integer expression.

[II.3.1.2] An identifier x, declared either in an integer variable declaration or an integer parameter declaration, is an integer expression.

[II.3.1.3] For every integer expression e, `(` e `)` is an integer expression.

[II.3.1.4] For two integer expressions e_1 and e_2 and for binary operator $op \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}\}$, e_1 *op* e_2 is an integer expression.

[II.3.1.5] For every integer expression e and for unary operator $op \in \{\texttt{-}\}$, *op* e is an integer expression.

[II.3.1.6] For every identifier f, declared as a name of a function returning an integer value, and for every set of expressions $e_1, \ldots, e_n$ (with $n \geqslant 0$), the *integer function call* f `(` e_1 `,` $\ldots$ `,` e_n `)` is an integer expression.

- The number of *actual arguments* $e_1, \ldots, e_n$ must coincide with the number of *formal parameters* in the declaration of the function f.
- The types of expressions $e_1, \ldots, e_n$ must be the same as the types of the corresponding formal parameters of the function f.

[II.3.2] *Boolean expression:*

[II.3.2.1] A Boolean value (that is, *true* or *false*) is a Boolean expression.

[II.3.2.2] An identifier x, declared either in a Boolean variable declaration or in a Boolean parameter declaration, is a Boolean expression.

[II.3.2.3] For two integer expressions e_1 and e_2 and for binary operator $op \in \{\texttt{<}, \texttt{>}, \texttt{<=}, \texttt{>=}, \texttt{<>}, \texttt{==}\}$, e_1 *op* e_2 is a Boolean expression.

[II.3.2.4] For two Boolean expressions e_1 and e_2 and for binary operator $op \in \{\texttt{<>}, \texttt{==}\}$, e_1 *op* e_2 is a Boolean expression.

[II.3.2.5] For every Boolean expression e and for unary operator $op \in \{\texttt{!}\}$, *op* e is a Boolean expression.

[II.3.2.6] A *Boolean function call* is defined analogously to an *integer function call*.

[II.4] *Statements:*

[II.4.1] *Declaration statement:* For every set of identifiers $x_1, \ldots, x_n$ (with $n \geqslant 1$), not declared before as names of variables or parameters within the current scope, and for every keyword $t \in \{\texttt{int}, \texttt{bool}\}$, the *variable declaration statement* t x_1 `,` $\ldots$ `,` x_n `;` is a statement.

[II.4.2] For every expression e, the *expression statement* e`;` is a statement.

[II.4.3] A *void function call* is a statement and is defined similarly to an *integer function call*.

[II.4.4] For every identifier x, declared either in an integer (Boolean) variable declaration or in an integer (Boolean, respectively)

parameter declaration, and for every integer (Boolean, respectively) expression e, the *assignment statement* x `=` e `;` is a statement.

[II.4.5] *Conditional statement:* For every Boolean expression e, and for all properly returning statements s_1 and s_2, the *if-statement* `if (` e `)` $\{s_1\}$ and the *if-then-else statement* `if (` e `)` $\{s_1\}$ `else` $\{s_2\}$ are statements.

[II.4.6] *Iteration statement:* For every Boolean expression e and for every properly returning statement s, the *while-statement* `while (` e `)` $\{s\}$ is a statement.

[II.4.7] For every sequence of statements $s_1, \ldots, s_n$ (with $n \geqslant 0$), the *compound statement* $\{s_1 s_2 \ldots s_n\}$ is a statement.

[II.5] *Properly returning statements:*

[II.5.1] For every expression e, the *return statement* `return` e `;` is a *properly returning statement*. The type of the expression e must be the same as the type of the function, within which the return statement is used.

[II.5.2] Every compound statement s, such that the last of the statements inside is a properly returning statement, or a conditional statement with an `else` clause, such that each of the alternatives is properly returning statement, is a properly returning statement.

[II.5.3] Every statement within a void function is a *properly returning statement*.

[II.6] *Scopes of visibility*:

[II.6.1] The scope of visibility of a formal parameter of a function is the body of that function.

[II.6.2] The scope of a variable declaration located within a compound statement covers all statements in this compound statement, starting from the declaration statement itself.

[II.6.3] The scopes of visibility for any two identifiers of the same kind sharing the same name must be disjoint.

3.2 Grammar with contexts for *Kieli*

Verbatim to the specification, a program written in the language *Kieli* is a sequence of function declarations, and it should have exactly one main function.

$S \rightarrow$ **Functions MainFunction Functions**
Functions $\rightarrow$ **Function Functions** | ε

A function declaration consists of a function header followed by a set of statements.

Function $\rightarrow$
    FunctionHeaderDeclaration
    tOpenBrace **Statement** tCloseBrace
MainFunction $\rightarrow$
    MainFunctionHeaderDeclaration
    tOpenBrace **Statement** tCloseBrace
FunctionHeaderDeclaration $\rightarrow$
    IntegerFunctionHeaderDeclaration |
    BooleanFunctionHeaderDeclaration |
    VoidFunctionHeaderDeclaration

Each of the nonterminals **IntegerFunctionHeaderDeclaration**, **BooleanFunctionHeaderDeclaration**, and **VoidFunctionHeaderDeclaration** defines a header of a function, which returns an integer, a Boolean, or a void value, respectively. The syntax of this construct includes a corresponding type keyword (or `void`) followed by a signature of a function, that is, its name and the list of its formal parameters.

IntegerFunctionHeaderDeclaration $\rightarrow$
    KeywordInt ␣ **FunctionSignature**
BooleanFunctionHeaderDeclaration $\rightarrow$
    KeywordBool ␣ **FunctionSignature**
VoidFunctionHeaderDeclaration $\rightarrow$
    KeywordVoid ␣ **FunctionSignature**
FunctionSignature $\rightarrow$
    InvalidFunctionIdentifier tOpenParenthesis
    ParametersOrNoParameters tCloseParenthesis

The name of a function should be unique among the names of functions within a program; this restriction is guaranteed by the nonterminal ***InvalidFunctionIdentifier***, the rules for which shall be explained in what follows.

A function can, in general, have no parameters at all.

ParametersOrNoParameters $\rightarrow$ **Parameters** | ε

In a list of parameters, every parameter is separated from each other by a comma.

Parameters →
    Parameter tComma **Parameters** |
    Parameter

A declaration of a function parameter should specify the type of the parameter and its name.

Parameter → TypeKeyword ␣ Identifier WS

The main function has to return an integer value and have exactly one integer parameter.

MainFunctionHeaderDeclaration →
    KeywordInt ␣ `m a i n` WS tOpenParenthesis
        KeywordInt ␣ Identifier tCloseParenthesis

A body of every function contains (a possibly empty) set of statements.

StatementsOrNoStatements → **Statements** | ε
Statements →
    Statement Statements |
    ProperlyReturningStatement

A statement is defined verbatim according to its specification.

Statement →
    VariableDeclaration |
    ExpressionStatement |
    VoidFunctionCall |
    AssignmentStatement |
    IfStatement |
    WhileStatement |
    tOpenBrace **Statements** tCloseBrace

The keywords and the white space are defined as follows.

KeywordInt → `i n t` WS
⋮
KeywordFalse → `f a l s e` WS
TypeKeyword → KeywordInt | KeywordBool
WS → ␣ WS | ε

An identifier is defined exactly to its specification.

Identifier → Letter LettersOrDigits WS & NotKeyword WS
NotKeyword → NotKeywordInt & ... & NotKeywordFalse

Every nonterminal NotKeywordInt, ..., NotKeywordFalse defines a corresponding regular language $\Sigma^* \setminus \{\texttt{int}\}$, ..., $\Sigma^* \setminus \{\texttt{false}\}$.

A declaration of variables starts with the keyword specifying their type, followed by a list of "invalid" identifiers.

VariableDeclaration →
    TypeKeyword ␣ **InvalidIdentifiers**
    InvalidIdentifier tSemicolon
InvalidIdentifiers →
    InvalidIdentifier tComma **InvalidIdentifiers** | ε

The rules for the nonterminal ***InvalidIdentifier*** shall be explained later.

Expressions can be either integer or Boolean.

Expression → **IntegerExpression** | **BooleanExpression**

Integer and Boolean expressions are defined inductively on their structure, as follows.

IntegerExpression →
    IntegerConstant |
    ValidIntegerIdentifier |
    tOpenParenthesis **IntegerExpression**
        tCloseParenthesis |
    IntegerExpression IntegerBinaryOperation
        IntegerExpression |
    IntegerUnaryOperation **IntegerExpression** |
    IntegerFunctionCall
BooleanExpression →
    KeywordTrue | KeywordFalse |
    ValidBooleanIdentifier |
    tOpenParenthesis **BooleanExpression**
        tCloseParenthesis |
    BooleanExpression BooleanBinaryOperation
        BooleanExpression |
    IntegerExpression BooleanBinaryOperation
        IntegerExpression |
    BooleanUnaryOperation **BooleanExpression** |
    BooleanFunctionCall

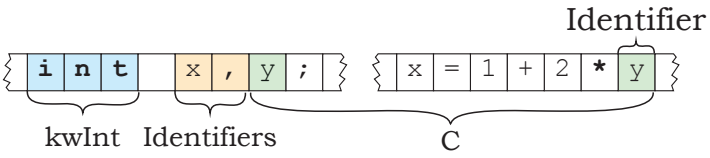


Figure 3.2: How the nonterminal ***ValidIntegerIdentifier*** works.

The nonterminal ***ValidIntegerIdentifier*** (***ValidBooleanIdentifier***) defines an identifier that has been previously declared either as an integer (Boolean, respectively) parameter of a function or as an integer (Boolean, respectively) variable inside a function, with the scopes of visibility respected.

The condition of being previously declared can be naturally expressed by an extended left context.

ValidIntegerIdentifier → Identifier
& ⩤ **Functions** *anything-except-function-header*
KeywordInt ␣ **Identifiers** C
ValidBooleanIdentifier → Identifier
& ⩤ **Functions** *anything-except-function-header*
KeywordBool ␣ **Identifiers** C
Identifiers → Identifier tComma **Identifiers** | ε

Nonterminals, written in *slanted font* (for example, *anything*), define simple regular languages that speak for themselves. That is why rules for such nonterminals are omitted here.

First, the nonterminal **Functions** "skips" the prefix of the program until a declaration of some function starts. Then, the keyword `int` (or `bool`) and a list of identifiers is matched.

Finally, the nonterminal C [58] is used to compare the identifier in applicative use to the identifier in declarative use.

C → CLEN & CITERATE | C WS
CLEN →
LetterOrDigit CLEN LetterOrDigit |
LetterOrDigit CMID LetterOrDigit
CA →
LetterOrDigit CA LetterOrDigit |
`a` LettersOrDigits CMID
⋮
C9 → LetterOrDigit C9 LetterOrDigit | `9` LettersOrDigits CMID

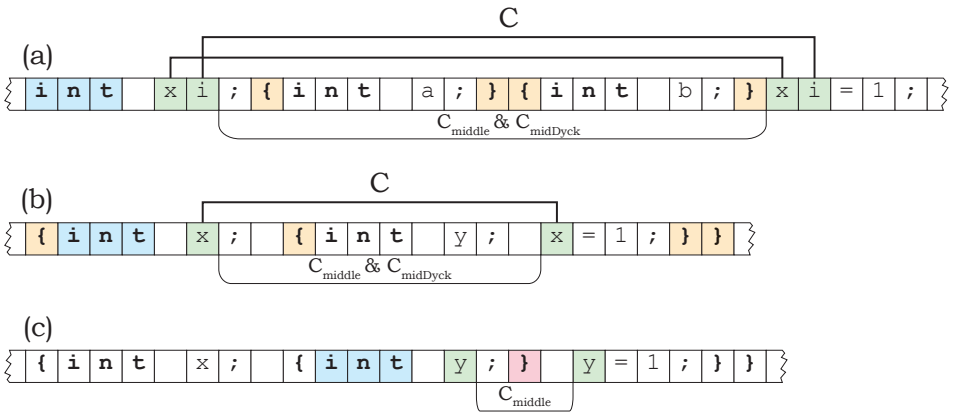


Figure 3.3: How nonterminal C compares names of identifiers and scopes of visibility: *(a)*, *(b)* valid strings; *(c)* invalid string.

CITERATE →
CA `a` & CITERATE `a` | ⋯ | CZ `z` & CITERATE `z` |
C0 `0` & CITERATE `0` | ⋯ | C9 `9` & CITERATE `9` |
LettersOrDigits CMID
CMID → CMIDDLE & CMIDDYCK
CMIDDLE → *any-punctuator any-string any-punctuator* |
␣ *any-string any-punctuator* | *any-punctuator any-string* ␣ |
␣ *any-string* ␣ | *any-punctuator* | ␣

The scopes of visibility are checked by the nonterminal CMIDDYCK, which defines the language of such strings $w \in \Sigma^*$, that braces in w are balanced, with possibly some open braces left unmatched.

CMIDDYCK →
anything-except-braces |
CMIDDYCK *anything-except-braces* CMIDDYCK |
tOpenBrace *anything-except-braces* tCloseBrace |
tOpenBrace *anything-except-braces* CMIDDYCK
anything-except-braces tCloseBrace |
tOpenBrace

This allows one to correctly process scopes of visibility, as shown in Figure 3.3.

Now when the nonterminals ***ValidIntegerIdentifier*** and ***ValidBooleanIdentifier*** have been defined, the rule for the nonterminal

InvalidIdentifier can be given.

```
InvalidIdentifier → Identifier
        & ⩽ Functions anything-except-function-header
                TypeKeyword ␣ Identifiers D,
```

Here the nonterminal D defines the conjunctive language of strings of the form $w_1x_1w_2x_2\ldots w_\ell x_\ell w_{\ell+1}$, such that every x_i starts and ends with anything but letters or digits **and**

- either all w_i are different,
- *or* some of them are equal but at least one x_i contains more "}"-braces than "{"-braces.

Though the negation is not defined within the formalism of grammars with contexts, its use might be particularly handy for shortness of notation.

```
InvalidIdentifier → Identifier & ¬ ValidIdentifier
ValidIdentifier →
        ValidIntegerIdentifier | ValidBooleanIdentifier
```

Similarly to the nonterminal ***InvalidIdentifier***, one can define the nonterminal ***InvalidFunctionIdentifier***.

```
InvalidFunctionIdentifier → Identifier & ⩽ Functions D
```

Calls to functions require checking the name of the function in question, as well as agreement of formal parameters and actual arguments types.

```
ValidFunctionNameAndArguments →
        C tOpenParenthesis Expressions tCloseParenthesis &
        Identifier tOpenParenthesis Q tCloseParenthesis
Expressions →
        Expressions tComma Expression |
        Expression
IntegerFunctionCall →
        Identifier tOpenParenthesis
                Expressions tCloseParenthesis
        & ⩽ Functions KeywordInt ␣
                ValidFunctionNameAndArguments
BooleanFunctionCall →
        Identifier tOpenParenthesis
                Expressions tCloseParenthesis
```

& ⩤ **Functions** KeywordBool ␣
ValidFunctionNameAndArguments

The first conjunct in the rule for nonterminal ***ValidFunctionNameAndArguments*** uses nonterminal C to check that the function name has been previously declared. The comparison of formal parameters and actual arguments of a function is done by the second conjunct of that rule, which uses the nonterminal Q, as shown in Figure 3.4. Informally, this nonterminal defines a variation of the copy language with central marker over the "alphabet" {Bool, Int} of types.

Q → QLEN & QITERATE | Q WS
QLEN → TypeKeyword ␣ Identifier QLEN **Expression** |
tComma QLEN tComma | QMID
INTVAROREXPR →
KeywordInt ␣ Identifier |
IntegerExpression
BOOLVAROREXPR →
KeywordBool ␣ Identifier |
BooleanExpression
QX → INTVAROREXPR | BOOLVAROREXPR | tComma
QY → QX QY | ε
QINT → QX QINT QX | INTVAROREXPR QY QMID
QBOOL → QX QBOOL QX | BOOLVAROREXPR QY QMID
QCOMMA → QX QCOMMA QX | tComma QY QMID
QITERATE →
QINT INTVAROREXPR &
QITERATE INTVAROREXPR |
QBOOL BOOLVAROREXPR &
QITERATE BOOLVAROREXPR |
QCOMMA tComma & QITERATE tComma |
QY QMID
QMID → tCloseParenthesis *anything* tOpenParenthesis

An expression statement is defined as an expression followed by a semicolon.

ExpressionStatement → **Expression** WS tSemicolon

A call to a void function is defined in a way similar to integer and Boolean function calls.

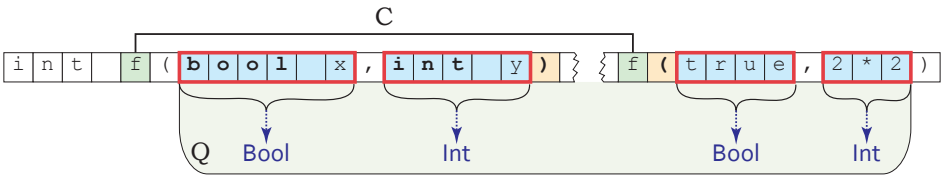


Figure 3.4: How the agreement of formal parameters and actual arguments in a function call is checked by nonterminal ***ValidFunctionNameAndArguments***.

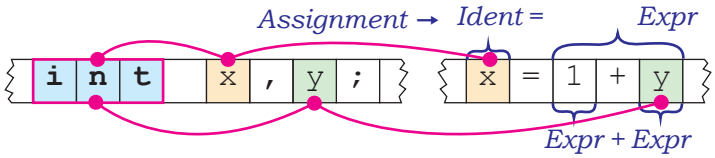


Figure 3.5: How the nonterminal **AssignmentStatement** works.

VoidFunctionCall →
Identifier tOpenParenthesis
Expressions tCloseParenthesis
& ◁ **Functions** KeywordVoid ␣
ValidFunctionNameAndArguments

An assignment statement respects the types of the variable being assigned and of the expression in the right-hand side part of the statement.

AssignmentStatement →
ValidIntegerIdentifier tEqual
IntegerExpression tSemicolon |
ValidBooleanIdentifier tEqual
BooleanExpression tSemicolon

Now conditional and iteration statements can be defined in a natural way, explicitly stating that the conditions should be of Boolean type.

IfStatement →
KeywordIf tOpenParenthesis
BooleanExpression WS tCloseParenthesis
tOpenBrace **Statements** tCloseBrace
OptionalElse WS

OptionalElse →
KeywordElse tOpenBrace **Statements**
tCloseBrace WS | ε
WhileStatement →
KeywordWhile tOpenParenthesis
BooleanExpression WS tCloseParenthesis
tOpenBrace **Statements** tCloseBrace WS

Every compound statement in a program must be properly returning. If the function has been declared as of integer or Boolean type, its body must explicitly return a value in a return statement.

ProperlyReturningStatement → KeywordReturn
IntegerExpression WS tSemicolon
& $\lhd$ **Functions** KeywordInt
anything-except-function-header
ProperlyReturningStatement → KeywordReturn
BooleanExpression WS tSemicolon
& $\lhd$ **Functions** KeywordBool
anything-except-function-header

If the function has been declared as void, no explicit return statement is needed, and every statement can be deemed as properly returning.

ProperlyReturningStatement →
Statement
& $\lhd$ **Functions** KeywordVoid
anything-except-function-header

Thus, an application of grammars with contexts to the formal definition of the syntax of programming languages has been studied. Type checking is carried out apparently for the first time by a formal grammar constructed along the lines of this chapter. It is worth noticing that if all context operators *were* removed from the grammar, it would still define the "syntactic structure" of programs, with no "semantic checks" done.

A direction for further research is to study constructions of programming languages [80] which can be formally expressed by grammars with contexts. Thus, for example, already in the grammar constructed in this chapter, the nonterminal Q could consider two methods of passing parameters – by reference and by value. In the former case, the argument, corresponding to a parameter passed by reference, should be a variable. Another requirement could be that every variable should be initialized. Such a constraint can be expressed using grammars with right contexts. Thus, when a variable

is being declared, one can state that "*there will be later* an assignment to an identifier with the same name". Yet another use of the right contexts is feasible when processing forward subroutine declarations. When a signature of a function is being declared, it could be checked that "*there will be later* a body of a function with the same name and the same list of formal parameters".

Part II

Parsing algorithms

Chapter 4

Tabular parsing

A *parsing algorithm* determines whether a given string is generated by a grammar (given or fixed), and if it does, constructs a parse tree of that string. The existence of parsing algorithms of low complexity is a crucial property for any family of formal grammars. The basic parsing algorithm for ordinary context-free grammars is the Cocke–Kasami–Younger algorithm, which determines the membership of all substrings of the input string in the languages generated by the nonterminals of the grammar, and stores the results in a table; a parse tree can then be constructed on the basis of the table. The algorithm works in time $\mathcal{O}(n^3)$, where n is the length of the input string. The Cocke–Kasami–Younger algorithm has earlier been extended to conjunctive grammars [50], with very few differences to the ordinary case.

This chapter presents a further extension of this algorithm, applicable to grammars with contexts. Though the differences from the original Cocke–Kasami–Younger algorithm are more substantial this time, the running time of the new algorithm is still *cubic* for *grammars with one-sided contexts*. In case of *grammars with two-sided contexts*, the running time of the algorithm is bounded by $\mathcal{O}(n^4)$.

The Cocke–Kasami–Younger algorithm for context-free grammars requires a grammar to be in *Chomsky (binary) normal form*, where every rule is either of the form $A \to BC$ (with $B, C \in N$) or $A \to a$ (with $a \in \Sigma$). The extension of binary normal form for conjunctive grammars [50] allows rules $A \to B_1C_1 \,\&\, \ldots \,\&\, B_kC_k$ with multiple conjuncts. For grammars with left contexts, the extension of the Chomsky normal form is defined as follows.

Definition 4.1. *A grammar with one-sided contexts $G = (\Sigma, N, R, S)$ is said to be in binary normal form, if each rule in R is of one of the following two forms.*

$$A \to B_1C_1 \,\&\, \ldots \,\&\, B_kC_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \tag{4.1a}$$

$$A \to a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \tag{4.1b}$$

(where $k \geqslant 1$, $m, n \geqslant 0$, $A, B_i, C_i, D_i, E_i \in N$, $a \in \Sigma$)

This normal form can be further generalized to grammars with two-sided contexts in a straightforward way.

Definition 4.2. *A grammar with two-sided contexts $G = (\Sigma, N, R, S)$ is said to be in binary normal form, if each rule in R is of one of the forms*

$$\begin{aligned} A \;\to\; & B_1C_1 \,\&\, \ldots \,\&\, B_kC_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \,\&\, \\ & \qquad \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{n'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m'} \\ A \;\to\; & a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \,\&\, \\ & \qquad \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{n'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m'}, \end{aligned}$$

where $k \geqslant 1$, $m, n, n', m' \geqslant 0$, $A, B_i, C_i, D_i, E_i, F_i, H_i \in N$, $a \in \Sigma$.

As will be shown later in Theorem 4.5, every grammar with one-sided contexts can be effectively transformed to a grammar in binary normal form. A similar result holds also for grammars with two-sided contexts, as stated by Theorem 4.6.

4.1 Parsing grammars with one-sided contexts

Let $G = (\Sigma, N, R, S)$ be a grammar with one-sided contexts in binary normal form, and let $w = a_1 \ldots a_n \in \Sigma^+$, with $n \geqslant 1$ and $a_i \in \Sigma$, be an input string to be parsed. For every two positions i, j, with $0 \leqslant i < j \leqslant n$, let

$$T_{i,j} = \{\, A \mid A \in N,\ \vdash_G A\big(a_1 \ldots a_i \langle a_{i+1} \ldots a_j \rangle\big) \,\}$$

be the set of nonterminals generating the substring of w delimited by these two positions. In particular, the string w is in $L(G)$ if and only if $S \in T_{0,n}$.

For context-free grammars, including their conjunctive variant, each entry $T_{i,j}$ of the parsing table logically depends upon all entries $T_{k,\ell}$ with $i \leqslant k < \ell \leqslant j$ and $\ell - k < j - i$. A direct dependence by a rule $A \to BC$, where the membership of B in $T_{i,k}$ and the membership of C in $T_{k,j}$ together imply that $A \in T_{i,j}$, is illustrated in Figure 4.1(a). Accordingly, all sets $T_{i,j}$ can be constructed inductively on the length $j - i$ of the corresponding substrings, that is, beginning with sets corresponding to shorter substrings and continuing with longer substrings.

For grammars with left contexts, each set $T_{i,j}$ additionally depends upon the sets $T_{0,i}$ and $T_{0,j}$, which are needed to evaluate left contexts $\lhd D$ and extended left contexts $\trianglelefteq E$, respectively. The dependencies induced by a rule $A \to BC \,\&\, \lhd D \,\&\, \trianglelefteq E$ are depicted in Figure 4.1(b). While the dependence on $T_{0,i}$ can be handled within the existing inductive approach, by constructing all elements $T_{0,j}, T_{1,j}, \ldots, T_{j-1,j}$ before proceeding to construct any $T_{k,\ell}$

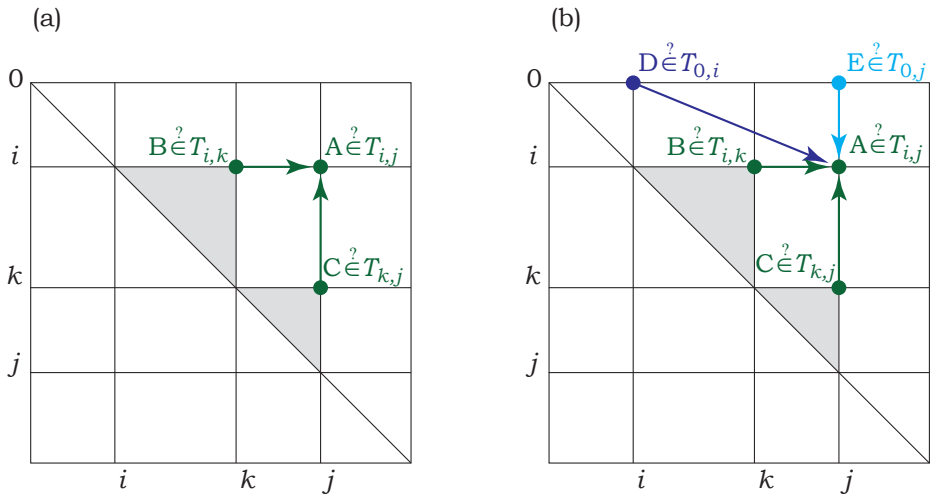


Figure 4.1: How the membership of A in $T_{i,j}$ depends on other data, for rules (a) $A \to BC$ and (b) $A \to BC \,\&\, {\lhd}D \,\&\, {\lessdot}E$.

with $\ell > i$, the dependence of $T_{i,j}$ on $T_{0,j}$ may in its turn form other cyclic dependencies, where the membership of a symbol in $T_{i,j}$ is defined by the membership of another symbol in $T_{0,j}$, which in turn depends on whether some other symbol is in $T_{i,j}$. Consider the following example.

Example 4.1. The following grammar generates a single string $w = ab$.

$$\begin{aligned} S &\to AC \\ A &\to a \\ B &\to b \\ C &\to b \,\&\, {\lessdot}E \\ E &\to AB \end{aligned}$$

This string is deduced as follows.

$$\begin{aligned} \vdash a\big(\varepsilon\langle a\rangle\big) &\qquad (axiom) \\ \vdash b\big(a\langle b\rangle\big) &\qquad (axiom) \\ a\big(\varepsilon\langle a\rangle\big) \vdash A\big(\varepsilon\langle a\rangle\big) &\qquad (A \to a) \\ b\big(a\langle b\rangle\big) \vdash B\big(a\langle b\rangle\big) &\qquad (B \to b) \\ A\big(\varepsilon\langle a\rangle\big), B\big(a\langle b\rangle\big) \vdash E\big(\varepsilon\langle ab\rangle\big) &\qquad (E \to AB) \\ b\big(a\langle b\rangle\big), E\big(\varepsilon\langle ab\rangle\big) \vdash C\big(a\langle b\rangle\big) &\qquad (C \to b \,\&\, {\lessdot}E) \\ A\big(\varepsilon\langle a\rangle\big), C\big(a\langle b\rangle\big) \vdash S\big(\varepsilon\langle ab\rangle\big) &\qquad (S \to AC) \end{aligned}$$

In terms of the table $T_{i,j}$ for the grammar in this example, shown in Figure 4.2, the elements $A \in T_{0,1}$ and $B \in T_{1,2}$ imply that E is in $T_{0,2}$, which in turn implies $C \in T_{1,2}$, and from the latter fact, one can infer that $S \in T_{0,2}$. Thus, an algorithm constructing the sets $T_{i,j}$ for this grammar has to alternate between the entries $T_{0,2}$ and $T_{1,2}$ until all their elements are gradually determined.

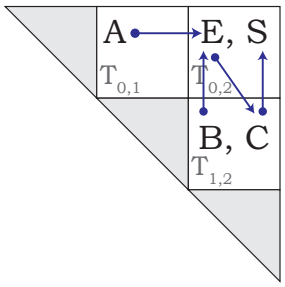


Figure 4.2: Parsing table of string $w = ab$ according to the grammar in Example 4.1.

In general, while calculating the j-th column of the table T, one has to re-calculate all entries $T_{j-1,j}, T_{j-2,j}, \ldots, T_{1,j}, T_{0,j}$ after adding any symbols to the set $T_{0,j}$. Such an iterative calculation is carried out in Algorithm 4.1.

Theorem 4.1. *For every grammar with left contexts G in binary normal form, Algorithm 4.1, given an input string $w = a_1 \ldots a_n$, constructs the sets $T_{i,j}$ and determines the membership of w in $L(G)$, and does so in time $\mathcal{O}(|G|^2 \cdot n^3)$, using space $\mathcal{O}(|G| \cdot n^2)$.*

Proof. The proof of the correctness of the algorithm is comprised of a series of claims that at a certain point in the computation, certain sets $T_{i,j}$ contain certain elements. The top-level statement of this kind is the following invariant of the outer loop: every iteration of this loop completely calculates another column of the table.

Claim 4.1.1. *After the j-th iteration of the loop in line 1, with $j \in \{1, \ldots, n\}$, all sets $T_{i,\ell}$ with $0 \leqslant i < \ell \leqslant j$ are correctly computed, that is, the value of each variable $T_{i,\ell}$ will be $\{ A \in N \mid a_1 \ldots a_i \langle a_{i+1} \ldots a_\ell \rangle \in L_G(A) \}$.*

The proof is by induction on j. Assuming that Claim 4.1.1 holds for the iteration $j-1$, consider the next iteration j. The nested loop in line 2 executes while any new symbols can be added to the set $T_{0,j}$. Its first iteration adds all symbols $A \in N$, for which the proposition $A\big(\varepsilon\langle a_1 \ldots a_j \rangle\big)$ can be deduced without using any extended left contexts. Then the second iteration may use any propositions $E\big(\varepsilon\langle a_1 \ldots a_j \rangle\big)$ obtained in the first iteration, in order to resolve extended left contexts $\lessdot E$, which may eventually lead to other symbols $A' \in N$ being added to $T_{0,j}$, etc.

Algorithm 4.1. Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts in binary normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ (with $n \geqslant 1$ and $a_i \in \Sigma$) be the input string. Let $T_{i,j}$ with $0 \leqslant i < j \leqslant n$ be variables, each representing a subset of N, and let $T_{i,j} = \varnothing$ be their initial values.

1: **for** $j = 1, \dots, n$ **do**
2: **while** $T_{0,j}$ changes **do**
3: **for all** $A \to a \,\&\, \lhd D_1 \,\&\, \dots \,\&\, \lhd D_{m'} \,\&\, \lhdeq E_1 \,\&\, \dots \,\&\, \lhdeq E_{m''} \in R$ **do**
4: **if** $a_j = a$ and $D_1, \dots, D_{m'} \in T_{0,j-1}$ and $E_1, \dots, E_{m''} \in T_{0,j}$ **then**
5: $T_{j-1,j} = T_{j-1,j} \cup \{A\}$
6: **for** $i = j - 2$ to 0 **do**
7: let $P = \varnothing$ $(P \subseteq N \times N)$
8: **for** $k = i + 1$ to $j - 1$ **do**
9: $P = P \cup (T_{i,k} \times T_{k,j})$
10: **for all** $A \to B_1 C_1 \,\&\, \dots \,\&\, B_m C_m \;\&\; \lhd D_1 \,\&\, \dots \,\&\, \lhd D_{m'} \;\&\; \lhdeq E_1 \,\&\, \dots \,\&\, \lhdeq E_{m''} \in R$ **do**
11: **if** $(B_1, C_1), \dots, (B_m, C_m) \in P$, $D_1, \dots, D_{m'} \in T_{0,i}$, and $E_1, \dots, E_{m''} \in T_{0,j}$ **then**
12: $T_{i,j} = T_{i,j} \cup \{A\}$
13: accept if and only if $S \in T_{0,n}$

These dependencies of symbols in $T_{0,j}$ on other symbols in $T_{0,j}$ form a tree of dependencies, and every symbol in $T_{0,j}$ has its height in this tree. For every symbol $A \in N$ that generates the prefix $\varepsilon\langle a_1 \dots a_j\rangle$, define its *height* with respect to this prefix, denoted by $h_j(A)$, as follows. Consider the shortest deduction of $A\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$. If no propositions of the form $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$, with $E \in N$, are ever used as an intermediate statement in the deduction, then let $h_j(A) = 1$. Otherwise, define $h_j(A) = \max_E h_j(E) + 1$, where the maximum is taken over all nonterminals E, for which $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ is used in the deduction. The height is undefined for any nonterminal A with $\varepsilon\langle a_1 \dots a_j\rangle \notin L_G(A)$. If the height of a nonterminal is defined, then it is at most $|N|$.

Iterations of the loop in line 2 correspond to the height of the symbols added to $T_{0,j}$. The following correctness statement ensures that at each iteration, the algorithm determines all symbols A of height greater by 1.

Claim 4.1.2. *Consider the j-th iteration of the loop in line 1. After the h-th iteration of the* while *loop in line 2, each variable $T_{i,j}$ with $0 \leqslant i < j$ is guaranteed to contain all symbols $A \in N$, for which it is possible to deduce the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle\big)$ without using any intermediate statements $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$.*

In particular, $T_{0,j}$ will contain all symbols A with $h_j(A) \leqslant h$.

This claim occurs within the inductive proof of Claim 4.1.1 and is proved by a nested induction on h. The h-th iteration begins with a loop in line 3–5, which is given a completely constructed set $T_{0,j-1}$ and a partially constructed set $T_{0,j}$, where the latter is guaranteed to contain all nonterminals A with $h_j(A) < h$; in particular, in the beginning of the first iteration, the set $T_{0,j}$ is empty. This loop deduces all true propositions $A\big(a_1 \dots a_{j-1}\langle a_j\rangle\big)$, whose deductions do not use any proposition $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$.

The next loop in lines 6–12 similarly handles the deductions of all correct propositions $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle\big)$, with $0 \leqslant i < j - 1$, which can be deduced without referring to any proposition $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$.

Claim 4.1.3. *Consider the j-th iteration of the loop in line 1 and the h-th iteration of the nested* while *loop in line 2. Then, after the iteration i of the loop in line 6, the variable $T_{i,j}$ contains all symbols $A \in N$, for which one can deduce $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle\big)$ without using any intermediate statement $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$.*

Claim 4.1.3 is proved by an induction on i, and its proof is nested within the inductions for Claim 4.1.2 and Claim 4.1.1.

The loop in lines 6–12 begins its execution by computing the set $P = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$. By Claim 4.1.1, each set $T_{i,k}$ was completely constructed at the k-th iteration of the outer loop (Claim 4.1.1 is used here as the induction hypothesis, because $k < j$). Each set $T_{k,j}$ has been updated at the earlier

k-th iteration of the loop in line 6, and, by Claim 4.1.3 (used as the induction hypothesis), it should reflect all deductions that avoid using $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$. Thereby, the set P will contain all pairs (B, C), for which there exists some position k, with $a_1 \dots a_i\langle a_{i+1} \dots a_k\rangle \in L_G(B)$ and $a_1 \dots a_k\langle a_{k+1} \dots a_j\rangle \in L_G(C)$, where $C\big(a_1 \dots a_k\langle a_{k+1} \dots a_j\rangle\big)$ is deduced without using any intermediate statements $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$.

The subsequent inner loop in lines 10–12 uses this set P to determine all nonterminals A, for which $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle\big)$ is deduced without using any intermediate statements $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$. By Claims 4.1.1 and 4.1.2 (both used as induction hypotheses), each lookup to $T_{0,i}$ correctly determines the nonterminals generating $\varepsilon\langle a_1 \dots a_i\rangle$, and each lookup to $T_{0,j}$ returns whether $\varepsilon\langle a_1 \dots a_i\rangle$ is in $L_G(E)$, as long as $h_j(E) < h$. This proves Claim 4.1.3 (for the values of j and h in the ongoing proof).

In order to conclude the proof of Claim 4.1.2, consider the state of the computation after the last iteration of the loop in lines 6–12, which is given by Claim 4.1.3 for $i = 0$. It asserts that $T_{0,j}$ contains all such $A \in N$, that $A\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ is deduced without using any intermediate statements $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) \geqslant h$. Therefore, all A with $h_j(A) \leqslant h$ are in $T_{0,j}$, and Claim 4.1.2 is proved (for the value of j in the current proof).

Once the *while* loop in line 2 terminates after h iterations, by Claim 4.1.2, this means that there exist no such nonterminals A, that the shortest deduction of $A\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ uses any intermediate statements $E\big(\varepsilon\langle a_1 \dots a_j\rangle\big)$ with $h_j(E) = h - 1$, and therefore there are no nonterminals of height h. Since there cannot be any gaps in the height, this indicates that the set $T_{0,j}$ has been constructed completely, and thus proves Claim 4.1.1.

Finally, Claim 4.1.1 asserts that when line 13 is executed, the symbol S is in $T_{0,n}$ if and only if $\varepsilon\langle a_1 \dots a_n\rangle$ is in $L_G(S)$. Therefore, the algorithm accepts exactly the strings in $L(G)$.

It remains to estimate the complexity of the algorithm. From the loops in lines 1, 2, 6 and 8, it is evident that the innermost statement in line 9 is executed $\mathcal{O}(|G| \cdot n^3)$ times. In order to do the calculations in line 9 in time proportional to $|G|$, one can replace the Cartesian product by considering only the conjuncts occurring somewhere in the grammar. The memory requirements of the algorithm are given by the size of the sets $T_{i,j}$. □

Once the table $T_{i,j}$ is calculated for a given input string, and it is verified that this string belongs to the language, its parse tree can be constructed by a recursive procedure *parse*(A, i, j). Its arguments are a symbol $A \in N$ and two positions i and j, and, assuming that $A \in T_{i,j}$, the procedure returns a pointer to the parse tree of $a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle$ from A. In order to avoid re-calculating the same data and constructing identical subtrees, the procedure should employ *memoization* (that is, storing the results of the first computation in memory and then looking it up for every subsequent

call of the same procedure with the same arguments), returning a pointer to an existing subtree whenever applicable. Thus, the procedure is generally the same as in the case of conjunctive grammars [70, Sect. 6].

Example 4.2. Consider the grammar from Example 2.1 generating the language $\{ a^n b^n c^n d^n \mid n \geqslant 0 \}$. As will be shown later in Section 4.4.6, it can be transformed to the following grammar in binary normal form.

$$\begin{aligned}
S &\to S_a D_0 \mid S_b C_0 \\
A &\to A_a B_0 \\
S_a &\to A_0 S \mid a \,\&\, \lessdot A \\
S_b &\to B_0 S \mid b \,\&\, \lessdot A \\
A_a &\to A_0 A \mid a \\
A_0 &\to a \\
B_0 &\to b \\
C_0 &\to c \\
D_0 &\to d
\end{aligned}$$

The parsing table constructed by Algorithm 4.1 on the input $w = aabbccdd$ is given in Figure 4.3. The arrows in the figure indicate the logical dependencies, and the dotted arrow marks the dependence defined by the conjunct $\lessdot A$ in the rule $S_b \to b \,\&\, \lessdot A$.

4.2 Parsing grammars with two-sided contexts

Similarly to the case of grammars with one-sided contexts, for grammars with two-sided contexts there exists a tabular parsing algorithm that determines the membership of the input string.

Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts in binary normal form, and let $w = a_1 \dots a_n \in \Sigma^+$, with $n \geqslant 1$ and $a_i \in \Sigma$, be an input string to be parsed. For every substring of w delimited by two positions i, j, with $0 \leqslant i < j \leqslant n$, consider the set of nonterminal symbols generating this substring.

$$T_{i,j} = \left\{ A \;\middle|\; A \in N,\ a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle a_{j+1} \dots a_n \in L_G(A) \right\}$$

In particular, the whole string w is in $L(G)$ if and only if $S \in T_{0,n}$.

In ordinary context-free grammars, a substring $a_{i+1} \dots a_j$ is generated by A if there is a rule $A \to BC$ and a partition of the substring into $a_{i+1} \dots a_k$ generated by B and $a_{k+1} \dots a_j$ generated by C. Similarly to the case of grammars with one-sided contexts, each set $T_{i,j}$ depends only on the sets

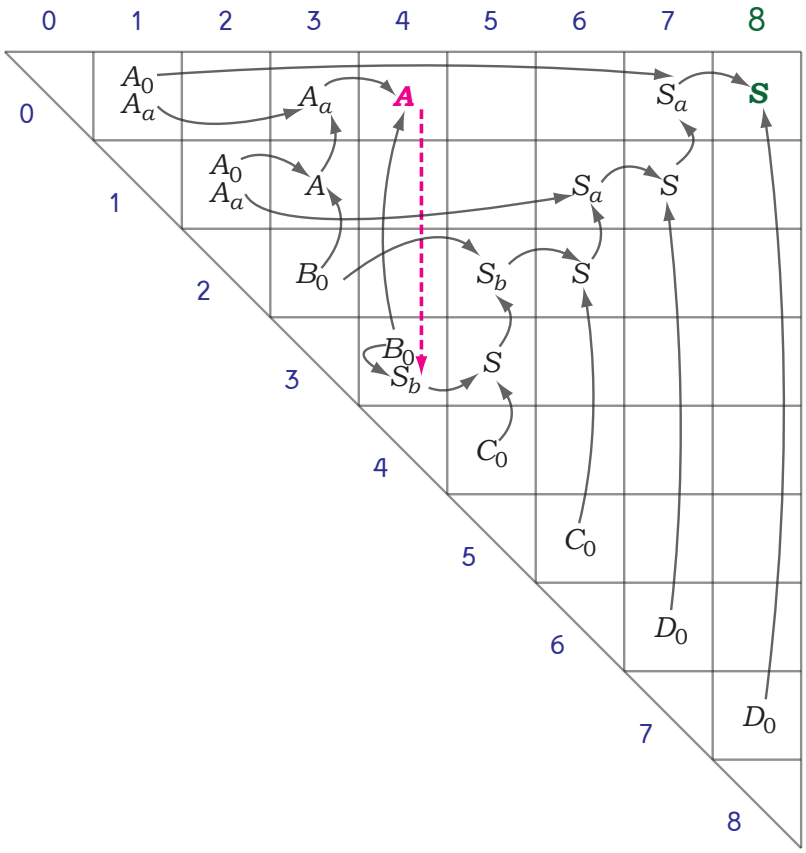


Figure 4.3: Parsing table of string $w = aabbccdd$.

$T_{i',j'}$ with $j' - i' < j - i$, and all these sets may be constructed inductively, beginning with shorter substrings and eventually reaching the set $T_{0,n}$. The more complicated structure of logical dependencies in grammars with two-sided contexts is shown in Figure 4.4. The following example demonstrates how these dependencies may form circles.

Example 4.3. Consider the following grammar.

$$
\begin{aligned}
S &\to AB \\
A &\to a \,\&\, \triangleright B \\
B &\to b \,\&\, \triangleleft C \\
C &\to a
\end{aligned}
$$

Let the input string be $w = ab$. It is immediately seen that $C \in T_{0,1}$.

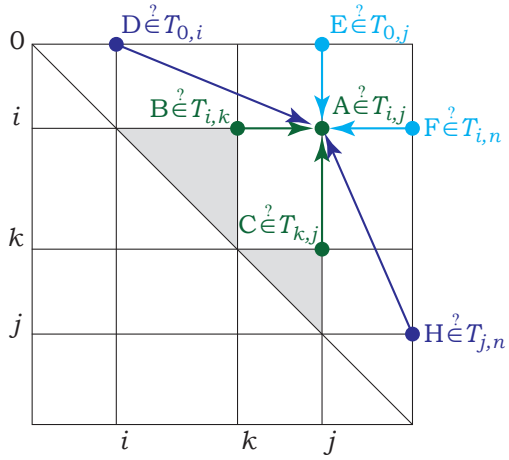


Figure 4.4: How the membership of A in $T_{i,j}$ depends on other data, for a rule $A \to BC \,\&\, {\lhd}D \,\&\, {\trianglelefteq}E \,\&\, {\trianglerighteq}F \,\&\, {\rhd}H$.

From this, one can infer that $B \in T_{1,2}$, and that knowledge can in turn be used to show that $A \in T_{0,1}$. These data imply that $S \in T_{0,2}$. Thus, none of the sets $T_{0,1}$ and $T_{1,2}$ can be fully constructed before approaching the other.

The proposed algorithm for constructing the sets $T_{i,j}$ works as follows. At the first pass, it makes all deductions $\vdash_G A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle a_{j+1} \dots a_n\big)$ that do not involve any contexts, and accordingly puts A to the corresponding $T_{i,j}$. This pass progressively considers longer and longer substrings, as done by the Cocke–Kasami–Younger algorithm for ordinary grammars. During this first pass, some symbols may be added to any sets $T_{0,j}$ and $T_{i,n}$, and thus it becomes known that some contexts are true. This triggers another pass over all entries $T_{i,j}$, from shorter substrings to longer ones, this time using the known true contexts in the deductions. This pass may result in adding more elements to $T_{0,j}$ and $T_{i,n}$, which will require yet another pass, and so on. Since a new pass is needed only if a new element is added to one of $2n-1$ subsets of N, the total number of passes is at most $(2n-1) \cdot |N| + 1$.

These calculations are implemented in Algorithm 4.2, which basically deduces all true statements about all substrings of the input string. For succinctness, the algorithm uses the following notation for multiple context operators. For a set $\mathcal{X} = \{X_1, \dots, X_\ell\}$, with $X_i \in N$, and for an operator $Q \in \{\lhd, \trianglelefteq, \trianglerighteq, \rhd\}$, denote $Q\mathcal{X} := QX_1 \,\&\, \dots \,\&\, QX_\ell$.

Theorem 4.2. *For every grammar with two-sided contexts G in binary normal form, Algorithm 4.2, given an input string $w = a_1 \dots a_n$, constructs the*

Algorithm 4.2. Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts in binary normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ (with $n \geqslant 1$ and $a_i \in \Sigma$) be the input string. Let $T_{i,j}$ with $0 \leqslant i < j \leqslant n$ be variables, each representing a subset of N, and let $T_{i,j} = \varnothing$ be their initial values.

```
1: while any of T_{0,j} (1 ⩽ j ⩽ n) or T_{i,n} (1 ⩽ i < n) change do
2:     for j = 1, ..., n do
3:         for all A → a & ◁D & ⩤E & ⩥F & ▷H ∈ R do
4:             if a_j = a ∧ D ⊆ T_{0,j-1} ∧ E ⊆ T_{0,j} ∧ F ⊆ T_{j-1,n} ∧
               H ⊆ T_{j,n} then
5:                 T_{j-1,j} = T_{j-1,j} ∪ {A}
6:         for i = j − 2 to 0 do
7:             let P = ∅ (P ⊆ N × N)
8:             for k = i + 1 to j − 1 do
9:                 P = P ∪ (T_{i,k} × T_{k,j})
10:            for all A → B_1C_1 & ... & B_mC_m & ◁D & ⩤E & ⩥F & ▷H ∈
               R do
11:                if (B_1, C_1), ..., (B_m, C_m) ∈ P ∧ D ⊆ T_{0,i} ∧ E ⊆
                   T_{0,j} ∧ F ⊆ T_{i,n} ∧ H ⊆ T_{j,n} then
12:                    T_{i,j} = T_{i,j} ∪ {A}
13: accept if and only if S ∈ T_{0,n}
```

sets $T_{i,j}$ and determines the membership of w in $L(G)$, and does so in time $\mathcal{O}(|G|^2 \cdot n^4)$, using space $\mathcal{O}(|G| \cdot n^2)$.

4.2.1 Improvements to the running time

Each pass of Algorithm 4.2 is the same as the entire parsing algorithm for grammars without contexts [50], and that algorithm can be accelerated by changing the order of computing the entries $T_{i,j}$, so that most of the calculations can be offloaded to a procedure for multiplying Boolean matrices [91, 70]. If $\mathrm{BMM}(n)$ is the complexity of multiplying two $n \times n$ Boolean matrices, the resulting algorithm works in time $\mathrm{BMM}(n)$. By the same method, Algorithm 4.2 can be restated to make $\mathcal{O}(n)$ such passes, with the following improvement in running time.

Theorem 4.3. *For every grammar with two-sided contexts G in binary normal form, there is an algorithm to determine whether a given string $w = a_1 \dots a_n$ is in $L(G)$, which works in time $\mathcal{O}(|G|^2 \cdot n \cdot \mathrm{BMM}(n))$, using space $\mathcal{O}(|G| \cdot n^2)$.*

Not long ago, Rabkin [74] developed a more efficient and more sophisticated parsing algorithm for grammars with two-sided contexts, with the running time $\mathcal{O}(|G| \cdot n^3)$, using space $\mathcal{O}(|G| \cdot n^2)$. Like Algorithm 4.2, Rabkin's

algorithm works by proving all true statements about the substrings of the given string, but does so using the superior method of Dowling and Gallier [13]. Nevertheless, Algorithm 4.2 retains some value as the elementary parsing method for grammars with two-sided contexts—just like the Cocke–Kasami–Younger algorithm for ordinary grammars remains useful, in spite of Valiant's asymptotically superior algorithm [91].

4.3 Parsing unambiguous grammars with left contexts

Consider an important property of formal grammars in general: the *syntactic ambiguity*, that is, having multiple parses of a single string. A grammar that defines a unique parse of each string it generates is known as *unambiguous*.

In this section unambiguous grammars with left contexts are considered. The following formal definition of such a grammar is adapted from the case of conjunctive and Boolean grammars [62].

Definition 4.3. *A grammar with one-sided contexts $G = (\Sigma, N, R, S)$ is said to be unambiguous, if it has the following two properties:*

Unambiguous choice of a rule. *For every proposition $A\big(u\langle v\rangle\big)$ with $A \in N$ and $u, v \in \Sigma^*$, there exists a unique rule, by which this proposition is deduced. In other words, different rules generate disjoint languages.*

Unambiguous concatenation. *For every conjunct $X_1 \dots X_\ell$, $\lhd X_1 \dots X_\ell$ or $\lhdeq X_1 \dots X_\ell$ that occurs in any rule, and for all $u, v \in \Sigma^*$, there exists at most one partition $v = v_1 \dots v_\ell$ with $uv_1 \dots v_{i-1}\langle v_i\rangle \in L_G(X_i)$ for all i.*

The grammar in Example 2.1, which generates the language $\{\, a^n b^n c^n d^n \mid n \geqslant 0 \,\}$, is unambiguous. To see that there is no ambiguity of choice, consider that each string in $L_G(S)$ either begins with a and is generated by the rule $S \to aSd$, or begins with b and is generated by $S \to bSc$, or is empty and is generated by the last rule $S \to \varepsilon \mathbin{\&} \lhd A$; hence, the rules for S generate disjoint languages. Similarly, one of the two rules for A generates non-empty strings that begin with a, and the other generates the empty string. Concatenations in this grammar are unambiguous, because all of them are of the form xTy, with $x, y \in \Sigma^*$ and $T \in N$.

On the other hand, the grammar in Example 2.4, representing the language of declarations before use, is ambiguous, because it directly transcribes the definition of the language (2.3), and that definition contains syntactic ambiguity in itself. Whenever for a substring $c^k a$, representing a "reference", there exist multiple earlier "declarations" $u_j = u_{j'} = d^k a$ with $j \neq j'$, the

grammar will allow different parses for these two cases. In terms of Definition 4.3, this constitutes ambiguity in the concatenation EFa in the rule $U \to C \,\&\, \lhd EFa$. The grammar in Example 2.5 has two further instances of ambiguity: every time a "reference" has matching "declarations" before and after it, there is an ambiguity of choice between the rules $S \to US$ and $S \to CS \,\&\, HaE$, and a "reference" followed by multiple "declarations" makes the concatenation HaE ambiguous.

Nevertheless, it is possible to construct unambiguous grammars generating the same languages as in Examples 2.4 and 2.5. Such grammars would need to fix, which of the multiple "declarations" should be matched to each "reference"; for instance, this could be the leftmost "declaration".

For ordinary context-free grammars, besides the cubic-time Cocke–Kasami–Younger algorithm, there also exists a slightly more complicated algorithm by Kasami and Torii [37], which works in time $\mathcal{O}(n^3)$ on every grammar, and in time $\mathcal{O}(n^2)$ for unambiguous grammars. This algorithm was generalized for unambiguous conjunctive grammars [62], and a further generalization for unambiguous grammars with left contexts is presented below.

Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts in binary normal form, let $w = a_1 \dots a_n$ with $n \geqslant 1$ and $a_i \in \Sigma$ be an input string. Note, that the algorithm requires a grammar to be in binary normal form, and, as will be shown later in Section 4.4.1, the transformation of a grammar with left contexts to this normal form preserves unambiguity of the grammar.

The main idea is to construct the same sets $T_{i,j} = \{\, A \mid A \in N,\ \vdash_G A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle\big) \,\}$ as in Algorithm 4.1, representing them in a different data structure: namely, as sets of positions

$$T_j[A] = \big\{\, i \;\big|\; \vdash_G A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle\big) \,\big\},$$

for all j with $1 \leqslant j \leqslant n$ and for all $A \in N$, each stored as a sorted list. Note that $i \in T_j[A]$ is equivalent to $A \in T_{i,j}$, and for each j, the lists $T_j[A]$, for all $A \in N$ contain exactly the same information as the j-th column of the table $T_{i,j}$. The input string $a_1 \dots a_n$ is in $L(G)$ if and only if the position 0 is in $T_n[S]$.

These lists are computed by Algorithm 4.3, which first constructs $T_1[A]$ for all $A \in N$, then all $T_2[A]$, and so on until $T_n[A]$. For each j, the algorithm carries out an iterative calculation while the lists $T_j[A]$ can be modified. However, the order of computation inside each iteration is not the same as in Algorithm 4.1, and this difference accounts for the faster operation on unambiguous grammars.

Theorem 4.4. *For every grammar with left contexts G in binary normal form, Algorithm 4.3 constructs the lists $T_j[A]$ and determines the membership of a given string $w = a_1 \dots a_n$ in $L(G)$, and does so in time $\mathcal{O}(|G|^2 \cdot n^3)$, using*

Algorithm 4.3. Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts in binary normal form. Denote by $\mathcal{P} = \{\, (B, C) \mid A \to BC \,\&\, \ldots \in R \,\}$ the set of all concatenations occurring in the grammar. Let $w = a_1 \ldots a_n \in \Sigma^+$ ($n \geqslant 1$) be an input string. For each $j \in \{1, \ldots, n\}$, let $T_j[A]$ be a variable ranging over the subsets of $\{0, \ldots, j-1\}$. Let P_k range over the subsets of $\mathcal{P}$, for each $k \in \{0, \ldots, n-1\}$.

1: let $T_j[A] = \varnothing$ for all $j = 1, \ldots, n$, $A \in N$
2: **for** $j = 1$ to n **do**
3: **while** 0 can be added to $T_j[A]$ for any $A \in N$ **do**
4: **for all** $A \in N$ **do**
5: **if** $A \to a_j \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{m''} \in R$ **then**
6: **if** $0 \in T_{j-1}[D_1] \cap \ldots \cap T_{j-1}[D_{m'}]$ and $0 \in T_j[E_1] \cap \ldots \cap T_j[E_{m''}]$ **then**
7: $T_j[A] = T_j[A] \cup \{j-1\}$
8: let $P_i = \varnothing$ for all $i \in \{0, \ldots, j-1\}$
9: **for** $k = j-1$ to 1 **do**
10: **for all** $(B, C) \in \mathcal{P}$ **do**
11: **if** $k \in T_j[C]$ **then**
12: **for all** $i \in T_k[B]$ **do**
13: $P_i = P_i \cup \{(B, C)\}$
14: **for all** $A \in N$ **do**
15: **for all** $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{m''} \in R$ **do**
16: **if** $(B_1, C_1), \ldots, (B_m, C_m) \in P_{k-1}$ and $0 \in T_{k-2}[D_1] \cap \ldots \cap T_{k-2}[D_{m'}]$ and $0 \in T_j[E_1] \cap \ldots \cap T_j[E_{m''}]$ **then**
17: $T_j[A] = T_j[A] \cup \{k-1\}$
18: accept if and only if $0 \in T_n[S]$

space $\mathcal{O}(|G|\cdot n^2)$. *If all concatenations in the grammar are unambiguous, then its running time is bounded by* $\mathcal{O}(|G|^2 \cdot n^2)$.

Sketch of a proof. Consider lines 4–17 of Algorithm 4.3, which, mathematically, have exactly the same effect as lines 3–12 of Algorithm 4.1. Each variable $P_i \subseteq N$, with $i \in \{0, \ldots, j-1\}$, is used to gather exactly the same set of nonterminals as the variable P in Algorithm 4.1 at the i-th iteration of its inner loop. There is one difference: Algorithm 4.3 iterates over intermediate positions k in partitions of substrings $a_{i+1} \ldots a_j$ into BC, and then follows the links given in $T_j[C]$ and $T_k[B]$ in order to determine the initial position i, while the above Algorithm 4.1 had a more obvious iteration over the initial positions i, and then a nested loop by k.

