VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

CANONICAL DERIVATIONS IN PROGRAMMED GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR ZEMEK
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KANONICKÉ DERIVACE PROGRAMOVANÝCH GRAMATIK
CANONICAL DERIVATIONS IN PROGRAMMED GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR ZEMEK
AUTHOR

VEDOUCÍ PRÁCE Prof. RNDr. ALEXANDER MEDUNA, CSc.
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/6572/2007/xzemek02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Zemek Petr**
Obor: Informační technologie
Téma: **Kanonické derivace programovaných gramatik**
Kategorie: Teorie informatiky

Pokyny:

1. Seznamte se s programovanými gramatikami.
2. Studujte kanonické derivace v těchto gramatikách. Zaměřte se na levé derivace.
3. Demonstrujte, že rozsah levého omezení derivací ovlivňuje generativní sílu gramatik.
4. Objasněte význam dosažených výsledků pro syntaktickou analýzu.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A.: Automata and Languages, Springer, London, 2000

Při obhajobě semestrální části projektu je požadováno:

- Body 1-2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

[illegible]

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení:	**Petr Zemek**
Id studenta:	79274
Bytem:	Malá Strana 10, 747 62 Mokré Lazce
Narozen:	13. 12. 1985, Opava

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

....

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP:	Kanonické derivace programovaných gramatik
Vedoucí/školitel VŠKP:	Meduna Alexander, prof. RNDr., CSc.
Ústav:	Ústav informačních systémů
Datum obhajoby VŠKP:	....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické formě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - [✓ ihned po uzavření této smlouvy]
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ....

....	[illegible] ....
Nabyvatel	Autor

Abstrakt

V této práci jsou studovány kanonické derivace (se zaměřením na nejlevější derivace) v programovaných gramatikách a rozsah levého omezení. Je ukázáno, že zavedením n-limitovaných derivací v programovaných gramatikách tak, jako byly zavedeny pro stavové gramatiky, dostaneme nekonečnou hierarchii jazykových tříd vyplývající z n-limitovaných programovaných gramatik, takže rozsah levého omezení ovlivňuje generativní sílu n-limitovaných programovaných gramatik. Tento výsledek má význam pro syntaktickou analýzu založenou na programovaných gramatikách.

Klíčová slova

Programovaná gramatika, kanonické derivace, nejlevější derivace, n-limitované derivace, stavová gramatika, neomezená stavová gramatika, nekonečná hierarchie jazykových tříd.

Abstract

This work studies canonical derivations (with focus on leftmost derivations) in programmed grammars and left restriction range. It is shown that if we introduce n-limited derivations in programmed grammars as they were defined for state grammars, we get an infinite hierarchy of language families resulting from n-limited programmed grammars, so the left restriction range affects the generative power of n-limited programmed grammars. This result is significant for syntactical analysis based on programmed grammars.

Keywords

Programmed grammar, canonical derivations, leftmost derivations, n-limited derivations, state grammar, unrestricted state grammar, infinite hierarchy of language families.

Citace

Petr Zemek: Canonical Derivations in Programmed Grammars, bakalářská práce, Brno, FIT VUT v Brně, 2008

Canonical Derivations in Programmed Grammars

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Alexandra Meduny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
Petr Zemek
May 10, 2008

Poděkování

Na tomto místě bych rád poděkoval mému vedoucímu prof. Alexandru Medunovi za odborné vedení, za poskytnuté rady a konzultace a za ochotu a čas, který mi při tvorbě práce věnoval.

© Petr Zemek, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

1 Introduction **2**

2 Preliminaries **3**
- 2.1 Mathematical Background . . . 3
  - 2.1.1 Sets . . . 3
  - 2.1.2 Relations and Functions . . . 4
- 2.2 Alphabet, Words and Languages . . . 5
- 2.3 Grammars and Language Families . . . 6
  - 2.3.1 Phrase-structure Grammar (Type 0) . . . 6
  - 2.3.2 Context-sensitive Grammar (Type 1) . . . 7
  - 2.3.3 Context-free Grammar (Type 2) . . . 7
  - 2.3.4 Language Families and Chomsky Classification . . . 7

3 Programmed Grammars **9**
- 3.1 Programmed Grammar . . . 9
- 3.2 Programmed Grammar With ε-rules . . . 11
- 3.3 Programmed Grammar With Appearance Checking . . . 11
- 3.4 Generative Power . . . 12

4 Canonical Derivations **14**
- 4.1 Leftmost Derivations of Type 1 . . . 15
- 4.2 Leftmost Derivations of Type 2 . . . 15
- 4.3 Leftmost Derivations of Type 3 . . . 17

5 Left Restriction Range and Its Consequences **18**
- 5.1 State Grammars . . . 18
  - 5.1.1 State Grammar . . . 18
  - 5.1.2 Unrestricted State Grammar . . . 19
- 5.2 n-limited Derivations . . . 20
- 5.3 Results . . . 20
  - 5.3.1 Equivalence of $\mathscr{L}_{n\text{-}lim}(PG)$ and $\mathscr{L}_{n\text{-}lim}({}_uSG)$ . . . 20
  - 5.3.2 Infinite Hieararchy of Language Families . . . 24
- 5.4 Significance for Syntactical Analysis . . . 26

6 Conclusion **28**
- 6.1 Open Problems . . . 28

Chapter 1

Introduction

In the formal language theory, especially in the area of parsing and compilers, there always has been a tendency to create a model, which will be based on context-free rules (because they are simple and easy to use), but will have a higher generative power than context-free grammars. Sure, one can use context-sensitive grammars or even unresticted grammars, but these models are too complex for the use in syntactical analysis and related fields.

In 1967, D. Rosenkratz introduced the programmed grammar [15, 16]. This grammar has only context-free rules, however, it regulates the derivation process (sequential application of rules) by restricting and specifying which rules can be used after some particular rule is used. It was shown that by restricting and regulating the derivation process this way the generative capacity highly increases, so we are able to generate all languages that are accepted by Turing machines.

Many other types of regulation were studied, for example S. Abraham proposed the matrix grammar [1] and the random context grammar was introduced [19]. These three types of regulation were investigated in details in [5]. At first, there was an enthusiasm for these grammars because the advantages were obvious. However, after few years, a slight “disillusion” appeared because it was proved that if we use only canonical (leftmost or rightmost) derivations, we decrease the generative power of programmed grammars to the family of languages generated by context-free grammars [5].

So, a natural question arises – why then use this kind of a regulation (programmed grammar) if we loose the advantage of higher generative power and get the same as by using context-free grammars?

The main goal of this work is to reduce the effect of this restriction and “improve” this result by studying the left restriction range and n-limited derivations. In Chapter 2, basic mathematical and grammar preliminaries are given. Then, in Chapter 3, the programmed grammar is defined along with its modifications and summary of the generative power of these grammars. Cannonical derivations in programmed grammars, with focus on leftmost derivations, are discussed in Chapter 4. Main results are presented in Chapter 5, where left restriction, n-limited derivations and its consequences are studied. Also, a significance for syntactical analysis is given. Then, in the last Chapter 6, obtained results are sumarized, open problems are given and possible further project research is discussed.

Chapter 2

Preliminaries

In this chapter, I will describe used terminology and remaind fundamental terms from the area of formal language theory. Section 2.1 describes the fundamental mathematical notations, concepts and techniques used in my work. Then, in Section 2.2, the meaning of an alphabet, words, languages and operations over them is given. Finally, Section 2.3 reviews the basic of grammars, which are classified into the Chomsky classification. This whole chapter is based on [12, 17, 10, 14], so for further information, please consult these books.

2.1 Mathematical Background

2.1.1 Sets

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written as $x \notin S$. The *cardinality* of S, denoted by $|S|$, is the number of members of S. The set that has no members is the *empty set*, denoted by $\emptyset$; note that $|\emptyset| = 0$. If a set S has a finite number of members, then S is a *finite set*. Otherwise, S is an *infinite set*.

A set S is specified by enclosing some description of its elements in curly brackets; for example, the set of three integers (1, 2 and 3) is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \dots, z\}$ stands for all the lowercase letters of the English alphabet. When the need arises, we use more explicit notation, in which a set S is specified by a property π so that S contains all elements satisfying π and this specification has the following format:

$$S = \{x \colon \pi(x)\}.$$

For example, the set of all even integers can be denoted by $S = \{i \colon i \text{ is even}\}$.

The usual set operations are *union* ($\cup$), *intersection* ($\cap$) and *difference* ($-$), defined as

$$\begin{aligned} S_1 \cup S_2 &= \{x\colon x \in S_1 \text{ or } x \in S_2\}, \\ S_1 \cap S_2 &= \{x\colon x \in S_1 \text{ and } x \in S_2\}, \\ S_1 - S_2 &= \{x\colon x \in S_1 \text{ and } x \notin S_2\}. \end{aligned}$$

Another basic operation is *complementation.* The complement of a set S is denoted by $\overline{S}$. To make this meaningful, we need to know what the *universal set* U of all possible elements is. If U is specified, then

$$\overline{S} = \{x\colon x \in U, x \notin S\}.$$

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not is S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If two sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \emptyset$, then the sets are said to be *disjoint.* If $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$, then S_1 and S_2 are said to be *equivalent* and we write $S_1 = S_2$. Otherwise, they are said to be *nonequivalent* and we write $S_1 \neq S_2$. A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets and sets of this kind are customarily called *families* of sets, rather than set of sets.

When the elements of a set S are ordered sequences of elements from other sets, such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of n sets, which itself is a set of ordered *n-tuples*, we write

$$S = S_1 \times S_2 \times \cdots \times S_n = \{(s_1, s_2, \ldots, s_n)\colon s_1 \in S_1, s_2 \in S_2, \ldots, s_n \in S_n\}.$$

2.1.2 Relations and Functions

An *n-ary relation* δ over the sets $S_1, S_2, \ldots, S_n$ is any subset of their Cartesian product, that is,

$$\delta \subseteq S_1 \times S_2 \times \cdots \times S_n.$$

A *binary relation* ρ from S_1 to S_2, which is a special case of an n-ary relation for $n = 2$, is any subset of $S_1 \times S_2$, that is,

$$\rho \subseteq S_1 \times S_2.$$

If $S_1 = S_2$, then we usually say that ρ is a *relation on* S_1 or *relation over* S_1. Because of simplicity, I will consider only binary relations throughout this subsection. Properties of n-ary relations would be defined accordingly.

