情報処理学会論文誌　プログラミング　Vol.12　No.2　1–9 (May 2019)

一般化された解析表現文法とPackrat構文解析手法の提案

多田 拓[1,a)]　倉光 君郎[2,b)]

受付日 2018年9月28日, 採録日 2019年1月25日

概要：曖昧さのある形式文法から生成されたパーサは異なる解釈をした複数の結果が導出されうる．このような複数の可能性を試すパーサは非線形の計算時間を必要とし，プログラミング言語などの人工言語の構文解析において望ましくない．Parsing Expression Grammar（PEG）の強みは優先度付き選択と貪欲な繰り返しによって曖昧さがないように形式化されている点である．しかしながら，近年の文法推論や自然言語を含んだ解析への応用では曖昧さが重要となっている．本研究では，PEG に文法的な拡張を加えることで，曖昧さを追加した新しい形式化基盤 Generalized PEG（GPEG）を提案する．GPEG は決定的な PEG の文法に対して，優先度なし選択を加えた拡張となっている．一般化構文解析手法である GLR や GLL で構築される曖昧な木とは異なり，曖昧さを文法から制御可能であるため部分的に曖昧な木を構築する．さらに，本研究で提案している generalized packrat parsing によって実用的な時間で構文解析が可能である．

キーワード：解析表現文法，Packrat 構文解析法，一般化構文解析法

GPEG: A Generalized Foundation for Packrat Parsing

Taku Tada[1,a)]　Kimio Kuramitsu[2,b)]

Received: September 28, 2018, Accepted: January 25, 2019

Abstract: Ambiguity in a formal grammar is undesirable in a parser generation of programming languages. The strength of Parsing Expression Grammars (PEGs) is ordered choice and greedy repetition which can eliminate ambiguity from grammars. Nevertheless, the elimination of ambiguity faces several new difficulties in grammar debugging and grammar inference. We propose a formal foundation of Generalized PEGs (GPEG) by introducing unordered choices. The ambiguity is still controlled, and GPEG allows partial ambiguous tree construction, unlike tree forests in a generalized parsing such as GLR and GLL parsing. The practical parsing can be built on a generalized packrat parsing.

Keywords: parsing expression grammar, Packrat parsing, generalized parsing

1. はじめに

CFG 文法からパーサを生成する開発者は曖昧さによって苦しめられてきた．たとえば，$1 + 2 * 3$ を受理するような CFG 文法（$E ::= E + E|E * E|N$）は図 **1** のように2つの解析木が導出される．このような複数の可能性を試すパーサは非線形の計算時間を必要とする．

曖昧さに対する典型的な解決策は，曖昧な規則が書けないように文法を制限する方法である．歴史的に LR，LL は

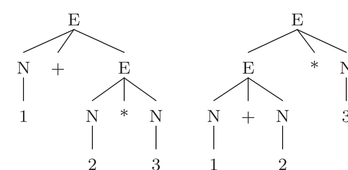

図 1　2つの解析木
Fig. 1　Two parse trees.

[1] 横浜国立大学大学院
Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
[2] 日本女子大学理学部数物科学科
Department of Mathematical and Physical Sciences, Japan Women's University, Bunkyo, Tokyo 112–868, Japan
[a)] tada-taku-jp@ynu.jp
[b)] kuramitsuk@fc.jwu.ac.jp

© 2019 Information Processing Society of Japan

主にバックトラックを避けるために CFG から曖昧さを除去した．また，近年注目されている形式文法の 1 つである Parsing Expression Grammar（PEG）[5] は優先度付き選択によって曖昧さがないように文法が設計されている．これらの形式文法から生成されたパーサはたかだか 1 つの解析木しか導出をしないため決定的である．

しかし，曖昧さが除去された文法を近年の文法開発技術 [1], [9], [15] に応用するのは困難である．これらの応用では曖昧さをより良い文法仕様の開発に役立てようと試みている．そのため，曖昧さを扱うことのできる構文解析である GLR [14]，GLL [12] といった一般化構文解析法が使用されている．

本論文では，PEG に曖昧さを加えた Generalized PEG（GPEG）を形式的に定義し，GPEG による一般化構文解析法を提案する．GPEG は PEG に優先度なし選択を加えた文法拡張であり，2 種類の曖昧さ除去が可能である．1 つは，文法の不要な優先度なし選択を優先度付き選択にすることによって曖昧さを除去する方法．もう一方は，GLR や GLL を使用する際に用いられる曖昧さ除去の手法によって解析木から曖昧さを取り除く方法である．

同じように優先度なし選択を PEG に加えた先行研究として，PEG with Unordered Choice [3] がある．本研究との違いは，マッチのみの操作的意味論を定義している点であり，これに対し本研究は一般化構文解析の曖昧な木表現（Parse Foreset）を扱えるように解析木の構築を含めた操作的意味論を提案する．

さらに，我々は効率的な一般化構文解析アルゴリズムとして Generalized Packrat 構文解析法を提案する．「Packrat」はおもに PEG のパーサ生成によく用いられる Packrat 構文解析 [4] にちなんでおり，メモ化によって再計算を防ぐことで効率化を図っている．これにより，GPEG パーサによる非常に曖昧な文法での最悪時間計算量は $O(k^n)$ から $O(n^3)$ に改善される．

本論文の構成は以下のとおりである．2 章では，PEG の一般化を行った動機について述べる．3 章では，事前知識として解析木をともなう PEG の一般化について述べる．4 章では，GPEG の形式的な定義と操作的意味論を述べる．5 章では，効率的な GPEG パーサの構文解析アルゴリズムとして Generalized Packrat 構文解析法を述べる．6 章では，5 章のアルゴリズムを実装した GPEG パーサによる性能の測定結果を述べる．7 章では，GPEG によって自然言語の構文解析を行った例を述べる．8 章では，本研究の関連研究を述べる．9 章では，結論を述べる．