The correctness statement of lines 4–17 of Algorithm 4.3 reads as follows.

Claim 4.4.1 (cf. Claim 4.1.3)**.** *Consider the* j*-th iteration of the loop in line 2 and the* h*-th iteration of the nested* while *loop in line 3. Then, after the iteration* k *of the loop in line 9,*

- *each list* $T_j[A]$ *contains all such positions* $i \in \{k-1, k, \ldots, j-1\}$*, that one can deduce* $A\big(a_1 \ldots a_i \langle a_{i+1} \ldots a_j \rangle\big)$ *without using any intermediate statements* $E\big(\varepsilon\langle a_1 \ldots a_j \rangle\big)$ *with* $h_j(E) \geqslant h$*;*
- *each set* P_i *with* $0 \leqslant i < j$ *contains all pairs* $(B, C) \in \mathcal{P}$*, for which there exists such a position* $\ell \in \{k, k+1, \ldots, j-1\}$*, that* $a_1 \ldots a_i \langle a_{i+1} \ldots a_\ell \rangle \in L_G(B)$ *and there is a deduction of* $C\big(a_1 \ldots a_\ell \langle a_{\ell+1} \ldots a_j \rangle\big)$ *that does not use any intermediate statements* $E\big(\varepsilon\langle a_1 \ldots a_j \rangle\big)$ *with* $h_j(E) \geqslant h$*.*

From this claim, one can infer the correctness of the entire algorithm, that is, that it determines the sets $T_j[A]$ correctly. It remains to estimate its running time.

The following properties of all operations on the lists $T_j[A]$ carried out in the course of a computation show that they can be implemented efficiently.

Claim 4.4.2 ([62, Lem. 2])**.** *Each list* $T_j[A]$ *always remains sorted. Each time the algorithm checks the condition in line 11, the set* $T_j[A]$ *does not contain any elements less than* k*. Each time the algorithm is about to execute the line 17, the set* $T_j[A]$ *does not contain any elements less than* k*.*

By Claim 4.4.2, checking the membership of an element in $T_j[A]$ in line 11 can be done by examining the top element of the list, while the modifications done in line 17 append an element on its top. Hence, all these operations can be done in constant time, and the overall running time of the algorithm is estimated as $\mathcal{O}(|G|^2 \cdot n^3)$ in the same way as in Theorem 4.1.

The quadratic upper bound for the unambiguous case relies on the following property.

Claim 4.4.3. *Assume that all concatenations in G are unambiguous, as in Definition 4.3. Then, for every j, for every pair $(B, C) \in \mathcal{P}$ and for every i, there exists at most one number k, such that (B, C) can be added to P_i at the iteration k of the loop in line 9 within the i-th iteration of the loop in line 2.*

Indeed, the existence of two such numbers would imply two different partitions of the substring $a_{i+1} \dots a_j$ into BC.

It follows from Claim 4.4.3 that the assignment statement $P_i = P_i \cup \{(B, C)\}$ in the inner loop is executed at most $|\mathcal{P}| \cdot |N| \cdot n^2$ times, which gives the desired upper bound on the running time. □

Below is given an example of a parsing table constructed by the algorithm.

Example 4.4. Consider the grammar with contexts from Example 2.1 and the equivalent grammar in binary normal form presented in Example 4.2. The grammar is unambiguous: unambiguity of concatenations follows from the fact that nonterminals A_0, B_0, C_0 and D_0 only define one-symbol strings, and concatenations of the form aA (with $a \in \Sigma$ and $A \in N$) are always unambiguous. The choice of rule for nonterminal S is unambiguous, since rules generate strings starting with different symbols and thus define disjoint languages. The same argument applies to the nonterminals S_a, S_b and A_a, where one of the rules always defines a one-symbol string and the other defines a string of a greater length.

The parsing table constructed by Algorithm 4.3 is given in Figure 4.5.

	S	A	S_a	S_b	A_a	A_0	B_0	C_0	D_0
$a_1 = a$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{0\}$	$\{0\}$	$\varnothing$	$\varnothing$	$\varnothing$
$a_2 = a$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{1\}$	$\{1\}$	$\varnothing$	$\varnothing$	$\varnothing$
$a_3 = b$	$\varnothing$	$\{1\}$	$\varnothing$	$\varnothing$	$\{0\}$	$\varnothing$	$\{2\}$	$\varnothing$	$\varnothing$
$a_4 = b$	$\varnothing$	$\{0\}$	$\varnothing$	$\{3\}$	$\varnothing$	$\varnothing$	$\{3\}$	$\varnothing$	$\varnothing$
$a_5 = c$	$\{3\}$	$\varnothing$	$\varnothing$	$\{2\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\{4\}$	$\varnothing$
$a_6 = c$	$\{2\}$	$\varnothing$	$\{1\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{5\}$	$\varnothing$
$a_7 = d$	$\{1\}$	$\varnothing$	$\{0\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{6\}$
$a_8 = d$	$\{0\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{7\}$

Figure 4.5: Parsing table of string $w = aabbccdd$ constructed by Algorithm 4.3.

Consider the elements of this table. For example, the set $\{2\}$ located in the position (B_0, a_3) of the table corresponds to the fact that the one symbol substring starting from position $i = 2 + 1$ and ending at position $j = 3$ has the property B_0.

Similarly, the set $\{3\}$ located in the position (S_b, a_4) states that the string $a_1 \dots a_3\langle a_{3+1} \dots a_4\rangle = a_1 \dots a_3\langle a_4\rangle = aab\langle b\rangle$ has the property S_b (by the rule $S_b \to b \,\&\, \preccurlyeq A$). The context $\preccurlyeq A$ in that rule is satisfied, since the singleton $\{0\}$ is in the position (A, a_4) of the parsing table, meaning that the string $\varepsilon\langle aabb\rangle$ is in $L_G(A)$.

The set $\{1\}$ located in the position (S_a, a_6) means that the string $a_1\langle a_{1+1} \dots a_6\rangle = a_1\langle a_2 \dots a_6\rangle = a\langle abbcc\rangle$ has the property S_a (by the rule $S_a \to A_0 S$).

Finally, the set $\{0\}$, located in the position (S, a_8) of the table, means that the string $a_0\langle a_{0+1} \dots a_8\rangle = \varepsilon\langle a_1 \dots a_8\rangle = \varepsilon\langle w\rangle$ has the desired property S. Therefore, the string $w = aabbccdd$ is in the language $L(G)$.

4.4 Establishing the normal form for grammars with one-sided contexts

For grammars with one-sided contexts, the transformation to the binary normal form is comprised of three stages. First, there is the *elimination of null conjuncts* (also called *empty conjuncts*), that is, any rules of the form $A \to \varepsilon \,\&\, \dots$; this is a rather elaborate generalization of the similar procedure for conjunctive grammars. The subsequent *elimination of null contexts* (that is, rules of the form $A \to \dots \,\&\, \lhd\varepsilon$) removes any checks that the current substring is in the beginning of the entire string. Finally, the *elimination of unit conjuncts* (any rules $A \to B \,\&\, \dots$ with $B \in N$) is the same as in the case of conjunctive grammars.

4.4.1 Elimination of null conjuncts

The task of eliminating null conjuncts is formulated in the same way as for conjunctive and ordinary context-free grammars: for any given grammar with contexts, the goal is to construct a grammar without null conjuncts that defines the same language as the original grammar—except for the empty string, which can no longer be generated by the constructed grammar [5].

An algorithm for carrying out such a transformation first calculates the set of nonterminals that generate the empty string, known as $\textsc{Nullable}(G) \subseteq N$, and then uses it to reconstruct the rules of the grammar. This set $\textsc{Nullable}(G) = \{\, A \in N \mid \varepsilon \in L_G(A) \,\}$ is calculated as a least upper bound of an ascending sequence of sets $\textsc{Nullable}_i(G)$, which is reached in at most $|N|$ steps. The set $\textsc{Nullable}_1(G) = \{\, A \in N \mid A \to \varepsilon \in R \,\}$ contains all nonterminals which directly define the empty string. Every next set $\textsc{Nullable}_{i+1}(G) = \{\, A \in N \mid A \to \alpha \in R,\ \alpha \in \textsc{Nullable}_i^*(G) \,\}$ contains nonterminals that generate ε by the rules referring to other nullable nonterminals. This knowledge is given by the Kleene star of $\textsc{Nullable}_i(G)$.

Example 4.5. Consider the following context-free grammar, which defines the language $\{abc, ab, ac, a, bcd, bd, cd, d\}$.

$$\begin{array}{lcl} S & \to & aA \mid Ad \\ A & \to & BC \\ B & \to & \varepsilon \mid b \\ C & \to & \varepsilon \mid c \end{array}$$

Since B generates the empty string, the rule $A \to BC$ can be used to generate just C; therefore, once the rule $B \to \varepsilon$ is removed, one should add a new rule $A \to C$, in which B is omitted. Similarly one can remove the rule $C \to \varepsilon$ and add a "compensatory" rule $A \to B$. Since both B and C generate ε, so does A by the rule $A \to BC$. Hence, extra rules $S \to a$ and $S \to d$, where A is omitted, have to be added. After the elimination of ε, the rules of the grammar are as follows.

$$\begin{array}{lcl} S & \to & aA \mid a \mid Ad \mid d \\ A & \to & BC \mid B \mid C \\ B & \to & b \\ C & \to & c \end{array}$$

For the grammar in Example 4.5, the calculation of the set $\textsc{Nullable}(G)$ proceeds as follows.

$$\begin{array}{lcl} \textsc{Nullable}_0(G) & = & \varnothing, \\ \textsc{Nullable}_1(G) & = & \{B, C\}, \\ \textsc{Nullable}_2(G) & = & \{B, C, A\}, \end{array}$$

and $\textsc{Nullable}(G) = \textsc{Nullable}_2(G)$.

For grammars with contexts, as shown in the next example, the same nonterminal symbol may generate the empty string in some contexts, and not generate it in the others.

Example 4.6. Consider the following grammar, which defines the language $\{a, ac, abc, aabc\}$.

$$
\begin{array}{lcl}
S & \to & aA \mid aaA \\
A & \to & BC \\
B & \to & \varepsilon \mathbin{\&} \lhd D \mid b \\
C & \to & \varepsilon \mathbin{\&} \lhd E \mid c \\
D & \to & a \\
E & \to & a
\end{array}
$$

Here B generates ε only in a context described by D, while C generates ε only in a context E (that is, in each case, only in the context a). Hence, A generates ε only in contexts described by both D and E (that is, again, only in the context a). Therefore, the grammar generates the string a by the rule $S \to aA$, but does not generate the string aa, because A no longer generates ε in the left context aa.

Thus, for grammars with contexts, instead of talking about nonterminals generating the empty string, one should consider generation of ε by a nonterminal in a context satisfying a given set of conditions, such as by A in the context $\{D, E\}$, as seen in Example 4.6.

For simplicity, assume that context operators are applied only to individual nonterminal symbols ($\lhd D$, $\lhd\!\!\!=E$); later on, in Lemma 4.2, it will be shown that there is no loss of generality in this assumption. Then the task is to determine all pairs of the form (U, A), with $A \in N$ and $U \subseteq N$, representing the intuitive idea that A generates ε in a context of the form described by each symbol in U. Returning to Example 4.6, one should first obtain the pairs $(\{D\}, B)$ and $(\{E\}, C)$ directly from the rules for B and C, then "concatenate" these pairs to find that BC generates ε in the context $\{D, E\}$, and finally infer another pair $(\{D, E\}, A)$ according to the rule $A \to BC$.

This symbolic "concatenation" of pairs is carried out by the following extension of the Kleene star (as described in the beginning of Section 4.4.1) to symbols encumbered with contexts. For any set of pairs $\mathcal{S} \subseteq 2^N \times N$, denote by $\mathcal{S}^\star$ the set of all pairs $(U_1 \cup \ldots \cup U_\ell,\ A_1 \ldots A_\ell)$ with $\ell \geqslant 0$ and $(U_i, A_i) \in \mathcal{S}$: in other words, the symbols are concatenated, while their contexts are put together in one set. It is worth noting that for $\ell = 0$, the concatenation of zero pairs yields a pair $(\varnothing, \varepsilon)$, which $\mathcal{S}^\star$ always contains, regardless of $\mathcal{S}$; in particular, $\varnothing^\star = \big\{(\varnothing, \varepsilon)\big\}$ (cf. $\varnothing^* = \{\varepsilon\}$ for the Kleene star on languages).

Using this notion, the set of nullable symbols in contexts is obtained as the limit of a sequence of sets, as follows.

Definition 4.4. *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, in which the context operators are applied only to single nonterminal symbols rather than to concatenations.*

Construct the sequence of sets $\textsc{Nullable}_i(G) \subseteq 2^N \times N$, *with* $i \geqslant 0$, *by setting*

$$\textsc{Nullable}_0(G) = \varnothing,$$

$$\begin{aligned}\textsc{Nullable}_{i+1}(G) = \big\{\, (\{D_1, \ldots, D_m\} \cup \{E_1, \ldots, E_n\} \cup U_1 \cup \ldots \cup U_k,\ A) \;\big|\; \\ \textit{there exists a rule} \\ A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \in R, \\ \textit{and the pairs } (U_1, \alpha_1), \ldots, (U_k, \alpha_k) \in \textsc{Nullable}_i^{\star}(G) \,\big\}.\end{aligned}$$

Finally, denote its limit by

$$\textsc{Nullable}(G) = \bigcup_{i \geqslant 0} \textsc{Nullable}_i(G).$$

At the first step, in the definition of $\textsc{Nullable}_1(G)$, the pairs $(U_1, \alpha_1), \ldots, (U_k, \alpha_k)$ must belong to $\textsc{Nullable}_0^{\star}(G) = \varnothing^{\star} = (\varnothing, \varepsilon)$, and accordingly, $\textsc{Nullable}_1(G)$ is the set of all pairs $(\varnothing, A)$ with $A \to \varepsilon \in R$.

In general, the formula defining $\textsc{Nullable}_{i+1}(G)$ expresses the following reasoning. Assume that all base conjuncts $\alpha_1, \ldots, \alpha_k$ in some rule are already known to generate the empty string *in some contexts*, and the algorithm should determine a possible set of contexts, in which A generates ε. First, the contexts needed for each conjunct α_i are assembled together by the star operator, and then are joined as the union $U_1 \cup \ldots \cup U_k$. All context specifications in the chosen rule are also added to the set. Then A generates the empty string in all contexts described by the resulting set.

Example 4.7. For the grammar in Example 4.6,

$$\begin{aligned}\textsc{Nullable}_0(G) &= \varnothing, \\ \textsc{Nullable}_1(G) &= \big\{(\{D\}, B),\ (\{E\}, C)\big\}, \\ \textsc{Nullable}_2(G) &= \big\{(\{D\}, B),\ (\{E\}, C),\ (\{D, E\}, A)\big\},\end{aligned}$$

and $\textsc{Nullable}(G) = \textsc{Nullable}_2(G)$. The pair $(\{D, E\}, A)$ represents the ability of A to generate the empty string in any context defined by both D and E.

The next lemma states that, indeed, the generation of ε in any context can be determined using the set $\textsc{Nullable}(G)$.

Lemma 4.1. *Let* $G = (\Sigma, N, R, S)$ *be a grammar with contexts, let* $A \in N$ *and* $u \in \Sigma^*$. *Then,* $u\langle\varepsilon\rangle \in L_G(A)$ *if and only if the set* $\textsc{Nullable}(G)$ *contains such a pair* $(\{K_1, \ldots, K_t\}, A)$ *that* $\varepsilon\langle u\rangle \in L_G(K_1)$, $\ldots$, $\varepsilon\langle u\rangle \in L_G(K_t)$.

Proof. $\Leftarrow$ Assume that $u\langle\varepsilon\rangle \in L_G(A)$. The existence of a desired pair in $\textsc{Nullable}(G)$ is proved by induction on p, the number of steps used in the deduction of the proposition $A\big(u\langle\varepsilon\rangle\big)$.

Basis. Let $p = 1$. If a proposition $A\big(u\langle\varepsilon\rangle\big)$ is deduced in one step, then it must be obtained by a rule $A \to \varepsilon$. Then the pair $(\varnothing, A)$ is added to $\textsc{Nullable}(G)$ at the first iteration.

Induction step. Let a proposition $A\big(u\langle\varepsilon\rangle\big)$ be deduced in p steps, with the last step using a rule

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n. \tag{4.2}$$

For each base conjunct α_i, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$, where $\ell_i \geqslant 0$ and $X_{i,j} \in N$. Then the string $u\langle\varepsilon\rangle$ is in $L_G(X_{i,j})$ for all $j \in \{1, \ldots, \ell_i\}$, and the last step of the deduction is of the form

$$\big\{X_{i,j}\big(u\langle\varepsilon\rangle\big)\big\}_{i,j}, D_1\big(\varepsilon\langle u\rangle\big), \ldots, D_m\big(\varepsilon\langle u\rangle\big), E_1\big(\varepsilon\langle u\rangle\big), \ldots, E_n\big(\varepsilon\langle u\rangle\big) \vdash A\big(u\langle\varepsilon\rangle\big).$$

For each symbol $X_{i,j}$, by the induction hypothesis, there exists such a set $U_{i,j} \subseteq N$, that $(U_{i,j}, X_{i,j}) \in \textsc{Nullable}(G)$ and $\varepsilon\langle u\rangle \in L_G(K)$ for all $K \in U_{i,j}$. Let $h > 0$ be the least such number that all these pairs, for all i and j, appear in $\textsc{Nullable}_h(G)$. Then the pair $\big(U_{i,1} \cup \ldots \cup U_{i,\ell_i}, \alpha_i\big)$ is in $\textsc{Nullable}^\star_h(G)$. Denote $U_i = U_{i,1} \cup \ldots \cup U_{i,\ell_i}$ and $U = U_1 \cup \ldots \cup U_k \cup \{D_1, \ldots, D_m, E_1, \ldots, E_n\}$.

The pair (U, A) belongs to $\textsc{Nullable}_{h+1}(G)$ by construction. The second claim is that $\varepsilon\langle u\rangle \in L_G(K)$ for all $K \in U$. Indeed, for symbols in each $U_{i,j}$ this has been proved above, while the remaining symbols in U are from the contexts in the rule (4.2), and the propositions $D_i\big(\varepsilon\langle u\rangle\big)$ (for $i \in \{1, \ldots, m\}$) and $E_i\big(\varepsilon\langle u\rangle\big)$ (for $i \in \{1, \ldots, n\}$) are known to be true.

$\Rightarrow$ Let $U = \{K_1, \ldots, K_t\}$ and $(U, A) \in \textsc{Nullable}(G)$. Then $(U, A) \in \textsc{Nullable}_h(G)$ for some $h \geqslant 0$. The goal is to prove that $u\langle\varepsilon\rangle \in L_G(A)$, which will be done inductively on h.

Basis. If $h = 0$, then $\textsc{Nullable}_0(G) = \varnothing$ and no symbols $A \in N$ satisfy the assumptions.

Induction step. Let $(U, A) \in \textsc{Nullable}_h(G)$ and $\varepsilon\langle u\rangle \in L_G(K)$ for all $K \in U$. According to Definition 4.4, the set R contains a rule of the form (4.2), where, for each base conjunct α_i in this rule, there is a pair (U_i, α_i) in $\textsc{Nullable}^\star_{h-1}(G)$, and $U = U_1 \cup \ldots \cup U_k \cup \{D_1, \ldots, D_m, E_1, \ldots, E_n\}$.

For each conjunct α_i, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$ with $\ell_i \geqslant 0$ and $X_{i,1}, \ldots, X_{i,\ell_i} \in \Sigma \cup N$. Then, by the definition of the "star" of $\textsc{Nullable}_{h-1}(G)$, there exist pairs $(U_{i,1}, X_{i,1})$, $\ldots$, $(U_{i,\ell_i}, X_{i,\ell_i})$ in $\textsc{Nullable}_{h-1}(G)$, for which $U_{i,1} \cup \ldots \cup U_{i,\ell_i} = U_i$. Therefore, by the induction hypothesis for each of these pairs, $u\langle\varepsilon\rangle \in L_G(X_{i,j})$ for all j, that is, $\vdash_G X_{i,1}\big(u\langle\varepsilon\rangle\big)$, $\ldots$, $X_{i,\ell_i}\big(u\langle\varepsilon\rangle\big)$.

This gives all the remaining premises for deducing the membership of $u\langle\varepsilon\rangle$ in $L_G(A)$.

$$X_{1,1}\big(u\langle\varepsilon\rangle\big), \ldots, X_{k,\ell_k}\big(u\langle\varepsilon\rangle\big), D_1\big(\varepsilon\langle u\rangle\big), \ldots, D_m\big(\varepsilon\langle u\rangle\big), \\ E_1\big(\varepsilon\langle u\rangle\big), \ldots, E_n\big(\varepsilon\langle u\rangle\big) \vdash_G A\big(u\langle\varepsilon\rangle\big)$$

The premises $D_i\big(\varepsilon\langle u\rangle\big)$ and $E_i\big(\varepsilon\langle u\rangle\big)$ are known to be true, because $\varepsilon\langle u\rangle \in L_G(K)$ for all $K \in U$ by the assumption. □

Now, once all cases of generation of the empty string in a grammar have been enumerated in the set $\textsc{Nullable}(G)$, this information can be used to rebuild the grammar without ever generating ε. For a context-free grammar, this is done by creating variants of the existing rules, in which the nullable symbols are omitted: for instance, if there is a rule $A \to BC$, and B generates ε, then the new grammar gets another rule $A \to C$. For a grammar with contexts, the generation of an empty string in the original grammar depends on the contexts, as reflected in the set $\textsc{Nullable}(G)$, and the same logical dependence must be ensured in the new grammar.

Example 4.8. Consider the grammar from Example 4.6. Once both null rules are removed, the effect of these rules on the generation of longer strings is replicated as follows.

The nonterminals B and C, as shown in Example 4.7, can generate the empty string only in the left contexts of the form D and E, respectively. For the rule $A \to BC$, when the symbol B is omitted, the resulting extra rule should be $A \to C \mathbin{\&} {\lhd}D$ (rather than just $A \to C$), which inherits the context dependence from B. Similarly, C can be omitted in $A \to BC$ by introducing a new rule $A \to B \mathbin{\&} {\lhd\!\!\!=}E$. Note that the latter rule uses an extended context operator, because it refers to the left context of the omitted symbol C, which includes the substring generated by B.

The nonterminal A itself generates the empty string, but only in the contexts described by both D and E. Thus, the generation of ε by A in the rule $S \to aA$ can be handled by adding an extra rule $S \to a \mathbin{\&} {\lhd\!\!\!=}D \mathbin{\&} {\lhd\!\!\!=}E$ (rather than $S \to a$), which again preserves both context dependencies. The rule $S \to aaA$ is processed in the same way.

The resulting set of rules of the new grammar is given below.

$$
\begin{aligned}
S &\to aA \mid a \,\&\, \lhd D \,\&\, \lhd E \mid aaA \mid aa \,\&\, \lhd D \,\&\, \lhd E \\
A &\to BC \mid B \,\&\, \lhd E \mid C \,\&\, \lhd D \\
B &\to b \\
C &\to c \\
D &\to a \\
E &\to a
\end{aligned}
$$

Given an arbitrary grammar with contexts, it is convenient to begin the elimination of null conjuncts with simplifying the form of the rules. After the following pre-processing stage, all concatenations are expressed in separate rules $A \to BC$ and all context operators are applied to individual nonterminals.

Lemma 4.2. *For every grammar with contexts $G_0 = (\Sigma, N_0, R_0, S_0)$, there exists and can be effectively constructed another grammar with contexts $G = (\Sigma, N, R, S)$ generating the same language, in which all rules are of the following form.*

$$A \to BC \qquad (A, B, C \in N) \tag{4.3a}$$

$$A \to a \qquad (a \in \Sigma) \tag{4.3b}$$

$$A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhd E_1 \,\&\, \ldots \,\&\, \lhd E_n \qquad (k \geqslant 1,\ m, n \geqslant 0,\ A, B_i, D_i, E_i \in N) \tag{4.3c}$$

$$A \to \varepsilon \tag{4.3d}$$

If the original grammar is unambiguous, then so is the resulting grammar.

Proof. Each conjunct in the given grammar $G_0 = (\Sigma, N_0, R_0, S_0)$ is of the form α, $\lhd\alpha$ or $\lhd\!\!\!=\alpha$, where $\alpha \in (\Sigma \cup N_0)^*$. First, construct a grammar G_1 as follows. Each context operator $\lhd\alpha$ or $\lhd\!\!\!=\alpha$, with $\alpha \in \Sigma$ or $|\alpha| > 1$, is replaced by $\lhd Y_\alpha$ or $\lhd\!\!\!= Y_\alpha$, respectively, where Y_α is a new nonterminal with the unique rule $Y_\alpha \to \alpha$. Operators of the form $\lhd\varepsilon$, defining empty contexts, are simulated by conjuncts $\lhd Z$, where the new nonterminal Z has the unique rule $Z \to \varepsilon$. Then, each conjunct α, with $|\alpha| > 2$, is shortened by introducing new nonterminals: for instance, $\alpha = BaC$ may become BX_{aC}, with a rule $X_{aC} \to aC$. Denote the new grammar by G_2. Finally, every instance of a symbol $a \in \Sigma$ in a conjunct α with $|\alpha| = 2$ is replaced with a new nonterminal X_a with a unique rule $X_a \to a$, resulting in a grammar G of the desired form.

Unambiguity of choice of a rule is preserved at every stage of the transformation. Indeed, every modified rule for an old nonterminal generates

the same language as its prototype rule, and the alternatives remain disjoint. Each new nonterminal has a unique rule, which rules out ambiguity of choice.

No new concatenations are introduced in the transformation of G_0 to G_1, and they remain unambiguous. In G_2, each new concatenation is of the form $s_1 \cdot X_{s_2 \ldots s_\ell}$, and if it is ambiguous, then so is the original factorization $s_1 \cdot s_2 \cdot \ldots \cdot s_\ell$, contradicting the assumption. No new concatenations are introduced in G. □

The following transformation eliminates all null conjuncts from a given grammar with contexts.

Construction 4.1. Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts with all rules of the form (4.3a)–(4.3d), as in Lemma 4.2. Consider the set $\textsc{Nullable}(G)$ and construct a new grammar with contexts $G' = (\Sigma, N \cup \{W\}, R', S)$, where R' contains the following rules.

1. The new nonterminal W generates all non-empty strings in all contexts, using the following rules:

$$\begin{aligned} W &\to a && (a \in \Sigma) \\ W &\to aW && (a \in \Sigma) \end{aligned}$$

2. Every rule of the form $A \to a$ in R is preserved in R':

$$A \to a \tag{4.4}$$

3. For every rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, {\lhd}D_1 \,\&\, \ldots \,\&\, {\lhd}D_m \,\&\, {\lhdeq}E_1 \,\&\, \ldots \,\&\, {\lhdeq}E_n$ in R, the set R' contains the same rule

$$A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, {\lhd}D_1 \,\&\, \ldots \,\&\, {\lhd}D_m \,\&\, {\lhdeq}E_1 \,\&\, \ldots \,\&\, {\lhdeq}E_n, \tag{4.5a}$$

and, possibly, the following extra rule:

$$A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, E_1 \,\&\, \ldots \,\&\, E_n \,\&\, {\lhd}\varepsilon, \tag{4.5b}$$

which is added if $m \geqslant 1$ and $\varepsilon\langle\varepsilon\rangle \in L_G(D_1) \cap \ldots \cap L_G(D_m)$.

The latter rule handles the case of a null left context, when the substring being defined is a prefix of the whole string. If there is at least one proper left context operator ${\lhd}D_i$ in the original rule, then the same operator in the new grammar would no longer admit null context, and hence a new rule without these operators has to be introduced. Note that in a null context, an extended left context ${\lhdeq}E_i$ has the same effect as a base conjunct E_i. Also note that the condition $\varepsilon\langle\varepsilon\rangle \in L_G(D_i)$ can be checked by listing all deductions among the propositions $X\big(\varepsilon\langle\varepsilon\rangle\big)$, which depend only on each other.

4. Every rule of the form $A \to BC$ in R is added to R',

$$A \to BC \tag{4.6a}$$

along with the following extra rules.

$$A \to B \,\&\, \trianglelefteqslant K_1 \,\&\, \ldots \,\&\, \trianglelefteqslant K_t \tag{4.6b}$$
$$\text{(for all } (\{K_1, \ldots, K_t\}, C) \in \textsc{Nullable}(G))$$
$$A \to C \,\&\, \triangleleft K_1 \,\&\, \ldots \,\&\, \triangleleft K_t \,\&\, \triangleleft W \tag{4.6c}$$
$$\text{(for all } (\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G))$$

In each case, a nullable nonterminal is omitted, and the condition of its being nullable is accordingly included in the rule. In the first case, the substring generated by B is a part of the omitted C's context, and hence extended context operators are used ($\trianglelefteqslant K_i$). In the second case, once B is omitted, its context is referenced by proper context operators ($\triangleleft K_i$); one more context operator ($\triangleleft W$) restricts the applicability of this rule to a non-empty left context. The case when B is omitted in an empty left context is handled in yet another rule

$$A \to C \,\&\, \triangleleft \varepsilon, \tag{4.6d}$$

which is added to the grammar if there exists such a set $\{K_1, \ldots, K_t\} \subseteq N$, that $(\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G)$ and $\varepsilon\langle\varepsilon\rangle \in L_G(K_1) \cap \ldots \cap L_G(K_t)$.

Thus, among the two rules for a nullable B, the former rule (4.6c) only generates strings in non-empty left contexts, whereas the latter rule (4.6d) only applies in the empty left context. This is essential for the construction to preserve unambiguity.

None of the rules (4.3d) generating the empty string are included in G'.

Correctness Statement 1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts and let $G' = (\Sigma, N \cup \{W\}, R', S)$ be the grammar obtained by Construction 4.1. Then $L_{G'}(A) = L_G(A) \setminus \Sigma^*\langle\varepsilon\rangle$ for every $A \in N$; in particular, $L(G') = L(G) \setminus \{\varepsilon\}$. Furthermore, if G is unambiguous, then so is G'.*

The first claim is that whenever a non-empty string in any context is generated by a symbol in the original grammar, it is also generated by that symbol in the new grammar.

Lemma 4.3. *Let a grammar $G = (\Sigma, N, R, S)$ be transformed to $G' = (\Sigma, N \cup \{W\}, R', S)$ according to Construction 4.1. Then, if a proposition $A\big(u\langle v\rangle\big)$, with $A \in N$, $u \in \Sigma^*$ and $v \in \Sigma^+$, is deduced in G, then it is also deduced in G'.*

Proof. The proof is carried out by induction on p, the number of steps in the deduction of $A\big(u\langle v\rangle\big)$ in G.

Basis. Let $p = 1$ and consider a proposition $A\big(u\langle v\rangle\big)$ deduced in G by a rule $A \to a$. Then $v = a$ and the last step of the deduction is $a\big(u\langle a\rangle\big) \vdash_G A\big(u\langle a\rangle\big)$. By the construction, the grammar G' also has the rule $A \to a$, and the same deduction can be carried out in G': $a\big(u\langle a\rangle\big) \vdash_{G'} A\big(u\langle a\rangle\big)$.

Induction step. Consider the rule in G used at the last step of a deduction of $A\big(u\langle v\rangle\big)$.

If this is a rule $A \to BC$, then the last step of the deduction takes the form $B\big(u\langle v_1\rangle\big)$, $C\big(uv_1\langle v_2\rangle\big) \vdash_G A\big(u\langle v\rangle\big)$, for some partition $v = v_1v_2$, and each of the premises is deduced in fewer than p steps. If both v_1 and v_2 are non-empty, then, by the induction hypothesis, both propositions $B\big(u\langle v_1\rangle\big)$ and $C\big(uv_1\langle v_2\rangle\big)$ are deducible in G', and therefore the proposition $A\big(u\langle v\rangle\big)$ can be obtained in G' by the rule (4.6a): $B\big(u\langle v_1\rangle\big)$, $C\big(uv_1\langle v_2\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$.

Let now either v_1 or v_2 be empty (both cannot be empty, because $v \neq \varepsilon$).

- Let $v_1 = v$ and $v_2 = \varepsilon$. Then, the last step of the deduction of $A\big(u\langle v\rangle\big)$ is $B\big(u\langle v\rangle\big)$, $C\big(uv\langle\varepsilon\rangle\big) \vdash_G A\big(u\langle v\rangle\big)$. The proposition $B\big(u\langle v\rangle\big)$ can be deduced in G' by the induction hypothesis. Since $uv\langle\varepsilon\rangle \in L_G(C)$, by Lemma 4.1, there exist such symbols $K_1, \ldots, K_t \in N$, that $(\{K_1, \ldots, K_t\}, C) \in \textsc{Nullable}(G)$ and all propositions $K_i\big(\varepsilon\langle uv\rangle\big)$ can be deduced in the grammar G, in fewer than p steps. Since $uv \neq \varepsilon$, the induction hypothesis is applicable to these propositions, and it asserts that each $K_i\big(\varepsilon\langle uv\rangle\big)$ can be deduced in the grammar G' as well. Then the proposition $A\big(u\langle v\rangle\big)$ is deduced in G' by the rule (4.6b), which exists because of the pair $(\{K_1, \ldots, K_t\}, C)$ in $\textsc{Nullable}(G)$; the deduction proceeds as follows: $B\big(u\langle v\rangle\big)$, $K_1\big(\varepsilon\langle uv\rangle\big)$, $\ldots$, $K_t\big(\varepsilon\langle uv\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$.

- If $v_1 = \varepsilon$ and $v_2 = v$, then the last step of deduction of the proposition $A\big(u\langle v\rangle\big)$ in G is $B\big(u\langle\varepsilon\rangle\big)$, $C\big(u\langle v\rangle\big) \vdash_G A\big(u\langle v\rangle\big)$. By the induction hypothesis, the proposition $C\big(u\langle v\rangle\big)$ can be deduced in G'. For $u\langle\varepsilon\rangle \in L_G(B)$, by Lemma 4.1, there exist $K_1, \ldots, K_t \in N$, such that $(\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G)$ and all propositions $K_i\big(\varepsilon\langle u\rangle\big)$ can be deduced in the grammar G.

  If $u \neq \varepsilon$, then each proposition $K_i\big(\varepsilon\langle u\rangle\big)$ can be deduced in the grammar G' by the induction hypothesis. The string $\varepsilon\langle u\rangle$ is also in $L_{G'}(W)$, and thus $A\big(u\langle v\rangle\big)$ can be deduced in G' by the rule (4.6c): $C\big(u\langle v\rangle\big)$, $K_1\big(\varepsilon\langle u\rangle\big)$, $\ldots$, $K_t\big(\varepsilon\langle u\rangle\big)$, $W\big(\varepsilon\langle u\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$. Note that the rule (4.6c) is in G' for the pair $(\{K_1, \ldots, K_t\}, B)$ in $\textsc{Nullable}(G)$.

  Let $u = \varepsilon$. Then $\varepsilon\langle\varepsilon\rangle \in L_G(K_i)$, for all $i \in \{1, \ldots, t\}$, and therefore the grammar G' has a rule (4.6d). The proposition $A\big(\varepsilon\langle v\rangle\big)$ can be deduced by this rule as $C\big(\varepsilon\langle v\rangle\big) \vdash_{G'} A\big(\varepsilon\langle v\rangle\big)$.

Consider now the other case, when the last step of the deduction of $A\langle u\langle v\rangle\rangle$ uses the rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$. That is, the deduction is $B_1\langle u\langle v\rangle\rangle$, …, $B_k\langle u\langle v\rangle\rangle$, $D_1\langle \varepsilon\langle u\rangle\rangle$, …, $D_m\langle \varepsilon\langle u\rangle\rangle$, $E_1\langle \varepsilon\langle uv\rangle\rangle$, …, $E_n\langle \varepsilon\langle uv\rangle\rangle \vdash_G A\langle u\langle v\rangle\rangle$.

If $u \neq \varepsilon$, then, by the induction hypothesis, each of the premises can be deduced in the grammar G'. This is sufficient to deduce $A\langle u\langle v\rangle\rangle$ in the grammar G' by the rule (4.5a).

Let $u = \varepsilon$. The propositions $B_i\langle \varepsilon\langle v\rangle\rangle$ for all $i \in \{1, \ldots, k\}$ and $E_i\langle \varepsilon\langle v\rangle\rangle$ for all $i \in \{1, \ldots, n\}$ can still be deduced in G' by the induction hypothesis. Since the propositions $D_1\langle \varepsilon\langle \varepsilon\rangle\rangle$, …, $D_m\langle \varepsilon\langle \varepsilon\rangle\rangle$ are deducible in G, the grammar G' has a rule (4.5b), by which the desired proposition $A\langle \varepsilon\langle v\rangle\rangle$ is deduced in G': $B_1\langle \varepsilon\langle v\rangle\rangle$, …, $B_k\langle \varepsilon\langle v\rangle\rangle$, $D_1\langle \varepsilon\langle \varepsilon\rangle\rangle$, …, $D_m\langle \varepsilon\langle \varepsilon\rangle\rangle$, $E_1\langle \varepsilon\langle v\rangle\rangle$, …, $E_n\langle \varepsilon\langle v\rangle\rangle \vdash_{G'} A\langle \varepsilon\langle v\rangle\rangle$. □

The converse statement is that whenever a string is generated by some symbol in the new grammar, the same string must be generated in the original grammar.

Lemma 4.4. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, let $G' = (\Sigma, N \cup \{W\}, R', S)$ be the grammar obtained by Construction 4.1 and let $A \in N$ be a nonterminal symbol. Then, if a proposition $A\langle u\langle v\rangle\rangle$ can be deduced in the grammar G', it can also be deduced in G, and $v \neq \varepsilon$.*

Moreover, consider the rule used at the last step of a deduction of the proposition $A\langle u\langle v\rangle\rangle$ in G'. Then,

I. if $A\langle u\langle v\rangle\rangle$ is deduced in G' by a rule of the form (4.4), then it is deduced in G by the (same) rule $A \to a$;

II. if $A\langle u\langle v\rangle\rangle$ is deduced in G' by a rule of the form (4.5a) or (4.5b), then it is deduced in G by the rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$;

III. if $A\langle u\langle v\rangle\rangle$ is deduced in G' by a rule (4.6a), (4.6b), (4.6c), or (4.6d), then it is deduced in G by the rule $A \to BC$, and furthermore,

- *(a) if it is deduced by a rule (4.6a), and there is such a partition $v = v_1 v_2$ with $v_1, v_2 \neq \varepsilon$, that $u\langle v_1\rangle \in L_{G'}(B)$ and $uv_1\langle v_2\rangle \in L_{G'}(C)$, then $u\langle v_1\rangle \in L_G(B)$ and $uv_1\langle v_2\rangle \in L_G(C)$;*
- *(b) if it is deduced by a rule (4.6b), then $u\langle v\rangle \in L_G(B)$ and $uv\langle \varepsilon\rangle \in L_G(C)$;*
- *(c) if the rule is (4.6c), then $u \neq \varepsilon$, $u\langle \varepsilon\rangle \in L_G(B)$ and $u\langle v\rangle \in L_G(C)$;*
- *(d) and finally, if the rule is (4.6d), then $u = \varepsilon$, $u\langle \varepsilon\rangle \in L_G(B)$ and $u\langle v\rangle \in L_G(C)$.*

The latter part of the statement (I–III) refers to the rule used at the last step of a deduction of the proposition $A\big(u\langle v\rangle\big)$: if a rule r' is used to deduce it in the new grammar, then this rule was constructed on the basis of some rule r in the original grammar, and the derivation of $A\big(u\langle v\rangle\big)$ in the original grammar (constructed in the lemma) uses this rule r.

Proof. The proof is by induction on p, the number of steps used to deduce the proposition $A\big(u\langle v\rangle\big)$ in the grammar G'.

Basis. Let $p = 1$ and let $A\big(u\langle v\rangle\big)$ be deduced in G' by the rule $A \to a$. Then $v = a \in \Sigma$ and the desired proposition is obtained from an axiom: $a\big(u\langle a\rangle\big) \vdash_{G'} A\big(u\langle a\rangle\big)$. The same deduction can be carried out in G by the same rule $A \to a \in R$.

Induction step. Consider a deduction of a proposition $A\big(u\langle v\rangle\big)$ in G', and let r' be the rule used at the last step of this deduction. Let r be the rule in G, from which r' is obtained in Construction 4.1. The argument splits into cases depending on the form of r'.

- If the rule r' is of the form (4.5a), then the last step of deduction of $A\big(u\langle v\rangle\big)$ in G' is $B_1\big(u\langle v\rangle\big)$, …, $B_k\big(u\langle v\rangle\big)$, $D_1\big(\varepsilon\langle u\rangle\big)$, …, $D_m\big(\varepsilon\langle u\rangle\big)$, $E_1\big(\varepsilon\langle uv\rangle\big)$, …, $E_n\big(\varepsilon\langle uv\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$. By the induction hypothesis, each of the premises can also be deduced in the grammar G, and this allows repeating the last step of deduction in G using the rule r, which is the same as r'.

  The string v is non-empty, because this is ensured by the induction hypothesis for $B_1\big(u\langle v\rangle\big)$. The argument for the non-emptiness of v is the same in all subsequent cases.

- Let r' be of the form (4.5b). Then $u = \varepsilon$ and the last step of the deduction is $B_1\big(\varepsilon\langle v\rangle\big)$, …, $B_k\big(\varepsilon\langle v\rangle\big)$, $E_1\big(\varepsilon\langle v\rangle\big)$, …, $E_n\big(\varepsilon\langle v\rangle\big)$ $\vdash_{G'} A\big(\varepsilon\langle v\rangle\big)$. By the induction hypothesis, each of the propositions $B_i\big(\varepsilon\langle v\rangle\big)$ (with $i \in \{1, \ldots, k\}$) and $E_i\big(\varepsilon\langle v\rangle\big)$ (with $i \in \{1, \ldots, n\}$) can be deduced in the grammar G. The rule r in G is of the form $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$, and the condition of existence of r' is that $\varepsilon\langle\varepsilon\rangle \in L_G(D_i)$, for $i \in \{1, \ldots, m\}$. Now the proposition $A\big(\varepsilon\langle v\rangle\big)$ is deduced in G as follows:

  $$\{B_i\big(\varepsilon\langle v\rangle\big)\}_{i\in\{1,\ldots,k\}}, \{D_i\big(\varepsilon\langle\varepsilon\rangle\big)\}_{i\in\{1,\ldots,m\}}, \{E_i\big(\varepsilon\langle v\rangle\big)\}_{i\in\{1,\ldots,n\}} \vdash_G A\big(\varepsilon\langle v\rangle\big).$$

- Let r' be of the form (4.6a), that is, $A \to BC$; the rule r in G is of the same form. The last step of deduction of $A\big(u\langle v\rangle\big)$ in the grammar G' is $B\big(u\langle v_1\rangle\big)$, $C\big(uv_1\langle v_2\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$, for some partition $v = v_1v_2$. By the induction hypothesis, both propositions $B\big(u\langle v_1\rangle\big)$ and $C\big(uv_1\langle v_2\rangle\big)$ can be deduced in the grammar G. This deduction in G can be continued with the same last step using the rule r: $B\big(u\langle v_1\rangle\big)$, $C\big(uv_1\langle v_2\rangle\big)$ $\vdash_G A\big(u\langle v\rangle\big)$.

Here the non-emptiness of v is given by the induction hypothesis for $B\big(u\langle v_1\rangle\big)$, which asserts that already v_1 is non-empty.

- Assume that r' is of the form (4.6b), that is, $A \to B \,\&\, \triangleleft K_1 \,\&\, \ldots \,\&\, \triangleleft K_t$, obtained from a rule $A \to BC$ and a pair $(\{K_1, \ldots, K_t\}, C) \in \textsc{Nullable}(G)$. Then the proposition $A\big(u\langle v\rangle\big)$ is deduced in G' out of the premises $B\big(u\langle v\rangle\big)$, $K_1\big(\varepsilon\langle uv\rangle\big)$, $\ldots$, $K_t\big(\varepsilon\langle uv\rangle\big)$. By the induction hypothesis, the proposition $B\big(u\langle v\rangle\big)$ can be deduced in the grammar G. The proposition $C\big(uv\langle\varepsilon\rangle\big)$ can be deduced in G by Lemma 4.1, applied to the pair $(\{K_1, \ldots, K_t\}, C)$ and the propositions $K_1\big(\varepsilon\langle uv\rangle\big)$, $\ldots$, $K_t\big(\varepsilon\langle uv\rangle\big)$, which are deducible in G by the induction hypothesis. Now one can deduce the proposition $A\big(u\langle v\rangle\big)$ in G by the rule $A \to BC$, as follows: $B\big(u\langle v\rangle\big)$, $C\big(uv\langle\varepsilon\rangle\big) \vdash_G A\big(u\langle v\rangle\big)$.

- Let r' be of the form (4.6c), that is, $A \to C \,\&\, \triangleleft K_1 \,\&\, \ldots \,\&\, \triangleleft K_t \,\&\, \triangleleft W$, obtained from a rule $A \to BC$ in G and a pair $(\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G)$. The last step of deduction of $A\big(u\langle v\rangle\big)$ in G' is then $C\big(u\langle v\rangle\big)$, $K_1\big(\varepsilon\langle u\rangle\big)$, $\ldots$, $K_t\big(\varepsilon\langle u\rangle\big)$, $W\big(\varepsilon\langle u\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big)$. By the induction hypothesis, $\vdash_G C\big(u\langle v\rangle\big)$ and $\vdash_G K_1\big(\varepsilon\langle u\rangle\big)$, $\ldots$, $\vdash_G K_t\big(\varepsilon\langle u\rangle\big)$. The string u must be non-empty, because $\varepsilon\langle u\rangle \in L_{G'}(W)$ (this fulfils III(c) in the statement of the lemma). By Lemma 4.1 for the pair $(\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G)$, the string $u\langle\varepsilon\rangle$ is in $L_G(B)$. Then the desired proposition is deduced in G as $B\big(u\langle\varepsilon\rangle\big)$, $C\big(u\langle v\rangle\big) \vdash_G A\big(u\langle v\rangle\big)$ by the rule $A \to BC$.

- The last case is that the rule r' is (4.6d), that is, $A \to C \,\&\, \triangleleft\varepsilon$, obtained from the rule $A \to BC$ in G and a pair $(\{K_1, \ldots, K_t\}, B)$ in $\textsc{Nullable}(G)$, such that $\varepsilon\langle\varepsilon\rangle \in L_G(K_i)$ for all $i \in \{1, \ldots, t\}$. The proposition $A\big(\varepsilon\langle v\rangle\big)$ is deduced as $C\big(\varepsilon\langle v\rangle\big) \vdash_{G'} A\big(\varepsilon\langle v\rangle\big)$, and for this deduction to be possible, u must be empty, due to the empty context operator $(\triangleleft\varepsilon)$. By the induction hypothesis, the proposition $C\big(\varepsilon\langle v\rangle\big)$ can be deduced in G. By Lemma 4.1, applied to the pair $(\{K_1, \ldots, K_t\}, B)$, the proposition $B\big(\varepsilon\langle\varepsilon\rangle\big)$ is also deduced in G. This allows the deduction $B\big(\varepsilon\langle\varepsilon\rangle\big)$, $C\big(\varepsilon\langle v\rangle\big) \vdash_G A\big(\varepsilon\langle v\rangle\big)$ by the rule $A \to BC$. □

4.4.2 Preservation of unambiguity in null conjunct elimination

An important property of Construction 4.1 is that it preserves unambiguity: if the original grammar is unambiguous, then so is the resulting grammar.

Lemma 4.5. *Let $G = (\Sigma, N, R, S)$ be an unambiguous grammar with contexts and let $G' = (\Sigma, N \cup \{W\}, R', S)$ be the grammar obtained in Construction 4.1. Then G' is unambiguous as well.*

Proof. All concatenations in G' are unambiguous, because they are used only in rules of the form $A \to BC$, which are the same as in G. Then, if some string $u\langle v\rangle$ is generated by such a rule in two different ways, as $u\langle v\rangle = u\langle v_1\rangle \cdot uv_1\langle v_2\rangle = u'\langle v'_1\rangle \cdot u'v'_1\langle v'_2\rangle$, then, by Lemma 4.4 (part III(a)), the same string $u\langle v\rangle$ is generated in two distinct ways in the grammar G as well. This is a contradiction to the unambiguity of G.

To see that the choice of a rule in the new grammar is unambiguous, suppose that some string $u\langle v\rangle$ is generated in the grammar G' by two distinct rules for the same symbol $A \in N$.

First, consider the case when the string $u\langle v\rangle$ is generated in G' by any two rules of the form (4.4)–(4.6d), which were created from two different rules in R. Then, by Lemma 4.4, each of the latter two rules generates the string $u\langle v\rangle$ in G, and this contradicts the assumption that G is unambiguous.

Consider the other possibility, when the string $u\langle v\rangle$ is generated in the grammar G' by two rules, and both of these rules were added to R' by processing a single rule from R. If this single rule is $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$, and it resulted in multiple rules in R', then these must be the two rules (4.5a) and (4.5b). By the construction, this can only happen if $m \geqslant 1$. Since G' does not have any null conjuncts, the former rule (4.5a) may only generate a string with a non-empty context, whereas the latter rule (4.5b) requires an empty context. Hence, no string can be deduced in G' by both rules (4.5a), (4.5b) at the same time.

The only remaining possibility is that $u\langle v\rangle$ is generated in G' by two distinct rules (4.6a)–(4.6d) obtained from a single rule $A \to BC$ in G. For each rule of the form (4.6a)–(4.6d), consider Lemma 4.4 and the corresponding condition III(a)–III(d). In all four cases, the string $u\langle v\rangle$ is defined in the grammar G as a concatenation $u\langle v\rangle = u\langle v_1\rangle \cdot uv_1\langle v_2\rangle$, where $u\langle v_1\rangle \in L_G(B)$ and $uv_1\langle v_2\rangle \in L_G(C)$. The four cases are distinguished by the emptiness status of v_1, v_2 and u: both v_1 and v_2 are non-empty in the first case (4.6a), $v_1 = v$ and $v_2 = \varepsilon$ in the second case (4.6b), while the remaining two cases both have $v_1 = \varepsilon$ and $v_2 = v$, and differ by having $u \neq \varepsilon$ in the third case (4.6c) and $u = \varepsilon$ in the fourth case (4.6d).

Assuming that $u\langle v\rangle$ is generated in G' by two rules of two different types (4.6a)–(4.6d), this yields two distinct partitions of $u\langle v\rangle$ into $u\langle v_1\rangle \in L_G(B)$ and $uv_1\langle v_2\rangle \in L_G(C)$. Thus, the concatenation BC is ambiguous in G, witnessed by the string $u\langle v\rangle$, and this contradicts the assumption.

Let now the two rules be of the same type; that is, let the string $u\langle v\rangle$ be generated either by two distinct rules of the form (4.6b) corresponding to different elements of $\textsc{Nullable}(G)$, or, similarly, by two distinct rules of the form (4.6c). Intuitively, any two different elements of $\textsc{Nullable}(G)$ correspond to the generation of ε in two different ways, which should constitute ambiguity. The following claim formalizes this intuition.

Claim 4.5.1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, let $A \in N$ and $w \in \Sigma^*$, and assume that there exist two distinct sets $U, U' \subseteq N$ with $(U, A), (U', A) \in \textsc{Nullable}(G)$ and $\varepsilon\langle w\rangle \in L_G(K)$ for each $K \in U \cup U'$. Then the grammar is ambiguous.*

Before proving the claim, consider how it can be used to finish the proof of the lemma. There are two cases to consider.

- *The string $u\langle v\rangle$ is generated by two different rules of the form (4.6b).* These rules were created for two distinct pairs (U, C) and (U', C) in $\textsc{Nullable}(G)$, where $U, U' \subseteq N$ and $U \neq U'$. Since $u\langle v\rangle$ is generated by each of these rules, $\varepsilon\langle uv\rangle \in L_G(K)$ for all $K \in U \cup U'$. Then Claim 4.5.1 is applicable to the pairs $(U, C), (U', C) \in \textsc{Nullable}(G)$ and the string $w = uv$. This establishes the ambiguity of the grammar G, which contradicts the assumptions.

- *The string $u\langle v\rangle$ is generated by two distinct rules of the form (4.6c).* Similarly to the previous case, since the rule exists, there are pairs $(U, B), (U', B)$ in $\textsc{Nullable}(G)$, with $U, U' \subseteq N$ and $U \neq U'$, and $\varepsilon\langle u\rangle \in L_G(K)$ for all $K \in U \cup U'$. Thus, Claim 4.5.1 is applicable, and it contradicts the unambiguity of G.

The above case analysis completes the proof of Lemma 4.5, and so it remains to prove Claim 4.5.1.

For the given pairs (U, A), (U', A), let $h \geqslant 1$ be the least number, for which $(U, A), (U', A) \in \textsc{Nullable}_h(G)$. Of all pairs (U, A), (U', A) satisfying the statement of the claim, choose those with the minimal value of h. That is, let (U, A) and (U', A) be two distinct pairs in $\textsc{Nullable}_h(G)$, for some $h \geqslant 1$, and assume that $\varepsilon\langle w\rangle \in L_G(K)$ for all $K \in U \cup U'$. It shall be proved that the choice of a rule for A is ambiguous for the string $w\langle\varepsilon\rangle$.

For the pair (U, A) in $\textsc{Nullable}_h(G)$, by the definition of this set, there exists a rule

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n, \tag{4.7}$$

and pairs $(U_1, \alpha_1), \ldots, (U_k, \alpha_k)$ in $\textsc{Nullable}^{\star}_{h-1}(G)$, for which $U = \{D_1, \ldots, D_m, E_1, \ldots, E_n\} \cup U_1 \cup \ldots \cup U_k$. Similarly, for the other pair $(U', A) \in \textsc{Nullable}_h(G)$, there is a rule

$$A \to \alpha'_1 \,\&\, \ldots \,\&\, \alpha'_{k'} \,\&\, \lhd D'_1 \,\&\, \ldots \,\&\, \lhd D'_{m'} \,\&\, \trianglelefteq E'_1 \,\&\, \ldots \,\&\, \trianglelefteq E'_{n'}, \tag{4.7$'$}$$

and pairs $(U'_1, \alpha'_1), \ldots, (U'_{k'}, \alpha'_{k'}) \in \textsc{Nullable}^{\star}_{h-1}(G)$ satisfying $U' = \{D'_1, \ldots, D'_{m'}, E'_1, \ldots, E'_{n'}\} \cup U'_1 \cup \ldots \cup U'_{k'}$.

First, suppose that (4.7) and (4.7′) are the same rule. Then, $\{D_1, \ldots, D_m, E_1, \ldots, E_n\} = \{D'_1, \ldots, D'_{m'}, E'_1, \ldots, E'_{n'}\}$, and therefore $U \neq$

U' implies $U_1 \cup \ldots \cup U_k \neq U'_1 \cup \ldots \cup U'_k$. For these unions to be different, they should differ in some position $i \in \{1, \ldots, k\}$, for which $U_i \neq U'_i$, and both pairs (U_i, α_i), (U'_i, α_i), with the same α_i, belong to $\textsc{Nullable}^{\star}_{h-1}(G)$.

Let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$, with $\ell_i \geqslant 0$ and $X_{i,1}, \ldots, X_{i,\ell_i} \in N$. By the definition of the "star", since the pair $(U_i, X_{i,1} \ldots X_{i,\ell_i})$ is in $\textsc{Nullable}^{\star}_{h-1}(G)$, it is obtained by combining some pairs $(U_{i,1}, X_{i,1}), \ldots, (U_{i,\ell_i}, X_{i,\ell_i})$ in $\textsc{Nullable}_{h-1}(G)$, which satisfy $U_{i,1} \cup \ldots \cup U_{i,\ell_i} = U_i$. Similarly, for the pair $(U'_i, X_{i,1} \ldots X_{i,\ell_i})$, there are corresponding pairs $(U'_{i,1}, X_{i,1}), \ldots, (U'_{i,\ell_i}, X_{i,\ell_i}) \in \textsc{Nullable}_{h-1}(G)$, and $U'_{i,1} \cup \ldots \cup U'_{i,\ell_i} = U'_i$. Since $U_i \neq U'_i$, these two unions should again differ in some j-th position, that is, there are two distinct pairs $(U_{i,j}, X_{i,j})$ and $(U'_{i,j}, X_{i,j})$ in $\textsc{Nullable}_{h-1}(G)$. Furthermore, as $U_{i,j} \subseteq U$ and $U'_{i,j} \subseteq U'$, it is known that $\varepsilon\langle w\rangle \in L_G(K)$ for all $K \in U_{i,j} \cup U'_{i,j}$. Thus, these two new pairs satisfy all conditions of this claim, for a smaller number of iterations than h. This contradicts the assumption made in the beginning of the proof, and thus demonstrates that the rules (4.7) and (4.7′) are distinct.

Now the goal is to show that both rules (4.7) and (4.7′) can be used to deduce the proposition $A\big(w\langle\varepsilon\rangle\big)$, which will form the desired ambiguity. Consider, for instance, the first rule (4.7). For each conjunct α_i in this rule, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$. Since $(U_i, \alpha_i) \in \textsc{Nullable}^{\star}_{h-1}(G)$, by definition of the "star", there are pairs $(U_{i,1}, X_{i,1})$, …, $(U_{i,\ell_i}, X_{i,\ell_i})$ in $\textsc{Nullable}_{h-1}(G)$, which satisfy $U_{i,1} \cup \ldots \cup U_{i,\ell_i} = U_i$. It is known that $\varepsilon\langle w\rangle$ is in $L_G(K)$ for all $K \in U_{i,j}$. Applying Lemma 4.1 to every pair $(U_{i,j}, X_{i,j})$ with $j \in \{1, \ldots, \ell_i\}$ gives that the string $w\langle\varepsilon\rangle$ is in $L_G(X_{i,j})$. It is also known that $\varepsilon\langle w\rangle \in L_G(D_i)$ for all i, and $\varepsilon\langle w\rangle \in L_G(E_i)$ for all i, because $D_i, E_i \in U$. Then the desired proposition $A\big(w\langle\varepsilon\rangle\big)$ can be deduced by the rule (4.7) as follows.

$$\big\{X_{i,j}\big(w\langle\varepsilon\rangle\big)\big\}_{i,j}, \big\{D_i\big(\varepsilon\langle w\rangle\big)\big\}_i, \big\{E_i\big(\varepsilon\langle w\rangle\big)\big\}_i \vdash_G A\big(w\langle\varepsilon\rangle\big)$$

By the same argument as above, the string $w\langle\varepsilon\rangle$ can be generated using the rule (4.7′). This constitutes ambiguity of the choice of a rule, and completes the proof of Claim 4.5.1, as well as of the entire lemma. □

4.4.3 Elimination of null contexts

The second stage of transforming a grammar to the binary normal form is the elimination of *null context specifications* of the form $\lhd\varepsilon$, which assert that the substring being defined is a prefix of the entire string.

