論文 / 著書情報
Article / Book Information

題目(和文)	先読みを持つ文法とその微分に関する研究
Title(English)	Studies on Grammars with Lookahead and Their Derivatives
著者(和文)	宮嵜貴之
Author(English)	Takayuki Miyazaki
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12512号, 授与年月日:2023年9月22日, 学位の種別:課程博士, 審査員:南出 靖彦,田中 圭介,増原 英彦,鹿島 亮,脇田 建
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12512号, Conferred date:2023/9/22, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctral Thesis

Studies on Grammars with Lookahead and Their Derivatives

Takayuki Miyazaki

Supervisor: Yasuhiko Minamide

Tokyo Institute of Technology
Department of Mathematical and Computer Science

July 2023

Abstract

Regular expressions and context-free grammars (CFGs) are central concepts in formal language theory and various important results are known. They are also important in practice and are applied in regular expression libraries and parsers. However, there are some differences between these concepts in practice and in theory. The aim of this study is to develop a theory of practical regular expressions and CFGs. Typical extension of practical regular expressions and CFGs are positive and negative lookahead. Lookahead is a constraint for the following string. Lookahead is important because it can represent intersection and complement, which are difficult to represent in regular expressions and CFGs.

We first develop a theory of regular expressions with lookahead (RELAs). We give the semantics of RELA by languages with lookahead, which is a set of pairs of strings. We then develop derivatives of RELA and a conversion from RELA to DFA using derivatives.

We then introduce context-free grammars with lookahead (CFGLAs). To accommodate negative lookahead, we give the limit semantics with the limit of iterations from the empty set. We show some closure properties and that the grammars are an extension of both CFGs and PEGs. However, the limit semantics has some problems: it is not defined for all grammars and not preserved by substitution and replacement. In addition, it has a problem for a computational complexity. To overcome the problems, we introduce the least interval semantics of CFGLA with the least fixed point of pairs of lower and upper bounds. Finally, we develop a recognition algorithm for CFGLA by combining derivatives of CFG and RELA. The recognition algorithm runs in $O(n^3|G|)$ time and $O(n^2|G|)$ space for an input string of the length n and a given CFGLA G under the least interval semantics.

Contents

Chapter 1

Introduction

Regular expressions and context-free grammars (CFGs) are central concepts in formal language theory and various important results are known. They are also important in practice and are applied in regular expression libraries and parsers. However, there are some differences between these concepts in practice and in theory. The aim of this study is to develop a theory of practical regular expressions and CFGs.

1.1 Background

1.1.1 Regular Expressions with Lookahead

A regular expression library is a tool for searching and matching text using regular expressions. Unlike theoretical regular expressions, many regular expression libraries use extended regular expressions. Typical extensions are positive lookahead, negative lookahead, capturing, and backreference. Regular expression libraries use backtracking matching algorithms to accommodate these extensions.

We describe regular expressions with lookahead (RELAs). In many implementations, a positive lookahead and a negative lookahead are written as $(?{=}e)$ and $(?!e)$ for a regular expression e, respectively. However, in this paper, we adopt the simpler notation from parsing expression grammars (PEGs) [16] and denote them as $\&e$ and $!e$. Lookahead is a constraint on the following string. A positive lookahead $\&e$ and a negative lookahead $!e$ indicate that the following string starts and does not start with e, respectively. For example, the expression $a\,\&b$ is a concatenation of a and $\&b$, and represents a letter a with a constraint that the following string starts with b. For a practical example, by searching for $\mathit{Java}!(\mathit{Script})$ in regular expression libraries, we can find a string *Java* that is not a prefix of *JavaScript*. Lookahead do not consume letters and affect only the success or failure of the matching.

Lookahead is important because it can represent intersection and complement, which are difficult to represent in regular expressions. The expression $\&(e_1\$)\,\&(e_2\$)\,\&(e_3\$)\,.^*$ represents the intersection of three languages of regular expressions e_1, e_2, and e_3, where the expression "." represents an arbitrary letter, and \$ represents the end of a string. Furthermore, the expression $!(e\$)\,.^*$ represents the complement of the language of e. For a practical example, the expression representing passwords can be written as the following.

$$\&(.^*[a\text{-}z])\,\&(.^*[0\text{-}9])\,[a\text{-}z0\text{-}9]\{8, 64\}$$

This represents the set of strings of 8 to 64 letters including both lowercase letters $[a\text{-}z]$ and digits [0-9]. Furthermore, using the features of lookahead, the expression representing comments in the C language,

which cannot be nested, can be written as the following.

$$/*\,(!(*/).)^*\,*/$$

Note that $*$ is a letter, and * represents iteration. This expression represents that a comment starts with $/*$, followed by characters such that each of them is not at the start of $*/$, and finally ends with $*/$.

The theoretical study of RELA was initiated by Morihata [27]. He showed a conversion similar to Thompson's construction [39] from RELA to Boolean automata [10] that recognize the same language. Therefore, a RELA can be converted to a deterministic finite automaton (DFA) that accepts the same language because a Boolean automaton can be converted to a DFA. This shows that the language of a RELA is a regular language. He also showed the number of states of the obtained Boolean automaton is linear and the number of states of DFA is double exponential in the size of the RELA. This indicates that adding lookahead is a better choice than adding intersection and complement because the number of states of DFA obtained from regular expressions with complement is not bounded by any elementary function [38].

1.1.2 Grammars with Lookahead

Some parsers use grammars with extensions that are not present in CFGs. Typical extensions are positive lookahead, negative lookahead, and ordered choice. For example, parsers that support these extensions include Scala's parser combinators and PEG parser. Such parsers use a backtracking matching algorithm to accommodate these extensions.

An ordered choice is written as e_1/e_2 for two expressions including variables e_1 and e_2. The backtracking matching algorithm runs for e_1/e_2, first trying e_1 and only trying e_2 if it fails in e_1. An ordered choice suppresses branching to e_2 and removes ambiguity. It is related to negative lookahead. An ordered choice e_1/e_2 can be represented as $e_1|(!e_1)e_2$ using negative lookahead and usual alternation. Scala's parser combinators supports positive and negative lookahead in addition to usual CFG syntax and does not support left recursion.

PEG is a grammar introduced by Ford [16]. It supports positive lookahead, negative lookahead, and ordered choices, and it does not support alternation. A recursive descending parsing algorithm is given for PEG, which defines the language of PEG. PEG is unambiguous, has expressiveness beyond deterministic context-free languages and has a linear time recognition algorithm [16]. However, the language of PEG sometimes differs from intuition. For example, the grammar

$$S \leftarrow aSa/\varepsilon$$

defines the language $\{a^{2^n-2} \mid n \geq 1\}$.

We briefly review related grammars. Chida and Kuramitsu introduced parsing expression grammars with unordered choices [11], which are grammars that support alternation in addition to PEG syntax. The semantics is given by a recursive descending parsing algorithm, hence it does not support left-recursion. Barash and Okhotin introduced context-free grammars with right contexts (CFGRCs) [5], which are grammars that support right context $\rhd$, extended right context $\trianglerighteq$ and intersection $\cap$. CFGRC can support positive lookahead by $\cap \trianglerighteq e\Sigma^*$. Okhotin introduced Boolean grammars [32], which support

intersection and complement in addition to CFG syntax. This is technically relevant to our study in the sence that the grammars supports negative operations.

1.1.3 Derivatives

Derivatives of regular expressions are classical concepts given by Brzozowski [9]. They are the computation of the left quotients on regular expressions, where the left quotient of a language L by a string x is $x^{-1}L = \{y \mid xy \in L\}$. It is the opration to remove x from L. Then, $x \in L$ holds if and only if $\varepsilon \in x^{-1}L$. Therefore, if we can compute derivatives and decide the membership of the empty string, then the membership of a string is decidable.

Brzozowski gave a conversion from a regular expression to a DFA by derivatives [9]. The set of states obtained by repeatedly applying derivatives is finite under some equivalence relation; thus, a regular expression can be converted to DFA with derivatives. Antimirov [3] showed that the number of states of the DFA is bounded by $2^{\|e\|} + 1$ under some equivalence relation, where $\|e\|$ represents the number of occurrence of letters. The conversion by derivatives is relatively easy to implement in programming languages with pattern matching, and it is known that the number of states of DFA obtained by derivatives is small [33]. About half a century later, Might et al. presented a recognition algorithm by derivatives for CFGs [26]. It is proven that a recognition algorithm of CFG by derivatives runs in $O(n^3)$ time [1] and $O(n^2)$ space [20], where n is the length of an input string. Derivatives of PEGs [17,29,30] are also introduced. Recognition by derivatives can handle CFGs and regular expressions with the same algorithm.

1.2 Contributions

We develop a theory of regular expressions and context-free grammars with lookahead by using derivatives as a main tool.

- We introduce a language with lookahead, which is a set of pairs of strings, and we show the set of languages with lookahead forms a Kleene algebra with tests. We reformulate the semantics of RELA by a language with lookahead. We then develop derivatives of RELA and a conversion from RELA to DFA using derivatives.
- We introduce context-free grammars with lookahead (CFGLAs). To accommodate negative lookahead, we give the limit semantics with the limit of iterations from the empty set. We show that the grammars are an extension of both CFGs and PEGs. We show that the language class is closed under union, intersection, complement, marked concatenation, and marked Kleene star.
- We point out the problems of the limit semantics: the semantics is not defined for all grammars and not preserved by substitution and replacement. In addition, it has a problem for a computational complexity. To overcome the problems, we introduce the least interval semantics of CFGLA with the least fixed point of pairs of lower and upper bounds.
- We develop a recognition algorithm for CFGLA by combining derivatives of CFG and RELA. The recognition algorithm runs in $O(n^3|G|)$ time and $O(n^2|G|)$ space for an input string of the length

n and a given CFGLA G under the least interval semantics.

We consider that practical aspects of the contribution include the following. Formal definitions of RELA and CFGLA allow us to distinguish between specifications and bugs. The recognition algorithm by derivatives is easy to implement and useful for prototype implementations. The introduction of CFGLA allows us to use CFG and PEG mixed together.

1.3 Overview

In Chapter 2, we study on RELA. The main contributions are a semantics of RELA closer to the standard definition of regular expressions, the derivatives of RELA, and a conversion to DFA using derivatives. In particular, we show that derivatives of RELAs are a neat extension of derivatives of regular expressions. These results correspond to Publication 1.

In Chapter 3, we initiate a study on CFGLA. We describe limit semantics, closure properties, and its relationship to other grammars. In particular, we show that it is an extension of both CFG and PEG. These results correspond to Publication 2.

In Chapter 4, we first describe the problem of the limit semantics for CFGLA. Next, we define the least interval semantics. We show that this semantics preserves most of the good properties of the limit semantics and solves its problems. We show that a binary normal form is obtained. These results correspond to half of Publication 3.

In Chapter 5, we introduce derivatives of CFGLA and a recognition algorithm by derivatives. We show the recognition algorithm runs in cubic time and quadratic space for the length of an input string under least interval semantics. These results correspond to half of Publication 3.

Chapter 2

Regular Expressions with Lookahead

We introduce l-languages and reformulate the semantics of RELA by a l-language. We define the left quotients of l-languages and define the derivatives of RELA. We show a conversion from RELA to DFA by derivatives.

2.1 Preliminaries

2.1.1 Languages, Regular Expressions

A string is a finite sequence of letters. The set of strings over an alphabet Σ is written as Σ^*. For strings x and y, the concatenation of x and y, written xy, is the string obtained by appending y to the end of x. For a string x, the length of x, written $|x|$, is the number of letters that it contains. The string of length zero is called the empty string and is written ε.

A language $L \subseteq \Sigma^*$ is a set of strings. For languages L_1 and L_2, the concatenation of L_1 and L_2, written $L_1 \cdot L_2$, is the language $\{xy \mid x \in L_1,\ y \in L_2\}$. For language L, the n-th power of L is defined by $L^0 = \{\varepsilon\}$ and $L^{n+1} = L \cdot L^n$. Then, the Kleene star of L is defined by $L^* = \bigcup_{i=0}^{\infty} L^i$.

A regular expression e is defined by the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid e|e \mid ee \mid e^* \quad (a \in \Sigma)$$

The Kleene star has the highest priority, then concatenation and then alternation. For a regular expression e, the language of e is defined by the following.

$$L(\emptyset) = \emptyset,\ L(\varepsilon) = \{\varepsilon\},\ L(a) = \{a\},$$
$$L(e_1|e_2) = L(e_1) \cup L(e_2),\ L(e_1e_2) = L(e_1) \cdot L(e_2),\ L(e^*) = L(e)^*.$$

For a language L, if there exists a regular expression e such that $L = L(e)$, then L is called a regular language.

2.1.2 Algebraic Structures

We review a Kleene algebra and a Boolean algebra.

A monoid $(M, \cdot, 1)$ is a tuple that consists of a set M, a binary operation $\cdot$, and an element $1 \in M$ and satisfies the following.

- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ for any $x, y, z \in M$.

- $x \cdot 1 = x = 1 \cdot x$ for any $x \in M$.

A commutative idempotent monoid $(M, \cdot, 1)$ is a monoid that satisfies the following.

- $x \cdot y = y \cdot x$ for any $x, y \in M$.
- $x \cdot x = x$ for any $x \in M$.

An idempotent semiring $(S, \vee, \cdot, 0, 1)$ is a tuple that consists of a set S, two binary operations $\vee, \cdot$, and two elements $0, 1 \in S$ and satisfies the following.

- $(S, \vee, 0)$ is a commutative idempotent monoid.
- $(S, \cdot, 1)$ is a monoid.
- $x \cdot (y \vee z) = x \cdot y \vee x \cdot z$ and $(x \vee y) \cdot z = x \cdot z \vee y \cdot z$ for any $x, y, z \in S$.
- $x \cdot 0 = 0 = 0 \cdot x$ for any $x \in S$.

A Kleene algebra $(K, \vee, \cdot, ^*, 0, 1)$ is a tuple that consists of a set K, two binary operations $\vee, \cdot$, an unary operation *, and two elements $0, 1 \in K$ and satisfies the following.

- $(K, \vee, \cdot, 0, 1)$ is an idempotent semiring.
- $x^* = x \cdot x^* \vee 1$ and $x^* = x^* \cdot x \vee 1$ for any $x \in K$.
- $x \cdot z \vee y \leq z \implies x^* \cdot y \leq z$ and $z \cdot x \vee y \leq z \implies y \cdot x^* \leq z$ for any $x, y, z \in K$, where $x \leq y \iff x \vee y = y$.

For example, the set of languages $(\mathcal{P}(\Sigma^*), \cup, \cdot, ^*, \emptyset, \{\varepsilon\})$ forms a Kleene algebra.

A bounded distributive lattice $(L, \vee, \wedge, 0, 1)$ is a tuple that consists of a set L, two binary operations $\vee, \wedge$, and two elements $0, 1 \in L$ and satisfies the following.

- $(L, \vee, 0)$ is a commutative idempotent monoid.
- $(L, \wedge, 1)$ is a commutative idempotent monoid.
- $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ for any $x, y, z \in L$.
- $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ for any $x, y, z \in L$.
- $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$ for any $x, y \in L$.

A Boolean algebra $(B, \vee, \wedge, \neg, 0, 1)$ is a tuple that consists of a set B, two binary operations $\vee, \wedge$, an unary operation $\neg$, and two elements $0, 1 \in B$ and satisfies the following.

- $(B, \vee, \wedge, 0, 1)$ is a bounded distributive lattice.
- $x \vee \neg x = 1$ and $x \wedge \neg x = 0$ for any $x \in B$.

For example, the set of languages $(\mathcal{P}(\Sigma^*), \cup, \cap, ^c, \emptyset, \Sigma^*)$ forms a Boolean algebra.

A Kleene algebra with tests [24] $(K, B, \vee, \cdot, ^*, \neg, 0, 1)$ is a tuple that consists of a set K, a subset $B \subseteq K$, two binary operations $\vee, \cdot$, two unary operations $^*, \neg$, and two elements $0, 1 \in K$ and satisfies the following.

- $(K, \vee, \cdot, ^*, 0, 1)$ is a Kleene algebra.
- $(B, \vee, \cdot, \neg, 0, 1)$ is a Boolean algebra.

2.2 Languages with Lookahead

In this section, we introduce languages with lookahead (l-languages) and define their operations: the concatenation, Kleene star, positive lookahead, and negative lookahead. We show that the set of l-languages with these operations forms Kleene algebra with tests. We also describe the mutual conversion between languages and l-languages.

To accommodate lookahead, it is necessary to use a set of pairs of strings. This concept was first introduced in the CFGRC paper [5] as the concept of language with context. Later, it was independently introduced as the concept of language with lookahead. These two concepts are equivalent as sets, but not as algebraic structures. The former has context operations and the latter has lookahead.

A *language with lookahead* (*l-language*) $R \subseteq \Sigma^* \times \Sigma^*$ is a set of pairs of strings. We call the first element of a pair "matched part" and the second element "remaining part". The remaining part represents the constraint on what can follow. For example, an l-language $\{(a, bx) \mid x \in \Sigma^*\}$ can be regarded a letter a that has the constraint that the following string starts with b.

The *concatenation* operation $\cdot$ is defined on l-languages as the following.

$$R \cdot S = \{(xy, z) \mid (x, yz) \in R, (y, z) \in S\}$$

By considering only the matched part, if x is matched in R, and y is matched in S, then xy is matched in $R \cdot S$. Hence, this is an usual concatenation operation. By considering the remaining part, if yz is allowed as the remaining string in R, y is matched in S, and z is allowed as the remaining string in S, then z is allowed as the remaining string in $R \cdot S$.

The set of l-languages form a monoid with the concatenation. Indeed, the following holds.

$$\begin{aligned}
R_1 \cdot (R_2 \cdot R_3) &= \{(xy, z) \mid (x, yz) \in R_1, (y, z) \in (R_2 \cdot R_3)\} \\
&= \{(xy_1y_2, z) \mid (x, y_1y_2z) \in R_1, (y_1, y_2z) \in R_2, (y_2, z) \in R_3\} \\
(R_1 \cdot R_2) \cdot R_3 &= \{(xy, z) \mid (x, yz) \in (R_1 \cdot R_2), (y, z) \in R_3\} \\
&= \{(x_1x_2y, z) \mid (x_1, x_2yz) \in R_1, (x_2, yz) \in R_2, (y, z) \in R_3\}
\end{aligned}$$

Thus, associativity $R_1 \cdot (R_2 \cdot R_3) = (R_1 \cdot R_2) \cdot R_3$ holds. The unit element is $\{\varepsilon\} \times \Sigma^*$ and $R \cdot (\{\varepsilon\} \times \Sigma^*) = (\{\varepsilon\} \times \Sigma^*) \cdot R = R$ holds. In addition, $\emptyset$ is absorbing element, i.e., $R \cdot \emptyset = \emptyset \cdot R = \emptyset$ holds. The distributivity $(R_1 \cup R_2) \cdot S = R_1 \cdot S \cup R_2 \cdot S$ and $S \cdot (R_1 \cup R_2) = S \cdot R_1 \cup S \cdot R_2$ hold. More generally, $(\bigcup_{i \in I} R_i) \cdot S = \bigcup_{i \in I} (R_i \cdot S)$ and $S \cdot (\bigcup_{i \in I} R_i) = \bigcup_{i \in I} (S \cdot R_i)$ hold for any index set I. Therefore, the set of l-languages $(\mathcal{P}(\Sigma^* \times \Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\} \times \Sigma^*)$ forms an idempotent semiring.

A subset of $\{\varepsilon\} \times \Sigma^*$ can be expressed as $\{\varepsilon\} \times L$ for some language L. It is just a constraint and only matches ε. For subsets of $\{\varepsilon\} \times \Sigma^*$, the following holds.

$$\begin{aligned}
(\{\varepsilon\} \times L_1) \cdot (\{\varepsilon\} \times L_2) &= \{(xy, z) \mid (x, yz) \in (\{\varepsilon\} \times L_1), (y, z) \in (\{\varepsilon\} \times L_2)\} \\
&= \{(\varepsilon, z) \mid (\varepsilon, z) \in (\{\varepsilon\} \times L_1), (\varepsilon, z) \in (\{\varepsilon\} \times L_2)\} \\
&= (\{\varepsilon\} \times L_1) \cap (\{\varepsilon\} \times L_2)
\end{aligned}$$

That is, for $R, S \subseteq (\{\varepsilon\} \times \Sigma^*)$, $R \cdot S = R \cap S$ holds. The concatenation operation behaves as intersection operation for constraints. Therefore, the set of constraints $(\mathcal{P}(\{\varepsilon\} \times \Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\} \times \Sigma^*)$ forms a bounded distributive lattice.

The *Kleene star* operation is defined in the same manner as for a usual language.

$$R^* = \bigcup_{i=0}^{\infty} R^i \qquad (R^0 = \{\varepsilon\} \times \Sigma^*,\ R^{n+1} = R^n \cdot R)$$

For a l-language R, we have $R^* = R \cdot R^* \cup (\{\varepsilon\} \times \Sigma^*)$ and $R^* = R^* \cdot R \cup (\{\varepsilon\} \times \Sigma^*)$ by infinite distributivity. The set of l-languages $(\mathcal{P}(\Sigma^* \times \Sigma^*), \cup, \cdot, ^*, \emptyset, \{\varepsilon\} \times \Sigma^*)$ forms a Kleene algebra. For the Kleene star operation, the following lemma holds.

Lemma 2.2.1. *Let R be an l-language and I be $\{\varepsilon\} \times \Sigma^*$.*

$$R^* = (R \setminus I) \cdot R^* \cup I.$$

Proof. For any l-language S, we show $(S \cup I)^n = \bigcup_{i=0}^{n} S^i$ by induction on n. The case of 0, both sides are I. The case of $n+1$, $(S \cup I)^{n+1} = (S \cup I)^n \cdot (S \cup I) = (\bigcup_{i=0}^{n} S^i) \cdot (S \cup I) = (\bigcup_{i=1}^{n+1} S^i) \cup (\bigcup_{i=0}^{n} S^i) = \bigcup_{i=0}^{n+1} S^i$. Therefore, $(S \cup I)^* = \bigcup_{i=0}^{\infty} (S \cup I)^i = \bigcup_{i=0}^{\infty} \bigcup_{j=0}^{i} S^j = S^*$ holds. We obtain $(R \setminus I)^* = ((R \setminus I) \cup I)^* = (R \cup I)^* = R^*$. Finally, we obtain $R^* = (R \setminus I)^* = (R \setminus I) \cdot (R \setminus I)^* \cup I = (R \setminus I) \cdot R^* \cup I$. □

We describe the lookahead operations. The *positive lookahead* operation & is defined as follows.

$$\&R = \{\varepsilon\} \times \{xy \mid (x, y) \in R\}$$

The language $\{xy \mid (x, y) \in R\}$ represents successful inputs for R. Hence, positive lookahead $\&R$ is a constraint that the following string is a successful input for R.