2. 動機

2.1 PEG と決定的な構文解析

PEG によって入力文字列は一意な解析木に変換される．このような決定的な性質は PEG に含まれる優先度付き選択や貪欲な繰返しによるものである．たとえば，“dangling else-if” のようなよく知られた問題は以下のように PEG で記述することができる．

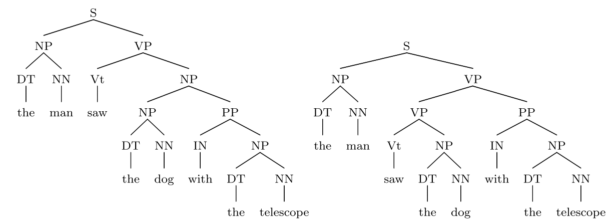

図 2　曖昧な解析木
Fig. 2　Ambiguous trees.

$$IF_STATEMENT$$
$$\leftarrow \text{ if (} EXPR \text{)} STATEMENT \text{ else } STATEMENT$$
$$/ \text{ if (} EXPR \text{) } STATEMENT$$

優先度付き選択は最初にマッチした結果を優先するため決定的であり，曖昧な解析結果を示さない．

しかしながら，優先度付き選択は CFG に慣れ親しんでいる文法開発者に評判が悪い．たとえば，$A \leftarrow a/ab$ は接頭辞が一致するため a のみにマッチし，直観に反する振舞いをする．ただ，“dangling else-if” の例はこの接頭辞に関する振舞いによって曖昧さ除去を解決した例である．

曖昧さを扱える一般化構文解析に対して, PEG は自然言語処理に向いていない．自然言語は本質的に曖昧であり，構文レベルで決定的に解析することは難しい．**図 2** に自然言語による曖昧さによって複数の解析木に解釈される例を示す．この例の曖昧さは前置詞（PP）による係り受けによって生じており，“the dog” を修飾しているのか “with the dog” を修飾するのかによってそれぞれの解析木が導出されている．

2.2 優先度なし選択

我々が提案する PEG の拡張で重要であるのは優先度付き選択を取り除くことなく，優先度なし選択を導入している点である．文法開発者は両方の選択を使用して文法の記述が可能である．優先度なし選択は記号 | を使用して以下のように記述する．

$$A \leftarrow a \mid ab$$

上記の例では a と ab のどちらにもマッチし，もし入力として ab の文字列を与えれば 2 つの異なる解析木を得ることができる．つまり，曖昧さを 2 種類の選択を使い分けることにより制御が可能であり，不要な曖昧さを取り除きバックトラッキングを抑止することで構文解析の性能を向上させることも可能である．

さらに，優先度なし選択は 2.1 節のような直観に反する振舞いから誘発されるバグを取り除くことができる．文法開発者にとって乏しいエラーメッセージからこのようなバグを発見することが難しいため，優先度なし選択によって書き

© 2019 Information Processing Society of Japan

直し，複数の解析木を生成させることで発見が容易になる．

また，GLR や GLL などの一般化構文解析は文法開発や文法推論を含む多くの興味深い応用例が存在する [1], [9], [15]．これらの応用の動機は PEG にも存在すると考えられるが，決定的な特性によって適用することが難しい．

3. 解析木をともなう PEG の形式化

GPEG は文字列から Parse Forest への変換器として定義するため，本章では文字列から木への変換器として PEG を定義する．PEG の形式化は後述の GPEG の形式化でも再度使用されることに注意されたい．

3.1 文法

PEG を文法の組として以下のように定義する．

Definition 3.1（PEG）**.** *Parsing Expression Grammar*（*PEG*）は 4 つの組 $G = (N, \Sigma, R, e_s)$ として定義する．ここで N は非終端記号の有限集合，Σ は終端記号の有限集合，R は生成規則の有限集合，e_s は開始表現である．

生成規則は非終端記号 $A \in N$ から解析表現 e への写像であり，$A \leftarrow e$ と表記される．また，$R(A)$ は $A \leftarrow e$ で関連付けられる e を表現している．

図 3 に PEG の解析表現を示す．以降では説明のためにメタ変数として $a, b, c \in \Sigma$，$A, B, C \in N$，$x, y, z \in \Sigma^*$ を使用する．空文字列 ε は空文字列にマッチする．文字 a は同じ入力文字である終端記号 a に正確にマッチする．任意の 1 文字 . は任意の終端記号 1 文字にマッチする．非終端記号 A は $R(A)$ の解析表現を試す．連結 $e_1\ e_2$ は e_1 に続けて e_2 を試す．優先度付き選択 e_1/e_2 はまず e_1 を試し，失敗した場合はバックトラックして e_2 を試す．0 回以上の繰り返し $e*$ は正規表現の繰り返しと同様に失敗するまで貪欲に e を繰り返す．否定先読み $!e$ は e が失敗するときに全体として成功とし e が成功するとき全体として失敗とするが，入力文字列を消費しない．

3.2 糖衣構文

任意の 1 文字 . の解析表現はすべての終端記号 Σ の優先度付き選択 $(a/b/...)$ として表現することができる．特別な場合がない限り，任意の 1 文字はこのような終端記号の選択の糖衣構文となる．

さらに，文字クラスやオプション，1 回以上の繰り返しなどの便利な記法は糖衣構文として以下のように扱われる．

$[abc]$	$=$	$a\ /\ b\ /\ c$	character class
e^+	$=$	ee^*	one or more repetition
$e?$	$=$	$e\ /\ \varepsilon$	option
$\&e$	$=$	$!!e$	and-predicate

3.3 解析木

解析木はラベル付き導出木として以下のように定義する．

t	$::=$	ε	empty
	$\vert$	x	string
	$\vert$	$t\ t$	concatenation
	$\vert$	$[A\ t]$	labeled tree

また，解析木の例を**図 4** に示す．

ここで，解析木と抽象構文木の間には違いがあることに注意されたい．我々の定義では，解析木は文法から導出されたものであり，抽象構文木は解析木から不必要な情報を取り除いたものである．PEG での解析木は非終端記号によってラベル付けされるものとする．