For example, a rule $A \to a \,\&\, \lhd\varepsilon$, where A generates a only in the empty left context, can be replaced with a rule $A \to a \,\&\, \trianglelefteq a$. In the general case, the condition $\lhd\varepsilon$ can be simulated by a conjunct defining all non-empty strings in an empty left context.

Lemma 4.6. *Let $G = (\Sigma, N, R, S)$ be grammar with one-sided contexts, and assume that no symbol $A \in N$ generates the empty string in any context, that is, $L_G(A) \subseteq \Sigma^*\langle\Sigma^+\rangle$. Construct another grammar $G' = (\Sigma, N \cup \{W, X\}, R', S)$, where every rule with a null context*

$$A \to \Phi \mathbin{\&} \lhd\varepsilon \tag{4.8}$$

in R, with Φ being a conjunction of conjuncts of the form α, $\lhd\beta$ and $\trianglelefteq\gamma$, is replaced by a rule

$$A \to \Phi \mathbin{\&} W. \tag{4.9}$$

All rules in R without a null context are kept in R' as they are. The symbol W generates all non-empty strings in an empty left context by the following rules.

$$W \to Wa \qquad (\textit{for all } a \in \Sigma) \tag{4.10a}$$
$$W \to a \mathbin{\&} \trianglelefteq X \qquad (\textit{for all } a \in \Sigma) \tag{4.10b}$$
$$X \to a \qquad (\textit{for all } a \in \Sigma) \tag{4.10c}$$

Then $L_G(A) = L_{G'}(A)$, for all $A \in N$. In particular, $L(G) = L(G')$. Furthermore, if G is unambiguous, then so is G'.

In the rules for W, the length of the current substring is reduced to a single symbol, and the extended context operator is then applied to that symbol. It ensures that the symbol is the prefix of the entire string, that is, its left context is empty.

Proof. $\Leftrightarrow$ It has to be shown that whenever a proposition $A\big(u\langle v\rangle\big)$ is deducible in G, it can also be deduced in G'. The proof is by induction on p, the number of steps used in a deduction of $A\big(u\langle v\rangle\big)$ in G.

Let the proposition $A\big(u\langle v\rangle\big)$ be deduced in p steps in G, and assume that the last step of the deduction uses a rule of the form (4.8), which has an empty left context. The conjunct $\lhd\varepsilon$ requires that $u = \varepsilon$ and thus the deduction takes the form

$$\mathcal{X} \vdash_G A\big(\varepsilon\langle v\rangle\big),$$

where $\mathcal{X}$ is the set of premises corresponding to the conjuncts Φ in the rule. Also note that $v \neq \varepsilon$, because no symbol in G generates the empty string in any context.

Each of the premises in $\mathcal{X}$ is deduced in fewer than p steps, and hence, by the induction hypothesis, is deducible in G'. Using the rules (4.10), one can deduce the proposition $W\big(\varepsilon\langle v\rangle\big)$. Now the desired proposition $A\big(\varepsilon\langle v\rangle\big)$ can be deduced in G' out of the premises $\mathcal{X}$ and $W\big(\varepsilon\langle v\rangle\big)$ by the rule (4.9), which corresponds to the rule (4.8) in the original grammar.

Consider the case when $A\big(u\langle v\rangle\big)$ is deduced in G, and the last step uses some rule r, which does not have null contexts. By the induction hypothesis,

each premise of such deduction can be deduced in the new grammar G' as well. By the construction, the same rule r also is in G', and the desired deduction $\vdash_{G'} A\big(u\langle v\rangle\big)$ can be repeated.

$\Leftarrow$ The converse statement is that $u\langle v\rangle \in L_{G'}(A)$ implies $u\langle v\rangle \in L_G(A)$. It is proved by induction on the number of steps used in the deduction of the proposition $A\big(u\langle v\rangle\big)$ in G'.

Let $A\big(u\langle v\rangle\big)$ be deduced in G', and let the last step of its deduction use a rule of the form (4.9), which was created from a rule (4.8) in G. Then this last step is

$$\mathcal{X}, W\big(u\langle v\rangle\big) \vdash_{G'} A\big(u\langle v\rangle\big),$$

where $\mathcal{X}$ is the set of premises corresponding to the conjuncts Φ in the rule. By the induction hypothesis, all the premises $\mathcal{X}$ can be deduced in G. Since all the strings generated by W are in an empty left context, $u\langle v\rangle \in L_{G'}(W)$ implies that $u = \varepsilon$. Then the desired proposition $A\big(\varepsilon\langle v\rangle\big)$ is deduced in G by the rule (4.8) from the premises $\mathcal{X}$.

Let the proposition $A\big(u\langle v\rangle\big)$ be deduced in G' and let the last step of the deduction use some rule r, which is not of the form (4.9). By the induction hypothesis, each premise used in such deduction is deducible in G. Using these premises, the deduction of $A\big(u\langle v\rangle\big)$ can be carried out in G using the same rule r. This proves that the new grammar generates the same language as the original grammar G.

It remains to show that the transformation preserves unambiguity of a grammar. Consider first the unambiguity of choice of a rule. The rules for nonterminal W define disjoint languages and so do the rules for X. For each nonterminal $A \in N$, the rules for A in G and in G' are in one-to-one correspondence, and the corresponding rules generate the same languages in both grammars. Thus, if these languages were disjoint in G, they will remain disjoint in G'.

Turning to the ambiguity of concatenation, the only new introduced concatenation in G' is Wa, which is unambiguous. Thus, if G is unambiguous, then so is the new grammar G'. □

4.4.4 Elimination of unit conjuncts

The last stage of the transformation to the normal form is removing *unit conjuncts* in rules of the form $A \to B \,\&\, \ldots$, where B is a nonterminal symbol. Already for conjunctive grammars, the only known transformation involves substituting all rules for B into all rules containing a unit conjunct B; in the worst case, this results in an exponential blowup [50]. The same construction applies verbatim to grammars with contexts.

The following lemma shows that a single substitution of rules into a unit conjunct does not affect the language defined by the grammar.

Lemma 4.7. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts. Let*

$$A \to B \,\&\, \Phi \tag{4.11}$$

be any rule with a unit conjunct $B \in N$, where Φ is a conjunction of any remaining conjuncts, which may contain context operators.

I. If $A \neq B$, construct a new grammar $G' = (\Sigma, N, R', S)$, in which the rule (4.11) is replaced by the following rules. Let

$$B \to \Psi_i \tag{4.12}$$

with $1 \leqslant i \leqslant s$, be all rules for the nonterminal B, where each Ψ_i is a conjunction of any conjuncts. Then the rule (4.11) is replaced with s rules

$$A \to \Psi_i \,\&\, \Phi \tag{4.13}$$

for $1 \leqslant i \leqslant s$. The rest of the rules in G' are the same as in G. Then $L_{G'}(C) = L_G(C)$ for all $C \in N$.

II. If $A = B$, then the grammar $G' = (\Sigma, N, R \setminus \{A \to A \,\&\, \Phi\}, S)$, with the rule (4.11) removed and with no rules added, satisfies $L_{G'}(C) = L_G(C)$ for all $C \in N$.

In both cases, if G is unambiguous, then so is G'.

Proof. **Part I ($A \neq B$).** It has to be shown that a proposition $A\big(u\langle v\rangle\big)$ can be deduced in G if and only if it also can be deduced in G'.

$\Rightarrow$ The proof is by induction on p, the number of steps used in the deduction of the proposition $A\big(u\langle v\rangle\big)$ in G.

Let $A\big(u\langle v\rangle\big)$ be deduced in p steps in G, and first assume that the pth step uses a rule of the form (4.11), which has a unit conjunct $B \neq A$. Then the last step of deduction is

$$B\big(u\langle v\rangle\big), \mathcal{X} \vdash_G A\big(u\langle v\rangle\big),$$

where $\mathcal{X}$ denotes the premises corresponding to the conjuncts in Φ. Then the proposition $B\big(u\langle v\rangle\big)$ has been deduced earlier in this deduction. Let (4.12) be the rule used for deducing $B\big(u\langle v\rangle\big)$, and let $\mathcal{Y}$ be all the premises.

$$\mathcal{Y} \vdash_G B\big(u\langle v\rangle\big)$$

All the propositions in the sets $\mathcal{X}$ and $\mathcal{Y}$ are deduced in fewer than p steps, and hence, by the induction hypothesis, they are deducible in G'. Now the new rule (4.13) can be used to deduce the desired proposition in G'.

$$\mathcal{Y}, \mathcal{X} \vdash_{G'} A\big(u\langle v\rangle\big)$$

Consider the case when the proposition $A\big(u\langle v\rangle\big)$ is deduced in G, and the last step of its deduction uses a rule without any unit conjuncts. Then there is the same rule in the new grammar G'. By the induction hypothesis, every premise of deduction of $A\big(u\langle v\rangle\big)$ in G can be deduced in the new grammar. Then, the deduction of $A\big(u\langle v\rangle\big)$ can be repeated in G'.

$\Leftarrow$ The proof of the converse implication is again carried out by an induction on p, the number of steps used in the deduction of $A\big(u\langle v\rangle\big)$ in G'.

Let $A\big(u\langle v\rangle\big)$ be deduced in G', and let the last step of this deduction use some rule $r \in R'$.

Assume that r is a new rule of the form (4.13), obtained from some rules (4.11) and (4.12). Denote the premises of the deduction by r as follows: let $\mathcal{X}$ be the set of premises corresponding to the conjuncts Φ, and let $\mathcal{Y}$ be the premises corresponding to the conjuncts Ψ_i. By the induction hypothesis, all the propositions in $\mathcal{X} \cup \mathcal{Y}$ can be deduced in G. Using these premises, one can deduce the desired proposition $A\big(u\langle v\rangle\big)$ in two steps as follows.

$$\begin{aligned} \mathcal{Y} &\vdash_G B\big(u\langle v\rangle\big) \\ B\big(u\langle v\rangle\big), \mathcal{X} &\vdash_G A\big(u\langle v\rangle\big) \end{aligned}$$

In case the rule r used in the deduction of $A\big(u\langle v\rangle\big)$ in G' is not of the form (4.13) defined in the construction, this rule is also contained in G. All the premises of the last step of the deduction of $A\big(u\langle v\rangle\big)$ in G' are deducible in G by the induction hypothesis, and hence $A\big(u\langle v\rangle\big)$ can be deduced in G.

This proves that the new grammar generates the same language as the original grammar G. Turning to the subject of ambiguity, the construction preserves the unambiguity of concatenation, because no new concatenations are introduced. Assuming that the choice of a rule is unambiguous in G, the proof that it remains unambiguous in G' is carried out by a contradiction; there are the following three cases of a supposed ambiguity in G'.

- Let some proposition $A\big(u\langle v\rangle\big)$ be deduced in G' by two distinct rules of the form (4.13). Then, by the construction, G has two different rules for B of the form (4.12), and each of them can be used to deduce the proposition $B\big(u\langle v\rangle\big)$. This contradicts the assumed unambiguity of G.

- Let now $A\big(u\langle v\rangle\big)$ be deduced in G' by one rule of the form (4.13) and by another rule r of a different form. By construction, G contains r, as well as a rule of the form (4.11), and the deduction $\vdash_G A\big(u\langle v\rangle\big)$ can be made using both these rules. This contradicts the assumption.

- Finally, the case when $A\big(u\langle v\rangle\big)$ has been deduced in G' by two distinct rules, both of which are copied to G' from G, again forms a contradiction, because then the same two rules would produce $A\big(u\langle v\rangle\big)$ in the grammar G.

Part II ($A = B$). Since the set of rules of G' is a proper subset of that of G, every deduction in G' is a deduction in G, and hence $L_{G'}(A) \subseteq L_G(A)$. It has to be shown that, conversely, whenever a proposition $A\big(u\langle v\rangle\big)$ is deduced in G, it can also be deduced in G'. The proof is an induction on the number of steps in the shortest deduction of $A\big(u\langle v\rangle\big)$ in G.

If the last step of the deduction of $A\big(u\langle v\rangle\big)$ in G uses a rule without unit conjuncts, then, by construction, such a rule is also in G'. By the induction hypothesis, each of the premises used in this deduction can be deduced in G' as well. Then the desired proposition $A\big(u\langle v\rangle\big)$ is deduced in G' from the same premises.

Suppose that the last step of the deduction of $A\big(u\langle v\rangle\big)$ in G uses a rule of the form (4.11). One of the premises of this deduction is the same proposition $A\big(u\langle v\rangle\big)$, which must have already been deduced earlier in this deduction. Therefore, there is a shorter deduction of $A\big(u\langle v\rangle\big)$ in G, which contradicts the assumption. □

In the theorem below, Lemma 4.7 is used multiple times to eliminate all unit conjuncts. The theorem summarizes all transformations towards the normal form developed in this section, and asserts that an arbitrary grammar with left contexts can be transformed to the binary normal form. Furthermore, the theorem estimates the worst-case size of the resulting grammar, where the size is measured by the total number of symbols used in the description of the grammar.

Theorem 4.5. *For each grammar with left contexts $G = (\Sigma, N, R, S)$ there exists and can be effectively constructed a grammar with left contexts $G' = (\Sigma, N', R', S')$ in binary normal form, such that $L(G) = L(G')$. The size of G' is at most exponential in the size of G. The construction preserves unambiguity of the grammar.*

Proof. First, the rules of the grammar G are pre-processed according to Lemma 4.2, resulting in a new grammar G_1, which generates the same language. If G is unambiguous, then so is G_1.

Given a grammar G_1, null conjuncts of the form $A \to \ldots \& \varepsilon$ are eliminated by Construction 4.1, leading to another grammar G_2, which, by Lemmata 4.3 and 4.4, defines the same language without the empty string (see Correctness Statement 1). If G_1 is unambiguous, then so is G_2, by virtue of Lemma 4.5.

The grammar G_2 may still have null contexts ($\lhd\varepsilon$), which are eliminated by Lemma 4.6. The new grammar G_3 is proved to generate the same language and the unambiguity of the grammar is preserved.

Unit conjuncts of the form $A \to \ldots \& B$ are removed from G_3 by applying the transformation in Lemma 4.7 multiple times, as in the case of conjunctive grammars [50, Lem. 2]. Again, the unambiguity of the grammar is preserved.

Finally, if $\varepsilon \in L(G)$, then a new initial symbol S' is introduced, each rule $S \to \Phi$ in R' is copied to a rule $S' \to \Phi$, and a new rule $S' \to \varepsilon$ is added. $\square$

4.4.5 Size of the resulting grammar

The preprocessing of the grammar according to Lemma 4.2 may increase the size of the grammar by a constant factor. The elimination of null conjuncts in Construction 4.1 leads to at most exponential blowup in the size of the grammar, because the size of the resulting grammar depends on the number of pairs in the set $\textsc{Nullable}(G)$, which is at most exponential. Applying Lemma 4.6 for eliminating null contexts increases the size by an additive constant. Finally, the elimination of unit conjuncts according to Lemma 4.7 may lead to another exponential blowup [50]. This leads to a rough double exponential upper bound on the size of the resulting grammar in the normal form.

A more careful analysis shows that, in fact, the blowup is bounded by a single exponential function. Consider the set of conjuncts occurring in the rules of the grammar G_1 obtained after the first pre-processing stage. Let all nonterminal symbols of G_1 be included in this set as well; the number of elements in the resulting set is linear in the size of G. Neither Construction 4.1, nor Lemma 4.7 add any new conjuncts to the grammar, whereas Lemma 4.6 adds a bounded number of conjuncts. Thus, the number of different conjuncts in the ultimate resulting grammar is still linear in the size of G, and one can construct at most exponentially many different rules from these conjuncts.

4.4.6 Example of transformation

Example 4.9. Consider the grammar from Example 2.1 and construct an equivalent grammar G' in binary normal form.

First, the grammar is preprocessed according to Lemma 4.2.

$$
\begin{aligned}
S &\to S_a D_0 \mid S_b C_0 && \text{(4.14a)}\\
S &\to \varepsilon \,\&\, {\lhd} A && \text{(4.14b)}\\
A &\to A_a B_0 && \text{(4.14c)}\\
A_a &\to A_0 A\ a && \text{(4.14d)}\\
A &\to \varepsilon && \text{(4.14e)}\\
S_a &\to A_0 S && \text{(4.14f)}\\
S_b &\to B_0 S && \text{(4.14g)}\\
A_0 &\to a && \text{(4.14h)}\\
B_0 &\to b && \text{(4.14i)}\\
C_0 &\to c && \text{(4.14j)}\\
D_0 &\to d && \text{(4.14k)}
\end{aligned}
$$

By Construction 4.1, the rules (4.14a), (4.14c) and (4.14f)–(4.14k) are appended to the new grammar G' without any alterations.

The set $\textsc{Nullable}(G)$ contains the pairs $(\varnothing, A)$ and $(\{A\}, S)$, corresponding to the fact that nonterminal A can always define the empty string, while nonterminal S only generates the empty string in left contexts of the form A. The new grammar has rules $S_a \to A_0 \,\&\, {\lhd} A$ and $S_b \to B_0 \,\&\, {\lhd} A$, originating from the rules (4.14f) and (4.14g) of the old grammar, respectively. The unit conjuncts in these rules are then eliminated, resulting in rules of the form $S_a \to a \,\&\, {\lhd} A$ and $S_b \to b \,\&\, {\lhd} A$.

Rules (4.14b) and (4.14e) of G are not preserved in the new grammar.

Since there are no unit conjuncts in the grammar G', it is in binary normal form.

The complete set of rules of the new grammar in binary normal form is given in Example 4.2.

4.5 Normal form for grammars with two-sided contexts

Similarly to the grammars with one-sided contexts, the transformation of grammars with two-sided contexts to the normal form consists of three stages: first, removing all *empty (null) conjuncts* ε; secondly, eliminating *empty contexts* ($\lhd\varepsilon$, $\rhd\varepsilon$); finally, getting rid of *unit conjuncts* of the form B, with $B \in N$.

4.5.1 Elimination of null conjuncts

In order to eliminate null conjuncts in the case of grammars with two-sided contexts, one has to consider yet another variant of the set $\textsc{Nullable}(G)$, which respects both left and right contexts.

Example 4.10. Consider the following grammar with two-sided contexts, obtained by adding context restrictions to the grammar in Example 4.5; this grammar defines the language $L = \{abc, ac, bcd, bd\}$.

$$\begin{array}{lcl} S & \to & aA \mid Ad \\ A & \to & BC \\ B & \to & \varepsilon \mathbin{\&} {\lhd} D \mid b \\ C & \to & \varepsilon \mathbin{\&} {\rhd} E \mid c \\ D & \to & a \\ E & \to & d \end{array}$$

In this grammar, the nonterminal B generates the empty string only in a left context of the form defined by D, while C defines the empty string only in a right context of the form E. Because of this, both B and C can be omitted in the rule $A \to BC$, giving two rules $A \to C \mathbin{\&} {\lhd} D$ and $A \to B \mathbin{\&} {\rhd} E$. Each of these rules ensures that B (or C) defines ε in this context. In those contexts where *both* B and C generate ε, so can A, by the rule $A \to BC$. Hence, in the rules for S, nonterminal A can be accordingly omitted by having rules $S \to a \mathbin{\&} {\lhd\!\lhd} D \mathbin{\&} {\rhd} E$ and $S \to d \mathbin{\&} {\lhd} D \mathbin{\&} {\rhd\!\rhd} E$. These rules have an extended context operator, since the left context of A includes a in the rule $S \to aA$ and in the rule $S \to Ad$ the right context of A includes d.

After all null conjuncts have been eliminated from the grammar, its rules are as follows.

$$\begin{array}{lcl} S & \to & aA \mid a \mathbin{\&} {\lhd\!\lhd} D \mathbin{\&} {\rhd} E \mid Ad \mid d \mathbin{\&} {\lhd} D \mathbin{\&} {\rhd\!\rhd} E \\ A & \to & BC \mid B \mathbin{\&} {\rhd} E \mid C \mathbin{\&} {\lhd} D \\ B & \to & b \\ C & \to & c \\ D & \to & a \\ E & \to & d \end{array}$$

The information about the left and right contexts, in which a nonterminal generates the empty string, is to be stored in the set $\textsc{Nullable}(G)$, which is defined as a subset of $2^N \times N \times 2^N$. An element (U, A, V) of this set

represents the intuitive idea that A defines ε in a left context of the form described by each nonterminal in U, and in a right context of the form given by nonterminals in V.

For the grammar in Example 4.10, such a set $\textsc{Nullable}(G)$ is constructed as follows.

$$\begin{aligned}
\textsc{Nullable}_0(G) &= \varnothing \\
\textsc{Nullable}_1(G) &= \big\{(\{D\}, B, \varnothing), (\varnothing, C, \{E\})\big\} \\
\textsc{Nullable}_2(G) &= \big\{(\{D\}, B, \varnothing), (\varnothing, C, \{E\}), (\{D\}, A, \{E\})\big\}
\end{aligned}$$

Then $\textsc{Nullable}(G) = \textsc{Nullable}_2(G)$. The elements $(\{D\}, B, \varnothing)$ and $(\varnothing, C, \{E\})$ are obtained directly from the rules of the grammar, and the element $(\{D\}, A, \{E\})$ represents the "concatenation" BC in the rule for A. Note the similarity of this construction to the constructions for the case of context-free grammars and grammars with one-sided contexts, described in Examples 4.5 and 4.6, respectively. The construction given here is different only in recording information about the contexts.

The above "concatenation" of triples $(\{D\}, B, \varnothing)$ and $(\varnothing, C, \{E\})$ should be defined to accumulate both left and right contexts. This can be regarded as a generalization of the Kleene star to sets of triples, denoted by $\textsc{Nullable}^\star(G)$. Formally, $\textsc{Nullable}^\star(G)$ is the set of all triples $(U_1 \cup \ldots \cup U_\ell,\ A_1 \ldots A_\ell,\ V_1 \cup \ldots \cup V_\ell)$ with $\ell \geqslant 0$ and $(U_i, A_i, V_i) \in \textsc{Nullable}(G)$. The symbols A_i are concatenated, while their left and right contexts are accumulated. In the special case when $\ell = 0$, the concatenation of zero symbols is the empty string, and thus $\varnothing^\star = \big\{(\varnothing, \varepsilon, \varnothing)\big\}$.

Before giving a formal definition of the set $\textsc{Nullable}(G)$, assume, for the sake of simplicity, that context operators are only applied to single nonterminal symbols, that is, every rule is of the form

$$\begin{aligned}
A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteqslant E_1 \,\&\, \ldots \,\&\, \trianglelefteqslant E_n \,\&\, \\
\trianglerighteqslant F_1 \,\&\, \ldots \,\&\, \trianglerighteqslant F_{m'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{n'},
\end{aligned} \tag{4.15}$$

with $A \in N$, $k \geqslant 1$, $m, n, m', n' \geqslant 0$, $\alpha_i \in (\Sigma \cup N)^*$ and $D_i, E_i, F_i, H_i \in N$. As will be shown in Lemma 4.11, there is no loss of generality in this assumption.

Definition 4.5. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts with all rules of the form (4.15). Construct the sequence of sets $\textsc{Nullable}_i(G) \subseteq 2^N \times N \times 2^N$, for $i \geqslant 0$, as follows.*

Let $\textsc{Nullable}_0(G) = \varnothing$. Every next set $\textsc{Nullable}_{i+1}(G)$ contains the following triples: for every rule (4.15) and for every k triples (U_1, α_1, V_1), $\ldots$, (U_k, α_k, V_k) in $\textsc{Nullable}_i^\star(G)$, the triple $\big(\{D_1, \ldots, D_m, E_1, \ldots, E_n\} \cup U_1 \cup \ldots \cup U_k,\ \ A,\ \ \{F_1, \ldots, F_{m'}, H_1, \ldots, H_{n'}\} \cup V_1 \cup \ldots \cup V_k\big)$ is in $\textsc{Nullable}_{i+1}(G)$.

Finally, let $\textsc{Nullable}(G) = \bigcup_{i \geqslant 0} \textsc{Nullable}_i(G)$.

The next lemma explains how exactly the set $\textsc{Nullable}(G)$ represents the generation of the empty string by different nonterminals in different contexts.

Lemma 4.8. *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, let $A \in N$ and $u, v \in \Sigma^*$. Then, $u\langle\varepsilon\rangle v \in L_G(A)$ if and only if there is a triple $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\})$ in $\textsc{Nullable}(G)$, such that $\varepsilon\langle u\rangle v \in L_G(J_i)$ for all i and $u\langle v\rangle\varepsilon \in L_G(K_j)$ for all j.*

Proof. $\Rightarrow$ Let $u\langle\varepsilon\rangle v \in L_G(A)$. The existence of a desired triple in $\textsc{Nullable}(G)$ is proved by induction on p, the number of steps in the deduction of the proposition $A\big(u\langle\varepsilon\rangle v\big)$.

Basis. Let $p = 1$. Then, the proposition $A\big(u\langle\varepsilon\rangle v\big)$ is deduced by the rule $A \to \varepsilon$. Since such a rule exists, the triple $(\varnothing, A, \varnothing)$ is in $\textsc{Nullable}_1(G)$.

Induction step. Assume that the proposition $A\big(u\langle\varepsilon\rangle v\big)$ is deduced in $p \geqslant 2$ steps, and consider the rule (4.15) used at the last step of the deduction. For each base conjunct α_i in this rule, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$, with $\ell_i \geqslant 0$ and $X_{i,j} \in \Sigma \cup N$. Then the last step of the deduction is

$$X_{i,j}\big(u\langle\varepsilon\rangle v\big), D_i\big(\varepsilon\langle u\rangle v\big), E_i\big(\varepsilon\langle u\rangle v\big), F_i\big(u\langle v\rangle\varepsilon\big), H_i\big(u\langle v\rangle\varepsilon\big) \vdash_G A\big(u\langle\varepsilon\rangle v\big).$$

Then $u\langle\varepsilon\rangle v \in L_G(X_{i,j})$ for each symbol $X_{i,j}$, and hence, by the induction hypothesis, there exists a triple $(U_{i,j}, X_{i,j}, V_{i,j}) \in \textsc{Nullable}(G)$ that satisfies $\varepsilon\langle u\rangle v \in L_G(J)$ for all $J \in U_{i,j}$, and $u\langle v\rangle\varepsilon \in L_G(K)$ for all $K \in V_{i,j}$. Let $h > 0$ be the least such number that all these triples, for all i and j, are present in the set $\textsc{Nullable}_h(G)$. Then, for each i, the triple (U_i, α_i, V_i), where $U_i = U_{i,1} \cup \ldots \cup U_{i,\ell_i}$ and $V_i = V_{i,1} \cup \ldots \cup V_{i,\ell_i}$, is in the set $\textsc{Nullable}^{\star}_h(G)$. Denote $U = \bigcup_i U_i \cup \{D_1, \ldots, D_m\} \cup \{E_1, \ldots, E_n\}$ and $V = \bigcup_i V_i \cup \{F_1, \ldots, F_{m'}\} \cup \{H_1, \ldots, H_{n'}\}$. Then the triple (U, A, V) is in $\textsc{Nullable}_{h+1}(G)$ by the construction.

It remains to show that $\varepsilon\langle u\rangle v \in L_G(J)$ for all $J \in U$ and $u\langle v\rangle\varepsilon \in L_G(K)$ for all $K \in V$. It has been proved above that $\varepsilon\langle u\rangle v \in L_G(J)$, for symbols J in each set $U_{i,j}$. The remaining symbols in U are obtained from the contexts in the rule (4.15), and the propositions $D\big(\varepsilon\langle u\rangle v\big)$, $E\big(\varepsilon\langle u\rangle v\big)$ are known to be true. For symbols in V, everything is symmetric: $u\langle v\rangle\varepsilon \in L_G(K)$, for all $K \in V_{i,j}$, and the rest of the symbols in V are from the contexts in the rule (4.15).

$\Leftarrow$ Consider any triple $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\})$ in $\textsc{Nullable}(G)$, and let $h \geqslant 0$ be the least number, for which $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}_h(G)$. Assuming that $\varepsilon\langle u\rangle v \in L_G(J_i)$ for all i and $u\langle v\rangle\varepsilon \in L_G(K_j)$ for all j, the goal is to prove that $u\langle\varepsilon\rangle v \in L_G(A)$. The proof goes by induction on h.

Basis. If $h = 0$, then $\textsc{Nullable}_h(G)$ is empty, and no symbols $A \in N$ satisfy the assumptions of the lemma.

Induction step. Let now $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}_h(G)$ and $\varepsilon\langle u\rangle v \in L_G(J_i)$, $u\langle v\rangle\varepsilon \in L_G(K_j)$, for all $i \in \{1, \ldots, s\}$, $j \in \{1, \ldots, t\}$. Then, by Definition 4.5, the grammar has a rule of the form (4.15), such that for every base conjunct α_i, the element $(U_i, \alpha_i, V_i) \in \textsc{Nullable}^{\star}_{h-1}(G)$, and

$$\begin{aligned} U_1 \cup \ldots \cup U_k \cup \{D_1, \ldots, D_m\} \cup \{E_1, \ldots, E_n\} &= \{J_1, \ldots, J_s\}, \\ V_1 \cup \ldots \cup V_k \cup \{F_1, \ldots, F_{m'}\} \cup \{H_1, \ldots, H_{n'}\} &= \{K_1, \ldots, K_t\}. \end{aligned}$$

For each conjunct α_i, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$ and $X_{i,1}, \ldots, X_{i,\ell_i} \in \Sigma \cup N$. Then, by the definition of a "star" of $\textsc{Nullable}_{h-1}(G)$, there exist sets $U_{i,1}, \ldots, U_{i,\ell_i} \subseteq N$, $V_{i,1}, \ldots, V_{i,\ell_i} \subseteq N$, such that $U_{i,1} \cup \ldots \cup U_{i,\ell_i} = U_i$, $V_{i,1} \cup \ldots \cup V_{i,\ell_i} = V_i$ and $(U_{i,j}, X_{i,j}, V_{i,j}) \in \textsc{Nullable}_{h-1}(G)$, for all j. By the induction hypothesis, applied to every symbol $X_{i,j}$, one gets that $u\langle\varepsilon\rangle v \in L_G(X_{i,j})$, for each j.

Repeating the same procedure for every element (U_i, α_i, V_i) of the set $\textsc{Nullable}^{\star}_{h-1}(G)$, gives that $\vdash_G X_{i,1}\big(u\langle\varepsilon\rangle v\big), \ldots, X_{i,\ell_i}\big(u\langle\varepsilon\rangle v\big)$.

Now the proposition $A\big(u\langle\varepsilon\rangle v\big)$ can be deduced in the grammar G using the rule (4.15) as follows.

$$\begin{aligned} &X_{1,1}\big(u\langle\varepsilon\rangle v\big), \ldots, X_{k,\ell_k}\big(u\langle\varepsilon\rangle v\big), \\ &\qquad D_1\big(\varepsilon\langle u\rangle v\big), \ldots, D_m\big(\varepsilon\langle u\rangle v\big), E_1\big(\varepsilon\langle u\rangle v\big), \ldots, E_n\big(\varepsilon\langle u\rangle v\big), \\ &\quad F_1\big(u\langle v\rangle\varepsilon\big), \ldots, F_{m'}\big(u\langle v\rangle\varepsilon\big), H_1\big(u\langle v\rangle\varepsilon\big), \ldots, H_{n'}\big(u\langle v\rangle\varepsilon\big) \vdash_G A\big(u\langle\varepsilon\rangle v\big). \end{aligned}$$

□

The plan is to reconstruct the grammar, so that for every triple $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\})$ in $\textsc{Nullable}(G)$, and for every occurrence of A in the right-hand side of any rule, the new grammar contains a companion rule, in which A is omitted and context operators for J_i and K_i are introduced.

The following case requires special handling in the new grammar. Assume that A generates ε in the empty left context (that is, $u = \varepsilon$ in Lemma 4.8). This is reflected by a triple $(\{J_1, \ldots, J_s\}, A, \{K_1, \ldots, K_t\})$ in $\textsc{Nullable}(G)$, in which all symbols J_i also generate ε in the left context ε, and such a generation may in turn involve some further right context operators. In the new grammar, the left context will be explicitly set to be empty ($\lhd\varepsilon$), whereas all those right contexts should be assembled together with the set $\{K_1, \ldots, K_t\}$, and used in the new rules, where A is omitted.

Example 4.11. Consider the following grammar with two-sided contexts, which defines the singleton language $\{b\}$.

$$
\begin{array}{lcl}
S & \to & AB \\
A & \to & \varepsilon \,\&\, {\lhd}D \\
D & \to & \varepsilon \,\&\, {\rhd}B \\
B & \to & b
\end{array}
$$

By the combination of the rules $A \to \varepsilon \,\&\, {\lhd}D$ and $D \to \varepsilon \,\&\, {\rhd}B$, the grammar allows A to generate the empty string, but only in the left context ε and in the right context defined by B. When A is omitted in the rule $S \to AB$, this condition is simulated by a rule $S \to B \,\&\, {\lhd}\varepsilon \,\&\, {\rhd}B$.

In order to define such a transformation for an arbitrary grammar, the following variant of the set $\textsc{Nullable}(G)$, that assumes empty left contexts, has to be used.

Definition 4.6. *Let $G = (\Sigma, N, R, S)$ be a grammar. Define sets ${\lhd}\varepsilon\text{-}\textsc{Nullable}_i(G) \subseteq N \times 2^N$, with $i \geqslant 0$:*

$$
\begin{aligned}
{\lhd}\varepsilon\text{-}\textsc{Nullable}_0(G) &= \{\, (A, V) \mid (\varnothing, A, V) \in \textsc{Nullable}(G) \,\}, \\
{\lhd}\varepsilon\text{-}\textsc{Nullable}_{i+1}(G) &= \big\{ (A, V \cup V_1 \cup \ldots \cup V_s) \;\big|\; \\
&\qquad (\{J_1, \ldots, J_s\}, A, V) \in \textsc{Nullable}(G), \\
&\qquad\quad \exists\, V_1, \ldots, V_s \subseteq N : (J_i, V_i) \in {\lhd}\varepsilon\text{-}\textsc{Nullable}_i(G) \big\}.
\end{aligned}
$$

Let ${\lhd}\varepsilon\text{-}\textsc{Nullable}(G) = \bigcup_{i \geqslant 0} {\lhd}\varepsilon\text{-}\textsc{Nullable}_i(G)$.

The set ${\lhd}\varepsilon\text{-}\textsc{Nullable}(G)$ is constructed as a least upper bound of an ascending sequence of sets ${\lhd}\varepsilon\text{-}\textsc{Nullable}_i(G)$. The initial set ${\lhd}\varepsilon\text{-}\textsc{Nullable}_0(G)$ contains all such pairs (A, V), that A defines ε in any left context (that is, including the empty one) and right contexts V.

The right contexts are accumulated, when every next set ${\lhd}\varepsilon\text{-}\textsc{Nullable}_i(G)$ with $i > 0$ is calculated. Indeed, consider a triple $(U, A, V) \in \textsc{Nullable}(G)$, where some elements of U are nullable. These elements generate the empty string in some left and right contexts; the left contexts are discarded, whereas the information about the right contexts is stored in the set ${\lhd}\varepsilon\text{-}\textsc{Nullable}(G)$.

The next example demonstrates the definition of the set ${\lhd}\varepsilon\text{-}\textsc{Nullable}$.

Example 4.12. For the grammar in Example 4.11,

$$
\begin{aligned}
\textsc{Nullable}(G) &= \big\{ (\{D\}, A, \varnothing), (\varnothing, D, \{B\}) \big\}, \\
{\lhd}\varepsilon\text{-}\textsc{Nullable}_0(G) &= \big\{ (D, \{B\}) \big\}, \\
{\lhd}\varepsilon\text{-}\textsc{Nullable}_1(G) &= \big\{ (D, \{B\}), (A, \{B\}) \big\},
\end{aligned}
$$

and ${\lhd}\varepsilon\text{-}\textsc{Nullable}(G) = {\lhd}\varepsilon\text{-}\textsc{Nullable}_1(G)$.

The element $(A, \{B\})$ means that every time A generates ε in the empty left context, its right context should be of the form B.

Lemma 4.9. *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, let $A \in N$ and $v \in \Sigma^*$. Then $\varepsilon\langle\varepsilon\rangle v \in L_G(A)$ if and only if there is a pair $(A, \{K_1, \ldots, K_t\})$ in $\lhd\varepsilon$-NULLABLE(G), such that the string $\varepsilon\langle v\rangle\varepsilon$ is in $L_G(K_i)$ for all $i \in \{1, \ldots, t\}$.*

Proof. $\Leftarrow$ Let $K_1, \ldots, K_t \in N$ and $(A, \{K_1, \ldots, K_t\}) \in \lhd\varepsilon$-NULLABLE$(G)$. Then, by Definition 4.5, $(A, \{K_1, \ldots, K_t\}) \in \lhd\varepsilon$-NULLABLE$_h(G)$ for some $h \geqslant 0$. The proof is an induction on h.

Basis. Let $h = 0$. Then, by Definition 4.5, $\lhd\varepsilon$-NULLABLE$_0(G) = \{\, (A, \{K_1, \ldots, K_t\}) \mid (\varnothing, A, \{K_1, \ldots, K_t\}) \in \text{NULLABLE}(G) \,\}$. Since $\varepsilon\langle v\rangle\varepsilon \in L_G(K_i)$ by the assumption and $(\varnothing, A, \{K_1, \ldots, K_t\})$ is in NULLABLE$_0(G)$, applying Lemma 4.8 gives that $u\langle\varepsilon\rangle v \in L_G(A)$ for all $u \in \Sigma^*$. Thus, $\varepsilon\langle\varepsilon\rangle v \in L_G(A)$, as desired.

Induction step. Let $h > 0$. By Definition 4.6, the element $(A, V \cup V_1 \cup \ldots \cup V_s)$ is in $\lhd\varepsilon$-NULLABLE$_h(G)$ if there exist such nonterminals $J_1, \ldots, J_s \in N$ that $(\{J_1, \ldots, J_s\}, A, V)$ is in NULLABLE(G) and every pair (J_i, V_i) is in $\lhd\varepsilon$-NULLABLE$_{h-1}(G)$.

Consider a pair $(J_i, V_i) \in \lhd\varepsilon$-NULLABLE$_{h-1}(G)$, for all $i \in \{1, \ldots, s\}$. By the induction hypothesis, $\varepsilon\langle\varepsilon\rangle v \in L_G(J_i)$. Then $\varepsilon\langle\varepsilon\rangle v \in L_G(A)$ by Lemma 4.8, as desired.

$\Rightarrow$ The proof goes by induction on p, the number of steps used in a deduction of the proposition $A\big(\varepsilon\langle\varepsilon\rangle v\big)$.

Basis. Let $p = 1$ and consider the rule $A \to \varepsilon$. The proposition $A\big(\varepsilon\langle\varepsilon\rangle v\big)$ can be deduced as an axiom. By Definition 4.5, the element $(\varnothing, D, \varnothing)$ is in NULLABLE(G). Hence, $(A, \varnothing) \in \lhd\varepsilon$-NULLABLE$(G)$ by Definition 4.6 and the statement of the lemma is thus satisfied.

Induction step. One has to prove that there exists a set $V \subseteq N$, such that $(A, V) \in \lhd\varepsilon$-NULLABLE$(G)$ and $\varepsilon\langle v\rangle\varepsilon$ belongs to the language of each element of V.

Let the proposition $A\big(\varepsilon\langle\varepsilon\rangle v\big)$ be deduced in p steps and let the last step of its deduction use a rule of the form (4.15), with $\{F_1, \ldots, F_{m'}, H_1, \ldots, H_{n'}\} \subseteq V$.

For a base conjunct α_i of this rule, let $\alpha_i = X_{i,1} \ldots X_{i,\ell_i}$, where $\ell_i \geqslant 0$ and $X_{i,j} \in N$. The last step of the deduction takes the following form.

$$X_{i,j}\big(\varepsilon\langle\varepsilon\rangle v\big), D_i\big(\varepsilon\langle\varepsilon\rangle v\big), E_i\big(\varepsilon\langle\varepsilon\rangle v\big), F_i\big(\varepsilon\langle v\rangle\varepsilon\big), H_i\big(\varepsilon\langle v\rangle\varepsilon\big) \vdash_G A\big(\varepsilon\langle\varepsilon\rangle v\big)$$

That is, $\varepsilon\langle\varepsilon\rangle v \in L_G(X_{i,j})$ for each symbol $X_{i,j}$, and, by the induction hypothesis, there exists a pair $(X_{i,j}, V_{i,j})$ in $\lhd\varepsilon$-NULLABLE(G) with $\varepsilon\langle v\rangle\varepsilon \in L_G(K)$, for all $K \in V_{i,j}$. Then, by construction, the pair $(A, V_{1,1} \cup \ldots \cup V_{k,\ell_k} \cup \{F_1, \ldots, F_{m'}\} \cup \{H_1, \ldots, H_{n'}\})$ is in the set $\lhd\varepsilon$-NULLABLE(G), as desired.

Consider some By Definition 4.6, $(A, V) \in \lhd\varepsilon$-NULLABLE$(G)$ means that there exist nonterminals $J_1, \ldots, J_s \in N$ and sets $V_1, \ldots, V_s \subseteq N$,

such that $(\{J_1, \ldots, J_s\}, A, V \cup V_1 \cup \ldots \cup V_s) \in \textsc{Nullable}(G)$ and $(J_i, V_i) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$.

By the induction hypothesis, $\varepsilon\langle v\rangle\varepsilon$ is in $L_G(K)$, for all $K \in V_i$ and $i \in \{1, \ldots, s\}$. Thus, $\varepsilon\langle v\rangle\varepsilon \in L_G(K)$ (for all $K \in V \cup V_1 \cup \ldots \cup V_s$). $\square$

There is a symmetrically defined set $\rhd\varepsilon\text{-}\textsc{Nullable}(G) \subseteq 2^N \times N$, which characterizes the generation of ε in an empty right context.

Similarly to the set $\lhd\varepsilon\text{-}\textsc{Nullable}(G)$, define the set $\rhd\varepsilon\text{-}\textsc{Nullable}(G)$ as follows.

Definition 4.7. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts. Define symmetrically to Definition 4.6 the set $\rhd\varepsilon\text{-}\textsc{Nullable}(G) \subseteq 2^N \times N$, by setting $\rhd\varepsilon\text{-}\textsc{Nullable}(G) = \bigcup_{i \geqslant 0} \rhd\varepsilon\text{-}\textsc{Nullable}_i(G)$, with*

$$\begin{aligned}
\rhd\varepsilon\text{-}\textsc{Nullable}_0(G) &= \{\, (U, A) \mid (U, A, \varnothing) \in \textsc{Nullable}(G) \,\}, \\
\rhd\varepsilon\text{-}\textsc{Nullable}_{i+1}(G) &= \big\{ (U \cup U_1 \cup \ldots \cup U_t, A) \;\big|\; \\
&\qquad (U, A, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G), \\
&\qquad\quad \exists\, U_1, \ldots, U_t \subseteq N :\ (U_i, K_i) \in \rhd\varepsilon\text{-}\textsc{Nullable}_i(G) \big\}.
\end{aligned}$$

Again, the following characterization for this set is established.

Lemma 4.10. *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, let $A \in N$ and $u \in \Sigma^*$. Then $u\langle\varepsilon\rangle\varepsilon \in L_G(A)$ if and only if there is a pair $(\{J_1, \ldots, J_s\}, A)$ in $\rhd\varepsilon\text{-}\textsc{Nullable}(G)$, such that the string $\varepsilon\langle u\rangle\varepsilon$ is in $L_G(J_i)$ for all $i \in \{1, \ldots, s\}$.*

With the generation of the empty string represented in these three sets, a grammar with two-sided contexts is transformed to the normal form as follows. First, it is convenient to simplify the rules of the grammar, so that every concatenation is of the form BC, with $B, C \in N$, and the context operators are only applied to individual nonterminals. For this, base conjuncts α with $|\alpha| > 2$ and context operators $\lhd\alpha$, $\lhdeq\alpha$, $\rhdeq\alpha$ and $\rhd\alpha$ with $|\alpha| > 1$ are shortened by introducing new nonterminals.

Lemma 4.11. *For every grammar $G_0 = (\Sigma, N_0, R_0, S_0)$, there exists and can be effectively constructed another grammar $G = (\Sigma, N, R, S)$ generating the same language, with all rules of the form:*

$$A \to a \tag{4.16a}$$

$$A \to BC \tag{4.16b}$$

$$A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n \,\&\, \rhdeq F_1 \,\&\, \ldots \,\&\, \rhdeq F_{m'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{n'} \tag{4.16c}$$

$$A \to \varepsilon, \tag{4.16d}$$

with $a \in \Sigma$ and $A, B, C, B_i, D_i, E_i, F_i, H_i \in N$.

Construction 4.2. Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts, with all rules of the form (4.16). Consider the sets NULLABLE(G), $\lhd\varepsilon$-NULLABLE(G) and $\rhd\varepsilon$-NULLABLE(G), and construct another grammar with two-sided contexts $G' = (\Sigma, N, R', S)$, with the following rules.

1. All rules of the form (4.16a) in R are added to R'.

$$A \to a \tag{4.17}$$

2. Every rule of the form (4.16c) is taken over from R to R'.

$$\begin{aligned} A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \,\&\\ \&\, \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{m'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{n'} \end{aligned} \tag{4.18}$$

In the original grammar, this rule (4.16c) may generate strings in empty contexts, as long as the symbols in the context operators ($\lhd D_i$, $\rhd H_i$) are nullable. In the new grammar, the above rule (4.18) will no longer generate those strings, and hence the grammar contains the following additional rules that generate them.

For any collection of m pairs (D_1, V_1), $\ldots$, $(D_m, V_m) \in \lhd\varepsilon$-NULLABLE($G$), with $m \geqslant 1$, add the rule

$$\begin{aligned} A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, E_1 \,\&\, \ldots \,\&\, E_n \,\&\, \trianglerighteq K_1 \,\&\, \ldots \,\&\, \trianglerighteq K_t \,\&\\ \&\, \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{m'} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{n'} \,\&\, \lhd\varepsilon, \end{aligned} \tag{4.19a}$$

where $\{K_1, \ldots, K_t\} = \bigcup_{i=1}^{m} V_i$. Nonterminals D_1, $\ldots$, D_m define the empty string in the right contexts given in the set $\lhd\varepsilon$-NULLABLE(G), and thus the rule (4.16c) can define strings in an empty left context. In order to remove the nullable symbols D_i from that rule, one has to add a conjunct $\lhd\varepsilon$ and preserve the information about the right contexts, in which nonterminals D_i define the empty string. This information is given in the set $\lhd\varepsilon$-NULLABLE(G) and is accordingly represented by the conjuncts $\trianglerighteq K_i$ in the rule. Extended left contexts $\trianglelefteq E_i$ are replaced with base conjuncts E_i, because in the empty left context they have the same effect.

Symmetrically, if (U_1, H_1), $\ldots$, $(U_{n'}, H_{n'}) \in \rhd\varepsilon$-NULLABLE($G$), with $n' \geqslant 1$, then there is a rule

$$\begin{aligned} A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, F_1 \,\&\, \ldots \,\&\, F_{m'} \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\\ \&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \,\&\, \trianglelefteq K_1 \,\&\, \ldots \,\&\, \trianglelefteq K_t \,\&\, \rhd\varepsilon, \end{aligned} \tag{4.19b}$$

where $\{K_1, \ldots, K_t\} = \bigcup_{i=1}^{n'} U_i$. In this case, nonterminal symbols H_i generate the empty string, and are removed from the rule and replaced

with the information about left contexts. Extended contexts $\trianglerighteqslant F_i$ are replaced with base conjuncts F_i.

Finally, if $m, n' \geqslant 1$ and $(D_1, V_1), \ldots, (D_m, V_m) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$, $(U_1, H_1), \ldots, (U_{n'}, H_{n'}) \in \rhd\varepsilon\text{-}\textsc{Nullable}(G)$ then the set R' contains a rule

$$\begin{aligned} A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, E_1 \,\&\, \ldots \,\&\, E_n \,\&\, F_1 \,\&\, \ldots \,\&\, F_{m'} \,\&\\ \&\, K_1 \,\&\, \ldots \,\&\, K_t \,\&\, \lhd\varepsilon \,\&\, \rhd\varepsilon, \end{aligned} \tag{4.19c}$$

where $\{K_1, \ldots, K_t\} = \bigcup_{i=1}^{m} V_i \cup \bigcup_{j=1}^{n'} U_j$.

In this case, both left and right contexts of a string are empty. All the symbols D_i and H_i define ε in the contexts specified in $\lhd\varepsilon\text{-}\textsc{Nullable}(G)$ and $\rhd\varepsilon\text{-}\textsc{Nullable}(G)$. These contexts apply to the entire string and are explicitly stated as $K_1 \,\&\, \ldots \,\&\, K_t$ in the new rule. The null contexts $\lhd\varepsilon$, $\rhd\varepsilon$ limit the applicability of this rule to the whole string. Again, as in the two previous cases, base conjuncts are used instead of extended context operators.

3. Every rule of the form (4.16b) in R is added to R':

$$A \to BC, \tag{4.20}$$

along with the following extra rules, where a nullable nonterminal is omitted and the fact that it generates ε is expressed by context operators.

$$A \to B \,\&\, \trianglelefteqslant J_1 \,\&\, \ldots \,\&\, \trianglelefteqslant J_s \,\&\, \rhd K_1 \,\&\, \ldots \,\&\, \rhd K_t, \quad \text{for } (\{J_1, \ldots, J_s\}, C, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G) \tag{4.21a}$$

$$A \to B \,\&\, \trianglelefteqslant J_1 \,\&\, \ldots \,\&\, \trianglelefteqslant J_s \,\&\, \rhd\varepsilon, \quad \text{for } (\{J_1, \ldots, J_s\}, C) \in \rhd\varepsilon\text{-}\textsc{Nullable}(G) \text{ with } s \geqslant 1 \tag{4.21b}$$

$$A \to C \,\&\, \lhd J_1 \,\&\, \ldots \,\&\, \lhd J_s \,\&\, \trianglerighteqslant K_1 \,\&\, \ldots \,\&\, \trianglerighteqslant K_t, \quad \text{for } (\{J_1, \ldots, J_s\}, B, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G) \tag{4.21c}$$

$$A \to C \,\&\, \trianglerighteqslant K_1 \,\&\, \ldots \,\&\, \trianglerighteqslant K_t \,\&\, \lhd\varepsilon, \quad \text{for } (B, \{K_1, \ldots, K_t\}) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G) \text{ with } t \geqslant 1 \tag{4.21d}$$

In the first case, nonterminal C defines ε in the left contexts J_i and the right contexts K_i, and this restriction is implemented by context operators in the new rule. Since the left context of C includes B, extended context operators ($\trianglelefteqslant J_i$) are used on the left, whereas the right context operators are proper ($\rhd K_i$).

The second case considers the possibility of a nullable nonterminal C, which defines ε in an empty right context. This condition is simulated by the conjunct $\rhd\varepsilon$ and extended left contexts $\trianglelefteqslant J_i$.

The two last rules handle symmetrical cases, when nonterminal B defines the empty string.

Lemma 4.12. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts. Then the grammar $G' = (\Sigma, N', R', S)$ obtained by Construction 4.2 generates the language $L(G') = L(G) \setminus \{\varepsilon\}$.*

Proof. ◎ It is claimed that whenever a proposition $A\big(u\langle w\rangle v\big)$ (with $A \in N$, $u, v \in \Sigma^*$ and $w \in \Sigma^+$) can be deduced in the grammar G, it can also be deduced in the new grammar G'. The proof is by induction on p, the number of steps in the deduction of $A\big(u\langle w\rangle v\big)$ in G.

Basis. Let $p = 1$. Consider a proposition $A\big(u\langle w\rangle v\big)$ deduced in G by a rule of the form $A \to a$. Then $w = a \in \Sigma$ and the last step of the deduction takes form $a\big(u\langle a\rangle v\big) \vdash_G A\big(u\langle a\rangle v\big)$. The same rule $A \to a$ is also contained in the new grammar G', and therefore one can deduce the proposition $A\big(u\langle a\rangle v\big)$ in G' by this rule: $a\big(u\langle a\rangle v\big) \vdash_{G'} A\big(u\langle a\rangle v\big)$.

Induction step. Let the proposition $A\big(u\langle w\rangle v\big)$ be deduced in G. Consider the rule used at the last step of this deduction.

If the rule is $A \to BC$, then the last step of the deduction is $B\big(u\langle w_1\rangle w_2 v\big)$, $C\big(uw_1\langle w_2\rangle v\big) \vdash_G A\big(u\langle w_1 w_2\rangle v\big)$, for some partition $w = w_1 w_2$. Each of the premises is deduced in fewer than p steps. Though the string w is non-empty, one of w_1, w_2 may be empty, and the proof splits into three cases, depending on the emptiness status of these strings.

- If both w_1 and w_2 are non-empty, then both propositions $B\big(u\langle w_1\rangle w_2 v\big)$ and $C\big(uw_1\langle w_2\rangle v\big)$ can be deduced in the grammar G' by the induction hypothesis. Since G' has the same rule $A \to BC$, it can be used to deduce the proposition $A\big(u\langle w\rangle v\big)$ in G' in the same way: $B\big(u\langle w_1\rangle w_2 v\big)$, $C\big(uw_1\langle w_2\rangle v\big) \vdash_{G'} A\big(u\langle w_1 w_2\rangle v\big)$.

- Let $w_1 = w$ and $w_2 = \varepsilon$. Then the last step of the deduction is $B\big(u\langle w\rangle v\big)$, $C\big(uw\langle\varepsilon\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$. By the induction hypothesis, the proposition $B\big(u\langle w\rangle v\big)$ can be deduced in G'. Though the other proposition $C\big(uw\langle\varepsilon\rangle v\big)$ can be deduced only in G, it is reflected by other propositions in G'. The proof splits into two cases, depending on whether v is empty or not.

  First consider the case of $v \neq \varepsilon$. Then, since the proposition $C\big(uw\langle\varepsilon\rangle v\big)$ can be deduced in G, by Lemma 4.8, there exist such nonterminals $J_1, \ldots, J_s, K_1, \ldots, K_t \in N$, that $(\{J_1, \ldots, J_s\}, C, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$ and all propositions $J_i\big(\varepsilon\langle uw\rangle v\big)$ and $K_j\big(uw\langle v\rangle\varepsilon\big)$, for all applicable i and j, can be deduced in the grammar G, and such a deduction takes fewer than p steps. By the induction hypothesis, each of these propositions can be deduced in the new grammar G'. From these premises, the desired proposition $A\big(u\langle w\rangle v\big)$ can be deduced by

the rule (4.21a) as follows: $B\big(u\langle w\rangle v\big)$, $J_1\big(\varepsilon\langle uw\rangle v\big)$, …, $J_s\big(\varepsilon\langle uw\rangle v\big)$, $K_1\big(uw\langle v\rangle\varepsilon\big)$, …, $K_t\big(uw\langle v\rangle\varepsilon\big) \vdash_{G'} A\big(u\langle w\rangle v\big)$.

Let now $v = \varepsilon$. In this case, the last step of the deduction of $A\big(u\langle w\rangle\varepsilon\big)$ is as follows: $B\big(u\langle w\rangle\varepsilon\big)$, $C\big(uw\langle\varepsilon\rangle\varepsilon\big) \vdash_G A\big(u\langle w\rangle\varepsilon\big)$. By the induction hypothesis, the proposition $B\big(u\langle w\rangle\varepsilon\big)$ can be deduced in the new grammar G'. Since $uw\langle\varepsilon\rangle\varepsilon \in L_G(C)$, by Lemma 4.10, there exist $J_1, \ldots, J_s \in N$, such that $(\{J_1, \ldots, J_s\}, C) \in \rhd\varepsilon\text{-}\textsc{Nullable}(G)$, and the propositions $J_1\big(\varepsilon\langle uw\rangle\varepsilon\big)$, …, $J_s\big(\varepsilon\langle uw\rangle\varepsilon\big)$ can be deduced in the grammar G in fewer than p steps. By the induction hypothesis, all these propositions can also be deduced in the grammar G. From these premises, one can deduce the desired proposition in G' by the rule (4.21b) as follows: $B\big(u\langle w\rangle\varepsilon\big)$, $J_1\big(\varepsilon\langle uw\rangle\varepsilon\big)$, …, $J_s\big(\varepsilon\langle uw\rangle\varepsilon\big) \vdash_{G'} A\big(u\langle w\rangle\varepsilon\big)$.

- The case of $w_1 = \varepsilon$ and $w_2 = w$, where the last step of deduction of $A\big(u\langle w\rangle v\big)$ in G is $B\big(u\langle\varepsilon\rangle wv\big)$, $C\big(u\langle w\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$, is handled symmetrically to the above case.