A *function* from S_1 to S_2 is a relation ϕ from S_1 to S_2 such that for every $x \in S_1$,

$$|\{y \colon y \in S_2, (x, y) \in \phi\}| \leq 1.$$

Let ϕ be a function from S_1 to S_2. If for every $y \in S_2, |\{x \colon x \in S_1, (x, y) \in \phi\}| \leq 1$, ϕ is an *injection.* If for every $y \in S_2, |\{x \colon x \in S_1, (x, y) \in \phi\}| \geq 1$, ϕ is a *surjection.* If ϕ is both a surjection and an injection, ϕ represents a *bijection.*

Instead of $(x, y) \in \rho$, we often write $x \in \rho(y)$ or $x\rho y$; in other words, $(x, y) \in \rho$, $x\rho y$, and $x \in \rho(y)$ are used interchangeably. If ρ is a function, we usually write $x = \rho(y)$. Let ρ be a relation over a set S_2. For $k \geq 1$, the k-fold product of ρ, ρ^k, is recursively defined as

$$\begin{array}{ll} (1) & x\rho^1 y \text{ if and only if } x\rho y, \\ (2) & x\rho^k y \text{ if and only if } x\rho z \text{ and } z\rho^{k-1} y, \end{array}$$

for some z and $k \geq 2$. The *transitive closure* of ρ, ρ^+, is defined as $x\rho^+ y$ if and only if $x\rho^k y$ for some $k \geq 1$, and the *reflexive and transitive closure* of ρ, ρ^*, is defined as $x\rho^* y$ if and only if $x\rho^k y$ for some $k \geq 0$.

2.2 Alphabet, Words and Languages

An *alphabet* is a finite, nonempty set of elements, which are called *symbols.* A sequence of symbols forms a *word* (*string* is a synonymous with *word*). The empty word, denoted by ε, is the word that contains no symbols; note that $|\varepsilon| = 0$.

Let x and y be two words over an alphabet Σ. Then, $x.y$, usually written as xy, is the *concatenation* of x and y. For an alphabet, Σ, Σ^* denotes the set of all strings over Σ (including ε). Set $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Let x be a word over an alphabet Σ. The *length* of x is the number of all symbols in x and it is denoted by $|x|$.

Let n be a nonnegative integer and x a word over an alphabet Σ. Then x^n is a word over Σ defined by

$$\begin{array}{ll} (1) & x^0 = \varepsilon, \\ (2) & x^n = xx^{n-1}. \end{array}$$

A *language* L over Σ is a set of words over Σ. The *empty language* is denoted by $\emptyset$. The *universal language* over Σ, which is the language consisting of all words over Σ, is Σ^*. For a language L, we denote by $|L|$ the cardinality of L. The *concatenation* of two languages $L_1, L_2 \subseteq \Sigma^*$, denoted by $L_1.L_2$ and usually written as L_1L_2, is the set $L_1L_2 = \{w_1w_2 \colon w_1 \in L_1 \text{ and } w_2 \in L_2\}$.

For an integer $n \geq 0$ and a language L, the nth power of L, denoted by L^n, is defined by

$$\begin{array}{ll} (1) & L^0 = \{\varepsilon\}, \\ (2) & L^n = L^{n-1}L. \end{array}$$

The star (*Kleene closure*) of a language L, denoted by L^*, is the set

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

Similarly, we define

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

2.3 Grammars and Language Families

This section reviews the basics of grammars. Specifically, it provides definitions of phrase-structure, context-sensitive, and context-free grammars along with language families they generate, which are classified into the Chomsky classification.

2.3.1 Phrase-structure Grammar (Type 0)

Definition 2.3.1. A *phrase-structure* or a *type* 0 Chomsky grammar (see [12]) is a quadruple

$$G = (N, T, P, S),$$

where

N is an alphabet of *nonterminals*;

T is an alphabet of *terminals* ($N \cap T = \emptyset$);

$S \in N$ is the starting nonterminal;

P is a finite relation from $(N \cup T)^* N (N \cup T)^*$ to $(N \cup T)^*$.

Pairs $(u, v) \in P$ are called *productions* or *rewriting rules* (abbreviated *rules*) and are written as $r\colon u \to v$, where r is a *rule label* associated with this rule. Accordingly, P is called the *set of all rules* in G. The *total alphabet* of G is $V = N \cup T$. A rewriting rule $r\colon u \to v \in P$, which holds $v = \varepsilon$, is called an *ε-rule* or an *erasing rule.* If there is no such rule in P, then we say that the grammar is *ε-free.* Let $lab(P)$ denote the set of all labels of rules from P.

The *direct derivation* relation induced by G is a binary relation between words over V^*, denoted by $\Rightarrow_G$, and defined as

$$\alpha \Rightarrow_G \beta \; [r],$$

if and only if $\alpha = xuy, \beta = xvy$ and $r\colon u \to v \in P$, where $\alpha, \beta, x, y \in V^*$. When no confusion exists, we simplify $\alpha \Rightarrow_G \beta \; [r]$ to $\alpha \Rightarrow \beta$. The *derivation relation* induced by G, denoted $\Rightarrow_G^*$, is the reflexive and transitive closure of the relation $\Rightarrow_G$. Further, $\Rightarrow_G^+$ denotes the transitive closure of $\Rightarrow_G$. If $S \Rightarrow_G^* x$ for some $x \in V^*$, x is called a *sentential form.*

The *language* generated by G, denoted by $L(G)$, is

$$L(G) = \{w \colon w \in T^*, S \Rightarrow_G^* w\}.$$

The family that is generated by this type of grammar is the family of all *recursively-enumerable languages*, denoted by **RE**.

2.3.2 Context-sensitive Grammar (Type 1)

Definition 2.3.2. A *context-sensitive* (*type* 1) *grammar* (see [12]) is a type 0 grammar,

$$G = (N, T, P, S),$$

such that each rule $u \rightarrow v \in P$ satisfies $|u| \leq |v|$. The direct derivation relation, derivation relation and language generated by this grammar is defined in the same manner as in the phrase-structure grammar.

A *context-sensitive language* is a language generated by a context-sensitive grammar. The family of all context-sensitive languages is denoted by **CS**.

2.3.3 Context-free Grammar (Type 2)

Definition 2.3.3. A *context-free* (*type* 2) *grammar* (see [12]) is a type 0 grammar,

$$G = (N, T, P, S),$$

such that each rule $u \rightarrow v \in P$ satisfies $u \in N$ ($|u| = 1$). The direct derivation relation, derivation relation and language generated by this grammar is defined in the same manner as in the phrase-structure grammar.

A *context-free language* is a language generated by a context-free grammar. The family of all context-free languages is denoted by **CF**.

2.3.4 Language Families and Chomsky Classification

Noam Chomsky, a founder of formal language theory, provided an initial classification into four language types, type 0 to type 3 [2]. Grammars of type 0 to type 2 were defined in the previous sections and the grammar of type 3 is omitted on purpose, because it is not related to this work. Only note that this type of a grammar is called *right-linear* and generates the family of all *regular languages*, denoted by **REG**.

For the families of languages generated by right-liner, context-free, context-sensitive and phrase-structure grammars, it holds:

Theorem 2.3.4. *(See [12])* $\boldsymbol{REG} \subset \boldsymbol{CF} \subset \boldsymbol{CS} \subset \boldsymbol{RE}$.

Thus, each language family of type i is a proper subset of the family of type $i - 1$. A diagram 2.1 exhibits the relation clearly; it shows the original Chomsky hierarchy.

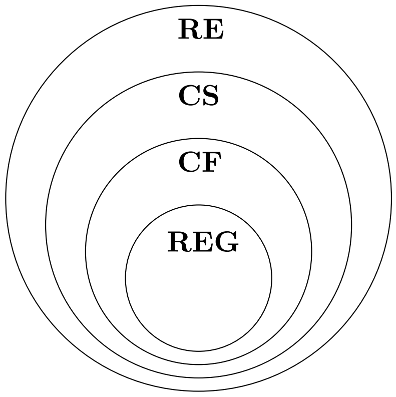

Figure 2.1: The Chomsky Hierarchy of Languages.

If some grammars define the same language, they are referred to as *equivalent grammars*.

Chapter 3

Programmed Grammars

Programmed grammar (in its basic form) was introduced by D. Rosenkratz in 1967 in his PhD thesis [15] and further studied in [16]. The main idea was to create a grammar originating from a context-free grammar (because context-free rules are simple and easy to use), but control the derivation process by prescribing the order in which individual rules are applied, thus obtaining a kind of contextual dependance of rules, which results into higher generative power of these grammars.

Derivation process is controled in the following way. Each context-free rule has assigned a set of rules (to be precise, rule labels) and after applying some rule, the next rule must be chosen from the assigned set of the last applied rule. I will, informally, show an example. Let ABC (A, B and C are nonterminals) be a sentential form derived by some programmed grammar. Then, the nonterminal A is rewritten to aA (a is a terminal), thus obtaining $aABC$. We would like to generate the string $aabbcc$, so we prescribe that B must be rewriten to bB after rewriting A and C must be rewritten to cC after B is rewritten. Note that this is not possible in a context-free grammar, because there is no restriction which rule should be used in the next step, so aA could be used again and again, thus obtaining, for example, $aaaaaaabbcc$, which is definitely not what we wanted.

Formal definition of a programmed grammar is given in Section 3.1, with the full example of a grammar that was informally presented in the previous paragraph. In Sections 3.2 and 3.3, two modifications of a programmed grammar are defined. Finally, in Section 3.4, the generative power (in context of the Chomsky classification) of all defined types of a programmed grammar is shown.

3.1 Programmed Grammar

Definition 3.1.1. A *programmed grammar* (see [5]) is a quadruple

$$G = (N, T, P, S),$$

where

N, T and S are as in a context-free grammar ($V = N \cup T$ is the total alphabet);

$P \subseteq L \times N \times V^+ \times 2^L$ is a finite relation, where L is a finite set of labels so that $lab(P) = L$.

An element $(r, A, v, \sigma(r)) \in P$ is (as in a phrase-structure grammar) called a rewriting production or a rewriting rule (abbreviated rule) and is usually written as $(r\colon A \to v, \sigma(r))$, where $r\colon A \to v$ is a context-free rule labeled by r and $\sigma(r)$ is a set of rule labels associated with this rule called *success field.*

Given a programmed grammar $G = (N, T, P, S)$, let $\Rightarrow$ be a relation on V^+ defined as follows: Let $x, y \in V^+$ be two sentential forms in G, then we write $x \Rightarrow y$ if and only if either

a) $x = S$, $y = v$, no rules have been applied and there is a rule $(r\colon S \to v, \sigma(r))$ in P for some $v \in V^+$.

or

b) $x = x_1Ax_2$, $y = x_1vx_2$, $A \in N$, $x_1, x_2 \in V^*$, last applied rule was r and there is a rule $(s\colon A \to v, \sigma(s)) \in P$ for some $v \in V^+$, where $s \in \sigma(r)$.

For $x, y \in V^+$, write $x \Rightarrow^* y$ if either $x = y$ or there exists $x_0, \dots, x_k$ such that $x_0 = x$, $x_k = y$, and $x_i \Rightarrow x_{i+1}$ for each i, $0 \leq i \leq k-1$. The sequence $x_0, \dots, x_k$ is called a derivation (of length k) and is denoted by $x_0 \Rightarrow \cdots \Rightarrow x_k$. Thus, $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$ and we define $\Rightarrow^+$ to be the transitive closure of $\Rightarrow$.