3.4 PEG パーサ

PEG パーサは文字列から木への変換器であり，$P[e]x \overset{PEG}{\leadsto} \langle t, y\rangle$ と形式化することができる．$P[e]$ パーサは与えられた G の開始表現 e_s を e に置き換えた G' に基づくパーサである．また，x は入力文字列，t は解析木，y は残り文字列である．$P[e]x \overset{PEG}{\leadsto} \langle t, y\rangle$ は $P[e]$ パーサが入力文字列 x を解析し，変換された解析木 t と残り文字列 y と読むことができる．さらに特殊な結果 $\bullet$ を解析の失敗として表現する．**図 5** に，PEG の操作的意味論を示す．

解析木の構築には以下のような注意点がある．$e*$ は任意の非終端記号 C を用いた $C = eC/\varepsilon$ と糖衣構文であるため，解析木は $[C\ t\ [C\ t, ..[C\ \varepsilon]]]$ のように右結合のリスト構造となる．一方で左結合のリスト構造は，左再帰が禁止されているために構築することができない．

4. Generalized PEG

本章では，Generalized Parsing Expression Grammar（GPEG）の形式化を行う．GPEG は PEG に優先度なし選択を追加して拡張である．

e	$::=$	ε	empty
	$\vert$	a	character
	$\vert$	.	any character
	$\vert$	A	nonterminal
	$\vert$	$e\ e$	sequence
	$\vert$	$e\ /\ e$	ordered choice
	$\vert$	$e*$	zero-or-more repetition
	$\vert$	$!e$	not-predicate

図 3　解析表現の形式的定義

Fig. 3　Syntax of a parsing expression.

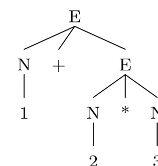

図 4　解析木（[E [N 1] + [E [N 2] * [N 3]]]）

Fig. 4　Parse tree ([E [N 1] + [E [N 2] * [N 3]]]).

© 2019 Information Processing Society of Japan

$$P[\varepsilon]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, x\rangle \quad \text{(EMPTY)} \qquad P[a]\ ax \overset{PEG}{\rightsquigarrow} \langle a, x\rangle \quad \text{(CHAR)} \qquad P[a]\ bx \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle \quad (b \neq a)\ \text{(CHAR2)}$$

$$\frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle t_1, y\rangle}{P[e_1\ /\ e_2]\ x \overset{PEG}{\rightsquigarrow} \langle t_1, y\rangle}\ \text{(CHOICE)} \qquad \frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle \quad P[e_2]x \overset{PEG}{\rightsquigarrow} \langle t_2, z\rangle}{P[e_1\ /\ e_2]\ x \overset{PEG}{\rightsquigarrow} \langle t_2, z\rangle}\ \text{(CHOICE2)}$$

$$\frac{R(A) = e \quad P[e]\ x \overset{PEG}{\rightsquigarrow} \langle t, y\rangle}{P[A]\ x \overset{PEG}{\rightsquigarrow} \langle [A\ t], y\rangle}\ \text{(NT)} \qquad \frac{R(A) = e \quad P[e]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}{P[A]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}\ \text{(NT2)}$$

$$\frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle t_1, y\rangle \quad P[e_2]\ y \overset{PEG}{\rightsquigarrow} \langle t_2, z\rangle}{P[e_1 e_2]\ x \overset{PEG}{\rightsquigarrow} \langle t_1\ t_2, z\rangle}\ \text{(SEC)} \qquad \frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle t_1, y\rangle \quad P[e_2]\ y \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}{P[e_1 e_2]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}\ \text{(SEC2)}$$

$$\frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}{P[e_1 e_2]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}\ \text{(SEC3)} \qquad \frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle t, y\rangle}{P[!e_1]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}\ \text{(NOT)} \qquad \frac{P[e_1]x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, \bullet\rangle}{P[!e_1]\ x \overset{PEG}{\rightsquigarrow} \langle \varepsilon, x\rangle}\ \text{(NOT2)}$$

図 5 PEG の操作的意味論
Fig. 5 Operational Semantics of PEG.

e	::=	ε	empty
	|	a	character
	|	.	any character
	|	A	nonterminal
	|	$e\ e$	sequence
	|	$e\ /\ e$	ordered choice
	|	$e\ \vert\ e$	unordered choice
	|	$e*$	zero-or-more repetition
	|	$!e$	not-predicate

図 6 一般化解析表現の形式的定義
Fig. 6 Syntax of a generalized parsing expression.

4.1 文法

GPEG は PEG の定義 [5] の拡張として形式化する．多くの文法構成は PEG からきているが，本論文の重要な拡張として優先度なし選択があり，| で表す．

GPEG は以下の文法の組として以下のように定義する．

Definition 4.1 (GPEG). *GPEG は 4 つの組* $G = (N, \Sigma, R, e_s)$ として定義する．ここで，N は非終端記号の有限集合，Σ は終端記号の有限集合，R は生成規則の有限集合，e_s は開始記号である．

生成規則は非終端記号 $A \in N$ から解析表現 e への写像であり，$A \leftarrow e$ と表記される．また，$R(A)$ は $A \leftarrow e$ で関連付けられる e を表現している．

図 6 に GPEG の一般化解析表現を示す．以降では説明のためにメタ変数として $a, b, c \in \Sigma$, $A, B, C \in N$, $x, y, z \in \Sigma^*$ を使用する．

GPEG での演算子は PEG の演算子を引き継いでいる．PEG の演算子の動作は 3.1 節を参照のこと．本論文では PEG の演算子に優先度なし選択 $e_1 \mid e_2$ を拡張する．この演算子は通常の正規表現の選択と同じように振る舞う．つまり，優先度なし選択は e_1 と e_2 の両方を試みる．もし，e_1 と e_2 の両方が成功した場合，その後は 2 通りの可能性で続けるため非決定的な振舞いとなる．