  If $u \neq \varepsilon$, then Lemma 4.8 gives a triple $(\{J_1, \ldots, J_s\}, B, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$, for which $\vdash_{G'} C\big(u\langle w\rangle v\big)$, $\vdash_{G'} J_1\big(\varepsilon\langle u\rangle wv\big)$, …, $J_s\big(\varepsilon\langle u\rangle wv\big)$, $K_1\big(u\langle wv\rangle\varepsilon\big)$, …, $K_t\big(u\langle wv\rangle\varepsilon\big)$, and these deductions take fewer than p steps. From these premises, one can then deduce $A\big(u\langle w\rangle v\big)$ in G' by the rule (4.21c).

  If $u = \varepsilon$, then the last step of the deduction of $A\big(\varepsilon\langle w\rangle v\big)$ takes the form $B\big(\varepsilon\langle\varepsilon\rangle wv\big)$, $C\big(\varepsilon\langle w\rangle v\big) \vdash_G A\big(\varepsilon\langle w\rangle v\big)$. Then, by Lemma 4.9, there is a pair $(B, \{K_1, \ldots, K_t\}) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$, for which all propositions $K_1\big(\varepsilon\langle wv\rangle\varepsilon\big)$, …, $K_t\big(\varepsilon\langle wv\rangle\varepsilon\big)$ are deducible in G' in fewer than p steps. These propositions, together with the proposition $C\big(\varepsilon\langle w\rangle v\big)$, are the premises for the deduction of $A\big(\varepsilon\langle w\rangle v\big)$ by the rule (4.21d).

Consider now the other case, when the last step of the deduction of $A\big(u\langle w\rangle v\big)$ is by some rule (4.16c), and is accordingly of the form

$$B_1\big(u\langle w\rangle v\big), \ldots, B_k\big(u\langle w\rangle v\big), \tag{4.22a}$$
$$D_1\big(\varepsilon\langle u\rangle wv\big), \ldots, D_m\big(\varepsilon\langle u\rangle wv\big), \tag{4.22b}$$
$$E_1\big(\varepsilon\langle uw\rangle v\big), \ldots, E_n\big(\varepsilon\langle uw\rangle v\big), \tag{4.22c}$$
$$F_1\big(u\langle wv\rangle\varepsilon\big), \ldots, F_{m'}\big(u\langle wv\rangle\varepsilon\big), \tag{4.22d}$$
$$H_1\big(uw\langle v\rangle\varepsilon\big), \ldots, H_{n'}\big(uw\langle v\rangle\varepsilon\big) \tag{4.22e}$$
$$\vdash_G A\big(u\langle w\rangle v\big).$$

- If $u \neq \varepsilon$ and $v \neq \varepsilon$, then, by the induction hypothesis, each of the premises can be deduced in the grammar G', and thus $A\big(u\langle w\rangle v\big)$ can

be deduced in G' by the same rule (4.16c), which is in R' by the construction.

- Let in the rule (4.16c) $m \geqslant 1$ and let $u = \varepsilon$ and $v \neq \varepsilon$. The propositions (4.22a), (4.22c)–(4.22e) can still be deduced in G' by the induction hypothesis.

  Since $\varepsilon\langle\varepsilon\rangle wv \in L_G(D_i)$ for all $i \in \{1, \ldots, m\}$, then, by Lemma 4.9, there exist $V_i \subseteq N$ such that $(D_i, V_i) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$, and the following propositions can be deduced, in fewer than p steps.

  $$\vdash_G J\big(\varepsilon\langle wv\rangle\varepsilon\big) \qquad (\text{for every } J \in V_i) \tag{4.23}$$

  By the induction hypothesis, such a proposition can be deduced in the grammar G' as well. Now the proposition $A\big(\varepsilon\langle w\rangle v\big)$ can be deduced in G' out of the premises (4.22a), (4.23), and (4.22c)–(4.22e) by the rule (4.19a). This rule is in R', since $m \geqslant 1$.

- Let in the rule (4.16c) $n' \geqslant 1$ and let $u \neq \varepsilon$ and $v = \varepsilon$. The propositions (4.22a), (4.22b)–(4.22d) can be deduced in G' by the induction hypothesis.

  Since $n' \geqslant 1$, the grammar G' has a rule of the form (4.19b) and the propositions $H_i\big(uw\langle\varepsilon\rangle\varepsilon\big)$ are deducible in G in fewer than p steps. Similarly to the previous case, by Lemma 4.10, the propositions

  $$K_i\big(\varepsilon\langle uw\rangle\varepsilon\big), \tag{4.24}$$

  for all $i \in \{1, \ldots, t\}$, are deducible in G. By the induction hypothesis, one can deduce each of them in the grammar G' as well. The deduction of the desired proposition $A\big(u\langle w\rangle\varepsilon\big)$ in G' out of the premises (4.22a), (4.22b)–(4.22d) and (4.24) can be done by the rule (4.19b), which is in R' by the construction.

- Let now $m \geqslant 1$ and $n' \geqslant 1$ and let $u = v = \varepsilon$. By the construction, the grammar G' has a rule (4.19c).

  By the induction hypothesis, the propositions (4.22a), (4.22c) and (4.22d) are deducible in G'. Similarly to the previous cases, one can show that

  $$\vdash_G J_i\big(\varepsilon\langle w\rangle\varepsilon\big), \vdash_G K_i\big(\varepsilon\langle w\rangle\varepsilon\big). \tag{4.25}$$

  The deduction of $A\big(\varepsilon\langle w\rangle\varepsilon\big)$ can now be carried out by a rule (4.19c) using the premises (4.22a), (4.22c)–(4.22d), and (4.25).

⊆ Conversely, it has to be proved that $\vdash_{G'} A\big(u\langle w\rangle v\big)$ implies that $\vdash_G A\big(u\langle w\rangle v\big)$ and $w \neq \varepsilon$. The proof is by induction on p, the number of steps used in a deduction of the proposition $A\big(u\langle w\rangle v\big)$ in G'.

Basis. Let $p = 1$, and let a proposition $A\big(u\langle w\rangle v\big)$ be deduced in G' by a rule $A \to a$. Then $w = a \in \Sigma$ and the deduction takes the form $a\big(u\langle a\rangle v\big) \vdash_{G'} A\big(u\langle a\rangle v\big)$. Since $A \to a$ is in R by the construction, this deduction can be repeated.

Induction step. Let a proposition $A\big(u\langle w\rangle v\big)$ be deduced in G', and let the last step of this deduction use a rule of the form r'. Then the following cases are possible.

- Let the rule r' be of the form (4.18). That is, the last step of deduction of $A\big(u\langle w\rangle v\big)$ in G' uses the premises (4.22a)–(4.22e). By the induction hypothesis, each of these premises can also be deduced in the grammar G. The proposition $A\big(u\langle w\rangle v\big)$ can be deduced in G out of these premises using the same rule r', which is in R by the construction.

- Let the rule r' be of the form (4.19a). That is, $u = \varepsilon$ and the last step of deduction of $A\big(\varepsilon\langle w\rangle v\big)$ takes the form $B_1\big(\varepsilon\langle w\rangle v\big)$, …, $B_k\big(\varepsilon\langle w\rangle v\big)$, $E_1\big(\varepsilon\langle w\rangle v\big)$, …, $E_n\big(\varepsilon\langle w\rangle v\big)$, $K_1\big(\varepsilon\langle wv\rangle\varepsilon\big)$, …, $K_t\big(\varepsilon\langle wv\rangle\varepsilon\big)$, $F_1\big(\varepsilon\langle wv\rangle\varepsilon\big)$, …, $F_{m'}\big(\varepsilon\langle wv\rangle\varepsilon\big)$, $H_1\big(w\langle v\rangle\varepsilon\big)$, …, $H_{n'}\big(w\langle v\rangle\varepsilon\big) \vdash_{G'} A\big(\varepsilon\langle w\rangle v\big)$. By the induction hypothesis, all of the premises can be deduced in the grammar G.

  By the construction, the rule (4.19a) is added to R', when R contains a rule (4.18) (with $m \geqslant 1$) and there exist (D_1, V_1), …, $(D_m, V_m) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$, such that $\bigcup_{i=1}^{m} V_i = \{K_1, \ldots, K_t\}$. Applying Lemma 4.9 to every pair (D_i, V_i), one can obtain that $\varepsilon\langle\varepsilon\rangle wv \in L_G(D_i)$. Thus, the proposition $A\big(\varepsilon\langle w\rangle v\big)$ can be deduced in G out of the premises $B_i\big(\varepsilon\langle w\rangle v\big)$, $D_i\big(\varepsilon\langle\varepsilon\rangle wv\big)$, $E_i\big(\varepsilon\langle w\rangle v\big)$, $F_i\big(\varepsilon\langle wv\rangle\varepsilon\big)$, and $H_i\big(w\langle v\rangle\varepsilon\big)$ by the rule (4.18).

- Let r' be of the form (4.19b). Symmetrically to the previous case, this rule requires $v = \varepsilon$, and the last step of deduction of $A\big(u\langle w\rangle\varepsilon\big)$ is $B_i\big(u\langle w\rangle\varepsilon\big)$, $F_i\big(u\langle w\rangle\varepsilon\big)$, $D_i\big(\varepsilon\langle u\rangle w\big)$, $E_i\big(\varepsilon\langle uw\rangle\varepsilon\big)$, $K_i\big(\varepsilon\langle uw\rangle\varepsilon\big) \vdash_{G'} A\big(u\langle w\rangle\varepsilon\big)$. All these premises can be deduced in G by the induction hypothesis.

  The rule (4.19b) is only added to G', when G has a rule (4.18) (with $n' \geqslant 1$) and there exist (U_1, H_1), …, $(U_{n'}, H_{n'}) \in \rhd\varepsilon\text{-}\textsc{Nullable}(G)$, such that $\bigcup_{i=1}^{n'} U_i = \{K_1, \ldots, K_t\}$. Similarly to the previous case, one can obtain by Lemma 4.10, that $uw\langle\varepsilon\rangle\varepsilon \in L_G(H_i)$ and deduce the proposition $A\big(u\langle w\rangle\varepsilon\big)$ out of the premises $B_i\big(u\langle w\rangle\varepsilon\big)$, $D_i\big(\varepsilon\langle u\rangle w\big)$, $H_i\big(uw\langle\varepsilon\rangle\varepsilon\big)$, $F_i\big(u\langle w\rangle\varepsilon\big)$, $E_i\big(\varepsilon\langle uw\rangle\varepsilon\big)$ by the rule (4.18).

- Let r' be of the form (4.19c). In this case $u = v = \varepsilon$ and the last step of deduction of $A\big(\varepsilon\langle w\rangle\varepsilon\big)$ in G' is $B_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $E_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $F_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $K_i\big(\varepsilon\langle w\rangle\varepsilon\big) \vdash_{G'} A\big(\varepsilon\langle w\rangle\varepsilon\big)$.

Similarly to the two previous cases, one can conclude that the propositions $D_i\big(\varepsilon\langle\varepsilon\rangle w\big)$ and $H_i\big(w\langle\varepsilon\rangle\varepsilon\big)$ can be deduced in G.

Finally, the deduction of the proposition $A\big(\varepsilon\langle w\rangle\varepsilon\big)$ in G can be carried out using the rule (4.18) (which is in R by the construction) as follows: $B_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $D_i\big(\varepsilon\langle\varepsilon\rangle w\big)$ $E_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $F_i\big(\varepsilon\langle w\rangle\varepsilon\big)$, $H_i\big(w\langle\varepsilon\rangle\varepsilon\big) \vdash_G A\big(\varepsilon\langle w\rangle\varepsilon\big)$.

- Let r' be of the form (4.20). In the grammar G', the last step of deduction of $A\big(u\langle w\rangle v\big)$ takes the form $B\big(u\langle w_1\rangle w_2 v\big)$, $C\big(uw_1\langle w_2\rangle v\big)$ $\vdash_{G'} A\big(u\langle w\rangle v\big)$, for some partition $w = w_1 w_2$. By the induction hypothesis, both of the premises can be deduced in the grammar G. Then, the proposition $A\big(u\langle w\rangle v\big)$ can be deduced out these premises in the grammar G using a rule $A \to BC$, which is in G by the construction: $B\big(u\langle w_1\rangle w_2 v\big)$, $C\big(uw_1\langle w_2\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$.

- Let r' be of the form (4.21a). Then $(\{J_1, \ldots, J_s\}, C, \{K_1, \ldots, K_t\}) \in$ Nullable(G), and the proposition $A\big(u\langle w\rangle v\big)$ is deduced in G' out of the premises $B\big(u\langle w\rangle v\big)$, $J_1\big(\varepsilon\langle uw\rangle v\big)$, $\ldots$, $J_s\big(\varepsilon\langle uw\rangle v\big)$, $K_1\big(uw\langle v\rangle\varepsilon\big)$, $\ldots$, $K_t\big(uw\langle v\rangle\varepsilon\big)$. By the induction hypothesis, the proposition $B\big(u\langle w\rangle v\big)$ can be deduced in G. By Lemma 4.8, the proposition $C\big(uw\langle\varepsilon\rangle v\big)$ can be deduced in G as well. Then, using these two premises, one can carry out the deduction of $A\big(u\langle w\rangle v\big)$ in the grammar G by the rule $A \to BC$: $B\big(u\langle w\rangle v\big)$, $C\big(uw\langle\varepsilon\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$.

- Let r' be of the form (4.21b). In order to deduce the proposition $A\big(u\langle w\rangle v\big)$ by this rule, v must be empty, and the last step of deduction of $A\big(u\langle w\rangle\varepsilon\big)$ is thus $B\big(u\langle w\rangle\varepsilon\big)$, $J_1\big(\varepsilon\langle uw\rangle\varepsilon\big)$, $\ldots$, $J_s\big(\varepsilon\langle uw\rangle\varepsilon\big)$ $\vdash_{G'} A\big(u\langle w\rangle\varepsilon\big)$.

  By the construction, the rule (4.21b) is in R', when $(\{J_1, \ldots, J_s\}, C) \in$ $\rhd\varepsilon$-Nullable(G). Then, by Lemma 4.10, the proposition $C\big(uw\langle\varepsilon\rangle\varepsilon\big)$ can be deduced in the grammar G. The proposition $B\big(u\langle w\rangle\varepsilon\big)$ can be deduced in G by the induction hypothesis. This allows one to obtain the proposition $A\big(u\langle w\rangle\varepsilon\big)$ in G out of these premises as follows: $B\big(u\langle w\rangle\varepsilon\big)$, $C\big(uw\langle\varepsilon\rangle\varepsilon\big) \vdash_G A\big(u\langle w\rangle\varepsilon\big)$.

- Let r' be of the form (4.21c). The last step of the deduction of $A\big(u\langle w\rangle v\big)$ in G' is $C\big(u\langle w\rangle v\big)$, $J_1\big(\varepsilon\langle u\rangle wv\big)$, $\ldots$, $J_s\big(\varepsilon\langle u\rangle wv\big)$, $K_1\big(u\langle wv\rangle\varepsilon\big)$, $\ldots$, $K_t\big(u\langle wv\rangle\varepsilon\big)$. Since the rule (4.21c) has been added to R', the set Nullable(G) contains an element $(\{J_1, \ldots, J_s\}, B, \{K_1, \ldots, K_t\})$, and therefore, by Lemma 4.8, one can deduce a proposition $B\big(u\langle\varepsilon\rangle wv\big)$ in G. Finally, the proposition $A\big(u\langle w\rangle v\big)$ can be deduced in G by the rule $A \to BC$ (which is in R by the construction) as follows: $B\big(u\langle\varepsilon\rangle wv\big)$, $C\big(u\langle w\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$.

- Let r' be of the form (4.21d). This rule requires $u = \varepsilon$, and thus the last step of the deduction of $A\big(\varepsilon\langle w\rangle v\big)$ takes the form $C\big(\varepsilon\langle w\rangle v\big)$, $K_1\big(\varepsilon\langle wv\rangle\varepsilon\big)$, …, $K_t\big(\varepsilon\langle wv\rangle\varepsilon\big) \vdash_{G'} A\big(\varepsilon\langle w\rangle v\big)$. The proposition $C\big(\varepsilon\langle w\rangle v\big)$ can be deduced in G by the induction hypothesis.

  The rule (4.21d) is only added to R', when $(B, \{K_1, \ldots, K_t\}) \in \lhd\varepsilon\text{-}\textsc{Nullable}(G)$. Then, by Lemma 4.9, the proposition $B\big(\varepsilon\langle\varepsilon\rangle wv\big)$ can be deduced in the grammar G. Finally, the proposition $A\big(\varepsilon\langle w\rangle v\big)$ can be deduced in G out of the premises $B\big(\varepsilon\langle\varepsilon\rangle wv\big)$, $C\big(\varepsilon\langle w\rangle v\big)$ by the rule $A \to BC$. □

4.5.2 Null contexts and unit conjuncts

The above construction eliminates the empty string in all base conjuncts, but the resulting grammar may still contain null context specifications ($\lhd\varepsilon$ and $\rhd\varepsilon$), which state that the current substring is a prefix or a suffix of the whole string. These operators are eliminated analogously to the case of grammars with one-sided contexts. First, define a new nonterminal symbol U that generates all non-empty strings in the empty left context and another symbol V that generates all non-empty strings in the empty right context. Then it remains to replace left and right null context operators ($\lhd\varepsilon$, $\rhd\varepsilon$) with U and V, respectively.

Lemma 4.13. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts, and assume that no symbol $A \in N$ generates the empty string in any context. Construct another grammar $G' = (\Sigma, N \cup \{U, V, X\}, R', S)$ as follows.*

All rules in R without null contexts are preserved in R'. In every rule in R, which has null contexts, each conjunct of the form $\lhd\varepsilon$ is replaced with a conjunct U, and each conjunct $\rhd\varepsilon$ is replaced with a conjunct V.

The symbol U defines all non-empty strings in an empty left context and V generates all non-empty strings in an empty right context, using the following rules.

$$U \to Ua \qquad \textit{(for all } a \in \Sigma\textit{)} \tag{4.26a}$$
$$U \to a \,\&\, \trianglelefteq X \qquad \textit{(for all } a \in \Sigma\textit{)} \tag{4.26b}$$
$$V \to aV \qquad \textit{(for all } a \in \Sigma\textit{)} \tag{4.26c}$$
$$V \to a \,\&\, \trianglerighteq X \qquad \textit{(for all } a \in \Sigma\textit{)} \tag{4.26d}$$
$$X \to a \qquad \textit{(for all } a \in \Sigma\textit{)} \tag{4.26e}$$

Then $L_G(A) = L_{G'}(A)$, for all $A \in N$. In particular, $L(G) = L(G')$.

The proof of this lemma proceeds similarly to the case of grammars with one-sided contexts and is thus omitted.

The third stage of the transformation to the normal form is removing the *unit conjuncts* in rules of the form $A \to \ldots \,\&\, B$ and is formulated verbatim to the case of grammars with one-sided contexts.

Lemma 4.14. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts. Let*

$$A \to B \,\&\, \Phi \tag{4.27}$$

be any rule with a unit conjunct $B \in N$, where Φ is a conjunction of conjuncts.

If $A \neq B$, let $B \to \Psi_i$ with $1 \leqslant i \leqslant s$, be all rules for nonterminal B. Then the rule (4.27) can be replaced with a collection of rules $A \to \Psi_i \,\&\, \Phi$, for $1 \leqslant i \leqslant s$, without altering the language generated by the grammar.

If $A = B$, then the rule (4.27) can be removed.

This three-stage transformation proves the following theorem.

Theorem 4.6. *For each grammar with two-sided contexts $G = (\Sigma, N, R, S)$ there exists and can be effectively constructed a grammar with two-sided contexts $G' = (\Sigma, N', R', S)$ in binary normal form, such that $L(G') = L(G) \setminus \{\varepsilon\}$.*

4.6 Recognition in linear space

Both the cubic-time algorithm in Section 4.1 and the algorithm in Section 4.2 use quadratic space, as do their context-free and conjunctive prototypes. For conjunctive grammars, the membership of a string can be recognized in linear space and exponential time [56] by using deterministic rewriting of terms of linearly bounded size. In this section, this method is extended to handle the case of grammars with two-sided contexts, leading to the following result.

Theorem 4.7. *Every language generated by a grammar with two-sided contexts is in* DSPACE(n).

Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts. Assume, without loss of generality, that it is in binary normal form (given by Definition 4.2), that is, each rule of the grammar is of one of the following forms.

$$\begin{aligned} A \;\to\; & B_1C_1 \,\&\, \ldots \,\&\, B_kC_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{n'} \,\&\, \\ & \qquad \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{n''} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m''} \end{aligned} \tag{4.28}$$

$$\begin{aligned} A \;\to\; & a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{n'} \,\&\, \\ & \trianglerighteq F_1 \,\&\, \ldots \,\&\, \trianglerighteq F_{n';} \,\&\, \rhd H_1 \,\&\, \ldots \,\&\, \rhd H_{m''} \end{aligned} \tag{4.29}$$

Moreover, let the set of rules R be linearly ordered and let $w = a_1 \ldots a_n$ be an input string. The linear-space recognition algorithm presented below

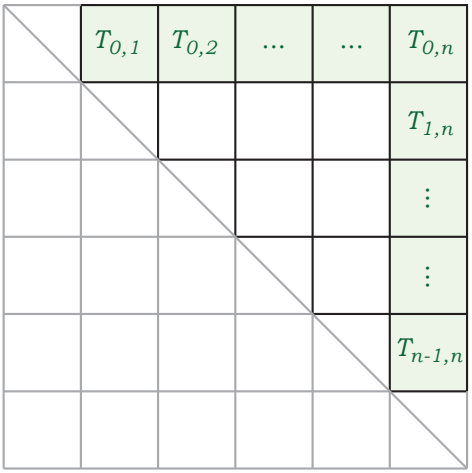


Figure 4.6: Sets constructed by the linear space recognition algorithm for grammars with two-sided contexts.

constructs sets $T_{0,1}$, $T_{0,2}$, ..., $T_{0,n}$, $T_{1,n}$, $T_{2,n}$, ..., and $T_{n-1,n}$ as in the top row and right column of the table in Algorithm 4.1, illustrated in Figure 4.6.

$$T_{0,i} = \{\, A \in N \mid \varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n \in L_G(A) \,\}$$
$$T_{j,n} = \{\, A \in N \mid a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon \in L_G(A) \,\}$$

The membership of symbols in sets $T_{0,i}$ and $T_{j,n}$ is computed using a term rewriting procedure similar to the corresponding procedure for conjunctive grammars [56]. In the case of grammars with two-sided contexts, such a procedure shall additionally verify left and right context operators, by referring to the sets $T_{0,i}$, $T_{0,j}$, $T_{i,n}$, $T_{j,n}$, which are given as partially constructed sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, and $\widehat{T}_{j,n}$, respectively.

Within this section, let a rule of the form (4.28) induce the following schema of *conditional deduction rules* with respect to the sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, $\widehat{T}_{j,n} \subseteq N$, with $1 \leqslant j \leqslant n$, $1 \leqslant i < n$. A proposition $A\big(u\langle w\rangle v\big)$ is deduced by this rule, if for all strings x_1, y_1, ..., x_k, $y_k \in \Sigma^*$, with $x_1y_1 = \ldots = x_ky_k = w$, it holds that $D_\ell \in \widehat{T}_{0,i}$ (with $\ell \in \{1, \dots, m'\}$), $E_\ell \in \widehat{T}_{0,j}$ (with $\ell \in \{1, \dots, n'\}$), $F_\ell \in \widehat{T}_{i,n}$ (with $\ell \in \{1, \dots, n''\}$), and $H_\ell \in \widehat{T}_{j,n}$ (with $\ell \in \{1, \dots, m'\}$).

$$B_1\big(u\langle x_1\rangle y_1 v\big), C_1\big(ux_1\langle y_1\rangle v\big), \dots, B_k\big(u\langle x_k\rangle y_k v\big), C_k\big(ux_k\langle y_k\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$$

A similar schema can be defined for rules of the form (4.29).

Lemma 4.15. *There exists a deterministic term-rewriting procedure*

$$rewrite(A, a_1 \dots a_n, i, j, \widehat{T}_{0,i}, \widehat{T}_{0,j}, \widehat{T}_{i,n}, \widehat{T}_{j,n})$$

which, given a string $a_1 \dots a_n \in \Sigma^+$, two positions i and j within this string, a nonterminal $A \in N$, and partially constructed sets $T_{0,i}$, $T_{0,j}$, $T_{i,n}$, $T_{j,n}$, represented as $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, $\widehat{T}_{j,n} \subseteq N$ (with $1 \leqslant j \leqslant n$, $1 \leqslant i < n$), determines, using linearly bounded space, whether there is a conditional deduction of the proposition $A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle a_{j+1} \dots a_n\big)$ in G with respect to these sets.

When the claimed procedure is invoked for the first time, the sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, and $\widehat{T}_{j,n}$ are empty and the procedure determines whether the proposition $A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle a_{j+1} \dots a_n\big)$ can be deduced in the grammar under the assumption that all context operators are not satisfied. During this run, some symbols may be added to the mentioned sets and some context operators get thus verified. Then, another run of the rewriting procedure, that shall use intermediate statements corresponding to the true contexts, is required. Again, during this second run, more contexts may become verified, and this therefore results in yet another run of the procedure, and so on. These iterative calls of *rewrite* are implemented in Algorithm 4.4 (cf. Algorithm 4.2).

Algorithm 4.4. Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts in binary normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ be the input string.

```
1: while any of T_{0,j} (1 ⩽ j ⩽ n) or T_{i,n} (1 ⩽ i < n) change do
2:     for all nonterminals A ∈ N do
3:         for all i, j with 0 ⩽ i ⩽ n − 1, 1 ⩽ j ⩽ n do
4:             if rewrite(A, w, 0, j, T_{0,1}, T_{0,j}, T_{0,n}, T_{i,n}) then
5:                 T_{0,j} = T_{0,j} ∪ {A}
6:             if rewrite(A, w, i, n, T_{0,1}, T_{0,i}, T_{0,n}, T_{i,n}) then
7:                 T_{i,n} = T_{i,n} ∪ {A}
```

Proof of Lemma 4.15. The claimed rewriting system is constructed as follows. Let R' be the set of rules of G with marked base conjuncts, that is

$$R' = \{\, r_i \mid r_i \text{ is the } i\text{-th base conjunct of } r,\ r \text{ is of the form (4.28) or (4.29)} \,\}.$$

Let $N^+ = \{\, A^+ \mid A \in N \,\}$ and $N^- = \{\, A^- \mid A \in N \,\}$.

For every rule $r = A \to \varphi \in R$, denote $L(r) = L_G(\varphi)$; if the k-th conjunct of r is a base conjunct BC, then denote $L(r_k) = L_G(BC)$.

Terms are represented as strings over the alphabet $\Sigma \cup N \cup N^+ \cup N^- \cup R' \cup \{\text{“(”}, \text{“)”}\}$ as follows:

- for every $u \in \Sigma^+$ and $A \in N$, the following are terms for A: $A(u)$, $A^+(u)$, $A^-(u)$;
- if r_i is a conjunct of the form BC and t_1, t_2 are terms for B and C respectively, then $r_i(t_1 t_2)$ is a term for A.

The *string value* of a term t is defined as follows:

- $\sigma(A(u)) = \sigma(A^+(u)) = \sigma(A^-(u)) = u$ for all $u \in \Sigma^+$;
- $\sigma(r_i(t_1 t_2)) = \sigma(t_1) \cdot \sigma(t_2)$.

Consider any term $\hat{t}(t)$, where t is a distinguished subterm. The *left context* of this subterm within its parent term is defined by replacing t with a special marker $\#$, and then obtaining the string value of all symbols to the left of this marker. This value is given by $\lambda(\hat{t}(\#))$, which is inductively defined as follows.

$$\begin{aligned} \lambda(r_i(t_1 t_2(\#))) &= \sigma(t_1)\lambda(t_2(\#)) \\ \lambda(r_i(t_1(\#) t_2)) &= \lambda(t_1(\#)) \\ \lambda(\#) &= \varepsilon \end{aligned}$$

Similarly, for a term $\hat{t}(t)$ with a distinguished subterm t, one can define the *right context* of the subterm t as follows.

$$\begin{aligned} \rho(r_i(t_1 t_2(\#))) &= \rho(t_2(\#)) \\ \rho(r_i(t_1(\#) t_2)) &= \rho(t_1(\#))\sigma(t_2) \\ \rho(\#) &= \varepsilon \end{aligned}$$

Let $\hat{t}(t)$ be a term with a subterm t. The notion of a *true subterm in the context of a term* is defined inductively on the size of the subterm:

- a term $\hat{t}(t)$ with a subterm $t = A(u)$ is always true;
- a term $\hat{t}(t)$ with a subterm $t = A^+(u)$ is true if and only if $\lambda(\hat{t}(\#))\langle u \rangle \rho(\hat{t}(\#)) \in L(A)$;
- a term $\hat{t}(t)$ with a subterm $t = A^-(u)$ is true if and only if $\lambda(\hat{t}(\#))\langle u \rangle \rho(\hat{t}(\#)) \notin L(A)$;
- a term $\hat{t}(t)$ with a subterm $t = r_i(t_1 t_2)$ is true if and only if all of the following conditions hold:

  I. both subterms t_1 and t_2 of the term $\hat{t}(r_i(t_1 t_2))$ are true;

  II. for every rule r' for A that precedes r, it holds that $\lambda(\hat{t}(\#))\langle \sigma(t) \rangle \rho(\hat{t}(\#)) \notin L(r')$;

III. for every conjunct r_j that precedes r_i in rule r, it holds that $\lambda(\hat{t}(\#))\langle\sigma(t)\rangle\rho(\hat{t}(\#)) \in L(r_j)$;

IV. for every factorization $\sigma(t_1)\sigma(t_2) = uv$, with $0 < |u| < |\sigma(t_1)|$, it holds that $\lambda(\hat{t}(\#))\langle u\rangle v\rho(\hat{t}(\#)) \notin L_G(B)$ or $\lambda(\hat{t}(\#))u\langle v\rangle\rho(\hat{t}(\#)) \notin L_G(C)$, where the conjunct r_i is of the form BC.

Now the set of *rewriting rules* can be defined.

1. In a term $\hat{t}(A(a))$, its subterm $A(a)$ with $a \in \Sigma$ is rewritten as follows.

   If there exists a rule of the form (4.29) in R, for which

   $$D_1, \ldots, D_{m'} \in T_{0,|\lambda(\hat{t}(\#))|}$$
   $$E_1, \ldots, E_{n'} \in T_{0,|\lambda(\hat{t}(\#))\sigma(A(a))|}$$
   $$F_1, \ldots, F_{n''} \in T_{|\sigma(A(a))\rho(\hat{t}(\#))|,n}$$
   $$H_1, \ldots, H_{m''} \in T_{|\rho(\hat{t}(\#))|,n}$$

   then the subterm $A(a)$ is rewritten with $A^+(a)$.

   If there are no rules satisfying the above conditions, then the subterm $A(a)$ is rewritten with $A^-(a)$.

2. Consider a term $\hat{t}(A(u))$ with a subterm $A(u)$ with $u = a_j a_{j+1} \ldots a_\ell$. If there are no rules of the form (4.28) in R, this subterm is rewritten with $A^-(u)$. Otherwise, let r be the first such rule and let $r_1 = B_1C_1$ be its first conjunct. Then the subterm $A(u)$ is rewritten with $r_1(B_1(a_j)C_1(a_{j+1} \ldots a_\ell))$.

3. In a term $\hat{t}(r_i({B_i}^+(u){C_i}^+(v)))$, its subterm $r_i({B_i}^+(u){C_i}^+(v))$ is rewritten as follows.

   - If $i = k$, that is, it is the last base conjunct in this rule, then proceed as follows. Let r be of the form (4.28) and let

     $$D_1, \ldots, D_{m'} \in T_{0,|\lambda(\hat{t}(\#))|}$$
     $$E_1, \ldots, E_{n'} \in T_{0,|\lambda(\hat{t}(\#))\sigma(r_k({B_k}^+(u){C_k}^+(v)))|}$$
     $$F_1, \ldots, F_{n''} \in T_{|\sigma(r_k({B_k}^+(u){C_k}^+(v)))\rho(\hat{t}(\#))|,n}$$
     $$H_1, \ldots, H_{m''} \in T_{|\rho(\hat{t}(\#))|,n}$$

     Then the subterm $r_k({B_k}^+(u){C_k}^+(v))$ is rewritten with $A^+(uv)$. If one of the above conditions is not met, the subterm $r_k({B_k}^+(u){C_k}^+(v))$ is rewritten with $A^-(uv)$.
   - If r_i is not the last base conjunct in the rule (that is, $i \neq k$), the subterm $r_i({B_i}^+(u){C_i}^+(v))$ is rewritten with $r_{i+1}(B_{i+1}(a_j)C_{i+1}(a_{j+1} \ldots a_\ell))$, where $uv = a_j a_{j+1} \ldots a_\ell$ and the $(i+1)$th conjunct in rule r is $B_{i+1}C_{i+1}$.

4. In a term $\hat{t}(t)$, the subterm $t = r_i({B_i}^+(u){C_i}^-(v))$, $t = r_i({B_i}^-(u){C_i}^+(v))$ or $t = r_i({B_i}^-(u){C_i}^-(v))$ is rewritten as follows:
   - if $|v| > 1$, let $v = ax$ (with $a \in \Sigma$, $x \in \Sigma^+$) and rewrite t with $r_i(B_i(ua)C_i(x))$;
   - if $|v| = 1$ and r is the last rule for A, then t is rewritten with $A^-(uv)$;
   - if $|v| = 1$ and r' is the next rule for A, then t is rewritten with $r'_1(B'_1(a_j)C'_1(a_{j+1} \dots a_\ell))$, where $uv = a_j a_{j+1} \dots a_\ell$ and the first conjunct of r' is $B'_1 C'_1$.

Claim 4.15.1. *The rewriting preserves truth, that is, a true term is rewritten with a true term.*

There are numerous cases of rewriting to consider. This proof handles a representative case, and the rest are omitted for brevity.

Consider a term $\hat{t}(r_k({B_k}^+(a_{i+1} \dots a_\ell){C_k}^+(a_{\ell+1} \dots a_j)))$ and the rewriting of the subterm

$$r_k({B_k}^+(a_{i+1} \dots a_\ell){C_k}^+(a_{\ell+1} \dots a_j)) \tag{4.30a}$$

with a subterm

$$A^+(a_{i+1} \dots a_j), \tag{4.30b}$$

where $r_k = B_k C_k$ is the last conjunct of a rule r for nonterminal A. According to the definition of a true subterm, the subterm (4.30b) is true if $a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle a_{j+1} \dots a_n \in L_G(A)$.

Let the rule r be of the form

$$\begin{aligned} A \;\to\; & B_1 C_1 \,\&\, \dots \,\&\, B_k C_k \,\&\, \lhd D_1 \,\&\, \dots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \dots \,\&\, \trianglelefteq E_{n'} \,\&\, \\ & \trianglerighteq F_1 \,\&\, \dots \,\&\, \trianglerighteq F_{n''} \,\&\, \rhd H_1 \,\&\, \dots \,\&\, \rhd H_{m''}. \end{aligned}$$

Since the subterm (4.30a) is true by assumption, its subterms ${B_k}^+(a_{i+1} \dots a_\ell)$ and ${C_k}^+(a_{\ell+1} \dots a_j)$ are true as well. Hence, $a_1 \dots a_i \langle a_{i+1} \dots a_\ell \rangle a_{\ell+1} \dots a_n \in L_G(B_k)$ and $a_1 \dots a_\ell \langle a_{\ell+1} \dots a_j \rangle a_{j+1} \dots a_n \in L_G(C_k)$.

Note, that i is the length of the left context of the true term (4.30a), j is the length of the concatenation of (4.30a) and its left context, $n - i$ is the length of concatenation of the latter term with its right context and $n - j$ is the length of the right context of the former term.

$$\begin{aligned} |\lambda(\hat{t}(\#))| = |a_1 \dots a_i| &= i \\ |\lambda(\hat{t}(\#))\sigma(r_k({B_k}^+(a_{i+1} \dots a_\ell){C_k}^+(a_{\ell+1} \dots a_j)))| &= j \\ |\sigma(r_k({B_k}^+(a_{i+1} \dots a_\ell){C_k}^+(a_{\ell+1} \dots a_j)))\rho(\hat{t}(\#))| &= n - i \\ |\rho(\hat{t}(\#))| = |a_{j+1} \dots a_n| &= n - j \end{aligned}$$

For the rewriting (4.30) to proceed, the following conditions should hold.

$$\begin{aligned} D_1, \ldots, D_{m'} &\in T_{0,i} \\ E_1, \ldots, E_{n'} &\in T_{0,j} \\ F_1, \ldots, F_{n''} &\in T_{i,n} \\ H_1, \ldots, H_{m''} &\in T_{j,n} \end{aligned}$$

Then, by the assumption of Lemma 4.15, the following can be obtained.

$$\begin{aligned} \varepsilon\langle a_1 \ldots a_i\rangle a_{i+1} \ldots a_n &\in \lhd L_G(D_1) \cap \ldots \cap \lhd L_G(D_{m'}), \\ \varepsilon\langle a_1 \ldots a_j\rangle a_{j+1} \ldots a_n &\in \lhdeq L_G(E_1) \cap \ldots \cap \lhdeq L_G(E_{n'}), \\ a_1 \ldots a_i\langle a_{i+1} \ldots a_n\rangle \varepsilon &\in \rhdeq L_G(F_1) \cap \ldots \cap \rhdeq L_G(F_{n''}), \\ a_1 \ldots a_j\langle a_{j+1} \ldots a_n\rangle \varepsilon &\in \rhd L_G(H_1) \cap \ldots \cap \rhd L_G(H_{m''}). \end{aligned}$$

Therefore, the string $a_1 \ldots a_i\langle a_{i+1} \ldots a_j\rangle a_{j+1} \ldots a_n$ is in $L_G(A)$, and the subterm $A^+(a_{i+1} \ldots a_j)$ is true.

Numerous other cases needed to prove Claim 4.15.1 can be handled in a similar way.

Claim 4.15.2 ([56])**.** *The rewriting preserves string value of terms. Moreover, there exists a linear order on the set of terms, such that the rewriting strictly increases the terms with respect to it.*

The claim can be proved similarly to the case of conjunctive grammars [56], with the only difference being that one has to additionally verify conditions on context operators.

The linear order on terms shall be defined as follows [56]. Let t_1 and t_2 be two terms. If $\sigma(t_1)$ is lexicographically less than $\sigma(t_2)$, then $t_1 < t_2$, and if $\sigma(t_1)$ is lexicographically greater than $\sigma(t_2)$, then $t_1 > t_2$. For terms with the same string value w, the general order is

$$\begin{aligned} A_1(w) < \ldots < A_m(w) < r_k(t_1 t_2) < \\ < A_1^+(w) < \ldots < A_m^+(w) < A_1^-(w) < \ldots < A_m^-(w). \end{aligned}$$

The relation between two terms $r_k(t_1 t_2)$ and $r'_\ell(t_3 t_4)$ is defined by saying that if $r \neq r'$, then the terms compare by the order on R; if $r = r'$ and $k < \ell$, then $r_k(t_1 t_2) < r'_\ell(t_3 t_4)$; if $r = r'$ and $k > \ell$, then $r_k(t_1 t_2) > r'_\ell(t_3 t_4)$; and finally, any terms $r_k(t_1 t_2)$ and $r_k(t_3 t_4)$ compare lexicographically as (t_1, t_2) and (t_3, t_4).

The above Claim 4.15.2 implies that after finitely many steps the rewriting shall converge to some term of the same string value, which cannot be further rewritten. Since only terms of the form $A^+(u)$ and $A^-(u)$ cannot be rewritten, this implies that the rewriting has the following outcome.

Claim 4.15.3. *Every term $A(u)$ is eventually converted either to $A^+(u)$ or to $A^-(u)$.*

This concludes the proof of Lemma 4.15. □

Let us now show that if a proposition can be conditionally deduced with respect to the partially constructed sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, and $\widehat{T}_{j,n}$, then there is a deduction of this proposition in the grammar.

Lemma 4.16. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts, let $a_1 \dots a_n \in \Sigma^+$ be a string and consider the partially constructed sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, and $\widehat{T}_{j,n}$, with*

$$\widehat{T}_{0,i} \subseteq T_{0,i} = \{\, D \in N \mid\!\vdash_G D\big(\varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n\big) \,\}, \tag{4.31a}$$

$$\widehat{T}_{0,j} \subseteq T_{0,j} = \{\, E \in N \mid\!\vdash_G E\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big) \,\}, \tag{4.31b}$$

$$\widehat{T}_{i,n} \subseteq T_{i,n} = \{\, F \in N \mid\!\vdash_G F\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big) \,\}, \tag{4.31c}$$

$$\widehat{T}_{j,n} \subseteq T_{j,n} = \{\, H \in N \mid\!\vdash_G H\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big) \,\}, \tag{4.31d}$$

with $1 \leqslant j \leqslant n$, $1 \leqslant i < n$. Then, if a proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$ can be conditionally deduced in G with respect to the sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, $\widehat{T}_{j,n}$, then this proposition can be deduced in G.

Indeed, the conditional deduction of the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$ with respect to the sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, $\widehat{T}_{j,n}$ implicitly uses propositions of the form $D\big(\varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n\big)$, $E\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$, $F\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big)$, $H\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big)$, with $D \in \widehat{T}_{0,i}$, $E \in \widehat{T}_{0,j}$, $F \in \widehat{T}_{i,n}$, and $H \in \widehat{T}_{j,n}$. These intermediate statements can be deduced in the grammar G and are used in the deduction of the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$.

In the other direction, it has to be shown that if a proposition can be deduced in the grammar, then there is a conditional deduction of this proposition, with respect to sets $\widehat{T}_{0,i}$, $\widehat{T}_{0,j}$, $\widehat{T}_{i,n}$, and $\widehat{T}_{j,n} \subseteq N$. For a nonterminal $A \in N$, define its *height* with respect to a string $w = a_1 \dots a_n$ and two positions i, j within this string, denoted by $h_{w,i,j}(A)$, as follows. Consider the shortest deduction of the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$. If no intermediate statements of the form $E\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$ and $F\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big)$ are ever used in this deduction, then let $h_{w,i,j}(A) = 1$. Otherwise, let $h_{w,i,j}(A) = \max_{E,F}(h_{w,i,j}(E), h_{w,i,j}(F)) + 1$, where the maximum is taken over all such E, $F \in N$, that intermediate statements of the form $E\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$ and $F\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big)$ are used in the deduction.

Lemma 4.17. *Let $G = (\Sigma, N, R, S)$ be a grammar with two-sided contexts, let $w = a_1 \dots a_n \in \Sigma^+$ be a string and consider a nonterminal $A \in N$. Then:*

- *if a proposition $A\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$ can be deduced in G and $h_{w,0,j}(A) = h_0$, then this proposition can be conditionally deduced with respect to the sets*

$$\widehat{T}_{0,i} = \{\, \widehat{D} \in N \mid \vdash_G \widehat{D}\big(\varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n\big),\ h_{w,0,i}(\widehat{D}) < h_0 \,\}, \tag{4.32a}$$

$$\widehat{T}_{0,j} = \{\, \widehat{E} \in N \mid \vdash_G \widehat{E}\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big),\ h_{w,0,j}(\widehat{E}) < h_0 \,\}, \tag{4.32b}$$

$$\widehat{T}_{i,n} = \{\, \widehat{F} \in N \mid \vdash_G \widehat{F}\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big),\ h_{w,i,n}(\widehat{F}) < h_0 \,\}, \tag{4.32c}$$

$$\widehat{T}_{j,n} = \{\, \widehat{H} \in N \mid \vdash_G \widehat{H}\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big),\ h_{w,j,n}(\widehat{H}) < h_0 \,\}; \tag{4.32d}$$

- *if a proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big)$ can be deduced in G and $h_{w,i,n}(A) = h_0$, then this proposition can be conditionally deduced with respect to the sets (4.32).*

Let us now prove correctness of Algorithm 4.4; this shall establish Theorem 4.7.

Proof of Theorem 4.7. $\Leftarrow$ In one direction, it has to be shown that if the algorithm ever adds a nonterminal A to a set $T_{i,j}$ (with $i = 0$ or $j = n$), then the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$ is deducible in the grammar G. The proof is an induction on the number of steps in the computation of the algorithm.

Consider line 5, which adds a new element A to the (partially constructed) set $T_{0,j}$. By Lemma 4.15, the procedure *rewrite*, called in line 4, correctly determines whether the proposition $A\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$ can be conditionally deduced in the grammar G with respect to the sets $T_{0,i}$, $T_{0,j}$, $T_{i,n}$, $T_{j,n}$.

By the induction hypothesis, for all elements $D \in T_{0,i}$, $E \in T_{0,j}$, $F \in T_{i,n}$, and $H \in T_{j,n}$, the propositions $D\big(\varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n\big)$, $E\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$, $F\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big)$, $H\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big)$ are deducible in the grammar G. Then, by Lemma 4.16, the proposition $A\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big)$ can be deduced in the grammar G.

The case when A is added to the set $T_{i,n}$ by line 7 is proved similarly.

$\Rightarrow$ Conversely, one has to show that if the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$ can be deduced in G, then the algorithm eventually adds A to $T_{i,j}$. The proof goes by induction on the height in the deduction of the proposition $A\big(a_1 \dots a_i\langle a_{i+1} \dots a_j\rangle a_{j+1} \dots a_n\big)$.

Let the height of the nonterminal A with respect to the string $w = a_1 \dots a_n$ and positions i, j be $h_{w,i,j}(A) = h_0$. Then, by Lemma 4.17, there is a conditional deduction of this proposition with respect to the sets

$$\widehat{T}_{0,i} = \{\, \widehat{D} \in N \mid\!\vdash_G \widehat{D}\big(\varepsilon\langle a_1 \dots a_i\rangle a_{i+1} \dots a_n\big),\ h_{w,0,i}(\widehat{D}) < h_0 \,\},$$
$$\widehat{T}_{0,j} = \{\, \widehat{E} \in N \mid\!\vdash_G \widehat{E}\big(\varepsilon\langle a_1 \dots a_j\rangle a_{j+1} \dots a_n\big),\ h_{w,0,j}(\widehat{E}) < h_0 \,\},$$
$$\widehat{T}_{i,n} = \{\, \widehat{F} \in N \mid\!\vdash_G \widehat{F}\big(a_1 \dots a_i\langle a_{i+1} \dots a_n\rangle\varepsilon\big),\ h_{w,i,n}(\widehat{F}) < h_0 \,\},$$
$$\widehat{T}_{j,n} = \{\, \widehat{H} \in N \mid\!\vdash_G \widehat{H}\big(a_1 \dots a_j\langle a_{j+1} \dots a_n\rangle\varepsilon\big),\ h_{w,j,n}(\widehat{H}) < h_0 \,\}.$$

Then, by the induction hypothesis, the algorithm eventually adds elements $\widehat{D}$ (with $h_{w,0,i}(\widehat{D}) < h_0$), $\widehat{E}$ (with $h_{w,0,j}(\widehat{E}) < h_0$), $\widehat{F}$ (with $h_{w,i,n}(\widehat{F}) < h_0$), $\widehat{H}$ (with $h_{w,j,n}(\widehat{H}) < h_0$) to the sets $T_{0,i}$, $T_{0,j}$, $T_{i,n}$, and $T_{j,n}$, respectively. During the next iteration of the while loop in line 1, these sets are given as arguments to the procedure *rewrite* in lines 4 and 6. Then, by Lemma 4.15, line 5 of the algorithm adds to the set $T_{0,j}$ the nonterminal A generating a prefix of length j of the input string. It can be proved in a similar way, that line 7 adds the element A, defining the suffix $a_{i+1} \dots a_n$, to the set $T_{i,n}$.

As in Algorithm 4.2, the input string is accepted if the initial symbol S of the grammar is contained in the set $T_{0,n}$, which represents the whole input string. □

Chapter 5

Generalized LR parsing

The LR(k) parsing algorithm, invented by Knuth [40], is one of the most well-known and widely used parsing methods. This algorithm applies to a subclass of the context-free grammars, and, for every grammar from this subclass, its running time is linear in the length of the input string. The *Generalized LR* (GLR) algorithm is an extension of the LR parsing applicable to the whole class of context-free grammars: wherever an LR(k) parser would not be able to proceed deterministically, a Generalized LR parser simulates all available actions and stores the results in a graph-structured stack. This simulation technique was discovered by Lang [47], whereas Tomita [88, 89] later independently reintroduced it as a practical parsing algorithm. On an LR(k) grammar, a GLR parser always works in linear time; it may work slower on other grammars, though, when carefully implemented, its running time is at most cubic in the length of the input [39]. These properties make the Generalized LR algorithm the most practical parsing method for context-free grammars beyond LR(k).

The Generalized LR also applies to some extensions of context-free grammars. In particular, the GLR algorithm for conjunctive grammars works in time $\mathcal{O}(n^3)$ [52], where n is the length of the input string, whereas its extension to Boolean grammars has worst-case running time $\mathcal{O}(n^4)$ [60].

In this chapter the Generalized LR parsing algorithm is extended to handle grammars with *left* contexts. As compared to the familiar algorithm for context-free grammars, the new algorithm has to check multiple conditions for a reduction operation, some of them referring to the context of the current substring. The conditions are represented as paths in a graph-structured stack, and accordingly require table-driven graph searching techniques. In spite of these complications, a direct implementation of the algorithm works in time $\mathcal{O}(n^4)$, whereas a more careful implementation leads to a $\mathcal{O}(n^3)$ upper bound on its running time. On "good" grammars, the running time can be as low as linear.

5.1 Data structure and operations on it

The algorithm introduced in this chapter is an extension of the GLR parsing algorithm for conjunctive grammars [52], which in its turn is an extension of Tomita's algorithm [89] for ordinary grammars. These algorithms are all based on the same data structure: the *graph-structured stack*, which represents the contents of the linear stack of a standard LR parser in all possible branches of a nondeterministic computation. The graph has a designated *initial node*, which represents the bottom of the stack. The arcs of the graph are labelled with symbols from $\Sigma \cup N$. For each path in the graph, the concatenation of labels of its arcs forms a string from $(\Sigma \cup N)^*$.

Let $a_1 \dots a_n$, with $n \geqslant 0$ and $a_1, \dots, a_n \in \Sigma$, be the input string. Each node in the graph-structured stack has a corresponding position in the input string. All nodes corresponding to the same position form a *layer* of the graph. Every arc from any node in the i-th layer to any node in the j-th layer labelled with a symbol $X \in \Sigma \cup N$ indicates that the substring $a_{i+1} \dots a_j$ has the property X; such an arc is possible only if $i \leqslant j$. The graph is accordingly drawn from left to right, in the ascending order of layers. At the beginning of the computation, the initial node forms layer 0. The p-th layer is added to the graph when the p-th symbol of the input is read. The layer corresponding to the last read input symbol is called *the top layer* of the stack.

For the time being, let the nodes of the graph be unlabelled. Later on, once the parsing table is defined, they will be labelled with the states of a finite automaton, and a node will be uniquely identified by its layer number and its label. Accordingly, the graph will contain $\mathcal{O}(n)$ nodes. The parser will use those states to determine which operations to apply. This section defines those operations, but not the conditions of applying them.

The computation of the algorithm is an alternation of *reduction phases*, when new arcs going to the existing top layer are added to the graph, and *shift phases*, when an input symbol is read and a new top layer is created.

In the **shift phase**, the algorithm reads a new input symbol and makes a transition from each node in the current top layer p to a node in next top layer $p + 1$ by the recently read $(p + 1)$-th symbol of the input string. The nodes created during a shift phase form the new top layer of the graph. Some transitions may fail, and the corresponding branches of the graph are removed. This operation is illustrated in Figure 5.1(a). Note that the decisions made for each state of the current top layer (that is, whether to extend it to the next layer, and to which node) are determined by a finite automaton based on the node labels; this will be discussed in the next section. The branches of the graph-structured stack that do not get extended to the new top layer are removed. If this is the case for all the nodes, then the entire graph is deleted and the algorithm terminates, reporting a syntax error.

In the **reduction phase**, the algorithm performs all possible *reduction operations*. A reduction is done whenever there is a collection of paths leading to the top layer that represent all conjuncts in a certain rule of the grammar. If these paths are found, then a new arc labelled with the nonterminal on the left-hand side of the rule is added to the graph. The application of this operation is actually triggered by the states in the top layer nodes; this will be discussed in Section 5.2.

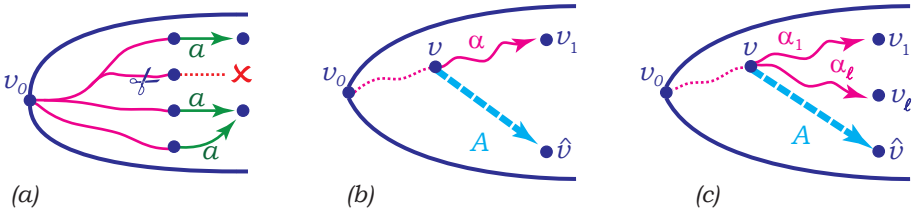


Figure 5.1: *(a)* Shift phase; *(b)* reduction operation for an ordinary context-free grammar; *(c)* reduction in the case of a conjunctive grammar.

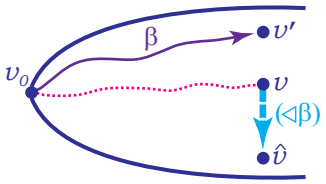
The details of a reduction shall first be explained for an *ordinary grammar*. A reduction by a rule $A \to \alpha$ is done as follows. If there is a path α from some node v to any node v_1 in the top layer, then a new arc labelled A from v to another node $\hat{v}$ can be added, as shown in Figure 5.1(b).

The case of a *conjunctive grammar* is illustrated in Figure 5.1(c). Here a rule may consist of multiple conjuncts, and the condition of performing a reduction has multiple premises, one for each conjunct in the rule. Consider a rule of the form $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell$: if there are paths $\alpha_1, \ldots, \alpha_\ell$ from a node v to any nodes $v_1, \ldots, v_\ell$ in the top layer, then a new arc labelled A going from v to another node $\hat{v}$ can be added.

The case of *grammars with left contexts* is more complicated. To begin with, consider a reduction by a rule without proper left contexts (extended contexts are allowed): $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \preccurlyeq\gamma_1 \,\&\, \ldots \,\&\, \preccurlyeq\gamma_n$. The base conjuncts α_i are checked as in a conjunctive grammar: one has to ensure the existence of paths $\alpha_1, \ldots, \alpha_\ell$ from a node v to any nodes $v_1, \ldots, v_\ell$ in the top layer. Furthermore, for each context operator $\preccurlyeq\gamma_i$, there should be a path γ_i *from the initial node* of the graph to a node v_i' in the top layer, corresponding to the first p symbols of the input. This case is illustrated in Figure 5.2(a).

In order to handle proper left contexts within this setting, a new type of arc labels is introduced. Arcs of the graph-structured stack shall now be labelled with symbols from the alphabet $\Sigma \cup N \cup \mathfrak{C}_\lhd$, where the set $\mathfrak{C}_\lhd$ contains special symbols of the form $(\lhd\beta)$, each representing a proper context operator and regarded as an indivisible symbol.

Let $\mathfrak{C}_{\lhd} = \{\,(\lhd\beta) \mid \lhd\beta \in \mathit{conjuncts}(R)\,\}$ and $\mathfrak{C}_{\lhd} = \{\,(\lhd\gamma) \mid \lhd\gamma \in \mathit{conjuncts}(R)\,\}$.



An arc labelled with a special symbol $(\lhd\beta)$ always connects two nodes in the same layer. Such an arc indicates that the corresponding context $\lhd\beta$ is recognized at this position, that is, the preceding prefix of the string is of the form β. This kind of arc is added to the graph by a new operation called *context validation*. Assume that some rule of the grammar contains a conjunct $\lhd\beta$. If there is a path β from the initial node v_0 to some node v' in the top layer, then an arc labelled $(\lhd\beta)$ from any node v in the top layer to another node $\hat{v}$ in the top layer can be potentially added. A decision to perform a context validation at some particular nodes, as always, is made by a finite automaton, to be defined later on.

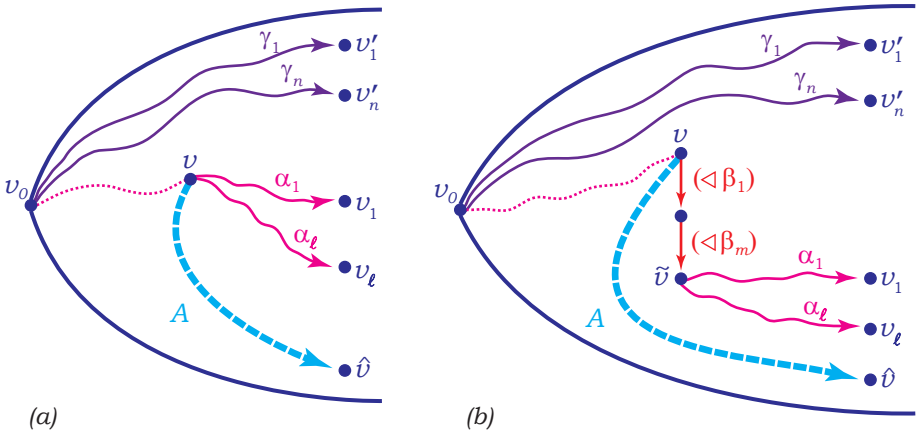


Figure 5.2: Reduction in case of grammars with contexts: *(a)* only extended contexts of the form $\lhd\gamma_i$ are present; *(b)* both kinds of contexts are present.