The language generated by G is

$$L(G) = \{w\colon w \in T^+, S \Rightarrow^* w\}.$$

Example 3.1.2. Let $G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$ [20] be a programmed grammar with the following rules in P:

$(r_1\colon S \to ABC, \{r_2, r_5\})$,

$(r_2\colon A \to aA, \{r_3\})$,

$(r_3\colon B \to bB, \{r_4\})$,

$(r_4\colon C \to cC, \{r_2, r_5\})$,

$(r_5\colon A \to a, \{r_6\})$,

$(r_6\colon B \to b, \{r_7\})$,

$(r_7\colon C \to c, \emptyset)$.

This grammar generates the noncontext-free language $\{a^nb^nc^n\colon n \geq 1\}$. Note that $\sigma(r_7) = \emptyset$, because after the rule r_7 is used, there are no nonterminals left in a sentential form.

The string $aabbcc$ is produced via the following derivation: $S \Rightarrow ABC$ $[r_1] \Rightarrow aABC$ $[r_2]$ $\Rightarrow aAbBC$ $[r_3] \Rightarrow aAbBcC$ $[r_4] \Rightarrow aabBcC$ $[r_5] \Rightarrow aabbcC$ $[r_6] \Rightarrow aabbcc$ $[r_7]$.

There are two special types of programmed grammars, which can be also considered. However, they are presented here only for completeness and that there will be references to them in Section 3.4.

3.2 Programmed Grammar With ε-rules

In the original definition of a programmed grammar, rules are of the form $(r\colon A \to v, \sigma(r))$, where $v \in V^+$, which disallowes ε-rules. Hence, a programmed grammar with ε-rules is defined the same way as a programmed grammar, but with $v \in V^*$.

Definition 3.2.1. A *programmed grammar with ε-rules* (see [5]) $G = (N, T, P, S)$ is a programmed grammar where P is defined as a finite relation $P \subseteq L \times N \times V^* \times 2^L$ (L is, as i a programmed grammar, a finite set of labels so that $lab(P) = L$).

3.3 Programmed Grammar With Appearance Checking

In a programmed grammar, it could be sometimes desirable to rewrite, for example, all occurences of a nonterminal A in a sentential form, and, after every A is rewritten, continue rewriting by using another rule. Or check, whether there is a nonterminal B in a sentential form and if so, use some rule, otherwise, use another rule. However, no such actions are possible in programmed grammers, because once you cannot use any rule from the success field, the derivation ends ("get stuck"). Hence, to allow such actions, a new modification of a programmed grammar was introduced, called *programmed grammar with appearance checking*.

Informally, a programmed grammar with appearance checking is a programmed grammar with another set associated with each rule, called *failure field*. In a derivation, you have to apply rules from the success field, but if you cannot apply any such rule, you must apply some rule from the failure field. Therefore, if you want to rewrite all occurences of a nonterminal A in a sentential form, you introduce a rule $(r_1\colon A \to v, \{r_1\}, \{r_2\})$, so you have to use this rule repeatedly until no A can be rewritten (that means the application of the rule r_1 *failed*), so you will use the rule r_2 from the failure field. Similarly, to check whether there is a nonterminal B in a sentential form, introduce a rule $(q_1\colon B \to B, \{q_2\}, \{q_3\})$. Using this rule, you can perform that check for the *appearance* of B, and, depending on the result, use q_2 or q_3, respectively.

Definition 3.3.1. A *programmed grammar with appearance checking* (see [5]) is a quadruple $G = (N, T, P, S)$, where N, T and S are as in a context-free grammar ($V = N \cup T$ is the total alphabet) and $P \subseteq L \times N \times V^+ \times 2^L \times 2^L$ is a finite relation, where L is a finite set of labels so that $lab(P) = L$.

An element $(r, A, v, \sigma(r), \varphi(r)) \in P$ is (as in a programmed grammar) called a rule and is usually written as $(r\colon A \to v, \sigma(r), \varphi(r))$, where $r\colon A \to v$ and $\sigma(r)$ have the same meaning as in a programmed grammar and $\varphi(r)$ is another set of rule labels called *failure field*. However, if $\varphi(r) = \emptyset$ holds for every $r \in lab(P)$, then G is equal to a programmed grammar without appearance checking, because if we omit the failure field of all rules, we got a programmed grammar.

Given a programmed grammar with appearance checking $G = (N, T, P, S)$, let $\Rightarrow$ be a relation on V^+ defined as follows: Let $x, y \in V^+$ be two sentential forms in G. Then we write $x \Rightarrow y$ if and only if either

a) $x = S$, $y = v$, no rules have been applied and there is a rule $(r\colon S \to v, \sigma(r), \varphi(r)) \in P$ for some $v \in V^+$.

or

b) $x = x_1Ax_2$, $y = x_1vx_2$, $A \in N$, $x_1, x_2 \in V^*$, last applied rule was r and there is a rule $(s\colon A \to v, \sigma(s), \varphi(s)) \in P$ for some $v \in V^+$, where $s \in \sigma(r)$.

or

c) $x = y$, last applied rule was r, there is no rule $(s\colon A \to v, \sigma(s), \varphi(s)) \in P$, $s \in \sigma(r)$, that is applicable to x and in the next step we have to use some rule $t \in \varphi(r)$.

In other words, either the rule $r\colon A \to v$ is effectively used and then we use a rule with a label in $\sigma(r)$ in the next step, or $r\colon A \to v$ cannot be applied and we pass on to a rule with label in $\varphi(r)$ for the next step.

In standard manner, we define $\Rightarrow^m$, $\Rightarrow^*$ and $\Rightarrow^+$ for some $m \geq 0$. The language generated by the grammar G is $L(G) = \{w\colon w \in T^+, S \Rightarrow^* w\}$.

A *programmed grammar with appearance checking and with ε-rules* would be defined in the same way as a programmed grammar with ε-rules, so the definition is omitted.

Example 3.3.2. Consider a programmed grammar $G = (\{S, A\}, \{a\}, P, S)$ [5], where P consists of the rules

$$(r_1\colon S \to AA, \{r_1, r_2\}, \{r_2, r_3\}),$$

$$(r_2\colon A \to S, \{r_2\}, \{r_1\}),$$

$$(r_3\colon A \to a, \{r_3\}, \emptyset).$$

Because $\sigma(r_i) = r_i$ for $i = 1, 2, 3$, the rules $S \to AA$, $A \to S$ and $A \to a$ must be used as many times as possible. Therefore, starting from S^n for some $n \geq 1$, we must pass to A^{2n} and then, using the rule r_2, to S^{2n}, or using the rule r_3, to a^{2n}. Each such cycle consisting of use of r_1 and r_2 doubles the number of symbols. In conclusion, we obtain the noncontext-free language $L(G) = \{a^{2^n} : n \geq 1\}$.

The string $aaaa$ is produced via the following derivation: $S \Rightarrow AA$ $[r_1] \Rightarrow SA$ $[r_2] \Rightarrow SS$ $[r_2] \Rightarrow AAS$ $[r_1] \Rightarrow AAAA$ $[r_1] \Rightarrow aAAA$ $[r_3] \Rightarrow aaAA$ $[r_3] \Rightarrow aaaA$ $[r_3] \Rightarrow aaaa$ $[r_3]$.

3.4 Generative Power

It is natural to ask, what is the generative power of these grammars, in relation with the Chomsky classification of languages. Let $\mathscr{L}(PG)$ denote the family of languages generated by programmed grammars, $\mathscr{L}(PG, \varepsilon)$ denotes the family of languages generated by programmed grammars with ε-rules, $\mathscr{L}(PG, ac)$ denotes the family of languages generated by programmed grammars with appearance checking and $\mathscr{L}(PG, \varepsilon, ac)$ denotes the family of languages generated by programmed grammars with ε-rules and with appearance checking. Then, Diagram 3.1 holds [3].

If two families are connected by a dotted arrow (solid arrow), then the upper family includes (includes properly) the lower family; if two families are not connected then they are not necessarily incomparable.

$$\begin{tikzcd}
\mathbf{RE} \arrow[r, leftrightarrow] & \mathscr{L}(PG,\varepsilon,ac) \\
\mathbf{CS} \arrow[u] & \\
\mathscr{L}(PG,ac) \arrow[u] & \mathscr{L}(PG,\varepsilon) \arrow[uul] \\
\mathscr{L}(PG) \arrow[u] \arrow[ur, dotted] & \\
\mathbf{CF} \arrow[u] &
\end{tikzcd}$$

Figure 3.1: Generative Power of Programmed Grammars

From Diagram 3.1 it is clear that programmed grammars are more powerful than context-free grammars and programmed grammars with ε-rules and with appearance checking are as powerful as phrase-structure grammars. The relation between programmed grammars with and without ε-rules, respectively, is not yet known and it will be discussed in the conclusion (except the inclusion $\mathscr{L}(PG) \subseteq \mathscr{L}(PG,\varepsilon)$, which is trivial and follows directly from the definition of a programmed grammar with ε-rules).

Chapter 4

Canonical Derivations

In programmed grammars, there is no restriction on which nonterminal a rule has to be applied; any nonterminal which can be rewritten may be rewritten. If we restrict the derivation process to rewrite only extreme nonterminals, we obtain *leftmost* or *rightmost* derivations, respectively. It can be shown [12], that in context-free grammars, we can (without loss of generality) consider only leftmost or rightmost derivations. However, as will be discussed later, this is not true in programmed grammars. As it makes no difference in what concerns the generative capacity (and thus other properties) [5], only leftmost derivations will be discussed.

In [5], three basic types of leftmost derivations were considered. The presented results were improved, some open problems were solved and new types of leftmost derivations were considered in [4, 6]. I will here only consider, describe and summarize results of the basic three types of leftmost derivations with specialization on the first type, which has the highest importance for this work. These three types of leftmost derivations are discussed in Sections 4.1, 4.2 and 4.3, respectively. First, a basic idea (taken from [5]) of each derivation type will be presented, then an informal description (to better understand the differences between particular types), definition and theorems (how do particular types of leftmost derivations affect the generative power of programmed grammars) will follow.

I will also present an example of a programmed grammar and show how can particular types of leftmost derivations affect the generated language. The following programmed grammar is a transformed and modified version of the original matrix grammar presented in [11].

Example 4.0.1. Let $H = (\{S, A, C, X, Y\}, \{a, b, c, d\}, P, S)$ be a programmed grammar with the following rules in P:

$(r_1\colon S \to XACC, \{r_2, r_3, r_4\})$, $(r_5\colon C \to c, \{r_6\})$,

$(r_2\colon A \to aAb, \{r_3\})$, $(r_6\colon C \to Y, \{r_7\})$,

$(r_3\colon C \to cC, \{r_2, r_3, r_4\})$, $(r_7\colon X \to d, \{r_8\})$,

$(r_4\colon A \to ab, \{r_5\})$, $(r_8\colon Y \to d, \emptyset)$.

If we place no restriction on which nonterminal in a single derivation step must be rewritten, then this grammar generates the language $L(H) = \{da^n b^n c^i dc^j \colon n \geq 1,\ (i \geq 0 \text{ and } j \geq 1) \text{ or } (i \geq 1 \text{ and } j \geq 0),\ i + j \geq n\}$.