優先度なし選択の優先順位はすべての演算子の中で一番低い．たとえば，$e_1 / e_2 \mid e_3$ は $(e_1 / e_2) \mid e_3$ に等しい．

また，優先度付き選択は優先度なし選択を用いて，以下の糖衣構文で表現できる．

$$e_1 / e_2 \quad = \quad e_1 \mid !e_1 e_2 \quad \text{ordered choice}$$

4.2 Parse Forest

GPEG パーサは優先度なし選択によって複数の異なる結果となるため複数の異なる解析木が導出される．しかし，複数の独立した解析木は実用的でないため，GPEG パーサは複数の解析木をまとめた 1 つの曖昧な解析木を導出する．この曖昧な解析木を Parse Forest と呼ぶ．Parse Forest は複数の導出された Parse Forest にラベル付けをしたものとして以下のように定義する．

t	::=	ε	empty
	|	x	string
	|	$t\ t$	concatenation
	|	$[A\ t]$	labeled forest
	|	$[\wedge\ t]$	ambiguious forest

特別なラベル $\wedge$ はいくつかの曖昧な Parse Forest をまとめていることを示している．このラベルのない Parse Forest は PEG の決定的な解析木に一致する．

本論文の Parse Forest は GLR や GLL の Shared Packed Parse Forest（SPPF）[13], [14] とは異なり，解析木の定義に曖昧な解析木をまとめた *ambiguious forest* をあらかじめ導入している．我々は曖昧な解析木を含めた定義にすべきであり，共有は効率的な構築法として分けるべきであると考えている．さらに，GPEG の Parse Forest は優先度なし選択と優先度付き選択を使い分けることで依然として文法から曖昧さを制御可能であるという点でも異なる

図 2 の曖昧な解析木は**図 7** として表現できる．

4.3 操作的意味論

Medeiros らは自然意味論のフレームワークによって正規表現と PEG を形式化した [11]．本節では，彼らの研究

© 2019 Information Processing Society of Japan

$$P[\varepsilon]\ x \overset{GPEG}{\rightsquigarrow} \{\langle \varepsilon, x\rangle\}\ \text{(EMPTY)} \qquad P[a]\ ax \overset{GPEG}{\rightsquigarrow} \{\langle a, x\rangle\}\ \text{(CHAR)} \qquad P[a]\ bx \overset{GPEG}{\rightsquigarrow} \emptyset \quad (b \neq a)\ \text{(CHAR2)}$$

$$\frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} S_1 \quad S_1 \neq \emptyset}{P[e_1\ /\ e_2]\ x \overset{GPEG}{\rightsquigarrow} S_1}\ \text{(CHOICE)} \qquad \frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} \emptyset \quad P[e_2]x \overset{GPEG}{\rightsquigarrow} S_2}{P[e_1\ /\ e_2]\ x \overset{GPEG}{\rightsquigarrow} S_2}\ \text{(CHOICE2)}$$

$$\frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} S_1 \quad P[e_2]x \overset{GPEG}{\rightsquigarrow} S_2}{P[e_1 \mid e_2]\ x \overset{GPEG}{\rightsquigarrow} S_1 \uplus S_2}\ \text{(ALT)}$$

$$\frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} S_1 \quad S_1 \neq \emptyset \quad \forall \langle t_{1y}, y\rangle \in S_1 . P[e_2]y \overset{GPEG}{\rightsquigarrow} S_{2y}}{P[e_1 e_2]\ x \overset{GPEG}{\rightsquigarrow} \biguplus_{y \in Y(S_1)} \{\langle t_{1y}\ t_{2y}, z\rangle \mid \langle t_{2y}, z\rangle \in S_{2y}\}}\ \text{(SEC)}$$

$$\frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} \emptyset}{P[e_1 e_2]\ x \overset{GPEG}{\rightsquigarrow} \emptyset}\ \text{(SEC2)} \qquad \frac{R(A) = e \quad P[e]\ x \overset{GPEG}{\rightsquigarrow} S}{P[A]\ x \overset{GPEG}{\rightsquigarrow} \{\langle [A\ t], y\rangle \mid \langle t, y\rangle \in S\}}\ \text{(NT)}$$

$$\frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} S_1 \quad S_1 \neq \emptyset}{P[!e_1]\ x \overset{GPEG}{\rightsquigarrow} \emptyset}\ \text{(NOT)} \qquad \frac{P[e_1]x \overset{GPEG}{\rightsquigarrow} \emptyset}{P[!e_1]\ x \overset{GPEG}{\rightsquigarrow} \{\langle \varepsilon, x\rangle\}}\ \text{(NOT2)}$$

図 8　GPEG の操作的意味論

Fig. 8　Operational Semantics of GPEG.

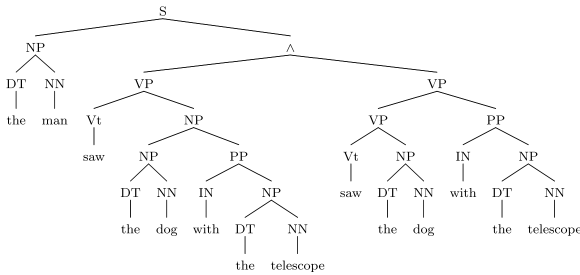

図 7　Parse Forest

Fig. 7　Parse Forest.