In the general case, a reduction is performed as follows. Consider a node v, from which the paths $\alpha_1, \ldots, \alpha_\ell$ corresponding to a rule

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \lhd\gamma_1 \,\&\, \ldots \,\&\, \lhd\gamma_n \qquad (5.1)$$

may potentially begin. Then, the paths $\alpha_1, \ldots, \alpha_\ell$ are worth being constructed only if the proper contexts $\lhd\beta_1$, …, $\lhd\beta_m$ hold true at this point. For that purpose, at the time when the layer of v is the top layer, the algorithm applies context validation for each of these contexts, creating a path $(\lhd\beta_1)\ldots(\lhd\beta_m)$ beginning in the node v. This is done for a certain fixed order of conjuncts; the exact order is unimportant. Let $\tilde{v}$ be the final node

on this path; the paths $\alpha_1, \ldots, \alpha_\ell$ begin at this point. Later on, a reduction by a rule (5.1) will be possible under the following conditions. Assume that for every conjunct α_i there is a path $(\lhd\beta_1)\ldots(\lhd\beta_m)\alpha_i$ from v to any node in the top layer, and for every conjunct $\leqslant\gamma_j$ there is a path γ_j from the initial node v_0 to some node in the top layer, as illustrated in Figure 5.2(b). Then one can add an arc labelled with A from the node v to a node $\hat{v}$ in the top layer.

5.2 Automaton guiding a parser

The set of operations on the graph performed by a parser was described in the previous section. In order to decide which operation to apply at a particular moment, the parser uses a deterministic finite automaton with a set of states Q, which reads the sequence of labels on each path of the graph. The automaton is constructed generally in the same way as Knuth's [40] LR automaton. States of this automaton essentially encode the information on the recognized bodies of conjuncts; the state reached by the automaton in a node is stored in that node as its label. The labels of the nodes in the top layer are used to decide on the actions to perform, such as whether to shift the next input and to which state, by which rule to reduce, etc.

In the case of ordinary grammars, each state of the automaton is a set of rules with a marked position in its right-hand side: these objects $A \to \mu \cdot \nu$, for a rule $A \to \mu\nu$, are called *items*. For grammars with contexts, even though the form of the rules is expanded, the items remain simple: for a rule defining a nonterminal symbol A, each conjunct $\mu\nu$ of this rule, with a position marked, gives rise to an item $A \to \mu \cdot \nu$. Conjuncts with context operators yield items with A replaced with a special symbol representing this operator, like $(\lhd\beta) \to \mu \cdot \nu$ and $(\leqslant\gamma) \to \xi \cdot \eta$, for corresponding partitions $\beta = \mu\nu$ and $\gamma = \xi\eta$. Let items(R) denote the set of all items.

The LR automaton is then defined as follows.

Definition 5.1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, and assume that there is a dedicated initial symbol $S' \in N$ with a unique rule $S' \to S$. The **LR automaton** of G is a deterministic finite automaton over the alphabet $\Sigma \cup N \cup \mathfrak{C}_\lhd$ with the set of states $Q = 2^{\mathrm{items}(R)}$. Its initial state and transitions are defined using the functions goto and closure.*

- *For every set I of items and for every symbol $X \in \Sigma \cup N \cup \mathfrak{C}_\lhd$, the function goto selects all elements of I with the dot before the symbol X, and moves the dot over that symbol. All other elements of I are discarded.*

$$goto(I, X) = \{\, V \to \alpha X \cdot \beta \mid V \to \alpha \cdot X\beta \in I,\ V \in N \cup \mathfrak{C}_\lhd \cup \mathfrak{C}_\leqslant \,\}$$

- *The set closure(I) is defined as the least set of items that contains I and, for each item $V \to \mu \cdot B\nu$ in closure(I), with $V \in N \cup \mathfrak{C}_{\lhd}$ and with a nonterminal symbol $B \in N$ after the dot, every item of the form $B \to \cdot\xi$, for each base conjunct ξ in any rule for B ($B \to \ldots \& \xi \& \ldots$), is in closure$(I)$.*

The initial state of the automaton is then

$$q_0 = closure\big(\{S' \to \cdot S\} \cup \{\,(\lhd\beta) \to \cdot\beta \mid (\lhd\beta) \in \mathfrak{C}_{\lhd}\,\} \cup \{\,(\trianglelefteq\gamma) \to \cdot\gamma \mid (\trianglelefteq\gamma) \in \mathfrak{C}_{\trianglelefteq}\,\}\big).$$

The transition from a state $q \subseteq \mathrm{items}(R)$ by a symbol $X \in \Sigma \cup N \cup \mathfrak{C}_{\lhd}$ is

$$\delta(q, X) = closure(goto(q, X)).$$

Every node of the graph is labelled by a state from Q. Whenever the graph contains an arc labelled with a symbol X from a node labelled with a state q, the label of the destination node is $\delta(q, X)$. The initial node is labelled with the state q_0, where the item $S' \to \cdot S$ initiates the processing of the whole input, whereas the items $(\lhd\beta) \to \cdot\beta$ and $(\trianglelefteq\gamma) \to \cdot\gamma$ similarly initiate the processing of contexts.

With the automaton defined, consider how each *shift operation* is performed. Let $a \in \Sigma$ be the next input symbol, and let v be a node in the current top layer, labelled with a state q. The goal is to connect v to a node in the next layer (which becomes the new top layer after the shift phase is completed). Assume that $\delta(q, a)$ is not empty, and let $\delta(q, a) = q'$. If there is a node v' in the next layer labelled with q', then the algorithm connects v to v' with an arc labelled with a. If no such node v' yet exists, it is created. In case $\delta(q, a)$ is an empty set, no new nodes are created, and the entire branch of the stack leading to v is removed. All nodes created during a shift phase form the new top layer.

In the reduction phase, the parser determines the rules to reduce by through a so-called *reduction function*. This function maps pairs of a state $q \in Q$ in a top layer node and the next k input symbols to a set of conjuncts recognized at this node. Denote by $\Sigma^{\leqslant k}$ the set of all strings of length at most k. The reduction function is then of the form $W \colon Q \times \Sigma^{\leqslant k} \to 2^{\mathrm{completed}(R)}$, where $\mathrm{completed}(R) \subset \mathrm{items}(R)$ is the set of *completed items* with a dot in the end: $V \to \alpha\cdot$, with $V \in N \cup \mathfrak{C}_{\lhd} \cup \mathfrak{C}_{\trianglelefteq}$ and $\alpha \in (\Sigma \cup N \cup \mathfrak{C}_{\lhd})^*$. In the graph, a state containing a completed item marks the end-point of a path α.

In order to compute the values of the reduction function, the algorithm requires sets $\textsc{First}_k(A) \subseteq \Sigma^{\leqslant k}$ and $\textsc{Follow}_k(A) \subseteq \Sigma^{\leqslant k}$, defined for a nonterminal symbol $A \in N$. The former set gives k-symbol prefixes of all strings generated by this nonterminal, while the latter one records all potential continuations of strings generated by A.

For every string w, let $First_k(w)$ denote w truncated to its first k symbols; if w contains fewer than k symbols, then $First_k(w) = w$. For a language L, $First_k(L) = \{ First_k(w) \mid w \in L \}$.

Then, for a nonterminal symbol A, the set $\textsc{First}_k(A)$ is defined as $\textsc{First}_k(A) = \{ First_k(v) \mid u\langle v \rangle \in L_G(A) \}$. Such a set cannot be effectively constructed already for conjunctive grammars, because their emptiness problem is undecidable. Therefore, the parsing algorithm shall use a suitable superset $\textsc{PFirst}_k(A) \subseteq \Sigma^{\leqslant k}$ constructed by Algorithm 5.1.

Algorithm 5.1. Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts. Let $k > 0$. For $\alpha = X_1 \dots X_\ell$, with $X_i \in \Sigma \cup N$, define $\textsc{PFirst}_k(\alpha) = First_k(\textsc{PFirst}_k(X_1) \cdot \ldots \cdot \textsc{PFirst}_k(X_\ell))$.
For all symbols $X \in \Sigma \cup N$, compute the set $\textsc{PFirst}_k(X)$ as follows.

let $\textsc{PFirst}_k(a) = \{a\}$, for all $a \in \Sigma$
let $\textsc{PFirst}_k(A) = \varnothing$, for all $A \in N$
for all nonterminals $A \in N$ **do**
  while new strings can be added to $\textsc{PFirst}_k(A)$ **do**
    for all $A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_\ell \;\&\; \lhd\beta_1 \,\&\, \dots \,\&\, \lhd\beta_m \;\&\; \trianglelefteqslant\gamma_1 \,\&\, \dots \,\&\, \trianglelefteqslant\gamma_n \in R$ **do**
      $\textsc{PFirst}_k(A) = \textsc{PFirst}_k(A) \cup \bigcup_{i=1}^{\ell} \textsc{PFirst}_k(\alpha_i)$

Lemma 5.1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts. Let $k > 0$. For all symbols $X \in \Sigma \cup N$ it holds that $u\langle v \rangle \in L_G(X)$ implies $First_k(v) \in \textsc{PFirst}_k(X)$.*

Now the definition of the set $\textsc{Follow}_k(A)$ can be given.

Definition 5.2. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts. A string $x \in \Sigma^*$ is said to* follow *a nonterminal symbol $A \in N$, if there exist strings $u, v \in \Sigma^*$, such that $uvx \in L(G)$ and there is a deduction of the proposition $S\big(\varepsilon\langle uvx \rangle\big)$, that essentially uses an intermediate proposition $A\big(u\langle v \rangle\big)$. Define*

$$\textsc{Follow}_k(A) = First_k(\{ x \mid x \text{ follows } A \}).$$

Supersets of these sets, called $\textsc{PFollow}_k(A)$, are determined by Algorithm 5.2, and the correctness of this algorithm is claimed in the following lemma.

Lemma 5.2. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts and let $k > 0$. Then for all symbols $A \in N$ and for all strings $x \in \Sigma^*$, if x follows A, then $First_k(x)$ is in the set $\textsc{PFollow}_k(A)$.*

Algorithm 5.2. Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts. Let $k > 0$. For a nonterminal symbol $A \in N$, compute the set $\textsc{PFollow}_k(A)$ as follows.

let $\textsc{PFollow}_k(S) = \{\varepsilon\}$
let $\textsc{PFollow}_k(A) = \varnothing$, for all $A \in N \setminus \{S\}$
while new strings can be added to $\textsc{PFollow}_k(A)$ **do**
  for all $B \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, {\lhd}\beta_1 \,\&\, \ldots \,\&\, {\lhd}\beta_m \,\&\, {\lhd\!\!\!=}\gamma_1 \,\&\, \ldots \,\&\, {\lhd\!\!\!=}\gamma_n \in R$ **do**
    for all conjuncts α_i **do**
      for all factorizations $\alpha_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup$
          $First_k(\textsc{PFirst}_k(\nu) \cdot \textsc{PFollow}_k(B))$
    for all conjuncts ${\lhd}\beta_i$ **do**
      for all factorizations $\beta_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup$
          $First_k(\textsc{PFirst}_k(\nu B) \cdot \textsc{PFollow}_k(B))$
    for all conjuncts ${\lhd\!\!\!=}\gamma_i$ **do**
      for all factorizations $\gamma_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup$
          $First_k(\textsc{PFirst}_k(\nu) \cdot \textsc{PFollow}_k(B))$

The set of possible reductions in a state q with a lookahead string $u \in \Sigma^{\leqslant k}$ is defined as follows, for $q \in Q$ and $u \in \Sigma^{\leqslant k}$.

$$W(q, u) = \{\, A \to \alpha\cdot \mid A \to \alpha\cdot \in q,\ u \in \textsc{PFollow}_k(A) \,\}$$

Context validation is similarly determined by a function $f\colon Q \to 2^{Q \times \mathfrak{C}_\lhd}$. For a state $q' \in Q$, $f(q')$ is the set of all possible pairs of the form $(q, (\lhd\beta))$, with $q \in Q$ and $(\lhd\beta) \in \mathfrak{C}_\lhd$, such that for the proper left context $\lhd\beta$, the state q' contains the completed item $(\lhd\beta) \to \beta\cdot$ and $(q, (\lhd\beta))$ is a valid entry of δ.

The algorithm uses the functions W and f to decide when to make a reduction or a context validation. Let the nodes of the graph be pairs $v = (q, p)$, where q is the state in this node and p is the number of the layer. A *reduction* by a rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, {\lhd}\beta_1 \,\&\, \ldots \,\&\, {\lhd}\beta_m \,\&\, {\lhd\!\!\!=}\gamma_1 \,\&\, \ldots \,\&\, {\lhd\!\!\!=}\gamma_n$ is done as follows. Let $\hat{p}$ be the number of the top layer, and let $v_1 = (q_1, \hat{p})$, $\ldots$, $v_\ell = (q_\ell, \hat{p})$, $v'_1 = (q'_1, \hat{p})$, $\ldots$, $v'_n = (q'_n, \hat{p})$ be nodes in the top layer. Assume that each item $A \to \alpha_i\cdot$, with $i \in \{1, \ldots, \ell\}$, is in the corresponding table entry $W(q_i, u)$, for a lookahead string $u \in \Sigma^{\leqslant k}$. Let $v = (q, p)$ be a node in any layer, from which there is a path $(\lhd\beta_1) \ldots (\lhd\beta_m)\alpha_i$ to each node v_i. Also assume that $q'_i = \delta(q_0, \gamma_i)$, for all $i \in \{1, \ldots, n\}$. Then an arc labelled with A from v to $\hat{v} = (\delta(q, A), \hat{p})$ is added, as illustrated in Figure 5.2(b).

Using the function f, a conjunct $\lhd\beta$ is validated as follows. Let $v = (q, \hat{p})$ and $v' = (q', \hat{p})$ be nodes in the top layer. Assume that there is a path β from the initial node v_0 to v', and let $(q, (\lhd\beta)) \in f(q')$. Then, the context $\lhd\beta$ can be validated by adding an arc labelled $(\lhd\beta)$ from v to another node $\hat{v} = (\delta(q, (\lhd\beta)), \hat{p})$.

The following example demonstrates how an LR-automaton is constructed and how the parsing is performed.

Example 5.1. Consider the following grammar defining the singleton language $\{ab\}$ and let us construct the LR automaton for it, as shown in Figure 5.3(a).

$$\begin{aligned} S &\to aB \\ B &\to b \,\&\, \lhd aE \\ E &\to \varepsilon \end{aligned}$$

By definition, the initial state q_0 of the automaton contains the item $S' \to \cdot S$. The set $closure(\{S' \to \cdot S\})$ contains the item $S \to \cdot aB$, which is in the state q_0 as well.

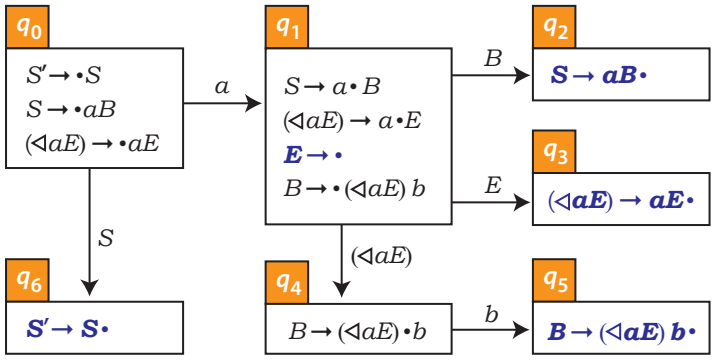


Figure 5.3: An LR automaton for the grammar in Example 5.1.

The initial state contains the item $(\lhd aE) \to \cdot aE$, corresponding to the proper left context $\lhd aE$. Note, that $\mathfrak{C}_\lhd = \{(\lhd aE)\}$.

From state q_0, there is a possible transition by symbol $a \in \Sigma$, that leads to another state q_1 containing the items $S \to a \cdot B$ and $(\lhd aE) \to a \cdot E$ with the dot propagated over a. This state q_1 also contains the item $B \to \cdot(\lhd aE)b$, which is in the set $closure(\{S \to a \cdot B\})$ and this item corresponds to the rule $B \to b \,\&\, \lhd aE$. Another item contained in the state q_1 is $E \to \cdot$, which belongs to $closure(\{(\lhd aE) \to a \cdot E\})$.

The transition by the special symbol $(\lhd aE) \in \mathfrak{C}_{\lhd}$ from state q_1 leads to state q_4, where the dot in the item $B \to (\lhd aE) \cdot b$ is propagated over this special symbol. Further transition by $b \in \Sigma$ leads to state q_5 containing the item $B \to (\lhd aE)b\cdot$.

The transition by $B \in N$ from state q_1 goes to state q_2 containing the item $S \to aB\cdot$. Transition by $E \in N$ from state q_1 leads to state q_3, which encodes the fact that the proper left context $\lhd D$ is satisfied.

The transition by $S \in N$ from state q_0 leads to the final state q_6.

Having constructed an LR automaton for the grammar, one can proceed with the parsing table.

q	δ						W		
	a	b	S	B	E	$(\lhd aE)$	ε	a	b
0	1		6						
1				2	3	4	ε		
2							aB		
3									
4		5							
5							b		
6							S		

The reduction function W gives possible reductions, when the lookahead is ε. Thus, for instance, in state q_2, a reduction can be made by the rule $S \to aB$.

The value of function f for the state 3, containing a recognized proper left context $\lhd aE$, is defined as $\{(1, (\lhd aE))\}$.

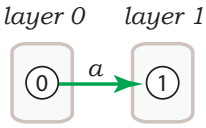
Let us consider the computation of the algorithm on the given grammar and the input string $w = ab \in L(G)$.

Reduction phase. Nothing to reduce.

Shift phase. The current configuration of graph-structured stack is shown below.

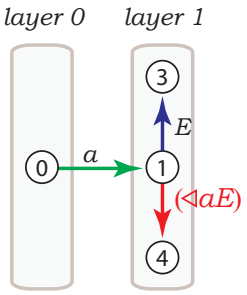
- Perform a transition by symbol a from *state 0* of layer 0 to *state 1* of layer 1.

Reduction phase. Now the current configuration of the stack is as follows.



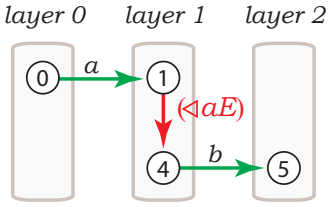
- Make a reduction by $E \to \varepsilon$ from *state 1* of layer 1 to *state 3* of layer 1.
- Validate the context $\lhd aE$: transition from *state 1* of layer 1 to *state 4* of layer 1.

Shift phase. The graph-structured stack has now the following structure.



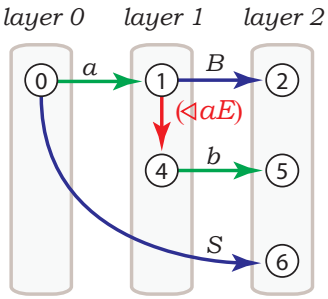
- Make a transition by symbol b from *state 4* of layer 1 to *state 5* of layer 2.

Reduction phase. The current configuration is shown below.



- Reduce by the rule $B \to b \,\&\, \lhd aE$ from *state 1* of layer 1 to *state 2* of layer 2.
- Make a reduction by the rule $S \to aB$ from *state 0* of layer 0 to *state 6* of layer 2.

Now the stack has the following structure.



5.3 Proof of correctness

For the sake of simplicity, the two following assumptions are made. First, the algorithm never removes any arcs (indeed, paths that do not lead to the top layer, do not affect the computation). Second, all special symbols $(\lhd\beta)$ shall be integrated into the grammar as extra nonterminal symbols: let $\widehat{N} = N \cup \mathfrak{C}_{\lhd}$. Consider a conjunct $\lhd\beta$ in some rule of the grammar. Then it is replaced with a concatenation $(\lhd\beta)\Sigma^*$, where the new nonterminal symbol $(\lhd\beta)$ has the unique rule $(\lhd\beta) \to \varepsilon \,\&\, \lhd\beta$.

An arc from a node $v = (q, i)$ to a node $v' = (q', j)$ labelled with a symbol $X \in \Sigma \cup N$ is called *correct*, if $q' = \delta(q, X)$, the string $a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle$ is in $L_G(X)$, and, if $X \in N$ is a nonterminal, then $\textsc{PFirst}_k(a_{j+1} \dots a_n) \in \textsc{PFollow}_k(X)$. An arc from $v = (q, i)$ to $v' = (q', j)$ labelled with a special symbol $(\lhd\beta)$ is correct, if $q' = \delta(q, (\lhd\beta))$, $i = j$ and $\varepsilon\langle a_1 \dots a_i \rangle \in L_G(\beta)$.

Lemma 5.3. *After the reduction phase in layer j, the graph contains all correct arcs to all layers up to j.*

Therefore, in the end of its computation, the algorithm constructs all correct arcs to the last ($|w|$-th) layer. Thus, it can verify whether the string is in the language by observing the arcs from the initial node to the top layer.

Proof. Consider any correct arc leading to the j-th layer, let $X \in \Sigma \cup \widehat{N}$ be its label, and let it proceed from a node $v = (q, i)$ to a node $v' = (\delta(q, X), j)$, where $i \leqslant j$ and $\delta(q, X) \neq \varnothing$. The proof that this arc is added to the graph by the parser proceeds by induction on a certain measure, called the *rank of an arc*, which will now be defined.

Consider the shortest path connecting the root node to v, continued with the arc (v, v'), and the *longest suffix of this path*, such that the heads of all arcs are contained in the j-th layer, as illustrated in Figure 5.4. Let d be the number of arcs in this path, and let $Y_1, \dots, Y_d$ be their labels; the last of them is the currently considered arc from v to v', with $Y_d = X$. Each of these arcs corresponds to a certain substring of the input (for all symbols except maybe Y_1 this substring shall be empty). For the t-th arc, let $K_t \geqslant 1$

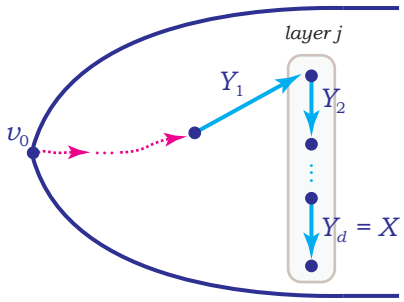


Figure 5.4: An arc of rank d (labelled with X).

be the number of deduction steps needed to generate that substring from Y_t; if Y_t is an alphabet symbol ($Y_t \in \Sigma$), let $K_t = 0$. Then the rank of the arc from v to v' is the sum $\sum_{t=1}^{d} K_t$.

The basis of induction is rank 0, when $X = a \in \Sigma$ is an alphabet symbol, the arc in question originates from the previous layer ($i = j - 1$), and $d = 1$. In this case, the arc should have been added during the shift phase.

Assume that the arc (v, v') has non-zero rank, and is hence labelled with a symbol $A \in \widehat{N}$. Also assume that all correct arcs of a lower rank are added by the algorithm. Since the arc is correct, A generates the substring $a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle$ and $a_{j+1} \dots a_z \in \textsc{Pfollow}_k(A)$. Consider the shortest deduction of the proposition $A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle\big)$. Let

$$A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_\ell \,\&\, \lhd\beta_1 \,\&\, \dots \,\&\, \lhd\beta_m \,\&\, \lhdeq\gamma_1 \,\&\, \dots \,\&\, \lhdeq\gamma_{m'} \qquad (5.2)$$

be the rule used at the last step of this deduction.

For any **base conjunct** α_h in this rule, with $h \in \{1, \dots, \ell\}$, let $\alpha_h = X_1 \dots X_r$, where $X_1, \dots, X_r \in \Sigma \cup N$. Then there is a factorization $a_{i+1} \dots a_j = u_1 \dots u_r$, with $u_1, \dots, u_r \in \Sigma^*$. For each $s \in \{0, \dots, r\}$, let $p_s = |a_1 \dots a_i u_1 \dots u_s|$; in particular, $p_0 = i$ and $p_r = j$. The premises in the deduction of the proposition $A\big(a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle\big)$, corresponding to the conjunct α_h, are $X_s\big(a_1 \dots a_{p_{s-1}} \langle a_{p_{s-1}+1} \dots a_{p_s} \rangle\big)$, for all $s \in \{1, \dots, r\}$.

It is claimed that the algorithm will construct a path $X_1 \dots X_r$ beginning in the node v and containing the following nodes.

$$\begin{aligned} v = v'_0 &= (q,\, p_0), \\ v'_1 &= (\delta(q, X_1),\, p_1), \\ v'_2 &= (\delta(q, X_1 X_2),\, p_2), \\ &\vdots \\ v'_r &= (\delta(q, X_1 X_2 \dots X_r),\, p_r) \end{aligned}$$

It has to be shown that each arc along this path is correct. The first claim is that each transition $\delta(q, X_1 \dots X_s)$, with $s \in \{1, \dots, r\}$, is possible.

Since $\delta(q, A) \neq \varnothing$, the set $goto(q, A)$ is non-empty, and therefore there is an item of the form $B \to \alpha \cdot A\beta$ (with $B \in N$, $\alpha, \beta \in (\Sigma \cup N)^*$) in the set q. Then, the item $A \to \cdot X_1 \dots X_r$ belongs to $closure(q) = q$, and accordingly the item $A \to X_1 \dots X_s \cdot X_{s+1} \dots X_r$ is in $\delta(q, X_1 \dots X_s)$.

For each $s \in \{1, \dots, r\}$, the string $a_1 \dots a_{p_s} \langle a_{p_s+1} \dots a_j \rangle$ is in the language $L_G(X_{s+1} \dots X_r)$. At the same time, $First_k(a_{j+1} \dots a_z)$ is in the set $\textsc{PFollow}_k(A)$ by assumption. Therefore, Algorithm 5.2 eventually adds the string

$$First_k(a_{p_s+1} \dots a_z)$$

to the set $\textsc{PFollow}_k(X_s)$.

Therefore, for each s, the arc from $v'_{s-1} = (\delta(q, X_1 \dots X_{s-1}),\ p_{s-1})$ to $v'_s = (\delta(q, X_1 \dots X_s),\ p_s)$ labelled with X_s is correct. If that arc goes to an earlier layer than the j-th one, then it was added at one of the previous shift or reduction phases. If it goes to the j-th layer, then its rank is less than that of the arc from v to v', and therefore, it is added by the algorithm by the induction hypothesis.

Consider any **extended context operator** $\lessdot\gamma_h$ used in the rule (5.2). Let $\gamma_h = X_1 \dots X_r$, where $X_1, \dots, X_r \in \Sigma \cup N$. Then again there is a factorization $a_1 \dots a_j = u_1 \dots u_r$, with $u_1, \dots, u_r \in \Sigma^*$. For each $s \in \{0, 1, \dots, r\}$, let $p_s = |u_1 \dots u_s|$. In the deduction by the rule (5.2), the premises corresponding to the conjunct $\lessdot\gamma_h$ are of the form $X_s\big(\varepsilon\langle a_1 \dots a_{p_s}\rangle\big)$, for all $s \in \{1, \dots, r\}$.

The algorithm shall construct a path $X_1 \dots X_r$ from the initial node v_0 of the graph. This path contains the following nodes.

$$\begin{aligned}
v_0 &= (q_0, 0) \\
v_1 &= (\delta(q_0, X_1), p_1) \\
v_2 &= (\delta(q_0, X_1X_2), p_2) \\
&\vdots \\
v_r &= (\delta(q_0, X_1X_2 \dots X_r), p_r)
\end{aligned}$$

In order to show that every arc in this path is correct, one has to show that, similarly to the case of base conjuncts, each transition $\delta(q_0, X_1 \dots X_s)$ is valid for all $s \in \{1, \dots, r\}$. By construction of the LR automaton, the item $(\lessdot\gamma_i) \to \cdot X_1 \dots X_r$ is contained in the initial state q_0. Thus, items $(\lessdot\gamma_i) \to X_1 \dots X_s \cdot X_{s+1} \dots X_r$, for all $s \in \{1, \dots, r\}$, are contained in the states $\delta(q_0, X_1 \dots X_s)$.

It remains to show that the algorithm will do a reduction by the rule (5.2) for A. For every base conjunct $\alpha_h = X_1 \dots X_r$, there is a node $v'_r = (\delta(q, X_1X_2 \dots X_r),\ j)$ in the top layer. The state in this node contains the item $A \to X_1X_2 \dots X_r\cdot$. Therefore, $A \to X_1X_2 \dots X_r\cdot$ is in $W(q, \textsc{First}_k(a_{j+1} \dots a_z))$.

Similarly, for every extended left context $\lhd\gamma_h$ with $\gamma_h = X_1 \dots X_r$, the top layer contains a node $(\delta(q_0, X_1 X_2 \dots X_r), j)$, and the item $A \to X_1 X_2 \dots X_r \cdot$ is contained in the state of this node.

Thus, all the conditions for the reduction are satisfied, and the algorithm eventually creates the desired arc labelled A from the node $v = (q, i)$ in layer i to another node $v' = (q', j)$ in layer j.

□

5.4 Implementation and complexity

The Generalized LR parsing algorithm for grammars with contexts operates on an input string $w = a_1 \dots a_n$, with $n \geqslant 0$ and $a_i \in \Sigma$. At the beginning of the computation, the graph-structured stack contains a single initial node $v_0 = (q_0, 0)$, which forms the top layer of the graph. The algorithm alternates between reduction phases and shift phases. It begins with a reduction phase in layer 0, using the first k symbols of the input as a lookahead string.

For each input symbol a_i, the algorithm first performs a shift phase for a_i. If after that the top layer is empty, the input string is rejected. Otherwise, the algorithm proceeds with a reduction phase using the next k input symbols $a_{i+1} \dots a_{i+k}$ as a lookahead. In a reduction phase, the algorithm performs context validations and reductions while any further arcs can be added.

All the nodes that cannot be reached from the initial node are removed. Finally, when all the input symbols are consumed, the algorithm checks whether there is an arc labelled with S from the initial node v_0 to a node $v = (\delta(q_0, S), n)$ in the top layer. If such an arc exists, the input string is accepted; otherwise it is rejected.

Input: a string $w = a_1 \dots a_n$, with $a_1, \dots, a_n \in \Sigma$.
Let a node v_0 labelled with q_0 form the top layer of the graph.

```
do Reduction Phase using lookahead u = First_k(w)
for i = 1 to n do
    do Shift Phase using a_i
    if the top layer is empty then
        Reject
    do Reduction Phase using lookahead u = First_k(a_{i+1} ... a_n)
    remove the nodes unreachable from the source node v_0
if there is an arc S from v_0 to δ(q_0, S) in the top layer then
    Accept
else
    Reject
```

Each node $v = (q, p)$ in the graph shall be represented as a data structure that holds the number of the state q and a list of pointers to all *predecessor nodes*, one for every node v' connected to v.

The graph has $\mathcal{O}(n)$ nodes, where n is the length of the input string to be parsed. A node in the p-th layer may have incoming arcs from any nodes in the layers $0, \ldots, p$; hence, the size of the graph is at most quadratic in n and the algorithm uses at most $\mathcal{O}(n^2)$ space.

A single *shift phase* only considers the nodes in the current top layer, and the number of those nodes is bounded by $|Q|$, the number of the states of the LR automaton. Therefore, the complexity of performing a shift phase does not depend on the size of the graph-structured stack or the length of the input string, which means that the shift phase has constant time complexity.

During a *reduction phase*, the algorithm first performs context validations in the way described in Section 5.2. During these operations, the parser only considers nodes in the top layer, and since their number is bounded by the constant $|Q|$, context validation takes constant time.

Before performing reductions, the algorithm first searches the graph-structured stack for any nodes that satisfy the conditions of applying a reduction operation by a certain rule. This is done by a search procedure called *conjunct gathering*, originally introduced for conjunctive grammars [52, 60] and here applied to grammars with contexts.

Let $T[]$ be an array of sets of nodes, indexed by completed LR items.
Conjunct Gathering*:*

for each node $v = (q, p)$ of the top layer **do**
  for each $A \to \alpha\cdot \in R(q, u)$ **do**
    $T[A \to (\lhd\beta_1) \ldots (\lhd\beta_m)\alpha\cdot] \cup = pred_{|(\lhd\beta_1 \ldots (\lhd\beta_m)\alpha|}(\{v\})$

For a node v and a number $i \geqslant 0$, let $pred_i(v)$ denote the set of all nodes connected to v by a path that has exactly i arcs. This set can be computed iteratively on i by letting $pred_0(v) = \{v\}$, and then constructing consequent sets $pred_{i+1}(v)$ as the sets of all such nodes v', for which there is an arc to some $v'' \in pred_i(v)$. This involves processing $\mathcal{O}(n)$ nodes of the graph, each of which has at most $\mathcal{O}(n)$ predecessors. This implementation of conjunct gathering requires $\mathcal{O}(n^2)$ operations to compute.

There is a way of doing conjunct gathering in linear time, using the method of Kipps [39] for ordinary grammars. For each node v, the algorithm shall maintain data structures for the sets $pred_i(v)$ for all applicable numbers i. Every time a new arc from a node v to a node v' is added to the graph, the predecessors of v are inherited by v' and by all successors of v'. Then, instead of computing the set $pred_i(v)$ every time, its value can be looked up in the memory. This enables conjunct gathering in time $\mathcal{O}(n)$.

With the data on conjuncts gathered, the algorithm proceeds with performing reductions.

REDUCTIONS:

for each rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \lhd\!\!\!|\,\gamma_1 \,\&\, \ldots \,\&\, \lhd\!\!\!|\,\gamma_{m'}$ **do**

  if there are nodes $v_j = (\delta(q_0, \gamma_j), p)$ in the top layer for all $j \in \{1, \ldots, m'\}$ **then**

    for each node $v = (q, p) \in \bigcap_{h=1}^{\ell} T[A, |(\lhd\beta_1) \ldots (\lhd\beta_m)\alpha_h|]$ **do**

      if v is not connected to the top layer by A **then**

        transition from v to $\hat{v} = (\delta(q, A), p)$ by A

Each individual *reduction operation* is concerned with a rule of the form $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \lhd\!\!\!|\,\gamma_1 \,\&\, \ldots \,\&\, \lhd\!\!\!|\,\gamma_{m'}$ in R. First, the algorithm considers extended left context operators $\lhd\!\!\!|\,\gamma_j$ of this rule and checks whether for an operator $\lhd\!\!\!|\,\gamma_j$ there is a node with a state $\delta(q_0, \gamma_j)$ in the top layer of the graph-structured stack. If this is true for all $j \in \{1, \ldots, m'\}$, the algorithm proceeds with checking the nodes corresponding to the gathered conjuncts. If all the conditions for a reduction are met, then the parser performs a reduction operation.

All in all, if conjunct gathering is implemented in the simple way, then the complexity of the reduction phase is cubic, which means that the complexity of the whole algorithm is bounded by $\mathcal{O}(n^4)$. If Kipps' method is used, then conjunct gathering only takes linear time, the reduction phase is done in time $\mathcal{O}(n^2)$, and the complexity of the whole algorithm is cubic. A downside of Kipps' approach is that the new data structures take time to update, and the addition of every arc can no longer be done in constant time. But, as there will be at most $C \cdot n^2$ arc additions, the time spent maintaining the stored values of $pred_i(v)$ sums up to $\mathcal{O}(n^3)$ for the entire algorithm.

Thus, the deterministic bottom-up parsing algorithm has been extended for the case of grammars with one-sided contexts, and this algorithm becomes the first sign of possible practical implementation of these grammars.

Chapter 6

Recursive descent parsing

Recursive descent parsing is probably the most well-known and intuitive technique applicable to a subclass of context-free grammars. A subroutine for each nonterminal should determine a rule according to which the substring shall be parsed. If such a rule can be determined using the k next symbols of the input, then this is a standard LL(k) parser [2, 19, 41, 43, 46, 84, 94].

If several suitable alternatives exist, the issue of choosing one of them arises. Some software implementations of recursive descent parser generators, such as *ANTLR* by Parr and Fisher [72, 71], allow using *ad hoc* methods to make the parsing deterministic when it is inherently not. For instance, a unique rule according to which the string shall be parsed, may be chosen by *scanning right context* of a string without consuming the input. Another example of a model with deterministic choice of rule is LL-regular grammars [29], where the form of a right context can be specified by a regular language.

Unlike deterministic predictive parsing, *limited backtracking parsing* uses prioritized choice of alternatives going over the set of suitable rules. The parsing is successful when a string is parsed according to at least one of them. Birman and Ullman introduced the recognition schemata [7] of top-down parsing languages ("TS/TDPL") and generalized top-down parsing languages ("gTS/GTDPL"), which are capable of specifying recognizers for deterministic context-free languages with restricted backtracking. One of the software implementations of this approach is *parsing expression grammars* [18], which additionally has a means to express Boolean operations over the right contexts of a string. The effective recognition power of these grammars has been proved [18] equivalent to the one of the mentioned schemata.

In this chapter, the recursive descent parsing is extended to handle the case of *grammars with right contexts*, additionally allowing a limited backtracking. The conjunction operation is implemented by scanning a single substring multiple times, as in the case of conjunctive grammars [61]. Possi-

ble rules are tested one by one in a given order; the applicability of the rule is determined by checking the context specifications. Thus, right contexts are used to choose the alternatives suitable during the parsing.

Applicability of the new algorithm is restricted to a subset of grammars with contexts which satisfy the *prefix property* and the property of *k-quasi-separability* [82, 94]. This allows one to prove the correctness of the algorithm with limited backtracking even in such undesirable situations as the one occurring in the grammar [41] with the rules $S \to Ac$ and $A \to a \mid ab$. Given such a grammar and an input string abc, the parser would fail when scanning symbol b, since after having consumed the prefix a and returned according to the first alternative for A, it would expect the symbol c.

A direct implementation of the parsing algorithm developed here may use exponential time on some extreme grammars. However, using the *memoization* technique guarantees linear time complexity of the parser.

6.1 Non-left-recursive grammars

The recursive descent parsing method for standard context-free grammars, as well as its generalizations for conjunctive and Boolean grammars [61], require the grammar to be free of *left recursion*, which means that no nonterminal A can derive $A\xi$ for any $\xi \in (\Sigma \cup N)^+$. The reason for that is that a parser can enter an infinite loop otherwise.

The notion of left recursion can be extended to grammars with right contexts similarly to the case of conjunctive grammars [61], using the *relation of context-free reachability in one step*, which is an adaption of context-free derivation [2] to specify dependence of nonterminals in a grammar.

Definition 6.1 ([61])**.** *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Define the relation of context-free reachability in one step, $\rightsquigarrow$, a binary relation on the set of strings with a marked substring $\{\, \alpha[\beta]\gamma \mid \alpha, \beta, \gamma \in (\Sigma \cup N)^* \,\}$ as: $\alpha[\beta A\gamma]\mu \rightsquigarrow \alpha\beta\xi[\sigma]\eta\gamma\mu$, for all $\alpha, \beta, \gamma, \mu, \xi, \sigma, \eta \in (\Sigma \cup N)^*$, $A \in N$, and for all base conjuncts $\xi\sigma\eta$ in the rules for the nonterminal A.*

Using this relation, the notion of a non-left-recursive grammar with right contexts can be now given.

Definition 6.2. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Then it is said to be* non-left-recursive *if and only if for all $A \in N$ and $\alpha, \beta \in (\Sigma \cup N)^*$, such that $\varepsilon[A]\varepsilon \rightsquigarrow^+ \alpha[A]\beta$, it holds that $\langle\varepsilon\rangle y \notin L_G(\alpha)$, for all $y \in \Sigma^*$.*

6.2 Outline of the algorithm

The recursive descent parser enables to perform the analysis of an input string, using the collection of recursive subroutines constructed from the form of the rules of the grammar. For each symbol of the grammar, the recursive descent parser contains a subroutine, which, having access to the input string and the current position of the parser, tries to recognize this symbol and advance the current position, until the whole input string is consumed.

In the case of LL(k) context-free grammars [43, 46, 48, 78], for which recursive descent parsing is possible, the code of each procedure $a()$, with $a \in \Sigma$, checks whether the next symbol in the input string is a, and, if so, advances the current position of the parser. Otherwise, an error is reported.

Listing 6.1. Code for a terminal symbol $a \in \Sigma$:

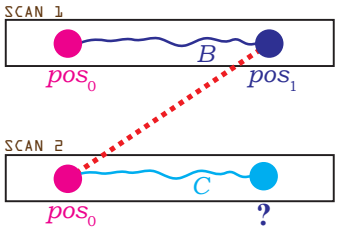
$a()$ {
    if $w_p = a$ **then**
        $p := p + 1$;
    else
        report error;
}

Every nonterminal symbol $A \in N$, in its turn, has a corresponding recursive subroutine, which tries to parse the substring according to some rule for A. The rule is chosen from the look-up table using some local information, that is, the next k symbols of the string. Once a rule $A \to X_1 \dots X_n$ is chosen, the code $X_1(); \dots; X_n()$ parses the following substring according to this rule, that is, by invoking the corresponding procedures for every symbol X_i, $i \in \{1, \dots, n\}$.

Listing 6.2. For every nonterminal $A \in N$, the code is as follows:

$A()$ {
    select a rule r using k look-ahead symbols
        case $r = A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_\ell \,\&\, \trianglerighteq\beta_1 \,\&\, \dots \,\&\, \trianglerighteq\beta_m \,\&\, \triangleright\gamma_1 \,\&\, \dots \,\&\, \triangleright\gamma_n$
            //code for the conjunct α_1
            $\vdots$
            //code for the conjunct α_ℓ
            //code for the conjunct β_1
            $\vdots$

```
            //code for the conjunct β_m
            //code for the conjunct γ_1
            ⋮
            //code for the conjunct γ_n
        case A → ...:
            ⋮
        else
            report error;
    }
```



In the extension of recursive descent parsing to *conjunctive grammars* [61], the conjunction operation is implemented by scanning a single substring multiple times. If the parser chooses a rule, say, $A \to B \,\&\, C$, then it first stores its current (initial) position in a local variable, and then invokes the procedure $B()$. So far, the substring is parsed according to the first conjunct. Then the parser again stores its (final) position in another local variable, thus remembering the position it gained after the first parse of the substring. Next, the current position of the parser is rewound back to the stored initial position and parsing of the substring according to the second conjunct is performed by invoking the procedure $C()$. Once it returns, the parser checks whether its current position is identical to what it had after the first parse of the string. If the two positions are identical, the procedure $A()$ returns; if not, an error is reported.

Listing 6.3. Code for the *first* base conjunct $\alpha = X_1 \dots X_\ell$, with $X_i \in \Sigma \cup N$:

```
start :=p;
X_1();
⋮
X_ℓ();
end :=p;
```

Listing 6.4. Code for every *consequent* base conjunct of a rule:

```
p :=start;
X_1();
```

$\vdots$

$X_\ell()$;

if $p \neq end$ **then report error**;

The generalization of the recursive descent parser method to the case of *grammars with right contexts*, additionally to implementing the conjunction operation, shall check that proper and extended right contexts are satisfied.

If the parser chooses a rule, say, $A \to B \,\&\, C \,\&\, \rhd F \,\&\, \trianglerighteq H$, then initially it behaves exactly the same as the parser for conjunctive grammars. After the two successful parses of the substring (according to B and C), the parser stores its current position in a local variable and invokes the procedure $F()$. If the position after the procedure F returns is identical to the length of the string, the context $\rhd F$ is deemed satisfied and the parser continues with the context $\trianglerighteq H$. If not, an error is reported.

Listing 6.5. Code for a conjunct $\rhd\gamma = X_1 \dots X_\ell$, with $X_i \in \Sigma \cup N$:

p :=*end*;

$X_1()$;

$\vdots$

$X_\ell()$;

if $p \neq |w|$ **then report error**;

When checking the context $\trianglerighteq H$, the parser rewinds its position to the beginning of the substring and invokes the procedure $H()$. Similarly to the case of the context of the form $\rhd F$, either an error is reported or the parser continues its work.

Listing 6.6. Code for a conjunct $\trianglerighteq\beta$, with $\beta = X_1 \dots X_\ell$, $X_i \in \Sigma \cup N$:

p :=*start*;

$X_1()$;

$\vdots$

$X_\ell()$;

if $p \neq |w|$ **then report error**;

Since no other contexts are left to check, the parser reports a successful parsing of the substring. In the general case, after the last conjunct of a rule has been successfully satisfied, the procedure for the nonterminal A returns.

The procedure $S()$ is called in the main procedure of the parser.

Listing 6.7. The main procedure of a recursive descent parser:

integer p :=1;

$S()$;

if $p \neq |w|$ **then** REJECT

ACCEPT

This is an extension of the recursive descent parsing technique (in the form as developed for conjunctive grammars [61]) for the grammars with right contexts. In the version of the algorithm presented in this section, operators, referring to the right contexts of strings, are not taken into account when a choice of rule is made. In what follows, the algorithm is further extended to handle the case of grammars with *ordered* rules and *ordered* conjuncts within rules. This shall allow one to verify first the extended context operators, which represent a form of a lookahead. If some $\triangleright$-conjunct of a rule is not satisfied, then the whole rule is considered mismatching, the actual parsing of the substring shall not be done, and another rule shall be tried.

6.3 Constructing a parsing table

The computation of a recursive descent parser is guided by a parsing table, which, for every $A \in N$ and $w \in \Sigma^*$, determines the rules to apply when A is to produce a string starting with w.

The algorithm for constructing a parsing table uses sets $\textsc{First}_k(A)$ and $\textsc{Follow}_k(A)$, similarly to the LR parsing algorithm, as described in Section 5.2. For a nonterminal symbol $A \in N$, the set $\textsc{First}_k(A)$ represents all k-symbol prefixes of the strings defined by A, and the set $\textsc{Follow}_k(A)$ gives all potential continuations of strings generated by this nonterminal.

Since the set $\textsc{First}_k(A)$ cannot be effectively computed, the algorithm shall use its superset $\textsc{PFirst}_k(A)$ constructed by Algorithm 6.1. This algorithm is a version of Algorithm 5.1 adapted for grammars with right contexts.

The correctness of the algorithm is claimed in the following lemma.

Lemma 6.1. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Then, for any symbol $X \in \Sigma \cup N$, $\textsc{First}_k(X) \subseteq \textsc{PFirst}_k(X)$.*

In the case of grammars with right contexts, the definition of the set $\textsc{Follow}_k(A)$ is not symmetric with Definition 5.2 for grammars with left contexts.

Definition 6.3. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. A string $v \in \Sigma^*$ is said to* follow *a nonterminal symbol $A \in N$, if there exist strings $x, u \in \Sigma^*$, such that $xuv \in L(G)$ and there is a deduction of*

Algorithm 6.1. Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts. Let $k > 0$. For $\alpha = X_1 \dots X_s$, with $X_i \in \Sigma \cup N$, define $\textsc{PFirst}_k(\alpha) = First_k(\textsc{PFirst}_k(X_1) \cdot \ldots \cdot \textsc{PFirst}_k(X_s))$.
For all symbols $X \in \Sigma \cup N$, compute the set $\textsc{PFirst}_k(X)$ as follows.

```
let PFirst_k(a) = {a}, for all a ∈ Σ
let PFirst_k(A) = ∅, for all A ∈ N
for all nonterminals A ∈ N do
    while new strings can be added to PFirst_k(A) do
        for all A → α_1 & ... & α_ℓ & ⩾β_1 & ... & ⩾β_m &
        ▷γ_1 & ... & ▷γ_n ∈ R do
            PFirst_k(A) = PFirst_k(A) ∪ ⋂_{i=1}^{k} PFirst_k(α_i)
```

the proposition $S(\langle xuv\rangle\varepsilon)$, that essentially uses an intermediate proposition $A(\langle u\rangle v)$. Define

$$\textsc{Follow}_k(A) = First_k(\{\, v \mid v \textit{ follows } A \,\}).$$

Again, the algorithm for constructing a parsing table shall use a suitable superset $\textsc{PFollow}_k(A)$ of this set, and this superset is computed by Algorithm 6.2 (cf. Algorithm 5.2).

The algorithm is stated correct as follows.

Lemma 6.2. *Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts and let $k \geqslant 0$. Then for all symbols $A \in N$ and for all strings $v \in \Sigma^*$, if v follows A, then $First_k(v) \in \textsc{PFollow}_k(A)$.*

Now the definition of the look-up table can be given.

Definition 6.4. *Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts. Let $k \geqslant 0$. A* look-up table *for G is a function $T_k : N \times \Sigma^{\leqslant k} \to 2^R$, such that for every rule $r \in R$ for some nonterminal $A \in N$ and $w, v \in \Sigma^*$, for which $\langle w\rangle v$ is in the language defined by the rule r and v follows A, it holds that $r \in T_k[A, \textsc{First}_k(wv)]$.*

Note, that the definition allows k to be zero. In that case, the function T_k maps the set of nonterminals to the powerset of the rules: $T_k : N \to 2^R$. This means, that whenever more than one rule is given for some nonterminal, all of them should be tried (until the first of them, matching the substring, is found) without using any lookahead symbols.

Note, that using supersets of the sets $\textsc{First}_k(A)$ and $\textsc{Follow}_k(A)$ may lead to fictional entries in the table, which, however, do not prevent the algorithm from being correct [61].

The following lemma claims correctness of Algorithm 6.3.

Algorithm 6.2. Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts. Let $k > 0$. For a nonterminal symbol $A \in N$, compute the set $\textsc{PFollow}_k(A)$ as explained below.

Define the set $\mathcal{X}$ of all such nonterminal symbols $A \in N$, that appear as the last symbol in conjuncts of the form $\trianglerighteq\beta$ or $\triangleright\gamma$: $\mathcal{X} = \{ A \in N \mid B \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \trianglerighteq\beta_1 \,\&\, \ldots \,\&\, \trianglerighteq\beta_n \,\&\, \triangleright\gamma_1 \,\&\, \ldots \,\&\, \triangleright\gamma_m \in R, \beta_i \in (\Sigma \cup N)^* A \text{ or } \gamma_i \in (\Sigma \cup N)^* A, \text{ for some applicable } i \}$.

In an analogous way, for a rule $r \in R$, define the set $\mathcal{Y}(r)$, containing nonterminals $A \in N$, that are the last symbols of base conjuncts in this rule: $\mathcal{Y}(r) = \{ A \in N \mid r = B \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \trianglerighteq\beta_1 \,\&\, \ldots \,\&\, \trianglerighteq\beta_n \,\&\, \triangleright\gamma_1 \,\&\, \ldots \,\&\, \triangleright\gamma_m \in R, \alpha_i \in (\Sigma \cup N)^* A, \text{ for some } 1 \leqslant i \leqslant \ell \}$.

let $\textsc{PFollow}_k(S) = \{\varepsilon\}$
let $\textsc{PFollow}_k(A) = \{\varepsilon\}$, for all $A \in \mathcal{X}$
let $\textsc{PFollow}_k(A) = \varnothing$, for all $A \in N \setminus (\{S\} \cup \mathcal{X})$
while new strings can be added to $\textsc{PFollow}_k(A)$ **do**
  for all rules $r = B \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \;\&\; \trianglerighteq\beta_1 \,\&\, \ldots \,\&\, \trianglerighteq\beta_n \;\&\; \triangleright\gamma_1 \,\&\, \ldots \,\&\, \triangleright\gamma_m \in R$ **do**
    if $A \in \mathcal{Y}(r)$ **then**
      $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup \bigcap_{i=1}^{m} \textsc{PFirst}_k(\gamma_i)$
    for all conjuncts α_i **do**
      for all factorizations $\alpha_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup$
          $First_k(\textsc{PFirst}_k(\nu) \cdot \textsc{PFollow}_k(B))$
    for all conjuncts $\trianglerighteq\beta_i$ **do**
      for all factorizations $\beta_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup \textsc{PFirst}_k(\nu)$
    for all conjuncts $\triangleright\gamma_i$ **do**
      for all factorizations $\gamma_i = \mu A \nu$, with $\mu, \nu \in (\Sigma \cup N)^*$ **do**
        $\textsc{PFollow}_k(A) = \textsc{PFollow}_k(A) \cup \textsc{PFirst}_k(\nu)$

Algorithm 6.3. Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts. Let $k \geqslant 0$. The elements of the $\mathrm{LL}(k)$ look-up table are computed as follows.

$T_k[A, x] = \varnothing$, for all $A \in N$ and $x \in \Sigma^{\leqslant k}$
for all rules $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_\ell \,\&\, \trianglerighteq\beta_1 \,\&\, \ldots \,\&\, \trianglerighteq\beta_n \,\&\, \triangleright\gamma_1 \,\&\, \ldots \,\&\, \triangleright\gamma_m \in R$ **do**
  for all $x \in \left(\bigcap_{i=1}^{\ell} \textsc{PFirst}_k(\alpha_i) \cdot \textsc{PFollow}_k(A)\right) \cap \bigcap_{i=1}^{n} \textsc{PFirst}_k(\beta_i)$ **do**
    $T_k[A, x] = T_k[A, x] \cup \{r\}$

Lemma 6.3. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Let $A \in N$, $x \in \Sigma^{\leqslant k}$, and $r \in R$ be some rule, which should be put in the cell $T_k[A, x]$ according to Definition 6.4. Then r is put in $T_k[A, x]$ by Algorithm 6.3.*

Let us consider an example of a grammar..

Example 6.1. Consider a language $\{ a^n c b^n \mid n \geqslant 0 \} \cup \{ a^n d b^{2n} \mid n \geqslant 0 \}$, which is not an LL($k$) language for any k [2, Ex. 5.1.20].

This language can be generated by the following grammar with right contexts.

$$\begin{aligned} S &\to \trianglerighteq AcX \,\&\, C \mid \trianglerighteq AdX \,\&\, D \\ C &\to aCb \mid c \\ D &\to aDbb \mid d \\ A &\to aA \mid \varepsilon \\ X &\to aX \mid bX \mid cX \mid dX \mid \varepsilon \end{aligned}$$

Nonterminals C and D generate languages $a^n c b^n$ (with $n \geqslant 0$) and $a^n d b^{2n}$ (with $n \geqslant 0$) in any right context, respectively. Both rules for S use a right context specification of the form $a^* x \Sigma^*$ with $x = c$ or $x = d$.

The following parsing table can be constructed for this grammar.

	ε	a	b	c	d
S	$\varnothing$	$\{S \to \trianglerighteq AcX \,\&\, C,$ $S \to \trianglerighteq AdX \,\&\, D\}$	$\varnothing$	$S \to \trianglerighteq AcX \,\&\, C$	$S \to \trianglerighteq AdX \,\&\, D$
A	$A \to \varepsilon$	$A \to aA$	$\varnothing$	$A \to \varepsilon$	$A \to \varepsilon$
X	$X \to \varepsilon$	$X \to aX$	$X \to bX$	$X \to cX$	$X \to dX$
C	$\varnothing$	$C \to aCb$	$\varnothing$	$C \to c$	$\varnothing$
D	$\varnothing$	$D \to aDbb$	$\varnothing$	$\varnothing$	$D \to d$

The table is nondeterministic, as the cell (S, a) contains two rules. Thus, both of the rules shall be tried in the given order by the parsing algorithm with limited backtrack, as described in the next section.

6.4 Parsing algorithm with limited backtrack

When the parser can not deterministically choose the rule to apply, it shall use the backtracking technique, which is fairly standard in recursive descent parsing [7, 18, 41, 76].

The following example is meant to demonstrate how extended right contexts can be used in disambiguating the syntax of programming languages.

Example 6.2. Consider the following fragment of an ordinary context-free grammar for the language of assignment statements and function calls.

$S \rightarrow$ **Assignment** | **FunctionCall**

Assignment $\rightarrow$ `ident` "`=`" **Expression** "`;`"

FunctionCall $\rightarrow$ `ident` "`(`" **ParameterList** "`)`"

Since both alternatives for the nonterminal S start with the same terminal symbol, the grammar is not LL(1) context-free. This can be obviously got around by factorizing the rules as follows.

$S \rightarrow$ `ident` **RightPart**

RightPart $\rightarrow$

"`=`" **Expression** "`;`" |

"`(`" **ParameterList** "`)`"

Now both rules for the nonterminal **RightPart** start with different symbols; however, such a factorization may not be possible, if different semantic actions are associated with the terminal `ident` in the rules for **Assigment** and **FunctionCall** in the first grammar.

Using right contexts, one can, before making the actual parsing, look-up the entire suffix of the input string, and make a decision about a rule to apply. This idea can be implemented by changing the rule for S as follows (nonterminal *anything* defines all strings in all right contexts, that is, $L(anything) = \{ \langle w \rangle v \mid w, v \in \Sigma^* \}$).

$S \rightarrow$

$\trianglerighteq$ `ident` "`=`" *anything* & **Assignment** |

$\trianglerighteq$ `ident` "`(`" *anything* & **FunctionCall**

Here the extended right contexts ($\trianglerighteq$ `ident` "`=`" *anything* and $\trianglerighteq$ `ident` "`(`" *anything*) are used to decide, based on whether the symbol after the identifier is an assigment symbol or an open bracket, how the substring shall be parsed. Note, that modifying the original rule for S as

$S \rightarrow$

Assignment & $\triangleright$ "`=`" *anything* |

FunctionCall & $\triangleright$ "`(`" *anything*

is of no sense, because in both alternatives the substring is parsed according to the base conjuncts first and only then the right contexts are verified. Moreover, placing proper right contexts ($\triangleright$ "`=`" *anything* and $\triangleright$ "`(`" *anything*)

before the base conjuncts (**Assignment** and **FunctionCall**, respectively) should not be allowed, since the right context of a substring can only be specified after the substring itself has been specified.

This is formalized in the following definition.

Definition 6.5. *A* grammar with ordered rules *is a quadruple* $G = (\Sigma, N, R, S)$*, where*

- Σ*,* N*, and* S *are the same as in Definition 2.1;*
- R *is a finite set of linearly ordered rules, each of the form*

$$A \to Q_1\alpha_1 \,\&\, \ldots \,\&\, Q_n\alpha_n, \tag{6.1}$$

with $A \in N$*,* $n \geqslant 1$*,* $\alpha_i \in (\Sigma \cup N)^*$*,* $Q_i \in \{\trianglerighteq, \square, \triangleright\}$*.*

A term $Q_i\alpha_i$ is called a *conjunct*, and, as in Definition 2.1, a conjunct $Q_i\alpha_i$ is called *base*, if $Q_i = \square$, *extended right context*, if $Q_i = \trianglerighteq$, and *proper right context*, if $Q_i = \triangleright$.

Every rule should have at least one base conjunct, that is, $\{\, i \in \{1, \ldots, n\} \mid Q_i = \square \,\} \neq \varnothing$. This restriction means that a substring itself should be always defined explicitly.