The string *dabcdcc* is produced via the following derivation: $S \Rightarrow XACC$ $[r_1]$ $\Rightarrow XAcCC$ $[r_3]$ $\Rightarrow XAcCcC$ $[r_3]$ $\Rightarrow XabcCcC$ $[r_4]$ $\Rightarrow XabcCcc$ $[r_5]$ $\Rightarrow XabcYcc$ $[r_6]$ $\Rightarrow dabcYcc$ $[r_7]$ $\Rightarrow dabcdcc$ $[r_8]$.

Throughout following examples, the language generated by the grammar H under leftmost derivations of type k, $k \in \{1, 2, 3\}$, will be denoted by $L_{left\text{-}k}(H)$.

4.1 Leftmost Derivations of Type 1

Main idea. At each step of a derivation the leftmost occurence of a nonterminal has to be rewritten.

This (most restrictive) restriction is the same as in context-free grammars under leftmost derivations, so regardless of the set of rules from the success field, the leftmost occurence of a nonterminal has to be rewritten. This can result (as will be shown in Example 4.1.2) into situations, where there is no applicable rule with its left-hand side equal to the leftmost nonterminal.

Definition 4.1.1. Let G be a programmed grammar. A derivation in G is called *leftmost of type* 1 (see [5]) if each rule used in this derivation either rewrites the leftmost nonterminal occurence in the current string, or (this is important for appearence checking mode of derivation) it cannot be applied to this leftmost nonterminal occurence and then we say that the rule is not applicable to the whole string.

By $L_{left\text{-}1}(G)$ we denote the language generated by G in this way, and we denote the families of languages generated in this way by $\mathscr{L}(PG,\ left\text{-}1)$, $\mathscr{L}(PG,\ \varepsilon,\ left\text{-}1)$, $\mathscr{L}(PG,\ ac,\ left\text{-}1)$ and $\mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}1)$.

Example 4.1.2. Consider the programmed grammar H from Example 4.0.1. In every derivation, S will be first rewritten to $XACC$ by using the rule r_1, so X is the leftmost nonterminal in this sentential form. However, there is no rule $s \in \sigma(r_1)$ with X on its left-hand side, so the derivations ends here. Therefore, the language generated under leftmost derivations of type 1 by H is $L_{left\text{-}1}(H) = \emptyset$.

Theorem 4.1.3. *(See [6])*

$\mathscr{L}(PG,\ left\text{-}1) = \mathscr{L}(PG,\ \varepsilon,\ left\text{-}1) = \mathscr{L}(PG,\ ac,\ left\text{-}1) = \mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}1) = \boldsymbol{CF}$.

So, this leftmost restriction highly decreases the generative power of programmed grammars. In Chapter 5, in order to "improve" this result, I will consider left restriction range – that means, informally, during each derivation step, at most the nth nonterminal (from the left) must be rewritten for some $n \geq 1$.

4.2 Leftmost Derivations of Type 2

Main idea. At each step of a derivation the leftmost occurence of a nonterminal which can be rewritten (note that in programmed grammars only certain nonterminal occurences can be rewritten) has to be rewritten.

Contrary to leftmost derivations of type 1, this type of leftmost derivations is not that strict. If the leftmost nonterminal cannot be rewritten, we proceed to the second nonterminal from the left and so on, until we find the first nonterminal that can be rewritten and we rewrite it.

Definition 4.2.1. Let $G = (N, T, P, S)$ be a programmed grammar (a programmed grammar with appearance checking, respectively). We say that a derivation according to G is *leftmost of type 2* (see [5]) if it develops as follows:

a) Start with S and apply any rule $(r\colon S \to x, \sigma(r)) \in P$ $((r\colon S \to x, \sigma(r), \varphi(r)) \in P$, respectively).

b) If the current string y has been obtained from some string x by an effective use of some rule p, then at the next step we use a rule $p' \in \sigma(p)$ such that $y = y_1Ay_2$, $(p'\colon A \to v, \sigma(p)) \in P$ and there is no rule $(p''\colon B \to u, \sigma(p)) \in P$ such that $p'' \in \sigma(p)$, $y = y_1{}'By_2{}'$, $|y_1{}'| < |y_1|$, and the string obtained in this way is $z = y_1vy_2$ (we say that y is rewritten in the leftmost manner by the rules of the set $\sigma(p)$), or (in case of programmed grammars with appearance checking) if no rule $p' \in \sigma(p)$ can be used for rewriting y, then we choose an arbitrary rule $p' \in \sigma(p)$ and we pass to rewrite y in the leftmost manner by the rules of $\varphi(p')$.

c) Continue this way until a terminal string is obtained.

By $L_{left\text{-}2}(G)$ we denote the language generated by G in this way, and we denote the families of languages generated in this way by $\mathscr{L}(PG,\ left\text{-}2)$, $\mathscr{L}(PG,\ \varepsilon,\ left\text{-}2)$, $\mathscr{L}(PG,\ ac,\ left\text{-}2)$ and $\mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}2)$.

Example 4.2.2. Consider the programmed grammar H from Example 4.0.1. The starting nonterminal S is always rewritten to $XACC$ by using the rule r_1, so X is the leftmost nonterminal in this sentential form. As in the previous example, X cannot be rewritten, so we consider the next nonterminal, which is A. We can either rewrite A by using r_2 or r_4, but in every case, we get the same number of a, b and c, because we cannot (unlike in Example 4.0.1) use rule r_3 to generate more c than a or b. This is because A is "everytime" the leftmost nonterminal that can be rewritten, so we cannot skip it and rewrite C (unless we have used r_2 or r_3). Also note that d will be generated from the last C in a sentential form, because when we use rule r_5 we have to rewrite the leftmost C. Therefore, the language generated under leftmost derivations of type 2 by H is $L(H) = \{da^nb^nc^nd\colon n \geq 1\}$.

The string *daabbccd* is produced via the following derivation: $S \Rightarrow XACC$ $[r_1] \Rightarrow XaAbCC$ $[r_2] \Rightarrow XaAbcCC$ $[r_3] \Rightarrow XaabbcCC$ $[r_4] \Rightarrow XaabbccC$ $[r_5] \Rightarrow XaabbccY$ $[r_6] \Rightarrow daabbccY$ $[r_7] \Rightarrow daabbccd$ $[r_8]$.

Theorem 4.2.3. *(See [6])*

i) $\mathscr{L}(PG,\ left\text{-}2) = \mathscr{L}(PG,\ ac,\ left\text{-}2) = \boldsymbol{CS}$

ii) $\mathscr{L}(PG,\ \varepsilon,\ left\text{-}2) = \mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}2) = \boldsymbol{RE}$.

4.3 Leftmost Derivations of Type 3

Main idea. To use each rule in a leftmost manner, that is the leftmost appearence of its left-hand member is rewritten.

The difference between this type of leftmost derivations and leftmost derivations of type 2 is that here we do not have to first consider the current sentential form and then choose a rule, more to the contrary, we first choose a rule and then we apply it to the leftmost nonterminal equal to its left-hand side. So, for example, if we have $ABBB$ as the current sentential form, last used rule was r and we have rules $(s_1\colon A \to x, \sigma(s_1))$ and $(s_2\colon B \to y, \sigma(s_1))$ so that in $s_1, s_2 \in \sigma(r)$, we can choose any of them. But, if we choose s_2, we have to rewrite the leftmost B in the sentential form (in this case it is that B right next to A).

Definition 4.3.1. Let G be a programmed grammar ((a programmed grammar with appearance checking, respectively). A derivation according to G is called *leftmost of type* 3 (see [5]) if each rule used in it replaces the leftmost occurence of its left-hand member (if no occurence of this symbol exists, then the rule is not applicable; it can be "applied" in the appearance checking mode).

By $L_{left\text{-}3}(G)$ we denote the language generated by G in this way, and we denote the families of languages generated in this way by $\mathscr{L}(PG,\ left\text{-}3)$, $\mathscr{L}(PG,\ \varepsilon,\ left\text{-}3)$, $\mathscr{L}(PG,\ ac,\ left\text{-}3)$ and $\mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}3)$.

Example 4.3.2. Consider the programmed grammar H from Example 4.0.1. Unlike in Example 4.0.1, here we can generate more c than a or b, because even if we have A as the leftmost nonterminal (after X), we can rewrite C by using rule r_3, but (this is important) everytime only the leftmost C. Thus, the language generated under leftmost derivations of type 3 by H is $L(H) = \{da^nb^nc^md\colon n \geq 1, m \geq n\}$.

The string *dabcccd* is produced via the following derivation: $S \Rightarrow XACC\ [r_1] \Rightarrow XAcCC\ [r_3] \Rightarrow XAccCC\ [r_3] \Rightarrow XabccCC\ [r_4] \Rightarrow XabcccC\ [r_5] \Rightarrow XabcccY\ [r_6] \Rightarrow dabcccY\ [r_7] \Rightarrow dabcccd\ [r_8]$.

Theorem 4.3.3. *(See [6])*

i) $\mathscr{L}(PG) \subset \mathscr{L}(PG,\ left\text{-}3) \subset \mathscr{L}(PG,\ ac,\ left\text{-}3) \subset \boldsymbol{CS}$

ii) $\mathscr{L}(PG,\ \varepsilon) \subset \mathscr{L}(PG,\ \varepsilon,\ left\text{-}3) \subset \mathscr{L}(PG,\ \varepsilon,\ ac,\ left\text{-}3) = \boldsymbol{RE}$

Chapter 5

Left Restriction Range and Its Consequences

As was stated in Chapter 4, when we use only leftmost derivations (by leftmost derivations I mean leftmost derivations of type 1), we decrease the generative power of programmed grammars. In order to study the left restriction range, first a new type of a grammar (called *state grammar*) along with a modification of this grammar is presented in Section 5.1 and then n-limited derivations are defined in Section 5.2. Main results of this work are presented in Section 5.3. Finally, in Section 5.4, results significance for syntactical analysis is given.

5.1 State Grammars

State grammars were introduced by T. Kasai in 1970 [7]. Informally, a state grammar is a combination of a context-free grammar and of a finite automaton, because it generates strings by using context-free rules, but also have states, like finite automata. So, at each derivation step, a nonterminal is rewritten and the current state is changed. Another interesting thing is that Kasai talks about *configurations* (like in automata), rather than about sentential forms.

5.1.1 State Grammar

Definition 5.1.1. A *state grammar* (see [7]) is a sextuple

$$G = (N, T, W, P, S, p_0),$$

where

N, T and S are as in a context-free grammar ($V = N \cup T$ is the total alphabet);

W is a nonempty finite set of *states*;

$P \subseteq W \times N \times W \times V^+$ is a finite relation;

$p_0 \in W$ is the starting state.

An element $(p, A, q, v) \in P$ is called a *state production* (abbreviated production or rule) and is usually written as $(p, A) \to (q, v)$, optionally labeled by a label, like $l\colon (p, A) \to (q, v)$. A nonterminal A is said to be *applicable* under a state p if $(p, A) \to (q, v)$ is in P for some $q \in Q$ and $v \in V^+$.

