正規表現型を用いた解析表現文法への型付け

山口 大輔　倉光 君郎

本稿では，正規表現型を解析表現文法に型付けする方法について述べる．解析表現文法に型付けられた正規表現型は導出木の集合を表し，導出木の構造 (サブノードの数やノードの名前，順序) を記述することができる．解析表現文法に型付けを行う目的は，解析表現文法をベースとしたパーサから出力される導出木の構造について型検査による検証を行うためである．我々は解析表現文法から導出木を出力するパーサへの関係を定義し，この関係に対し型が保存することを確認した．

In this paper, we describe a method of typing regular expression types into parsing expression grammar. Regular expression types that is typed into parsing expression grammar represent a set of derived trees and describe the structure of the derived tree(i.e. the number of subnodes, the name of the node and the order). The purpose of typing parsing expression grammar is to describe the structure of the derived tree outputted from the parser based on the parsing expression grammar and to verify with type checking. We defined relation from parsing expression grammar to parsers that output derived trees and confirmed that the type preserves in this relation.

1　はじめに

解析表現文法 (parsing expression grammar, PEG) [3] は，2004 年に発表された形式文法である．解析表現文法の意味論は，再帰下降パーサとして定義される．

解析表現文法をベースとしたパーサを利用するプログラマは，パーサの出力する導出木に対し操作を行うプログラムを実装する必要がある．しかし，導出木の構造 (例えば，ノードの数やノードの名前，順序) はパーサの入力に依存するため，存在しないノードへの参照によるエラーや意図しないノードへの参照といった実行時エラーを引き起こしやすいという問題がある．

本稿では，型検査によってこのようなエラーが起こらないを保証するために，解析表現文法への型付けを行った．型は，正規表現型 (regular expression types) [5] を用いており，導出木の集合を定義するだけでなく，導出木の構造を記述することができる．本稿では解析表現文法とパーサの導出木の二項関係に対して，型が保存することを示した．これによって，解析表現文法が正規表現型で型付けされるとき，パーサの導出木の集合は正規表現型の表す構造を持つ導出木の集合に一致することがいえる．

本研究によって，任意の解析表現文法に対し型付けができることが示された．また，パーサの導出木の構造に対して型による仕様を記述することが可能となり，パーサがこの仕様に適合するか否かを型検査によって判定することが可能となった．

以降の構成は次の通りである．2 節では，正規表現型を用いた解析表現文法への型付けの直観を具体例を用いて説明する．3 節では，準備として解析表現文法と正規表現型について説明をし，導出木の記法の定義をする．4 節では，解析表現文法とパーサの導出木の二項関係を定義する．5 節では型付け規則を定義し，解析表現文法とパーサの導出木の二項関係に対し型が保存されることを示す．6 節では，関連研究をまと

Type System for Parsing Expression Grammar based on Regular Expression Types

Daisuke Yamaguchi, Kimio Kuramitsu, 横浜国立大学大学院工学府, Graduate School of Engineering, Yokohama National University.

める．最後に，7 節でまとめと今後の展望を述べる．

2 正規表現型の解析表現文法への型付け

正規表現型は，細谷らの XML 処理言語 [4] の基盤として，XML 文書への型付けを行うために定義された．正規表現型はラベル付けられた型 (`label[...]`) や正規表現にある記法 (`*`,`|`,`?`など) によって XML 文書の構造を表すことができる．我々は，XML 文書の構造は木構造であることに着目しパーサの導出木の構造を表す型として正規表現型を用いて正規表現型を解析表現文法に型付けする規則を定義した．

正規表現型を解析表現文法に型付けする単純な例として，図 1 に定義される解析表現文法への型付けを考える． この解析表現文法は，`1*2` や `1*2*5*8` の

$$\begin{aligned} Prod &\leftarrow Val\ (*\ Val)^* \\ Val &\leftarrow [0-9] \end{aligned}$$

図 1 単純な解析表現文法の例

ような乗算の式を認識するパーサに対応し，以下に定義される型 `Prod` が型付けされる．

```
type Prod = Prod[Val,(String,Val)*]
type Val = Val[String]
```

型 `Prod` の表す導出木の構造は，ノードに *Prod* がラベルつけられており，子要素として型 `Val` が表す構造を持った導出木と，`String`，`Val` のそれぞれが表す構造を持った導出木の連接が 0 個以上結合することを表している．型 `Val` が表す導出木の構造は，ノードが *Val* がラベル付けられており，子要素として `String` が表す構造を持った導出木が結合することを表している．