Moreover, every proper right context should be preceded by at least one base conjunct, that is, $Q_i = \triangleright$ implies that $Q_j = \square$ for some $j < i$. This reflects the fact that a right context of a string can only be specified after the string itself has been specified. The same restriction is not needed for extended right contexts, since they describe the form of both *the string* and its right context.

In a parser with limited backtracking, if several rules r_1, ..., r_n for a nonterminal symbol A are suitable, the parser first tries to parse the substring according to the first rule, r_1, having stored its (initial) position in a local variable beforehand. If no error has been reported, the substring is recognized according to the rule r_1, and the parser moves on to the next symbol to parse. Otherwise, the parser shall rewind its current position to the stored one, and try the next rule, r_2. The parser continues working in such a manner until either after some rule r_i no error is reported (then the substring is recognized according to this rule), or all possible rules for nonterminal A are exhausted. In the latter case, an error is reported. Such behaviour of the parser can be implemented with the mechanism of structured exception handling [21].

A recursive descent parser with backtracking, that uses k look-ahead symbols, is defined below for grammars with right contexts. It shall use two global variables accessible to all subroutines: the input string w and the current position p of the parser within that string (a positive integer).

Listing 6.8. Code for a nonterminal symbol $A \in N$:

```
A() {
    boolean failed :=false;
    try {
        //code for the first rule in T_k[A, w_p w_{p+1} ... w_n]
    }
    catch {
        try {
            p :=pos;
            //code for the next rule in T_k[A, w_p w_{p+1} ... w_n]
        }
        catch {
            ⋱
                try {
                    p :=pos;
                    //code for the last rule
                    //in T_k[A, w_p w_{p+1} ... w_n]
                }
                catch {
                    failed :=true;
                }
             ⋰
        }
    }
    if failed then raise;
}
```

In the recursive descent parser with backtracking, the code fragments for conjuncts in a rule are the same as in Listings 6.3–6.5. The only difference is that the code for the very first conjunct in the rule has to initialize the local variable *start* with the value p (*start* :=p).

Listing 6.9. The main procedure of a recursive descent parser:

```
try {
    integer p :=1;
    S();
    if p ≠ |w| then raise
}
catch {
    REJECT
}
ACCEPT
```

6.5 Grammars allowing parsing with backtrack

Backtracking in a recursive descent parser has to impose some restrictions on how the rules for the same nonterminal are ordered in a grammar.

Example 6.3. Consider a grammar with the rules $S \to a$ and $S \to ab$, to demonstrate that the order in which the rules appear in a grammar is dramatically important for a recursive descent parser with partial backtrack. Given the input string ab, the parser consumes the symbol a and the procedure S returns according to the rule $S \to a$. Thus, the string ab can never be recognized in such a grammar.

However, ordering the rules as $S \to ab \mid a$ solves this issue: given the string ab, the parser first consumes a, then b and returns by the rule $S \to ab$. Given the string a, the parser first consumes a and then expects symbol b. But this is not the case and the next alternative for S has to be tried by the parser: it consumes a and returns according to the second rule $S \to a$.

The following notion formalizes the intuition of how the rules should be ordered in a grammar.

Definition 6.6. *Let $G = (\Sigma, N, R, S)$ be a grammar with ordered rules, let uvy be the input string, and let A be a nonterminal symbol such that $A \to \varphi_1$, $\ldots$, $A \to \varphi_n$ are all the rules (in order) for A. Then A is said to have* prefix property*, if $\langle uv \rangle y \in L_G(\varphi_j)$ implies $\langle u \rangle vy \notin L_G(\varphi_i)$, for $i < j$.*

Note, that this definition can be considered as an extension of the prefix property, formulated by Wood [94], to the case of grammars with ordered rules.

Example 6.4. Consider the grammar G with the following rules.

$$S \to ab \mid a$$

For nonterminal S, $\varphi_1 = ab$ and $\varphi_2 = a$. Then, $\langle uv \rangle y \in L_G(\varphi_2)$ implies $uv = a$ and $y \in \Sigma^*$ and hence two possible factorizations of uv exist:

- $u = a$, $v = \varepsilon$, $y \in \Sigma^*$, then $\langle u \rangle vy = \langle a \rangle y \notin L_G(\varphi_1)$, as desired;
- $u = \varepsilon$, $v = a$, $y \in \Sigma^*$, that is, $\langle u \rangle vy = \langle \varepsilon \rangle ay$, which does not belong to $L_G(\varphi_1)$ either.

Thus, nonterminal S has the prefix property.

Consider now the grammar G, where the rules for S are reordered, that is, $\varphi_1 = a$ and $\varphi_2 = ab$.

$$S \to a \mid ab$$

Let $\langle uv \rangle y \in L_G(\varphi_2)$, with $y \in \Sigma^*$, that is, $uv = ab$ and the following cases are possible:

- $u = ab$, $v = \varepsilon$, $y \in \Sigma^*$ and $\langle u \rangle vy = \langle ab \rangle y \notin L_G(\varphi_1)$, as required;
- $u = \varepsilon$, $v = ab$, $y \in \Sigma^*$ gives $\langle u \rangle vy = \langle \varepsilon \rangle aby$, which is not in $L_G(\varphi_1)$ as well;
- $u = a$, $v = b$, $y \in \Sigma^*$ means that $\langle u \rangle vy = \langle a \rangle by$, and $\langle a \rangle by \in L_G(\varphi_1)$, which violates the condition.

Thus, another ordering of rules for S deprives the grammar of the prefix property and the grammar cannot be treated by the recursive descent parsing with backtracking.

Example 6.5. The following grammar G with ordered rules defines the nondeterministic context-free language $\{ a^n cb^n \mid n \geqslant 0 \} \cup \{ a^n cb^{2n} \mid n \geqslant 0 \}$.

$$\begin{aligned} S &\to C \mid D \\ C &\to aCb \mid c \\ D &\to aDbb \mid c \end{aligned}$$

Consider the rules for nonterminal S, and consider the string $\langle uv \rangle y = \langle a^n cb^n b^n \rangle y \in L_G(D)$, that is, with $y \in \Sigma^*$. Letting $u = a^n cb^n$ and $v = b^n$ gives that $\langle u \rangle vy = \langle a^n cb^n \rangle b^n y$ is in $L_G(C)$, which violates the prefix property. A grammar, where the the rules for S are reordered as $S \to D \mid C$, does not have the prefix property either. Indeed, let $\langle uv \rangle y = \langle a^n cb^n \rangle y \in L_G(C)$, with

$y \in \Sigma^*$, and consider the case when $n = 0$, $u = \varepsilon$, $v = c$, $y = \varepsilon$. Then the string $\langle\varepsilon\rangle c$ is in $L_G(D)$, which is not allowed by Definiton 6.6.

The following grammar is very similar to the grammar from Example 6.4, but it turns out that any order of rules preserves the prefix property.

Example 6.6. Consider the following grammar with contexts G_1:

$$S \to ab \mid a \,\&\, \rhd\varepsilon,$$

and the following grammar G_2, where the rules are ordered in a different way.

$$S \to a \,\&\, \rhd\varepsilon \mid ab$$

Both grammars define the language $\{a, ab\}$.

In the first case, $\varphi_1 = ab$ and $\varphi_2 = a \,\&\, \rhd\varepsilon$. Consider $\langle uv\rangle y \in L_{G_1}(\varphi_2)$, that is, $\langle uv\rangle y = \langle a\rangle\varepsilon$.

- $u = a$, $v = \varepsilon$, $y = \varepsilon$, then $\langle u\rangle vy = \langle a\rangle\varepsilon \notin L_{G_1}(\varphi_1)$, as desired;
- $u = \varepsilon$, $v = a$, $y = \varepsilon$, that is, $\langle u\rangle vy = \langle\varepsilon\rangle a$, which is not in $L_{G_1}(\varphi_1)$, as required.

In the second case, $\varphi_1 = a \,\&\, \rhd\varepsilon$ and $\varphi_2 = ab$. Considering $\langle uv\rangle y = \langle ab\rangle y$, $y \in \Sigma^*$, gives the following three possibilities:

- $u = a$, $v = b$, $y \in \Sigma^*$, and $\langle a\rangle by \notin L_{G_2}(\varphi_1)$;
- $u = \varepsilon$, $v = ab$, $y \in \Sigma^*$, and $\langle\varepsilon\rangle aby \notin L_{G_2}(\varphi_1)$;
- $u = ab$, $v = \varepsilon$, $y \in \Sigma^*$, and $\langle ab\rangle y \notin L_{G_2}(\varphi_1)$, as desired.

The following example demonstrates that the prefix property alone is not sufficient for a grammar to be tractable by the recursive descent method with limited backtrack.

Example 6.7. Consider the grammar with the following rules.

$$\begin{aligned} A &\to aB \\ B &\to b \\ B &\to \varepsilon \\ S &\to Abb \end{aligned}$$

In this grammar, only nonterminal B has alternative rules and in fact, this nonterminal has the prefix property. Let the input string be abb. The

parser starts with a call to procedure S, which, in its turn, invokes A and then b twice. The procedure A consumes symbol a and invokes B, which consumes b by the rule $B \to b$. Thus far, the prefix ab has been consumed by the rule $A \to aB$. At this point the parser, according to the rule $S \to Abb$, expects two symbols b, and successfully consumes the first of them. Now the unread part of the input is ε, whereas the parser still expects the last b to appear.

Thus, alongside with the prefix property, one more restriction has to be imposed on a grammar with ordered rules. Such a condition, called *separability* of a nonterminal, shall represent an intuitive idea of a "border" between two strings. Before a formal definition can be given, a technical notion has to be introduced.

Definition 6.7. *Let $L, K \subseteq \Sigma^* \times \Sigma^*$ be two languages of pairs. Then the* proper left quotient *of L by K, denoted as $K^{-1}L$, is the set $\{ \langle w \rangle v \mid \langle x \rangle wv \in K, \langle xw \rangle v \in L \}$. For $k \geqslant 0$, the pair (L, K) is said to be* k-quasi-separable *[94], if $\textsc{First}_k\left(L^{-1}L\right) \cap \textsc{First}_k(K) = \varnothing$.*

Example 6.8. Consider the following grammar generating the language $\{ a^m b^n \mid m \geqslant n, n \geqslant 0 \}$, which is not LL($k$) context-free for any $k \geqslant 0$.

$$\begin{array}{rcl} S & \to & AB \\ A & \to & aA \mid \varepsilon \\ B & \to & aBb \mid \varepsilon \end{array}$$

The language generated by nonterminal A is a^* (in any right context), while nonterminal B generates the language $a^n b^n$, with $n \geqslant 0$ (in any right context).

The languages $\{ \langle a^n \rangle y \mid n \geqslant 1,\ y \in \Sigma^* \}$ and $\{ \langle a \rangle a^{n-1} b^n \mid n \geqslant 1 \}$ are not separable.

Example 6.9. Consider the following grammar with right contexts.

$$\begin{array}{rcl} S & \to & X \,\&\, C \\ C & \to & A \,\&\, \rhd ab \\ A & \to & aA \mid \varepsilon \\ X & \to & aX \mid bX \mid \varepsilon \end{array}$$

Clearly, A defines a^* in any right context. The context specification in the rule for nonterminal C assumes that a^* should be followed by the string ab. As in the previous example, the languages $\{ \langle a^n \rangle y \mid n \geqslant 1,\ y \in \Sigma^* \}$ and $\{\langle ab \rangle \varepsilon\}$ are not separable.

Now the definition of a grammar allowing recursive descent parsing with limited backtracking can be given.

Definition 6.8. *Let $G = (\Sigma, N, R, S)$ be a grammar with ordered rules and let $k > 0$. Then G allows limited backtracking, if*

(I) every nonterminal $A \in N$ has the prefix property;

(II) for any $A, B \in \Sigma \cup N$ such that B follows A, the pair $(L_G(A), L_G(B))$ is k-quasi-separable.

Note, that in general, checking whether an arbitrary grammar with contexts satisfies Definition 6.8 is undecidable. This is caused by the undecidability of the separability property, which, in its turn, follows from the undecidability of emptiness of intersection [2].

6.6 Proof of correctness

The correctness of the recursive descent parsing algorithm with limited backtracking for grammars with right contexts is stated in the following lemmata. It has to be shown that the algorithm always terminates and that it accepts a string $w \in \Sigma^*$ if and only if this string is defined by the grammar.

Lemma 6.4. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts. Let $k \geqslant 0$. Let $T : N \times \Sigma^{\leqslant k} \to 2^R$ be an arbitrary function, with the sole assumption that $T(A, w)$ always gives a (possible empty) set of rules for A. Let the set of procedures be constructed with respect to G and T. Then:*

1. *for every $A \in \Sigma \cup N$ and $w, v \in \Sigma^*$, if $A()$ invoked on the input wv returns, consuming w, then $\langle w \rangle v \in L_G(A)$;*

2. *for every $A, B \in N$ and $w, v \in \Sigma^*$, if $A()$ is invoked on wv, and the resulting computation eventually leads to a call to $B()$ on the input v, then $\varepsilon[A]\varepsilon \rightsquigarrow^+ \xi[B]\eta$, where $\langle w \rangle v \in L_G(\xi)$.*

Proof. The **first part** of the lemma is proved inductively on the height of the tree of recursive calls made by the procedure $A()$ on the input wv. Since $A()$ terminates by the assumption, this tree is finite and its height is well-defined.

Basis. Let the call of the procedure $s()$ make no recursive calls and return.

If $s = a \in \Sigma$ and $a()$ returns on wv, consuming w, then $w = a$ and obviously $\langle a \rangle v \in L_G(a)$.

If $s = A \in N$ and $A()$ returns without making any recursive calls, then at least one of the rules which could be chosen upon entering $A()$, has to contain a conjunct ε. Then $w = \varepsilon$ and $\langle \varepsilon \rangle v \in L_G(A)$.

Induction step. Let $A()$ return on the input wv, consuming w, and let the height of the tree of recursive calls made by the procedure $A()$ be h. At the beginning of the computation, the procedure $A()$ looks up the cell of the parsing table $T_k[A, \textsc{First}_k(wv)]$, to find the set of rules of the form

$$A \to Q_1\alpha_1 \,\&\, \ldots \,\&\, Q_n\alpha_n, \tag{6.2}$$

with $n \geqslant 1$, $\alpha_i \in (\Sigma \cup N)^*$. Then for every rule in the set, whenever the control is passed to it, the code fragments for all its conjuncts are executed.

- Consider a conjunct α_j, with $\alpha_j = s_1 \ldots s_\ell$. Then the code $s_1(); \ldots ; s_\ell()$ is executed on wv, and it returns, consuming w. Consider a factorization $w = w_1 \ldots w_\ell$ defined by the positions of the pointer within the input after each procedure $s_i()$ returns and before the next procedure, $s_{i+1}()$, is called. Each $s_i()$ returns on $w_i w_{i+1} \ldots w_\ell v$, consuming w_i, and the height of recursive calls made by $s_i()$ does not exceed $h-1$. By the induction hypothesis, the string $\langle w_i\rangle w_{i+1} \ldots w_\ell v$ is in $L_G(s_i)$. Concatenating such strings for all i, one obtains that

$$\langle w\rangle v = \langle w_1\rangle w_2 \ldots w_\ell v \cdot \ldots \cdot \langle w_\ell\rangle v \in L_G(s_1) \cdot \ldots \cdot L_G(s_\ell) = L_G(\alpha_j). \tag{6.3}$$

- Consider a conjunct $\triangleright\alpha_j$, with $\alpha_j = s_1 \ldots s_\ell$. Then the code $s_1(); \ldots ; s_\ell()$ is executed on wv, and it returns, consuming wv. Consider a factorization $wv = x_1 \ldots x_\ell$ defined by the positions of the pointer within the input after each procedure $s_i()$ returns and before the next procedure, $s_{i+1}()$, is called. Each $s_i()$ returns on $x_i x_{i+1} \ldots x_\ell$, consuming x_i, and the height of recursive calls made by $s_i()$ does not exceed $h-1$. By the induction hypothesis, $\langle x_i\rangle x_{i+1} \ldots x_\ell \in L_G(s_i)$. Again, concatenating the latter expression for all i gives that

$$\langle wv\rangle\varepsilon = \langle x_1\rangle x_2 \ldots x_\ell \cdot \ldots \cdot \langle x_\ell\rangle\varepsilon \in L_G(s_1) \cdot \ldots \cdot L_G(s_\ell) = L_G(\alpha_j). \tag{6.4}$$

- Consider a conjunct $\triangleright\alpha_j$, with $\alpha_j = s_1 \ldots s_\ell$. Then the code $s_1(); \ldots ; s_\ell()$ is executed on v, and it returns, consuming v. Consider a factorization $v = v_1 \ldots v_\ell$ defined by the positions of the pointer within the input after each procedure $s_i()$ returns and before the next procedure, $s_{i+1}()$, is called. Each $s_i()$ returns on $v_i v_{i+1} \ldots v_\ell$, consuming v_i, and the height of recursive calls made by $s_i()$ does not exceed $h-1$. By the induction hypothesis, $\langle v_i\rangle v_{i+1} \ldots v_\ell \in L_G(s_i)$. Concatenating the latter for all i, the following is obtained:

$$\langle v\rangle\varepsilon = \langle v_1\rangle v_2 \ldots v_\ell \cdot \ldots \cdot \langle v_\ell\rangle\varepsilon \in L_G(s_1) \cdot \ldots \cdot L_G(s_\ell) = L_G(\alpha_j). \tag{6.5}$$

Combining (6.3)–(6.5) together for all j, the desired item $\vdash_G A\big(\langle w\rangle v\big)$ can be deduced, thus proving the induction step.

Turning to the **second part** of the lemma, if the procedure $A()$ starts on the input wv, and the procedure $B()$ is called on v at some point of the computation, then consider the partial tree of recursive calls made up to this point. The proof is by induction on h, the length of the path from the root to this last instance of $B()$.

Basis. Let $h = 0$. If $A()$ coincides with $B()$, that is, $B()$ is called on the same string $wv = v$, then $w = \varepsilon$, $\varepsilon[A]\varepsilon \rightsquigarrow^* \varepsilon[A]\varepsilon$ and $\langle\varepsilon\rangle v \in L_G(\varepsilon)$.

Induction step. The procedure $A()$, called on the string wv, begins with determining a set of rules of the form (6.2) using $T_k[A, \textsc{First}_k(wv)]$, and then proceeds with calling the subroutines corresponding to the symbols in the right-hand side of such rules. Some of these calls terminate, while the last one recorded in the partial computation history leads down to $B()$. Let $\xi C\eta$ be a conjunct in which this happens, and $C()$ be this call leading down. Consider a factorization $w = xy$ (with $x = x_1 \dots x_\ell$), such that $C()$ is called on the string yv.

Let $\xi = s_1 \dots s_\ell$. The call to $C()$ is preceded by the calls to $s_1(); \dots; s_\ell()$, where each $s_i()$ is called on $x_i \dots x_\ell yv$ and returns, consuming x_i. By the first part of the present lemma, this implies that $\langle x_i\rangle x_{i+1} \dots x_\ell yv \in L_G(s_i)$, and hence $\langle x\rangle yv \in L_G(\xi)$.

Since a conjunct of the form $\xi C\eta$ always exists, $\varepsilon[A]\varepsilon \rightsquigarrow \xi[C]\eta$. For the partial computation of $C()$ on yv (up to call of the procedure $B()$), the distance between $C()$ and $B()$ is $h - 1$, which allows applying the induction hypothesis and obtain that $\varepsilon[C]\varepsilon \rightsquigarrow^* \gamma[B]\delta$, with $\langle y\rangle v \in L_G(\gamma)$.

According to Definition 6.1, $\varepsilon[A]\varepsilon \rightsquigarrow^+ \xi\gamma[B]\delta\eta$.

Moreover, $\xi\big(\langle x\rangle yv\big) \cdot \gamma\big(\langle xy\rangle v\big) = \xi\gamma\big(\langle xy\rangle v\big) = \xi\gamma\big(\langle w\rangle v\big)$, that is, $\langle w\rangle v \in L_G(\xi\gamma)$. □

Lemma 6.5. *Let $G = (\Sigma, N, R, S)$ be a non-left-recursive grammar with right contexts allowing limited backtracking. Let $k \geqslant 0$. Let $T : N \times \Sigma^{\leqslant k} \to 2^R$ be an arbitrary function, with the sole assumption that $T(A, w)$ does always give a (possibly empty) set of rules for A. Let the set of procedures be constructed with respect to G and T. Then, for every $s \in \Sigma \cup N$ and $w \in \Sigma^*$, the procedure $s()$ terminates on the input w, either by consuming a prefix of w and returning, or by raising an exception.*

Proof. Suppose there exists $s \in \Sigma \cup N$ and $w \in \Sigma^*$, such that $s()$ does not halt on the input w. Consider the tree of recursive calls (which is infinite in that case), the nodes of which are labeled with pairs (t, u), where $t \in \Sigma \cup N$ and u is some suffix of w.

Since the right-hand side of every rule is finite, every procedure makes finitely many recursive calls, and the tree has finite branching. Hence, by König's lemma, this tree should contain an infinite path

$$(A_1, u_1), (A_2, u_2), \dots, (A_i, u_i), \dots, \tag{6.6}$$

where $(A_1, u_1) = (s, w)$, $A_i \in N$ and each procedure $A_i()$ is invoked on u_i and, after calling some procedures that terminate, eventually calls A_{i+1} on the string u_{i+1}, which is a suffix of u_i. This means that $|u_1| \geqslant |u_2| \geqslant \ldots \geqslant |u_i| \geqslant \ldots$

Since w is of finite length, it has finitely many different suffixes, and the decreasing second component in the path (6.6) should converge to some shortest reached suffix of the input w. Let u be such a suffix, and consider any node (A, u) that appears multiple times in the path (6.6). Consider any two instances of (A, u); then, by Lemma 6.4, $\varepsilon[A]\varepsilon \rightsquigarrow^* \xi[A]\eta$, with $\varepsilon \in L(\xi)$. This contradicts the assumption that G is a non-left-recursive grammar. □

Lemma 6.6. *Let $G = (\Sigma, N, R, S)$ be a grammar with right contexts allowing limited backtracking. Let $k \geqslant 0$. Let $T : N \times \Sigma^{\leqslant k} \to 2^R$ be a parsing table for G, and let the set of procedures be constructed with respect to G and T. Let $w, v \in \Sigma^*$ and $s_1, \ldots, s_\ell \in \Sigma \cup N$ (with $\ell \geqslant 0$), and assume that there exists $\hat{v} \in \Sigma^*$, such that $\hat{v}$ follows $s_1 \ldots s_\ell$ and $\textsc{First}_k(\hat{v}) = \textsc{First}_k(v)$. Then the code $s_1(); \ldots; s_\ell()$ returns on the input wv, consuming w, if and only if $\langle w \rangle v \in L_G(s_1 \ldots s_\ell)$.*

Proof. According to Lemma 6.5, the code $s_1(); \ldots s_\ell()$ always terminates, either by returning or by raising an exception. Consider the tree of recursive calls of the code $s_1(); \ldots s_\ell()$ executed on the input wv, as in Lemma 6.5. This tree is finite, and let h be its height.

The proof is by induction on the pair (h, ℓ), where pairs are ordered as $(h, \ell) \prec (h', \ell')$ if $h < h'$ or if $h = h'$ and $\ell < \ell'$. Besides the trivial case when $h = \ell = 0$, another natural base case is $h = 0$ and $\ell = 1$.

Basis I is $(0, 0)$, that is, $s_1 \ldots s_\ell = \varepsilon$. Obviously, $L_G(\varepsilon) = \{ \langle \varepsilon \rangle w \mid w \in \Sigma^* \}$, and accordingly the empty statement returns on wv, consuming w, if and only if $w = \varepsilon$.

Basis II is $(0, 1)$, $s = a \in \Sigma$. The procedure $a()$ is constructed so that it returns on the input wv, consuming w, if and only if $w = a$. That is, $w \in L_G(a) = \{ \langle a \rangle v \mid v \in \Sigma^* \}$.

Induction step I is $(h - 1, \ldots) \longrightarrow (h, 1)$, or $(0, 1)$ and $s \in N$. Let $\ell = 1$ and $s_1 = A \in N$, let $\hat{v}$ follow A, with $\textsc{First}_k(\hat{v}) = \textsc{First}_k(v)$, and let h be the height of the tree of recursive calls made by the procedure $A()$ executed on wv.

$\Leftarrow$ If $A()$ returns on wv, consuming w, then $T_k[A, \textsc{First}_k(wv)]$ gives a nonempty set of rules of the form

$$A \to Q_1\alpha_1 \,\&\, \ldots \,\&\, Q_n\alpha_n. \tag{6.7}$$

Suppose that this set contains more than one rule. Let $r \in T_k[A, \textsc{First}_k(wv)]$ be a rule such that it is the first with respect to the order introduced on R.

Then, for this rule, the following computations take place.

- For every **base conjunct** of the form α_i, with $\alpha_i = s_1 \dots s_\ell$, the code $s_1(); \dots; s_\ell()$ is called on the string wv. It either returns, consuming w, or raises an exception; in the latter case the next rule from $T_k[A, \textsc{First}_k(wv)]$ is chosen and treated in a similar way.

  Since the computation of $s_1(); \dots s_\ell()$ on wv is a subcomputation of the computation of $A()$ on wv, the height of the tree of recursive calls corresponding to this subcomputation does not exceed $h - 1$. The string $\hat{v}$ follows $s_1 \dots s_\ell$, since $\hat{v}$ follows A and $s_1 \dots s_\ell$ is a conjunct of some rule for A.

  Hence, by the induction hypothesis, $\langle w \rangle v \in L_G(\alpha_i)$.

- For every **conjunct of the form** $\triangleright\!\!\!=\alpha_i$, with $\alpha_i = s_1 \dots s_\ell$, the code $s_1(); \dots; s_\ell()$ is called on the string wv. It returns consuming wv and then rewinding the position of the parser to where it had been before the processing of the substring wv.

  Similarly to the previous case, by the induction hypothesis, $\langle wv \rangle \varepsilon \in L_G(s_1 \dots s_\ell)$.

- For every **conjunct of the form** $\triangleright\alpha_i$, similarly to the previous cases, $\langle v \rangle \varepsilon \in L_G(s_1 \dots s_\ell)$. It returns consuming v and then rewinding the position of the parser to the beginning of the substring v.

Combining the results for individual conjuncts, one can obtain that

$$\langle w \rangle v \in L_G(Q_1\alpha_1 \,\&\, \dots \,\&\, Q_n\alpha_n),$$

where $A \to Q_1\alpha_1 \,\&\, \dots \,\&\, Q_n\alpha_n$ is exactly one of the rules (6.7).

Thus, $\langle w \rangle v \in L_G(A)$.

$\Leftarrow$ If $\langle w \rangle v \in L_G(A)$, then there exists a nonempty set of rules of the form (6.7), such that

$$\langle w \rangle v \in L_G(Q_1\alpha_1 \,\&\, \dots \,\&\, Q_n\alpha_n) \tag{6.8}$$

for one of them, which is the least with respect to the order on R.

Since $\hat{v}$ follows A,

- $\varepsilon[S]\varepsilon \rightsquigarrow^* \xi[A]\eta$, with $\hat{v} \in L_G(\eta)$;
- for each conjunct $\triangleright\!\!\!=\alpha_i$, $\varepsilon[\alpha_i]\varepsilon \rightsquigarrow^* \xi[A]\eta$, with $\hat{v} \in L_G(\eta)$;
- for each conjunct $\triangleright\alpha_i$, $\varepsilon[\alpha_i]\varepsilon \rightsquigarrow^* \xi[A]\eta$, with $\hat{v} \in L_G(\eta)$.

Thus, by (6.8), $\langle w\hat{v} \rangle \varepsilon \in L_G((Q_1\alpha_1 \,\&\, \dots \,\&\, Q_n\alpha_n) \cdot \eta)$.

Then, by Definition 6.4, $T_k[A, \textsc{First}_k(w\hat{v})]$ contains the rule (6.7).

Furthermore, since $\textsc{First}_k(w\hat{v}) = \textsc{First}_k(wv)$, the rule (6.7) is contained (among other possible rules) in $T_k[A, \textsc{First}_k(wv)]$.

The computation of $A()$ on wv starts by choosing some rule of the form (6.7). For each such rule, consider its conjuncts in the order of the corresponding code fragments, and prove that each of them is successfully passed.

1. For each conjunct of the form α_i, with $\alpha_i = s_1 \dots s_\ell$, by (6.8), $\langle w \rangle v \in L_G(s_1 \dots s_\ell)$, $\hat{v}$ follows A and hence $\hat{v}$ follows $s_1 \dots s_\ell$ (since $s_1 \dots s_\ell$ is a conjunct of the rule for A).

   By the induction hypothesis, the code $s_1(); \dots ; s_\ell()$ returns on wv, consuming w.

2. For each conjunct of the form $\triangleright\alpha_i$, with $\alpha_i = s_1 \dots s_\ell$, similarly to the previous case, $\langle wv \rangle \varepsilon \in L_G(s_1 \dots s_\ell)$, and by the induction hypothesis, the code $s_1(); \dots ; s_\ell()$ returns on wv, consuming wv and rewinding the pointer where it has been before the processing of the substring wv.

3. For each conjunct of the form $\triangleright\alpha_i$, with $\alpha_i = s_1 \dots s_\ell$. Similarly to the previous cases, $\langle v \rangle \varepsilon \in L_G(s_1 \dots s_\ell)$. By the induction hypothesis, the code $s_1(); \dots ; s_\ell()$ returns on v, consuming v, and rewinding the pointer where it has been before the processing of the substring v.

Thus, all the conjuncts of the rule (6.7) are processed.

The final assignment in the code for this rule rewinds the pointer to the location where it was put by the code for the first *base* conjunct.

Thus, $A()$ returns, consuming y.

Induction step II is $(h, \ell-1) \to (h, \ell)$. Let $\ell \geqslant 2$, let $s_1 \dots s_\ell \in (\Sigma \cup N)^*$ and let $\hat{v}$ follow $s_1 \dots s_\ell$, where $\textsc{First}_k(\hat{v}) = \textsc{First}_k(v)$.

$\Rightarrow$ Let the code $s_1(); \dots ; s_{\ell-1}()$ return on input wv, consuming w. Consider the position of the parser after $s_{\ell-1}()$ returns and before s_ℓ is to be called: the string w can be factorized as $w = xy$, such that the code $s_1(); \dots ; s_{\ell-1}()$ returns on xyv, consuming x, and the code $s_\ell()$ returns on yv, consuming y.

The height of the recursive calls in these subcomputation does not exceed that of the whole computation.

Since $\hat{v}$ follows s_ℓ, the induction hypothesis is applicable to the computation of $s_\ell()$ on yv, yielding $\langle y \rangle v \in L_G(s_\ell)$.

The induction hypothesis can be applied to the former $\ell - 1$ calls, provided the string $y\hat{v}$ follows $s_1 \dots s_{\ell-1}$: since $\hat{v}$ follows $s_1 \dots s_\ell$, then $\varepsilon[\delta]\varepsilon \rightsquigarrow^* \xi[s_1 \dots s_\ell]\eta$, where $\hat{v} \in L_G(\eta)$, and $\delta = S$ or δ is a conjunct with a context operator. Thus, $\varepsilon[\delta]\varepsilon \rightsquigarrow^* \xi[s_1 \dots s_{\ell-1}]s_\ell\eta$. Since $\langle y \rangle \hat{v} \in L_G(s_\ell)$ and $\langle \hat{v} \rangle \varepsilon \in L_G(\eta)$, it holds that $\langle y\hat{v} \rangle \varepsilon \in L_G(s_\ell\eta)$.

Since $\textsc{First}_k(y\hat{v}) = \textsc{First}_k(yv)$, by the induction hypothesis applied to the computation of the code $s_1(); \dots ; s_\ell()$ on xyv, which returns, consuming x, $\langle x \rangle yv \in L_G(s_1 \dots s_{\ell-1})$.

Since $\langle x\rangle yv \in L_G(s_1 \dots s_{\ell-1})$ and $\langle y\rangle v \in L_G(s_\ell)$, $\langle xy\rangle v \in L_G(s_1 \dots s_\ell)$. That is, $\langle w\rangle v \in L_G(s_1 \dots s_\ell)$.

$\Leftarrow$ To prove the other direction, consider the string $\langle w\rangle v \in L_G(s_1 \dots s_{\ell-1}s_\ell)$, where w can be factorized as $w = xy$ such that $\langle x\rangle yv \in L_G(s_1 \dots s_{\ell-1})$ and $\langle y\rangle v \in L_G(s_\ell)$.

The computation of the code $s_1(); \dots ; s_{\ell-1}()$ is a subcomputation of the computation $s_1(); \dots s_{\ell-1}(); s_\ell()$. Hence, the recursion depth for the subcomputation does not exceed that of the whole computation.

Since $\langle y\rangle v \in L_G(s_\ell)$ and $\hat{v}$ follows $s_1 \dots s_{\ell-1}s_\ell$, $y\hat{v}$ follows $s_1 \dots s_{\ell-1}$.

Thus, by the induction hypothesis applied to this subcomputation, the code $s_1(); \dots ; s_{\ell-1}()$ returns on xyv, consuming x.

After the subcomputation $s_1(); \dots s_{\ell-1}()$ returns on xyv, the computation of $s_1(); \dots s_{\ell-1}(); s_\ell()$ invokes $s_\ell()$ on yv. Hence, $s_\ell()$ on yv is also a subcomputation with the height not greater than that of the whole computation.

Since $\hat{v}$ follows s_ℓ, the induction hypothesis can be applied, and $\langle y\rangle v \in L_G(s_\ell)$ implies that the procedure $s_\ell()$ returns on yv, consuming y.

Therefore, the sequential composition of the two subcomputations, that is, $s_1(); \dots s_{\ell-1}(); s_\ell()$ returns on wv, consuming $xy = w$. □

6.7 Implementation

This section discusses the implementation of the recursive descent parsing algorithm for grammars with right contexts.

6.7.1 Complexity of the algorithm

The algorithm in the form it is presented, has exponential time complexity, which is caused by recomputing the same procedures with the same value of the pointer in different branches of the computation. The technique of *memoization*, that is, storing the results of the first computation in memory and then looking it up for every subsequent call of the same procedure with the same arguments, can be used to avoid such behaviour of the parser.

Listing 6.10. The redefined code for a nonterminal $A \in N$:

```
A() {
    if M[A][p] ∈ {1, ..., n, n + 1}
        then p :=M[A][p]
        else if M[A][p] = error
                then report error
                else {
```

```
                    let start :=p;
                    M[A][start] :=error;
                    //the code for A() as in Listing 6.2 (or 6.8)
                    M[A][start] :=p;
                }
    }
```

The *memoized recursive descent* uses a two-dimensional array $M[A][i]$, indexed by a nonterminal in one dimension and a position of the pointer in the other. Each element belongs to $\{undef, error\} \cup \{1, \ldots, n, n+1\}$ and in the beginning, $M[A][i] = undef$ for all $A \in N$ and $i \in \{1, \ldots, n+1\}$. The value $M[A][i] = undef$ means that the procedure $A()$ has not been called thus far with $p = i$. If $M[A][i] = j$, this means that $A()$ called with the value of pointer $p = i$ returns and the value of the pointer after it is j. In the case $M[A][i] = error$, the call to $A()$ with $p = i$ leads to a parsing error. This allows reducing the complexity of the algorithm to *linear*.

6.7.2 Extended grammars with contexts

Let us define an extension of the model, where rules with regular right-hand side parts are allowed, similarly to the case of context-free grammars [8, 23, 24]. First a notion of an *expression* is defined.

Definition 6.9. *Let Δ be a finite alphabet.* An expression *is defined inductively as follows:*

- *ε is an expression (*"empty expression"*);*
- *for all $a \in \Delta$, a is an expression (*"symbol expression"*);*
- *$\mathbb{E}_1 \cdot \ldots \cdot \mathbb{E}_n$ is an expression (*"concatenation expression"*), for expressions $\mathbb{E}_1, \ldots, \mathbb{E}_n$;*
- *$\mathbb{E}_1 \mid \ldots \mid \mathbb{E}_n$ is an expression (*"disjunction expression"*), for expressions $\mathbb{E}_1, \ldots, \mathbb{E}_n$;*
- *$Q_1\mathbb{E}_1 \,\&\, \ldots \,\&\, Q_n\mathbb{E}_n$ is an expression (*"conjunction expression"*), for expressions $\mathbb{E}_1, \ldots, \mathbb{E}_n$, and for operators $Q_1, \ldots, Q_n \in \{\trianglerighteq, \square, \triangleright\}$, such that there exists $1 \leqslant i \leqslant n$, such that $Q_i = \square$, and whenever $Q_i = \triangleright$, then $Q_j = \square$ for some $j < i$;*
- *$(\mathbb{E})^*$ is an expression (*"iteration expression"*), for an expression $\mathbb{E}$;*
- *$(\mathbb{E})^+$ is an expression (*"positive iteration expression"*), for an expression $\mathbb{E}$;*

- $[\mathbb{E}]$ *is an expression (*"optional expression"*), for an expression* $\mathbb{E}$*.*

Now a definition of a new model can be given.

Definition 6.10. An extended grammar with contexts *is a quadruple* $G = (\Sigma, N, R, S)$*, where*

- Σ*,* N *and* S *are the same as in Definition 2.1;*
- R *is a finite set of* extended rules*, each of the form* $A \to \mathbb{E}$*, where* $A \in N$ *and* $\mathbb{E}$ *is a disjunction expression over* $\Sigma \cup N$*.*

In fact, every extended grammar with contexts can be converted to a grammar which does not have any regular expressions in the right-hand sides of the rules.

Theorem 6.1. *Let* $G = (\Sigma, N, R, S)$ *be an extended grammar with contexts. Then there can be effectively constructed a grammar with ordered rules* $G' = (\Sigma, N', R', S)$*, such that* $L(G) = L(G')$*.*

The construction is similar to the conversion of a context-free grammar in the extended Backus–Naur form to an ordinary grammar, and is omitted here.

The subroutines of a recursive descent parser for extended grammars are defined as follows.

Listing 6.11. For every nonterminal $A \in N$ and for every rule $A \to \mathcal{E}$ the subroutine is

$A()$ {
    $\mathcal{C}(\mathcal{E})$
}

where $\mathcal{C}(\mathcal{E})$ is the corresponding code for the expression $\mathcal{E}$.

For every expression $\mathcal{E}$, the code $\mathcal{C}(\mathcal{E})$ is defined inductively on the structure of the expression as follows.

- If $\mathcal{E} = \varepsilon$ is an empty expression, then $\mathcal{C}(\mathcal{E})$ is defined as a statement performing no action.

    skip

- If $\mathcal{E} = \delta$, with $\delta \in \Delta = \Sigma \cup N$, is a symbol expression, then $\mathcal{C}(\mathcal{E})$ is defined as follows.

    call δ ();

- If $\mathcal{E} = \mathcal{E}_1 \cdot \ldots \cdot \mathcal{E}_n$ is a concatenation expression, then $\mathcal{C}(\mathcal{E})$ corresponds to a compound statement composed of the codes of the expressions $\mathcal{E}_1$, $\ldots$, $\mathcal{E}_n$, i.e. $\mathcal{C}(\mathcal{E}) = \mathcal{C}(\mathcal{E}_1) \ldots \mathcal{C}(\mathcal{E}_n)$.

- If $\mathcal{E} = \mathcal{E}_1 \mid \ldots \mid \mathcal{E}_n$ is a disjunction expression, then $\mathcal{C}(\mathcal{E})$ is defined as follows.

```
boolean failed :=false;
integer pos; (omit this if n = 1)
try {
    pos :=p;
    C(E1)
}
catch {
    try {
    p :=pos;
    C(E2)
        ...
            try {
                p :=pos;
                C(En)
            }
            catch {
                failed :=true;
            }
    }
        ...
if failed then raise;
```

  The underlying idea of the presented code is that the first of rule for a nonterminal is tried, and if it fails, then the next (in the order given) shall be tried. This process repeats until one of the possible rules matches the string. If this does not happen (that is, all the rules have failed), an error is raised.

- If $\mathcal{E} = (\mathcal{E}_1)^*$ is an iteration expression, then $\mathcal{C}(\mathcal{E})$ is defined as follows.

```
boolean failed = false;
integer pos :=p;
try {
    while ¬ failed {
```

$\mathcal{C}(\mathcal{E}_1)$
pos :=p;
}
}
catch {
$failed$:=**true**;
}
p :=pos;

- If $\mathcal{E} = \mathcal{E}_1^+$ is a positive iteration expression, $\mathcal{C}(\mathcal{E})$ is defined as follows.

boolean $failed$ = **false**;
boolean $looped$ = **false**;
integer pos :=p;
try {
do {
$\mathcal{C}(\mathcal{E}_1)$
pos :=p;
$looped$:=**true**;
} **while** ¬ $failed$;
}
catch {
$failed$ = **true**;
p :=pos;
}
if ¬ $looped$ **then raise**;

- If $\mathcal{E} = [\mathcal{E}_1]$ is an optional expression, $\mathcal{C}(\mathcal{E})$ is defined as follows.

integer pos :=p;
try {
$\mathcal{C}(\mathcal{E}_1)$
pos :=p;
}
catch {
p :=pos;
}

- If $\mathcal{E} = Q_1\mathcal{E}_1 \,\&\, \ldots \,\&\, Q_n\mathcal{E}_n$, $\mathcal{C}(\mathcal{E})$ is defined as follows.

integer *start*;
integer *end*;
$\mathcal{D}(Q_1\mathcal{E}_1)$
...
$\mathcal{D}(Q_n\mathcal{E}_n)$

Depending on the operator, the code $\mathcal{D}(Q_i\mathcal{E}_i)$ is defined as follows.

- if $Q_i = \trianglerighteq$, then $\mathcal{D}(Q_i\mathcal{E}_i)$ is

  start :=*p*; *(if this is the very first conjunct in the rule)*
  end :=*p*; *(if this is the very first conjunct in the rule)*
  p :=*start*; *(otherwise)*
  $\mathcal{C}(\mathcal{E}_i)$
  if $p \neq |w|$ **then raise**;

- if $Q_i = \square$, then $\mathcal{D}(Q_i\mathcal{E}_i)$ is

  start :=*p*; *(if this is the very first conjunct in the rule)*
  p :=*start*; *(otherwise)*
  $\mathcal{C}(\mathcal{E}_i)$
  end :=*p*; *(if this is the first base conjunct in the rule)*
  if $p \neq$ *end* **then raise**;

- if $Q_i = \triangleright$, then $\mathcal{D}(Q_i\mathcal{E}_i)$ is

  p :=*end*;
  $\mathcal{C}(\mathcal{E}_i)$
  if $p \neq |w|$ **then raise**;

6.7.3 Prototype implementation

Letting the number of lookahead symbols $k = 1$ constitutes a special case of the recursive descent parser: it becomes possible to reformulate the code fragments for disjunction, iteration and optional expressions.

For every expression $\mathcal{E}$ over the alphabet $\Sigma \cup N$, the code $\mathcal{CT}(\mathcal{E})$ is defined inductively on the structure of the expression as follows:

- if $\mathcal{E}$ is a simple expression, then $\mathcal{CT}(\mathcal{E}) = \mathcal{C}(\mathcal{E})$;
- if $\mathcal{E}$ is an epsilon expression, then $\mathcal{CT}(\mathcal{E}) = \mathcal{C}(\mathcal{E})$;
- if $\mathcal{E}$ is a concatenation expression, then $\mathcal{CT}(\mathcal{E}) = \mathcal{C}(\mathcal{E})$;
- if $\mathcal{E}$ is a conjunction expression, then $\mathcal{CT}(\mathcal{E}) = \mathcal{C}(\mathcal{E})$;
- if $\mathcal{E} = (\mathcal{E}_1)^*$ is an iteration expression, then $\mathcal{CT}(\mathcal{E})$ is

integer *pos* :=*p*;
while $w_p \in \text{PFIRST}_1(\mathcal{E}_1)$ {
    $\mathcal{CT}(\mathcal{E}_1)$
    pos :=*p*;
}
p :=*pos*;

- if $\mathcal{E} = (\mathcal{E}_1)^+$ is a positive iteration expression, then $\mathcal{CT}(\mathcal{E})$ is

integer *pos* :=*p*;
do {
    $\mathcal{CT}(\mathcal{E}_1)$
    pos :=*p*;
} **while** $w_p \in \text{PFIRST}_1(\mathcal{E}_1)$
p :=*pos*;

- if $\mathcal{E} = [\mathcal{E}_1]$ is an optional expression, then $\mathcal{CT}(\mathcal{E})$ is

integer *pos* :=*p*;
if $w_p \in \text{PFIRST}_1(\mathcal{E}_1)$ **then**
    $\mathcal{CT}(\mathcal{E}_1)$
else
p :=*pos*;

- if $\mathcal{E} = \mathcal{E}_1 \mid \ldots \mid \mathcal{E}_n$ is a disjunction expression such that $A \to \mathcal{E} \in R$ for some nonterminal $A \in N$, then $\mathcal{CT}(\mathcal{E})$ is

switch *Peek*() {
    ⋮
    case $\text{PFIRST}_1(\mathcal{E}_i)$:
        $\mathcal{C}(\mathcal{E}_i)$
    break;
    case ... :
    ⋮
    default:
        raise;
    break;
}

Thus, a model of a recursive descent parsing technique has been defined. A prototype implementation of a parser generator for grammars with contexts, which further allows regular expressions in the right-hand sides of the rules [8, 23, 24, 55], has been developed. The expressive means of the grammars are experimentally augmented with the *negation* operation [56, 61].

Part III

Theoretical results

Chapter 7

Grammars with regular contexts

A special case of context restrictions that is often useful in applications in one or another form are *regular contexts*, meaning that a substring under consideration is preceded or followed by a string from a given *regular language*. For instance, one can consider regular expressions with context operators [38], or finite automata incorporating context restrictions [86]. In grammar-based parsing, there are LL-regular and LR-regular grammars [29, 11, 6], which have no explicit context operators, but the parsers analyze contexts using finite automata.

Although, by the complexity of parsing, grammars with contexts are comparable to ordinary grammars, they may be too powerful for some applications, where parsing in substantially less than cubic time is essential. For that reason, this chapter introduces a natural special case of grammars with contexts, where each context operator is applied to a regular language, given, for instance, by a regular expression.

As an introductory example, consider the following grammar describing the language $\{ a^n b^n \mid n \geqslant 0 \}$.

$$S \to aSa \mid aSb \mid bSa \mid bSb \mid \varepsilon \,\&\, {\lhd}a^* \,\&\, {\rhd}b^*$$

With the context operators removed, the grammar would define simply the language of all strings of even length. The left context operator in the rule $S \to \varepsilon \,\&\, {\lhd}a^* \,\&\, {\rhd}b^*$ ensures that the first half of the string is of the form a^*, while the right context operator restricts the right half of the string to be of the form b^*. Thus, S only defines strings of the desired form. Note, that the version of this grammar without context operators defines a regular language, but adding regular context operators makes the language defined by the grammar non-regular.

In this particular grammar, regular context operators can be eliminated by removing all rules that generate prefixes and suffixes contradicting the context specifications. The resulting grammar has only the rules $S \to aSb$ and $S \to \varepsilon$. As shown in the present chapter, regular context operators can be eliminated in any grammar, though this requires more than just omitting some rules.

7.1 Specifying a grammar with regular contexts

Grammars with regular contexts may be defined with regular languages specified by any standard means: for instance, the above example uses regular expressions. As the constructions presented in this chapter are best expressed using *deterministic finite automata*, the definition of grammars below assumes exactly that representation. First, consider the notation for automata assumed in what follows.

Definition 7.1. *A deterministic finite automaton (DFA) without an initial state is a quadruple $\mathcal{A} = (\Sigma, \mathcal{Q}, \{\delta_x\}, \mathcal{F})$, where:*

- *Σ is the input alphabet;*
- *$\mathcal{Q}$ is a finite set of states;*
- *for any input symbol $a \in \Sigma$, a partial transition function $\delta_a \colon \mathcal{Q} \to \mathcal{Q}$ defines the behaviour of the automaton on this symbol;*
- *$\mathcal{F}$ is the set of accepting states.*

The behaviour of the automaton on an arbitrary string over the alphabet Σ is defined as follows. For a string $w \in \Sigma^*$, there is an extended transition function $\delta_w \colon \mathcal{Q} \to \mathcal{Q}$, defined recursively as $\delta_\varepsilon(q) = q$ and $\delta_{wa}(q) = \delta_a(\delta_w(q))$, with $a \in \Sigma$.

In the above definition, the behaviour of an automaton on the empty string (δ_ε) is the *identity function* $\mathrm{id} \colon \mathcal{Q} \to \mathcal{Q}$, with $\mathrm{id}(q) = q$, for all $q \in \mathcal{Q}$.

Definition 7.2. *A grammar with regular contexts is a quintuple $G = (\Sigma, N, R, S, \mathcal{A})$, where Σ is the alphabet of the language being defined, N is a finite set of nonterminals, S is a symbol representing well-formed sentences of the language, $\mathcal{A} = (\Sigma, \mathcal{Q}, \delta, \mathcal{F})$ is a DFA without an initial state and each rule in R is of the form*

$$\begin{aligned} A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd p_1 \,\&\, \ldots \,\&\, \lhd p_m \,\&\, \trianglelefteq q_1 \,\&\, \ldots \,\&\, \trianglelefteq q_n \\ \&\, \trianglerighteq r_1 \,\&\, \ldots \,\&\, \trianglerighteq r_{m'} \,\&\, \rhd s_1 \,\&\, \ldots \,\&\, \rhd s_{n'}, \end{aligned} \tag{7.1}$$

where $p_i, q_i, r_i, s_i \in \mathcal{Q}$ are states of the DFA.

Every rule of the form

$$A \to X_{1,1} \dots X_{1,\ell_1} \,\&\, \dots \,\&\, X_{k,1} \dots X_{k,\ell_k} \,\&\, \lhd p_1 \,\&\, \dots \,\&\, \lhd p_m \,\&\, \trianglelefteq q_1 \,\&\, \dots \,\&\, \trianglelefteq q_n \\ \&\, \trianglerighteq r_1 \,\&\, \dots \,\&\, \trianglerighteq r_{m'} \,\&\, \rhd s_1 \,\&\, \dots \,\&\, \rhd s_{n'}$$

in R induces the following schema of deduction rules. A string with contexts $u\langle w\rangle v$ is defined by this rule, if for all strings $x_{i,j} \in \Sigma^*$, with $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant \ell_i$, that satisfy $x_{1,1} \dots x_{1,\ell_1} = \dots = x_{k,1} \dots x_{k,\ell_k} = w$, it holds that $\delta_u(p_1), \dots, \delta_u(p_m), \delta_{uw}(q_1), \dots, \delta_{uw}(q_n), \delta_{wv}(r_1), \dots, \delta_{wv}(r_{m'}), \delta_v(s_1), \dots, \delta_v(s_{n'}) \in \mathcal{F}$.

$$X_{1,1}\big(u\langle x_{1,1}\rangle x_{1,2} \dots x_{1,\ell_1} v\big), \dots, X_{1,\ell_1}\big(u x_{1,1} x_{1,2} \dots x_{1,\ell_1-1}\langle x_{1,\ell_1}\rangle v\big), \\ \vdots \\ X_{k,1}\big(u\langle x_{k,1}\rangle x_{k,2} \dots x_{k,\ell_k} v\big), \dots, X_{k,\ell_k}\big(u x_{k,1} x_{k,2} \dots x_{k,\ell_k-1}\langle x_{k,\ell_k}\rangle v\big) \\ \vdash_G A\big(u\langle w\rangle v\big)$$

The following example is given to demonstrate the definitions.

Example 7.1. Consider the following grammar with regular contexts defining the language $\{\, a^n b^m c^m d^n \mid n, m \geqslant 1,\ n \text{ is even} \,\}$.

$$S \to aSd \mid bSc \mid \varepsilon \,\&\, \lhd a^+ b^+ \,\&\, \lhd (b^* a b^* a b^*)^*$$

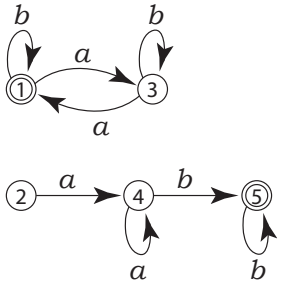
If the grammar did not have the context specifications, it would define all strings of the form $w \cdot h(w^R)$, where $w \in \{a, b\}^*$ and the homomorphism h maps symbol a to symbol d and symbol b to symbol c. The context operators in the last rule for S are used to check that the first half of the string is of the form $a^+ b^+$ and contains an even number of occurrences of the symbol a. Thus, the nonterminal S only defines strings of the desired form.

Using the automaton representation of the regular language defined by the context operator, the grammar can be reformulated as the following grammar with regular contexts $G = (\Sigma, N, R, S, \mathcal{A})$.

$$S \to aSd \mid bSc \mid \varepsilon \,\&\, \lhd ① \,\&\, \lhd ②$$

The set of states of the finite automaton $\mathcal{A}$ is $\mathcal{Q} = \{①, ②, ③, ④, ⑤\}$ and the transition functions are defined as follows.

$$\delta_a(①) = ③ \\ \delta_a(②) = \delta_a(④) = ④ \\ \delta_a(③) = ① \\ \delta_b(①) = ① \\ \delta_b(③) = ③ \\ \delta_b(④) = \delta_b(⑤) = ⑤$$



The accepting states of the automaton are states ① and ⑤.

Consider the following logical derivation of the fact that the string *aabcdd* is defined by the grammar.

$$\begin{array}{rll}
 & \vdash a\big(\varepsilon\langle a\rangle abcdd\big) & (axiom)\\
 & \vdash a\big(a\langle a\rangle bcdd\big) & (axiom)\\
 & \vdash b\big(aa\langle b\rangle cdd\big) & (axiom)\\
 & \vdash c\big(aab\langle c\rangle dd\big) & (axiom)\\
 & \vdash d\big(aabc\langle d\rangle d\big) & (axiom)\\
 & \vdash d\big(aabcd\langle d\rangle \varepsilon\big) & (axiom)\\
 & \vdash S\big(aab\langle\varepsilon\rangle cdd\big) & (S \to \varepsilon \,\&\, \lhd① \,\&\, \lhd②,\\
 & & \delta_{aab}(①), \delta_{aab}(②) \in \mathcal{F})\\
b\big(aa\langle b\rangle cdd\big), S\big(aab\langle\varepsilon\rangle cdd\big), c\big(aab\langle c\rangle dd\big) \vdash & S\big(aa\langle bc\rangle dd\big) & (S \to bSc)\\
a\big(a\langle a\rangle bcdd\big), S\big(aa\langle bc\rangle dd\big), d\big(aabc\langle d\rangle d\big) \vdash & S\big(a\langle abcd\rangle d\big) & (S \to aSd)\\
a\big(\varepsilon\langle a\rangle abcdd\big), S\big(a\langle abcd\rangle d\big), d\big(aabcd\langle d\rangle\varepsilon\big) \vdash & S\big(\varepsilon\langle aabcdd\rangle\varepsilon\big) & (S \to aSd)
\end{array}$$

The tree corresponding to this deduction is given in Figure 7.1(*a*); this tree represents a parse of the string according to the grammar.

7.2 Example of transformation

Let $G = (\Sigma, N, R, S, \mathcal{A})$ be a grammar with regular contexts. The goal is to construct an ordinary grammar $G' = (\Sigma, N', R', S')$ with $L(G') = L(G)$.

In the original grammar with regular contexts, parse trees of strings are augmented with runs of automaton $\mathcal{A}$, as shown in Figure 7.1(*a*). Strings defined by the new grammar have standard parse trees, and those trees shall have the same structure as the corresponding trees defined by the old grammar. In order to respect context operators, nonterminals of the new grammar shall be labelled with special marks referring to the behaviour of the automaton $\mathcal{A}$ on certain substrings.

Let us now consider how a grammar with regular *left* contexts can be transformed to an ordinary grammar. Consider a rule with a context operator $\lhd①$, with $① \in \mathcal{Q}$, which defines a string $u\langle w\rangle v$, with u, w, $v \in \Sigma$. On the string u, which is the left context of w, the automaton starts its computation in the state ①. If $\delta_u(①)$ is in $\mathcal{F}$, then the left context is considered satisfied. In the new grammar, every nonterminal symbol shall have a subscript that encodes the behaviour of the automaton on the left context of the current string.

Example 7.2. Consider the grammar from Example 7.1 and construct an ordinary grammar $G' = (\Sigma, N', R', S')$ that defines the same language.

For the start symbol S, the new grammar has a symbol $S_{①\to①,②\to②}$,

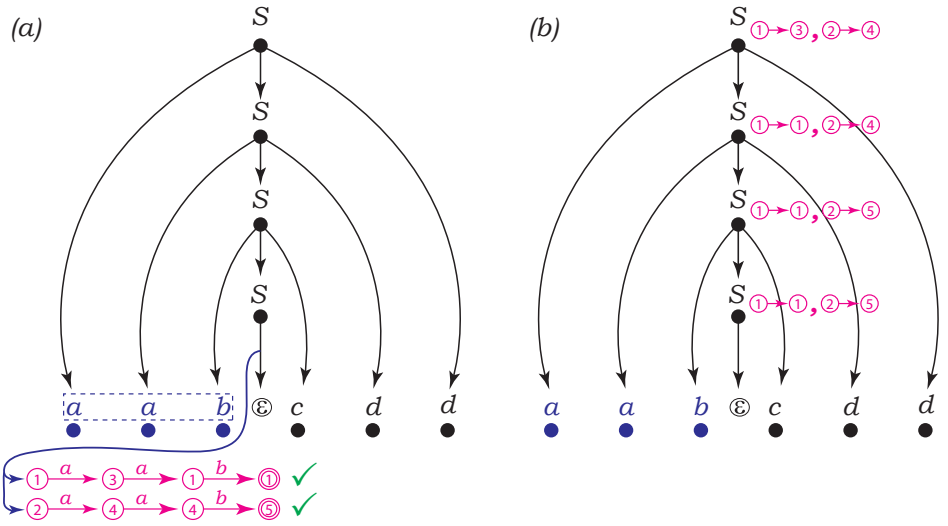


Figure 7.1: (a) A parse tree for the grammar in Example 7.1; (b) the corresponding tree for the transformed grammar in Example 7.2.

where the subscript represents the behaviour of the automaton on the empty string (which is, of course, the *identity function*). Thus, the subscript of the nonterminal represents the fact that the left context of S is the empty string.