をもとに GPEG の形式化を行う．

PEG パーサは決定的であるため，解析結果は 1 つとなる．一方で，GPEG パーサは解析結果が複数になるという意味で曖昧である．1 つの結果を $\langle t, y\rangle$，t は解析木，y は残り文字列としてこのような曖昧な結果を $S = \{\langle t_1, y_1\rangle, \langle t_2, y_2\rangle, \ldots\}$ と表現する．

本論文ではマッチする関係を $\overset{GPEG}{\rightsquigarrow}$ と表記する．マッチする関係を表現した $P[e]x \overset{GPEG}{\rightsquigarrow} S$ は $P[e]$ というパーサが入力文字列 x を解析したとき，解析結果が集合 S となると読むことができる．また，特別な結果である $\emptyset$ は解析に成功した結果がなく，すべての可能性において失敗していることを示す．つまり，$P[e]x \overset{GPEG}{\rightsquigarrow} \emptyset$ は $P[e]$ パーサによって入力文字列 x を解析したときに成功した結果がないという意味である．**図 8** に，GPEG の操作的意味論を示す．

Definition 4.2. $Y(S) = \{y | \langle t, y\rangle \in S\}$ は残り文字列の集合である．

Definition 4.3. *Parse Forest* を構築する $\uplus$ 演算子を以下のように定義する．

$$\begin{aligned} S_1 \uplus S_2 = \{&\langle [\wedge\ t_1\ t_2], y\rangle | \langle t_1, y\rangle \in S_1, \langle t_2, y\rangle \in S_2\} \\ &\bigsqcup \{\langle t_1, y_1\rangle | \langle t_1, y_1\rangle \in S_1, y_1 \notin Y(S_2)\} \\ &\bigsqcup \{\langle t_2, y_2\rangle | \langle t_2, y_2\rangle \in S_2, y_2 \notin Y(S_1)\} \end{aligned}$$

5. 構文解析アルゴリズム

本章では，Parse Forest の構築をともなう一般化構文解析アルゴリズムについて述べる．GPEG による構文解析は PEG の構文解析と同様に再帰下降構文解析となる．我々は PEG の構文解析法としてよく用いられる Packrat 構文解析法と同様にメモ化によって再計算を防ぐことが可能になると考えた．そのため，GPEG による効率的なアルゴリズムを一般化 Packrat 構文解析と呼ぶ．4 章の形式化を使用した擬似コードによってアルゴリズムを説明する．さらに，ヒープメモリを節約して Parse Forest を構築する方法についても提案する．

5.1 Parse Forest の構築

疑似コードによって Parse Forest の構築を述べる．

優先度なし選択のアルゴリズムを擬似コードによって表現したものを **Algorithm 1** に示す．

Algorithm 1 Unordered choice

Input: x, e_1, e_2
Output: $S_1 \uplus S_2$
　$S_1 \Leftarrow P[e_1]\ x$
　$S_2 \Leftarrow P[e_2]\ x$
　return $S_1 \uplus S_2$

もし入力文字列 x に $e_1 \mid e_2$ を適用したとき e_1 と e_2 の両方が成功するならば，それぞれの結果が 2 つの異なる形になってしまう．そこで，両方の解析結果を $\uplus$ 演算子によってまとめる．$\uplus$ 演算子は同じ残り文字列を持つ解析結果内の Parse Forest をすべてまとめることで *ambiguous forests* を構築する．一方で，もし入力文字列 x に $e_1 \mid e_2$ を適用したとき e_1 と e_2 のどちらか一方が成功しているなら，成功した結果を出力する．入力文字列 x に $e_1 \mid e_2$ を

© 2019 Information Processing Society of Japan

適用したときのアルゴリズムは，4.3 節の (ALT) に一致する．ここで，$S \uplus \emptyset = S$ と $\emptyset \uplus S = S$ が成り立つことに注意されたい．

Algorithm 2 は，連結のアルゴリズムを擬似コードによって表現したものである．

Algorithm 2 Sequence

Input: x, e_1, e_2
Output: S
　$S \Leftarrow \emptyset$
　$S_1 \Leftarrow P[e_1]\ x$
　for all $\langle t_{1y}, y\rangle \in S_1$ **do**
　　$S_{2y} \Leftarrow P[e_2]\ y$
　　$S' \Leftarrow \emptyset$
　　for all $\langle t_{2y}, z\rangle \in S_{2y}$ **do**
　　　$S' \Leftarrow S' \bigsqcup \{\langle t_{1y}\ t_{2y}, z\rangle\}$
　　end for
　　$S \Leftarrow S \uplus S'$
　end for
　return S

もし入力文字列 x に $e_1\ e_2$ を適用したときに e_1 が成功するならば，4.3 節の (SEC) に一致する．x に e_1 を適用した結果 S_1 のすべての要素に対して e_2 を適用したとき，その結果をすべて $\uplus$ 演算子によってまとめる．一方で，入力文字列 x に $e_1\ e_2$ を適用したときに e_1 が失敗するならば，4.3 節の (SEC2) に一致する．

連結のアルゴリズムは S_1 と S_{2y} の二重ループであることから，S_1 と S_{2y} の要素数がパーサ性能に影響する．$\uplus$ 演算子は残り文字列が同じである重複した要素をまとめているため，S の要素数を n を入力文字列長として $(n+1)$ 以下になるように制限する役割を果たしている．

5.2　メモ化

一般化 Packrat 構文解析は一般化構文解析にメモ化を組み合わせたものである．メモ化テーブルを $N \times \Sigma^*$ から解析結果 S を参照する族として以下のように形式的に定義する．

$$M ::= \{S_{(A,x)} | A \in N, x \in \Sigma^*\}$$

4.3 節の (NT) に対応する非終端記号のアルゴリズムを表した擬似コードを **Algorithm 3** に示す．

もしメモ化テーブル M に $P[A]x \overset{GPEG}{\leadsto} S_{(A,x)}$ となるような $S_{(A,x)}$ を含むなら，非終端記号の結果は $S_{(A,x)}$ となる．そうでなければ，A に関連付けられている e を入力文字列 x に適用し，その結果を A でラベル付けを行い M に追加する．これにより，一般化 Packrat 構文解析器は非終端記号と文字列の組合せに対して 1 回のみの計算しか行わない．