図 1 のパーサに `1*2` を入力すると図 2 に示される導出木が得られる． この導出木の構造は，型 `Prod` の表す導出木の構造に一致している．導出木の型付け規則は，型の表す導出木の構造に型付けされる導出木の構造が一致することを反映している．

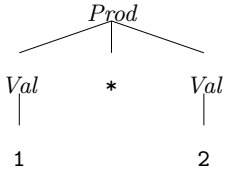

図 2 `1*2` を入力した場合の導出木

3 準備

3.1 解析表現文法

解析表現文法は，非終端記号の有限集合 N_G，終端記号の有限集合 Σ，導出規則の有限集合 P_G，開始表現 e_s の 4 つ組 (N_G, Σ, P_G, e_s) として定義される．

P_G に属する導出規則は，非終端記号と解析表現の対応を表しており，$A \leftarrow e$ という形式で記述される．ただし，$A \in N_G$，e は解析表現である．P_G は，非終端記号から解析表現への写像とみることができる．本稿では，P_G の定義域を $\mathrm{dom}(P_G)$ で表す．本稿で扱う解析表現の構文は，図 3 に示す通りである．

e	::=	ε	空文字
		a $(a \in \Sigma)$	終端記号
		A $(A \in N_G)$	非終端記号
		$e\ e$	連接
		e/e	優先度付き選択
		e^*	0 回以上の繰り返し
		$!e$	否定先読み

図 3 解析表現の構文

解析表現の構文には，図 3 の他に便利なシンタックスシュガーが定義されている [3] が本稿ではそれらの説明は省略する．

3.2 解析表現文法の導出木

本稿で扱う導出木の記法について定義を行う．解析表現文法 $G = (N_G, \Sigma, P_G, e_s)$ の導出木は，非終端記号の有限集合 N_G と終端記号の有限集合 Σ の直積集合の上で定義される多分木である．ノードは非終端記号または終端記号列でラベル付けられている．

$N_G \times \Sigma$ 上の導出木の集合を $\mathcal{T}_{N_G,\Sigma}$ で表すとき，

導出木 $t \in \mathcal{T}_{N_G,\Sigma}$ は次のように定義される．

$$\begin{array}{rcll} t & ::= & l[t] \quad (l \in N_G \cup \Sigma^*) & \text{ノード} \\ & & t,t & \text{連接} \\ & & () & \text{空の木} \end{array}$$

ノード $l[t]$ は，N_G 上の非終端記号または，終端記号列のラベル l が付けられており，導出木 t を子ノードとして持つ．複数のサブノードをもつ場合は，導出木の連接を表す演算子',' を用いる．ただし，演算子',' は可換ではない．

例として，図 2 の導出木をここで定義した記法で表すと，

```
Prod[Val[1],*,Val[2]]
```

となる．ただし，`1`，`*`，`2` はそれぞれ，`1[()]`，`*[()]`，`2[()]` の縮約表記である．以降もこの表記を用いる．

Prod でラベルつけられたノードの子ノード，`Val[1],*,Val[3]` は，最左木から `Val[2]` ，`*`，`Val[3]` という順で親ノードに結合することを表す．

また，演算子',' は可換でないほかに，次の等価関係が成り立つ．

$$\begin{aligned} t,() &\equiv t \\ (),t &\equiv t \end{aligned}$$

空の木 () は 0 個の導出木を表しており，演算子',' の単位元である．

3.3 正規表現型

正規表現型 [5] は，型をプログラマが読み書きするのに適した external form という形式と，部分型のアルゴリズムの実装，証明に適した internal form という形式がある．本稿の解析表現文法への型付けには external form を用いるが，external form から internal form への変換は translation 関数 [5] によって可能である．

ラベルの集合と型変数の加算無限集合を仮定するとき，それぞれの元を `L`,`X` で表す．正規表現型 `T` の構文は，図 4 の通り定義される．

型変数の束縛は，`type X = T` という形式で環境 E に保持される．E は，型変数から型への写像であり，$\mathrm{dom}(E)$ によって，E に定義されている型変数の集合を表す．