The new grammar has the following rule with the start symbol on the left-hand side, produced out of the rule $S \to aSd$ in the old grammar.

$$S_{①\to①,②\to②} \to a\, S_{①\to③,②\to④}\, d \tag{7.2a}$$

In the right-hand side of this rule, the subscript of $S_{①\to③,②\to④}$ should represent the behaviour of the automaton on the left context up to the symbol a (including the symbol a as well). On the substring preceding a, this behaviour is the identity function (as stored in $S_{①\to①,②\to②}$), and its composition with the transition function δ_a yields the function stored in the new nonterminal.

One could also have another rule for the start symbol, originating from the rule $S \to bSc$.

$$S_{①\to①,②\to②} \to b\, S_{①\to①}\, c$$

However, as the function in the subscript of $S_{①\to①}$ is undefined on state ②, the context operator $\lhd②$ will never be satisfied. The nonterminal $S_{①\to①}$ would accordingly generate the empty language, and hence it can be omitted in the present construction, along with the above rule.

Another rule for S in the original grammar is $S \to \varepsilon \,\&\, \lhd① \,\&\, \lhd②$. This rule cannot be implemented for nonterminal $S_{①\to①,②\to②}$ in the new grammar, because state ② is mapped to a rejecting state ②, and thus the context operator is not satisfied.

For the nonterminal $S_{①\to③,②\to④}$ in the rule (7.2a), the new grammar shall have two rules corresponding to the automaton's transitions by a and by b.

$$S_{①\to③,②\to④} \quad\to\quad a\, S_{①\to①,②\to④}\, d \tag{7.2b}$$
$$S_{①\to③,②\to④} \quad\to\quad b\, S_{①\to③,②\to⑤}\, c \tag{7.2c}$$

In each case, the behaviour on the accumulated left context in the subscript of $S_{①\to③,②\to④}$ is composed with the automaton's behaviour on the corresponding symbol, resulting in $S_{①\to①,②\to④}$ for a and in $S_{①\to③,②\to⑤}$ for b.

The grammar has the following rules for these nonterminals.

$$S_{①\to①,②\to④} \quad\to\quad a\, S_{①\to③,②\to④}\, d \tag{7.2d}$$
$$S_{①\to①,②\to④} \quad\to\quad b\, S_{①\to①,②\to⑤}\, d \tag{7.2e}$$
$$S_{①\to③,②\to⑤} \quad\to\quad b\, S_{①\to③,②\to⑤}\, c \tag{7.2f}$$

In the first two rules, the function $① \to ①, ② \to ④$ is composed with δ_a and δ_b, and it yields the functions $① \to ③, ② \to ④$ and $① \to ①, ② \to ⑤$, respectively.

The third rule implements the rule $S \to bSc$ for $S_{①\to③,②\to⑤}$: the function $① \to ③, ② \to ⑤$ is composed with δ_b and gives the same function. The rule $S \to aSd$ could be implemented for $S_{①\to③,②\to⑤}$ as well, but it would not ultimately generate any string, and is therefore omitted.

The rule $S \to \varepsilon \,\&\, {\lhd①} \,\&\, {\lhd②}$ from the original grammar can be implemented for the nonterminal $S_{①\to①,②\to⑤}$, because the function in its subscript maps state ① to the accepting state ① and state ② to the accepting state ⑤. The left contexts are thus verified, and the context operators $\lhd①$ and $\lhd②$ are removed from the grammar.

$$S_{①\to①,②\to⑤} \to \varepsilon \tag{7.2g}$$

Thus, the construction results in an ordinary grammar with five nonterminal symbols generating the same language.

7.3 Elimination of regular context operators

When a grammar uses both left and right context operators, one has to store functions (say, f and h) representing the behaviour of the automaton on the contexts, as well as one more function (say, g), corresponding to the automaton's behaviour on the current substring. The behaviour of the automaton on the extended left context is then expressed as a composition of functions f and g, while the composition of g and h gives the automaton's behaviour on the extended right context.

Consider a grammar with regular contexts $G = (\Sigma, N, R, S, \mathcal{A})$, and assume, for the sake of simplicity, that its rules are of the following forms.

$$A \;\to\; BC \tag{7.3a}$$

$$A \;\to\; B \,\&\, \lhd p_1 \,\&\, \ldots \,\&\, \lhd p_m \,\&\, \trianglelefteq q_1 \,\&\, \ldots \,\&\, \trianglelefteq q_n \,\&\, \trianglerighteq r_1 \,\&\, \ldots \,\&\, \trianglerighteq r_{m'} \,\&\, \rhd s_1 \,\&\, \ldots \,\&\, \rhd s_{n'} \tag{7.3b}$$

$$A \;\to\; a \tag{7.3c}$$

$$A \;\to\; \varepsilon \tag{7.3d}$$

Then construct an ordinary grammar $G' = (\Sigma, N', R', S')$, with the set of nonterminals $N' = \{\, A_{f,\,g,\,h} \mid A \in N,\ f, g, h : \mathcal{Q} \to \mathcal{Q} \,\} \cup \{S'\}$ and with the following set of rules R'.

- For a rule of the form (7.3a) in R, and for all functions f, g, g', $h\colon \mathcal{Q} \to \mathcal{Q}$, add the following rules to R'.

$$A_{f,\,g'\circ g,\,h} \to B_{f,\,g,\,h\circ g'}\, C_{g\circ f,\,g',\,h} \tag{7.4a}$$

- For a rule of the form (7.3b) in R, and for all functions f, g, $h : \mathcal{Q} \to \mathcal{Q}$, if it holds that $f(p_i)$, $(g \circ f)(q_i)$, $(h \circ g)(r_i)$, $h(s_i) \in \mathcal{F}$, then add the following rules to R'.

$$A_{f,\,g,\,h} \to B_{f,\,g,\,h} \tag{7.4b}$$

- For every rule of the form (7.3c) in R, and for all functions $f, h : \mathcal{Q} \to \mathcal{Q}$, the following rules are added to the set R'.

$$A_{f,\,\delta_a,\,h} \to a \tag{7.4c}$$

- For every rule of the form (7.3d) in R, and for all functions $f, h\colon \mathcal{Q} \to \mathcal{Q}$, the following rules are added to the set R'.

$$A_{f,\,\mathrm{id},\,h} \to \varepsilon \tag{7.4d}$$

- The new grammar has the following rules for the initial symbol.

$$S' \to S_{\mathrm{id},\,g,\,\mathrm{id}} \qquad \text{(for all } g\colon \mathcal{Q} \to \mathcal{Q}\text{)}$$

The constructed ordinary grammar G' defines the same language as the original grammar with regular contexts G, as claimed below.

Lemma 7.1. *If $u\langle w\rangle v \in L_G(A)$, then $w \in L_{G'}(A_{\delta_u,\,\delta_w,\,\delta_v})$.*

Proof. The proof is by induction on t, the number of steps used in the derivation of the proposition $A\big(u\langle w\rangle v\big)$ in the grammar G.

Basis. Let $t = 1$ and consider the rule used to deduce the proposition $A\big(u\langle w\rangle v\big)$, with $u, w, v \in \Sigma^*$. If the rule is of the form $A \to a \in R$, then the deduction takes the following form.

$$a\big(u\langle a\rangle v\big) \vdash_G A\big(u\langle a\rangle v\big)$$

By construction, G' should contain a rule of the form $A_{f,\,\delta_a,\,h} \to a$, with $f = \delta_u$ and $h = \delta_v$, which can be used to deduce the proposition $A_{\delta_u,\,\delta_a,\,\delta_v}(a)$.

$$a(a) \vdash_{G'} A_{\delta_u,\,\delta_a,\,\delta_v}(a)$$

If the rule used to deduce the proposition $A\big(u\langle w\rangle v\big)$ is $A \to \varepsilon \in R$, then $w = \varepsilon$ and, accordingly, $\delta_w = \mathrm{id}$. The deduction of the proposition $A\big(u\langle\varepsilon\rangle v\big)$ is as follows.

$$\vdash_G A\big(u\langle\varepsilon\rangle v\big)$$

By construction, the new grammar G' should have a rule $A_{f,\,\mathrm{id},\,h} \to \varepsilon$ with $f = \delta_u$ and $h = \delta_v$. Then the desired proposition $A_{\delta_u,\,\mathrm{id},\,\delta_v}(\varepsilon)$ can be deduced in G'.

$$\vdash_{G'} A_{\delta_u,\,\mathrm{id},\,\delta_v}(\varepsilon)$$

Induction step. Let $t > 1$ and consider the rule, used at the last step of deduction of the proposition $A\big(u\langle w\rangle v\big)$ in the grammar G.

- Let the rule be of the form (7.3a), that is, the deduction took the following form.

  $$B\big(u\langle w_1\rangle w_2 v\big), C\big(uw_1\langle w_2\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big) \quad (w = w_1w_2, w_1, w_2 \in \Sigma^*)$$

  By construction, the new grammar has a rule (7.4a) with $f = \delta_u$, $g = \delta_{w_1}$, $g' = \delta_{w_2}$, and $h = \delta_v$. Note, that $\delta_v \circ \delta_{w_2} = \delta_{w_2 v}$ and $\delta_{w_1} \circ \delta_u = \delta_{uw_1}$, and therefore the rule has the following form.

  $$A_{\delta_u,\,\delta_{w_2}\circ\delta_{w_1},\,\delta_v} \to B_{\delta_u,\,\delta_{w_1},\,\delta_{w_2 v}} C_{\delta_{uw_1},\,\delta_{w_2},\,\delta_v}$$

  Since the derivation of the proposition $B\big(u\langle w_1\rangle w_2 v\big)$ in G is shorter than t, by the induction hypothesis, $w_1 \in L_{G'}(B_{\delta_u,\,\delta_{w_1},\,\delta_{w_2 v}})$. Similarly, $w_2 \in L_{G'}(C_{\delta_{uw_1},\,\delta_{w_2},\,\delta_v})$. Now the proposition $A_{\delta_u,\,\delta_{w_2}\circ\delta_{w_1},\,\delta_v}$ can be deduced in the grammar G' by that new rule.

  $$B_{\delta_u,\,\delta_{w_1},\,\delta_{w_2 v}}(w_1), C_{\delta_{uw_1},\,\delta_{w_2},\,\delta_v}(w_2) \vdash_{G'} A_{\delta_u,\,\delta_{w_2}\circ\delta_{w_1},\,\delta_v}(w_1w_2)$$

- Let the last step of the deduction of the proposition $A\big(u\langle w\rangle v\big)$ use the rule of the form (7.3b), for some states p_i, q_i, r_i, s_i representing regular contexts.

$$B\big(u\langle w\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$$

For this rule to be applicable, the states p_i, q_i, r_i and s_i should satisfy the following conditions.

$$\delta_u(p_i), \delta_{uw}(q_i), \delta_{wv}(r_i), \delta_v(s_i) \in \mathcal{F}$$

Let $f = \delta_u$, $g = \delta_w$ and $h = \delta_v$; the above conditions are then written down as $f(p_i)$, $(g \circ f)(q_i)$, $(h \circ g)(r_i)$, $h(s_i) \in \mathcal{F}$. Then, by construction, the new grammar has a rule (7.4b) corresponding to the original rule (7.3b) and the functions f, g, h.

$$A_{\delta_u,\, \delta_w,\, \delta_v} \to B_{\delta_u,\, \delta_w,\, \delta_v}$$

By the induction hypothesis, the string w is in $L_{G'}(B_{\delta_u,\, \delta_w,\, \delta_v})$. The proposition $A_{\delta_u,\, \delta_w,\, \delta_v}(w)$ can then be deduced in the new grammar by the above rule.

$$B_{\delta_u,\, \delta_w,\, \delta_v}(w) \vdash_{G'} A_{\delta_u,\, \delta_w,\, \delta_v}(w)$$

□

Conversely, the following statement holds.

Lemma 7.2. *If $w \in L_{G'}(A_{f,\, g,\, h})$, then:*

1. *$\delta_w = g$, and*
2. *for every string $u \in \Sigma^*$ with $\delta_u = f$ and for every string $v \in \Sigma^*$ with $\delta_v = h$, it holds that $u\langle w\rangle v \in L_G(A)$.*

Proof. The proof is by induction on t, the number of steps in the derivation of the proposition $A_{f,\, g,\, h}(w)$ in the grammar G'.

Basis. Let $t = 1$, then the rule used to derive the proposition $A_{f,\, g,\, h}(w)$ is either of the form (7.4c) or (7.4d).

In the former case, $w = a$, $g = \delta_a$ and the deduction of the proposition $A_{f,\, \delta_a,\, h}(a)$ takes the following form.

$$a(a) \vdash_{G'} A_{f,\, g,\, h}(a)$$

Since the rule $A_{f,\, g,\, h} \to a$ is in R', the grammar G should have a rule $A \to a$ by construction. Then the proposition $A\big(u\langle a\rangle v\big)$ can be deduced in G by this rule.

$$a\big(u\langle a\rangle v\big) \vdash_G A\big(u\langle a\rangle v\big)$$

In the latter case, $w = \varepsilon$. Then $g = \mathrm{id}$ and the proposition $A_{f,\,\mathrm{id},\,h}(\varepsilon)$ can be deduced in the grammar G' out of an axiom. By construction, the grammar G has a rule $A \to \varepsilon$, by which one can deduce the proposition $A\big(u\langle\varepsilon\rangle v\big)$.

$$\varepsilon\big(u\langle\varepsilon\rangle v\big) \vdash_G A\big(u\langle\varepsilon\rangle v\big)$$

Induction step. Let $t > 1$ and consider the rule used at the last step of derivation of the proposition $A_{f,\,g,\,h}(w)$ in the grammar G'.

- Let the rule be of the form (7.4a), constructed for some rule $A \to BC$ in G. In order to derive the proposition $A_{f,\,g,\,h}(w)$, the rule must be constructed for the functions f and h and for any partition of g as $g = g_2 \circ g_1$, where $g_1,\ g_2,\ h\colon \mathcal{Q} \to \mathcal{Q}$. Then the last step of deduction takes the following form.

$$B_{f,\,g_1,\,h\circ g_2}(w_1), C_{g_1\circ f,\,g_2,\,h}(w_2) \vdash_{G'} A_{f,\,g_2\circ g_1,\,h}(w_1 w_2) \qquad (w = w_1 w_2, w_1, w_2 \in \Sigma^*)$$

  By the induction hypothesis (part 1) applied to the proposition $B_{f,\,g_1,\,h\circ g_2}(w_1)$, it holds that $g_1 = \delta_{w_1}$. Similarly, by applying the induction hypothesis (part 1) to the proposition $C_{g_1\circ f,\,g_2,\,h}(w_2)$, one obtains that $g_2 = \delta_{w_2}$. Then, $g_2 \circ g_1 = \delta_{w_2} \circ \delta_{w_1} = \delta_{w_1 w_2}$, which proves part 1 of the claim.

  Turning to part 2, consider any two strings $u,\ v \in \Sigma^*$, such that $f = \delta_u$ and $h = \delta_v$. Then, $h \circ g_2 = \delta_v \circ \delta_{w_2} = \delta_{w_2 v}$, and therefore $w_1 \in L_{G'}(B_{\delta_u,\,\delta_{w_1},\,\delta_{w_2 v}})$. Since δ_u and $\delta_{w_2 v}$ are transition functions of the left context of w_1 and on its right context, respectively, the induction hypothesis (part 2) applies, and it asserts that $u\langle w_1\rangle w_2 v$ is in $L_G(B)$. Similarly, $g_1 \circ f = \delta_{w_1} \circ \delta_u = \delta_{u w_1}$ and $w_2 \in L_{G'}(C_{\delta_{u w_1},\,\delta_{w_2},\,\delta_v})$. Again, $\delta_{u w_1}$ and δ_v are transition functions on the contexts, and thus, by the induction hypothesis (part 2), $u w_1\langle w_2\rangle v$ is in $L_G(C)$.

  Since the grammar G has a rule $A \to BC$, the proposition $A\big(u\langle w\rangle v\big)$ can be deduced as follows.

$$B\big(u\langle w_1\rangle w_2 v\big), C\big(u w_1\langle w_2\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$$

- Let the rule be of the form (7.4b). By construction, this rule is obtained from the rule of the form (7.3b), for functions $f,\ g,\ h\colon \mathcal{Q} \to \mathcal{Q}$, such that $f(p_i)$, $(g \circ f)(q_i)$, $(h \circ g)(r_i)$, and $h(s_i) \in \mathcal{F}$.

  The deduction of the proposition $A_{f,\,g,\,h}(w)$ in G' takes the following form.

$$B_{f,\,g,\,h}(w) \vdash_{G'} A_{f,\,g,\,h}(A)$$

By the induction hypothesis (part 1) for $B_{f,\,g,\,h}(w)$, the function g coincides with δ_w, and this proves part 1 for $A_{f,\,g,\,h}(w)$.

In order to prove part 2, consider some strings u and $v \in \Sigma^*$, for which $\delta_u = f$ and $\delta_v = h$. By the induction hypothesis (part 2), $\vdash_G B\big(u\langle w\rangle v\big)$. Now, since all $f(p_i)$, $(g \circ f)(q_i)$, $(h \circ g)(r_i)$, and $h(s_i)$ belong to $\mathcal{F}$, the desired proposition $A\big(u\langle w\rangle v\big)$ can be deduced in G by the rule (7.3b).

$$B\big(u\langle w\rangle v\big) \vdash_G A\big(u\langle w\rangle v\big)$$

□

7.4 Construction for an arbitrary grammar

Let us now give a general version of the construction that does not require any pre-processing of rules of a grammar.

Let $G = (\Sigma, N, R, S, \mathcal{A})$ be a grammar with regular contexts, where $\mathcal{A} = (\Sigma, \mathcal{Q}, \delta, \mathcal{F})$ is a finite automaton.

For $\alpha = x_0 B^1 x_1 B^2 \dots x_{\ell-1} B^\ell x_\ell$, with $x_i \in \Sigma^*$ and $B^i \in N$, and for functions $g_1, \dots, g_\ell : \mathcal{Q} \to \mathcal{Q}$, define $\mathbf{g}(\alpha)$ as $\mathbf{g}(\alpha) = \delta_{x_\ell} \circ g_\ell \circ \delta_{x_{\ell-1}} \circ \dots \circ g_1 \circ \delta_{x_0}$.

Construct an ordinary grammar $G' = (\Sigma, N', R', S')$ as follows.

The goal is that for nonterminal A, the corresponding nonterminal in the new grammar is $A_{f,\,\mathbf{g}(X_1 \dots X_\ell),\,h}$, where $\mathbf{g}$ is a composition of the functions $g_1, \dots, g_\ell$ stored in the symbols $X_1, \dots, X_\ell$. If $X_i = a \in \Sigma$ is a terminal symbol, then $g_i = \delta_a$, and in the case of the empty string, g_i is the identity function.

For every rule of the form

$$A \to x_0 B^1 x_1 B^2 \dots x_{\ell-1} B^\ell x_\ell \,\&\lhd p_1 \,\& \dots \,\& \lhd p_m \,\& \trianglelefteq q_1 \,\& \dots \,\& \trianglelefteq q_n \,\& \\ \&\trianglerighteq r_1 \,\& \dots \,\& \trianglerighteq r_{m'} \,\& \rhd s_1 \,\& \dots \,\& \rhd s_{n'}$$

in R, and for all functions f, g_1, …, g_ℓ, $h : \mathcal{Q} \to \mathcal{Q}$, if $f(p_i)$, $(\mathbf{g}(x_0 B^1 x_1 B^2 \dots x_{\ell-1} B^\ell x_\ell) \circ f)(q_i)$, $(h \circ \mathbf{g}(x_0 B^1 x_1 B^2 \dots x_{\ell-1} B^\ell x_\ell))(r_i)$, $h(s_i) \in \mathcal{F}$, then the set R' contains the following rules.

$$A_{f,\,\mathbf{g}(x_0 B^1 x_1 B^2 \dots x_{\ell-1} B^\ell x_\ell),\,h} \to x_0\, B^1_{\delta_{x_0} \circ f,\; g_1,\; \mathbf{g}(x_1 B^2 \dots B^\ell x_\ell) \circ h}\, x_1 \\ B^2_{\delta_{x_1} \circ g_1 \circ \delta_{x_0} \circ f,\; g_2,\; \mathbf{g}(x_2 B^3 \dots B^\ell x_\ell) \circ h} \cdots B^\ell_{\delta_{x_{\ell-1}} \circ g_{\ell-1} \circ \dots \circ \delta_{x_1} \circ g_1 \circ \delta_{x_0} \circ f,\; g_\ell,\; \delta_{x_\ell} \circ h}\, x_\ell$$

The correctness of this construction can be stated verbatim as in Lemmata 7.1 and 7.2, and the proofs proceed in a similar way.

7.5 Conjunctive grammars with regular contexts

Consider the following conjunctive grammar and construct an equivalent grammar with regular contexts with a smaller number of nonterminals.

Example 7.3. The following grammar defines the language $\{ a^n b^n c^n \mid n \geqslant 0 \}$, represented as an intersection of two context-free languages $\{ a^i b^j c^j \mid i, j \geqslant 0 \}$ and $\{ a^k b^k c^\ell \mid k, \ell \geqslant 0 \}$.

$$\begin{aligned} S &\to AB \,\&\, DC \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \\ D &\to aDb \mid \varepsilon \\ C &\to cC \mid \varepsilon \end{aligned}$$

This language can be defined by the following *conjunctive grammar with regular contexts*.

$$\begin{aligned} S &\to X \,\&\, Y \\ X &\to aX \,\&\, \lhd a^* \mid bXc \,\&\, \lhd a^* b^* \mid \varepsilon \\ Y &\to aYb \,\&\, \lhd a^* \mid Yc \,\&\, \lhd a^+ b^+ c^* \mid \varepsilon \end{aligned}$$

In this grammar, concatenations AB and DC in the first rule for nonterminal S are represented by symbols X and Y, respectively.

The first rule for nonterminal X is only applicable when the current accumulated left context of the string is a^*, and then the control is passed to the rule $X \to bXc \,\&\, \lhd a^* b^*$, which can be applied as far as the left context is still satisfied. This rule appends symbols b, thus making its left context be of the form $a^* b^+$, and the rule $X \to aX \,\&\, \lhd a^*$ is not applicable anymore.

Similarly, the first rule for nonterminal Y adds symbols a and b, and then at some point the control is given to the second rule for Y, which appends symbols c. Again, the order in which these rules can be applied is restricted by the regular context operators.

For an arbitrary conjunctive grammar with regular contexts, the goal is to construct a conjunctive grammar which would define the same language as the original grammar. Such a transformation again proceeds similarly to the previous cases, with the only difference in the way how rules with multiple base conjuncts are processed. As an elementary case, consider a rule $A \to B \,\&\, C$; for this rule and for all functions f, g_1, g_2, $h : \mathcal{Q} \to \mathcal{Q}$, the new grammar shall have rules $A_{f,\, g_1,\, h} \to B_{f,\, g_1,\, h} \,\&\, C_{f,\, g_2,\, h}$, as long as the

functions representing the current substring in both conjuncts coincide, that is, $g_1 = g_2$. The general construction is given below.

Let $G = (\Sigma, N, R, S, \mathcal{A})$ be a conjunctive grammar with regular contexts, where $\mathcal{A} = (\Sigma, \mathcal{Q}, \delta, \mathcal{F})$ is a finite automaton.

Construct a conjunctive grammar $G' = (\Sigma, N', R', S')$, with the set of rules defined as follows.

For a rule of the form

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd p_1 \,\&\, \ldots \,\&\, \lhd p_m \,\&\, \trianglelefteq q_1 \,\&\, \ldots \,\&\, \trianglelefteq q_n \,\&\, \trianglerighteq r_1 \,\&\, \ldots \,\&\, \trianglerighteq r_{m'} \,\&\, \rhd s_1 \,\&\, \ldots \,\&\, \rhd s_{n'}$$

in R, with

$$\alpha_i = x_{i,0} B^{i,1} x_{i,1} \ldots x_{i,\ell_i - 1} B^{i,\ell_i} x_{i,\ell_i},$$

and for all functions f, $g_{1,1}$, …, g_{k,ℓ_k}, $h\colon \mathcal{Q} \to \mathcal{Q}$, if

$$g_{1,\ell_1} \circ \ldots \circ g_{1,2} \circ g_{1,1} = \ldots = g_{k,\ell_k} \circ \ldots \circ g_{k,2} \circ g_{k,1}$$

and

$$f(p_i), \qquad (\mathbf{g}(x_{1,0} B^{1,1} x_{1,1} B^{1,2} \ldots x_{1,\ell_1 - 1} B^{1,\ell_1} x_{1,\ell_1}) \circ f)(q_i),$$
$$(h \circ \mathbf{g}(x_{1,0} B^{1,1} x_{1,1} B^{1,2} \ldots x_{1,\ell_1 - 1} B^{1,\ell_1} x_{1,\ell_1}))(r_i), \qquad h(s_i) \in \mathcal{F},$$

then add to the new grammar G' the rules

$$A_{f,\, \mathbf{g}(x_{1,0} B^{1,1} x_{1,1} B^{1,2} \ldots x_{1,\ell_1 - 1} B^{1,\ell_1} x_{1,\ell_1}),\, h} \to \alpha'_1 \,\&\, \ldots \,\&\, \alpha'_k,$$

with

$$\alpha'_i = x_{i,0}\, B^{i,1}_{\delta_{x_{i,0}} \circ f,\, g_{i,1},\, \mathbf{g}(x_{i,1} B^{i,2} \ldots B^{i,\ell_i} x_{i,\ell_i}) \circ h}\, x_{i,1}$$
$$B^{i,2}_{\delta_{x_{i,1}} \circ g_{i,1} \circ \delta_{x_{i,0}} \circ f,\, g_{i,2},\, \mathbf{g}(x_{i,2} B^{i,3} \ldots B^{i,\ell_i} x_{i,\ell_i}) \circ h} \cdots$$
$$\cdots\, B^{i,\ell_i}_{\delta_{x_{i,\ell_i - 1}} \circ g_{i,\ell_i - 1} \circ \ldots \circ \delta_{x_{i,1}} \circ g_{i,1} \circ \delta_{x_{i,0}} \circ f,\, g_{i,\ell_i},\, \delta_{x_{i,\ell_i}} \circ h}\, x_{i,\ell_i},$$

and the function $\mathbf{g}$ defined as in Section 7.4.

Altogether, the following theorem is established.

Theorem 7.1. *Let G be an ordinary regular, linear context-free, conjunctive, linear conjunctive grammar augmented with regular contexts. Then there exists and can be effectively constructed a grammar of the same type without any context operators that describes the same language.*

Chapter 8

Linear grammars with contexts

This chapter studies the *linear subclass* of grammars with one-sided contexts, where linearity is understood as a restriction to concatenate nonterminal symbols only to terminal strings.

A similar family of languages, defined by *linear conjunctive grammars*, which allow using the conjunction operation, but no context specifications, was found to be computationally equivalent to *one-way real-time cellular automata* [14, 87], also known under a proper name of *trellis automata* [12, 27]. This chapter sets off by developing an analogous automaton representation for linear grammars with one-sided contexts. The proposed *trellis automata with feedback* augment the original cellular automaton model by an extra communication channel. Its motivation comes from the understanding of those automata as circuits with uniform connections [12], to which one can add a new type of connection. As the model is proved to be equivalent to linear grammars with one-sided contexts, it follows that this new type of connections has exactly the same power as context specifications do in grammars.

The main result is a method for simulating a Turing machine by a trellis automaton with feedback processing an input string over a *one-symbol alphabet*. This method subsequently allows uniform undecidability proofs for linear grammars with contexts, which parallel the recent results for conjunctive grammars due to Jeż [31] and Jeż and Okhotin [32, 33, 34], but are based upon an entirely different underlying construction.

Section 8.3 gives a simple example of a 3-state trellis automaton with feedback, which recognizes the language $\{ a^{2^k-2} \mid k \geqslant 2 \}$. Recall that, standard trellis automata over a one-symbol alphabet recognize only regular languages [12].

The next Section 8.4 presents a simulation of a Turing machine by a trellis automaton with feedback, so that the latter automaton, given an input a^n, simulates several steps of the Turing machine's computation on an empty

input, and accordingly can accept or reject the input a^n depending on the current state of the Turing machine. This result is then used in Section 8.5 to prove the undecidability of the emptiness problem for linear grammars with one-sided contexts over a one-symbol alphabet. The finiteness problem for these grammars is proved to be complete for the second level of the arithmetical hierarchy.

Some further examples of linear grammars with contexts, which describe the languages $\{\, a^n b^{in} \mid i, n \geqslant 1 \,\}$ and $\{\, a^{n^2} \mid n \geqslant 0 \,\}$, are presented in Section 8.6.

Finally, the last Section 8.7 establishes some closure properties of the linear grammars with contexts. Here the automaton representation becomes particularly useful, as it gives an immediate proof of the closure of this language family under complementation, which, using grammars alone, would require a complicated construction.

Example 8.1. The following grammar defines the singleton language $\{abac\}$.

$$\begin{array}{lcl} S & \to & aBc \\ B & \to & bA \,\&\, \lhd A \\ A & \to & a \end{array}$$

The derivation of the string $abac$ begins by deriving the inner substring ba from B. First, the rule $A \to a$ defines the symbol a in the empty context and in the context ab.

$$\begin{array}{rll} \vdash a\big(\varepsilon\langle a\rangle\big) & & (axiom) \\ \vdash a\big(ab\langle a\rangle\big) & & (axiom) \\ a\big(\varepsilon\langle a\rangle\big) \vdash A\big(\varepsilon\langle a\rangle\big) & & (A \to a) \\ a\big(ab\langle a\rangle\big) \vdash A\big(ab\langle a\rangle\big) & & (A \to a) \end{array}$$

Then the rule $B \to bA \,\&\, \lhd A$ defines the substring $a\langle ba\rangle$ by concatenating b to a one-symbol string $ab\langle a\rangle$ derived from A. Furthermore, this rule requires the left context of $a\langle ba\rangle$ to be described by A, and this has also been derived above.

$$\begin{array}{rll} \vdash b\big(a\langle b\rangle\big) & & (axiom) \\ b\big(a\langle b\rangle\big), A\big(ab\langle a\rangle\big), A\big(\varepsilon\langle a\rangle\big) \vdash B\big(a\langle ba\rangle\big) & & (B \to bA \,\&\, \lhd A) \end{array}$$

Finally, the rule $S \to aBc$ concatenates the substrings a, ba and c to obtain the desired string $abac$.

$$\begin{aligned} &\vdash c\big(aba\langle c\rangle\big) && (axiom)\\ a\big(\varepsilon\langle a\rangle\big), B\big(a\langle ba\rangle\big), c\big(aba\langle c\rangle\big) &\vdash S\big(\varepsilon\langle abac\rangle\big) && (S \to aBc) \end{aligned}$$

This derivation of the string $w = abac$ is illustrated in Figure 8.1, where elements of the triangle correspond to non-empty substrings of w, and each element contains all nonterminal symbols generating the corresponding substring. The dotted line shows the effect of a context operator ($\lhd A$), whereas the band rising up from the second A through B to S demonstrates the rules appending symbols to substrings.

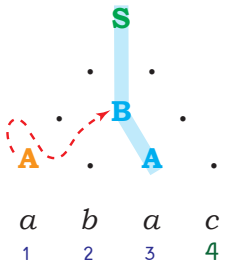


Figure 8.1: Derivation of the string $abac$ in Example 8.1, informally illustrated.

The following example is an elaborated version of Example 2.4 in Section 2.3.1, which describes an abstract language representing *declarations of objects before their use*.

Example 8.2. Consider the grammar in Example 2.4, which is not linear as it is, and let us show how it can be converted to one.

The first non-linear rule $S \to DS$ expresses a concatenation of d^*a with S; in a linear grammar, this concatenation can be defined by a new nonterminal D_S with the rules $D_S \to dD_S$ and $D_S \to aS$. In a similar way, one can express concatenations CE and DE in the rules for E. Thus, for instance, a new symbol C_E defining concatenation CE will have rules $C_E \to cC_E$ and $C_E \to aE$.

The rules $S \to US$ and $U \to C \,\&\, \leqslant EFa$ concatenate a string from c^*a to S, and also apply a context operator to this string. In a linear grammar, this is done by a new nonterminal U_S, which has two rules, $U_S \to cU_S$ and $U_S \to aS \,\&\, \lhd E_F$. These rules simulate concatenation of a string from c^*a to S, and the latter rule applies a context operator, where a new symbol E_F simulates the concatenation EF. Unlike in the original grammar, the context specification in this rule is proper, since it is enough to match symbols c

(defined by the rule $U_S \to cU_S$), without taking into account the last a of a block. One can transform the rest of the rules in a similar way.

Below a complete linear grammar is given, with nonterminals renamed for better readability.

$$\begin{aligned}
S &\to D \mid U \mid \varepsilon \\
D &\to dD \mid aS \\
C &\to cC \mid aS \\
U &\to cU \mid aS \mathbin{\&} {\lhd}H \\
E &\to P \mid Q \mid \varepsilon \\
P &\to dP \mid aE \\
Q &\to cQ \mid aE \\
H &\to X \mid Y \mid F \\
X &\to dX \mid aH \\
Y &\to cY \mid aH \\
F &\to dFc \mid aE
\end{aligned}$$

8.1 Normal form for linear grammars with contexts

As has been shown in Chapter 4, every grammar with contexts can be transformed to a certain normal form, which extends the Chomsky normal form for ordinary context-free grammars. While the original Chomsky normal form has all rules of the form $A \to BC$ and $A \to a$, its extension for grammars with contexts allows using multiple conjuncts BC and context specifications ${\lhd}D$, ${\trianglelefteq}E$.

Recently, Okhotin [69] has shown that every grammar with left contexts can be transformed to a stronger normal form, where only proper context operators ${\lhd}D$ are allowed. A similar normal form shall now be established for the linear subclass of grammars.

Theorem 8.1. *For every linear grammar with left contexts $G = (\Sigma, N, R, S)$, there exists another linear grammar with left contexts $G' = (\Sigma, N', R', S)$ that defines the same language and has all rules of the form*

$$A \to bB_1 \mathbin{\&} \ldots \mathbin{\&} bB_\ell \mathbin{\&} C_1c \mathbin{\&} \ldots \mathbin{\&} C_kc \tag{8.1a}$$

$$A \to a \mathbin{\&} {\lhd}D_1 \mathbin{\&} \ldots \mathbin{\&} {\lhd}D_m, \tag{8.1b}$$

where $A, B_i, C_i, D_i \in N$, $a, b, c \in \Sigma$, $\ell + k \geqslant 1$ and $m \geqslant 0$.

The size of G' is at most triple exponential in the size of G, where the size is measured by the total number of symbols used in the description of the grammar.

The transformation is carried out along the same lines as in the general case. Given an arbitrary linear grammar with contexts $G = (\Sigma, N, R, S)$, the rules of the grammar are first preprocessed: long conjuncts are cut until all of them are of the form bB, Cc or a, and every context specification $\lhd\gamma$ or $\trianglelefteq\gamma$ with $\gamma \in \Sigma$ or $|\gamma| > 1$ is restated as $\lhd X_\gamma$ or $\trianglelefteq X_\gamma$, respectively, where X_γ is a new nonterminal with a unique rule $X_\gamma \to \gamma$.

This results in a grammar $G_1 = (\Sigma, N_1, R_1, S)$ with the rules of the following form:

$$A \to bB \tag{8.2a}$$
$$A \to Cc \tag{8.2b}$$
$$A \to a \tag{8.2c}$$
$$A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \tag{8.2d}$$
$$A \to \varepsilon, \tag{8.2e}$$

where $a, b, c \in \Sigma$ and $A, B, C, B_i, D_i, E_i \in N$.

The transformation continues with the elimination of *null conjuncts*, that is, any rules of the form $A \to \ldots \,\&\, \varepsilon$. First, one has to determine, which nonterminals generate the empty string, and in which contexts they generate it. This information is represented in a set $\textsc{Nullable}(G_1) \subseteq 2^N \times N$, as defined in Section 4.4.1. Given a nonterminal $A \in N$ and a string $u \in \Sigma^*$, the string $u\langle\varepsilon\rangle$ is defined by A if and only if the set $\textsc{Nullable}(G_1)$ contains a pair $(\{K_1, \ldots, K_t\}, A)$, with $A, K_1, \ldots, K_t \in N$, such that $\varepsilon\langle u\rangle \in L_{G_1}(K_1)$, $\ldots$, $\varepsilon\langle u\rangle \in L_{G_1}(K_t)$.

Using the set $\textsc{Nullable}(G_1)$, a new grammar $G_2 = (\Sigma, N_1, R_2, S)$ without null conjuncts can be constructed as follows.

1. The rules of the form (8.2a)–(8.2d) are copied to the new grammar.

2. For every rule of the form (8.2a) and for every pair $(\{K_1, \ldots, K_t\}, B) \in \textsc{Nullable}(G_1)$, a rule $A \to b \,\&\, \trianglelefteq K_1 \,\&\, \ldots \,\&\, \trianglelefteq K_t$ is added to the new grammar.

3. For every rule of the form (8.2b) and for every pair $(\{K_1, \ldots, K_t\}, C) \in \textsc{Nullable}(G_1)$, the new grammar has the rule $A \to c \,\&\, \lhd K_1 \,\&\, \ldots \,\&\, \lhd K_t$. Moreover, if $\varepsilon\langle\varepsilon\rangle \in L_{G_1}(K_i)$ for all $i \in \{1, \ldots, t\}$, then a rule $A \to c \,\&\, \lhd\varepsilon$ should be added.

4. For every rule of the form (8.2d), a rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, E_1 \,\&\, \ldots \,\&\, E_n \,\&\, \lhd\varepsilon$ shall be added to the new grammar, if $\varepsilon\langle\varepsilon\rangle \in L_{G_1}(D_i)$ for all $i \in \{1, \ldots, m\}$.

After this step, the rules of the grammar are of the following form.

$$
\begin{aligned}
A &\to a \\
A &\to bB \\
A &\to Cc \\
A &\to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n \\
A &\to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd \varepsilon \\
A &\to b \,\&\, \lhdeq K_1 \,\&\, \ldots \,\&\, \lhdeq K_t \\
A &\to c \,\&\, \lhd K_1 \,\&\, \ldots \,\&\, \lhd K_t \\
A &\to c \,\&\, \lhd \varepsilon
\end{aligned}
$$

Having constructed this grammar, one can apply verbatim the rest of the steps of the general transformation to the normal form, described in Chapter 4. This transformation constitues in elimination of *null contexts* $\lhd\varepsilon$ (described in Section 4.4.3) and of *unit conjuncts* (see Section 4.4.4) as in the rules $A \to \ldots \,\&\, B$. The final step is elimination of *extended left contexts* $\lhdeq E$, which are all expressed through proper left contexts $\lhd D$ [69]. Each step preserves the linearity of a grammar.

8.2 Automaton representation

Linear conjunctive grammars are known to be computationally equivalent to one of the simplest types of cellular automata: the *one-way real-time cellular automata*, also known under the proper name of *trellis automata*. This section presents a generalization of trellis automata, which similarly corresponds to linear grammars with one-sided contexts.

A standard trellis automaton, as introduced by Čulík, Gruska and Salomaa [12], processes an input string of length $n \geqslant 1$ using a uniform array of $\frac{n(n+1)}{2}$ nodes, as presented in Figure 8.2(left). Each node computes a value from a fixed finite set Q. The nodes in the bottom row obtain their values directly from the input symbols using a function $I\colon \Sigma \to Q$. The rest of the nodes compute the function $\delta\colon Q \times Q \to Q$ of the values in their predecessors. The string is accepted if and only if the value computed by the topmost node belongs to the set of accepting states $F \subseteq Q$.

Theorem A (Okhotin [57])**.** A language $L \subseteq \Sigma^+$ is defined by a linear conjunctive grammar if and only if L is recognized by a trellis automaton.

In terms of cellular automata, every horizontal row of states in Figure 8.2(left) represents an automaton's configuration at a certain moment of time. An alternative motivation developed in the literature on trellis automata [12, 14, 27] is to consider the entire grid as a digital circuit with

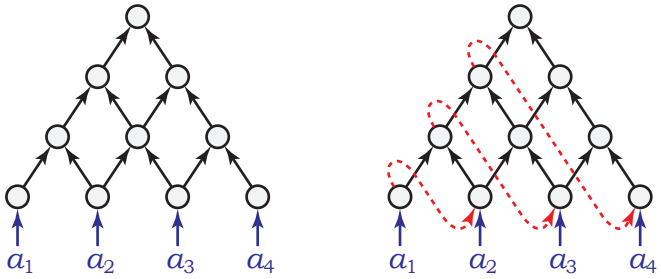


Figure 8.2: Trellis automata (left) and trellis automata with feedback (right).

uniform structure of connections. In order to obtain a similar representation of linear grammars with left contexts, the trellis automaton model is extended with another type of connection, illustrated in Figure 8.2(right).

Definition 8.1. *A trellis automaton with feedback is a sextuple* $M = (\Sigma, Q, I, J, \delta, F)$*, in which:*

- Σ *is the input alphabet;*
- Q *is a finite non-empty set of states;*
- $I\colon \Sigma \to Q$ *is a function that sets the initial state for the first symbol;*
- $J\colon Q \times \Sigma \to Q$ *sets the initial state for every subsequent symbol; using the state computed on the preceding substring as a feedback;*
- $\delta\colon Q \times Q \to Q$ *is the transition function;*
- $F \subseteq Q$ *is the set of accepting states.*

The behaviour of the automaton is described by a function $\Delta\colon \Sigma^* \times \Sigma^+ \to Q$*, which defines the state* $\Delta(u\langle v\rangle)$ *computed on each string* $u\langle v\rangle$*. The value of this function is defined by the following formulae formalizing the connections in Figure 8.2(right).*

$$\begin{aligned} \Delta(\varepsilon\langle a\rangle) &= I(a) \\ \Delta(w\langle a\rangle) &= J\big(\Delta(\varepsilon\langle w\rangle), a\big) && (w \neq \varepsilon) \\ \Delta(u\langle bvc\rangle) &= \delta\big(\Delta(u\langle bv\rangle), \Delta(ub\langle vc\rangle)\big) && (b, c \in \Sigma) \end{aligned}$$

The language recognized by the automaton is $L(M) = \{\, w \in \Sigma^+ \mid \Delta(\varepsilon\langle w\rangle) \in F \,\}$*.*

Theorem 8.2. *A language* $L \subseteq \Sigma^+$ *is defined by a linear grammar with left contexts if and only if* L *is recognized by a trellis automaton with feedback.*

The proof is by effective constructions in both directions.

Lemma 8.1. *Let $G = (\Sigma, N, R, S)$ be a linear grammar with left contexts, in which every rule is of the form*

$$A \to bB_1 \,\&\, \ldots \,\&\, bB_\ell \,\&\, C_1 c \,\&\, \ldots \,\&\, C_k c \quad (b, c \in \Sigma,\ A, B_i, C_i \in N, \ell + k \geqslant 1) \tag{8.4a}$$

$$A \to a \,\&\, {\lhd}D_1 \,\&\, \ldots \,\&\, {\lhd}D_m \qquad (a \in \Sigma,\ m \geqslant 0,\ A, D_i \in N) \tag{8.4b}$$

and define a trellis automaton with feedback $M = (\Sigma, Q, I, J, \delta, F)$ as follows.

$$\begin{aligned} Q &= \Sigma \times 2^N \times \Sigma \\ I(a) &= (a, \{\, A \mid A \to a \in R \,\}, a) \\ J\big((b, X, c), a\big) &= \big(a, \{\, A \mid \exists \textit{ rule (8.4b) with } D_1, \ldots, D_m \in X \,\}, a\big) \\ \delta\big((b, X, c'), (b', Y, c)\big) &= \big(b, \{\, A \mid \exists \textit{ rule (8.4a) with } C_i \in X \textit{ and } B_i \in Y \,\}, c\big) \\ F &= \big\{\, (b, X, c) \;\big|\; S \in X,\ b, c \in \Sigma \,\big\} \end{aligned}$$

For every string with context $u\langle v\rangle$, let b be the first symbol of v, let c be the last symbol of v, and let $Z = \{\, A \mid u\langle v\rangle \in L_G(A) \,\}$. Then $\Delta(u\langle v\rangle) = (b, Z, c)$.

In particular, $L(M) = \{\, w \mid \varepsilon\langle w\rangle \in L_G(S) \,\} = L(G)$.

Proof. Induction on pairs $(|uv|, |v|)$, ordered lexicographically.

Basis: $\varepsilon\langle a\rangle$ with $a \in \Sigma$. The state computed on this string is $\Delta(\varepsilon\langle a\rangle) = I(a) = (a, Z, a)$ with $Z = \{\, A \mid A \to a \in R \,\}$. The latter set Z is the set of all symbols $A \in N$ with $\varepsilon\langle a\rangle \in L_G(A)$.

Induction step I: $u\langle a\rangle$ with $u \in \Sigma^*$ and $a \in \Sigma$. The state computed by the automaton on the string $u\langle a\rangle$ is defined as $\Delta(u\langle a\rangle)) = J(\Delta(\varepsilon\langle u\rangle), a)$. By the induction hypothesis, the state reached on the string $\varepsilon\langle u\rangle$ is $\Delta(\varepsilon\langle u\rangle) = (b, X, c)$, where b is the first symbol of u and $X \subseteq N$ is the set of symbols that generate $\varepsilon\langle u\rangle$. Substituting this value into the expression for the state reached on $u\langle a\rangle$ yields $\Delta(u\langle a\rangle) = J((b, X, c), a) = (b, Z, a)$, where

$$\begin{aligned} Z &= \{\, A \mid \text{there exists a rule (8.4b) with } D_1, \ldots, D_m \in X \,\} = \\ &= \{\, A \mid \text{there exists a rule (8.4b) with } \varepsilon\langle u\rangle \in L_G(D_i) \text{ for all } i \,\}. \end{aligned}$$

The latter condition means that Z is the set of all symbols $A \in N$ that generate the string $u\langle a\rangle$ using a rule of the form (8.4b). Since this string can only be generated by rules of that form, this is equivalent to $Z = \{\, A \mid u\langle a\rangle \in L_G(A) \,\}$, as claimed.

Induction step II: $u\langle bvc\rangle$ with $u, v \in \Sigma^*$ and $b, c \in \Sigma$. The state computed on such a string is $\Delta(u\langle bvc\rangle) = \delta(\Delta(u\langle bv\rangle), \Delta(ub\langle vc\rangle))$. By the induction hypothesis, the states reached by the automaton on the strings $u\langle bv\rangle$ and $ub\langle vc\rangle$ are respectively $\Delta(u\langle bv\rangle) = (b, X, b')$ and $\Delta(ub\langle vc\rangle) = (c', Y, c)$, where b' is the last symbol of bv, c' is the first symbol of vc, $X \subseteq N$ is the set of nonterminal symbols generating $u\langle bv\rangle$ and $Y \subseteq N$ contains all such nonterminals that generate the string $ub\langle vc\rangle$.

Substituting the states reached on these shorter strings into the expression for the state computed on $u\langle bvc\rangle$ gives $\Delta(u\langle bvc\rangle) = \delta\big((b, X, b'), (c', Y, c)\big) = (b, Z, c)$, where

$$\begin{aligned} Z &= \big\{\, A \mid \text{there exists a rule (8.4a) with } C_i \in X \text{ and } B_i \in Y \,\big\} = \\ &= \big\{\, A \mid \text{there exists a rule (8.4a) with } u\langle bv\rangle \in L_G(C_i) \text{ and} \\ &\qquad\qquad ub\langle vc\rangle \in L_G(B_j), \text{ for all } i, j \,\big\}. \end{aligned}$$

That is, Z is exactly the set of nonterminals that generate the string $ub\langle vc\rangle$ by a rule of the form (8.4a). The string $u\langle bvc\rangle$ can only be generated by a rule of such a form, and, thus, $Z = \{\, A \mid u\langle bvc\rangle \in L_G(A) \,\}$, as desired. □

Lemma 8.2. *Let $M = (\Sigma, Q, I, J, \delta, F)$ be a trellis automaton with feedback and define the grammar with left contexts $G = (\Sigma, N, R, S)$, where $N = \{\, A_q \mid q \in Q \,\} \cup \{S\}$, and the set R contains the following rules.*

$$A_{I(a)} \to a \,\&\, \lhd\varepsilon \qquad (a \in \Sigma) \tag{8.5a}$$

$$A_{J(q,a)} \to a \,\&\, \lhd A_q \qquad (q \in Q,\ a \in \Sigma) \tag{8.5b}$$

$$A_{\delta(p,q)} \to bA_q \,\&\, A_p c \qquad (p, q \in Q,\ b, c \in \Sigma) \tag{8.5c}$$

$$S \to A_q \qquad (q \in F) \tag{8.5d}$$

Then, for every string with context $u\langle v\rangle$, $\Delta(u\langle v\rangle) = r$ if and only if $u\langle v\rangle \in L_G(A_r)$. In particular, $L(G) = \{\, w \mid \Delta(\varepsilon\langle w\rangle) \in F \,\} = L(M)$.

Proof. Induction on lexicographically ordered pairs $(|uv|, |v|)$.

Basis: $\varepsilon\langle a\rangle$ with $a \in \Sigma$. Then $\Delta(\varepsilon\langle a\rangle) = I(a)$. At the same time, $\varepsilon\langle a\rangle$ may only be generated by the rule of the form (8.5a), and such a rule for A_r exists if and only if $I(a) = r$.

Induction step I: $u\langle a\rangle$ with $u \in \Sigma^+$ and $a \in \Sigma$.

⟹ Let $\Delta(u\langle a\rangle) = r$. Then, $r = J(\Delta(\varepsilon\langle u\rangle), a)$. Let $q = \Delta(\varepsilon\langle u\rangle)$. By the induction hypothesis, $\varepsilon\langle u\rangle \in L_G(A_q)$. Since $J(q, a) = r$, the grammar contains a corresponding rule of the form (8.5b), which can be used to deduce the membership of $u\langle a\rangle$ in $L_G(A_r)$ as follows.

$$\vdash a\big(u\langle a\rangle\big) \qquad (axiom) \tag{8.6}$$

$$a\big(u\langle a\rangle\big), A_q\big(\varepsilon\langle u\rangle\big) \vdash A_r\big(u\langle a\rangle\big) \qquad (A_r \to a \,\&\, \lhd A_q) \tag{8.7}$$

⟸ Conversely, assume that $u\langle a\rangle \in L_G(A_r)$. Then its deduction must end with an application of a rule of the form (8.5b), as in step (8.7). By construction, the existence of such a rule implies $r = J(q, a)$. Applying the induction hypothesis to $A_q\big(\varepsilon\langle u\rangle\big)$ gives $\Delta(\varepsilon\langle u\rangle) = q$. Then the automaton calculates as follows: $\Delta(u\langle a\rangle) = J(\Delta(\varepsilon\langle u\rangle), a) = J(q, a) = r$, as desired.

Induction step II: $u\langle bvc\rangle$ with $u, v \in \Sigma^*$ and $b, c \in \Sigma$.

$\Rightarrow$ Assume first that $\Delta(u\langle bvc\rangle) = r$. Then $r = \delta(p, q)$, where $p = \Delta(u\langle bv\rangle)$ and $q = \Delta(ub\langle vc\rangle)$. By the induction hypothesis, $u\langle bv\rangle \in L_G(A_p)$ and $ub\langle vc\rangle \in L_G(A_q)$. From this, using a rule of the form (8.5c), one can have the following deduction.

$$\vdash b\big(u\langle b\rangle\big) \qquad (axiom) \tag{8.8}$$

$$\vdash c\big(ubv\langle c\rangle\big) \qquad (axiom) \tag{8.9}$$

$$b\big(u\langle b\rangle\big), A_q\big(ub\langle vc\rangle\big), A_p\big(u\langle bv\rangle\big), c\big(ubv\langle c\rangle\big) \vdash A_r\big(u\langle bvc\rangle\big) \quad (A_r \to bA_q \,\&\, A_pc) \tag{8.10}$$

Thus, $u\langle bvc\rangle$ is in $L_G(A_r)$, as claimed.

$\Leftarrow$ Conversely, if $u\langle bvc\rangle \in L_G(A_r)$, then the deduction establishing the proposition $A_r\big(u\langle bvc\rangle\big)$ must end as (8.10), using a rule of the form (8.5c). Then, by the construction, $r = \delta(p, q)$. Since the propositions $A_p\big(u\langle bv\rangle\big)$ and $A_q\big(ub\langle vc\rangle\big)$ are deduced in the grammar, by the induction hypothesis, $\Delta(u\langle bv\rangle) = p$ and $\Delta(ub\langle vc\rangle) = q$. Then $\Delta(u\langle bvc\rangle) = \delta(p, q) = r$. □

8.3 Defining a non-regular unary language

Ordinary context-free grammars over a unary alphabet $\Sigma = \{a\}$ define only regular languages. Unary linear conjunctive languages are also regular, because a trellis automaton operates on an input a^n as a deterministic finite automaton [12].

The non-triviality of unary conjunctive grammars was discovered by Jeż [31], who constructed a grammar for the language $\{\, a^{4^k} \mid k \geqslant 0 \,\}$ using iterated conjunction and concatenation of languages.

Let us show how one can construct formal grammars for non-regular languages over a unary alphabet by making use of a left context operator and not relying upon non-linear concatenation. The simplest case of the new method is demonstrated by the following automaton.

Example 8.3. Consider a trellis automaton with feedback $M = (\Sigma, Q, I, J, \delta, F)$ over the alphabet $\Sigma = \{a\}$ and with the set of states $Q = \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$, where $I(a) = \mathbf{p}$ is the initial state, the feedback function gives states $J(\mathbf{p}, a) = \mathbf{q}$ and $J(\mathbf{r}, a) = \mathbf{p}$, and the transition function is defined by $\delta(t, \mathbf{p}) = \mathbf{p}$ for all $t \in Q$, $\delta(\mathbf{q}, \mathbf{q}) = \delta(\mathbf{r}, \mathbf{q}) = \mathbf{q}$, $\delta(\mathbf{p}, \mathbf{q}) = \mathbf{r}$ and $\delta(\mathbf{p}, \mathbf{r}) = \mathbf{p}$. The only accepting state is $\mathbf{r}$. Then M recognizes the language $\{\, a^{2^k-2} \mid k \geqslant 2 \,\}$.

The computation of this automaton is illustrated in Figure 8.3. The state computed on each one-symbol substring $a^\ell\langle a\rangle$ is determined by the state computed on $\varepsilon\langle a^\ell\rangle$ according to the function J. Most of the time, $\Delta(\varepsilon\langle a^\ell\rangle) = \mathbf{p}$ and hence $\Delta(a^\ell\langle a\rangle) = \mathbf{q}$, and the latter continues into a

triangle of states **q**. Once for every power of two, the automaton computes the state **r** on $\varepsilon\langle a^{2^k-2}\rangle$, which sends a signal through the feedback channel, so that J sets $\Delta(a^{2^k-2}\langle a\rangle) = \mathbf{p}$. This in turn produces the triangle of states **p** and the next column of states **r**.

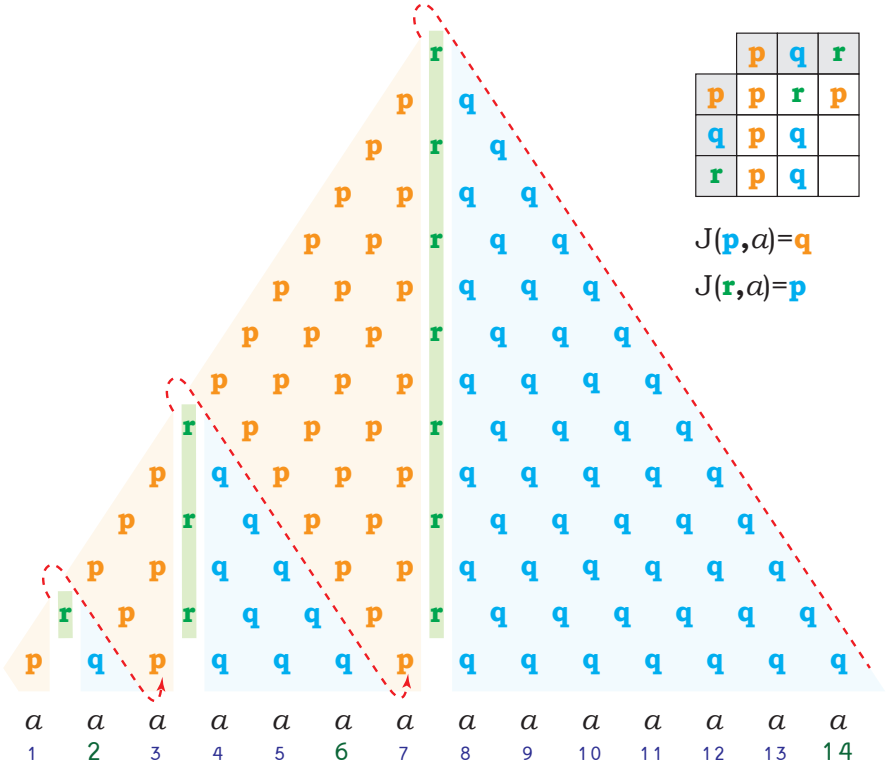


Figure 8.3: How the automaton in Example 8.3 recognizes $\{\, a^{2^k-2} \mid k \geqslant 2 \,\}$.

