解析表現文法の結合優先度を保持した左再帰除去手法の提案

高林 佳稀　倉光 君郎

形式文法の 1 つである PEG(Parsing Expression Grammar) は曖昧性が無く，記述された文法は再帰下降構文解析で実装することができる．そのため，プログラミング言語の解析などに向いている．しかしながら，再帰を用いて解析するため単純な実装では左再帰で記述された文法を扱う事ができない．左再帰を扱うために, 文法変換によって左再帰を除去する方法や左再帰を扱うための解析アルゴリズムの提案がなされているが，生成される木の構造が変化してしまうという問題点がある．本論文では，文法変換によって結合優先度を保持したまま左再帰を除去する手法についての提案を行う．

Parsing Expression Grammars(PEGs) are specifications of unambiguous recursive-descent style parsers. Therefore,PEGs are used as parsers for programming languages. However, recursive-descent parser can't parse left-recursive grammars. Some approaches to handle left-recursive grammars by transforming grammars or improving parsing algorithm are proposed. They have a problem to change the structure of generated tree. We propose an approach to handle left-recursive grammars while maintaining left-associativity by transforming grammar.

1 はじめに

構文解析で用いられる形式文法の一つに PEG(Parsing Expression Grammar)[1] がある．PEG の解析は決定的であり解析結果に曖昧性が生じないため，プログラミング言語の解析に向いている．PEG は CFG(Context Free Grammar) の広範囲を解析可能である．加えて PEG を用いた解析では，再帰下降構文解析とメモ化を組み合わせた Packrat Parsing を用いる事で入力長に対して線形時間で解析が可能である．

しかし、再帰下降構文解析は左再帰を含む文法を直接解析する事が困難である．そのため，左再帰を扱うために文法変換や解析アルゴリズムの改良についての提案がされている．既存の手法では，左再帰を含む文法を解析することは可能であるが，生成される木の構造が変化してしまうという問題点がある．特に左再帰の文法を右再帰の文法に変換することで左結合である文法が右結合に変化する．

我々は，PEG のパーサライブラリの一つである Nez[2] を対象に左再帰を解析する手法の提案を行う．PEG の文法変換と Nez の Left-folding を用いることで，木を伸びる方向を任意に指定することができ，結合性を保持した解析ができる．そのため，より自然な文法の記述が可能となる．

本論文の構成は以下の通りである．まず，第 2 節では背景として使用する文法である PEG とパーサライブラリ Nez についての説明と，動機として左再帰の問題点を挙げる．第 3 節では，提案手法である左再帰を除去しつつ結合性を維持した文法の変換について述べる．第 5 節では，提案手法の実装について述べる．第 5 節では，関連研究について述べる．第 6 節で，本論文の結論を述べる．

A Proposal of Left Recursive Eliminating Method Preserving Binding Priority for PEGs

Yoshiki Takabayashi, Kimio Kuramitsu, 横浜国立大学, Dept. of Electronic and Computer Engineering, Yokohama National University.

表 1　PEG の解析表現

PEG	Proc.	Description
”s”	5	Literal string
$[s]$	5	Character class
.	5	Any character
A	5	Non-terminal
$e*$	4	Zero-or-more
$e+$	4	One-or-more
$e!$	3	Not-predicate
e_1e_2	2	Sequence
e_1/e_2	1	Prioritized Choice

2　背景と動機

2.1　Parsing Expression Grammar

PEG は Ford により提案された形式文法である．PEG は BNF(Backus-Naur form) とよく似た構文を持つ．形式的に PEG の文法は，4 つ組 (V_N, V_T, R, e_s) で定義される．それぞれ，V_N は非終端記号の有限集合，V_T 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 $A(\in V_N)$ から解析表現 e への写像として定義され，$A \leftarrow e$ という形式で表記する．