`String` 型のような基底となる型は，`String[()]` というラベル型の縮約表記として扱い以降もこの表記を用いる．

`T`	::=	`()`	empty
		`X`	型変数
		`L[T]`	ラベル
		`T,T`	連接
		`T|T`	選択
		`T*`	繰り返し

図 4　正規表現型の構文

4 解析表現文法とパーサの導出木の二項関係

解析表現文法に型付けを行う目的に，パーサのインスタンスが出力する導出木の集合を型によって記述することがある．これが満たされるためには，解析表現文法とパーサの導出木への二項関係について型が保存されることが重要である．これを示すため，本節では解析表現文法の意味論を導出木を出力するパーサとして再定義を行う．

我々は解析表現文法 $G = (N_G, \Sigma, P_G, e_s)$ の意味論を，対 (e,x) から対 (o,y) への二項関係 $\Downarrow_G$ で定義する．ただし，e は解析表現，x は入力される非終端記号列を表す．y は読み残された非終端記号列を表す．o は解析結果を表しており $o \in \mathcal{T}_{N_G,\Sigma} \cup \{f\}$ である．f は，構文解析が失敗しバックトラックすることを表す記号である．

本稿では，$((e,x),(o,y)) \in \Downarrow_G$ の表記を $e \Downarrow_y^x o$ とする．$e \Downarrow_y^x o$ は，解析表現 e は x を入力とし読み残された記号列 y を伴って出力 o に写されると読む．

いま，$t \in \mathcal{T}_{N_G,\Sigma}$， $a,b,c \in \Sigma$, $x,y,z \in \Sigma^*$ とすれば，$\Downarrow_G$ は図 5 の規則からなる最小の関係である．

Ford による意味論の定義 [3] に対し，解析表現文法の意味論を導出木を出力する再帰下降パーサとして定義したことによって，ε は，空の木 () を出力するパーサに写されるという違いがある (E-Empty)．それ以外に，終端記号 a は，ラベル a のノードに写される (E-Term1)，非終端記号 A は，ラベル A のノードに写される (E-Nt)，否定先読み $!e$ が失敗しない場合 (E-Not2) は，$!e$ が空の木 () に写されるという違いがある．

$$\varepsilon \Downarrow_x^x () \quad \text{(E-Empty)}$$

$$a \Downarrow_x^{ax} a[()] \quad \text{(E-Term1)}$$

$$a \Downarrow_x^{bx} f \quad (a \neq b) \quad \text{(E-Term2)}$$

$$\frac{A \in \mathrm{dom}(P_G) \qquad P_G(A) \Downarrow_y^x t}{A \Downarrow_y^x A[t]} \quad \text{(E-Nt)}$$

$$\frac{e_1 \Downarrow_{x_2 y}^{x_1 x_2 y} t_1 \qquad e_2 \Downarrow_y^{x_2 y} t_2}{e_1 e_2 \Downarrow_y^{x_1 x_2 y} t_1, t_2} \quad \text{(E-Seq1)}$$

$$\frac{e_1 \Downarrow_x^x f}{e_1 e_2 \Downarrow_x^x f} \quad \text{(E-Seq2)}$$

$$\frac{e_1 \Downarrow_y^{x_1 y} t_1 \qquad e_2 \Downarrow_y^y f}{e_1 e_2 \Downarrow_y^{x_1 y} f} \quad \text{(E-Seq3)}$$

$$\frac{e_1 \Downarrow_y^{xy} t_1}{e_1 / e_2 \Downarrow_y^{xy} t_1} \quad \text{(E-Alt1)}$$

$$\frac{e_1 \Downarrow_{xy}^{xy} f \qquad e_2 \Downarrow_y^{xy} t_2}{e_1 / e_2 \Downarrow_y^{xy} t_2} \quad \text{(E-Alt2)}$$

$$\frac{e_1 \Downarrow_x^x f \qquad e_2 \Downarrow_x^x f}{e_1 / e_2 \Downarrow_x^x f} \quad \text{(E-Alt3)}$$

$$\frac{e \Downarrow_{x_2 y}^{x_1 x_2 y} t_1 \qquad e^* \Downarrow_y^{x_2 y} t_n}{e^* \Downarrow_y^{x_1 x_2 y} t_1, t_n} \quad \text{(E-Rep1)}$$

$$\frac{e \Downarrow_x^x f}{e^* \Downarrow_x^x ()} \quad \text{(E-Rep2)}$$

$$\frac{e \Downarrow_y^{xy} t}{!e \Downarrow_{xy}^{xy} f} \quad \text{(E-Not1)}$$