The automaton in Example 8.3 can be transformed to a grammar by Lemma 8.2. However, it is possible to construct a more succinct grammar, which, for each base $k \geqslant 2$, defines the language $L_k = \{\, a^{k^n} \mid n \geqslant 0 \,\}$.

$$S \to a \,\&\, \lhd\varepsilon \mid \varepsilon \,\&\, \lhd S \mid aSa^{k-1}$$

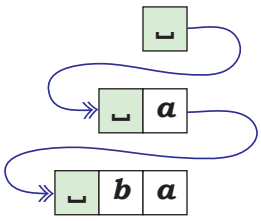
It is now known that linear grammars with contexts over a one-symbol alphabet are non-trivial. How far does their expressive power go? For conjunctive grammars (which allow non-linear concatenation, but no context specifications), Jeż and Okhotin [32, 33, 34] developed a method for manipulating base-k notation of the length of a string in a grammar, which allowed representing the following language: for every trellis automaton M over an alphabet $\{0, 1, \ldots, k-1\}$, there is a conjunctive grammar gener-

ating $L_M = \{ a^\ell \mid \text{the base-}k \text{ notation of } \ell \text{ is in } L(M) \}$ [32]. This led to the following undecidability method: given a Turing machine T, one first constructs a trellis automaton M for the language $\mathrm{VALC}(T) \subseteq \Sigma^*$ of computation histories of T; then, assuming that the symbols in Σ are digits in some base-k notation, one can define the unary version of $\mathrm{VALC}(T)$ by a conjunctive grammar.

Linear grammars with contexts are an entirely different model, and the automaton in Example 8.3 has nothing in common with the basic unary conjunctive grammar discovered by Jeż [31], in spite of defining almost the same language. The new model seems to be unsuited for manipulating base-k digits, and thus a different route to undecidability results has been taken, as explained below.

8.4 Simulating a Turing machine

The idea is that given a Turing machine, one shall construct a trellis automaton with feedback that simulates $\mathcal{O}(n)$ steps of the machine and accepts the input a^n, which is a string over a unary alphabet, depending on the state of the computation of the Turing machine at a certain time. Each individual cell, computed by a trellis automaton with feedback, should hold information about the computation of the Turing machine, such as the contents of a certain tape square at a certain time.



In order to simulate a Turing machine in such a way, it is useful to assume a machine of the following special kind. This machine operates on an initially blank tape, which is infinite to the left, and proceeds by making sweeps from left to right over the tape. When the machine arrives at the end of the tape, it appends a blank tape square to the left and begins a new sweep from that square.

Definition 8.2 (cf. [27, p. 126])**.** *A* rotating Turing machine *is a quintuple* $T = (\Gamma, \mathcal{S}, s_0, \nabla, \mathcal{F})$*, where*

- Γ *is a finite tape alphabet containing a blank symbol* $\square \in \Gamma$*;*
- $\mathcal{S}$ *is a finite set of states;*
- $s_0 \in \mathcal{S}$ *is the initial state;*
- *the function* $\nabla \colon \mathcal{S} \times \Gamma \to \mathcal{S} \times \Gamma$ *determines the next move of the Turing machine and is called a move function, to distinguish it from transition functions of trellis automata;*
- $\mathcal{F}$ *is a finite set of flickering states.*

A configuration *of T is a string of the form $[\![k]\!]usav$, where $k \geqslant 1$ is the number of the sweep, and $usav$ with $u, v \in \Gamma^*$, $a \in \Gamma$ and $s \in \mathcal{S}$ represents the tape contents uav with the head scanning the symbol a in the state s.*

The initial configuration of the machine is $[\![1]\!]s_0\square$. Each k-th sweep of the machine deals with a tape with k symbols, and consists of k steps of the following form.

$$[\![k]\!]uscdv \vdash_T [\![k]\!]uc's'dv \qquad (\nabla(s, c) = (s', c'))$$

After the last move of a sweep, the machine appends a blank symbol to the tape and then begins another sweep.

$$[\![k]\!]wsc \vdash_T [\![k+1]\!]s'\square wc' \qquad (\nabla(s, c) = (s', c'))$$

A rotating Turing machine never halts; at the end of each sweep, it may flicker *by entering a state from $\mathcal{F}$. Define the set of numbers accepted by T as follows.*

$$S(T) = \{\, k \mid [\![1]\!]s_0\square \vdash_T^* [\![k]\!]ws_{\mathrm{f}}c \text{ for } w \in \Gamma^*, s_{\mathrm{f}} \in \mathcal{F}, c \in \Gamma, |wc| = k \,\}$$

A similar class of Turing machines was studied by Ibarra and Kim [27]. Unlike the model defined in this section, the state of their machine is reset to the initial state in the beginning of each new sweep. Because of this, these devices are trivial over a unary alphabet.

The aforementioned Turing machines by Ibarra and Kim [27] can be understood as a model equivalent to trellis automata. A similar result holds for rotating Turing machines: they can be proved equivalent to trellis automata with feedback.

Let $T = (\Gamma, \mathcal{S}, s_0, \nabla, \mathcal{F})$ be a rotating Turing machine. Construct a trellis automaton with feedback $M_T = (\{a\}, Q, I, J, \delta, F)$ for simulating the machine as follows. Its set of states is $Q = \{\mathbf{q}_{sc} \mid s \in \mathcal{S},\ c \in \Gamma\}$, with each state $\mathbf{q}_{sc}$ representing the Turing machine's being in a state $s \in \mathcal{S}$, with its head scanning a symbol $c \in \Gamma$. Let the initial state of the automaton be $I(a) = \mathbf{q}_{s_0\square}$; this corresponds to the tape square $s_0\square$ in the beginning of the simulation.

Each diagonal of the automaton corresponds to the k-th sweep of the Turing machine, and the tape symbols written there represent the tape contents at the end of that sweep, as illustrated in Figure 8.4(b). In this kind of a computation, each transition of the trellis automaton has to simulate two moves of the Turing machine T. For any two neighbouring states in the M_T's computation, (s, c) and (t, d), the former state represents T in some k-th sweep at some position $i + 1$, whereas the latter state refers to T in the sweep number $k + 1$ at the position i, as illustrated in Figure 8.5(b). Then the trellis automaton has to determine the configuration in the $(k + 1)$-th

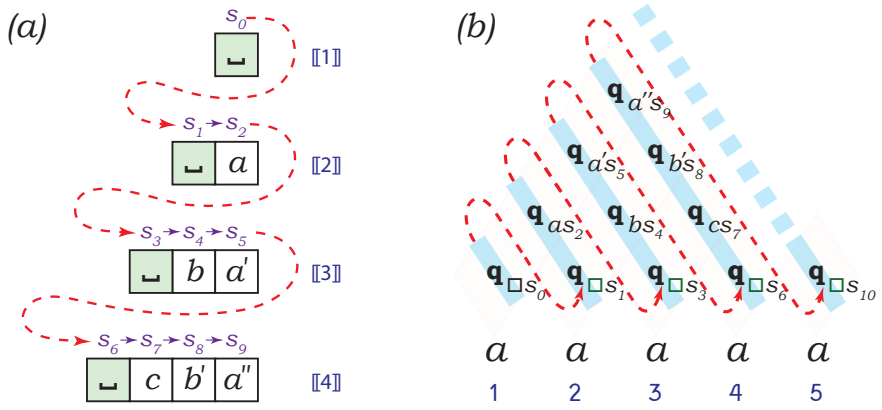


Figure 8.4: *(a)* Computation of a rotating Turing machine; *(b)* a trellis automaton with feedback simulating such a machine.

sweep at position $i+1$. The Turing machine arrives to that configuration in the state produced at the last step by a move $\nabla(t,d) = (t',d')$. Here, it sees a symbol produced at the previous sweep by another move $\nabla(s,c) = (s',c')$. These data are combined in the following transition.

$$\delta\,(\mathbf{q}_{sc},\ \mathbf{q}_{td}) = \mathbf{q}_{t'c'}$$

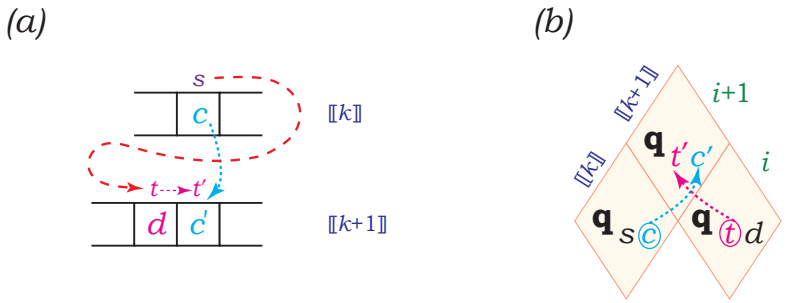


Figure 8.5: (a) Computation of a rotating Turing machine during k-th and $(k+1)$th sweeps; (b) trellis automaton's cells in the k-th and $(k+1)$-th diagonals corresponding to those sweeps.

After the last move of the machine in the current sweep, the feedback data channel J is used to append a new blank symbol to the bottom element of the next diagonal.

$$J(\mathbf{q}_{sc}) = \mathbf{q}_{s'\square} \qquad\qquad (\nabla(s,c) = (s',c'))$$

The set of accepting states of the automaton is $F = \{\, \mathbf{q}_{cs_{\mathrm{f}}} \mid c \in \Gamma,\ s_{\mathrm{f}} \in \mathcal{F}\,\}$.

The correctness of the construction is established in the following lemma.

Lemma 8.3. *Let $T = (\Gamma, \mathcal{S}, s_0, \nabla, \mathcal{F})$ be a rotating Turing machine, and let $M_T = (\{a\}, Q, I, J, \delta, F)$ be the trellis automaton with feedback constructed from T as described above. Then $L(M_T) = \{\, a^k \mid k \in S(T) \,\}$.*

The claim can be proved by a straightforward induction, inferring the state in each cell from the previously determined values of the neighbouring cells.

8.5 Implications

The simulation of Turing machines by trellis automata with feedback over a one-symbol alphabet is useful for proving undecidability of basic decision problems for these automata, such as emptiness or equivalence. Due to Theorem 8.2, the same results equally hold for linear grammars with contexts. Besides establishing the undecidability, these results determine the exact level of undecidability of these problems in the arithmetical hierarchy.

The *arithmetical hierarchy* is defined as follows. A set is in Σ_k^0, if it can be expressed as $\{\, y \mid \exists x_1 \forall x_2 \ldots Q_k x_k \;\; P(x_1, \ldots, x_k, y) \,\}$, for some recursive predicate P, where $Q_k = \exists$ if k is odd, and $Q_k = \forall$ if k is even. Similarly, a set is in Π_k^0, if its complement is in Σ_k^0, that is, if it admits a representation $\{\, y \mid \forall x_1 \exists x_2 \ldots Q_k x_k \;\; P(x_1, \ldots, x_k, y) \,\}$. In particular, the class Σ_1^0 at the first level of the arithmetical hierarchy is the class of recursively enumerable (r.e.) sets. Each class Σ_i^0 and Π_i^0, with $i \geqslant 1$, has complete sets with respect to many-one reductions by Turing machines. For example, the Turing machine halting problem is Σ_1^0-complete, whereas non-halting is Π_1^0-complete.

The first decision problem for linear grammars with contexts is testing whether the language defined by a given grammar is empty. The undecidability of the *emptiness problem* follows from Lemma 8.3. To be precise, the problem is complete for the complements of the r.e. sets.

Theorem 8.3. *The emptiness problem for linear grammars with left contexts over a one-symbol alphabet is Π_1^0-complete. It remains in Π_1^0 for each alphabet.*

Proof. The non-emptiness problem is clearly recursively enumerable, because one can simulate a trellis automaton with feedback on all inputs, accepting if it ever accepts. If the automaton accepts no strings, the algorithm does not halt.

The Π_1^0-hardness is proved by reduction from the Turing machine halting problem (non-halting, to be precise). Given a machine T and an input w, construct a rotating Turing machine T_w, which first prints w on the tape (over $1 + \log |w|$ sweeps, using around $|w|$ states), and then proceeds by simulating T, using one sweep for each step of T. If the simulated machine

T ever halts, then T_w changes into a special state s_f and continues moving its head until the end of the current sweep.

Construct a trellis automaton with feedback M simulating the machine T_w according to Lemma 8.3, and define its set of accepting states as $F = \{\, \mathbf{q}_{cs_\mathrm{f}} \mid c \in \Sigma \,\}$. Then, by the theorem, M accepts some string a^ℓ if and only if T_w ever enters the state s_f, which is in turn equivalent to T's halting on w. □

For conjunctive grammars, there is a stronger result than just the undecidability of the emptiness problem. Namely, *for every fixed conjunctive language* L_0 over any alphabet, the problem of testing whether a given conjunctive grammar describes L_0 is Π_1^0-complete [68, Thms. 25,26]. The same property holds for linear grammars with left contexts, and has a much simpler proof than for conjunctive grammars.

Corollary 8.1. For every fixed language L_0 defined by a linear grammar with left contexts, the problem of testing whether a given linear grammar with left contexts defines the language L_0 is Π_1^0-complete.

Proof. The problem is in Π_1^0, because its complement, the *inequivalence to* L_0, is recursively enumerable, solved by simulating a trellis automaton with feedback on all inputs, looking for any mismatches to L_0.

The Π_1^0-hardness is proved by reduction from the emptiness problem. Given a grammar G, one can construct a new grammar G_1 that describes the *symmetric difference* of $L(G)$ and L_0; the corresponding construction shall be explained later in Lemma 8.6. Then, $L(G_1) = L_0$ if and only if $L(G) = \varnothing$, which completes the reduction. □

The more general problem of checking the equivalence of two given grammars is also Π_1^0-complete (its complement is still recursively enumerable, through the same method of simulating automata on all inputs).

Corollary 8.2. The equivalence problem for linear grammars with left contexts is Π_1^0-complete.

The second slightly more difficult undecidability result asserts that testing the finiteness of a language generated by a given grammar is complete for the second level of the arithmetical hierarchy.

Theorem 8.4. *The finiteness problem for linear grammars with left contexts over a one-symbol alphabet is* Σ_2^0*-complete. It remains* Σ_2^0*-complete for any alphabet.*

Proof (a sketch). Reduction from the finiteness problem for a Turing machine, which is Σ_2^0-complete, see Rogers [77, §14.8]. Given a Turing machine T, construct a rotating Turing machine T', which simulates T running on all

inputs, with each simulation using a segment of the tape. Initially, T' sets up to simulate T running on ε, and then it regularly begins new simulations. Every time one of the simulated instances of T accepts, the constructed machine "flickers" by entering an accepting state in the end of one of its sweeps. Construct a trellis automaton with feedback M corresponding to this machine. Then $L(M)$ is finite if and only if $L(T)$ is finite. □

8.6 Further examples of grammars

This section presents two additional examples of linear grammars with one-sided contexts. These examples are useful for understanding the expressive power of this new class of grammars.

The language described in the first example is known to have no linear conjunctive grammar [96].

Example 8.4 (Törmä [90])**.** The following linear grammar with contexts defines the language $\{\, a^n b^{in} \mid i, n \geqslant 1 \,\}$.

$$\begin{aligned} S &\to ab \mid aSb \mid aB \\ B &\to bB \mid b \,\&\, \lhd S \end{aligned}$$

The languages defined by nonterminals B and S are as follows.

$$\begin{aligned} L(B) &= \{\, a^n b^{in-j} \langle b^j \rangle \mid n \geqslant 1,\ 1 \leqslant j \leqslant in \,\} \\ L(S) &= \{\, a^j \langle a^{n-j} b^{in-j} \rangle \mid n \geqslant 1,\ 0 \leqslant j < n \,\} \end{aligned}$$

For each $n \geqslant 1$, the strings $a^n b^{in}$, with $i \geqslant 1$, are defined inductively on i. The first string $a^n b^n$ is defined by the first two rules for nonterminal S. In Figure 8.6, this is illustrated by the leftmost vertical S-column.

Every next string $w' = a^n b^{(i+1)n}$ is obtained from the previous string $w = a^n b^{in}$ by appending as many symbols b as there are symbols a in w. First, a context operator in the rule $B \to b \,\&\, \lhd S$ produces a one-symbol string $a^n b^{in} \langle b \rangle$. Then, B generates the string $a^n \langle b^{in+1} \rangle$ by the rule $B \to bB$ applied $i \cdot n$ times, as shown in a B-diagonal in the figure.

Next, the rule $S \to aB$ generates the string $a^{n-1} \langle a b^{in+1} \rangle$, thus setting up the matching of the new symbols b to the existing symbols a. Then, each application of the rule $S \to aSb$ appends one a and one b, and, overall, it appends as many symbols b as there as as in the beginning of the string, as illustrated in an S-column emerging from a B-diagonal. The resulting string with an empty context is $\varepsilon \langle a^n b^{(i+1)n} \rangle$, as desired.

The second example defines the set of squares over a unary alphabet. This example is based upon an idea of Birman and Ullman [7], who used it

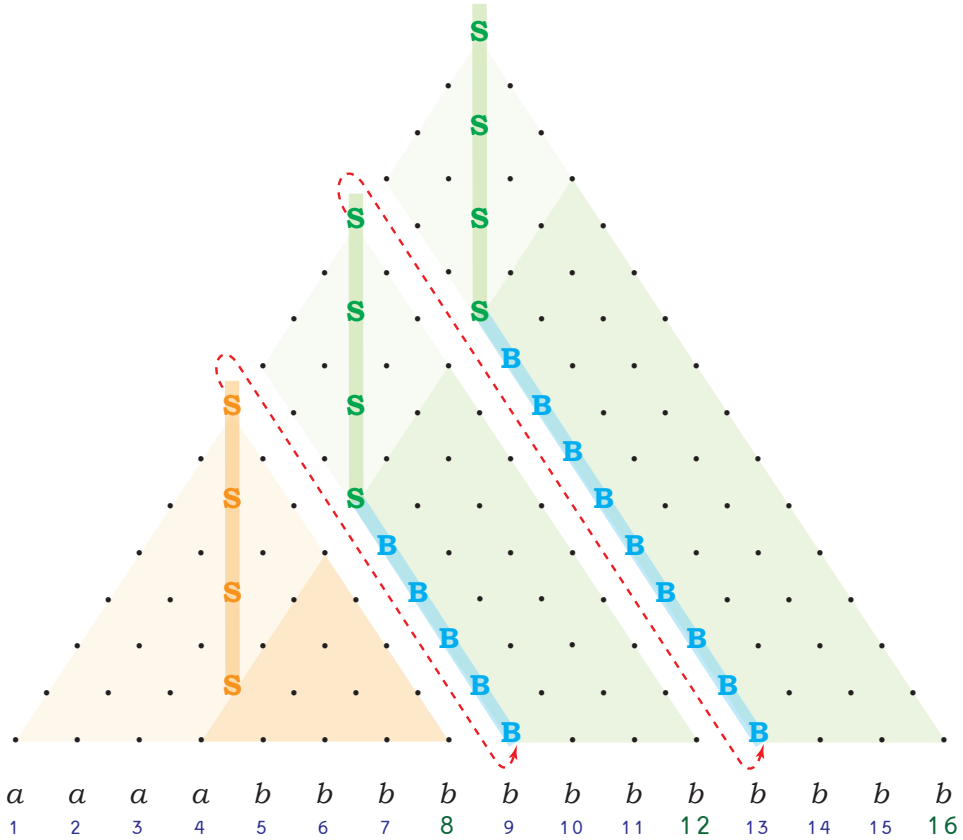


Figure 8.6: The grammar in Example 8.4 defining the language $\{ a^n b^{in} \mid i, n \geqslant 1 \}$.

to recognize the set of squares using a special recursive descent parser with backtracking.

Example 8.5 (adapted from Birman and Ullman [7, p. 21])**.** The following linear grammar with contexts describes the language $\{ a^{n^2} \mid n \geqslant 0 \}$.

$$\begin{array}{lcl} S & \to & a \,\&\, {\lhd}\varepsilon \mid aaaa \,\&\, {\lhd}\varepsilon \mid aBaa \,\&\, {\lhd}S \mid aS \\ B & \to & a \,\&\, {\lhd}S \mid aBa \,\&\, {\lhd}C \\ C & \to & a \,\&\, {\lhd}S \mid aC \mid Ba \end{array}$$

The grammar is centered around the nonterminal S, which should define, among others, strings of the form $\varepsilon\langle a^m \rangle$, with m being a perfect square. The two shortest strings $\varepsilon\langle a \rangle$ and $\varepsilon\langle a^4 \rangle$ are explicitly defined in the first two rules for S. Other prefixes of the string, that is, $\varepsilon\langle a^m \rangle$, where m is *not* a

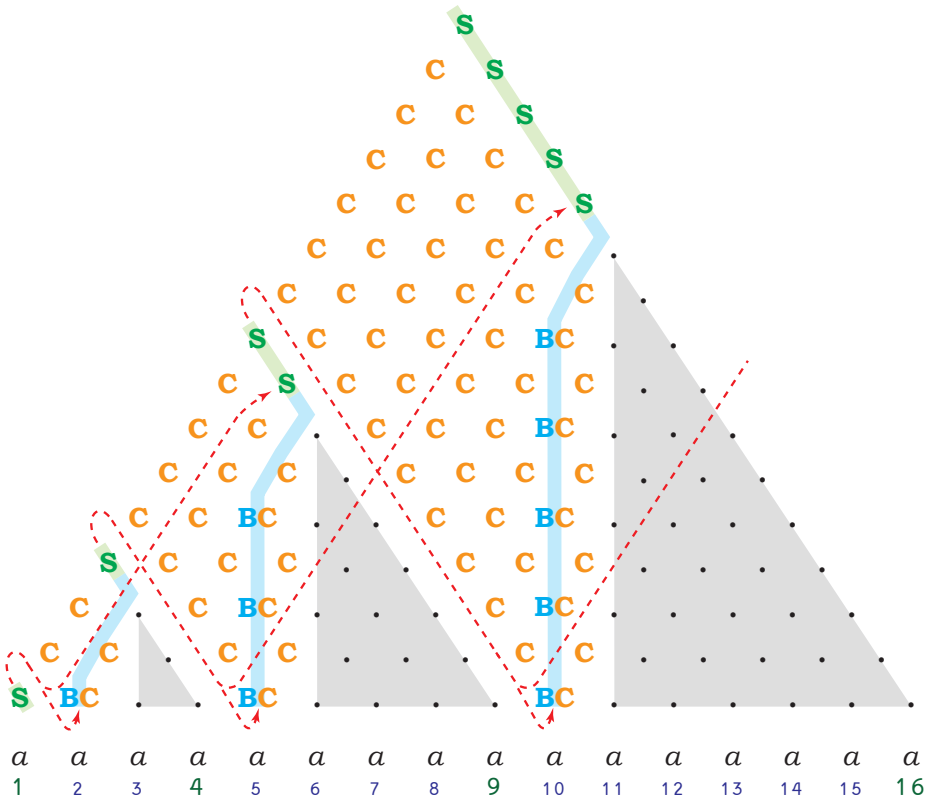


Figure 8.7: The grammar in Example 8.5 describing the language $\{\, a^{n^2} \mid n \geqslant 0 \,\}$.

perfect square, are defined by the symbol C, which acts more or less as the complement of S.

Nonterminal B generates vertical B-columns, each beginning in a one-symbol string $a^{n^2}\langle a \rangle$, with $n \geqslant 1$, and then rising up through $a^{n^2-i}\langle a^{2i+1} \rangle$, for all $i = 1, \ldots, 2n-2$. The rule $B \to a \,\&\, \lhd S$ defines the bottom element of every such column, and the companion rule $C \to a \,\&\, \lhd S$ simultaneously defines symbol C in the same position. Every next element of a vertical B-column is defined from the previous element by the rule $B \to aBa \,\&\, \lhd C$. The context C in this rule is satisfied *unless the left context is a perfect square*. Therefore, the height of this column of Bs is equal to the number of symbols C between the two previous occurrences of a perfect square in the main diagonal. Thus, the height of every consecutive B-column is by 2 greater than the height of the previous one.

The symbols C are spawned from B-columns. The rule $C \to Ba$ defines an initial C-column to the right of a B-column, whereas the rule $C \to aC$ propagates C towards the main diagonal.

The nonterminal S generates S-diagonals, starting with a string $a^{(n-2)^2}\langle a^{n^2}\rangle$ and propagating up to desired strings of the form $\varepsilon\langle a^{n^2}\rangle$. The rule $S \to aBaa \,\&\, \lhd S$ defines the bottom element of an S-diagonal as a combination of two conditions. First, $aBaa$ makes a jump from the topmost symbol B in a B-column in a fixed direction and distance (this can be done from any B in this column). At the same time, the context S in this rule refers to the next to last perfect square. These two conditions together define the bottom of an S-diagonal; the rule $S \to aS$ propagates symbols S towards the main diagonal.

The languages generated by the nonterminals S, B and C are as follows.

$$\begin{aligned} L(S) &= \{\, a^i\langle a^{n^2-i}\rangle \mid n \geqslant 1,\ 0 \leqslant i \leqslant (n-2)^2 \,\} \\ L(B) &= \{\, a^{n^2-i}\langle a^{2i+1}\rangle \mid n \geqslant 1,\ 0 \leqslant i \leqslant 2n-2 \,\} \\ L(C) &= \{\, a^i\langle a^{n^2+j}\rangle \mid n \geqslant 1,\ 1 \leqslant j \leqslant 2n,\ 0 \leqslant i \leqslant \min\{n^2-j+2, n^2\} \,\} \end{aligned}$$

8.7 Closure properties

A few additional results on the expressive power of linear grammars with contexts concern their closure under several operations.

The first result is the closure under concatenating a linear conjunctive language from the right, which is interesting because the family of linear conjunctive languages is itself not closed under concatenation [87].

Lemma 8.4. *Let $K \subseteq \Sigma^*$ be defined by a linear grammar with contexts, and let $L \subseteq \Sigma^*$ be a linear conjunctive language. Then the language $K \cdot L$ can be defined by a linear grammar with contexts.*

Proof. Let $G_1 = (\Sigma, N_1, R_1, S_1)$ and $G_2 = (\Sigma, N_2, R_2, S_2)$ be the grammars generating the languages K and L, respectively. Construct a linear conjunctive grammar with contexts $G = (\Sigma, N_1 \cup N_2 \cup \{S\}, R_1 \cup R_2 \cup R, S)$, where R contains the following rules.

$$\begin{aligned} &S \to aS && \text{(for all } a \in \Sigma\text{)} \\ &S \to S_2 \,\&\, \lhd S_1 \end{aligned}$$

The latter rule represents the concatenation of a string u from K with a string v from L by expressing v written in the context u, that is, $u\langle v\rangle$. The goal is to describe $\varepsilon\langle uv\rangle$, which is done by applying the rules of the form $S \to aS$ for every symbol of u. □

This, in particular, implies that the language

$$L = \{\, a^{i_1}b^{j_1} \ldots a^{i_m}b^{j_m} \mid m \geqslant 2,\ i_t, j_t \geqslant 1,\ \exists \ell : i_1 = j_\ell \wedge i_{\ell+1} = j_m \,\},$$

used by Terrier [87] to show that linear conjunctive languages are not closed under concatenation, can be defined by a linear grammar with contexts.

By the same method as in Lemma 8.4, one can show that the Kleene star of any linear conjunctive language can be represented by a linear grammar with contexts.

Lemma 8.5. *Let L be a linear conjunctive language. Then the language L^* can be defined by a linear grammar with contexts.*

Proof. Let $G = (\Sigma, N, R, S)$ be a linear conjunctive grammar that defines L. Construct a linear grammar with contexts $G' = (\Sigma, N \cup \{S', \widetilde{S}\}, R \cup R', S')$, with the following rules in R'.

$$\begin{aligned} S' &\to \widetilde{S} \,\&\, \lhd\varepsilon \mid \varepsilon \,\&\, \lhd\varepsilon \\ \widetilde{S} &\to S \,\&\, \lhd S' \\ \widetilde{S} &\to a\widetilde{S} \qquad\qquad\qquad\qquad \text{(for all } a \in \Sigma) \end{aligned}$$

Then $L(G') = L(G)^*$.

In this grammar, the nonterminal S' defines all strings of the form $\varepsilon\langle u_1 \dots u_k\rangle$, with $k \geqslant 0$ and $u_1, \dots, u_k \in L(G)$. For $k = 0$, the empty string is generated by the rule $S \to \varepsilon \,\&\, \lhd\varepsilon$. Every next string $\varepsilon\langle u_1 \dots u_k u_{k+1}\rangle$ is represented by concatenating the previous string $\varepsilon\langle u_1 \dots u_k\rangle$ from S' to a string u_{k+1} given by S in the way similar to the previous lemma. □

Similarly to the case of linear conjunctive languages [54, Thm. 7], the languages defined by linear grammars with contexts are closed under concatenation and star over disjoint alphabets, through a center marker, etc.

Turning to Boolean operations on languages, the closure under union and under intersection is obvious, as these operations can be expressed in grammars. In spite of having no negation operator, the language family defined by linear grammars with contexts is closed under all Boolean operations, just like the linear conjunctive languages.

Lemma 8.6. *If the languages K, L are defined by linear grammars with left contexts, then so are the languages $K \cup L$, $K \cap L$ and $\overline{L}$.*

For complementation, one can define a direct grammar-to-grammar construction, as for linear conjunctive grammars [54]. However, an easier approach is to construct a trellis automaton with feedback recognizing the given language, and then invert its set of accepting states.

Another standard operation on formal languages is the *quotient:* for two languages $K, L \subseteq \Sigma^*$, their left quotient is $K^{-1} \cdot L = \{\, v \mid \exists u \in K : uv \in L \,\}$ and their right quotient is $L \cdot K^{-1} = \{\, u \mid \exists v \in K : uv \in L \,\}$. Already for linear conjunctive languages, it is known that every recursively enumerable

set is representable as a quotient of a linear conjunctive language and a regular language [54, Thm. 11]. Therefore, the languages defined by linear grammars with contexts are also not closed under this operation. However, quotient with a finite language preserves this family. The construction is slightly different for quotient on the left and on the right, because context operators act only on the left.

Lemma 8.7. *Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $K \subset \Sigma^*$ be a finite language. Then the language $L(G) \cdot K^{-1}$ can be defined by a linear grammar with contexts.*

Proof (a sketch). Whenever the language K consists of multiple strings, the desired quotient can be represented as a union of quotients with singletons.

$$LK^{-1} = \bigcup_{u \in K} L\{u\}^{-1}$$

For a finite K, this is finite union. Furthermore, the quotient with a string $u = a_1 \dots a_\ell$ is representable as a sequence of ℓ quotients with one-symbol strings. Then, since the family of languages defined by linear grammars with contexts is closed under union, it is sufficient to prove the closure in the case of K containing a single one-symbol string: $K = \{d\}$, with $d \in \Sigma$.

Assume, without loss of generality, that G is in the linear normal form, provided by Theorem 8.1. Construct a grammar $G' = (\Sigma, N \cup N', R', S')$, where $N' = \{ A' \mid A \in N \}$. The intention is to have $L_{G'}(A) = L_G(A)$ and $L_{G'}(A') = \{ u\langle v \rangle \mid u\langle vd \rangle \in L_G(A) \}$ for all $A \in N$. The new rules in R' are defined as follows.

- For each rule of the form $A \to bB_1 \,\&\, \dots \,\&\, bB_\ell \,\&\, C_1 c \,\&\, \dots \,\&\, C_k c$, the set R' contains the same rule for A.

  $$A \to bB_1 \,\&\, \dots \,\&\, bB_\ell \,\&\, C_1 c \,\&\, \dots \,\&\, C_k c$$

  If all symbols concatenated from the right are equal to d (that is, if $c = d$ or if $k = 0$), then there is also a truncated rule for A'.

  $$A' \to bB'_1 \,\&\, \dots \,\&\, bB'_\ell \,\&\, C_1 \,\&\, \dots \,\&\, C_k \qquad (\text{if } c = d)$$

- Every rule $A \to a \,\&\, \lhd D_1 \,\&\, \dots \,\&\, \lhd D_m$ is included in R' as it is, and if $a = d$, a truncated rule for A' is added.

  $$\begin{aligned} A &\to a \,\&\, \lhd D_1 \,\&\, \dots \,\&\, \lhd D_m \\ A' &\to \varepsilon \,\&\, \lhd D_1 \,\&\, \dots \,\&\, \lhd D_m \qquad (\text{if } a = d) \end{aligned}$$

The facts that $\vdash_{G'} A\big(u\langle v\rangle\big)$ if and only if $\vdash_G A\big(u\langle v\rangle\big)$ and $\vdash_{G'} A'\big(u\langle v\rangle\big)$ if and only if $\vdash_G A\big(u\langle vd\rangle\big)$ can be proved by an easy induction. Hence, $L(G') = L(G) \cdot \{d\}^{-1}$. □

The construction for the closure with a symbol on the left is similar but not symmetric.

Lemma 8.8. *Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $K \subset \Sigma$ be a finite language. Then the language $K^{-1} \cdot L(G)$ can be defined by a linear grammar with contexts.*

Proof (a sketch). Similarly to the previous lemma, the quotient from the left can be represented as a union of quotients with singletons.

Let G be in the linear normal form. Construct a grammar $G' = (\Sigma, N \cup N', R', S')$, where $N' = \{\, A' \mid A \in N \,\}$, such that $L_{G'}(A) = L_G(A)$ and $L_{G'}(A') = \{\, u\langle v\rangle \mid du\langle v\rangle \in L_G(A) \,\}$ for all $A \in N$, as follows.

- For each rule of the form $A \to bB_1 \,\&\, \ldots \,\&\, bB_\ell \,\&\, C_1c \,\&\, \ldots \,\&\, C_kc$, the set R' contains this rule, and, if $b = d$ or $\ell = 0$, then a truncated rule for A' is also added to R'.

$$\begin{aligned} &A \to bB_1 \,\&\, \ldots \,\&\, bB_\ell \,\&\, C_1c \,\&\, \ldots \,\&\, C_kc \\ &A' \to B_1 \,\&\, \ldots \,\&\, B_\ell \,\&\, C'_1c \,\&\, \ldots \,\&\, C'_kc \qquad\qquad (\text{if } b = d) \end{aligned}$$

- For a rule $A \to a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m$, the set R' contains the following rule.

$$A' \to a \,\&\, \lhd D'_1 \,\&\, \ldots \,\&\, \lhd D'_m$$

  If additionally $a = d$ and $m = 0$, then an extra rule

$$A' \to \varepsilon$$

  is added to R'.

Similarly to the previous lemma, one can prove that $\vdash_{G'} A\big(u\langle v\rangle\big)$ if and only if $\vdash_G A\big(u\langle v\rangle\big)$ and $\vdash_{G'} A'\big(u\langle v\rangle\big)$ if and only if $\vdash_G A\big(u\langle vd\rangle\big)$. □

The last class of operations to be considered are *homomorphisms*. Given two alphabets Σ and Ω, a homomorphism $h\colon \Sigma^* \to \Omega^*$ is a mapping that satisfies $h(\varepsilon) = \varepsilon$ and $h(uv) = h(u)h(v)$ for all $u, v \in \Sigma$; it is completely defined by the images of one-symbol strings. A homomorphism is a *code*, if $h(u) = h(v)$ implies $u = v$. It is known that the family of linear conjunctive languages is closed under a homomorphism h if and only if either h is a code, or h trivially maps everything to the empty string [63]. The closure under codes also holds for linear grammars with contexts.

Lemma 8.9. *Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $h\colon \Sigma^* \to \Omega^*$ be a code. Then the language $h(L(G))$ is defined by a linear grammar with contexts.*

Proof. Construct a new linear grammar with contexts $G' = (\Omega, N, R', S)$ with the following set of rules. Consider any rule in R.

$$A \to x_1 B_1 y_1 \,\&\, \ldots \,\&\, x_k B_k y_k \,\&\, \lhd x'_1 D_1 y'_1 \,\&\, \ldots \,\&\, \lhd x'_m D_m y'_m \,\&\, \trianglelefteq x''_1 E_1 y''_1 \,\&\, \ldots \,\&\, \trianglelefteq x''_n E_n y''_n$$

Then the new grammar contains a rule with all strings encoded by h.

$$A \to h(x_1) B_1 h(y_1) \,\&\, \ldots \,\&\, h(x_k) B_k h(y_k) \,\&\, \lhd h(x'_1) D_1 h(y'_1) \,\&\, \ldots \,\&\, \lhd h(x'_m) D_m h(y'_m) \,\&\, \trianglelefteq h(x''_1) E_1 h(y''_1) \,\&\, \ldots \,\&\, \trianglelefteq h(x''_n) E_n h(y''_n)$$

It is claimed that every symbol $A \in N$ generates the language $L_{G'}(A) = h(L_G(A))$. In one direction, it has to be shown that for every string $u\langle v\rangle$ generated by A in G, the encoding $h(u)\langle h(v)\rangle$ is in $L_{G'}(A)$. This is actually true for every homomorphism h. In the other direction, if an item $A\big(x\langle y\rangle\big)$ can be deduced in the grammar G', then x and y are images of uniquely determined strings $u, v \in \Sigma^*$, with $h(u) = x$ and $h(v) = y$, and one can prove that the string $u\langle v\rangle$ is in $L_G(A)$. Both proofs follow by a straightforward induction on the number of steps in the derivations. □

As compared to the well-researched closure properties of other families of formal grammars [68, Sect. 8.2], the results for grammars with contexts, whether linear or with general concatenation, are still quite fragmentary. It is conjectured that they are not closed under most of the basic operations, such as concatenation and star, but this cannot yet be proved due to the lack of negative proof methods for these grammars.

Chapter 9

Conclusions and future work

Thus, the formalism of context-free grammars has been extended to allow context specifications, and the basic properties of the resulting models have been explored. Most importantly, the new grammars manage to maintain the principle of defining the properties of strings recursively from the properties of their substrings, and introducing the properties of left and right contexts into this scheme did not break it. Such inductive definitions form parse trees, which can be constructed by efficient parsing algorithms.

One can speculate that the models introduced in this thesis are apparently the only way of defining contexts in context-free grammars, which maintains their meaning as a formalism for syntactic descriptions. However, it would be premature to suggest that grammars with context specifications can be indeed used for practical representation of languages. In fact, of all countless extensions of context-free grammars proposed in the literature, probably only the tree-adjoining grammars [35] have seen any practical use. However, the proposed grammars are at the very least not totally absurd, and applying them is not out of question.

The new model leaves many theoretical questions to ponder. The most crucial open problem is *whether grammars with contexts can define any languages that cannot be generated by a conjunctive grammar* [50, 68], that is, without ever using the context operators. Unfortunately, there are no known non-trivial methods for proving that a given language is not defined by any conjunctive grammar [68]. Thus, showing that conjunctive grammars are weaker in power than grammars with contexts requires further understanding of conjunctive grammars.

Establishing any limitations of the expressive power of grammars with contexts appears to be an even more challenging task. A possible language that might be not representable by these grammars is the language $\{\, ww \mid w \in \{a, b\}^* \,\}$, which, on the other hand, can be defined both by Boolean grammars [56] and by tree-adjoining grammars.

For the subclass of unambiguous grammars with contexts, it would be interesting to know whether it is weaker in power than the full family of grammars with contexts. In other words, *are there any inherently ambiguous grammars with contexts*? This question is also beyond the current state of knowledge on formal grammars: already for conjunctive grammars and their unambiguous subclass, no separation is known [62].

Consider another natural theoretical question: *is there a parsing algorithm for grammars with one-sided contexts working in substantially less than cubic time*? For ordinary context-free grammars, Valiant [91] discovered an algorithm that offloads the most intensive computations into calls to a Boolean matrix multiplication procedure, and thus can work in time $\mathcal{O}(n^\omega)$, with $\omega < 3$; according to the current knowledge on matrix multiplication, ω can be reduced to 2.373. The main idea of Valiant's algorithm equally applies to conjunctive and Boolean grammars, which can be parsed in time $\mathcal{O}(n^\omega)$ as well [70]. However, extending it to grammars with one-sided contexts seems to be inherently impossible, because the logical dependencies between the properties of substrings (that is, between the entries of the table $T_{i,j}$) now have a more complicated structure, and the order of calculating these entries apparently rules out grouping multiple operations into Boolean matrix multiplication. However, there might exist a different $\mathcal{O}(n^\alpha)$-time parsing strategy for these grammars, with $\alpha < 3$, which would be interesting to discover.

The complexity of the known parsing algorithms for different families of formal grammars is as follows (with $n^{o(1)}$ factor omitted):

- n^2 for unambiguous conjunctive and Boolean grammars, as well as for all their subclasses down to linear grammars with disjunction only [62];
- n^ω for the general case of context-free grammars, including their conjunctive and Boolean variants [91, 70];
- n^3 for grammars with one-sided contexts, as proved in Theorem 4.1;
- n^3 for grammars with two-sided contexts, as proved by Rabkin [74] while this thesis was under preparation;
- $n^{2\omega}$ for tree-adjoining grammars [75].

The cubic-time parsing algorithm for grammars with left contexts, its version for unambiguous grammars as well as the tabular parsing algorithm for grammars with two-sided contexts have been implemented by the author in a prototype software. Given a description of a grammar, the tool generates a parser for this grammar as a program in the programming language C#. However, this tool is a research-level implementation of the mentioned algorithms, developed to demonstrate their correctness, and only provides

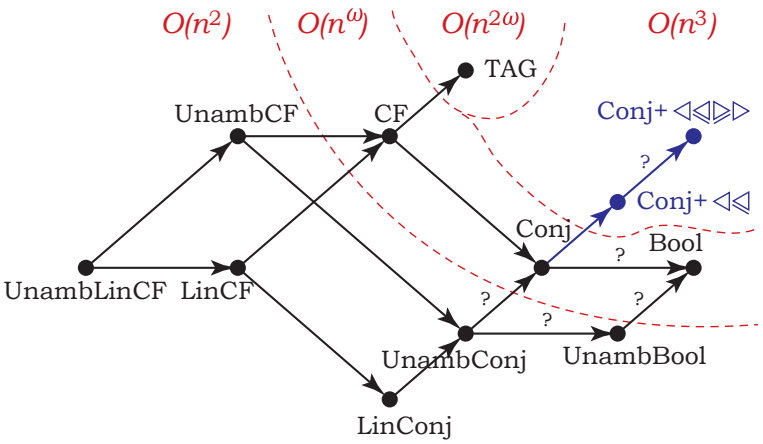


Figure 9.1: Hierarchy of grammar families. Complexity of known parsing algorithms.

a primitive programmer interface. Thus, *implementing the algorithms for grammars with contexts in a practical software tool* remains an open task.

Among other theoretical questions to research about the new models are *their closure properties*, *efficient parsing algorithms* and *subfamilies that admit more efficient parsing*. Yet another possibility for further studies is *investigating Boolean and stochastic variants of grammars with contexts*, following the recent related work [16, 44, 97].

Knowing that regular context operators do not increase the expressive power of the basic families of formal grammars leaves the following problem to study: *what is the effect of the regular context elimination procedure on the size of grammars*? It is also interesting to know whether *there exist any languages described by small grammars with regular contexts, which would require large grammars without context operators*. If the succinctness tradeoff is significant, then, instead of eliminating context operators in a grammar, it could be more practical to augment the existing parsing algorithms to handle these context operators directly.

As for linear grammars with contexts, a suggested topic for future research is to investigate the main ideas in the literature on trellis automata [12, 14, 27, 87] and see *whether they can be extended to trellis automata with feedback* (and hence to linear grammars with contexts). In particular, it is essential to learn *how to prove that some languages cannot be recognized by any automaton of this kind*. It is also interesting to see *how the parsing algorithms for grammars with contexts*, conjunctive grammars and related models [3, 68] *could be adapted to linear grammars with contexts*.

⋆⋆⋆

There must have been other good ideas in the theory of formal grammars that were inadequately formalized before and they may be worth being re-investigated using the logical approach.

This might be the future of the area...

Bibliography

[1] A. V. Aho, "Indexed grammars – an extension of context-free grammars", *Journal of the ACM*, 15:4 (1968), 647–671.

[2] A. V. Aho, J. D. Ullman, "The theory of parsing, translation, and compiling", Prentice-Hall (1972).

[3] T. Aizikowitz, M. Kaminski, "LR(0) conjunctive grammars and deterministic synchronized alternating pushdown automata", *Computer Science in Russia* (CSR 2011, St. Petersburg, Russia, 14–18 June 2011), LNCS 6651, 345–358.

[4] J. Autebert, J. Berstel, L. Boasson, "Context-free languages and pushdown automata", in: Rozenberg, Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, 1997, 111–174.

[5] Y. Bar-Hillel, M. Perles, E. Shamir, "On formal properties of simple phrase-structure grammars", *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14 (1961), 143–177.

[6] M. E. Bermudez, K. M. Schimpf, "Practical arbitrary lookahead LR parsing", *Journal of Computer and System Sciences*, 41:2 (1990), 230–250.

[7] A. Birman, J. Ullman, "Parsing algorithms with backtrack", *Information and Control*, 23, 1–34 (1973).

[8] A. Brüggemann-Klein, D. Wood, "On predicative parsing and extended context-free grammars", *Implementation and Application of Automata, 7th International Conference*, (CIAA 2002, Tours, France, 3-5 July 2002), LNCS 2608, 239–247.

[9] N. Chomsky, "On certain formal properties of grammars", *Information and Control*, 2:2 (1959), 137–167.

[10] J. Cleaveland, R. Uzgalis, *Grammars for programming languages*, Elsevier (1977).

[11] K. Čulík II, R. Cohen, "LR-regular grammars—an extension of LR(k) grammars", *Journal of Computer and System Sciences*, 7:1 (1973), 66–96.

[12] K. Čulík II, J. Gruska, A. Salomaa, "Systolic trellis automata", *International Journal of Computer Mathematics*, 15 (1984) 195–212; 16 (1984) 3–22.

[13] W. F. Dowling, J. H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional Horn formulae", *Journal of Logic Programming*, 1:3 (1984), 267–284.

[14] C. Dyer, "One-way bounded cellular automata", *Information and Control*, 44 (1980), 261–281.

[15] J. Earley, "An efficient context-free parsing algorithm", *Communications of the ACM*, 13:2 (1970), 94–102.

[16] Z. Ésik, W. Kuich, "Boolean fuzzy sets", *International Journal of Foundations of Computer Science*, 18:6 (2007), 1197–1207.

[17] R. W. Floyd, "On the nonexistence of a phrase structure grammar for ALGOL 60", *Communications of the ACM*, 5 (1962), 483–484.

[18] B. Ford, "Parsing expression grammars: a recognition-based syntactic foundation", *Proceeding of the 31st ACM SIGPLAN-SIGACT Symposium of Principles of Programming Languages* (POPL 2004, Venice, Italy, 14-16 January 2004), 111–122.

[19] W. Fraczak, A. Podolak, "A characterization of s-languages", *Information Processing Letters*, 89:2, 65–70 (2004).

[20] S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.

[21] J. B. Goodenough, "Exception handling: issues and a proposed notation", *Communications of the ACM*, 18:12, 683–696 (1975).

[22] R. Hausser, *Foundations of computational linguistics: human-computer communication in natural language*, Springer (2014).

[23] R. Heckmann, "An efficient ELL(1)-parser generator", *Acta Informatica*, 23:2, 127–148 (1986).

[24] S. Heilbrunner, "On the definition of ELR(k) and ELL(k) grammars", *Acta Informatica*, 11:2, 169–176 (1979).

[25] S. Heilbrunner, L. Schmitz, "An efficient recognizer for the Boolean closure of context-free languages", *Theoretical Computer Science*, 80 (1991), 53–75.

[26] P. Holmes, I. Hinchliffe, *Swedish: a comprehensive grammar* (2013).

[27] O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.

[28] O. H. Ibarra, S. M. Kim, S. Moran, "Sequential machine characterizations of trellis and cellular automata and applications", *SIAM Journal on Computing*, 14:2 (1985), 426–447.

[29] S. Jarząbek, T. Krawczyk, "LL-regular grammars", *Information Processing Letters*, 4 (1975), 31–37.

[30] K. Jensen, N. Wirth, "Pascal user manual and report – ISO Pascal Standard", 4th ed., Springer-Verlag (1991), 1–266.

[31] A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.

[32] A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Theory of Computing Systems*, 46:1 (2010), 27–58.

[33] A. Jeż, A. Okhotin, "Complexity of equations over sets of natural numbers", *Theory of Computing Systems*, 48:2 (2011), 319–342.

[34] A. Jeż, A. Okhotin, "One-nonterminal conjunctive grammars over a unary alphabet", *Theory of Computing Systems*, 49:2 (2011), 319–342.

[35] A. K. Joshi, L. S. Levy, M. Takahashi, "Tree adjunct grammars", *Journal of Computer and System Sciences*, 10:1 (1975), 136–163.

[36] A. K. Joshi, K. Vijay-Shanker, D. J. Weir, "The convergence of mildly context-sensitive grammatical formalisms", In: Sells, Shieber, Wasow (Eds.), *Foundational Issues in Natural Language Processing*, 1991.

[37] T. Kasami, K. Torii, "A syntax-analysis procedure for unambiguous context-free grammars", *Journal of the ACM*, 16:3 (1969), 423–431.

[38] S. Kearns, "Extending regular expressions with context operators and parse extraction", *Software: Practice and Experience* 21:8 (1991), 787–804.

[39] J. R. Kipps, "GLR parsing in time $\mathcal{O}(n^3)$", in: M. Tomita (Ed.), *Generalized LR Parsing*, Kluwer, 1991, 43–59.

[40] D. E. Knuth, "On the translation of languages from left to right", *Information and Control*, 8 (1965), 607–639.

[41] D. E. Knuth, "Top-down syntax analysis", *Acta Informatica*, 1, 79–110 (1971).

[42] D. E. Knuth, "The genesis of attribute grammars", In: P. Deransart, M. Jourdan (Eds.): *Attribute Grammars and their Applications* (International Conference WAGA, Paris, France, September 19–21, 1990) LNCS 461 (1990), 1–12.

[43] A. Korenjak, J. E. Hopcroft, "Simple deterministic languages", in: *Seventh Annual IEEE Switching and Automata Theory Conference*, 36–46 (1966).

[44] V. Kountouriotis, Ch. Nomikos, P. Rondogiannis, "Well-founded semantics for Boolean grammars", *Information and Computation*, 207:9 (2009), 945–967.

[45] R. Kowalski, *Logic for Problem Solving*, North-Holland, Amsterdam, 1979.

[46] R. Kurki-Suonio, "Notes on top-down languages", *BIT Numerical Mathematics*, 9:3, 225–238 (1969).

[47] B. Lang, "Deterministic techniques for efficient non-deterministic parsers", *ICALP 1974*, LNCS 14, 255–269.

[48] P. M. Lewis II, R. E. Stearns, "Syntax-directed transduction", *Journal of the ACM*, 15:3, 465–488 (1968).

[49] P. Naur, J. W. Backus, J. H. Wegstein, A. van Wijngaarden, M. Woodger, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, "Revised report on the algorithmic language ALGOL 60", *Commucations of the ACM*, 6:1 (1963), 1–17.

[50] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[51] A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28:5 (2002), 243–249.

[52] A. Okhotin, "LR parsing for conjunctive grammars", *Grammars*, 5 (2002), 81–124.

[53] A. Okhotin, "A recognition and parsing algorithm for arbitrary conjunctive grammars", *Theoretical Computer Science*, 302:1–3 (2003), 365–399.

[54] A. Okhotin, "On the closure properties of linear conjunctive languages", *Theoretical Computer Science*, 299:1–3 (2003), 663–685.

[55] A. Okhotin, "WhaleCalf, a parser generator for conjunctive grammars", *Implementation and Application of Automata*, LNCS 2608, 127–140 (2003).

[56] A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1, 19–48 (2004).

[57] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.

[58] A. Okhotin, "On the existence of a Boolean grammar for a simple programming language", *Automata and Formal Languages* (Proceedings of AFL 2005, 17–20 May 2005, Dobogókő, Hungary).

[59] A. Okhotin, "The dual of concatenation", *Theoretical Computer Science*, 345:2–3 (2005), 425–447.

[60] A. Okhotin, "Generalized LR parsing algorithm for Boolean grammars", *International Journal of Foundations of Computer Science*, 17:3 (2006), 629–664.

[61] A. Okhotin, "Recursive descent parsing for Boolean grammars", *Acta Informatica*, 44:3–4 (2007), 167–189.

[62] A. Okhotin, "Unambiguous Boolean grammars", *Information and Computation*, 206 (2008), 1234–1247.

[63] A. Okhotin, "Homomorphisms preserving linear conjunctive languages", *Journal of Automata, Languages and Combinatorics*, 13:3–4 (2008), 299–305.

[64] A. Okhotin, C. Reitwießner, "Conjunctive grammars with restricted disjunction", *Theoretical Computer Science*, 411:26–28 (2010), 2559–2571.

[65] A. Okhotin, "A simple P-complete problem and its language-theoretic representation", *Theoretical Computer Science*, 412:1–2 (2011), 68–82.

[66] A. Okhotin, "Expressive power of LL(k) Boolean grammars", *Theoretical Computer Science*, 412:39, 5132–5155 (2011).

[67] A. Okhotin, P. Rondogiannis, "On the expressive power of univariate equations over sets of natural numbers", *Information and Computation*, 212 (2012), 1–14.

[68] A. Okhotin, "Conjunctive and Boolean grammars: the true general case of the context-free grammars", *Computer Science Review*, 9 (2013), 27–59.

[69] A. Okhotin, "Improved normal form for grammars with one-sided contexts", *Descriptional Complexity of Formal Systems* (DCFS 2013, London, Ontario, Canada, 22-25 July 2013), LNCS 8031, 205–216.

[70] A. Okhotin, "Parsing by matrix multiplication generalized to Boolean grammars", *Theoretical Computer Science*, 516 (2014), 101–120.

[71] T. Parr, R. Quong, "ANTLR: A predicated LL(k) parser generator", *Software: Practice and Experience*, 25:7, 789–810 (1995).

[72] T. Parr, K. Fisher, "LL(*): the foundation of the ANTLR parser generator", *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, (PLDI 2011, San Jose, CA, USA, 4-8 June 2011) PLDI 2011, 425–436.

[73] F. C. N. Pereira, D. H. D. Warren, "Parsing as deduction", *ACL 1983*, 137–144.

[74] M. Rabkin, "Recognizing two-sided contexts in cubic time", *Computer Science—Theory and Applications* (CSR 2014, Moscow, Russia, 6–12 June 2014), LNCS 8476, 314–324.

[75] S. Rajasekaran, S. Yooseph, "TAL recognition in $\mathcal{O}(M(n^2))$ time", *Journal of Computer and System Sciences*, 56:1 (1998), 83–89.

[76] R. R. Rędziejowski, "Parsing expression grammar as a primitive recursive-descent parser with backtracking" *Fundamenta Informaticae*, 79:3–4, 513–524 (2007).

[77] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

[78] D. J. Rosenkrantz, R. E. Stearns, "Properties of deterministic top-down grammars", *Information and Control*, 17, 226–256 (1970).

[79] W. C. Rounds, "LFP: A logic for linguistic descriptions and an analysis of its complexity", *Computational Linguistics*, 14:4 (1988), 1–9.

[80] R. W. Sebesta, "Concepts of programming languages", 9th ed., Boston: Pearson Addison-Wesley, 2010.

[81] H. Seki, T. Matsumura, M. Fujii, T. Kasami, "On multiple context-free grammars", *Theoretical Computer Science*, 88:2 (1991), 191–229.

[82] H. J. Shyr, "Free monoids and languages", Lecture Notes, National Chung-Hsing University, Taichung (1991).

[83] K. Sikkel, *Parsing Schemata*, Springer-Verlag, 1997.

[84] S. Sippu, E. Soisalon-Soininen, "Parsing theory", Vol. 1: Languages and parsing, *EATCS monographs on theoretical computer science*, Springer, Berlin (1988).

[85] I. H. Sudborough, "A note on tape-bounded complexity classes and linear context-free languages", *Journal of the ACM*, 22:4 (1975), 499–500.

[86] P. Tapanainen, "Applying a finite-state intersection grammar", in: E. Roche, Y. Schabes (Eds.), *Finite-State Language Processing*, 1997, 311–327.

[87] V. Terrier, "On real-time one-way cellular array", *Theoretical Computer Science*, 141:1–2 (1995), 331–335.

[88] M. Tomita, *Efficient Parsing for Natural Language*, Kluwer, 1986.

[89] M. Tomita, "An efficient augmented context-free parsing algorithm", *Computational Linguistics*, 13:1 (1987), 31–46.

[90] I. Törmä, personal communication, February 2013.

[91] L. G. Valiant, "General context-free recognition in less than cubic time", *Journal of Computer and System Sciences*, 10:2 (1975), 308–314.

[92] K. Vijay-Shanker, D. J. Weir, A. K. Joshi, "Characterizing structural descriptions produced by various grammatical formalisms", *25th Annual Meeting of the Association for Computational Linguistics* (ACL 1987), 104–111.

[93] A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, C. H. A. Koster, M. Sintzoff, C. H. Lindsey, L. G. L. T. Meertens, R. G. Fisker, "Revised report on the algorithmic language ALGOL 68", *Acta Informatica*, 5 (1975), 1–236.

[94] D. Wood, "A further note on top-down deterministic languages", *Computer Journal*, 14:4, 396–403 (1971).

[95] D. Wotschke, "The Boolean closures of deterministic and nondeterministic context-free languages", In: W. Brauer (ed.), *Gesellschaft für Informatik e. V., 3. Jahrestagung 1973*, LNCS 1, 113–121.

[96] S. Yu, "A property of real-time trellis automata", *Discrete Applied Mathematics*, 15:1 (1986), 117–119.

[97] R. Zier-Vogel, M. Domaratzki, "RNA pseudoknot prediction through stochastic conjunctive grammars", *Computability in Europe 2013. Informal Proceedings*, 80–89.

Index

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyöttiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Faculty of Science and Engineering
- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics
- Information Systems

ISBN 978-952-12-3261-9
ISSN 1239-1883

Mikhail Barash

Defining Contexts in Context-Free Grammars

Mikhail Barash

Defining Contexts in Context-Free Grammars

Mikhail Barash

Defining Contexts in Context-Free Grammars