表 1 に PEG で用いる解析表現を示す．文字列リテラル ”abc” は同じ入力とマッチし，文字クラス $[abc]$ はその中のどれか 1 文字とマッチする．. は任意の 1 文字とマッチする．成功した場合マッチした文字数分だけ解析位置を進める．$e*$ と $e+$ は繰り返しであり，解析は貪欲に行うため最長の位置でマッチする．連接 e_1e_2 は，e_1 の解析を行いマッチに成功したら解析位置を進め e_2 を解析する．優先度付き選択 e_1/e_2 は，最初に e_1 を試し，失敗した場合はバックトラックし e_1 を解析する前まで解析位置を戻した後に e_2 の解析を行う．否定先読み $!e$ は e のマッチが成功したら失敗とするが，解析した部分の文字消費は行われない．

PEG は字句解析と構文解析を同時に行う．PEG の解析アルゴリズムである Packrat Parsing は再帰下降構文解析とメモ化を組み合わせたものであり，入力文字列長に対して線形時間で解析可能である．

表 2　Nez で使用する拡張構文

AST	Proc.	Description
$\{e\}$	5	Constructor
$\$(e)$	5	Connector
$\#t$	5	Tagging
$\{\$e\}$	5	Left-folding

2.2　Nez

Nez は倉光らが開発した PEG パーサライブラリであり，文法変換器やパーサ生成器のフレームワークを提供している．また Nez では，構文木の構築やシンボルテーブルを用いた文脈依存な入力位置の解析を行うための記法を備えている．

表 2 に Nez で用いられる PEG の拡張構文を示す．$\{e\}$ は Constructor で，{} で囲まれた内部を木のノードに指定する．$\$(e)$ は Connector で，$() で囲まれた内部を子ノードとして指定する．$\#t$ は Tagging で，ノードの識別名を指定する．$\{\$\ e\}$ は Left-folding で，木の構築を再帰的に行う事が可能である．$e_1\{\$\ e_2\}$ は $\{\$(e_1)\ e_2\}$ は等価である，つまり Lef-folding の左側にある表現を子ノードの第一要素として木の構築を行う．

例として，”a”$\{\$\ \$(\text{”}b\text{”})\}*$ について考える．これは，a にマッチした後に 0 回以上の b にマッチする．マッチに成功した場合の木は図 1 のようになる．まず，最初にマッチした a と 1 つ目の b が葉となり木を形成する．以降は部分木と b をノードとして木を成長させていく．このように，Left-folding を用いてボトムアップ的に木を生成する事が可能となる．

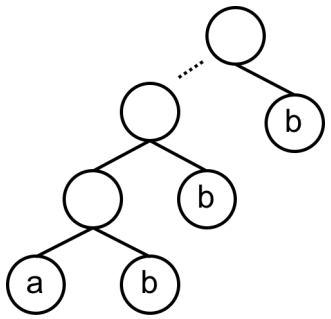

図 1　Left-folding を用いた木の生成

2.3 左再帰

PEG のパーサは再帰下降型であるため，左再帰を含む規則の解析が困難である．左再帰とは，非終端記号の生成規則がその非終端記号から始まるような規則の事である．左再帰の例を式 (1) に示す．

$$\begin{aligned} E &\leftarrow L\ \text{"}a\text{"} \\ &/\ \ \varepsilon \end{aligned} \tag{1}$$

式 (1) のような左再帰の式を解析しようとすると，入力を全く消費せずに再帰が繰り返し呼び出されるため，無限再帰の陥り失敗する．式 (1) は式 (2) のような繰り返しを用いた書き換えが可能である．

$$E \leftarrow \text{"}a\text{"}\ + \tag{2}$$

次に，左再帰と右再帰の両方を含む場合について考える．このようなパターンは主に中置記法の二項演算を解析する場合に多く用いられる．式 (3) に左再帰と右再帰を含む場合の例を示す．

$$\begin{aligned} E &\leftarrow E\ \text{"}+\text{"}\ E \\ &/\ \ E\ \text{"}*\text{"}\ E \\ &/\ \ T \end{aligned} \tag{3}$$