For a positive lookahead operation, $\&(R \cup S) = \&R \cup \&S$ holds. If $R \subseteq (\{\varepsilon\} \times \Sigma^*)$, then $\&(R \cdot S) = R \cdot \&S$ holds. These properties are similar to the linear mapping of vector spaces. In fact, it can be regarded as a homomorphism of the left semimodule [19], which is a generalization of vector space. Furthermore, $\&(\&R) = \&R$ holds.

The *negative lookahead* operation ! is defined as the following.

$$!R = \{\varepsilon\} \times (\Sigma^* \setminus \{xy \mid (x, y) \in R\})$$

The negative lookahead can be written as $!R = (\{\varepsilon\} \times \Sigma^*) \setminus \&R$ by using positive lookahead and the set difference. Hence, for $R \subseteq (\{\varepsilon\} \times \Sigma^*)$, $!R = (\{\varepsilon\} \times \Sigma^*) \setminus R$ holds. Therefore, the set of constraints $(\mathcal{P}(\{\varepsilon\} \times \Sigma^*), \emptyset, \{\varepsilon\} \times \Sigma^*, \cup, \cdot, !)$ forms a Boolean algebra. The positive lookahead is written as $\&R = !(!R)$ by using negative lookahead. Therefore, positive lookahead can be treated as an abbreviation.

Finally, the set of l-languages $(\mathcal{P}(\Sigma^* \times \Sigma^*), \mathcal{P}(\{\varepsilon\} \times \Sigma^*), \cup, \cdot, ^*, !, \emptyset, \{\varepsilon\} \times \Sigma^*)$ forms a Kleene algebra with tests. In addition, the following two properties hold.

- For l-languages R and S, $!(R \cup S) = !R \cdot !S$ holds.
- For $R \subseteq (\{\varepsilon\} \times \Sigma^*)$ and an l-language S, $!(R \cdot S) = !R \cup !S$ holds.

These properties are used to prove the finiteness of the number of expressions obtained by derivatives.

A language can be converted to l-language by $L \mapsto L \times \Sigma^*$. Here, Σ^* means that all strings can be followed, i.e., there is no constraint. This is obviously injection, i.e., this is a bijection to their image $\mathcal{P}(\Sigma^*) \times \{\Sigma^*\}$. Moreover, this is an isomorphism of Kleene algebra to their image. Conversely, there is

a standard way to convert an l-language to a language, which is $R \mapsto \{x \mid (x, \varepsilon) \in R\}$. The fact that the remaining part contains the empty string means that the end of the string can be followed. Hence, the obtained language is a collection of strings that can be followed by the end of a string. The restriction of this mapping to $\mathcal{P}(\Sigma^*) \times \{\Sigma^*\}$ is the inverse mapping of the first convertion. These convertions and their properties are used to prove the compatibility with regular expressions.

2.3 Regular Expressions with Lookahead

The *regular expressions with lookahead* $RELA(\Sigma)$ is given by the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid e|e \mid ee \mid e^* \mid !e \quad (a \in \Sigma)$$

We define several abbreviations. The expression "." represents any letter and is an abbreviation for $a_1|\ldots|a_n$ when $\Sigma = \{a_1, \ldots, a_n\}$. The expression \$ represents the end of input and is an abbreviation for "!.". A positive lookahead $\&e$ is an abbreviation for $!(!e)$. An expression e^n is an expression in which e is concatenated n times. We write $\|e\|$ for the number of occurrence of letters (elements of Σ) in RELA e.

The *semantics* (or behavior) B is a mapping from $RELA(\Sigma)$ to l-languages given by the following.

$$\begin{aligned} B(\emptyset) &= \emptyset \\ B(\varepsilon) &= \{\varepsilon\} \times \Sigma^* \\ B(a) &= \{a\} \times \Sigma^* \\ B(e_1|e_2) &= B(e_1) \cup B(e_2) \\ B(e_1e_2) &= B(e_1) \cdot B(e_2) \\ B(e^*) &= B(e)^* \\ B(!e) &= !B(e) \end{aligned}$$

A regular expression r can be regarded as a RELA. Then, $B(r) = L(r) \times \Sigma^*$ holds. For abbreviations, the following hold.

$$\begin{aligned} B(.) &= \Sigma \times \Sigma^* \\ B(\$) &= \{\varepsilon\} \times \{\varepsilon\} \\ B(\&e) &= \&B(e) \\ B(e^n) &= B(e)^n \end{aligned}$$

The *language of RELA* $L(e)$ is defined by the following.

$$L(e) = \{x \mid (x, \varepsilon) \in B(e)\}$$

This definition extends that of the language of regular expressions. That is, for a regular expression, the language of regular expression and the language of RELA coinside. For abbreviations, the following hold.

$$\begin{aligned} B(e\$) &= L(e) \times \{\varepsilon\} \\ B(\&(e\$)) &= \{\varepsilon\} \times L(e) \end{aligned}$$

RELA given by the following grammar are called *logical expressions of RELA*.

$$l ::= \emptyset \mid \varepsilon \mid l|l \mid ll \mid !e \quad (e \in RELA(\Sigma))$$

The reason these RELA are called logical expressions is that concatenation operation behaves similarly to intersection. For any logical expressions l, $B(l) \subseteq \{\varepsilon\} \times \Sigma^*$ holds.

2.4 Derivatives

Let R be an l-language and x be a string. The *left quotient of matched part* $x^{-1}R$ and the *left quotient of remaining part* $\widetilde{x}^{-1}R$ are defined as follows.

$$x^{-1}R = \{(y, z) \mid (xy, z) \in R\}$$
$$\widetilde{x}^{-1}R = \{(\varepsilon, y) \mid (\varepsilon, xy) \in R\} \quad (\widetilde{x} \neq \varepsilon)$$

Here, $x^{-1}R$ is the left quotient for the matching part, which is the operation to remove x from the matching part. This operation is an extension of the left quotient of usual languages without lookahead. $\widetilde{x}^{-1}R$ is the left quotient for the remaining part. We write it as $\widetilde{x}^{-1}R$ using a letter with tilde as $\widetilde{x}$. Because it is the left quotient for the remaining part, it is the operation to remove x from the remaining part of a pair that does not consume any letters. Thus, it removes x only from a pair where the matching part is ε.

For one combination of the two left quotients, if $x, y \neq \varepsilon$, then $y^{-1}(\widetilde{x}^{-1}R) = \emptyset$ holds. For the other combination, we write $(x\widetilde{y})^{-1}R$ for $\widetilde{y}^{-1}(x^{-1}R)$, and $(x, y) \in R \iff (\varepsilon, \varepsilon) \in (x\widetilde{y})^{-1}R$ holds. From the above, in order to decide whether (x, y) is included in R, it is sufficient to calculate the left quotient of R by $x\widetilde{y}$ and check whether it includes the end of input $(\varepsilon, \varepsilon)$. If it is possible to calculate the left quotient on RELA and judge whether the end of input is included, then we can implement matching of RELA.

For the left quotient by letter a, the following holds.

Lemma 2.4.1.

$$a^{-1}(R \cup S) = (a^{-1}R) \cup (a^{-1}S)$$
$$a^{-1}(R \cdot S) = (a^{-1}R) \cdot S \cup (\widetilde{a}^{-1}R) \cdot (a^{-1}S)$$
$$a^{-1}(R^*) = (a^{-1}R) \cdot R^*$$
$$a^{-1}(!R) = \emptyset$$

Proof. This lemma follows from the equational reasoning on sets. In the case of concatenation, $R \cdot S = (R \setminus I) \cdot S \cup (R \cap I) \cdot S$ is used. In the case of star, $R^* = I \cup (R \setminus I) \cdot R^*$ is used. □

For the left quotient by letter $\widetilde{a}$, the following holds.

Lemma 2.4.2.

$$\widetilde{a}^{-1}(R \cup S) = (\widetilde{a}^{-1}R) \cup (\widetilde{a}^{-1}S)$$
$$\widetilde{a}^{-1}(R \cdot S) = (\widetilde{a}^{-1}R) \cdot (\widetilde{a}^{-1}S)$$
$$\widetilde{a}^{-1}(R^*) = I$$
$$\widetilde{a}^{-1}(!R) = !(a^{-1}R \cup \widetilde{a}^{-1}R)$$

Proof. This lemma follows from the equational reasoning on sets. In the case of star, $R^* = I \cup (R \setminus I) \cdot R^*$ is used. In the case of negative lookahead, $R = (R \setminus I) \cup (R \cap I)$ is used. □

Derivatives calculate the left quotient on RELA. There are two types of derivatives, D_a and $D_{\widetilde{a}}$, both of which are functions from RELA to RELA and are defined by mutual recursion.

Definition 2.4.1. The *derivative of e by a*, written $D_a e$, is defined as follows.

$$\begin{aligned}
D_a\emptyset &= \emptyset \\
D_a\varepsilon &= \emptyset \\
D_a b &= \begin{cases} \varepsilon & (a = b) \\ \emptyset & (a \neq b) \end{cases} \\
D_a(e_1|e_2) &= D_a e_1 | D_a e_2 \\
D_a(e_1 e_2) &= (D_a e_1) e_2 | (D_{\widetilde{a}} e_1)(D_a e_2) \\
D_a(e^*) &= (D_a e) e^* \\
D_a(!e) &= \emptyset
\end{aligned}$$

The *derivative of e by $\widetilde{a}$*, written $D_{\widetilde{a}} e$, is defined as follows.

$$\begin{aligned}
D_{\widetilde{a}}\emptyset &= \emptyset \\
D_{\widetilde{a}}\varepsilon &= \varepsilon \\
D_{\widetilde{a}} b &= \emptyset \\
D_{\widetilde{a}}(e_1|e_2) &= D_{\widetilde{a}} e_1 | D_{\widetilde{a}} e_2 \\
D_{\widetilde{a}}(e_1 e_2) &= (D_{\widetilde{a}} e_1)(D_{\widetilde{a}} e_2) \\
D_{\widetilde{a}}(e^*) &= \varepsilon \\
D_{\widetilde{a}}(!e) &= !(D_a e | D_{\widetilde{a}} e)
\end{aligned}$$

These derivatives are extensions of derivatives of regular expressions. Derivatives of regular expressions require an auxiliary function. In contrast, in RELA, $D_{\widetilde{a}}$ is an extension of the auxiliary function, and the two derivatives only require the corresponding functions.

Example 2.4.1. $D_a(a^*), D_a(a!b)$, and $D_{\widetilde{b}}(!b)$ are calculated as follows.

$$\begin{aligned}
D_a(a^*) &= (D_a a) a^* = \varepsilon a^* \\
D_a(a!b) &= (D_a a)!b | (D_{\widetilde{a}} a)(D_a b) = \varepsilon !b | \emptyset\emptyset \\
D_{\widetilde{b}}(!b) &= !(D_b b | D_{\widetilde{b}} b) = !(\varepsilon|\emptyset)
\end{aligned}$$

The following theorem is obtained from the lemma of the left quotient.

Theorem 2.4.3. *(Correctness of Derivatives)*

$$\begin{aligned}
B(D_a e) &= a^{-1} B(e) \\
B(D_{\widetilde{a}} e) &= \widetilde{a}^{-1} B(e)
\end{aligned}$$

Derivatives are also extended to a string $w \in (\Sigma \cup \widetilde{\Sigma})^*$.

$$D_\varepsilon e = e$$

$$D_{cw}e = D_w(D_c e) \qquad (c \in \Sigma \cup \widetilde{\Sigma},\ w \in (\Sigma \cup \widetilde{\Sigma})^*)$$

For the extended derivative, the following holds.

$$B(D_w e) = w^{-1}B(e)$$

In addition, from the definition of derivative, $D_{\tilde{a}}e$ is a logical expression of RELA. Therefore, if $v \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, then $D_v e$ is a logical expression of RELA.

2.5 Construction of DFA by Derivatives

We show that the set of states obtained by repeatedly applying derivatives is finite under some equivalence relation. Afterward, we show the upper bound $2^{2^{\|e\|}} + 1$ of the size of the set of states in the worst case and discuss the lower bound. Lastly, we show the conversion to DFA and prove several theorems as its corollaries.

2.5.1 Conversion to Automata

It is possible to decide whether an RELA accepts the end of input. ν is a function from RELA to Boolean values and is defined as follows.

$$\begin{aligned}
\nu(\emptyset) = \nu(a) &= \mathit{false} \\
\nu(\varepsilon) = \nu(e^*) &= \mathit{true} \\
\nu(e_1|e_2) &= \nu(e_1) \vee \nu(e_2) \\
\nu(e_1 e_2) &= \nu(e_1) \wedge \nu(e_2) \\
\nu(!e) &= \neg\, \nu(e)
\end{aligned}$$

$\nu(e)$ is true if and only if $(\varepsilon, \varepsilon) \in B(e)$. Therefore, matching is achieved: $\nu(D_{x\tilde{y}}e)$ determines whether $(x, y) \in B(e)$.

We describe the conversion to DFA. The set of states $Q(e)$ is defined as follows.

$$Q(e) = \{D_w e \mid w \in (\Sigma \cup \widetilde{\Sigma})^*\}$$

If $Q(e)$ is finite, then we can convert RELA to DFA on $\Sigma \cup \widetilde{\Sigma}$. The DFA by derivatives, $\mathcal{A}(e_0)$, is defined as follows.

$$\begin{aligned}
\mathcal{A}(e_0) &= \langle Q(e_0), \Sigma \cup \widetilde{\Sigma}, \delta, e_0, F \rangle \\
F &= \{e \in Q(e_0) \mid \nu(e)\} \\
\delta(e, c) &= D_c e
\end{aligned}$$

If $Q(e_0)$ is finite, then the following holds.

$$\begin{aligned}
&L(\mathcal{A}(e_0)) \subseteq \Sigma^* \widetilde{\Sigma}^* \\
&x\tilde{y} \in L(\mathcal{A}(e_0)) \iff (x, y) \in B(e_0)
\end{aligned}$$

However, $Q(e_0)$ is not necessarily finite.

If some equivalence relation $\equiv$ exists such that the following three conditions are true:

1. $Q(e)/\equiv$ is finite,
2. $e_1 \equiv e_2 \implies \delta(e_1, c) \equiv \delta(e_2, c)$,
3. $e_1 \equiv e_2 \implies (e_1 \in F \iff e_2 \in F)$,

then it is possible to construct the DFA on the quotient of $\mathcal{A}(e)$ by the equivalence relation. In the next subsection, we show that such an equivalence relation exists.

2.5.2 Finiteness of Equivalence Classes

A congruence on RELA is an equivalence relation that satisfies the following.

$$\begin{aligned} e_1 \equiv e_2, e_3 \equiv e_4 &\implies e_1|e_3 \equiv e_2|e_4 \\ e_1 \equiv e_2, e_3 \equiv e_4 &\implies e_1 e_3 \equiv e_2 e_4 \\ e_1 \equiv e_2 &\implies e_1^* \equiv e_2^* \\ e_1 \equiv e_2 &\implies !e_1 \equiv !e_2 \end{aligned}$$

We define an equivalence relation that satisfies the three conditions presented in the previous subsection.

Definition 2.5.1. Let $\equiv$ be a congruence generated by the following rule.

(1)	$e_1\|(e_2\|e_3) \equiv (e_1\|e_2)\|e_3$	($\|$ assoc)
(2)	$e_1\|e_2 \equiv e_2\|e_1$	($\|$ comm)
(3)	$e\|e \equiv e$	($\|$ idem)
(4)	$e\|\emptyset \equiv e,\ \emptyset\|e \equiv e$	($\emptyset$ unit)
(5)	$e\varepsilon \equiv e,\ \varepsilon e \equiv e$	(ε unit)
(6)	$e\emptyset \equiv \emptyset,\ \emptyset e \equiv \emptyset$	($\emptyset$ zero)
(7)	$(e_1\|e_2)e_3 \equiv (e_1e_3)\|(e_2e_3)$	(right dist)
(8)	$e_1(e_2\|e_3) \equiv (e_1e_2)\|(e_1e_3)$	(left dist)
(9)	$e_1(e_2e_3) \equiv (e_1e_2)e_3$	($\cdot$ assoc)
(10)	$l_1l_2 \equiv l_2l_1$	($\cdot$ comm)
(11)	$ll \equiv l$	($\cdot$ idem)
(12)	$!\emptyset \equiv \varepsilon$	($!\emptyset$ rule)
(13)	$!\varepsilon \equiv \emptyset$	($!\varepsilon$ rule)
(14)	$!(e_1\|e_2) \equiv !e_1!e_2$	($!\|$ rule)
(15)	$!(le) \equiv !l\|!e$	($!\cdot$ rule)

(1)–(9) are the rules of the idempotent semiring. (10)–(13) are some rules of Boolean algebra. (14) and (15) are the rules corresponding to properties of negative lookahead.

For regular expressions, the sufficient set of rules to prove finiteness is (1)–(6). In Brzozowski's paper [9], it is shown that (1)–(3) are sufficient. However, unlike our definition, the derivative of concatenation e_1e_2 is defined by case analysis on the auxiliary function. In order to prove the number of states is bounded by an exponential function, it is sufficient to add right distributivity (7). For RELA, the sufficient set of rules to prove finiteness is (1)–(6) and (8)–(11). The sufficient set of rules to prove that the number of states is bounded by a double exponential function is (1)–(15).

For the congruence $\equiv$, the following holds.

$$\begin{aligned} e_1 \equiv e_2 &\implies B(e_1) = B(e_2) \\ e_1 \equiv e_2 &\implies \nu(e_1) = \nu(e_2) \\ e_1 \equiv e_2 &\implies D_c e_1 \equiv D_c e_2 \end{aligned}$$

The proposition for B follows from the discussion on l-languages in Section 2.2. The proposition for ν follows from the proposition for B and $\nu(e) \iff (\varepsilon, \varepsilon) \in B(e)$. The proposition for D_c is proved by induction on $\equiv$.

We show that $Q(e_0)/\equiv$ is finite. First, we find the normal form of $D_w e$. For $\emptyset, \varepsilon$, and a, the following holds.

$$\begin{aligned} D_w\emptyset &= \emptyset \\ D_w\varepsilon &= \begin{cases} \varepsilon & (w \in \widetilde{\Sigma}^*) \\ \emptyset & (otherwise) \end{cases} \\ D_w a &= \begin{cases} a & (w = \varepsilon) \\ \varepsilon & (w \in a\widetilde{\Sigma}^*) \\ \emptyset & (otherwise) \end{cases} \end{aligned}$$

Moreover, for $e_1|e_2$, the following holds.

$$D_w(e_1|e_2) = D_w e_1 | D_w e_2$$

We define the abbreviation $\sum$ as follows:

$$\sum_{i=1}^{n} e_i = e_1|\ldots|e_n,$$

where $\sum_{i=1}^{n} e_i = \emptyset$ if $n = 0$. In addition, we write $\sum_{i=1}^{n} e_i\, e'_i$ for $\sum_{i=1}^{n}(e_i\, e'_i)$. For derivatives, $D_a(\sum_{i=1}^{n} e_i) = \sum_{i=1}^{n}(D_a e_i)$ and $D_{\widetilde{a}}(\sum_{i=1}^{n} e_i) = \sum_{i=1}^{n}(D_{\widetilde{a}} e_i)$ hold.

In the case of concatenation, we find the following normal form of $D_w e$.

Lemma 2.5.1. *For any w, there exist some n, $v_i \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \neq \varepsilon$ such that the following holds.*

$$D_w(e_1 e_2) \equiv \begin{cases} (D_w e_1)e_2 | \sum_{i=1}^{n}(D_{v_i} e_1)(D_{w_i} e_2) & (w \in \Sigma^*) \\ \sum_{i=1}^{n}(D_{v_i} e_1)(D_{w_i} e_2) & (otherwise) \end{cases}$$

Proof. This lemma is proved by induction on w. In the case of ε, we use the fact $\emptyset$ is the unit element. That is, $D_\varepsilon(e_1 e_2) \equiv (D_\varepsilon e_1)e_2|\emptyset$. In the case of wc, we use the associativity of $|$. We show the case where $c = a$, and $D_w(e_1 e_2) \equiv (D_w e_1)e_2|\sum_{i=1}^{n}(D_{v_i} e_1)(D_{w_i} e_2)$.

$$\begin{aligned} D_{wa}(e_1 e_2) &\equiv D_a((D_w e_1)e_2|\textstyle\sum_{i=1}^{n}(D_{v_i} e_1)(D_{w_i} e_2)) \\ &\equiv (((D_{wa} e_1)e_2|(D_{w\widetilde{a}} e_1)(D_a e_2))| \\ &\qquad \textstyle\sum_{i=1}^{n}((D_{v_i a} e_1)(D_{w_i} e_2)|(D_{v_i \widetilde{a}} e_1)(D_{w_i a} e_2))) \\ &\equiv (D_{wa} e_1)e_2|\textstyle\sum_{i=1}^{2n+1}(D_{v'_i} e_1)(D_{w'_i} e_2) \end{aligned}$$

□

In the case of negative lookahead, we find the following normal form of $D_w e$.

Lemma 2.5.2. *For any w, there exist n and $w_i \neq \varepsilon$ such that the following holds.*

$$D_w(!e) \equiv \begin{cases} !((\sum_{i=1}^{n} D_{w_i} e)|D_w e) & (w \in \widetilde{\Sigma}^*) \\ \emptyset & (\mathit{otherwise}) \end{cases}$$

Proof. This is proved by induction on w. It is proved in a manner similar to the case of concatenation. □

We define the abbreviation $\prod$ as follows:

$$\prod_{i=1}^{n} e_i = e_1 \dots e_n,$$

where $\prod_{i=1}^{n} e_i = \varepsilon$ if $n = 0$. For derivatives, $D_{\widetilde{a}}(\prod_{i=1}^{n} e_i) = \prod_{i=1}^{n}(D_{\widetilde{a}} e_i)$ holds. Moreover, we have $D_a(\prod_{j=1}^{n} e_j) \equiv \sum_{i=1}^{n}(\prod_{j=1}^{n}(D_{w_{ij}} e_j))$, where $w_{ij} = \widetilde{a}\ (i > j)$, $a\ (i = j)$, $\varepsilon\ (i < j)$.