Given a state grammar $G = (N, T, W, P, S, p_0)$, let $\Rightarrow$ be a relation on $W \times V^+$ defined as follows: Let $p \in W$ and $w = xAy \in V^+$. If this A is the leftmost occurence of an applicable nonterminal in w under p and $(p, A) \to (q, v) \in P$, then we write $(p, xAy) \Rightarrow (q, xvy)$.

For $x, y \in W \times V^+$, write $x \Rightarrow^* y$ if either $x = y$ or there exists $x_0, \ldots, x_k$ such that $x_0 = x$, $x_k = y$, and $x_i \Rightarrow x_{i+1}$ for each i, $0 \leq i \leq k-1$. The sequence $x_0, \ldots, x_k$ is called a derivation (of length k) and is denoted by $x_0 \Rightarrow \cdots \Rightarrow x_k$. Thus, $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$ and we define $\Rightarrow^+$ to be the transitive closure of $\Rightarrow$.

The language of G is

$$L(G) = \{w\colon w \in T^+, (p_0, S) \Rightarrow^* (q, w) \text{ for some } q \in W\}.$$

For all $w \in L(G)$, each pair (q, v) in a derivation $(p_0, S) \Rightarrow^* (q, v) \Rightarrow^* (r, w)$, where $q, r \in W$ and $v \in V^+$ is called a *configuration* of G.

Example 5.1.2. Let $G = (\{S, X, Y\}, \{a, b\}, \{p_0, p_1, p_2, p_3, p_4, p_5\}, P, S, p_0)$ be a state grammar with the following rules in P:

$1\colon (p_0, S) \to (p_0, XY)$,

$2\colon (p_0, X) \to (p_1, aX)$,

$3\colon (p_1, Y) \to (p_0, aY)$,

$4\colon (p_0, X) \to (p_2, bX)$,

$5\colon (p_2, Y) \to (p_0, bY)$,

$6\colon (p_0, X) \to (p_3, a)$,

$7\colon (p_3, Y) \to (p_0, a)$,

$8\colon (p_0, X) \to (p_4, b)$,

$9\colon (p_4, Y) \to (p_0, b)$.

This grammar generates the noncontext-free language $\{ww\colon w \in T^+\}$. This is achieved by first rewriting S to XY and then, using states, ensure that everytime X is rewritten to aX or bX, change state to force rewrite of Y to aY or bY, respectively. Derivation is finished by rewriting X to a or b and then Y to a or b, respectively.

The string $abab$ is produced via the following derivation: $(p_0, S) \Rightarrow (p_0, XY)$ [1] $\Rightarrow (p_1, aXY)$ [2] $\Rightarrow (p_0, aXaY)$ [3] $\Rightarrow (p_4, abaY)$ [8] $\Rightarrow (p_0, abab)$ [9].

5.1.2 Unrestricted State Grammar

Definition 5.1.3. An *unrestricted state grammar* $G = (N, T, W, P, S, p_0)$ is a state grammar with $\Rightarrow$ defined as follows. Let $p \in W$ and $w = xAy \in V^+$. If $(p, A) \to (q, v) \in P$, then $(p, xAy) \Rightarrow (q, xvy)$. Informally, during a single derivation step, G can rewrite any occurence of an applicable nonterminal; that is, the case that $x = x_1 A x_2$ for some $x_1, x_2 \in V^*$ is not ruled out, as opposed to the Definition 5.1.1.

5.2 n-limited Derivations

Informally, the meaning of n-limited derivations (also called n-leftmost derivations) is that at each step of a derivation, at most the nth occurence of a nonterminal (from the left) has to be rewritten, where $n \geq 1$. This type of derivations was originally defined for state grammars in [7], but the following definition applies also for programmed grammars and, in the following sections, interesting properties resulting from the introduction of n-limited derivations into programmed grammars will be shown.

Definition 5.2.1. Let $G = (N_G, T_G, P_G, S_G)$ be a programmed grammar, $H = (N_H,\ T_H,\ W_H,\ P_H,\ S_H,\ p_{H0})$ be a state grammar and $U = (N_U,\ T_U,\ W_U,\ P_U,\ S_U,\ p_{U0})$ be an unrestricted state grammar, respectively, and let n be a positive integer. An *n-limited derivation* (see [7]) is a derivation of the form $\alpha_0 \overset{j(1)}{\Rightarrow} \alpha_1 \dots \overset{j(r)}{\Rightarrow} \alpha_r$ such that $j(i) \leq n$ for each i, where $j(i)$ means that the j-th nonterminal is rewritten. In this case we write $\alpha_0\ {}_n\!\Rightarrow^* \alpha_r$ instead of $\alpha_0 \Rightarrow^* \alpha_r$ in order to express that it is made as an n-limited derivation. The languages generated by G, H and U are denoted by $L_{n\text{-}lim}(G)$, $L_{n\text{-}lim}(H)$ and $L_{n\text{-}lim}(U)$, respectively, and defined as $L_{n\text{-}lim}(G) = \{w \in T_G^+ \colon S_G\ {}_n\!\Rightarrow^* w\}$, $L_{n\text{-}lim}(H) = \{w \in T_H^+ \colon (p_{H0}, S_H)\ {}_n\!\Rightarrow^* (q, w)$ for some $q \in W_H\}$ and $L_{n\text{-}lim}(U) = \{w \in T_U^+ \colon (p_{U0}, S_U)\ {}_n\!\Rightarrow^* (q, w)$ for some $q \in W_U\}$, respectively.

By $\mathscr{L}_{n\text{-}lim}(PG)$, $\mathscr{L}_{n\text{-}lim}(SG)$ and $\mathscr{L}_{n\text{-}lim}({}_uSG)$ we denote the families of languages generated by n-limited programmed grammars, n-limited state grammars and n-limited unrestricted state grammars, respectively.

5.3 Results

5.3.1 Equivalence of $\mathscr{L}_{n\text{-}lim}(PG)$ and $\mathscr{L}_{n\text{-}lim}({}_uSG)$

This subsection establishes an equivalence between the family of languages generated by n-limited programmed grammars and the family of languages generated by n-limited unrestricted state grammars.

Lemma 5.3.1. *For all $n \geq 1$, $\mathscr{L}_{n\text{-}lim}({}_uSG) \subseteq \mathscr{L}_{n\text{-}lim}(PG)$.*

Proof.

Construction. Let $H = (N, T, W, P, S, p_0)$ be an n-limited unrestricted state grammar, $V = N \cup T$. We construct an n-limited programmed grammar $G = (N \cup \{S_G\}, T, P_G, S_G)$ such that $L(H) = L(G)$, where $S_G \notin N$ is a new nonterminal and P_G is constructed by performing the following steps:

1. Denote all rules in P by labels $\langle i \rangle$, $i \in \{1, 2, \dots, |P|\}$.
2. Add $(r_0 \colon S_G \to S, \sigma(r_0))$ with $\sigma(r_0) = \{r_i \colon \langle i \rangle \colon (p_0, S) \to (q, v) \in P,\ q \in W,\ v \in V^+\})$ to P_G.
3. For each rule $\langle i \rangle \colon (p, A) \to (q, v) \in P$, extend P_G by $(r_i \colon A \to v, \sigma(r_i))$ with $\sigma(r_i) = \{r_j \colon \langle j \rangle \colon (q, B) \to (m, u) \in P\}$, where $p, q, m \in W$, $A, B \in N$ and $u, v \in V^+$ for some positive integers $j \in \{1, 2, \dots, |P|\}$.

Basic idea. G simulates H's n-limited derivations and instead of states in H it uses success fields to regulate the derivation process. There is an injection from P to P_G because every rule in P corresponds to a single rule in P_G, but there is no rule which corresponds to the first rule in P_G (this rule is the only rule applicable to S_G and thus it starts the derivation process).

More precisely, let $(p, x_0A_1x_1 \dots A_hx_h)\ {}_n\!\Rightarrow\ (q, x_0A_1x_1 \dots A_{j-1}x_{j-1}vx_jA_{j+1}x_{j+1} \dots A_hx_h)$ be an n-limited derivation in H using some rule $\langle s\rangle\colon (p, A_j) \to (q, v)$, where $p, q \in W$, x_0, $x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N$, $v \in V^+$ and $1 \leq j \leq n$. Then, the corresponding n-limited derivation in G would be $x_0A_1x_1 \dots A_hx_h\ {}_n\!\Rightarrow x_0A_1x_1 \dots A_{j-1}x_{j-1}vx_jA_{j+1}x_{j+1} \dots$ A_hx_h using a rule $(r_s\colon A_j \to v, \sigma(r_s))$, $\sigma(r_s) = \{r_k \in lab(P_G)\colon \langle k\rangle\colon (q, C) \to (t, c) \in P$ for some $C \in N$, $c \in V^+$ and $t \in W\}$.

Claim 5.3.2. *If $(p_0, S)\ {}_n\!\Rightarrow^m (q, x_0A_1x_1 \dots A_hx_h)$ in H, then $S_G\ {}_n\!\Rightarrow^+ x_0A_1x_1 \dots A_hx_h$ in G for some $m \geq 0$, $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N$ and $q \in W$.*

This claim is established by induction on m.

Basis: Let $m = 0$. For $(p_0, S)\ {}_n\!\Rightarrow^0 (p_0, S)$ in H there exists $S_G\ {}_n\!\Rightarrow S\ [r_0]$ in G.

Induction hypothesis: Assume that Claim 5.3.2 holds for all n-limited derivations of length m or less for some $m \geq 0$.

Induction step: Consider $(p_0, S)\ {}_n\!\Rightarrow^m (o, y)$, where $o \in W$ and $y = x_0A_1x_1 \dots A_hx_h$ for some $x_0, x_1, \dots, x_h \in T^*$ and $A_1, A_2, \dots, A_h \in N$, so that $(o, y)\ {}_n\!\Rightarrow (q, x)\ [\langle s\rangle]$. For $\langle s\rangle\colon (o, A_j) \to (q, v)$, $v \in V^+$, is x of the form $x = x_0A_1x_1 \dots A_{j-1}x_{j-1}vx_jA_{j+1}x_{j+1} \dots A_h$ x_h, $1 \leq j \leq n$. Based on the induction hypothesis, there exists $S_G\ {}_n\!\Rightarrow^+ y\ \ {}_n\!\Rightarrow x\ [r_s]$, where $(r_s\colon A_j \to v, \sigma(r_s))$, $\sigma(r_s) = \{r_k \in lab(P_G)\colon \langle k\rangle\colon (q, C) \to (t, c) \in P$ for some $C \in N$, $c \in V^+$ and $t \in W\}$.

By Claim 5.3.2 for $h = 0$, if $(p_0, S)\ {}_n\!\Rightarrow^z (q, x)$ in H, then $S_G\ {}_n\!\Rightarrow^+ x$ in G for some $z \geq 0$, $q \in W$ and $x \in T^*$, so $L(H) \subseteq L(G)$.

Claim 5.3.3. *If $S_G\ {}_n\!\Rightarrow^m x_0A_1x_1 \dots A_hx_h$ in G, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x_0A_1x_1 \dots A_hx_h)$ in H for some $m \geq 1$, $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N$ and $q \in W$.*

This claim is established by induction on m.