式 (3) は加算と乗算を組み合わせた式を解析する．PEG のパーサジェネレータの一つである Rats! [3] は，左再帰を繰り返しを用いた式に変換する．式 (4) に繰り返しを用いた場合の式を示す．

$$\begin{aligned} E &\leftarrow T\ \{\ \text{"}+\text{"}\ E\}\ + \\ &/\ \ T\ \{\ \text{"}*\text{"}\ E\ \}\ + \\ &/\ \ T \end{aligned} \tag{4}$$

式 (4) は式 (3) と同じ入力を解析可能であるが，構文木を生成する場合に自然な形で構文木が生成されないという問題点がある．また，Rats!の場合だと式 (3) から式 (4) のような変換を手動で行う必要がある．

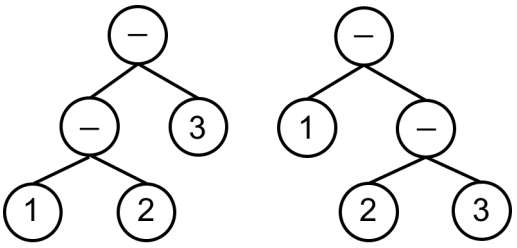

図 2　左結合の木と右結合の木

2.4 左結合性

左再帰が解析できない問題を文法変換によって解決しようとすると，結合性が変化するという問題点がある．左再帰を文法変換によって解決する場合，左再帰を無くす必要がある．この文法変換で受理する言語は変化しないが，木を生成する際に問題が生じる．

例として減算の式を用いて説明する．式 (5) と式 (6) に左再帰を用いた場合と用いない場合の減算の式の文法を示す．

$$E_0 \leftarrow E\ \text{"}-\text{"}\ E\ /\ T \tag{5}$$

$$E_1 \leftarrow T\ \text{"}-\text{"}\ E\ /\ T \tag{6}$$

E_0 と E_1 は受理する言語は同じである．しかし，E_1 の解析結果は右結合的になる．例えば，式 (6) で $1-2-3$ を解析し得られる木は図 2 の右の木の通りになる．右の木は右結合であるが，右の木を評価すると $1-2-3$ の計算結果と異なる．つまり，減算においては図 2 の左の木のような左結合である方が自然に処理を行うことができる．文法変換により左再帰が無く右再帰だけになったため右結合となった．つまり，元の式である式 (5) では左再帰と右再帰の両方があったため，ノードの左右両方に広がる自由があったが，文法変換後の式 (6) ではノードの左の子は終端記号のみであるため，左側に広がる自由が無くなっている．

また，ベキ乗演算子 ^ などの右結合である演算子も存在するため，結合性の暗黙な変換は文法設計者の意図しない結果を生じる可能性がある．

3 提案手法

そこで本論文では，任意の結合を保持できるような，Nez 向けの左再帰除去手法についての機能の提案を行う．また，本論文ではプログラミング言語で多く使われている二項演算子に使用するような左再帰の文法に関する変換について述べる．左結合の場合と右結合の場合について分けて変換手法について考える．

3.1 右結合である左再帰の除去

まず，右結合である場合の左再帰除去について述べる．式 (7) に左再帰を含む文法の一般式を示す．E_1 は再帰となる非終端記号であり，$\alpha_1, \cdots, \alpha_n$ と T はそれぞれ再帰を含まない非終端記号である．

$$E_1 \leftarrow E_1\alpha_1E_1/\cdots/E_1\alpha_nE_1/T \tag{7}$$

提案手法では，優先度付き選択内の左再帰を除去する．まず，式 (7) の 1 番目の優先度付き選択の左再帰を除去する．左再帰を除去した結果を式 (8) に示す．

$$\begin{aligned} E_1 &\leftarrow E_2\alpha_1E_1/E_2 \\ E_2 &\leftarrow E_2\alpha_2E_2/\cdots/E_2\alpha_nE_2/T \end{aligned} \tag{8}$$