In the case of the star, we find the following form of $D_w e$.

Lemma 2.5.3. *For any w, there exist n, m_i, $v_{ij} \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \in \Sigma^+$ such that the following holds.*

$$D_w e^* \equiv \begin{cases} e^* & (w = \varepsilon) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i}(D_{v_{ij}} e))(D_{w_i} e)e^* & (w \in \Sigma^+) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i}(D_{v_{ij}} e)) & (\mathit{otherwise}) \end{cases}$$

Proof. This is proved by induction on w. The case of ε is clear. In the case of wc, we show the case where $c = a$, and $D_w e^* \equiv (D_v e)(D_{w_1} e)e^*$.

$$\begin{aligned} D_{wa} e^* &\equiv D_a((D_v e)(D_{w_1} e)e^*) \\ &\equiv (D_{va} e)(D_{w_1} e)e^* | (D_{v\widetilde{a}} e)(D_a((D_{w_1} e)e^*)) \\ &\equiv (D_{va} e)(D_{w_1} e)e^* | (D_{v\widetilde{a}} e)((D_{w_1 a} e)e^* | (D_{w_1 \widetilde{a}} e)(D_a e)e^*) \\ &\equiv (D_{va} e)(D_{w_1} e)e^* | (D_{v\widetilde{a}} e)(D_{w_1 a} e)e^* | (D_{v\widetilde{a}} e)(D_{w_1 \widetilde{a}} e)(D_a e)e^* \\ &\equiv (D_{v_{11}} e)(D_{w_1} e)e^* | (D_{v_{21}} e)(D_{w_2} e)e^* | (D_{v_{31}} e)(D_{v_{32}} e)(D_{w_3} e)e^* \end{aligned}$$

□

Theorem 2.5.4. *$Q(e)/\equiv$ is finite.*

Proof. Let $f(e) = |Q(e)/\equiv|$. Because $\equiv$ is idempotent and commutative, the following holds for each normal form of $D_w e$.

$$\begin{aligned} f(\emptyset) &= 1 \\ f(\varepsilon) &= 2 \\ f(a) &= 3 \\ f(e_1|e_2) &\leq f(e_1) \times f(e_2) \\ f(e_1 e_2) &\leq 2^{f(e_1) \times f(e_2)} \\ f(e^*) &\leq 2^{2^{2 \times f(e)}} + 1 \\ f(!e) &\leq 2^{f(e)} + 1 \end{aligned}$$

Hence, $Q(e)/\equiv$ is finite.

We explain the case of the star. From Lemma 2.5.3, there exist n, m_i, $v_{ij} \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, w_i, and $e_i \in \{(D_{w_i}e)e^*, \varepsilon\}$ such that the following holds.

$$D_w e^* \equiv \begin{cases} e^* & (w = \varepsilon) \\ \sum_{i=1}^{n} (\prod_{j=1}^{m_i} (D_{v_{ij}} e)) e_i & (otherwise) \end{cases}$$

Then, $D_{v_{ij}}e$ is a logical expression of RELA. Thus, the number of equivalence classes of $\prod_{j=1}^{m_i}(D_{v_{ij}}e)$ is less than or equal to $2^{f(e)}$. Because the number of equivalence classes of e_i is less than or equal to $f(e) + 1 \leq 2^{f(e)}$, the number of equivalence classes of $(\prod_{j=1}^{m_i}(D_{v_{ij}}e))e_i$ is less than or equal to $2^{f(e)} \times 2^{f(e)} = 2^{2\times f(e)}$. Therefore, the number of equivalence classes of $D_w e^*$ is less than or equal to $2^{2^{2\times f(e)}} + 1$. □

We have proved that $Q(e)/\equiv$ is finite; thus, we obtained a conversion to DFA by derivatives. In the next two subsections, we discuss the number of states of the DFA. In the subsection of application, we show several theorems as corollaries of the conversion to DFA.

2.5.3 Upper Bound

We have proved $Q(e)/\equiv$ is finite. Although it is sufficient for DFA construction, we show a stronger result that

$$|Q(e)/\equiv| \leq 2^{2^{\|e\|}} + 1.$$

This is shown in the following manner. First, for $w \neq \varepsilon$, we represent $D_w e$ in the form $\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i$. Next, we identify the sets of expressions $V_m(e)$ and $V_l(e)$ that appear as e_i, l_{ij}. Lastly, we show $|Q(e)/\equiv| \leq 2^{2^{\|e\|}} + 1$ because $|V_m(e)| + |V_l(e)| \leq \|e\|$.

We generalize concatenation and negative lookahead. For a set E_1 of RELA and an RELA e_2, $E_1 e_2 = \{e_1 e_2 \mid e_1 \in E_1\}$. For a set E of RELA, $!E = \{!e \mid e \in E\}$.

The set of RELA, $V_m(e)$, is defined as follows.

$$\begin{aligned} V_m(\emptyset) &= \emptyset \\ V_m(\varepsilon) &= \emptyset \\ V_m(a) &= \{\varepsilon\} \\ V_m(e_1|e_2) &= V_m(e_1) \cup V_m(e_2) \\ V_m(e_1 e_2) &= V_m(e_1)e_2 \cup V_m(e_2) \\ V_m(e^*) &= V_m(e)e^* \\ V_m(!e) &= \emptyset \end{aligned}$$

If r is a regular expression, $V_m(r) \cup \{r\}$ is the set of regular expressions obtained by repeatedly applying partial derivatives [3].

The set of RELA, $V_l(e)$, is defined as follows.

$$\begin{aligned} V_l(\emptyset) &= \emptyset \\ V_l(\varepsilon) &= \emptyset \\ V_l(a) &= \emptyset \\ V_l(e_1|e_2) &= V_l(e_1) \cup V_l(e_2) \end{aligned}$$

$$
\begin{aligned}
V_l(e_1e_2) &= V_l(e_1) \cup V_l(e_2) \\
V_l(e^*) &= V_l(e) \\
V_l(!e) &= !(V_m(e) \cup V_l(e))
\end{aligned}
$$

$V_l(e)$ is a concept unique to RELA because $V_l(r) = \emptyset$ for a regular expression r.

$V_m(e)$ and $V_l(e)$ satisfy $|V_m(e)| + |V_l(e)| \leq \|e\|$. We confirm this in the case of concatenation.

$$
\begin{aligned}
|V_m(e_1e_2)| + |V_l(e_1e_2)| &= |V_m(e_1)e_2 \cup V_m(e_2)| + |V_l(e_1) \cup V_l(e_2)| \\
&\leq |V_m(e_1)e_2| + |V_m(e_2)| + |V_l(e_1)| + |V_l(e_2)| \\
&\leq \|e_1\| + \|e_2\| = \|e_1e_2\|
\end{aligned}
$$

Lemma 2.5.5. *For any w, there exist n, m_i, $l_{ij} \in V_l(e)$, and $e_i \in V_m(e)$ such that the following holds.*

$$
D_we \equiv \begin{cases} e & (w = \varepsilon) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i & (w \in \Sigma^+) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij}) & (otherwise) \end{cases}
$$

Proof. This is proved by induction on e. We use associativity and distributivity. We show a part of the case of concatenation. From Lemma 2.5.1, there exist N, $v_i \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \neq \varepsilon$ such that the following holds.

$$
D_w(e_1e_2) \equiv \begin{cases} (D_we_1)e_2|\sum_{i=1}^{N}(D_{v_i}e_1)(D_{w_i}e_2) & (w \in \Sigma^*) \\ \sum_{i=1}^{N}(D_{v_i}e_1)(D_{w_i}e_2) & (otherwise) \end{cases}
$$

We prove the case of $w \in \Sigma^*$ and $N = 1$. For the first half part $(D_we_1)e_2$, from induction hypothesis, the following holds.

$$
\begin{aligned}
(D_we_1)e_2 &\equiv (\textstyle\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i')e_2 \\
&\equiv \textstyle\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})(e_i'e_2)
\end{aligned}
$$

For the latter half part $(D_{v_1}e_1)(D_{w_1}e_2)$, we use the following equality.

$$
\begin{aligned}
(D_{v_1}e_1)(D_{w_1}e_2) &\equiv (\textstyle\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij}))(\sum_{k=1}^{n'}(\prod_{j=1}^{m_k'} l_{kj}')e_k') \\
&\equiv \textstyle\sum_{i=1}^{n}(\sum_{k=1}^{n'}(\prod_{j=1}^{m_i} l_{ij})(\prod_{j=1}^{m_k'} l_{kj}')e_k')
\end{aligned}
$$

The other cases are the same except for the negative lookahead. For the case of negative lookahead, the following holds from the rule of congruence for negative lookahead.

$$
\begin{aligned}
!(D_{w_1}e) &\equiv !(\textstyle\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i) \\
&\equiv \textstyle\prod_{i=1}^{n}!((\prod_{j=1}^{m_i} l_{ij})e_i) && \text{(rules (12) (14))} \\
&\equiv \textstyle\prod_{i=1}^{n}((\sum_{j=1}^{m_i} !l_{ij})|!e_i) && \text{(rules (13) (15))}
\end{aligned}
$$

Then, this case is shown by using associativity and distributivity. □

Theorem 2.5.6. $|Q(e)/\equiv| \leq 2^{2^{\|e\|}} + 1$

Proof. From the above lemma, for any w, there exist n, m_i, $l_{ij} \in V_l(e)$, and $e_i \in (V_m(e) \cup \{\varepsilon\})$ such that the following holds.

$$
D_we \equiv \begin{cases} e & (w = \varepsilon) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i & (otherwise) \end{cases}
$$

Therefore, we have the following.

$$\begin{aligned}|Q(e)/\equiv| &\leq 2^{2^{|V_l(e)|}(|V_m(e)+1|)}+1\\ &\leq 2^{2^{|V_l(e)|+|V_m(e)|}}+1\\ &\leq 2^{2^{\|e\|}}+1\end{aligned}$$

This completes the proof of the theorem. □

In the proof of the upper bound, the form $\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i$ is essential. As a result, it is expected that the double exponential size is essential, and the number of states cannot be bounded by an exponential. In the next section, we show that this intuition is correct.

2.5.4 Worst Case Lower Bound

When we convert an RELA to a DFA, we estimate a lower bound of the number of states of the DFA in the worst case. We give a lower bound of $2^{2^{\Omega(\sqrt{m})}}$ where m is the size of RELA.

Let p_i be an i th prime number; we consider the following RELA.

$$T_n = .^*a\&((.^{p_1})^*\$)\ldots\&((.^{p_n})^*\$).^*\$$$

$B(T_n)$ is $\Sigma^*a(\Sigma^{p_1\times\cdots\times p_n})^*\times\{\varepsilon\}$. The language of the corresponding DFA $L(\mathcal{A})$ is $\Sigma^*a(\Sigma^{p_1\times\cdots\times p_n})^*$.

For the size of T_n, the following holds.

Lemma 2.5.7. *The size of RELA T_n is $O(p_n^2)$.*

Proof. The size of T_n is $O(\sum_{i=1}^{n} p_i)$. If $i<j$ then $p_i<p_j$. Hence, $\sum_{i=1}^{n} p_i \leq \sum_{i=1}^{p_n} i \leq p_n^2$. □

For the size of DFA, the following holds.

Lemma 2.5.8. *The number of states of the minimum DFA on $(\Sigma\cup\widetilde{\Sigma})^*$ that satisfies $L(\mathcal{A})=\Sigma^*a(\Sigma^N)^*$ is 2^N+1.*

Proof. 2^N states are needed to memorize whether a came at the $i+$ (multiple of N) th position for $1\leq i\leq N$. In addition, the state of the empty set is necessary because the automaton does not accept strings containing $\widetilde{a}$. Thus, 2^N+1 states are necessary. □

We use the following lemma proved in [35].

Lemma 2.5.9. $\prod_{i=1}^{n} p_i = 2^{\Omega(p_n)}$

From the presented lemmas, the following theorem is obtained.

Theorem 2.5.10. *The lower bound of the number of states of DFA in the worst case is $2^{2^{\Omega(\sqrt{m})}}$, where m is the size of an RELA.*

Proof. From Lemma 2.5.7, the size of T_n is $O(p_n^2)$. The number of states corresponding to the minimum DFA is $2^{\prod_{i=1}^{n} p_i}+1$. It is $2^{2^{\Omega(p_n)}}$. □

As a result, we have given a lower bound $2^{2^{\Omega(\sqrt{m})}}$.

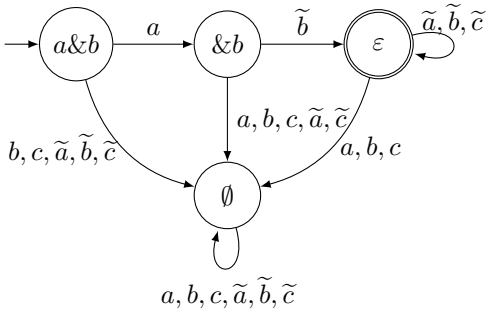


Fig. 2.1 DFA for $a\&b$

Morihata pointed out that a better lower bound is obtained by the application of research on XPath [28]. The lower bound is $2^{2^{\Omega(m)}}$ if the dot expression "." is introduced as a primitive. For $\Sigma = \{a, b, c, x_1, \ldots, x_m\}$, he analyzed $.^*a(b|.\&(.^*x_1)) \ldots (b|.\&(.^*x_m)).^*\$$.

The following results were shown in related work. The lower bound $2^{2^{\Omega(m)}}$ is shown for semi-extended regular expressions that are extended by intersection [18]. The lower bound is not bounded by an elementary function for extended regular expressions that are extended by complement [38].

As pointed out by Morihata [27], it is worth noting that despite RELA have negative lookahead, the size of the corresponding DFA can still be bounded by a double exponential.

2.5.5 Applications

We convert the RELA e_0 to DFA $\mathcal{A}(e_0)/\equiv$ on $\Sigma \cup \widetilde{\Sigma}$. $\mathcal{A}(e_0)/\equiv$ is defined as follows:

$$\begin{aligned}(\mathcal{A}(e_0)/\equiv) &= \langle Q(e_0)/\equiv, \Sigma \cup \widetilde{\Sigma}, \delta, [e_0], F\rangle,\\ F &= \{[e] \in Q(e_0)/\equiv \mid \nu(e)\},\\ \delta([e], c) &= [D_c e],\end{aligned}$$

where $[e]$ is equivalence classes defined as $[e] = \{e' \mid e \equiv e'\}$.

$Q(e_0)/\equiv$ is a finite set by Theorem 2.5.4. In addition, F and δ are well-defined. Thus, $\mathcal{A}(e_0)/\equiv$ is a DFA.

Theorem 2.5.11. *(Correctness of DFA)*

$$\begin{aligned}&L(\mathcal{A}(e_0)/\equiv) \subseteq \Sigma^*\widetilde{\Sigma}^*\\ &x\widetilde{y} \in L(\mathcal{A}(e_0)/\equiv) \iff (x, y) \in B(e_0)\end{aligned}$$

Example 2.5.1. When $\Sigma = \{a, b, c\}$, the DFA converted from $a\&b$ is shown in Figure 2.1.

Several theorems are obtained as corollaries.

Corollary 2.5.12. *(Matching) Let $\mathcal{A}$ be the DFA converted from the RELA e, n be the length of x, and m be the number of states. It is decidable whether $x \in L(e)$ in $O(n)$ time, and it is possible to calculate $\{(x_1, x_2) \in B(e) \mid x = x_1x_2\}$ in $O(nm)$ time.*

Proof. In order to determine whether $x \in L(e)$, it is sufficient to run $\mathcal{A}$ with input x. Since $\mathcal{A}$ is DFA,

Table 2.1 Experimental result

RELA	letters	states
a^*	1	3
$a\&b$	2	4
$(!(ab).)^*$	3	4
$ba(!(ab).)^*ab$	7	6
$\&(.^*a)\&(.^*b)(.^8).^*\$$	13	33
$.^*a\&((.^3)^*\$)\&((.^5)^*\$).^*\$$	11	32769

it is determined in $O(n)$ time. In order to calculate $\{(x_1, x_2) \in B(e) \mid x = x_1x_2\}$, when $x = a_1 \dots a_n$, for $0 \leq i \leq n$, it is sufficient to run $\mathcal{A}$ with input $a_1 \dots a_i\widetilde{a_{i+1}} \dots \widetilde{a_n}$. These can run in parallel, which is similar to the behavior of a nondeterministic automaton. The number of states at each step is bounded by the number of states m. Thus, it is calculated in $O(nm)$ time. □

Corollary 2.5.13. *(Backward Movement of Lookahead) For any RELA e, there exist regular expressions $r_0, \dots, r_n$, $s_0, \dots, s_n$ such that $B(e) = B(r_0\&(s_0\$)| \dots |r_n\&(s_n\$))$.*

Proof. Let $\mathcal{A}$ be the DFA converted from RELA e, $\mathcal{A} = \langle Q, \Sigma \cup \widetilde{\Sigma}, \delta, q_0, F\rangle$, and $Q = \{q_0, \dots, q_n\}$. Let $\mathcal{A}_i = \langle Q, \Sigma, \delta|_{Q\times\Sigma}, q_0, \{q_i\}\rangle$, and $\mathcal{B}_i = \langle Q, \widetilde{\Sigma}, \delta|_{Q\times\widetilde{\Sigma}}, q_i, F\rangle$. $L(\mathcal{A}) = \bigcup_{i=0}^{n} L(\mathcal{A}_i)L(\mathcal{B}_i)$ holds because $L(\mathcal{A}) \subseteq \Sigma^*\widetilde{\Sigma}^*$. Therefore, $L(\mathcal{A}) = L(r_0\widetilde{s_0}| \dots |r_n\widetilde{s_n})$ where r_i and $\widetilde{s_i}$ are the regular expressions converted from DFA $\mathcal{A}_i$ and $\mathcal{B}_i$. Thus, $B(e) = B(r_0\&(s_0\$)| \dots |r_n\&(s_n\$))$ from Theorem 2.5.11. □

Corollary 2.5.14. *(Regularity) For any RELA e, there exist regular languages $A_1, \dots, A_n, B_1, \dots, B_n$ such that $B(e) = \bigcup_{i=1}^{n}(A_i \times B_i)$. Therefore, $L(e)$ is a regular language.*

Proof. This follows from Corollary 2.5.13. □

A similar representation appears also in recognizable relation [8] and ω regular languages [22].

Corollary 2.5.15. *(Equivalence Checking) Given RELA e_1, e_2, it is decidable whether $B(e_1) = B(e_2)$.*

Proof. This can be determined by converting to DFA, minimizing, and comparing. □

2.6 Implementation

We implemented the conversion from RELA to DFA in the programming language Scala. It is easy to implement derivatives in programming languages having pattern matching, as indicated in [33].

In the previous section, states are equivalence classes, and each equivalence class is a possibly infinite set. Therefore, the implementation calculates representatives and makes them states. The computation of representatives is based on the implementation of derivatives of regular expressions in Isabelle/HOL [25].

Some examples were converted to DFA. Table 2.1 shows the number of states of obtained DFA. The fourth from the top is an expression for comments of C languages, the fifth is an expression similar to passwords, and the sixth is an example of state explosion. The number of letters of "." is counted as 1,

and the number of letters of \$ is counted as 0.

We describe one scheme used in our implementation. The expression $(a|b)(a|b)$ expands to $aa|ab|ba|bb$ by distributivity. We use literal classes rather than letters as primitives, and we represent the expression as $[ab][ab]$, preventing excessive expansion by the distributive law. Furthermore, we can handle letters and the dot expression uniformly as a special case of literal class.

Chapter 3

Context-Free Grammars with Lookahead and Limit Semantics

We introduce CFGLAs. We give the limit semantics with the limit of iterations from the empty set. We then show that the language class is closed under union, intersection, complement, marked concatenation, and marked Kleene star. We show that the grammars are an extension of both CFGs and PEGs.

3.1 Preliminaries

3.1.1 Complete Partially Ordered Set and Limit of Sets

A complete partially ordered set (CPO) is used in the denotational semantics of programs [31]. A subset of a partially ordered set is called a *chain* if it is a totally ordered set. A partially ordered set P is called a *CPO* if P has a least element and any chain A of P has the least upper bound $\bigsqcup A$. A function f is called a *monotonically increasing function* if $f(x) \leq f(y)$ holds for any $x \leq y$. A function f between two CPOs P and Q is called a *continuous function* if f is a monotonically increasing function and $f(\bigsqcup A) = \bigsqcup f(A)$ for any chain A. For two CPOs P and Q, the product set $P \times Q$ forms a CPO by the componentwise ordering, which is defined by $(p_1, q_1) \leq (p_2, q_2) \iff p_1 \leq p_2,\ q_1 \leq q_2$. The set of functions from a set to a CPO forms a CPO by pointwise ordering, which is defined by $f \leq g \iff f(x) \leq g(x)$ for any x. The set of languages $\mathcal{P}(\Sigma^*)$ with set inclusion $\subseteq$ forms a CPO.

Theorem 3.1.1 ([31]). *Let $f : P \to P$ be a continuous function on the CPO P with least element $\bot$. Then, $\bigsqcup_{n \geq 0} f^n(\bot)$ defines an element of P, and this element is the least fixed point of f.*

Let A_n be a sequence over sets. The *limit superior* $\overline{\lim}_{n\to\infty} A_n$ and the *limit inferior* $\underline{\lim}_{n\to\infty} A_n$ are defined by $\bigcap_{i\geq 0}\bigcup_{j\geq i} A_j$ and $\bigcup_{i\geq 0}\bigcap_{j\geq i} A_j$, respectively. If the limit superior and the limit inferior coincide, then A_n *converges* and the *limit* $\lim_{n\to\infty} A_n$ is defined and equal to the common value. For convergent sequences over sets A_n and B_n, we define the limit $\lim_{n\to\infty}(A_n, B_n)$ as $(\lim_{n\to\infty} A_n, \lim_{n\to\infty} B_n)$. Let f_n be a sequence over functions to sets. If for all x the limit $\lim_{n\to\infty} f_n(x)$ converges, then f_n converges and the limit is $(\lim_{n\to\infty} f_n)(x) = \lim_{n\to\infty} f_n(x)$. For a monotonically increasing sequence A_n, $\lim_{n\to\infty} A_n = \bigcup_{n\geq 0} A_n$.