$$\frac{e \Downarrow_x^{xy} f}{!e \Downarrow_x^x ()} \quad \text{(E-Not2)}$$

図 5　二項関係 $\Downarrow_G$

解析表現文法 $G = (N_G, \Sigma, P_G, e_s)$ のパーサから導出木が出力される場合は，ある終端記号列 x が与えられたとき，e_s が (E-Empty),(E-Term1),(E-Nt),(E-Seq1),(E-Alt1),(E-Alt2),(E-Rep1),(E-Rep2),(E-Not2) のいずれかによって導出木 t に写される場合に限る．これらの規則が成り立つときの終端記号列 x の集合を $\mathcal{L}(e_s)$ と表すことにすれば，解析表現文法 G とそのパーサの導出木の関係は，入力される終端記号列を $x \in \mathcal{L}(e_s)$ としたときの (E-Empty),(E-Term1),(E-Nt),(E-Seq1),(E-Alt1),(E-Alt2),(E-Rep1),(E-Rep2),(E-Not2) で定義される．

5　正規表現型の型付け

正規表現型は解析表現文法と，導出木に型付けられる．

解析表現文法 $G = (N_G, \Sigma, P_G, e_s)$ が型 $\mathtt{T}$ で型付けられるとは，開始表現 e_s が型 $\mathtt{T}$ で型付けられることをいう．この型付け関係は $\Gamma \vdash e_s : \mathtt{T}$ と表し，図 6 に定義される規則の集合からなる最小の関係と定義する． 図 6 の型付け規則は，補助的に非終端記号に型変数を割り当てる規則を定義した環境 Γ を用いる．Γ は N_G から型変数名の集合に 1 対 1 に対応付ける写像である．Γ の定義域を $\mathrm{dom}(\Gamma)$ と表し，$A \in N_G$ に割り当てられる型変数を $\Gamma(A)$ と表す．

$$\Gamma \vdash \varepsilon : () \quad \text{(T-Empty)}$$

$$\Gamma \vdash a : \mathtt{String} \quad \text{(T-Term)}$$

$$\frac{E(\mathtt{X}) = A[\mathtt{T_A}] \qquad \Gamma, A : \mathtt{X} \vdash P_G(A) : \mathtt{T_A}}{\Gamma \vdash A : \mathtt{X}} \quad \text{(T-Nt)}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{T_1} \qquad \Gamma \vdash e_2 : \mathtt{T_2}}{\Gamma \vdash e_1 e_2 : \mathtt{T_1}, \mathtt{T_2}} \quad \text{(T-Seq)}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{T_1} \qquad \Gamma \vdash e_2 : \mathtt{T_2}}{\Gamma \vdash e_1 / e_2 : \mathtt{T_1} | \mathtt{T_2}} \quad \text{(T-Alt)}$$

$$\frac{\Gamma \vdash e : \mathtt{T}}{\Gamma \vdash e^* : \mathtt{T}*} \quad \text{(T-Rep)}$$

$$\vdash !e : () \quad \text{(T-Not)}$$

図 6　解析表現の型付け規則

型変数は `type X = T` という形式で環境 E に束縛される．

簡単な例として解析表現文法の型付け導出を考える．解析表現文法:

$$\begin{aligned} G \quad = \quad & (\{Val, Num\} \\ , \quad & \{0\} \\ , \quad & \{Val \leftarrow Num, Num \leftarrow 0\} \\ , \quad & Val) \end{aligned}$$

が以下の型変数の束縛環境 E に束縛される正規表現型 $\mathtt{X}$ で型付けされるとき，

```
type X = Val[Y]
type Y = Num[String]
```

$\Gamma = \{Val : \mathtt{X}, Num : \mathtt{Y}\}$ を導入することで型付け導出は図 7 のようになる．

解析表現文法についての型検査のアルゴリズムは，図 6 に示す規則が構文主導であることから導出規則を下から上に辿ることで得られる．

次に，導出木 t に型 $\mathtt{T}$ が型付けられることを関係 $t : \mathtt{T}$ で定義する．関係 $t : \mathtt{T}$ は図 8 に定義される規則の集合からなる最小の関係である．