Basis: Let $m = 1$. For $S_G\ {}_n\!\Rightarrow S\ [r_0]$ in G, there exists $(p_0, S)\ {}_n\!\Rightarrow^0 (p_0, S)$ in H.

Induction hypothesis: Assume that Claim 5.3.3 holds for all n-limited derivations of length m or less for some $m \geq 1$.

Induction step: Consider $S_G\ {}_n\!\Rightarrow^m y$, where $y = x_0A_1x_1 \dots A_hx_h$ for some $x_0, x_1, \dots, x_h \in T^*$ and $A_1, A_2, \dots, A_h \in N$, so that $y\ {}_n\!\Rightarrow x\ [r_k]$ for some $r_k \in lab(P_G)$, $0 \leq k \leq |P_G|$. For $(r_k\colon A_j \to v, \sigma(r))$, $v \in V^+$, is x of the form $x = x_0A_1x_1 \dots A_{j-1}x_{j-1}vx_jA_{j+1}x_{j+1} \dots A_h$ x_h, $1 \leq j \leq n$. Based on the induction hypothesis, there exists $(p_0, S)\ {}_n\!\Rightarrow^* (q, y)$ and there is a rule $\langle k\rangle\colon (q, A_j) \to (s, v)$ for some $q, s \in W$, so that $(q, y)\ {}_n\!\Rightarrow (s, x)\ [\langle k\rangle]$.

By Claim 5.3.3 for $h = 0$, if $S_G\ {}_n\!\Rightarrow^z x$ in G, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x)$ in H for some $z \geq 1$, $q \in W$ and $x \in T^*$, so $L(G) \subseteq L(H)$.

As $L(H) \subseteq L(G)$ and $L(G) \subseteq L(H)$, $L(H) = L(G)$. Thus, Lemma 5.3.1 holds. $\square$

Lemma 5.3.4. *For all $n \geq 1$, $\mathscr{L}_{n\text{-}lim}(PG) \subseteq \mathscr{L}_{n\text{-}lim}({}_uSG)$.*

Proof.

Construction. Let $G = (N, T, P, S)$ be an n-limited programmed grammar. We construct an n-limited unrestricted state grammar $H = (N \cup \{S_H\}, T, W, P_H, S_H, \langle\rho\rangle)$ such that $L(G) = L(H)$, where $S_H \notin N$ is a new nonterminal, ρ is a new symbol and P_H and W are constructed by performing the following steps:

1. For each $(r\colon S \to v, \sigma(r)) \in P$, $v \in V^+$, add $(\langle\rho\rangle, S_H) \to (\langle r\rangle, S)$ to P_H, where $\langle\rho\rangle$ is a new state in W.
2. For each $(r\colon A \to v, \sigma(r)) \in P$, $A \in N$, $v \in V^+$, if $\sigma(r) \neq \emptyset$, then for each $s \in \sigma(r)$ add $(\langle r\rangle, A) \to (\langle s\rangle, v)$ to P_H, where $\langle r\rangle$ and $\langle s\rangle$ are new states in W. Otherwise, add $(\langle r\rangle, A) \to (\langle\phi\rangle, v)$ to P_H, where $\langle r\rangle$ and $\langle\phi\rangle$ are new states in W.

Basic idea. H simulates G's n-limited derivations and instead of success fields in G's rules it uses states to regulate the derivation process. There is an injection from $lab(P)$ to W, because each rule label in $lab(P)$ corresponds to a single state in H and no rule label corresponds to states $\langle\rho\rangle$ and $\langle\phi\rangle$.

More precisely, let $x_0A_1x_1 \ldots A_hx_h$ be a sentential form derived by G, where $x_0, x_1, \ldots, x_h \in T^*$, $A_1, A_2, \ldots, A_h \in N$ and let $(r\colon A_j \to v, \sigma(r))$, $v \in V^+$, be a rule in P that is applicable in the next step to A_j for $1 \leq j \leq n$. Then, H's new configuration is of the form $(\langle r\rangle, x_0A_1x_1 \ldots A_hx_h)$, which encodes the information about the next applicable rule label in G in the state.

Claim 5.3.5. *If $S\ {}_n\!\Rightarrow^m\ x_0A_1x_1 \ldots A_hx_h\ [p_1p_2 \ldots p_m]$ (or $[\varepsilon]$ if $m = 0$) in G, then $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^+ (\langle s\rangle, x_0A_1x_1 \ldots A_hx_h)$ in H, for $m \geq 0$, $x_0, x_1, \ldots, x_h \in T^*$, $A_1, A_2, \ldots, A_h \in N$ and $p_1, p_2, \ldots, p_m \in lab(P)$. If $m \geq 1$ and $\sigma(p_m) \neq \emptyset$, then exists a rule $(p_{m+1}\colon A \to v, \sigma(p_{m+1}))$ in P, $A \in N$, $v \in V^+$, $p_{m+1} \in \sigma(p_m)$ and a rule $(\langle s\rangle, A) \to (\langle t\rangle, v)$ in P_H, where $s = p_{m+1}$ and $\langle s\rangle, \langle t\rangle \in W$.*

This claim is established by induction on m.

Basis: Let $m = 0$. For $S\ {}_n\!\Rightarrow^0\ S$ in G there exists $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow (\langle r\rangle, S)$ in H, where $(r\colon S \to v, \sigma(r)) \in P$, $v \in V^+$ and $(\langle\rho\rangle, S_H) \to (\langle r\rangle, S) \in P_H$.

Induction hypothesis: Assume that Claim 5.3.5 holds for all n-limited derivations of length m or less for some $m \geq 0$.

Induction step: Consider $S\ {}_n\!\Rightarrow^m\ y\ [p_1p_2 \ldots p_m]$, where $y = x_0A_1x_1 \ldots A_hx_h$, $x_0, x_1, \ldots, x_h \in T^*$, $A_1, A_2, \ldots, A_h \in N$, and $p_1, p_2, \ldots, p_m, p_{m+1} \in lab(P)$, so that $y\ {}_n\!\Rightarrow\ x\ [p_{m+1}]$. If $m = 0$, then $p_{m+1} \in \{p \in lab(P)\colon (p\colon S \to u, \sigma(p)) \in P$ for some $u \in V^+\}$, otherwise $p_{m+1} \in \sigma(p_m)$. For $(p_{m+1}\colon A_j \to v,\ \sigma(p_{m+1}))$, where $v \in V^+$, is x in the form $x = x_0A_1x_1 \ldots A_{j-1}x_{j-1}vx_jA_{j+1} \ldots A_hx_h$. Based on the induction hypothesis, there exists $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^+ (\langle p_{m+1}\rangle, y)\ {}_n\!\Rightarrow (\langle t\rangle, x)$, where $(\langle p_{m+1}\rangle, A_j) \to (\langle t\rangle, v) \in P_H$ for some $\langle t\rangle \in W$. If $\sigma(p_{m+1}) \neq \emptyset$, then exists a rule $p_{m+2} \in \sigma(p_{m+1})$, $(p_{m+2} : B \to u, \sigma(p_{m+2})) \in P$, $t = p_{m+1}$, and there is a rule $(\langle t\rangle, B) \to (\langle r\rangle, u)$ in P_H for some $B \in N$, $\langle r\rangle \in W$ and $u \in V^+$. Otherwise, $t = \phi$ and there is no rule $(\langle\phi\rangle, B) \to (\langle r\rangle, u)$ in P_H for any $B \in N$ and $u \in V^+$.

By Claim 5.3.5 for $h = 0$, if $S\ {}_n\!\Rightarrow^z\ y$ in G, then $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^+ (\langle s\rangle, y)$ in H for some $z \geq 0$, $\langle s\rangle \in W$ and $y \in T^*$, so $L(G) \subseteq L(H)$.

Claim 5.3.6. *If* $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^m (\langle s\rangle, x_0A_1x_1\dots A_hx_h)$ *in* H*, then* $S\ {}_n\!\Rightarrow^* x_0A_1x_1\dots A_hx_h$ *in* G *for some* $m \geq 1$, $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N$ *and* $\langle s\rangle \in W$.

This claim is established by induction on m.

Basis: Let $m = 1$. For $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow (\langle r\rangle, S)\ [s]$ in H there exists $S\ {}_n\!\Rightarrow^0 S$ in G, where $(r : S \to v, \sigma(r)) \in P$ and $s\colon (\langle\rho\rangle, S_H) \to (\langle r\rangle, S) \in P_H$ for some $\langle r\rangle \in W$ and $v \in V^+$.

Induction hypothesis: Assume that Claim 5.3.6 holds for all n-limited derivations of length m or less for some $m \geq 1$.

Induction step: Consider $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^m (\langle r\rangle, y)$, where $\langle r\rangle \in W$, $y = x_0A_1x_1\dots A_hx_h$ for some $x_0, x_1, \dots, x_h \in T^*$ and $A_1, A_2, \dots, A_h \in N$, so that $(\langle r\rangle, y) \Rightarrow (\langle p\rangle, x)$ by some $(\langle r\rangle, A_j) \Rightarrow (\langle p\rangle, v) \in P_H$, where $x = x_0A_1x_1\dots A_{j-1}x_{j-1}vx_jA_{j+1}x_{j+1}\dots A_h\ x_h$, $1 \leq j \leq n$ and $v \in V^+$. Based on the induction hypothesis, there exists $S\ {}_n\!\Rightarrow^* x$ in G and some rule $(r : A_j \to v, \sigma(r)) \in P$ such that $x \Rightarrow y\ [r]$.

By Claim 5.3.6 for $h = 0$, if $(\langle\rho\rangle, S_H)\ {}_n\!\Rightarrow^z (\langle s\rangle, y)$ in H, then $S\ {}_n\!\Rightarrow^* y$ in G for some $z \geq 1$, $\langle s\rangle \in W$ and $y \in T^*$, so $L(H) \subseteq L(G)$.

As $L(G) \subseteq L(H)$ and $L(H) \subseteq L(G)$, $L(G) = L(H)$. Thus, Lemma 5.3.4 holds. □

Example 5.3.7. Let $G = (\{S, X, C\}, \{a, b, c\}, P, S)$ be an n-limited programmed grammar, where $n = 2$, with the following rules in P:

$(r_1\colon S \to XC, \{r_2, r_4\})$,

$(r_2\colon X \to aXb, \{r_3\})$,

$(r_3\colon C \to cC, \{r_2, r_4\})$,

$(r_4\colon X \to ab, \{r_5\})$,

$(r_5\colon C \to c, \emptyset)$.

The equivalent n-limited unrestricted state grammar H, where $n = 2$, would be $H = (\{S, X, C, S_H\}, \{a, b, c\}, \{\langle\rho\rangle, \langle r_1\rangle, \langle r_2\rangle, \langle r_3\rangle, \langle r_4\rangle, \langle r_5\rangle, \langle\phi\rangle\}, P_H, S_H, \langle\rho\rangle)$, with the following rules in P_H:

1: $(\langle\rho\rangle, S_H) \to (\langle r_1\rangle, S)$,

2: $(\langle r_1\rangle, S) \to (\langle r_2\rangle, XC)$,

3: $(\langle r_1\rangle, S) \to (\langle r_4\rangle, XC)$,

4: $(\langle r_2\rangle, X) \to (\langle r_3\rangle, aXb)$,

5: $(\langle r_3\rangle, C) \to (\langle r_2\rangle, cC)$,

6: $(\langle r_3\rangle, C) \to (\langle r_4\rangle, cC)$,

7: $(\langle r_4\rangle, X) \to (\langle r_5\rangle, ab)$,

8: $(\langle r_5\rangle, C) \to (\langle\phi\rangle, c)$.