3.1.2 Context-Free Grammars

The language of a CFG is often defined by rewriting or parse trees [37]. In this subsection, we explain the definition by the least solution [4, 12]. The semantics makes it easier to handle negative operations. We define the syntax of CFGs in a way that allows for nesting expressions, such as $X = (a|b)X \mid \varepsilon$.

First, the set of expressions including variables $\mathrm{RE}(\Sigma, V)$ is defined by the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid X \mid e|e \mid ee \quad (a \in \Sigma,\ X \in V)$$

The language of e is determined by defining the language of all variables. An *interpretation* is a function $L : V \to \mathcal{P}(\Sigma^*)$. It can be extended to the function $\hat{L} : \mathrm{RE}(\Sigma, V) \to \mathcal{P}(\Sigma^*)$ by $\hat{L}(\emptyset) = \emptyset$, $\hat{L}(\varepsilon) = \{\varepsilon\}$, $\hat{L}(a) = \{a\}$, $\hat{L}(X) = L(X)$, $\hat{L}(e_1|e_2) = \hat{L}(e_1) \cup \hat{L}(e_2)$, and $\hat{L}(e_1e_2) = \hat{L}(e_1) \cdot \hat{L}(e_2)$. The *empty interpretation* $L_\emptyset$ is defined by $L_\emptyset(X) = \emptyset$ for all $X \in V$.

A *context-free grammar* G is a tuple (V, Σ, P, S) where

- V is a set of variables,
- Σ is an alphabet disjoint from V,
- $P : V \to \mathrm{RE}(\Sigma, V)$ is a *production function*,
- $S \in V$ is a *start variable*.

An interpretation L is called a *solution* of P if L satisfies $L = \hat{L} \circ P$. Note that $L = \hat{L} \circ P$ represents $L(X) = \hat{L}(P(X))$ for any X. We define a function $[\![P]\!]$ by $[\![P]\!](L) = \hat{L} \circ P$. This is a continuous function on the CPO $\mathcal{P}(\Sigma^*)$. Thus, the least solution L_{CFG} can be defined by $L_{CFG} = \bigcup_{n \geq 0} [\![P]\!]^n(L_\emptyset)$. In addition, the following holds.

$$\begin{aligned} L_{CFG} &= \bigcup_{n \geq 0} [\![P]\!]^n(L_\emptyset) \\ &= \lim_{n \to \infty} [\![P]\!]^n(L_\emptyset) \\ &= \bigcap \{L \mid L = [\![P]\!](L)\}. \end{aligned}$$

The language of CFG G is defined by $\mathcal{L}(G) = L_{CFG}(S)$.

An extended production function $\hat{P} : \mathrm{RELA}(\Sigma, V) \to \mathrm{RELA}(\Sigma, V)$ is defined by $\hat{P}(\emptyset) = \emptyset$, $\hat{P}(\varepsilon) = \varepsilon$, $\hat{P}(a) = a$, $\hat{P}(X) = P(X)$, $\hat{P}(e_1|e_2) = \hat{P}(e_1)|\hat{P}(e_2)$, and $\hat{P}(e_1e_2) = \hat{P}(e_1)\hat{P}(e_2)$. The power of a production function is defined by $P^0(X) = X$ and $P^{n+1}(X) = \hat{P}(P^n(X))$. For $P(X) = XX \mid a$, the square of P is $P^2(X) = (XX|a)(XX|a) \mid a$. For an interpretation L and an expression e, $\widehat{[\![P]\!](L)}(e) = \hat{L}(\hat{P}(e))$ holds. Therefore, $[\![P]\!]^n(L_\emptyset)(X) = \hat{L}_\emptyset(P^n(X))$ for a variable X.

We write $G : X_1 = P(X_1),\ \ldots,\ X_n = P(X_n)$ for CFG $G = (\{X_1, \ldots, X_n\}, \Sigma, P, X_1)$, where Σ is the set of letters appearing in P.

Example 3.1.1. $G : X = aXb \mid \varepsilon$

$$\begin{aligned} &[\![P]\!]^1(L_\emptyset)(X) = \hat{L}_\emptyset(aXb|\varepsilon) = \{\varepsilon\} \\ &[\![P]\!]^2(L_\emptyset)(X) = \hat{L}_\emptyset(a(aXb|\varepsilon)b|\varepsilon) = \{\varepsilon, ab\} \\ &[\![P]\!]^3(L_\emptyset)(X) = \hat{L}_\emptyset(a(a(aXb|\varepsilon)b|\varepsilon)b|\varepsilon) = \{\varepsilon, ab, aabb\} \end{aligned}$$

$$[\![P]\!]^n(L_\emptyset)(X) = \{a^m b^m \mid m < n\}$$

Therefore, $\mathcal{L}(G) = \bigcup_{n\geq 0} [\![P]\!]^n(L_\emptyset)(X) = \{a^m b^m \mid m \geq 0\}$.

The calculation of P^n is similar to rewriting. Indeed, the semantics of both rewriting and least solution define the same language for a given grammar.

3.2 Context-Free Grammars with Lookahead

In this section, we introduce CFGLA and illustrate some examples. We discuss closure properties and show CFGLA is an extension of PEG [16].

First, we show the continuity of the operations of l-languages. Secondly, we introduce RELA including variables and their interpretation. Thirdly, we introduce CFGLA and define the l-language of CFGLA by the limit semantics, which is the limit of iterations from the empty set. A CFGLA is called valid if it converges. This technique is related to Boolean grammars [32] and partial fixed-point logic [41]. Lastly, we see that CFGLA is an extension of CFG and illustrate some examples.

We show the operations of l-languages are continuous with respect to the limit of sets. It is a standard result that union is continuous, i.e. $\lim_{n\to\infty}(R_n \cup S_n) = \lim_{n\to\infty} R_n \cup \lim_{n\to\infty} S_n$ for convergent sequences R_n and S_n. We show that concatenation and negative lookahead are also continuous.

Lemma 3.2.1. *For convergent sequences R_n and S_n, $\lim_{n\to\infty}(R_n \cdot S_n) = \lim_{n\to\infty} R_n \cdot \lim_{n\to\infty} S_n$.*

Proof. The key is that for a given a string pair r, there is a finite number of combinations of r_1 and r_2 such that $r = r_1 \cdot r_2$. First, if $r \in \overline{\lim_{n\to\infty}}(R_n \cdot S_n)$, then there are infinitely many n such that $r \in R_n \cdot S_n$. By the finiteness of combinations, there are r_1, r_2 and infinitely many n such that $r = r_1 \cdot r_2$, $r_1 \in R_n$, and $r_2 \in S_n$. Therefore, $\overline{\lim_{n\to\infty}}(R_n \cdot S_n) \subseteq (\overline{\lim_{n\to\infty}} R_n) \cdot (\overline{\lim_{n\to\infty}} S_n)$. Easily, $\underline{\lim_{n\to\infty}}(R_n \cdot S_n) \supseteq (\underline{\lim_{n\to\infty}} R_n) \cdot (\underline{\lim_{n\to\infty}} S_n)$. Thus, $\lim_{n\to\infty}(R_n \cdot S_n) = (\lim_{n\to\infty} R_n) \cdot (\lim_{n\to\infty} S_n)$. □

Lemma 3.2.2. *For convergent sequences R_n, $\lim_{n\to\infty} !R_n = !\lim_{n\to\infty} R_n$.*

Proof. We can show this using the following equations.

$$!(\bigcup_{n\in I} R_n) = \bigcap_{n\in I} !R_n \qquad\qquad !(\bigcap_{n\in I} R_n) = \bigcup_{n\in I} !R_n$$

□

The following lemma is useful to prove convergence of a sequence over sets.

Lemma 3.2.3. *Let R_n and M_n be sequences of sets, if $R_n \cap M_n$ converges and M_n converges to the universal set, then R_n converges and $\lim_{n\to\infty} R_n = \lim_{n\to\infty}(R_n \cap M_n)$.*

Proof. In general, $\underline{\lim_{n\to\infty}}(R_n \cap M_n) = \underline{\lim_{n\to\infty}} R_n \cap \underline{\lim_{n\to\infty}} M_n$ holds. If M_n converges to the universal set, then $\overline{\lim_{n\to\infty}}(R_n \cap M_n) = \overline{\lim_{n\to\infty}} R_n$. Therefore, R_n converges and $\lim_{n\to\infty} R_n = \lim_{n\to\infty}(R_n \cap M_n)$. □

Definition 3.2.1 (Regular expressions with lookahead)**.** Let Σ be an alphabet and V be a set of variables disjoint from Σ. The *regular expressions with lookahead* including variables $\mathrm{RELA}(\Sigma, V)$ are given by

the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid X \mid e|e \mid ee \mid !e \quad (a \in \Sigma,\ X \in V)$$

The lookahead has the highest priority, then concatenation and then alternation. For example, an expression $!XY \mid \varepsilon$ is equal to $((!X)Y) \mid \varepsilon$. The positive lookahead $\&e$ is an abbreviation for $!(!e)$. Any character "." is an abbreviation for $a_1|\ldots|a_n$ when $\Sigma = \{a_1, \ldots, a_n\}$. The end of a string \$ is an abbreviation for "!.". The size of an expression $|e|$ is defined by $|\emptyset| = |\varepsilon| = |a| = |X| = 1$, $|e_1|e_2| = |e_1e_2| = |e_1| + |e_2| + 1$, and $|!e| = |e| + 1$.

An *interpretation* is a function $B : V \to \mathcal{P}(\Sigma^* \times \Sigma^*)$. For an interpretation B, the extended interpretation $\hat{B} : \mathrm{RELA}(\Sigma, V) \to \mathcal{P}(\Sigma^* \times \Sigma^*)$ is defined by the following.

$$\hat{B}(\emptyset) = \emptyset,\ \hat{B}(\varepsilon) = \{\varepsilon\} \times \Sigma^*,\ \hat{B}(a) = \{a\} \times \Sigma^*,$$
$$\hat{B}(X) = B(X),\ \hat{B}(e_1|e_2) = \hat{B}(e_1) \cup \hat{B}(e_2),$$
$$\hat{B}(e_1e_2) = \hat{B}(e_1) \cdot \hat{B}(e_2),\ \hat{B}(!e) = !\hat{B}(e)$$

The *empty interpretation* $B_\emptyset$ is defined by $B_\emptyset(X) = \emptyset$ for all $X \in V$. For example, $B_\emptyset(a\,\&b) = \{(a, bx) \mid x \in \Sigma^*\}$ and $B_\emptyset(!X) = !\emptyset = \{\varepsilon\} \times \Sigma^*$. For two interpretations B_1 and B_2, we define $B_1 \subseteq B_2$ by $B_1(X) \subseteq B_2(X)$ for all X. For an interpretation B and $e \in \mathrm{RELA}(\Sigma, V)$, the *language* $L_B(e)$ is defined as $\{x \mid (x, \varepsilon) \in B(e)\}$. The language is the set of strings that the end of a string can follow. We have $B(e\$) = L_B(e) \times \{\varepsilon\}$ and $B(\&(e\$)) = \{\varepsilon\} \times L_B(e)$. If e does not contain lookahead, $B_\emptyset(e) = L_{B_\emptyset}(e) \times \Sigma^*$.

Definition 3.2.2 (Context-free grammars with lookahead)**.** A *context-free grammar with lookahead* G is a tuple (V, Σ, P, S) where

- V is a set of variables,
- Σ is an alphabet disjoint from V,
- $P : V \to \mathrm{RELA}(\Sigma, V)$ is a *production function*,
- $S \in V$ is a *start variable*.

An extended production function $\hat{P} : \mathrm{RELA}(\Sigma, V) \to \mathrm{RELA}(\Sigma, V)$ is defined by $\hat{P}(\emptyset) = \emptyset$, $\hat{P}(\varepsilon) = \varepsilon$, $\hat{P}(a) = a$, $\hat{P}(X) = P(X)$, $\hat{P}(e_1|e_2) = \hat{P}(e_1)|\hat{P}(e_2)$, $\hat{P}(e_1e_2) = \hat{P}(e_1)\hat{P}(e_2)$, and $\hat{P}(!e) = !\hat{P}(e)$. The power of a production function is defined by $P^0(X) = X$ and $P^{n+1}(X) = \hat{P}(P^n(X))$. For example, if $P(X) = aXb|\varepsilon$, then $P^2(X) = a(aXb|\varepsilon)b|\varepsilon$. The size of a CFGLA $|G|$ is defined as $\sum_{X \in V} |P(X)|$.

A production function can be regarded as a system of equations. An interpretation B is called a *solution* of P if B satisfies $B = \hat{B} \circ P$. Note that $B = \hat{B} \circ P$ represents $B(X) = \hat{B}(P(X))$ for any X. We define a function $[\![P]\!]$ by $[\![P]\!](B) = \hat{B} \circ P$.

Lemma 3.2.4. *For an interpretation B, if a sequence $\hat{B} \circ P^n$ converges, then the limit $B_{P,nat} = \lim_{n\to\infty}(\hat{B} \circ P^n)$ is a solution of P.*

Proof. First, we show $\hat{B}_{P,nat} = \lim_{n\to\infty}(\hat{B} \circ \hat{P}^n)$. We show the case of concatenation.

$$\begin{aligned}\lim_{n\to\infty} \hat{B}(\hat{P}^n(e_1e_2)) &= \lim_{n\to\infty} \hat{B}(\hat{P}^n(e_1)\hat{P}^n(e_2)) && \text{(induction on } n\text{)}\\ &= \lim_{n\to\infty} (\hat{B}(\hat{P}^n(e_1)) \cdot \hat{B}(\hat{P}^n(e_2)))\end{aligned}$$

$$= \lim_{n\to\infty} \hat{B}(\hat{P}^n(e_1)) \cdot \lim_{n\to\infty} \hat{B}(\hat{P}^n(e_2)) \qquad \text{(Lemma 3.2.1)}$$

Next, we have $\lim_{n\to\infty} (\hat{B} \circ \hat{P}^n) \circ P = \lim_{n\to\infty} (\hat{B} \circ P^{n+1}) = \lim_{n\to\infty} (\hat{B} \circ \hat{P}^n)$. Thus, $B_{P,nat}$ is a solution. □

The *limit semantics* is defined by $B_{P,nat} = \lim_{n\to\infty} (B_\emptyset \circ P^n)$. A CFGLA is called *valid* if $B_{P,nat}$ exists. A CFGLA is called *uniquely convergent* if for all interpretation B, the limit $\lim_{n\to\infty} (B \circ P^n)$ converges to the same value. A uniquely convergent CFGLA is valid and has a unique solution.

Definition 3.2.3 (Language with lookahead)**.** The *language with lookahead* (l-language) of a valid CFGLA $G = (V, \Sigma, P, S)$ is the set $\mathcal{B}(G) = B_{P,nat}(S)$ and the *language* of G is the set $\mathcal{L}(G) = L_{B_{P,nat}}(S)$.

For a CFGLA without lookahead $G = (V, \Sigma, P, S)$, the function $[\![P]\!]$ is a monotone function, hence $\mathcal{B}(G) = \bigcup_{n\geq 0} (B_\emptyset \circ P^n)$. In addition, $\mathcal{B}(G) = \mathcal{L}(G) \times \Sigma^*$ holds. Therefore, the language of CFGLA without lookahead and the language of CFG coincide. Thus, CFGLA is an extension of CFG.

Example 3.2.1 (Valid Grammars)**.**

1. A CFGLA $X = aXb \mid \varepsilon$ is uniquely convergent. The limit semantics is calculated as follows.

$$\begin{aligned}
&B_\emptyset(P^0(X)) = B_\emptyset(X) = \emptyset,\\
&B_\emptyset(P^1(X)) = B_\emptyset(aXb|\varepsilon) = \{\varepsilon\} \times \Sigma^*,\\
&B_\emptyset(P^2(X)) = B_\emptyset(a(aXb|\varepsilon)b|\varepsilon) = \{\varepsilon, ab\} \times \Sigma^*,\\
&B_\emptyset(P^3(X)) = B_\emptyset(a(a(aXb|\varepsilon)b|\varepsilon)b|\varepsilon) = \{\varepsilon, ab, aabb\} \times \Sigma^*.
\end{aligned}$$

   Therefore, $B_{P,nat}(X) = \lim_{n\to\infty} B_\emptyset(P^n(X)) = \{a^n b^n \mid n \geq 0\} \times \Sigma^*$.
2. A CFGLA $X = X$ is valid and $B_{P,nat}(X) = \emptyset$. It is the same as CFG.
3. A CFGLA "$X = !Y,\ Y = \&Y$" is valid. For any language L, an interpretation $B(X) = \{\varepsilon\} \times L^c,\ B(Y) = \{\varepsilon\} \times L$ is a solution. Unlike in the case of CFG, there is no least solution. The limit semantics is $B_{P,nat}(X) = \{\varepsilon\} \times \Sigma^*,\ B_{P,nat}(Y) = \emptyset$.
4. A CFGLA $X = !(aX)$ is valid. When $\Sigma = \{a\}$, we have the following.

$$B_\emptyset(P^n(X)) = \begin{cases} \{\varepsilon\} \times \{a^i \mid i < n,\ i \text{ is even}\} & (n \text{ is even}) \\ \{\varepsilon\} \times (\{a\}^* \setminus \{a^i \mid i < n,\ i \text{ is odd}\}) & (n \text{ is odd}) \end{cases}$$

   Therefore, $B_{P,nat}(X) = \{(\varepsilon, a^{2n}) \mid n \geq 0\}$. A proper right recursion in negative lookahead is allowed.
5. A CFGLA G defined by

$$S = a\&(Y(a|\$))XSb \mid ab,\quad X = bX \mid b,\quad Y = bYb \mid a$$

   is valid and $\mathcal{L}(G) = \{(ab^n)^n \mid n \geq 1\}$. If you ignore the lookahead part, the language is $\{ax_1 \dots ax_{n-1}ab^n \mid n \geq 1,\ x_1, \dots, x_{n-1} \in \{b\}^*\}$. Lookahead requires that all lengths of x_i be equal.
6. For $k \geq 1$, let G be $(\{X\}, \{a\}, P, X)$, where $P(X) = a\$|\&(X\$)|a^{k-1}Xa$. The CFGLA G is valid, $\mathcal{B}(G) = \{(a^n, a^m) \mid \exists l.\ n + km = k^l\}$, and $\mathcal{L}(G) = \{a^{k^n} \mid n \geq 0\}$. This grammar takes advantage of the property of lookahead and contains only positive lookahead without the end of a string. This example is essentially the same as Example 4 in [6].

Example 3.2.2 (Invalid Grammars)**.**

1. A CFGLA $X = !X$ has no solution, hence it is invalid. Indeed, if it has a solution $\{\varepsilon\} \times L$, then $L = L^c$ holds, which is a contradiction.
2. A CFGLA $X = !(Xx)$ has no solution because $(\varepsilon, x) \in B(X)$ if and only if $(\varepsilon, x) \notin B(!(Xx))$. A left recursion in negative lookahead is not allowed.
3. A CFGLA "$S = !X!S,\ X = \&X$" is invalid but has a unique solution $B(S) = \emptyset, B(X) = \{\varepsilon\} \times \Sigma^*$. Thus, validity and the existence of solutions are different. Also, uniquely convergence and the existence of a unique solution are different.
4. A CFGLA "$X = !Y, Y = !X$" is invalid. For any language L, an interpretation $B(X) = \{\varepsilon\} \times L^c,\ B(Y) = \{\varepsilon\} \times L$ is a solution. However, iterations from the empty set oscillate between $\emptyset$ and $\{\varepsilon\} \times \Sigma^*$ and do not converge. This invalid CFGLA is obtained by introducing a variable Y from a valid grammar $X = !(!X)$. The introduction of new variables does not change the set of solutions but change the convergence.

What kind of grammars are valid? We conjecture that a CFGLA is valid if there is no left recursion through negative lookahead. Intuitively, there are no such patterns $X = !(Xx)$ or "$X = !Y, Y = !X$". It is easy to see that a CFGLA with only positive lookahead is valid because the function $[\![P]\!]$ is a monotone function.

3.3 Closure Properties

We write $\mathcal{B}$(CFGLA) for the class of l-languages of CFGLA and $\mathcal{L}$(CFGLA) for the class of languages of CFGLA. First, we show that $\mathcal{B}$(CFGLA) is closed under union, concatenation, lookahead, and Kleene star. Next, we show that $\mathcal{L}$(CFGLA) is closed under union, intersection, complement, and the weak version of concatenation and Kleene star. Lastly, we show undecidability of emptiness checking.

Lemma 3.3.1. *For two valid CFGLA $G_1 = (V_1, \Sigma, P_1, S_1)$ and $G_2 = (V_2, \Sigma, P_2, S_2)$, if $V_1 \subseteq V_2$ and $P_1(X) = P_2(X)$ for all $X \in V_1$, then $B_{P_1,nat}(X) = B_{P_2,nat}(X)$ for any $X \in V_1$.*

Proof. For any $e \in \mathrm{RELA}(\Sigma, V_1)$, $P_1(e) = P_2(e)$ by induction on e. For any $X \in V_1$, $P_1^n(X) = P_2^n(X)$ by induction on n, thus $B_{P_1,nat}(X) = B_{P_2,nat}(X)$. □

Corollary 3.3.2. *$\mathcal{B}$(CFGLA) is closed under union, concatenation, and lookahead.*

For Kleene star, we first show a restricted version.