(T-Sub1) は，型 `String` で型付けされる構文木の集合は，l を任意のラベルとするとき，型 $l[()]$ で型付けされる構文木の集合を内包することを表す．(T-Sub2) は，型 () で型付けされる構文木は任意の型 $\mathtt{T}$ によって型付けされる構文木の集合に内包されることを表す．これら 2 つの規則以外は，正規表現型の型付け規則と同様である．

5.1 型の保存

図 6,8 で定義された型付け規則は，4 節で定義された解析表現文法と導出木の関係について保存する．

定理 5.1 (保存)．解析表現文法 $G = (N_G, \Sigma, P_G, e_s)$ について $e_s \Downarrow_y^{x \in \mathcal{L}(e_s)} t$ かつ，$e_s : \mathtt{T}$ ならば，$t : \mathtt{T}$

Proof e_s についての場合分けと，$e_s \Downarrow_y^{\mathcal{L}(e_s)} t$ の導出規則に関する帰納法による．

自明でない場合は $e_s = A \in N_G$ の場合である．命題の仮定は，(E-Nt) より，

$$A \Downarrow_y^{x \in \mathcal{L}(A)} A[t_A] \quad \text{(A1)}$$

$$P_G(A) \Downarrow_y^{x \in \mathcal{L}(A)} t_A \quad \text{(A1')}$$

と，(T-Nt) より，

$$A : \mathtt{X} \quad \text{(A2)}$$

$$E(\mathtt{X}) = A[\mathtt{T_A}] \quad \text{(A2')}$$

が成り立つ．また，帰納法の仮定から，

$$P_G(A) : \mathtt{T_A} \quad \text{(IH1)}$$

$$t_A : \mathtt{T_A} \quad \text{(IH2)}$$

が成り立つ．(IH2) を (T-Node) に適用すれば，

$$A[t_A] : A[\mathtt{T_A}] \quad \text{(IH2')}$$

が成り立つ．(A2'),(IH2') を (T-Var) に適用すれば，$A[t_A] : \mathtt{X}$ となり成立する．

□

6 関連研究

解析表現文法は Ford による形式化以降盛んに研究がなされてきた．特に解析表現文法のパーサへの応用に関連して多くの研究が挙げられる [6] [7] [12] [11] [2] [9]．解析表現文法理論的な背景の研究としては，これまで明らかでなかった言語のクラスに関する研究などが挙げられる [8] [1] ．

しかし，我々の知る限りでは解析表現文法への型付けはこれまでなされなかった．我々は本稿で示した解析表現文法への型付けは新しい着眼点であると考えている．

これまで，解析表現文法へ型付けがなされなかった理由は，解析表現文法の意味論は消費された文字列を出力するパーサとして定義されることから型付けを行う動機が得られなかったためであると考えられる．多くの解析表現文法を用いたパーサジェネレータでは，解析表現文法を用いて構文を定義することができるがパーサの出力する導出木を含めた定義は行われない．そのため，導出木の構築はアドホックな補強項によって定義され，多くの場合それらはパーサホスト言語の型システムによって型付けがされる．

一方，我々は解析表現文法の意味論を導出木を出力するパーサとして再定義を行った．これによって，解析表現文法を用いて導出木の定義を行うことが可能となった．その結果，導出木を操作するプログラムを実装する上での難しさから解析表現文法への型付けを行う動機を得た．本研究で示した解析表現文法への型付けは，パーサの導出木の構造について型検査による検証を行うことを目的としており，本研究で定義された型付け規則によって任意の解析表現文法への型付けが可能であることを示した．

7 結び

本稿では，はじめに解析表現文法の意味論を導出木を出力するパーサとして再定義した．また，解析表現文法と導出木それぞれに正規表現型を型付けする規則を定義し，この型付けが解析表現文法とそのパーサから生成される導出木の二項関係関係で型が保存

$$\dfrac{E(\mathtt{X}) = Val[\mathtt{Y}] \qquad \dfrac{E(\mathtt{Y}) = Num[\mathtt{String}] \qquad \dfrac{}{\Gamma, Val : \mathtt{X}, Num : \mathtt{Y} \vdash 0 : \mathtt{String}}\ \text{(T-Term)}}{\Gamma, Val : \mathtt{X} \vdash Num : \mathtt{Y}}\ \text{(T-Nt)}}{\Gamma \vdash Val : \mathtt{X}}\ \text{(T-Nt)}$$