5.3　Shared Forests

文法が曖昧であれば曖昧であるほど，Parse Forest は大きく構築される．さらに悪いことに，メモ化は巨大な Parse Forest をヒープメモリに保持する必要がある．その結果，メモリアロケーションやガベージコレクションによるオーバヘッドが大きくなってしまう．

Algorithm 3 Nonterminal

Input: $A \in N, x, M$
Output: S
　$S \Leftarrow \emptyset$
　if $S_{(A,x)} \in M$ **then**
　　$S \Leftarrow S_{(A,x)}$
　else
　　$e \Leftarrow R(A)$
　　$S_A \Leftarrow P[e]\ x$
　　for all $\langle t, y\rangle \in S_A$ **do**
　　　$S \Leftarrow S \bigsqcup \{\langle [A\ t], y\rangle\}$
　　end for
　　$S_{(A,x)} \Leftarrow S$
　　$M \Leftarrow M \cup S_{(A,x)}$
　end if
　return S

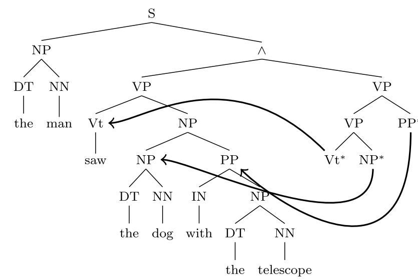

図 9　Shared Forest
Fig. 9　Shared Forest.

性能を改善するために，本研究では関数型データ構造によってヒープメモリの削減を行うことを提案する．関数型データ構造は，破壊的代入を制限された関数型プログラミングにおいて設計されたデータ構造である．データ構造の永続性を利用し，データ構造を再利用することで命令型データ構造と同じ計算時間を実現可能にする手法である．一般化 Packrat 構文解析は解析中に Parse Forest の変更を行わないため，関数型データ構造を適用可能である．さらに，Parse Forest は図 7 のように多くの共通した部分木を持つ．つまり，同じ部分木を再利用することでヒープメモリを節約することができる．この部分木を共有した Parse Forest を Shared Forest と呼ぶ．

図 7 の共通の部分木を共有した Shared Forest を**図 9** に示す．図中の矢印は部分木への参照を表現しており，同じ部分木の構築は参照ポインタを複製するのみでよい．

6.　測定結果

5 章のアルゴリズムを評価するために 2 種類の測定を行った．1 つは，Shared Forest による性能改善の評価測定．もう一方は，曖昧さを制御した文法に対してのメモ化

© 2019 Information Processing Society of Japan

による性能改善の評価測定である．

本研究では一般化 Packrat 構文解析器をパーサコンビネータとしてプログラミング言語 Rust で実装した．Rust のコンパイラのバージョンは stable の 1.26.2 であり，**-release** オプションを付与してコンパイルした．測定環境は octal-core の Intel 製 Core i7 3.4GHz CPU, RAM 7.7GB,Ubuntu OS 14.04 である．

6.1　Shared Forests の効果測定

Shared Forest による改善を評価するために，巨大な Parse Forest を構築する非常に曖昧な文法を使用する．しかしながら，以下の GLR や GLL パーサを評価するために使われる文法には左再帰が含まれている．

$$S \leftarrow SSS \mid SS \mid b$$

この文法では左再帰が禁止されている PEG をもとにした GPEG では表現できないため，以下のような左再帰を除去した文法を使用する．

$$S \leftarrow S'SS \mid S' \qquad S' \leftarrow bS \mid b$$

比較のために，Shared Forest と Non-shared Forest の 2 種類の Parse Forest を構築する実装を用意した．動的解析ツール Valgrind のヒーププロファイラ Massif [*1] を利用し，2 種類の実装の入力文字列長に対する最大ヒープサイズを測定した．**図 10** に測定結果を示す．図から明らかなように共有を行うことでヒープメモリを大きく削減できていることが分かる．

さらに，非常にあいまいな文法における文字列長に対する時間計算量についても 2 種類の実装で測定を行った．それぞれのテストは 5 回行い，その中央値を報告する．測定結果を**図 11** に示す．図から明らかなようにヒープメモリの効率化によって時間計算量は大幅に改善されている．

GLL の実用に向けたパーサフレームワークに Iguana [2] がある．測定環境や構築される解析木が異なるが，同様の文法で測定を行っている文献値では 400 [文字] で 3.6 [sec] 程度である．このことから，Generalized Packrat Parser は非常に曖昧な文法において実用的な GLL パーサに匹敵するといえる．

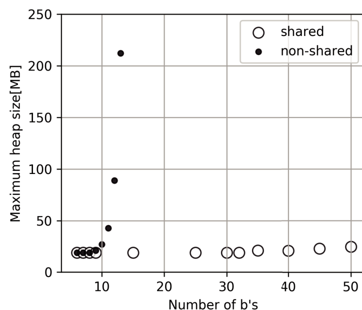

図 10　最大ヒープサイズ

Fig. 10　Maximum heap size.

*1　Massif: http://valgrind.org/docs/manual/ms-manual.html

6.2　曖昧さを制御した文法での測定

GPEG の強みは文脈によって優先度なし選択と優先度有り選択を使い分けることで曖昧さを制御できることである．また，GPEG パーサは曖昧さが少ないほど解析速度が向上する．つまり，GPEG パーサの性能を向上させるために不要な優先度なし選択を優先度付き選択で置き換えることが望ましい．

メモ化の効果について評価するために，文法はバックトラックを含んでいる必要がある．本研究の測定では，バックトラックを多く含み，優先度なし選択と優先度付き選択を置き換えて曖昧さを制御した以下の文法を使用する．

$$\begin{array}{llll} \text{AMB1:} & S \leftarrow S'S \mid S' & \text{AMB2:} & S \leftarrow S'S \,/\, S' \\ & S' \leftarrow S''S' \mid S'' & & S' \leftarrow S''S' \mid S'' \\ & S'' \leftarrow bS' \mid b & & S'' \leftarrow bS' \mid b \end{array}$$

$$\begin{array}{llll} \text{AMB3:} & S \leftarrow S'S \mid S' & \text{DET:} & S \leftarrow S'S \,/\, S' \\ & S' \leftarrow S''S' \,/\, S'' & & S' \leftarrow S''S' \,/\, S'' \\ & S'' \leftarrow bS' \mid b & & S'' \leftarrow bS' \,/\, b \end{array}$$