Proposition 3.3.3. *If $R \subseteq \Sigma^+ \times \Sigma^*$ is a l-language of CFGLA, then R^* is a l-language of CFGLA.*

Proof. Let G_1 be (V, Σ, P_1, X) where $\mathcal{B}(G_1) = R$ and G_2 be $(V \cup \{Y\}, \Sigma, P_1 \cup P_2, Y)$ where $Y \notin V$ and $P_2(Y) = XY|\varepsilon$. Let R_n be $B_\emptyset(P_1^n(X))$ and S_n be $\bigcup_{0 \le k \le n} R_n \cdots R_{n-k}$. Since $\mathcal{B}(G_2) = \lim_{n\to\infty} S_n \cup (\{\varepsilon\} \times \Sigma^*)$, it is sufficient that we show $\lim_{n\to\infty} S_n = R^+$. First, for any k, $R^{k+1} = \lim_{n\to\infty} (R_n \cdots R_{n-k}) = \underline{\lim}_{n\to\infty} (R_n \cdots R_{n-k})$ by Lemma 3.2.1. Therefore $R^{k+1} \subseteq \underline{\lim}_{n\to\infty} S_n$, thus $R^+ \subseteq \underline{\lim}_{n\to\infty} S_n$. Second, if $r \in \overline{\lim}_{n\to\infty} S_n$, then there exists infinitely many n_i and k_i such that $r \in R_{n_i} \cdots R_{n_i - k_i}$. There exists $r_{i0}, \ldots, r_{ik_i}$

such that $r = r_{i0} \cdots r_{ik_i}$, $r_{i0} \in R_{n_i}, \ldots, r_{ik_i} \in R_{n_i - k_i}$. By assumption, for $x \in \Sigma^*$, there are only finitely many n such that $(\varepsilon, x) \in R_n$. Therefore, the set $\{k_i \mid i \geq 1\}$ is finite. Hence, there exists $k \geq 0$ and infinitely many n'_i such that $r \in R_{n'_i} \cdots R_{n'_i - k}$. That is $r \in \overline{\lim_{n\to\infty}}(R_n \cdots R_{n-k}) = R^{k+1}$. Thus, $\overline{\lim_{n\to\infty}} S_n \subseteq R^+$. □

Proposition 3.3.4. *If R is a l-language of CFGLA, then $R \setminus (\{\varepsilon\} \times \Sigma^*)$ is a l-language of CFGLA.*

Proof. Let V' be a copy of V. A function $f : \mathrm{RELA}(\Sigma, V) \to \mathrm{RELA}(\Sigma, V')$ is defined as follows.

$$f(\emptyset) = \emptyset,\ f(\varepsilon) = \emptyset,\ f(a) = a,\ f(X) = X'$$
$$f(e_1|e_2) = f(e_1)|f(e_2),\ f(e_1e_2) = f(e_1)e_2|e_1f(e_2),\ f(!e) = \emptyset$$

We have $B_\emptyset(f(e)) = B_\emptyset(e) \setminus (\{\varepsilon\} \times \Sigma^*)$ by induction on $e \in \mathrm{RELA}(\Sigma, V)$. For a given valid CFGLA $G_1 = (V, \Sigma, P, S)$, we consider $G_2 = (V \cup V', \Sigma, P_2, S')$ where $P_2(X) = P(X)$ for any $X \in V$ and $P_2(X') = f(P(X))$ for any $X' \in V'$. For any $e \in \mathrm{RELA}(\Sigma, V)$, we have $P_2(f(e)) = f(P(e))$, hence $P_2^n(f(e)) = f(P^n(e))$. Thus, $B_\emptyset(P_2^n(S')) = B_\emptyset(P_2^n(f(S))) = B_\emptyset(f(P^n(S))) = B_\emptyset(P^n(S)) \setminus (\{\varepsilon\} \times \Sigma^*)$. Therefore, G_2 is valid and $\mathcal{B}(G_2) = \mathcal{B}(G_1) \setminus (\{\varepsilon\} \times \Sigma^*)$. □

By two propositions, we obtain the following result.

Corollary 3.3.5. *$\mathcal{B}(CFGLA)$ is closed under Kleene star.*

Proof. Let R be a l-language of CFGLA. From Proposition 3.3.4, $R \setminus (\{\varepsilon\} \times \Sigma^*)$ is a l-language of CFGLA. From Proposition 3.3.3, $(R \setminus (\{\varepsilon\} \times \Sigma^*))^* = R^*$ is a l-language of CFGLA. □

It is open whether $\mathcal{B}(\mathrm{CFGLA})$ is closed under intersection, complement.

Next, we show closure properties of $\mathcal{L}(\mathrm{CFGLA})$.

Proposition 3.3.6. *$\mathcal{L}(CFGLA)$ is closed under union, intersection, and complement.*

Proof. For two variables X and Y, $B(\&(X\$)Y\$) = (L_B(X) \cap L_B(Y)) \times \{\varepsilon\}$. If $B(Y) = \Sigma^* \times \Sigma^*$, then $B(!(X\$)Y\$) = (\Sigma^* \setminus L_B(X)) \times \{\varepsilon\}$. Thus, the statement holds by Lemma 3.3.1. □

Hence, CFGLA can represent $\{a^n b^n c^n \mid n \geq 0\}$ and $\{xx \mid x \in \Sigma^*\}$. It is open whether $\mathcal{L}(\mathrm{CFGLA})$ is closed under concatenation and Kleene star. However, a weak version holds.

Lemma 3.3.7. *Let $L \subseteq \Sigma^*$ be a language of CFGLA and $\# \notin \Sigma$ be a letter. $L\# \times (\Sigma \cup \{\#\})^*$ is a l-language of CFGLA.*

Proof. Let G_1 be (V, Σ, P_1, X) where $\mathcal{L}(G_1) = L$. Let $G_2 = (V \cup \{Y\}, \Sigma \cup \{\#\}, P_2, Y)$ where $P_1 \subseteq P_2$ and $P_2(Y) = X\#$. We define a predicate $p(R)$ as "$\forall x, y \in \Sigma^*, w \in (\Sigma \cup \{\#\})^*.\ (x, y) \in R \iff (x, y\#w) \in R$". If $p(B_{P_2,nat}(X))$ holds, then $\mathcal{B}(G_2) = L\# \times (\Sigma \cup \{\#\})^*$. For $e \in \mathrm{RELA}(\Sigma, V)$, $p(B_\emptyset(e))$ holds by induction on e. Since $P_2^n(X) \in \mathrm{RELA}(\Sigma, V)$, $p(B_{P_2,nat}(X))$ holds. □

Corollary 3.3.8. *Let $L_1, L_2 \subseteq \Sigma^*$ be languages of CFGLA and $\# \notin \Sigma$ be a letter. $L_1\#L_2$ and $(L_1\#)^*$ are languages of CFGLA.*

Lastly, it is undecidable whether the intersection of two context-free languages is empty. For two context-free languages L_1 and L_2, $(L_1 \cap L_2) \times \{\varepsilon\}$ is a l-language of CFGLA. Therefore, the following holds.

Proposition 3.3.9. *For a valid CFGLA G, emptiness $\mathcal{B}(G) = \emptyset$ and $\mathcal{L}(G) = \emptyset$ are also undecidable.*

For a given valid CFGLA $G_1 = (V, \Sigma, P_1, X)$, we consider a CFGLA $G_2 = (V \cup \{Y\}, \Sigma, P_1 \cup P_2, Y)$ with $P_2(Y) = \&(X\$)!Y$. The CFGLA G_2 is valid if and only if $\mathcal{L}(G_1) = \emptyset$. Therefore, validity is undecidable.

3.4 Related Grammars

3.4.1 Parsing Expression Grammars

The limit semantics can be considered denotational semantics. We introduce operational semantics of CFGLA and show that two semantics are consistent for a limited range of valid CFGLA. The results show that PEG can be regarded as a subclass of CFGLA.

For an expression $e \in \mathrm{RELA}(\Sigma, V)$, we define the derivation $(e, x) \Rightarrow O$ where x is an input string and O is a set of the rest of the input string.

$$\frac{}{(\emptyset, x) \Rightarrow \emptyset} \qquad \frac{}{(\varepsilon, x) \Rightarrow \{x\}} \qquad \frac{}{(a, ax) \Rightarrow \{x\}} \qquad \frac{}{(a, bx) \Rightarrow \emptyset} \qquad \frac{}{(a, \varepsilon) \Rightarrow \emptyset}$$

$$\frac{(e_1, x) \Rightarrow O_1 \quad (e_2, x) \Rightarrow O_2}{(e_1|e_2, xy) \Rightarrow O_1 \cup O_2}$$

$$\frac{(e_1, x) \Rightarrow \{y_1, \ldots, y_n\} \quad (e_2, y_1) \Rightarrow O_1 \quad \ldots \quad (e_2, y_n) \Rightarrow O_n}{(e_1 e_2, x) \Rightarrow O_1 \cup \cdots \cup O_n}$$

$$\frac{(e, x) \Rightarrow \emptyset}{(!e, x) \Rightarrow \{x\}} \qquad \frac{(e, x) \Rightarrow \{y\} \cup O}{(!e, x) \Rightarrow \emptyset} \qquad \frac{(P(X), x) \Rightarrow O}{(X, x) \Rightarrow O}$$

For example, $(a|ab, abc) \Rightarrow \{bc, c\}$.

For $e \in \mathrm{RELA}(\Sigma, V)$, the *operational semantics* $B_{P,op}(e)$ is defined as $\{(x, y) \mid (e, xy) \Rightarrow O,\ y \in O\}$. A CFGLA G is *operationally complete* if, for any string x and any variable X, there exists O such that $(X, x) \Rightarrow O$. We consider that the semantics is given only if G is operationally complete. In general, $(e, x) \Rightarrow O$ if and only if $(P(e), x) \Rightarrow O$, hence $B_{P,op} = B_{P,op} \circ P$. For an operationally complete CFGLA, $B_{P,op}$ is an interpretation and a solution of P.

We show if a CFGLA is operationally complete then the CFGLA is uniquely convergent. The converse does not hold. We write $(e, x) \Rightarrow_0 O$ for a derivation that the rule of variables is not used. We can define $\Rightarrow_0$ inductively as $\Rightarrow$.

Lemma 3.4.1. *If $(e, x) \Rightarrow_0 O$, then $B_{P,op}(e) \cap M_x = B(e) \cap M_x$ for all interpretation B where $M_x = \{(y, z) \mid x = yz\}$.*

Proof. By induction on $\Rightarrow_0$. We show the case of concatenation. First, we have $(e_1, x) \Rightarrow_0 \{y_1, \ldots, y_n\}$, $(e_2, y_1) \Rightarrow_0 O_1$, $\ldots$, $(e_2, y_n) \Rightarrow_0 O_n$. By the induction hypothesis, $B_{P,op}(e_1) \cap M_x = B(e_1) \cap M_x$, $B_{P,op}(e_2) \cap M_{y_1} = B(e_2) \cap M_{y_1}$, $\ldots$, $B_{P,op}(e_2) \cap M_{y_n} = B(e_2) \cap M_{y_n}$. Thus, $B_{P,op}(e_1 e_2) \cap M_x = B(e_1 e_2) \cap M_x$. □

Lemma 3.4.2. *If $(e, x) \Rightarrow O$, then there exists n such that $(P^m(e), x) \Rightarrow_0 O$ for any $m \geq n$.*

Proof. By induction on $\Rightarrow$. We show the case of union. We have $(e_1, x) \Rightarrow_0 O_1$, $(e_2, x) \Rightarrow_0 O_2$, and $O = O_1 \cup O_2$. By the induction hypothesis, there exist n_1 and n_2 such that for all $m_1 \geq n_1, m_2 \geq n_2$, $(P^{m_1}(e_1), x) \Rightarrow_0 O_1$, $(P^{m_2}(e_2), x) \Rightarrow_0 O_2$. Let $n = \max(n_1, n_2)$, for all $m \geq n$, $(P^m(e_1), x) \Rightarrow_0 O_1$, $(P^m(e_2), x) \Rightarrow_0 O_2$. We have $P^m(e_1e_2) = P^m(e_1)P^m(e_2)$. Therefore, $(P^m(e_1e_2), x) \Rightarrow_0 O$. □

Proposition 3.4.3. *An operationally complete CFGLA is uniquely convergent.*

Proof. Let $M(e) = \{(x, y) \mid (e, xy) \Rightarrow_0 O\}$. By Lemma 3.4.2 and operational completeness, we have $\lim_{n\to\infty} M(P^n(X)) = \Sigma^* \times \Sigma^*$.

$$
\begin{aligned}
B_{P,op}(X) &= \lim_{n\to\infty} (B_{P,op}(X) \cap M(P^n(X))) \\
&= \lim_{n\to\infty} (B_{P,op}(P^n(X)) \cap M(P^n(X))) && (B_{P,op} \text{ is a solution}) \\
&= \lim_{n\to\infty} (B(P^n(X)) \cap M(P^n(X))) && \text{(Lemma 3.4.1)} \\
&= \lim_{n\to\infty} B(P^n(X)) && \text{(Lemma 3.2.3)}
\end{aligned}
$$

□

Two semantics coincide for an operationally complete CFGLA, i.e. $B_{P,nat} = B_{P,op}$. The result shows that PEG can be regarded as a subclass of CFGLA.

Definition 3.4.1 (Parsing expression grammars)**.** We define an *ordered choice* e_1/e_2 is an abbreviation for $e_1|!e_1e_2$. An expression $e \in \mathrm{RELA}(\Sigma, V)$ is called a parsing expression if e can be written using only / without using |. A *PEG* $G = (V, \Sigma, P, S)$ is a CFGLA that $P(X)$ is a parsing expression for any X.

The original definition of the language of PEG [16] is the set of successful input strings and for the derivation $(e, x) \Rightarrow o$, o is a consumed string or fail. We can easily show the language of PEG coincides with our definition.

PEG can represent all deterministic context-free languages and $\mathcal{L}$(PEG) is closed under union, intersection, and complement [16]. For PEG's derivation $(e, x) \Rightarrow O$, there exists y such that $O = \{y\}$ or $O = \emptyset$. That is if $(x_1, y_1), (x_2, y_2) \in B_{P,op}(e)$ and $x_1y_1 = x_2y_2$, then $y_1 = y_2$. Hence, the class $\mathcal{B}$(CFGLA) is a proper superset of $\mathcal{B}$(PEG).

Example 3.4.1.

1. A CFGLA $X = Xa \mid \varepsilon$ is valid and uniquely convergent but is not operationally complete.
2. A PEG $X = Xa/\varepsilon$ is equivalent to $X = Xa \mid !(Xa)$ and it is invalid. The reason is the same as Example 2.2.
3. A PEG $X = aXa \mid !(aXa)$ is uniquely convergent. $B(X) = \{(a^{2n}, a^m) \mid n + m = 2^k - 1\}$ is the solution and $\mathcal{L}(G) = \{a^{2^n-2} \mid n \geq 1\}$.

3.4.2 Context-Free Grammars with Right Contexts

We describe the relationship between CFGLA and CFGRC [5]. A grammar with the right contexts (CFGRC) is a quadruple $G = (\Sigma, V, P, S)$, where P is a finite set of rules. Each rule has the form

$X \to \alpha_1 \cap \cdots \cap \alpha_k \cap \rhd\beta_1 \cap \cdots \cap \rhd\beta_m \cap \trianglerighteq\gamma_1 \cap \cdots \cap \trianglerighteq\gamma_n$ with $X \in V,\ k \geq 1,\ m, n \geq 0,\ \alpha_i, \beta_i, \gamma_i \in (\Sigma \cup V)^*$. The semantics of $\rhd$ and $\trianglerighteq$ are defined by $\rhd R = \{(x, y) \mid (y, \varepsilon) \in R, x \in \Sigma^*\}$ and $\trianglerighteq R = \{(x, y) \mid (xy, \varepsilon) \in R\}$, respectively. The semantics of concatenation is the same as in this paper. The semantics of a grammar is defined by the least solution of the corresponding system of equations because there are no negative operations. We call CFGRC(1) a grammar with the right contexts that satisfy $k = 1$ for any rule. We call context-free grammars with positive lookahead (CFGPLA) a CFGLA with no negative lookahead but with positive lookahead and the end of a string. The following holds.

$$\begin{aligned}
&R \cap \rhd R'_1 \cdots \cap \rhd R'_m \cap \trianglerighteq R''_1 \cdots \cap \trianglerighteq R''_n \\
&= \{(x, y) \mid (x, y) \in R,\ \forall i.\ (y, \varepsilon) \in R'_i,\ \forall j.\ (xy, \varepsilon) \in R''_j\} \\
&= \&(R''_1\$) \ldots \&(R''_n\$)\, R\, \&(R'_1\$) \ldots \&(R'_m\$)
\end{aligned}$$

Therefore, CFGRC(1) can be converted into CFGPLA directly. Since CFGPLA allows replacement of expressions by variables and $\&R = \&(R(\Sigma^* \times \Sigma^*)\$)$ holds, the expressions in CFGPLA can be converted into the expressions defined by the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid X \mid e|e \mid XY \mid \&(X\$) \quad (a \in \Sigma,\ X, Y \in V)$$

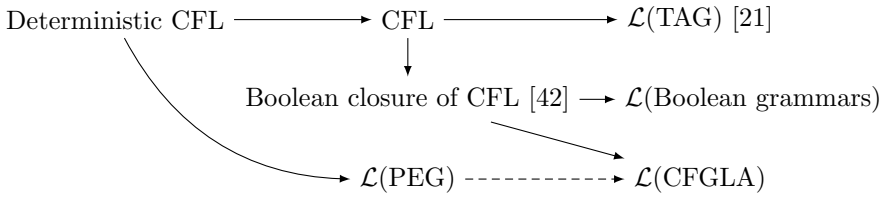
Hence, CFGPLA can be converted into CFGRC(1). The results for CFGRC can be used for CFGPLA. For example, CFGRC has a cubic time recognition algorithm similar to the CYK algorithm; hence, CFGPLA has a cubic time recognition algorithm.

3.4.3 Other Grammars

Finite lookahead is used in the traditional theory of parsing. Regular lookahead is used in various domains of formal language theory, e.g. RELA [27], LL(*) [34], transducers with lookahead [36], and tree transducers with lookahead [13]. A l-language of RELA is a finite union of the form $A \times B$ where A and B are regular languages, hence it is expected that a l-language of CFG with regular lookahead is a finite union of the form $A \times B$ where A is a context-free language and B is a regular language.

Regular expressions with complement and CFG with complement are called extended regular expressions (ERE) [9, 38] and Boolean grammars [32], respectively. Lookahead differs from complement in the scope of influence and non-consumption of letters. RELA are more suitable for parsing than ERE [27].

The relationships of the language class are summarized below where an arrow means proper inclusion, a dashed arrow means inclusion.



The class $\mathcal{L}$(CFGLA) is a proper superset of Boolean closure of context-free languages [42]. A context-free language over one letter alphabet is regular, hence the Boolean closure of unary context-free languages is also regular. On the other hand, a non-regular unary language $\{a^{2^n} \mid n \geq 1\}$ is represented by CFGLA.

The class $\mathcal{L}$(CFGLA) is not a subset of $\mathcal{L}$(TAG) because of the decidability of emptiness. It is expected that $\mathcal{L}$(CFGLA) is not a superset of $\mathcal{L}$(TAG) because the complexity of recognition of TAG is relatively large. A similar argument applies to some supersets of $\mathcal{L}$(TAG) such as indexed languages [2].

We consider it is difficult to prove of inclusion between $\mathcal{L}$(CFGLA) and $\mathcal{L}$(Boolean grammars). It is because of the difference in the properties between lookahead and complement. For example, the class $\mathcal{L}$(Boolean grammars) is closed under concatenation and Kleene star.

Chapter 4

Least Interval Semantics of Context-Free Grammars with Lookahead

First, we point out problems of the limit semantics. We then introduce the least interval semantics with the least fixed point of intervals.

4.1 Problems of Limit Semantics

The limit semantics does not allow for substitution and replacement. That is, the limit semantics does not always coincide for two grammars even if the sets of solutions coincide. It is also not allowed to use the results of computation on a subgrammar, where $G' = (V', \Sigma, P', S')$ is called subgrammar of $G = (V, \Sigma, P, S)$ if $V' \subseteq V$ and P' is the restriction of P to V'.

Example 4.1.1. We consider the following two grammars.

- $G_1 : X = !Y \mid X,\ Y = \varepsilon$.
  The language is computed as follows.

$$\begin{aligned}
&[\![P]\!]^0(B_\emptyset)(X) = \emptyset \\
&[\![P]\!]^1(B_\emptyset)(X) = \hat{B}_\emptyset(!Y \mid X) = !\emptyset \cup \emptyset = \{\varepsilon\} \times \Sigma^* \\
&[\![P]\!]^2(B_\emptyset)(X) = !(\{\varepsilon\} \times \Sigma^*) \cup (\{\varepsilon\} \times \Sigma^*) = \{\varepsilon\} \times \Sigma^*
\end{aligned}$$

  Similarly, $[\![P]\!]^n(B_\emptyset)(X) = \{\varepsilon\} \times \Sigma^*$ holds for $n \geq 3$. Therefore, $\mathcal{L}(G_1) = \{\varepsilon\}$ holds.
- $G_2 : X = !\varepsilon \mid X,\ Y = \varepsilon$.
  First, $[\![P]\!](B_\emptyset)(X) = B_\emptyset(!\varepsilon \mid X) = !(\{\varepsilon\} \times \Sigma^*) \cup \emptyset = \emptyset$. Similarly, $[\![P]\!]^n(B_\emptyset) = \emptyset$ holds for $n \geq 2$. Therefore, $\mathcal{L}(G_2) = \emptyset$ holds.

G_2 is obtained from G_1 by substituting ε for Y. Conversely, G_1 is obtained from G_2 by replacing ε with Y. Since the languages of G_1 and G_2 are different, substitution and replacement do not preserve the limit semantics. "$Y = \varepsilon$" is subgrammar of G_1, and $B(Y) = \{\varepsilon\} \times \Sigma^*$ is a unique solution. Then, we consider using the results of computation on subgrammar. That is, let $B(X) = \emptyset$ and $B(Y) = \{\varepsilon\} \times \Sigma^*$. We consider $\lim_{n\to\infty} [\![P]\!]^n(B)$. Then, $\lim_{n\to\infty} [\![P]\!]^n(B)(X) = \emptyset$, and this is different from the result of G_1. Hence, it is not allowed to use the results of computation on subgrammar in the limit semantics.

Example 4.1.2. We consider the following two grammars.

- $G_3 : X = !(!X),\ Y = !X$
  $[\![P]\!]^n(B_\emptyset)(X) = \emptyset$ holds for any n. Therefore, $\mathcal{L}(G_3) = \emptyset$ holds.
- $G_4 : X = !Y,\ Y = !X$
  $[\![P]\!]^{2n}(B_\emptyset)(X) = \emptyset$ and $[\![P]\!]^{2n+1}(B_\emptyset)(X) = \{\varepsilon\} \times \Sigma^*$. Therefore, the language of G_4 does not exist.

The two grammars illustrate the same problem. They are obtained from each other by substitution and replacement, but their languages are different.

Example 4.1.3. We consider another type of examples.