Generated language is $L(G) = L(H) = \{a^nb^nc^n\colon n \geq 1\}$.

Theorem 5.3.8. *For all* $n \geq 1$, $\mathscr{L}_{n\text{-}lim}(PG) = \mathscr{L}_{n\text{-}lim}({}_uSG)$.

Proof. The result directly follows from Lemmas 5.3.1 and 5.3.4. □

5.3.2 Infinite Hieararchy of Language Families

This subsection establishes an infinite hierarchy of language families resulting from n-limited programmed grammars.

Lemma 5.3.9. *For all $n \geq 1$, let $G = (N, T, W, P, S, p_0)$ be an n-limited state grammar. If there is no more than one occurence of each nonterminal from N in every configuration in each n-limited derivation step for each $x \in L(G)$, then there is an n-limited unrestricted state grammar $H = (N, T, W, P, S, p_0)$ such that $L(G) = L(H)$.*

Proof. According to our definitions, the only difference between an n-limited state grammar and an n-limited unrestricted state grammar is the definition of $\Rightarrow$. So, if there is no more than one occurence of each nonterminal from N in every configuration in each n-limited derivation step for each $x \in L(G)$, then every rewritten nonterminal in a single n-limited derivation step in H is also the leftmost occurence of such nonterminal in the corresponding configuration, because this nonterminal occures at most once in each configuration. $\square$

Definition 5.3.10. For all $n \geq 1$, let L_n be the subset of $\{a_1, \dots, a_{4n-2}\}^*$ defined by $L_n = \{a_1^k a_2^k \cdots a_{4n-2}^k : k \geq 1\}$ (see [7]).

Lemma 5.3.11. *For all $n \geq 1$, there is an n-limited unrestricted state grammar that generates L_n.*

Proof. Recall the n-limited state grammar $G_n = (V_n - \Sigma_n, \Sigma_n, K_n, P_n, \sigma, p_0)$ from the proof of Theorem 4 in [7]. As this grammar generates L_n [7] and it satisfies the condition from Lemma 5.3.9, there is an n-limited unrestricted state grammar $H_n = (V_n - \Sigma_n, \Sigma_n, K_n, P_n, \sigma, p_0)$ such that $L(G_n) = L(H_n) = L_n$. $\square$

Lemma 5.3.12. *For all $n \geq 1$, $\mathscr{L}_{n\text{-}lim}({}_u SG) \subseteq \mathscr{L}_{n\text{-}lim}(SG)$.*

Proof.

Construction. Let $G = (N_G, T, W_G, P_G, S, p_0)$ be an n-limited unrestricted state grammar, $V_G = N_G \cup T$. We construct an n-limited state grammar $H = (N_H, T, W_H, P_H, S, p_0)$ such that $L(G) = L(H)$, where initially $N_H = N_G$, $W_H = W_G$ and $P_H = P_G$. If $n \geq 2$, to complete the construction, perform the following steps:

1. Denote all rules in P_G by labels r_i, $i \in \{1, 2, \dots, |P_G|\}$.
2. For each rule $r_i\colon (p, A) \to (q, v) \in P_G$, $p, q \in W_G$, $A \in N_G$ and $v \in V_G^+$, do:

   For all $j \in \{1, 2, \dots, n-1\}$, do:

   (a) If $j = 1$, for all $B \in N_G$ add $(p, B) \to (\overrightarrow{p}^{i,j}, B^j)$ to P_H, otherwise for all $B \in N_G$ add $(\overrightarrow{p}^{i,j-1}, B) \to (\overrightarrow{p}^{i,j}, B^j)$ to P_H, where $\overrightarrow{p}^{i,j}$ is a new state in W_H and B^j is a new nonterminal in N_H.

   (b) Add $(\overrightarrow{p}^{i,j}, A) \to (\overleftarrow{q}^{i,j}, v)$ to P_H, where $\overleftarrow{q}^{i,j}$ is a new state in W_H.

   (c) If $j = 1$, for all $B \in N_G$ add $(\overleftarrow{q}^{i,j}, B^j) \to (q, B)$ to P_H, otherwise for all $B \in N_G$ add $(\overleftarrow{q}^{i,j}, B^j) \to (\overleftarrow{q}^{i,j-1}, B)$ to P_H.

Basic idea. For a rule $r_i\colon (p, A) \to (q, v) \in P_G$, G can rewrite any occurence of A to v within first n nonterminals in an n-limited derivation step. As H always rewrites the leftmost occurence of A, when simulating G, it must be able to rewrite any occurence of A to v within first n nonterminals, too. Clearly, by definition, there is no problem if $n = 1$, because G will always rewrite the leftmost occurence of A.

So, assuming that $n \geq 2$ and that G rewrites A to v by $r_i\colon (p, A) \to (q, v)$, H rewrites A to v in a sequence of n-limited derivation steps, beginning by rewriting all nonterminals before the A that was rewritten in G to their respective indexed versions by using rules introduced in (a), then rewriting A to v by a rule introduced in (b) and finishing the simulation of that n-limited derivation step in G by rewriting indexed nonterminals back to their respective nonindexed versions by rules introduced in (c).

A state $\overrightarrow{p}^{i,k}$ ($\overleftarrow{q}^{i,k}$) encodes that H is simulating a rule $r_i\colon (p, A) \to (q, v) \in P_G$ in G, k nonterminals have been rewritten to their respective indexed versions in the current configuration and A has not yet been (has been) rewritten to v, respectively.

Claim 5.3.13. *If $(p_0, S)\ {}_n\!\Rightarrow^m (q, x_0A_1x_1 \dots A_hx_h)$ in G, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x_0A_1x_1 \dots A_hx_h)$ in H for some $m \geq 0$, $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N_G$ and $q \in W_G$.*

This claim is established by induction on m.

Basis: Let $m = 0$. For $(p_0, S)\ {}_n\!\Rightarrow^0 (p_0, S)$ in G there exists $(p_0, S)\ {}_n\!\Rightarrow^0 (p_0, S)$ in H.

Induction hypothesis: Assume that Claim 5.3.13 holds for all $0 \leq m \leq k$, where $k \geq 0$.

Induction step: Let $(p_0, S)\ {}_n\!\Rightarrow^{k+1} (q, x)$ in G, where $q \in W_G$ and $x = x_0A_1x_1 \dots A_{j-1}x_{j-1}v$ $x_jA_{j+1} \dots A_hx_h$ for some $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N_G$, $v \in V_G$ and $j \leq n \leq h$. Because $k + 1 \geq 1$, express $(p_0, S)\ {}_n\!\Rightarrow^{k+1} (q, x)$ as $(p_0, S)\ {}_n\!\Rightarrow^k (o, y)\ {}_n\!\Rightarrow (q, x)$ $[r_i]$, where $o \in W_G$, $r_i\colon (o, A_j) \to (q, v) \in P_G$ and $y = x_0A_1x_1 \dots A_hx_h$. By the induction hypothesis, $(p_0, S)\ {}_n\!\Rightarrow^* (o, y)$ in H. As $r_i\colon (o, A_j) \to (q, v) \in P_G$, if $n = 1$, then $(p_0, S)\ {}_n\!\Rightarrow^* (o, y)\ {}_n\!\Rightarrow$ (q, x) $[r_i]$ in H and the induction step is completed. Otherwise, H sequentially rewrites all $A_1, A_2, \dots, A_{j-1}$ to $A_1^1, A_2^2, \dots, A_{j-1}^{j-1}$ by $(\overrightarrow{o}^{i,1}, A_1) \to (\overrightarrow{o}^{i,1}, A_1^1)$, $(\overrightarrow{o}^{i,1}, A_2) \to (\overrightarrow{o}^{i,2}, A_2^2)$, $\dots$, $(\overrightarrow{o}^{i,j-2}, A_{j-1}) \to (\overrightarrow{o}^{i,j-1}, A_{j-1}^{j-1}) \in P_H$ (added in (a)). Then, H rewrites A_j to v by using $(\overrightarrow{o}^{i,j-1}, A_j) \to (\overleftarrow{q}^{i,j-1}, v) \in P_H$ (added in (b)). To finish the simulation of $(o, y)\ {}_n\!\Rightarrow (q, x)$ $[r_i]$ in G, H sequentially rewrites all $A_{j-1}^{j-1}, A_{j-2}^{j-2}, \dots, A_1^1$ back to $A_{j-1}, A_{j-2}, \dots, A_1$ by applying a sequence of rules $(\overleftarrow{q}^{i,j-1}, A_{j-1}^{j-1}) \to (\overleftarrow{q}^{i,j-2}, A_{j-1})$, $\dots$, $(\overleftarrow{q}^{i,1}, A_1^1) \to (q, A_1) \in P_H$ (added in (c)), which completes the induction step.

By Claim 5.3.13 for $h = 0$, if $(p_0, S)\ {}_n\!\Rightarrow^z (q, x)$ in G, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x)$ in H for some $z \geq 0$, $q \in W_G$ and $x \in T^*$, so $L(G) \subseteq L(H)$.

Claim 5.3.14. *Let $(p_0, S)\ {}_n\!\Rightarrow^m (q, x_0A_1x_1 \dots A_hx_h)$ in H for some $m \geq 0$, $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N_H$ and $q \in W_H$. If $q \in W_G$ and $A_1, A_2, \dots, A_h \in N_G$, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x_0A_1x_1 \dots A_hx_h)$ in G, otherwise $(q, x_0A_1x_1 \dots A_hx_h)\ {}_n\!\Rightarrow^+ (t, x_0A_1^{'}x_1 \dots A_h^{'}$ $x_h)$ in H and $(p_0, S)\ {}_n\!\Rightarrow^* (t, x_0A_1x_1 \dots A_hx_h)$ in G, where $A_1^{'}, A_2^{'}, \dots, A_h^{'} \in N_G$ and $t \in W_G$.*

This claim is established by induction on m.

Basis: Let $m = 0$ and $(p_0, S)\ {}_n\!\Rightarrow^0 (p_0, S)$ in H. As $p_0 \in W_G$, $S \in N_G$ and $(p_0, S)\ {}_n\!\Rightarrow^0$ (p_0, S) in G, the basis holds.

Induction hypothesis: Assume that Claim 5.3.14 holds for all $0 \leq m \leq k$, where $k \geq 0$.