まず，対象となる優先度付き選択以外をまとめた非終端記号を生成する．次に，対象の選択の左再帰を新しく生成した非終端記号に置き換る．そして，元の式の 2 番目以降の優先度付き選択を新しく生成した非終端記号にする．以上のステップが右結合に対する左再帰除去の流れである．式 (7) のように左再帰が複数の優先度付き選択の中に存在する場合は，新しく生成した非終端記号に関しても同様の変換を行う．最終的に，式 (7) は式 (9) のような形になる．

$$\begin{aligned} E_1 &\leftarrow E_2\alpha_1E_1/E_2 \\ E_2 &\leftarrow E_3\alpha_2E_2/E_3 \\ &\vdots \\ E_n &\leftarrow T\alpha_nE_n/T \end{aligned} \tag{9}$$

具体例を用いて述べる．2.3 節で用いた式 (3) を提案手法により変換すると式 (10) の通りになる．

$$\begin{aligned} E_1 &\leftarrow E_2\ \text{”}+\text{”}\ E_1/E_2 \\ E_2 &\leftarrow T\ \text{”}*\text{”}\ E_2/T \end{aligned} \tag{10}$$

式 (10) は左再帰を含まないため，解析時に無限再帰に陥らずに正しく式を解析し，木を生成する．また，演算子の優先度についても，元の式を維持している．また，木の構造は右結合であるが，加算と乗算のみを用いた式では正しい計算結果が得られるような木を生成する．

3.2 左結合である左再帰の除去

次に，左結合である左再帰の除去について述べる．左結合にするため，Nez の Left-folding を用いた式に変換する．まず，3.1 節で述べた左再帰の除去手法を使って左再帰を除去した後に，Left-folding の形へ変換する．

まず，左再帰を除去した一般式を式 (11) に示す．

$$E \leftarrow T_1\alpha E/T_2 \tag{11}$$

式 (11) は繰り返しを用いた式に変換が可能であるため，Left-folding を用いた式に変換が可能である．Left-folding と繰り返しを用いた式を式 (12) に示す．

$$E \leftarrow T_1\ \{\ \$\ \alpha\ \$(T_2)\}* \tag{12}$$

2.3 節の式 (6) を Left-folding を用いた形に変換すると式 (13) の通りになる．

$$E_1 \leftarrow T\ \{\ \$\ \text{”}-\text{”}\ \$(T)\}* \tag{13}$$

式 (13) は Left-folding を用いているため，マッチした部分からボトムアップ的に木を生成する．そのため，式 (13) で解析を行うと左結合の木が得られる．

```
E1 = { $(E1) '+' $(E1) #Add }
   / { $(E1) '-' $(E1) #Sub }
   / { $(E1) '*' $(E1) #Mul }
   / { $(E1) '/' $(E1) #Div }
   / T
T  = { [0-9]+ #Num }
```

図 3　Nez による四則演算の文法

```
E1 = { $(E2) '+' $(E1) #Add } / E2
E2 = { $(E3) '-' $(E2) #Sub } / E3
E3 = { $(E4) '*' $(E3) #Mul } / E4
E4 = { $(T)  '/' $(E4) #Div } / T
```

図 4　右結合である右再帰文法への変換

4　実装

本論文で提案した手法を Nez の機能として実装を行なった．実装した機能は，(1) 左再帰を含む文法を等価な右再帰へ変換する機能と (2) 右再帰を左結合である繰り返しの文法へ変換する機能の 2 種類である．

実装した機能について四則演算の文法を例に挙げ説明を行う．図 3 に Nez における四則演算の文法を示す．この文法は四則演算の入力を解析し，解析に成功したら二分木を出力する．それぞれ，根及び節が演算子で葉が自然数になるような木を生成する．

図 3 の文法は左再帰を含むため，Nez では直接では直接解析する事ができない．そのため，まず 3.1 節で述べた手法で左再帰の除去を行う．左再帰を除去した後の文法を図 4 に示す．

図 4 の文法は四則演算を示した文字列を正しくマッチする．しかしながら，右結合であるため正しい木を生成できない．この状態で，計算を行おうとすると式の後ろの方から評価を行い間違った計算結果を示す．四則演算は全て左結合であるため，Left-folding を用いた繰り返しの式に変換する必要がある．