- $G_5 : S = XY,\ X = !(aX),\ Y = aY \mid \varepsilon$

$$\begin{aligned}
[\![P]\!]^{2n}(B_\emptyset)(X) &= \{\varepsilon\} \times \{a^{2m} \mid m < n\} \\
[\![P]\!]^{2n+1}(B_\emptyset)(X) &= \{\varepsilon\} \times (\{a\}^* \setminus \{a^{2m+1} \mid m < n\}) \\
\lim_{n\to\infty} [\![P]\!]^n(B_\emptyset)(X) &= \{\varepsilon\} \times \{aa\}^*
\end{aligned}$$

  Therefore, $\mathcal{L}(G_5) = \{aa\}^*$ holds.
- $G_6 : S = XY,\ X = !(aX) \mid X,\ Y = aY \mid \varepsilon$
  $[\![P]\!]^0(B_\emptyset)(X) = \emptyset$ holds. $[\![P]\!]^n(B_\emptyset)(X) = \{\varepsilon\} \times \{a\}^*$ holds for $n \geq 1$. Therefore, $\mathcal{L}(G_6) = \{a\}^*$.

Since G_5 is a unique solution grammar, there is no other solution of G_5. G_6 has some solutions, but the language of G_6 is bigger than the language of G_5 in the limit semantics. This is caused by an l-language being overestimated in the middle of the computation and then fixed. The variable X is computed as the empty set first; hence, aX is minimized and $!(aX)$ is maximized. This overestimated value $\{\varepsilon\} \times \{a\}^*$ is fixed by $|X$. This is also true for G_1. Considering that CFG is characterized by the least solution, the computation should be done so that $!(aX)$ is minimized.

We consider computing lower and upper bounds simultaneously, which is the idea of the new semantics. For unknown l-languages X and Y, consider a situation where the lower bounds R_X and R_Y and the upper bounds U_X and U_Y are available. If $R_X \subseteq X \subseteq U_X,\ R_Y \subseteq Y \subseteq U_Y$, then $R_X \cdot R_Y \subseteq X \cdot Y \subseteq U_X \cdot U_Y$ holds; hence, the lower and upper bounds of concatenation $X \cdot Y$ can be computed. If $R_X \subseteq X \subseteq U_X$, then $!U_X \subseteq !X \subseteq !R_X$ holds; Hence, the lower and upper bounds of the negative lookahead $!X$ can be computed.

4.2 Least Interval Semantics

An *interval* of l-languages is a pair of l-languages (R, U) that satisfies $R \subseteq U$. We write it as $[R, U]$ and the set of intervals as $\mathrm{Int}(\mathcal{P}(\Sigma^* \times \Sigma^*))$. The *union*, *concatenation*, *positive lookahead*, and *negative lookahead* are defined as follows.

$$\begin{aligned}
[R_1, U_1] \cup [R_2, U_2] &= [R_1 \cup R_2, U_1 \cup U_2] \\
[R_1, U_1] \cdot [R_2, U_2] &= [R_1 \cdot R_2, U_1 \cdot U_2] \\
\&[R, U] &= [\&R, \&U] \\
![R, U] &= [!U, !R]
\end{aligned}$$

Positive lookahead $\&I$ can be expressed by $!(!I)$ for an interval I. For $I = [R, U]$, we define $I_\downarrow$ and $I_\uparrow$ as R and U, respectively.

The *knowledge ordering* is defined by $[R_1, U_1] \sqsubseteq [R_2, U_2] \iff R_1 \subseteq R_2,\ U_2 \subseteq U_1$. The interval $\bot = [\emptyset, \Sigma^* \times \Sigma^*]$ is the least element. It represents that $\emptyset$ is a lower bound and $\Sigma^* \times \Sigma^*$ is an upper bound. This is the weakest information. For l-language R, an interval $[R, R]$ is a maximal element of $\mathrm{Int}(\mathcal{P}(\Sigma^* \times \Sigma^*))$. For a set of intervals A, the greatest lower bound exists, and we write it as $\bigsqcap A$. $\bigsqcap A = [\bigcap_{[R,U]\in A} R,\ \bigcup_{[R,U]\in A} U]$ holds. If A is a chain, then the least upper bound exists, and we write it as $\bigsqcup A$. Therefore, the set of intervals forms CPO. $\bigsqcup A = [\bigcup_{[R,U]\in A} R,\ \bigcap_{[R,U]\in A} U]$ holds.

Proposition 4.2.1 (Continuity of operations)**.** *The union, concatenation, and negative lookahead are continuous functions with respect to knowledge ordering. That is, for a chain of pairs of intervals $\{(I_k, I'_k) \mid k \in \Lambda\}$, the following hold.*

$$\begin{aligned}
\textstyle\bigsqcup_k (I_k \cup I'_k) &= (\textstyle\bigsqcup_k I_k) \cup (\bigsqcup_k I'_k),\\
\textstyle\bigsqcup_k (I_k \cdot I'_k) &= (\textstyle\bigsqcup_k I_k) \cdot (\bigsqcup_k I'_k),\\
\textstyle\bigsqcup_k (!I_k) &= !(\textstyle\bigsqcup_k I_k).
\end{aligned}$$

Proof. To show this, it is sufficient to show that for a chain of pairs of l-languages $A = \{(R_i, R'_i) \mid i \in \Lambda\}$, the following hold.

1. $\bigcup_i (R_i \cup R'_i) = (\bigcup_i R_i) \cup (\bigcup_i R'_i)$,
2. $\bigcap_i (R_i \cup R'_i) = (\bigcap_i R_i) \cup (\bigcap_i R'_i)$,
3. $\bigcup_i (R_i \cdot R'_i) = (\bigcup_i R_i) \cdot (\bigcup_i R'_i)$,
4. $\bigcap_i (R_i \cdot R'_i) = (\bigcap_i R_i) \cdot (\bigcap_i R'_i)$,
5. $\bigcap_i (!R_i) = !(\bigcup_i R_i)$,
6. $\bigcup_i (!R_i) = !(\bigcap_i R_i)$

Each statement can be shown as follows.

1. $r \in \bigcup_i (R_i \cup R'_i) \iff \exists i.\ (r \in R_i \text{ or } r \in R'_i)$
   $\iff (\exists i.\ r \in R_i) \text{ or } (\exists i.\ r \in R'_i)$
   $\iff r \in (\bigcup_i R_i) \cup (\bigcup_i R'_i)$.
2. $r \in (\bigcap_i R_i) \cup (\bigcap_i R'_i) \iff (\forall i.\ r \in R_i) \text{ or } (\forall i.\ r \in R'_i)$
   $\implies \forall i.\ (r \in R_i \text{ or } r \in R'_i) \iff r \in \bigcap_i (R_i \cup R'_i)$. Conversely, if $r \notin (\bigcap_i R_i) \cup (\bigcap_i R'_i)$, then there exist i and j such that $r \notin R_i$ and $r \notin R'_j$. We assume that $r \in R'_i$ and $r \in R_j$ hold, then (R_i, R'_i) and (R_j, R'_j) are not comparable, which contradicts that A is a chain. Therefore, $r \notin R'_i$ or $r \notin R_j$. We obtain $r \notin R_i \cup R'_i$ or $r \notin R_j \cup R'_j$. Thus, $r \notin (\bigcap_i R_i) \cup (\bigcap_i R'_i) \implies r \notin \bigcap_i (R_i \cup R'_i)$.
3. $r \in \bigcup_i (R_i \cdot R'_i) \iff \exists i, x, y, z.\ r = (xy, z),\ (x, yz) \in R_i,\ (y, z) \in R'_i \implies \exists x, y, z.\ r = (xy, z),\ \exists i.\ (x, yz) \in R_i,\ \exists j.\ (y, z) \in R'_j \iff r \in (\bigcup_i R_i) \cdot (\bigcup_i R'_i)$.
   If $(x, yz) \in R_i$ and $(y, z) \in R'_j$, then $(x, yz) \in R_j$ or $(y, z) \in R'_i$ holds because A is a chain. Therefore, the reverse direction is also true.
4. $r \in (\bigcap_i R_i) \cdot (\bigcap_i R'_i) \iff \exists x, y, z.\ r = (xy, z),\ \forall i.\ (x, yz) \in R_i,\ (y, z) \in R'_i \implies \forall i.\ \exists x, y, z.\ r = (xy, z),\ (x, yz) \in R_i,\ (y, z) \in R'_i \iff r \in \bigcap_i (R_i \cdot R'_i)$.

Conversely, if $r \in \bigcap_i (R_i \cdot R'_i)$, let $C_i = \{(x,y,z) \mid r = (xy,z),\ (x,yz) \in R_i,\ (y,z) \in R'_i\}$, then $C_i \neq \emptyset$ for any i. Since $\{(x,y,z) \mid r = (xy,z)\}$ is a finite set and A is a chain, $\{C_i \mid i \in \Lambda\}$ is a finite chain and $\bigcap_i C_i \neq \emptyset$. Therefore, there exists $(x,y,z) \in \bigcap_i C_i$ such that $(x,yz) \in R_i,\ (y,z) \in R'_i$ for any i. Thus, $r \in (\bigcap_i R_i) \cdot (\bigcap_i R'_i)$.

5. Both sides are subsets of $\{\varepsilon\} \times \Sigma^*$.
   $(\varepsilon, x) \in \bigcap_i (!R_i) \iff \forall i.\ (\varepsilon, x) \in !R_i \iff \forall i, x_1, x_2.\ (x = x_1x_2 \implies (x_1, x_2) \notin R_i) \iff \forall x_1, x_2.\ (x = x_1x_2 \implies \forall i.\ (x_1, x_2) \notin R_i) \iff \forall x_1, x_2.\ (x = x_1x_2 \implies (x_1, x_2) \notin \bigcup_i R_i) \iff (\varepsilon, x) \in !(\bigcup_i R_i)$.
6. Both sides are subsets of $\{\varepsilon\} \times \Sigma^*$.
   $(\varepsilon, x) \in \bigcup_i (!R_i)$
   $\iff \exists i.\ \forall x_1, x_2.\ (x = x_1x_2 \implies (x_1, x_2) \notin R_i)$
   $\implies \forall x_1, x_2.\ (x = x_1x_2 \implies \exists i.\ (x_1, x_2) \notin R_i)$
   $\iff (\varepsilon, x) \in !(\bigcap_i R_i)$.
   Conversely, if $(\varepsilon, x) \in !(\bigcap_i R_i)$, then for any x_1, x_2 $(x = x_1x_2)$, there exists i such that $(x_1, x_2) \notin R_i$. Since $\{(x_1, x_2) \mid x = x_1x_2\}$ is a finite set and A is a chain, there exists i such that $(x_1, x_2) \notin R_i$ for any x_1, x_2 $(x = x_1x_2)$. Thus, $(\varepsilon, x) \in \bigcup_i (!R_i)$.

□

An interval interpretation is a function $B : V \to \mathrm{Int}(\mathcal{P}(\Sigma^* \times \Sigma^*))$. For an interval interpretation B, the extended interpretation $\hat{B} : \mathrm{RELA}(\Sigma, V) \to \mathrm{Int}(\mathcal{P}(\Sigma^* \times \Sigma^*))$ is defined by $\hat{B}(\emptyset) = [\emptyset, \emptyset]$, $\hat{B}(\varepsilon) = [\{\varepsilon\} \times \Sigma^*, \{\varepsilon\} \times \Sigma^*]$, $\hat{B}(a) = [\{a\} \times \Sigma^*, \{a\} \times \Sigma^*]$, $\hat{B}(X) = B(X)$, $\hat{B}(e_1|e_2) = \hat{B}(e_1) \cup \hat{B}(e_2)$, $\hat{B}(e_1e_2) = \hat{B}(e_1) \cdot \hat{B}(e_2)$ and $\hat{B}(!e) = !\hat{B}(e)$. The *bottom interpretation* $B_\perp$ is defined by $B_\perp(X) = \perp$ for all $X \in V$. For a production function P, an interval interpretation B is called a *solution* of P if B satisfies $B = \hat{B} \circ P$. We define a function $[\![P]\!]$ by $[\![P]\!](B) = \hat{B} \circ P$. Then, $[\![P]\!]$ is a continuous function. Therefore, the *least solution* is defined by $B_{\mathrm{least}} = \bigsqcup_{n \geq 0} [\![P]\!]^n(B_\perp)$. This new semantics is called least interval semantics. The *language* of a CFGLA $G = (V, \Sigma, P, S)$ under the least interval semantics is the set $\mathcal{L}_{int}(G) = \{x \mid (x, \varepsilon) \in B_{\mathrm{least}}(S)_\downarrow\}$.

Example 4.2.1.

- $G_7 : X = aXb \mid \varepsilon \mid X$

$$[\![P]\!](B_\perp)(X) = [\{\varepsilon\} \times \Sigma^*,\ \Sigma^* \times \Sigma^*]$$
$$[\![P]\!]^2(B_\perp)(X) = [\{\varepsilon, ab\} \times \Sigma^*,\ \Sigma^* \times \Sigma^*]$$
$$[\![P]\!]^n(B_\perp)(X) = [\{a^m b^m \mid m < n\} \times \Sigma^*,\ \Sigma^* \times \Sigma^*]$$

  Therefore, $\mathcal{L}_{int}(G_7) = \{a^m b^m \mid m \geq 0\}$.
- $G_1 : X = !Y \mid X,\ Y = \varepsilon$
  $[\![P]\!](B_\perp)(X) = [\emptyset,\ \Sigma^* \times \Sigma^*] = \perp$. $[\![P]\!]^n(B_\perp)(X) = \perp$ for any n. Therefore, $\mathcal{L}_{int}(G_1) = \emptyset$.
- $G_6 : S = XY,\ X = !(aX) \mid X,\ Y = aY \mid \varepsilon$
  $[\![P]\!](B_\perp)(X) = [\{\varepsilon\} \times \{\varepsilon\},\ \Sigma^* \times \Sigma^*]$. $[\![P]\!]^n(B_\perp)(X) = [\{\varepsilon\} \times \{\varepsilon\},\ \Sigma^* \times \Sigma^*]$ for $n \geq 1$. Therefore, $\mathcal{L}_{int}(G_6) = \{\varepsilon\}$.

For two grammars, if the sets of solutions coincide, the least solutions coincide. Therefore, substitution and replacement can be done freely. Formally, we define a context C by the following syntax.

$$C ::= \Box \mid C|e \mid e|C \mid Ce \mid eC \mid !C \quad (e \in \mathrm{RELA}(\Sigma, V))$$

A context C can be regarded as a function $C : \mathrm{RELA}(\Sigma, V) \to \mathrm{RELA}(\Sigma, V)$ by $\Box(e) = e$, $(C|e')(e) = C(e)|e'$, $(e'|C)(e) = e'|C(e)$, $(Ce')(e) = C(e)e'$, $(e'C)(e) = e'C(e)$ and $(!C)(e) = !C(e)$. For a context C, if $B(e_1) = B(e_2)$, then $B(C(e_1)) = B(C(e_2))$ holds by induction on C.

Proposition 4.2.2. *(Substitution and replacement) For two grammars $G_1 = (V, \Sigma, P_1, S)$ and $G_2 = (V, \Sigma, P_2, S)$, if there exist $X, Y \in V$ $(X \neq Y)$, P, and context C such that $P_1|_{V\setminus\{X\}} = P_2|_{V\setminus\{X\}} = P$, $P_1(X) = C(Y)$, and $P_2(X) = C(P(Y))$, then $B_{P_1,least} = B_{P_2,least}$.*

Proof. Let B be a solution of P_1, then $B(Y) = B(P(Y))$ holds. Hence, $B(C(Y)) = B(C(P(Y)))$ holds, and we obtain $B(X) = B(P_2(X))$. Thus, B is a solution of P_2. Conversely, let B be a solution of P_2, then $B(Y) = B(P(Y))$ holds. Hence, $B(C(Y)) = B(C(P(Y)))$ holds, and we obtain $B(X) = B(P_1(X))$. Thus, B is a solution of P_1. Since the sets of solutions of P_1 and P_2 coincide, the least solutions coincide. □

Since a replacement is possible, a binary normal form is obtained. For an alphabet Σ and a set of variables V, $\mathrm{RELA}^{\mathrm{bin}}(\Sigma, V)$ is the set of expressions given by the following syntax.

$$e ::= \emptyset \mid \varepsilon \mid a \mid X \mid e|e \mid XY \mid !X \quad (a \in \Sigma,\ X, Y \in V)$$

A *binary normal form* grammar is a CFGLA $G = (V, \Sigma, P, S)$ such that $P(X) \in \mathrm{RELA}^{\mathrm{bin}}(\Sigma, V)$ for any $X \in V$. Any CFGLA can be converted into a binary normal form by repeatedly applying replacement. Then, the order of the grammar size does not change.

In the least interval semantics, it is allowed to use the results of computation on a subgrammar. For CFGLA $G = (V, \Sigma, P, S)$ and its subgrammar $G' = (V', \Sigma, P', S')$, let B_{least} and B'_{least} be the least solutions of P and P', respectively. B'_{least} can be extended to a function from V by $B'_{least}(X) = \bot$ for $X \in V \setminus V'$. We consider $\lim_{n\to\infty} [\![P]\!]^n(B'_{least})$. Since $B_\bot \sqsubseteq B'_{least} \sqsubseteq B_{least}$, we have $B_{least} = \lim_{n\to\infty} [\![P]\!]^n(B_\bot) \sqsubseteq \lim_{n\to\infty} [\![P]\!]^n(B'_{least}) \sqsubseteq B_{least}$. Therefore, $\lim_{n\to\infty} [\![P]\!]^n(B'_{least}) = B_{least}$ holds. This property is used to reduce space complexity in the next section.

If CFGLA has no negative lookahead, then lower bounds are calculated using only lower bounds. Therefore, for CFGPLA G and a variable X, $B_{least}(X) = [B_{nat}(X), U]$ holds for some l-language U. If G is CFG, then $B_{least}(X) = [L_{CFG}(X) \times \Sigma^*,\ L \times \Sigma^*]$ holds for some language L. Hence, it is proven that CFGLA is an extension of CFG and CFGRC(1).

Closure properties are preserved except for complement. That is, the language class of CFGLA is closed under union, intersection, marked concatenation, and marked Kleene star. This is the same reason as presented in the previous chapter. These operations only use lower bounds to calculate lower bounds. The constructions of grammars to prove closure properties can be used as in the previous chapter. For complement, if G has a unique solution in the set of intervals, then $\mathcal{L}_{int}(G)^c$ is the language of CFGLA. CFG can be converted into Greibach normal form, which has a unique solution; hence, complement of a context-free language can be expressed by CFGLA.

4.3 Relationship to Parsing Expression Grammars

As mentioned, PEG is characterized by derivation $(e, x) \Rightarrow O$, where e is an expression, x is an input string, and O is a set of the remaining input strings. Then, we define $B_{op}(e)$ as $[R, U]$, where $R = \{(x, y) \mid (e, xy) \Rightarrow O, y \in O\}$ and $U = R \cup \{(x, y) \mid \forall O.\ (e, xy) \not\Rightarrow O\} = \{(x, y) \mid (e, xy) \Rightarrow O, y \notin O\}^c$. The set U is a collection of all pairs except those explicitly rejected.

Theorem 4.3.1. *For $e \in RELA(\Sigma, V)$, $B_{op}(e) \sqsubseteq \hat{B}_{least}(e)$ holds.*

Proof. If the derivation $(e, x) \Rightarrow O$ does not contain the variable rule, we write $(e, x) \Rightarrow_0 O$, where the variable rule is the rule that derives $(X, x) \Rightarrow O$ from $(P(X), x) \Rightarrow O$. We define $B_{op,0}(e) = [\{(x, y) \mid (e, xy) \Rightarrow_0 O,\ y \in O\},\ \{(x, y) \mid (e, xy) \Rightarrow_0 O,\ y \notin O\}^c]$. We have $B_{op} = \lim_{n\to\infty} B_{op,0} \circ P^n$. Since $B_{least} = \lim_{n\to\infty} \hat{B}_\perp \circ P^n$, if we show that $B_{op,0} \sqsubseteq \hat{B}_\perp$, then $B_{op} \sqsubseteq \hat{B}_{least}$ holds.

The statement $B_{op,0}(e) \sqsubseteq \hat{B}_\perp(e)$ is equivalent to $\{(x, y) \mid (e, xy) \Rightarrow_0 O,\ y \in O\} \subseteq \hat{B}_\perp(e)_\downarrow$ and $\hat{B}_\perp(e)_\uparrow \subseteq \{(x, y) \mid (e, xy) \Rightarrow_0 O,\ y \notin O\}^c$. Therefore, we show by induction on e that for $(e, xy) \Rightarrow_0 O$, if $y \in O$, then $(x, y) \in \hat{B}_\perp(e)_\downarrow$, and if $y \notin O$, then $(x, y) \notin \hat{B}_\perp(e)_\uparrow$.