Induction step: Let $(p_0, S)\ {}_n\!\Rightarrow^{k+1} (q, x)$ in H, where $q \in W_H$ and $x = x_0A_1x_1 \dots A_{j-1}x_{j-1}$ $vA_{j+1} \dots A_hx_h$ for some $x_0, x_1, \dots, x_h \in T^*$, $A_1, A_2, \dots, A_h \in N_H$, $v \in (N_H \cup T)^+$ and $j \leq n \leq h$. Because $k + 1 \geq 1$, express $(p_0, S)\ {}_n\!\Rightarrow^{k+1} (q, x)$ as $(p_0, S)\ {}_n\!\Rightarrow^{k} (o, y)\ {}_n\!\Rightarrow (q, x)$ $[s]$, where $o \in W_H$, $s\colon (o, A_j) \to (q, v) \in P_H$ and $y = x_0A_1x_1 \dots A_hx_h$. By the induction hypothesis, $o \in W_G$ and $A_1, A_2, \dots, A_h \in N_G$. If $q \in W_G$ and $v \in V_G^+$, then $(p_0, S)\ {}_n\!\Rightarrow^* (o, y)\ {}_n\!\Rightarrow (q, x)$ in G and the induction step is completed. Otherwise, by using rule s in the last step in H, H starts a simulation of some rule $(o, A_j) \to (t, v)$, where $t \in W_G$ and $v \in V_G^+$, so $q = \overrightarrow{o}^{i,1}$ and $v = A_1^1$ for some $1 \leq i \leq |P_G|$, so that $(r_i\colon A_j \to v, \sigma(r_i)) \in P_G$. To complete the simulation of $(o, A_j) \to (t, v)$, H sequentially rewrites all $A_2, \dots, A_{j-1}$ to $A_2^2, \dots, A_{j-1}^{j-1}$ by $(\overrightarrow{o}^{i,1}, A_2) \to (\overrightarrow{o}^{i,2}, A_2^2), \dots, (\overrightarrow{o}^{i,j-2}, A_{j-1}) \to (\overrightarrow{o}^{i,j-1}, A_{j-1}^{j-1}) \in P_H$ (added in (a)). Then, H rewrites A_j to v by using $(\overrightarrow{o}^{i,j-1}, A_j) \to (\overleftarrow{t}^{i,j-1}, v) \in P_H$ (added in (b)). Finally, H sequentially rewrites all $A_{j-1}^{j-1}, A_{j-2}^{j-2}, \dots, A_1^1$ back to $A_{j-1}, A_{j-2}, \dots, A_1$ by applying a sequence of rules $(\overleftarrow{t}^{i,j-1}, A_{j-1}^{j-1}) \to (\overleftarrow{t}^{i,j-2}, A_{j-1})$, $\dots$, $(\overleftarrow{t}^{i,1}, A_1^1) \to (t, A_1) \in P_H$ (added in (c)). Then, $(o, A_j)\ {}_n\!\Rightarrow (t, v)$ in G, which completes the induction step.

By Claim 5.3.14 for $h = 0$, let $(p_0, S)\ {}_n\!\Rightarrow^z (q, x)$ in H for some $z \geq 0$, $q \in W_H$ and $x \in T^*$. If $q \in W_G$, then $(p_0, S)\ {}_n\!\Rightarrow^* (q, x)$ in G, otherwise $(q, x)\ {}_n\!\Rightarrow^+ (r, x)$ in H and $(p_0, S)\ {}_n\!\Rightarrow^* (r, x)$ in G for some $r \in W_G$, so $L(H) \subseteq L(G)$.

As $L(G) \subseteq L(H)$ and $L(H) \subseteq L(G)$, $L(G) = L(H)$. Thus, Lemma 5.3.12 holds. □

Lemma 5.3.15. *For all $n \geq 2$, L_n is not in $\mathscr{L}_{(n-1)\text{-}lim}({}_uSG)$.*

Proof. Recall that for all $n \geq 2$, L_n is not in $\mathscr{L}_{(n-1)\text{-}lim}(SG)$ (see Theorem 5 in [7]) and for all $m \geq 1$, $\mathscr{L}_{m\text{-}lim}({}_uSG) \subseteq \mathscr{L}_{m\text{-}lim}(SG)$ by Lemma 5.3.12. Thus, this lemma holds. □

The following result is the most important result of this work, because it says that n-limited programmed grammars are less powerful than $(n+1)$-limited programmed grammars and thus the left restriction range affects the generative power of n-limited programmed grammars.

Theorem 5.3.16. *For all $n \geq 1$, $\mathscr{L}_{n\text{-}lim}(PG) \subset \mathscr{L}_{(n+1)\text{-}lim}(PG)$.*

Proof. This result follows from Lemma 5.3.15 and Theorem 5.3.8. □

5.4 Significance for Syntactical Analysis

As far as I was able to determine, there has been no systematic research in the direction of parsing methods based on programmed grammars, or based on regulated grammars in general. Current compilers use parsers based on context-free grammars, because there is a well-researched underlying theory (see [13]), which includes top-down parsing based on $LL(k)$ grammars and pushdown automata (recursive decent, predictive or table-based parsers) and botom-up parsing based on $LR(k)$ grammars and extended pushdown automata (SLR or LALR parsers).

However, these models can be found insufficient when one need to parse a language which cannot be defined by a context-free grammar. In such situations, programmed grammars can be considered as a suitable model for description and parsing of such languages. In [8] and [9], use of regulated grammars (especially programmed grammars) in programming languages was studied and discussed. In [5], a sketch of a possible parser based on programmed grammars was described. However, there is no complete and usable parsing theory based on programmed grammars by knowledge, except that A. Rußmann in [18] introduced a *dynamic $LL(k)$ parser* based on so-called *dynamic context-free grammars* and managed to characterize $LR(1)$ grammars by a grammar model involving leftmost derivations which can be easily seen as the deterministic version of regulated grammars with leftmost derivations [4].

The main result presented in this work is significant for syntactical analysis when studying or writing a parser for some noncontext-free language that is based on programmed grammars. One surely would like to have the parser deterministic – that involves use only of leftmost derivations. However, Theorem 4.1.3 states that this is not possible, because we loose the generative power of programmed grammars. So, by using the result stated by Theorem 5.3.16, one can use n-leftmost derivations (where n is high enough to define the language by an n-limited programmed grammar) instead of leftmost derivations, which can be more efficient than using universal (unrestricted) derivations, so at least preserving "some" determinism.

Chapter 6

Conclusion

Canonical derivations in programmed grammars (with focus on leftmost derivations) and left restriction range were studied in this work. It was shown that if we introduce n-limited derivations in programmed grammars as they were defined for state grammars, we get an infinite hierarchy of language families resulting from n-limited programmed grammars (Theorem 5.3.8), so the left restriction range affects the generative power of n-limited programmed grammars. In order to proof this hierarchy, a modification of a state grammar, called unrestricted state grammar, was introduced and it was shown that the family of languages generated by n-limited programmed grammars is equivalent to the family of languages generated by n-limited unrestricted state grammars (Theorem 5.3.8).

The main contribution of this work is in the significance for syntactical analysis, as discussed in Section 5.4 of Chapter 5. The main result gives the possibility to define a noncontext-free language by an n-limited programmed grammar that can be parsed using n-leftmost derivations where $n \geq 2$ is high enough to define the language by an n-limited programmed grammar, while it would be impossible to parse it using strict leftmost derivations ($n = 1$).

6.1 Open Problems

There are still some open questions related to this work. Future project research could be focused on solving these open problems.

- As stated in Section 3.4 in Chapter 3, it is not yet known if the inclusion between $\mathscr{L}(PG) \subseteq \mathscr{L}(PG, \varepsilon)$ is proper or these two families are equivalent. If they are not equivalent, is there also an inifite hierarchy of language families resulting from n-limited programmed grammars with ε-rules?
- Lemma 5.3.12 states that for all $n \geq 1$, $\mathscr{L}_{n\text{-}lim}({}_uSG) \subseteq \mathscr{L}_{n\text{-}lim}(SG)$. Does the opposite inclusion ($\mathscr{L}_{n\text{-}lim}(SG) \subseteq \mathscr{L}_{n\text{-}lim}({}_uSG)$) hold? If so, then the family of languages generated by n-limited unrestricted state grammars is equivalent to the family of languages generated by n-limited state grammars.

Bibliography

[1] S. Abraham. Some questions of language theory. In *Proceedings of the 1965 conference on Computational linguistics*, pages 1–11, Morristown, NJ, USA, 1965. Association for Computational Linguistics.

[2] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[3] J. Dassow. Grammars with regulated rewriting [online]. Lecture in the 5th PhD Program Formal Languages and Applications. Otto-von-Guericke-Universität, Magdeburg, 2003. Available on URL: http://theo.cs.uni-magdeburg.de/dassow/fphdpro.pdf (April 2008).

[4] J. Dassow, H. Fernau, and G. Pãun. On the leftmost derivation in matrix grammars. In *International Journal of Foundations of Computer Science*, pages 61–80, 1997.

[5] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory.* Springer, New York, 1989. ISBN 38751-414-7.

[6] H. Fernau. Regulated grammars under leftmost derivation. *Grammars*, 3(1):37–62, 1999.

[7] T. Kasai. An hierarchy between context-free and context-sensitive languages. *Journal of Computer and System Sciences*, 4:492–508, 1970.

[8] M. Kot. Regulated grammars. Master's thesis, VSB – Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science, 2002.

[9] Z. Křivka. Use of regulated grammars in programming languages [online]. Project for the Programming Language Theory. Brno, University of Technology, Faculty of Information Technology, 2002. Available on URL: http://www.fit.vutbr.cz/study/courses/TJD/public/0506TJD-Krivka.pdf (April 2008).

[10] P. Linz. *An Introduction to Formal Languages and Automata, 3rd ed.* Jones and Bartlett Publishers, 2000. ISBN 0-7637-1422-4.

[11] A. Meduna. Modern theoretical computer science [online]. Lectures in the Modern Theoretical Computer Science. Brno, University of Technology, Faculty of Information Technology, 2007. Available on URL: http://www.fit.vutbr.cz/ meduna/work/doku.php?id=lectures:phd:tid:tid (April 2008).

[12] A. Meduna. *Automata and Languages: Theory and Applications.* Springer, London, 2000. ISBN 1-85233-074-0.

[13] A. Meduna. *Elements of Compiler Design.* Taylor and Francis Informa plc, New York, 2008. ISBN 978-1-4200-6323-3.

[14] A. Meduna and M. Švec. *Grammars with Context Conditions and Their Applications.* Wiley, Hoboken, New Jersey, 2005. ISBN 0-471-71831-9.

[15] D. J. Rosenkrantz. *Programmed grammar - A New Device for Generating Formal Languages.* PhD thesis, Columbia University, New York, 1967.

[16] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.

[17] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages: Word, Language, Grammar, Volume 1.* Springer, Berlin, 1997. ISBN 3-540-60420-0.

[18] A. Rußmann. Dynamic ll(k) parsing. *Acta Informatica*, 34(4):267–289, 1997.

[19] A. P. J. van der Walt. Random context grammars. In *Proceedings of Symposium on Formal Languages*, 1970.

[20] S. J. Whittaker. Finding grammars that are more than context-free, but not fully context-sensitive. Technical Report 2006-514, School of Computing, Queen's University, Canada, 2006.