左結合の繰り返しの式への変換後の文法を図 5 に示す．図 5 の文法は Left-folding を用いる事で左結合である木を生成するため．四則演算の式を正しく解析し，正しい木を出力するような文法が得られた．

```
E1 = E2 {$ '+' $(E2) #Add }*
E2 = E3 {$ '-' $(E3) #Sub }*
E3 = E4 {$ '*' $(E4) #Mul }*
E4 = T  {$ '/' $(T)  #Div }*
```

図 5　左結合である繰り返し文法への変換

5　関連研究

PEG の左再帰に関する研究について述べる．

Grimm は PEG を用いた Java 向けのパーサジェネレータである Rats! [3] の開発を行った．Rats!では様々な最適化オプションを含んでおり，生成するパーサのヒープ使用量の削減やスループットの向上といった最適化を行う事ができる．また，文法変換による左再帰に対応しており，文法の設計者が左再帰を含む文法を特定の形へ変換したものを記述する事で左再帰に対応することができる．

Medeiros らは左再帰を含む PEG の解析を行うために，セマンティクスの拡張を行った [4]．まず，左再帰を含む非終端記号に関して，再帰の回数に応じてマッチするような部分非終端記号を生成する．そして，小さい部分非終端記号から試していき成功した場合は，一つ大きい部分記号を試すようにセマンティクスを拡張した．

Frost らは，再帰下降構文解析において左再帰を含む文法を解析するアルゴリズムの提案を行なった [5]．Frost らの手法では左再帰が呼ばれた回数を記憶しておき，左再帰が呼ばれた回数が残りの入力文字列長を超えたときに失敗するように動作させる．このようにする事で，無限再帰に陥らずに解析を進める事ができる．

Warth らは Packrat 構文解析において左再帰を含む文法を解析するアルゴリズムの提案を行なった [6]．Warth らの手法では，左再帰を含む式を解析する際に最初にメモに失敗したという情報を格納しておく事で最初の左再帰を強制的に失敗させる．そして，呼び出し元でループを用いてボトムアップ的に構文解析を行う．

6 おわりに

本論文では，PEG のパーサライブラリである Nez を用いて左再帰を含む文法を解析する手法についての提案を行った．従来の左再帰に関する研究では，左結合性を保った解析が困難であった．しかし，本論文における提案では，文法の設計者の意図した通りに結合性をもたせて，左再帰を含む文法の解析を行う事ができる．そのため，より自然な文法の定義が可能となる．

今後は，性能評価実験を行う事で既存手法との比較を行い提案手法の有効性について示していく．また今回は，Nez の Left-folding を用いて実装を行なったが，一般の Packrat アルゴリズムを用いて左結合を維持する方法について検討を行う．

謝辞 本論文の初期の版について議論していただいた山口大輔氏 (横浜国立大学)，多田拓氏 (横浜国立大学)，千田忠賢氏 (NTT)，山口真弥氏 (NTT) に感謝する．

参 考 文 献

[1] Bryan Ford, Parsing Expression Grammars:A Recognition-Based Syntactic Foundation, *Symposium on Principles of Programming Languages*, 2004.

[2] Kimio Kuramitsu, Fast, Flexible, and Declarative Construction of AbstractSyntax Trees with PEGs, *Journal of Information Processing Vol.24 No.1*, Japan, IPSJ, 2016, pp. 123–131.

[3] Robert Grimm, Better Extensibility through Modular Syntax, *PLDI '06 Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Canada, ACM, 2006, pp. 38–51

[4] Sergio Medeiros, Fabio Mascarenhas, Roberto Ierusalimschy, Left Recursion in Parsing Expression Grammars, *SBLP 2012: Programming Languages*,Brazil, Brazilian Symposium on Programming Languages, 2012, pp. 27–41

[5] Richard A. Frost, Rahmatullah Hafiz, Paul C. Callaghan, Modular and efficient top-down parsing for ambiguous left-recursive grammars, *IWPT '07 Proceedings of the 10th International Conference on Parsing Technologies*, Prague, ACL, 2007, pp. 109–120

[6] Alessandro Warth, James R. Douglass, Todd Millstein, Packrat parsers can support left recursion, *PEPM '08 Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, USA, ACM, 2008, pp. 103–110