- We omit the cases of $\emptyset$, ε, and X.
- When $(a, xy) \Rightarrow_0 O$ and $y \in O$, $x = a$ holds; hence, $(x, y) \in \hat{B}_\perp(a)_\downarrow$.
- When $(a, xy) \Rightarrow_0 O$ and $y \notin O$, $x \neq a$ holds; hence, $(x, y) \notin \hat{B}_\perp(a)_\uparrow$.
- When $(e_1|e_2, xy) \Rightarrow_0 O$ and $y \in O$, $(e_1, xy) \Rightarrow_0 O_1$, $(e_2, xy) \Rightarrow_0 O_2$, and $O = O_1 \cup O_2$ hold. Since $y \in O_1$ or $y \in O_2$ holds, $(x, y) \in \hat{B}_\perp(e_1)_\downarrow$ or $(x, y) \in \hat{B}_\perp(e_2)_\downarrow$ holds. Thus, $(x, y) \in \hat{B}_\perp(e_1)_\downarrow \cup \hat{B}_\perp(e_2)_\downarrow = \hat{B}_\perp(e_1|e_2)_\downarrow$.
- When $(e_1|e_2, xy) \Rightarrow_0 O$ and $y \notin O$, $(e_1, xy) \Rightarrow_0 O_1$, $(e_2, xy) \Rightarrow_0 O_2$, and $O = O_1 \cup O_2$ hold. Since $y \notin O_1$ and $y \notin O_2$, $(x, y) \notin \hat{B}_\perp(e_1)_\uparrow$ and $(x, y) \notin \hat{B}_\perp(e_2)_\uparrow$ hold. Thus, $(x, y) \notin \hat{B}_\perp(e_1)_\uparrow \cup \hat{B}_\perp(e_2)_\uparrow = \hat{B}_\perp(e_1|e_2)_\uparrow$.
- When $(e_1e_2, xy) \Rightarrow_0 O$ and $y \in O$, $(e_1, xy) \Rightarrow_0 \{z_1, \ldots, z_n\}$, $(e_2, z_1) \Rightarrow_0 O_1$, $\ldots$, $(e_2, z_n) \Rightarrow_0 O_n$, and $O = O_1 \cup \cdots \cup O_n$ hold. There exists i such that $y \in O_i$, and there exist x_1, x_2 such that $xy = x_1z_i,\ z_i = x_2y$. Therefore, $(x_1, x_2y) \in \hat{B}_\perp(e_1)_\downarrow$, and $(x_2, y) \in \hat{B}_\perp(e_2)_\downarrow$ hold. Thus, $(x, y) \in \hat{B}_\perp(e_1e_2)_\downarrow$.
- When $(e_1e_2, xy) \Rightarrow_0 O$ and $y \notin O$, $(e_1, xy) \Rightarrow_0 \{z_1, \ldots, z_n\}$, $(e_2, z_1) \Rightarrow_0 O_1$, $\ldots$, $(e_2, z_n) \Rightarrow_0 O_n$ and $O = O_1 \cup \cdots \cup O_n$ hold. For any i, $y \notin O_i$. Therefore, for any x_1, x_2 $(x = x_1x_2)$, if $x_2y \notin \{z_1, \ldots, z_n\}$, then $(x_1, x_2y) \notin \hat{B}_\perp(e_1)_\uparrow$ holds, and if $x_2y \in \{z_1, \ldots, z_n\}$, then $(x_2, y) \notin \hat{B}_\perp(e_2)_\uparrow$ holds. Thus, $(x, y) \notin \hat{B}_\perp(e_1e_2)_\uparrow$.
- When $(!e, xy) \Rightarrow_0 O$ and $y \in O$, we have $x = \varepsilon$ and $(e, y) \Rightarrow_0 \emptyset$. For any y_1, y_2 $(y = y_1y_2)$, $(y_1, y_2) \notin \hat{B}_\perp(e)_\uparrow$ holds. Thus, $(x, y) \in \hat{B}_\perp(!e)_\downarrow$.
- When $(!e, xy) \Rightarrow_0 O$ and $y \notin O$, $(e, xy) \Rightarrow_0 O'$ and $O' \neq \emptyset$ or $x \neq \varepsilon$ hold. Since $\hat{B}_\perp(!e)_\uparrow \subseteq \{\varepsilon\} \times \Sigma^*$, if $x \neq \varepsilon$, then $(x, y) \notin \hat{B}_\perp(!e)_\uparrow$. Otherwise, $O' \neq \emptyset$ holds; hence, there exist y_1, y_2 $(y = y_1y_2)$ such that $y_2 \in O$ and $(y_1, y_2) \in \hat{B}_\perp(e)_\downarrow$. Thus, $(x, y) \notin \hat{B}_\perp(!e)_\uparrow$.

□

This represents the soundness of the derivation. If PEG is complete, then R and U coincide. Since $[R, R]$ is a maximal element, B_{op} and $\hat{B}_{least}$ coincide for a complete PEG.

Example 4.3.1. In general, B_{op} and $\hat{B}_{least}$ do not coincide.

- $G_8 : X = !Y,\ Y = Ya$
  The derivation falls into a loop at Y by the left recursion. Hence, $B_{op}(X) = [\emptyset, \Sigma^* \times \Sigma^*],\ B_{op}(Y) = [\emptyset, \Sigma^* \times \Sigma^*]$. $B_{least}(X) = [\{\varepsilon\} \times \Sigma^*, \{\varepsilon\} \times \Sigma^*],\ B_{least}(Y) = [\emptyset, \emptyset]$. Therefore, the languages are $\emptyset$ and $\{\varepsilon\}$.

4.4 Related Work

The study of the well-founded semantics for Boolean grammars [23] is closely related to this paper. Boolean grammars [32] are CFGs with intersection and complement operations. There are two differences between our study and the method used in the study by [23]. First, our study uses intervals, whereas their study uses three-valued languages. Second, our study uses Fitting semantics, whereas their study uses well-founded semantics.

A three-valued language [23] is a function from Σ^* to $\{0, \frac{1}{2}, 1\}$. Intuitively, 0, $\frac{1}{2}$, and 1 represent *true*, *undefined*, and *false*, respectively. There is a one-to-one correspondence between three-valued l-languages and intervals of l-languages. From a three-valued l-language $T : \Sigma^* \times \Sigma^* \to \{0, \frac{1}{2}, 1\}$, we obtain an interval $[R, U]$, where $R = \{(x, y) \mid T(x, y) = 1\}$ and $U = \{(x, y) \mid T(x, y) \neq 0\}$. For the other direction, for an interval $[R, U]$, we obtain the corresponding three-valued l-languages T by $T(x, y) = 1$ if $(x, y) \in R$, $T(x, y) = 0$ if $(x, y) \notin U$, and $T(x, y) = \frac{1}{2}$ otherwise. Since there is a one-to-one correspondence, they are not essentially different. The use of intervals makes the definition of operations simple and easy to understand.

Fitting semantics [14] and well-founded semantics [40] are terms used in the area of logic programming [15]. Fitting semantics is the semantics used in this paper. To explain well-founded semantics, we consider $[\![P]\!]_{B_{neg}}$, where B_{neg} is an interpretation. Then, $[\![P]\!]_{B_{neg}}(B_{pos})$ uses B_{pos} for positive ocurrence variables and B_{neg} for negative ocurrence variables. Let $\Omega(B_{neg}) = \bigcup_{n \geq 0} [\![P]\!]^n_{B_{neg}}(B_\emptyset)$ and $B_{wf} = \bigsqcup_{n \geq 0} \Omega^n(B_\perp)$. One of the advantages of well-founded semantics is that the elimination of unit rules preserves the semantics. For example, $X = !(aX) \mid X$ and $X = !(aX)$ have the same B_{wf}. Indeed, $\mathcal{L}_{wf}(G_6) = \{aa\}^*$ under the well-founded semantics. One of the disadvantages of well-founded semantics is that B_{wf} does not always coincide for two grammars even if the sets of solutions coincide. As a result, substitution and replacement may not preserve the semantics. For example, $B_{wf}(Y) = [\{\varepsilon\} \times \Sigma^*, \{\varepsilon\} \times \Sigma^*]$ for G_3 but $B_{wf}(Y) = [\emptyset, \{\varepsilon\} \times \Sigma^*]$ for G_4. This is the reason why we have adopted Fitting semantics. In addition, Fitting semantics is simpler, and Fitting model in a propositional logic program can be computed in linear time, but the computation of a well-founded model takes quadratic time in general [7].

Chapter 5

Derivatives of Context-Free Grammars with Lookahead

We introduce the derivatives of CFGLA and the recognition algorithm by derivatives under limit semantics. We explain that the limit semantics of CFGLA has some problems and that the recognition algorithm does not run in polynomial time. We then show that the recognition algorithm runs in cubic time and quadratic space for the length of the input string under the least interval semantics.

5.1 Derivatives in Limit Semantics

We introduce derivatives of CFGLA and the recognition algorithm based on it.

Lemma 5.1.1. *The left quotients are continuous, i.e., for convergent sequences R_n, sequences $x^{-1}R_n$ and $\widetilde{x}^{-1}R_n$ converge and the following hold.*

$$\lim_{n\to\infty} x^{-1}R_n = x^{-1}\lim_{n\to\infty} R_n$$
$$\lim_{n\to\infty} \widetilde{x}^{-1}R_n = \widetilde{x}^{-1}\lim_{n\to\infty} R_n$$

Proof. For any index set I and $w \in \{x, \widetilde{x}\}$, we have $\bigcup_{i\in I} w^{-1}R_i = w^{-1}\bigcup_{i\in I} R_i$ and $\bigcap_{i\in I} w^{-1}R_i = w^{-1}\bigcap_{i\in I} R_i$. Therefore, the lemma holds. □

We define the derivatives of $e \in \mathrm{RELA}(\Sigma, V)$ by extending the derivatives of RELA. For a set of variables V and $w \in (\Sigma \cup \widetilde{\Sigma})^*$, let $V_w = \{X_w \mid X \in V\}$ be a copy of V if $w \neq \varepsilon$ and $V_\varepsilon = V$. For $W \subseteq (\Sigma \cup \widetilde{\Sigma})^*$, we define $V_W = \{X_w \mid X \in V,\ w \in W\}$. For $e \in \mathrm{RELA}(\Sigma, V)$ and $a \in \Sigma$, the *derivative of matched part* $D_a e$ and the *derivative of remaining part* $D_{\widetilde{a}} e$ are defined by mutual recursion.

$$\begin{aligned}
D_a : \mathrm{RELA}(\Sigma, V) &\to \mathrm{RELA}(\Sigma, V_{\{\varepsilon, a, \widetilde{a}\}}) \\
D_a\emptyset &= \emptyset \\
D_a\varepsilon &= \emptyset \\
D_a b &= \begin{cases} \varepsilon & (a = b) \\ \emptyset & (a \neq b) \end{cases} \\
D_a X &= X_a \\
D_a(e_1|e_2) &= D_a e_1 | D_a e_2 \\
D_a(e_1 e_2) &= (D_a e_1)e_2 | (D_{\widetilde{a}} e_1)(D_a e_2) \\
D_a(!e) &= \emptyset
\end{aligned}$$

$$
\begin{aligned}
D_{\widetilde{a}} : \mathrm{RELA}(\Sigma, V) &\to \mathrm{RELA}(\Sigma, V_{\{\varepsilon,a,\widetilde{a}\}}) \\
D_{\widetilde{a}}\emptyset &= \emptyset \\
D_{\widetilde{a}}\varepsilon &= \varepsilon \\
D_{\widetilde{a}}b &= \emptyset \\
D_{\widetilde{a}}X &= \&X_{\widetilde{a}} \\
D_{\widetilde{a}}(e_1|e_2) &= D_{\widetilde{a}}e_1|D_{\widetilde{a}}e_2 \\
D_{\widetilde{a}}(e_1e_2) &= (D_{\widetilde{a}}e_1)(D_{\widetilde{a}}e_2) \\
D_{\widetilde{a}}(!e) &= !(D_ae|D_{\widetilde{a}}e)
\end{aligned}
$$

We write $\hat{B}$ and $\hat{P}$ simply as B and P, respectively. The derivatives compute the left quotient correctly, i.e., the following hold.

$$
\begin{aligned}
B_\emptyset(D_ae) &= a^{-1}B_\emptyset(e) \\
B_\emptyset(D_{\widetilde{a}}e) &= \widetilde{a}^{-1}B_\emptyset(e)
\end{aligned}
$$

It is proved by induction on e. In the case of variables, both sides are the empty set. The other base cases are easy to prove. The inductive cases directly follow from lemmas in Section 2.4.

For a CFGLA $G = (V, \Sigma, P, S)$ and $a \in \Sigma$, the *derivative* of CFGLA is defined by $D_aG = (V_{\{\varepsilon,a,\widetilde{a}\}}, \Sigma, P', S_a)$, where $P'(X_c) = D_cP(X)$ for all $X_c \in V_{\{\varepsilon,a,\widetilde{a}\}}$. Here, $D_\varepsilon e = e$. From the construction of P', for any $e \in \mathrm{RELA}(\Sigma, V)$, the following hold.

$$
\begin{aligned}
B_\emptyset(P'(D_ae)) &= B_\emptyset(D_aP(e)) \\
B_\emptyset(P'(D_{\widetilde{a}}e)) &= B_\emptyset(D_{\widetilde{a}}P(e))
\end{aligned}
$$

It is proven by induction on e. In the case of variables and $\widetilde{a}$, $B_\emptyset(P'(D_{\widetilde{a}}X)) = B_\emptyset(\&P'(X_{\widetilde{a}})) = B_\emptyset(\&(D_{\widetilde{a}}P(X)) = B_\emptyset(D_{\widetilde{a}}P(X))$ holds. The other cases are easy to prove.

For $x = a_1 \cdots a_n$, we define the derivative D_xG as $D_{a_n}(\cdots(D_{a_1}G))$.

Theorem 5.1.2. *For a valid CFGLA G and $x \in \Sigma^*$, D_xG is valid and $\mathcal{L}(D_xG) = x^{-1}\mathcal{L}(G)$ holds. Hence, the languages of CFGLA are closed under the left quotient by a string.*

Proof. It is sufficient to show $\mathcal{L}(D_aG) = a^{-1}\mathcal{L}(G)$ for any $a \in \Sigma$. For $D_aG = (V_{\{\varepsilon,a,\widetilde{a}\}}, \Sigma, P', S_a)$, we prove that $\lim_{n\to\infty} B_\emptyset(P'^n(X_c)) = c^{-1}B_{P,\mathrm{nat}}(X)$ holds for any $c \in \{\varepsilon, a, \widetilde{a}\}$ and $X \in V$. First, we have $B_\emptyset(P'^{n+1}(X_c)) = B_\emptyset(P'^n(D_cP(X))) = B_\emptyset(D_cP^{n+1}(X)) = c^{-1}B_\emptyset(P^{n+1}(X))$. Therefore, from Lemma 5.1.1, $\lim_{n\to\infty} B_\emptyset(P'^n(X_c)) = \lim_{n\to\infty} c^{-1}B_\emptyset(P^n(X)) = c^{-1}\lim_{n\to\infty} B_\emptyset(P^n(X)) = c^{-1}B_{P,nat}(X)$. Thus, $B_{P',nat}$ exists, D_aG is valid, and $\mathcal{L}(D_aG) = a^{-1}\mathcal{L}(G)$. □

A function $D_\$: \mathrm{RELA}(\Sigma, V) \to \mathrm{RELA}(\emptyset, V)$ is defined by $D_\$\emptyset = \emptyset$, $D_\$\varepsilon = \varepsilon$, $D_\$a = \emptyset$, $D_\$X = X$, $D_\$(e_1|e_2) = D_\$e_1|D_\$e_2$, $D_\$(e_1e_2) = (D_\$e_1)(D_\$e_2)$, and $D_\$(!e) = !(D_\$e)$. For any $e \in \mathrm{RELA}(\Sigma, V)$, $B_\emptyset(D_\$e) = B_\emptyset(e) \cap \{(\varepsilon, \varepsilon)\}$ holds.

For a CFGLA $G = (V, \Sigma, P, S)$, a CFGLA $D_\$G$ is $(V, \emptyset, P', S)$, where $P'(X) = D_\$P(X)$ for all $X \in V$. For a valid CFGLA G, $D_\$G$ is valid. $\varepsilon \in \mathcal{L}(G) \iff \varepsilon \in \mathcal{L}(D_\$G)$ holds. It can be proven in the same way as Theorem 5.1.2. When $\Sigma = \emptyset$, the set of l-languages $(\mathcal{P}(\Sigma^* \times \Sigma^*), \cup, \cdot, !)$ is isomorphic to the set of truth values $(\{0, 1\}, \max, \min, \lambda x.\ 1 - x)$. Therefore, $D_\$G$ can be regarded as a system of equations over truth values.

Theorem 5.1.3. *For a given valid CFGLA G, $\varepsilon \in \mathcal{L}(G)$ is decidable.*

Proof. We compute $D_\$G$. Next, we assign *false* to any variable at the beginning and update the variable assignments by iterations. Since $D_\$G$ is valid, the iterations halt. Finally, we output the assignment of the start variable. □

The number of iterations is at most $2^{|V|}$. Hence, this method can take exponential time for the grammar size. For example, let $(e_1 \text{ xor } e_2)$ be $!e_1e_2|e_1!e_2$. We consider the grammar $S = X_4X_3X_2X_1,\ X_4 = (X_4 \text{ xor } X_3X_2X_1) \mid S,\ X_3 = (X_3 \text{ xor } X_2X_1) \mid S,\ X_2 = (X_2 \text{ xor } X_1) \mid S,\ X_1 = !X_1 \mid S$. A variable X_n oscillates with period 2^n until S becomes true; hence, the number of iterations is 2^4.

Theorem 5.1.4. *For a given valid CFGLA G and $x \in \Sigma^*$, $x \in \mathcal{L}(G)$ is decidable.*

Proof. We compute the derivative D_xG and decide $\varepsilon \in \mathcal{L}(D_xG)$. We have $x \in \mathcal{L}(G) \iff \varepsilon \in \mathcal{L}(D_xG)$; thus, the theorem holds. □

For $e_1, e_2 \in \mathrm{RELA}(\Sigma, V)$, we write $e_1 \simeq e_2$ if $B(e_1) = B(e_2)$ holds for any interpretation B.

Example 5.1.1. $G : X = aXb \mid \varepsilon$, where let aXb be $a(Xb)$.
We show $ab \in \mathcal{L}(G)$ using derivatives.

$$
\begin{aligned}
D_a(aXb \mid \varepsilon) &= \varepsilon(Xb) \mid \emptyset(D_a(Xb)) \mid \emptyset \simeq Xb \\
D_{\tilde{a}}(aXb \mid \varepsilon) &= \emptyset(D_{\tilde{a}}(Xb)) \mid \varepsilon \simeq \varepsilon \\
D_b(aXb \mid \varepsilon) &\simeq \emptyset \\
D_{\tilde{b}}(aXb \mid \varepsilon) &\simeq \varepsilon \\
D_{ab}(aXb \mid \varepsilon) &\simeq D_b(Xb) \simeq X_bb \mid X_{\tilde{b}} \\
D_{a\tilde{b}}(aXb \mid \varepsilon) &\simeq D_{\tilde{b}}(Xb) \simeq \emptyset \\
D_{\tilde{a}b}(aXb \mid \varepsilon) &\simeq D_b(\varepsilon) = \emptyset \\
D_{\tilde{a}\tilde{b}}(aXb \mid \varepsilon) &\simeq D_{\tilde{b}}(\varepsilon) = \varepsilon
\end{aligned}
$$

Therefore, we obtain the following grammar.

$$D_{ab}G : X_{ab} = X_bb \mid X_{\tilde{b}},\ X_b = \emptyset,\ X_{\tilde{b}} = \varepsilon$$

Since $\varepsilon \in \mathcal{L}(D_{ab}G)$, we obtain $ab \in \mathcal{L}(G)$.

The derivatives of PEG can be computed in PEG because the following hold.

$$
\begin{aligned}
D_a(!e) &\simeq \emptyset \\
D_{\tilde{a}}(!e) &\simeq !(D_ae)!(D_{\tilde{a}}e) \\
D_a(e_1/e_2) &\simeq D_ae_1/!(D_{\tilde{a}}e_1)D_ae_2 \\
D_{\tilde{a}}(e_1/e_2) &\simeq D_{\tilde{a}}e_1/!(D_ae_1)D_{\tilde{a}}e_2
\end{aligned}
$$

The derivatives of positive lookahead are defined without using negative lookahead.

$$
\begin{aligned}
D_a(\&e) &\simeq \emptyset \\
D_{\tilde{a}}(\&e) &\simeq \&(D_ae|D_{\tilde{a}}e)
\end{aligned}
$$

In the case of CFG, for $x \neq \varepsilon$ and an empty interpretation $L_\emptyset$, $\hat{L}_\emptyset(D_{\widetilde{x}}e) = \hat{L}_\emptyset(D_{\$}e) \in \{\emptyset,\ \{\varepsilon\}\}$ holds. Hence, for the derivatives of CFG [20], the variable corresponding to $X_{\widetilde{x}}$ is eliminated in the middle of the computation.

For G and input string x, the recognition algorithm determines the truth value of $(x_2, x_3) \in B(X)$ for any strings x_1, x_2, x_3 ($x = x_1x_2x_3$) and variable X. This can be regarded as filling in the table as well as the CYK algorithm.

(ε, abc)	(a, bc)	(ab, c)	(abc, ε)
	(ε, bc)	(b, c)	(bc, ε)
		(ε, c)	(c, ε)
			$(\varepsilon, \varepsilon)$

In the case of CFG, each cell is dependent on the left cells and the lower cells. In CFGLA, cell (ε, x) also depends on a cell (x_1, x_2) because of lookahead, where x_1 and x_2 are strings such that $x = x_1x_2$. Thus, cells in the same row are interdependent and must be determined simultaneously. Hence, it is important that membership of the empty string is decidable in polynomial time with respect to the number of variables. However, the presented recognition algorithm of the empty string takes exponential time with respect to the number of variables. Although we have not proven NP-hardness or other complexity results, we believe that it is difficult to obtain a polynomial-time recognition algorithm in the limit semantics.

5.2 Derivatives in Least Interval Semantics

In this section, we show that the derivatives can be defined in the least interval semantics. We show that the recognition algorithm by derivatives runs in cubic time and quadratic space.

Let $I = [R, U]$ be an interval and x be a string. The *left quotient of matched part* $x^{-1}I$ and the *left quotient of remaining part* $\widetilde{x}^{-1}I$ ($x \neq \varepsilon$) are defined as follows.

$$x^{-1}[R, U] = [x^{-1}R, x^{-1}U]$$
$$\widetilde{x}^{-1}[R, U] = [\widetilde{x}^{-1}R, \widetilde{x}^{-1}U]$$

For a letter $a \in \Sigma$, intervals I and I', the following hold.

$$\begin{aligned}
a^{-1}\bot &= \bot \\
a^{-1}(I \cup I') &= a^{-1}I \cup a^{-1}I' \\
a^{-1}(I \cdot I') &= (a^{-1}I) \cdot I' \cup (\widetilde{a}^{-1}I) \cdot a^{-1}I' \\
a^{-1}(!I) &= [\emptyset, \emptyset] \\
\widetilde{a}^{-1}\bot &= [\emptyset, \{\varepsilon\} \times \Sigma^*] \\
\widetilde{a}^{-1}(I \cup I') &= \widetilde{a}^{-1}I \cup \widetilde{a}^{-1}I'
\end{aligned}$$

$$\widetilde{a}^{-1}(I \cdot I') = (\widetilde{a}^{-1}I) \cdot (\widetilde{a}^{-1}I')$$
$$\widetilde{a}^{-1}(!I) = !(a^{-1}I \cup \widetilde{a}^{-1}I)$$

The left quotient is continuous, i.e., for a chain of intervals $A = \{I_k \mid k \in \Lambda\}$, $\bigsqcup_k x^{-1}I_k = x^{-1}\bigsqcup A$ and $\bigsqcup_k \widetilde{x}^{-1}I_k = \widetilde{x}^{-1}\bigsqcup A$ hold.

The derivative computes the left quotient correctly, i.e., the following holds.

Theorem 5.2.1. *For a CFGLA G and $x \in \Sigma^*$, $\mathcal{L}_{int}(D_xG) = x^{-1}\mathcal{L}_{int}(G)$ holds. Hence, the languages of CFGLA are closed under the left quotient by a string.*

This theorem is proven in the same way as Theorem 5.1.2.