これらの文法による一般化 Packrat 構文解析によって入力文字列長に対する時間計算量を測定した．それぞれのテストは 5 回行い，その中央値を報告する．測定結果を**図 12** に示す．図の **AMB1w/oM** と **DETw/oM** は **AMB1** と **DET** の文法においてメモ化なしでそれぞれ計測した結果

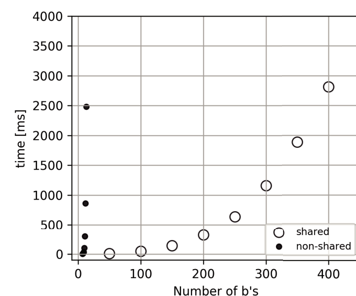

図 11　実行時間

Fig. 11　Execution time.

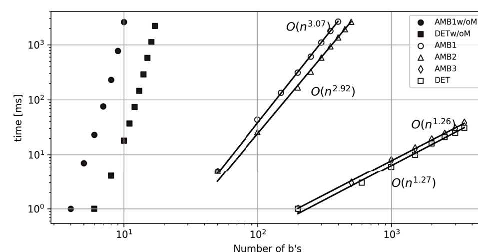

図 12　曖昧さが制御された文法での入力文字列長に対する実行時間

Fig. 12　Running the GPEG parsers on grammars controlled ambiguity.

© 2019 Information Processing Society of Japan

である．また，それぞれの結果に計算オーダと累乗近似直線を併記している．測定結果から，メモ化による性能の改善は明らかであり，n を入力文字列として **AMB1** と **AMB2** の文法では $O(n^3)$ 程度，**AMB3** と **DET** の文法では $O(n)$ 程度の計算量となっている．

7. 自然言語への応用例

本章では GPEG によって図 7 の解析木を構築する例について述べる．GPEG パーサへの厳密な自然言語の入力は "the man saw the dog with the telescope" であるが，簡単のため空白を取り除いた "themansawthedogwiththetelescope" とする．この入力文字列に対して図 7 の解析木を出力するような GPEG の記述には左再帰の除去が必要になる．しかし，左再帰を除去する際に必要な非終端記号を追加してしまうと，図 7 の解析木に付けられているラベルと異なるラベルがついてしまう．そこで，CPEG [17] のキャプチャ演算子を文法に追加することでラベルの制御を行った．以下に，入力文字列の構文解析に使用するキャプチャ演算子を追加した GPEG を示す．

$$
\begin{array}{lll}
S & \leftarrow & \{NP\ VP\ \#S\} \\
NP & \leftarrow & \{NP1\ PP\ \#NP\} \mid NP1 \\
NP1 & \leftarrow & \{DT\ NN\ \#NP\} \\
VP & \leftarrow & \{VP1\ PP\ \#VP\} \mid VP1 \\
VP1 & \leftarrow & \{Vt\ NP\ \#VP\} \\
PP & \leftarrow & \{IN\ NP\ \#PP\} \\
DT & \leftarrow & \{the\ \#DT\} \\
NN & \leftarrow & \{man\ /\ dog\ /\ telescope\ \#NN\} \\
Vt & \leftarrow & \{saw\ \#Vt\} \\
IN & \leftarrow & \{with\ \#IN\}
\end{array}
$$

4.2 節での Parse Forest の定義は非終端記号を labeled forest のラベルとしていたが，CPEG [17] のキャプチャ演算子 $\{e\ \#L\}$ によって L のラベルを付けるように宣言している．定義した GPEG による一般化 Packrat 構文解析器の生成をプログラミング言語 Python で実装を行った．一般化 Packrat 構文解析器によって入力文字列から変換された Parse Forest は以下の 2 種類が出力される．

- すべての文字列を消費した場合に構築される Parse Forest

```
[S [NP [DT the] [NN man]] [∧
[VP [VP [Vt saw] [NP [DT the] [NN dog]]]
    [PP [IN with] [NP [DT the]
    [NN telescope]]]]
[VP [Vt saw] [NP [NP [DT the] [NN dog]]
    [PP [IN with] [NP [DT the]
    [NN telescope]]]]]]]
```

- "themansawthedog" の文字列のみを消費した Parse Forest

```
[S [NP [DT the] [NN man]]
[VP [Vt saw] [NP [DT the] [NN dog]]]]
```

すべての文字列を消費した Parse Forest は図 7 の解析木と一致しており，2 種類の構文を解釈できるような文字列に対して曖昧さを残した解析木を出力することができている．

8. 関連研究

本研究は，言語構文のための認識ベース基盤である PEG [5] の拡張を提案するものである．PEG の文法は EBNF [16] に似ているが，重要な違いは優先度なし選択に対して優先度付き選択を採用することで曖昧さを含まないように文法が設計されている点である．PEG は決定的な特徴によって多くの実用的なパーサジェネレータが提案されてきた [6], [7], [8], [10].