図 7　型付け導出の例

$$\frac{t : \mathtt{T}}{l[t] : l[\mathtt{T}]}\ \text{(T-Node)}$$

$$\frac{t_1 : \mathtt{T}_1 \qquad t_2 : \mathtt{T}_2}{t_1 t_2 : \mathtt{T}_1, \mathtt{T}_2}\ \text{(T-Seqt)}$$

$$() : ()\ \text{(T-Emptyt)}$$

$$\frac{E(\mathtt{X}) = \mathtt{T} \qquad t : \mathtt{T}}{t : \mathtt{X}}\ \text{(T-Var)}$$

$$\frac{t : \mathtt{T}_1}{t : \mathtt{T}_1 | \mathtt{T}_2}\ \text{(T-Or1)}$$

$$\frac{t : \mathtt{T}_2}{t : \mathtt{T}_1 | \mathtt{T}_2}\ \text{(T-Or2)}$$

$$\frac{t_i : \mathtt{T} \text{ for each } i}{t_1, \ldots, t_n : \mathtt{T}*}\ \text{(T-Rept)}$$

$$\frac{t : l[()]}{t : \mathtt{String}}\ \text{(T-Sub1)}$$

$$\frac{t : ()}{t : \mathtt{T}}\ \text{(T-Sub2)}$$

図 8　導出木の型付け規則

されることを示した．本稿で示した型付け規則によって，任意の解析表現文法に正規表現型を型付けできることが示された．また，パーサの導出木の構造に対して型による仕様を記述することが可能となり，パーサがこの仕様に適合するか否かを型検査によって判定することが可能となった．しかし，本稿で定義した解析表現文法への型付け規則は構文主導である反面，正規表現型の部分型関係に基づいた型付け規則を認めないため融通の利かないものとなった．正規表現型は部分型関係が定義されているが，これを決定するアルゴリズムは well-formed [5] である正規表現型を前提としている．本稿で示した規則によって解析表現文法に型付けを行った結果は必ずしも well-formed 性を満たさないため，そのまま正規表現型の部分型付けのアルゴリズムを利用することはできない．解析表現文法に型付けられた正規表現型の部分型付けのアルゴリズムについては今後の課題としたい．

参 考 文 献

[1] Chida, N. and Kuramitsu, K.: *Linear Parsing Expression Grammars*, Springer International Publishing, 2017, pp. 275–286.

[2] Ford, B.: Packrat Parsing:: Simple, Powerful, Lazy, Linear Time, Functional Pearl, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, New York, NY, USA, ACM, 2002, pp. 36–47.

[3] Ford, B.: Parsing Expression Grammars: A Recognition-based Syntactic Foundation, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, New York, NY, USA, ACM, 2004, pp. 111–122.

[4] Hosoya, H. and Pierce, B. C.: XDuce: A Statically Typed XML Processing Language, *ACM Trans. Internet Technol.*, Vol. 3, No. 2(2003), pp. 117–148.

[5] Hosoya, H., Vouillon, J., and Pierce, B. C.: Regular Expression Types for XML, *ACM Trans. Program. Lang. Syst.*, Vol. 27, No. 1(2005), pp. 46–90.

[6] Koprowski, A. and Binsztok, H.: TRX: A Formally Verified Parser Interpreter, *Proceedings of the 19th European Conference on Programming Languages and Systems*, ESOP'10, Berlin, Heidelberg, Springer-Verlag, 2010, pp. 345–365.

[7] Medeiros, S. and Ierusalimschy, R.: A Parsing Machine for PEGs, *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, New York, NY, USA, ACM, 2008, pp. 2:1–2:12.

[8] Medeiros, S., Mascarenhas, F., and Ierusalimschy, R.: From Regexes to Parsing Expression Grammars, *Sci. Comput. Program.*, Vol. 93(2014), pp. 3–18.

[9] Parr, T., Harwell, S., and Fisher, K.: Adaptive LL(*) Parsing: The Power of Dynamic Analysis, *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, New York, NY, USA, ACM, 2014, pp. 579–598.

[10] Pierce, B.: *Types and Programming Languages*, MIT Press, 2002.

[11] Redziejowski, R. R.: Parsing Expression Grammar As a Primitive Recursive-Descent Parser with

Backtracking, *Fundam. Inf.*, Vol. 79, No. 3-4(2007), pp. 513–524.

[12] Redziejowski, R. R.: Applying Classical Concepts to Parsing Expression Grammar, *Fundam. Inf.*, Vol. 93, No. 1-3(2009), pp. 325–336.