In the case of CFG, pruned derivatives [20], which are derivatives without some unnecessary variables, are introduced to reduce the computational complexity. We follow this idea and introduce pruned derivatives of CFGLA. However, the construction is quite different because the production function of CFGLA is defined as a function to the set of expressions.

We define two binary functions $\oplus$, $\otimes$ on $\mathrm{RELA}(\Sigma, V)$ by the following. Here, e'_1 and e'_2 are expressions that are not the empty set expression.

$$\emptyset \oplus \emptyset = \emptyset,\ e'_1 \oplus \emptyset = e'_1,\ \emptyset \oplus e'_2 = e'_2,\ e'_1 \oplus e'_2 = e'_1|e'_2$$
$$\emptyset \otimes \emptyset = \emptyset,\ e'_1 \otimes \emptyset = \emptyset,\ \emptyset \otimes e'_2 = \emptyset,\ e'_1 \otimes e'_2 = e'_1e'_2$$

We modify the definition of derivatives as follows and call them *pruned derivatives*.

$$D_a : \mathrm{RELA}(\Sigma, V_{(\Sigma\cup\widetilde{\Sigma})^*}) \to \mathrm{RELA}(\Sigma, V_{(\Sigma\cup\widetilde{\Sigma})^*})$$
$$D_aX_w = \begin{cases} X_{wa} & (w \in \Sigma^*) \\ \emptyset & (otherwise) \end{cases}$$
$$D_a(e_1|e_2) = D_ae_1 \oplus D_ae_2$$
$$D_a(e_1e_2) = (D_ae_1) \otimes e_2 \oplus (D_{\widetilde{a}}e_1) \otimes (D_ae_2)$$
$$D_a(!e) = \emptyset$$
$$D_{\widetilde{a}} : \mathrm{RELA}(\Sigma, V_{(\Sigma\cup\widetilde{\Sigma})^*}) \to \mathrm{RELA}(\Sigma, V_{(\Sigma\cup\widetilde{\Sigma})^*})$$
$$D_{\widetilde{a}}X_w = \&X_{w\widetilde{a}}$$
$$D_{\widetilde{a}}(e_1|e_2) = D_{\widetilde{a}}e_1 \oplus D_{\widetilde{a}}e_2$$
$$D_{\widetilde{a}}(e_1e_2) = (D_{\widetilde{a}}e_1) \otimes (D_{\widetilde{a}}e_2)$$
$$D_{\widetilde{a}}(!e) = !(D_ae \oplus D_{\widetilde{a}}e)$$

The base cases except variables are not changed. Expressions $e_1|e_2$ and e_1e_2 change to $e_1 \oplus e_2$ and $e_1 \otimes e_2$, respectively. In addition, $D_aX_{w\widetilde{b}} = \emptyset$ is introduced.

For pruned derivatives, $a \in \Sigma$, e_1 and $e_2 \in \mathrm{RELA}(\Sigma, V)$, the following hold.

$$D_a(e_1 \oplus e_2) = D_ae_1 \oplus D_ae_2$$
$$D_{\widetilde{a}}(e_1 \oplus e_2) = D_{\widetilde{a}}e_1 \oplus D_{\widetilde{a}}e_2$$
$$D_a(e_1 \otimes e_2) = D_ae_1 \otimes e_2 \oplus D_{\widetilde{a}}e_1 \otimes D_ae_2$$
$$D_{\widetilde{a}}(e_1 \otimes e_2) = D_{\widetilde{a}}e_1 \otimes D_{\widetilde{a}}e_2$$

For the case of $D_a(e_1 \otimes e_2)$, if $e_1 = \emptyset$ or $e_2 = \emptyset$, then both sides are $\emptyset$. Otherwise, the equation holds from the definition of pruned derivatives. For $e_{(x_1,x_2)} \in \mathrm{RELA}(\Sigma, V)$, we define $\sum_{x_1x_2=x} e_{(x_1,x_2)}$ as $e_{(\varepsilon,\varepsilon)}$

if $x = \varepsilon$ and $(\sum_{x_1x_2=x'} e_{(ax_1,x_2)}) \oplus e_{(\varepsilon,x)}$ if $x = ax'$. For $c \in \{a, \widetilde{a}\}$, we have $D_c(\sum_{x_1x_2=x} e_{(x_1,x_2)}) = \sum_{x_1x_2=x} D_c\, e_{(x_1,x_2)}$.

For pruned derivatives, $D_b(D_{\widetilde{a}}e) = \emptyset$ holds for $a, b \in \Sigma$ and $e \in \mathrm{RELA}(\Sigma, V)$. It can be shown by induction on e. For the case of X, $D_b(D_{\widetilde{a}}X) = D_b(\&X_{\widetilde{a}}) = \emptyset$. For the case of e_1e_2, $D_b(D_{\widetilde{a}}e_1e_2) = D_b(D_{\widetilde{a}}e_1 \otimes D_{\widetilde{a}}e_2) = D_b(D_{\widetilde{a}}e_1) \otimes (D_{\widetilde{a}}e_2) \oplus D_{\widetilde{ab}}e_1 \otimes D_b(D_{\widetilde{a}}e_2) = \emptyset \otimes (D_{\widetilde{a}}e_2) \oplus D_{\widetilde{ab}}e_1 \otimes \emptyset = \emptyset$.

For $x = a_1 \cdots a_n$, $y = b_1 \cdots b_m$ and $e \in \mathrm{RELA}(\Sigma, V)$, we define $D_{x\widetilde{y}}\, e$ as $D_{\widetilde{b_m}}(\cdots(D_{\widetilde{b_1}}(D_{a_n}(\cdots(D_{a_1}e)))))$.

Lemma 5.2.2. *For pruned derivatives and $x, y \in \Sigma^*$, the following hold.*

$$
\begin{aligned}
D_{x\widetilde{y}}\,\emptyset &= \emptyset \\
D_{x\widetilde{y}}\,\varepsilon &= \begin{cases} \varepsilon & (x = \varepsilon) \\ \emptyset & (x \neq \varepsilon) \end{cases} \\
D_{x\widetilde{y}}\,a &= \begin{cases} a & (xy = \varepsilon) \\ \varepsilon & (x = a) \\ \emptyset & (\mathit{otherwise}) \end{cases} \\
D_{x\widetilde{y}}X &= \begin{cases} X_x & (y = \varepsilon) \\ \&X_{x\widetilde{y}} & (y \neq \varepsilon) \end{cases} \\
D_{x\widetilde{y}}(e_1 \oplus e_2) &= D_{x\widetilde{y}}e_1 \oplus D_{x\widetilde{y}}e_2 \\
D_{x\widetilde{y}}(e_1 \otimes e_2) &= \sum_{x_1x_2=x} D_{x_1\widetilde{x_2y}}e_1 \otimes D_{x_2\widetilde{y}}e_2 \\
D_{x\widetilde{y}}(!e) &= \begin{cases} !(\sum_{y_1y_2=y} D_{y_1\widetilde{y_2}}e) & (x = \varepsilon) \\ \emptyset & (x \neq \varepsilon) \end{cases}
\end{aligned}
$$

Proof. For base cases and the case of $e_1 \oplus e_2$, we can easily show the equation. For the case of $e_1 \otimes e_2$, we show the equation by induction on $x\widetilde{y}$.

$$
\begin{aligned}
D_\varepsilon(e_1 \otimes e_2) &= \textstyle\sum_{x_1x_2=\varepsilon} D_{x_1\widetilde{x_2}}e_1 \otimes D_{x_2}e_2 \\
D_{xa}(e_1 \otimes e_2) &= D_a\left(\textstyle\sum_{x_1x_2=x} D_{x_1\widetilde{x_2}}e_1 \otimes D_{x_2}e_2\right) \\
&= \textstyle\sum_{x_1x_2=x} D_{x_1\widetilde{x_2a}}e_1 \otimes D_{x_2}e_2 \\
&\quad \oplus D_{x_1\widetilde{x_2a}}e_1 \otimes D_{x_2a}e_2 \\
&= D_{xa}e_1 \otimes e_2 \\
&\quad \oplus \sum_{x_1x_2=x} D_{x_1\widetilde{x_2a}}e_1 \otimes D_{x_2a}e_2 \\
&= \textstyle\sum_{x_1x_2=xa} D_{x_1\widetilde{x_2}}e_1 \otimes D_{x_2}e_2 \\
D_{x\widetilde{ya}}(e_1 \otimes e_2) &= D_{\widetilde{a}}\left(\textstyle\sum_{x_1x_2=x} D_{x_1\widetilde{x_2y}}e_1 \otimes D_{x_2\widetilde{y}}e_2\right) \\
&= \textstyle\sum_{x_1x_2=x} D_{x_1\widetilde{x_2ya}}e_1 \otimes D_{x_2\widetilde{ya}}e_2
\end{aligned}
$$

We can prove the case of $!e$ in the same way. Finally, for the case of X, we show the equation by induction on $x\widetilde{y}$. $D_\varepsilon X = X$, $D_{xa}X = X_{xa}$, $D_{x\widetilde{a}}X = \&X_{x\widetilde{a}}$, and for $y \neq \varepsilon$, $D_{x\widetilde{ya}}X = D_{\widetilde{a}}(\&X_{x\widetilde{y}}) = !(D_a(!X_{x\widetilde{y}}) \oplus D_{\widetilde{a}}(!X_{x\widetilde{y}})) = !(!(X_{x\widetilde{ya}} \oplus X_{x\widetilde{ya}})) = \&X_{x\widetilde{ya}}$. In the computation, $X_{x\widetilde{ya}}(y \neq \varepsilon)$ is replaced with $\emptyset$. □

From Lemma 5.2.2, $D_{x\widetilde{y}}(XY) = \sum_{x_1x_2=x} X_{x_1\widetilde{x_2y}}Y_{x_2\widetilde{y}}$ and $D_{\widetilde{y}}(!X) = !(\sum_{y_1y_2=y} X_{y_1\widetilde{y_2}})$ hold. Therefore, for $e \in \mathrm{RELA}^{\mathrm{bin}}(\Sigma, V)$, we obtain $|D_{x\widetilde{y}}e| = O(|xy||e|)$.

We write $\mathrm{Part}(x)$ for $\{x_1\widetilde{x_2} \mid x = x_1x_2\}$ and $\mathrm{Suf}(x)$ for $\{x_2 \mid x = x_1x_2\}$. Hence, $\mathrm{Part}(\mathrm{Suf}(x)) = \{x_2\widetilde{x_3} \mid x = x_1x_2x_3\}$. For $e \in \mathrm{RELA}(\Sigma, V)$, if $x_2\widetilde{x_3} \in \mathrm{Part}(\mathrm{Suf}(x))$, then $D_{x_2\widetilde{x_3}}e \in \mathrm{RELA}(\Sigma, V_{\mathrm{Part}(\mathrm{Suf}(x))})$ holds from Lemma 5.2.2. Therefore, for a CFGLA $G = (V, \Sigma, P, S)$, $x \in \Sigma^*$ and pruned derivative $D_xG = (V', \Sigma, P', S_x)$, we obtain $V' = V_{\mathrm{Part}(\mathrm{Suf}(x))}$. Since $|\mathrm{Part}(\mathrm{Suf}(x))| = O(|x|^2)$, the following holds.

Theorem 5.2.3. *For a binary normal form CFGLA $G = (V, \Sigma, P, S)$, $x \in \Sigma^*$ and pruned derivative D_xG, we have $|D_xG| = O(|x|^3|G|)$.*

Proof. The size of the grammar can be computed by the following: $|D_xG| = \sum_{X_{x_2\widetilde{x_3}} \in V_{\mathrm{Part}(\mathrm{Suf}(x))}} |D_{x_2\widetilde{x_3}}P(X)| = O(|x|^2(\sum_{X\in V} |x||P(X)|)) = O(|x|^3|G|)$. □

The definition of $D_\$G$ can be used without modification. Here, $\varepsilon \in \mathcal{L}_{int}(G) \iff \varepsilon \in \mathcal{L}_{int}(D_\$G)$ holds. When $\Sigma = \emptyset$, the set of intervals $(\mathrm{Int}(\mathcal{P}(\Sigma^* \times \Sigma^*)), \cup, \cdot, !)$ is isomorphic to the set of three-valued truth values $(\{0, \frac{1}{2}, 1\}, \max, \min, \lambda x.\ 1 - x)$. Therefore, computation of the least solution for $D_\$G$ can be reduced to a computation of Fitting model [14] for a propositional logic program.

Theorem 5.2.4. *For a given CFGLA G, $\varepsilon \in \mathcal{L}_{int}(G)$ is decidable in $O(|G|)$ time.*

Proof. We compute $D_\$G = (V, \emptyset, P', S)$ and the least solution $B_{P',\mathrm{least}}$ for $D_\$G$. It can be done in linear time, e.g., see Proposition 4 in [7]. Finally, we output *true* if $(\varepsilon, \varepsilon) \in B_{P',\mathrm{least}}(S)_\downarrow$; otherwise, we output *false*. □

Finally, we obtain the following theorem.

Theorem 5.2.5. *For a CFGLA G and $x \in \Sigma^*$ of length n, $x \in \mathcal{L}_{int}(G)$ is decidable in $O(n^3|G|)$ time and $O(n^2|G|)$ space.*

Proof. We convert G into the binary normal form grammar G', compute the pruned derivative D_xG', and decide $\varepsilon \in \mathcal{L}_{int}(D_xG')$. The grammar D_xG' can be constructed in $O(n^3|G|)$ time, and the membership of the empty string is decidable in $O(|D_xG'|) = O(n^3|G|)$ time. Therefore, this algorithm takes $O(n^3|G|)$ time.

Let x be $a_1 \dots a_n$. We consider the subgrammar sequence $D_\$G \leq D_{a_n\$}G \leq D_{a_{n-1}a_n\$}G \leq \cdots \leq D_{a_i\dots a_n\$}G \leq \cdots \leq D_{a_1\dots a_n\$}G$. The numbers of variables of these grammar are $O(n^2|V|)$, and the size difference $|D_{a_{i-1}\dots a_n\$}G| - |D_{a_i\dots a_n\$}G|$ is $O(n^2|G|)$ for any i. We can use the results of computation on subgrammar. Therefore, the algorithm runs in $O(n^3|G|)$ time and $O(n^2|G|)$ space. □

This algorithm fills the cells in the table in Section 5.1, row by row, from the bottom.

Chapter 6

Conclusions

We have studied regular expressions with lookahead (RELAs). We have first given the semantics of RELA by languages with lookahead, which is a set of pairs of strings. We have then developed derivatives of RELA and a conversion from RELA to DFA using derivatives.

We have introduced context-free grammars with lookahead (CFGLAs). To accommodate negative lookahead, we have given the limit semantics with the limit of iterations from the empty set. We have shown that the grammars are an extension of both CFGs and PEGs. We also have shown that the language class is closed under union, intersection, complement, marked concatenation, and marked Kleene star. To overcome the problems of the limit semantics, we have introduced the least interval semantics of CFGLA with the least fixed point of pairs of lower and upper bounds. We finally have developed a recognition algorithm for CFGLA by combining derivatives of CFG and RELA. The recognition algorithm runs in $O(n^3|G|)$ time and $O(n^2|G|)$ space for an input string of the length n and a given CFGLA G under the least interval semantics.

6.1 Open Problems

There are several open problems. First, there is the question of whether a CFGLA can be converted into an equivalent CFGLA which has a unique solution. This is possible for CFGs and Boolean grammars, but unsolved for CFGLAs. The difference is due to the fact that a CFGLA is characterized by l-languages. In particular, the difficulty arises from the fact that there is an infinite number of l-languages of the form $\{\varepsilon\} \times L$. This is related to the question of whether the language class of CFGLA is closed under complement in the least interval semantics. If CFGLA can be converted into an equivalent CFGLA which has a unique solution, then the language class of CFGLA is closed under complement. It is also open whether the language class of CFGLA is closed under concatenation and Kleene star. If the class is not closed, it is difficult to prove this. This is because there are no enough methods to show that a language is not a language of CFGLA. PEGs and Boolean grammars have the same problem. Hence, it is open whether there is a language of CFGLA that is not a language of a PEG and the relationship between the language class of Boolean grammars. The former is related to the well-known conjecture that there will be a context-free language that is not a language of a PEG.

Publications

1. Takayuki Miyazaki and Yasuhiko Minamide. Derivatives of regular expressions with lookahead. *Journal of Information Processing*, Vol. 27, pp. 422–430, 2019.
2. Takayuki Miyazaki and Yasuhiko Minamide. Context-free grammars with lookahead. In *International Conference on Language and Automata Theory and Applications*, pp. 213–225. Springer, 2021.
3. Takayuki Miyazaki and Yasuhiko Minamide. Derivatives of context-free grammars with lookahead. *Journal of Information Processing*, Vol. 31, pp. 421-431, 2023.

References

[1] Michael D Adams, Celeste Hollenbeck, and Matthew Might. On the complexity and performance of parsing with derivatives. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 224–236. ACM, 2016.

[2] Alfred V Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM*, Vol. 15, No. 4, pp. 647–671, 1968.

[3] Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, Vol. 155, No. 2, pp. 291–319, 1996.

[4] Jean-Michel Autebert, Jean Berstel, and Luc Boasson. Context-free languages and pushdown automata. In *Handbook of formal languages*, pp. 111–174. Springer, 1997.

[5] Mikhail Barash and Alexander Okhotin. An extension of context-free grammars with one-sided context specifications. *Information and Computation*, Vol. 237, pp. 268–293, 2014.

[6] Mikhail Barash and Alexander Okhotin. Linear grammars with one-sided contexts and their automaton representation. *RAIRO - Theoretical Informatics and Applications*, Vol. 49, No. 2, pp. 153–178, 2015.

[7] Kenneth A Berman, John S Schlipf, and John V Franco. Computing the well-founded semantics faster. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 113–126. Springer, 1995.

[8] Jean Berstel. *Transductions and context-free languages*. Teubner Verlag, 1979.

[9] Janusz A Brzozowski. Derivatives of regular expressions. *J. ACM*, Vol. 11, No. 4, pp. 481–494, 1964.

[10] Janusz A. Brzozowski and Ernst Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, Vol. 10, No. 1, pp. 19–35, 1980.

[11] Nariyoshi Chida and Kimio Kuramitsu. Parsing expression grammars with unordered choices. *Journal of Information Processing*, Vol. 25, pp. 975–982, 2017.

[12] N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems*, Vol. 35, pp. 118–161. Elsevier, 1963.

[13] Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical systems theory*, Vol. 10, No. 1, pp. 289–303, 1977.

[14] Melvin Fitting. A Kripke-Kleene semantics for logic programs. *The Journal of Logic Programming*, Vol. 2, No. 4, pp. 295–312, 1985.

[15] Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theoretical computer science*, Vol. 278, No. 1-2, pp. 25–51, 2002.

[16] Bryan Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.

111–122. ACM, 2004.

[17] Tony Garnock-Jones, Mahdi Eslamimehr, and Alessandro Warth. Recognising and generating terms using derivatives of parsing expression grammars. *arXiv:1801.10490*, 2018.

[18] Wouter Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theoretical Computer Science*, Vol. 411, No. 31-33, pp. 2987–2998, 2010.

[19] Jonathan S Golan. *Semirings and their Applications*. Kluwer Academic Publishers, Dordrecht, 1999.

[20] Ian Henriksen, Gianfranco Bilardi, and Keshav Pingali. Derivative grammars: a symbolic approach to parsing with derivatives. *Proceedings of the ACM on Programming Languages*, Vol. 3, No. OOPSLA, pp. 1–28, 2019.

[21] Aravind K Joshi, Leon S Levy, and Masako Takahashi. Tree adjunct grammars. *Journal of computer and system sciences*, Vol. 10, No. 1, pp. 136–163, 1975.

[22] Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, Vol. 21. Birkhäuser, Boston, 2001.

[23] Vassilis Kountouriotis, Christos Nomikos, and Panos Rondogiannis. Well-founded semantics for boolean grammars. *Information and Computation*, Vol. 207, No. 9, pp. 945–967, 2009.

[24] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, Vol. 19, No. 3, pp. 427–443, 1997.

[25] Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, Vol. 49, No. 1, pp. 95–106, 2012.

[26] Matthew Might, David Darais, and Daniel Spiewak. Parsing with derivatives: a functional pearl. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, pp. 189–195. ACM, 2011.

[27] Akimasa Morihata. Translation of regular expression with lookahead into finite state automaton. *Computer Software*, Vol. 29, No. 1, pp. 147–158, 2012.

[28] Akimasa Morihata. On the size of deterministic finite automata obtained from xpath expression. 情報処理学会第 106 回プログラミング研究発表会, 2015.

[29] Aaron Moss. Derivatives of parsing expression grammars. In *Proceedings 15th International Conference on Automata and Formal Languages*, pp. 180–194, 2017.

[30] Aaron Moss. Simplified parsing expression derivatives. In *International Conference on Language and Automata Theory and Applications*, pp. 425–436. Springer, 2020.

[31] Hanne Riis Nielson and Flemming Nielson. *Semantics with applications: an appetizer*. Springer Science & Business Media, 2007.

[32] Alexander Okhotin. Boolean grammars. *Information and Computation*, Vol. 194, No. 1, pp. 19–48, 2004.

[33] Scott Owens, John Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *Journal of Functional Programming*, Vol. 19, No. 2, pp. 173–190, 2009.

[34] Terence Parr and Kathleen Fisher. LL(*) the foundation of the ANTLR parser generator. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 425–436, 2011.

[35] Barkley Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Math-*

ematics, Vol. 63, No. 1, pp. 211–232, 1941.

[36] Yuto Sakuma, Yasuhiko Minamide, and Andrei Voronkov. Translating regular expression matching into transducers. *Journal of Applied Logic*, Vol. 10, No. 1, pp. 32–51, 2012.

[37] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, Vol. 27, No. 1, pp. 27–29, 1996.

[38] Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th annual ACM symposium on Theory of computing*, pp. 1–9. ACM, 1973.

[39] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, Vol. 11, No. 6, pp. 419–422, 1968.

[40] Allen Van Gelder, Kenneth A Ross, and John S Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, Vol. 38, No. 3, pp. 619–649, 1991.

[41] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the 14th annual ACM symposium on Theory of computing*, pp. 137–146. ACM, 1982.

[42] Detlef Wotschke. The boolean closures of the deterministic and nondeterministic context-free languages. In *GI Gesellschaft für Informatik e. V.*, pp. 113–121. Springer, 1973.