しかしながら，近年の CFG の曖昧さを活かした応用 [1], [9], [15] に PEG は適していない．Afroozef らは拡張言語を埋め込んだ曖昧なプログラミング言語を解析する手法を提案している [1]．また，Leung らは曖昧な入出力例からの文法推論として Parsify を提案した [9]．これらは曖昧な文法を含むすべての CFG を扱うことができる GLL [12] を採用した応用例である．本研究では，4 章で優先度なし選択を導入した GPEG を形式化した．GPEG パーサは GLL と同様に曖昧さを扱うことが可能である．

GLL パーサは非常に曖昧な文法において入力文字列長 n に対し最悪時間計算量 $O(n^3)$ である．実用化のために Afroozef と Izmaylova は効率的な GLL の実装として Iguana [2] を提案した．このような GLL の研究に対して GPEG パーサの効率化は明らかに必要である．本研究では Packrat 構文解析 [4] の利点を継承した一般化 Packrat 構文解析を 5 章にて提案した．

9. 結論と課題

本研究では，PEG に優先度なし選択を拡張した GPEG を形式化した．GPEG の操作的意味論を Parse Forest の構築を含めて形式化し，効率的な構文解析アルゴリズムを一般化 Packrat 構文解析として提案した．測定結果では，曖昧さを含まない場合に線形時間で動作し，最悪時間計算量は立方時間であった．

今後の課題は，GPEG による構文解析の時間計算量や空間計算量を理論的に示すことである．また，実用化に向けた課題として出力された曖昧な解析木の曖昧さ除去がある．すでに GLR や GLL では曖昧さ除去の手法 [1], [9] が提案されているため，GPEG において既存手法が適用可能であるか検討していきたい．

参考文献

[1] Afroozeh, A., Bach, J.-C., Van Den Brand, M., Johnstone, A., Manders, M., Moreau, P.-E. and Scott,

© 2019 Information Processing Society of Japan

E.: Island Grammar-based Parsing using GLL and Tom, *SLE 2012 - 5th International Conference on Software Language Engineering*, pp.224–243, Springer (online), DOI: 10.1007/978-3-642-36089-3_13 (2012).

[2] Afroozeh, A. and Izmaylova, A.: Faster, Practical GLL Parsing, *Compiler Construction*, Franke, B. (Ed.), pp.89–108, Springer Berlin Heidelberg (2015).

[3] Chida, N. and Kuramitsu, K.: Parsing Expression Grammars with Unordered Choices, *Journal of Information Processing*, Vol.25, pp.975–982 (online), DOI: 10.2197/ipsjjip.25.975 (2017).

[4] Ford, B.: Packrat Parsing:: Simple, Powerful, Lazy, Linear Time, Functional Pearl, *Proc. 7th ACM SIGPLAN International Conference on Functional Programming*, *ICFP '02*, pp.36–47, ACM (online), DOI: 10.1145/581478.581483 (2002).

[5] Ford, B.: Parsing Expression Grammars: A Recognition-based Syntactic Foundation, *Proc. 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, *POPL '04*, pp.111–122, ACM (online), DOI: 10.1145/964001.964011 (2004).

[6] Grimm, R.: Better Extensibility Through Modular Syntax, *Proc. 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, *PLDI '06*, pp.38–51, ACM (online), DOI: 10.1145/1133981.1133987 (2006).

[7] Hirsch, C. and Frey, D.: Parsing Expression Grammar Template Library (2014), available from ⟨https://code.google.com/p/pegtl/⟩.

[8] Kuramitsu, K.: Nez: Practical Open Grammar Language, *Proc. 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, *Onward! 2016*, pp.29–42, ACM (online), DOI: 10.1145/2986012.2986019 (2016).

[9] Leung, A., Sarracino, J. and Lerner, S.: Interactive Parser Synthesis by Example, *Proc. 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, *PLDI '15*, pp.565–574, ACM (online), DOI: 10.1145/2737924.2738002 (2015).

[10] Majda, D.: PEG.js - Parser Generator for JavaScript (2015).

[11] Medeiros, S., Mascarenhas, F. and Ierusalimschy, R.: From Regexes to Parsing Expression Grammars, *Sci. Comput. Program.*, Vol.93, pp.3–18 (online), DOI: 10.1016/j.scico.2012.11.006 (2014).

[12] Scott, E. and Johnstone, A.: GLL Parsing, *Electron. Notes Theor. Comput. Sci.*, Vol.253, No.7, pp.177–189 (online), DOI: 10.1016/j.entcs.2010.08.041 (2010).

[13] Scott, E. and Johnstone, A.: GLL parse-tree generation, *Science of Computer Programming*, Vol.78, No.10, pp.1828–1844 (online), DOI: https://doi.org/10.1016/j.scico.2012.03.005 (2013).

[14] Tomita, M.: An Efficient Context-free Parsing Algorithm for Natural Languages, *Proc. 9th International Joint Conference on Artificial Intelligence - Volume 2*, *IJCAI'85*, pp.756–764, Morgan Kaufmann Publishers Inc. (online), available from ⟨http://dl.acm.org/citation.cfm?id=1623611.1623625⟩ (1985).

[15] van den Brand, M., Heering, J., Klint, P. and Olivier, P.A.: Compiling Language Definitions: The ASF+SDF Compiler, *CoRR*, Vol.cs.PL/0007008 (2000) (online), available from ⟨http://arxiv.org/abs/cs.PL/0007008⟩.

[16] Wirth, N.: Extended backus-naur form (EBNF), *ISO/IEC*, Vol.14977, p.2996 (1996).

[17] Yamaguchi, D. and Kuramitsu, K.: CPEG: A Typed Tree Construction from Parsing Expression Grammars with Regex-Like Captures, *Proc. 34th Annual ACM Symposium on Applied Computing*, *SAC '19*, ACM (online), DOI: 10.1145/3297280.3297433 (2019).

多田 拓

2018 年横浜国立大学理工学部卒業．2018 年 12 月現在同大学大学院理工学府数物・電子情報系理工学専攻修士課程在籍．主に構文解析器の研究開発に従事している．

倉光 君郎　（正会員）

2000 年東京大学大学院理学系研究科情報科学専攻博士課程中途退学．同年東京大学大学院情報学環助手．2007 年横浜国立大学工学部准教授．2018 年より日本女子大学理学部教授．プログラミング言語 Konoha の研究開発，構文解析器の研究開発に従事する．博士（理学），2008 年山下研究記念賞受賞．日本ソフトウェア科学会，ACM 各会員．

© 2019 Information Processing Society of Japan