情報処理学会論文誌　プログラミング　Vol. 4　No. 2　104–115 (Mar. 2011)

同一入力位置で複数発生する左再帰へ対応したPackrat Parsingの設計と実装

白田佳章[†1]　木山真人[†1]　芦原評[†1]

構文解析法で Packrat Parsing という手法がある．Packrat Parsing は，バックトラックや無限先読みを用いた解析において，線型時間で解析可能である．また，Packrat Parsing は再帰下降構文解析法の一種であるため，左再帰を含む文法を解析できない．そこで従来は，左再帰を含む文法の解析を要求される際，左再帰部分を同一の入力を解析可能な右再帰へと変換し，左再帰を除去して解析を行う．だが，特定の左再帰は除去できないため，この手法で解析できない文法がある．Alessandro らは，Packrat Parsing において，左再帰を含む文法を，右再帰への文法の変換なしに解析可能にした．しかし，Alessandro らの実装では，同一の入力位置で左再帰が複数発生する文法において，特定の入力の解析に失敗するという問題点がある．そこで本論文では，左再帰が複数発生する文法において，解析できる手法を提案・実装し，評価を行った．

Design and Implementation of Packrat Parsing to Parse Grammars That Have Multiple Left Recursions at an Input Position

Yoshiaki Shirata,[†1] Masato Kiyama[†1] and Hyo Ashihara[†1]

There is Packrat Parsing among parsing methods. Packrat Parsing can parse backtracking and unlimited look-ahead in linear parse time. And Packrat Parsing is Recursive Descent Parsing. So, Packrat Parsing can't parse left recursive rules. Traditionally, we must remove left recursions by converting to right recursions to parse same inputs as original recursion when parse left recursive rules. There are grammars that can't be parsed with this method because some left recursions can't be converted to right recursions. Alessandro et al made possible to support left recursive rules without converting in Packrat Parsing. But the method can't parse some grammars that have multiple left recursions at an input position. This paper presents method to parse grammars that have multiple left recursions and implementation and evaluation of this method.

1. はじめに

Packrat Parsing [1] という構文解析手法がある．Packrat Parsing は，バックトラックや無限先読みを用いた解析において，線形時間で解析可能であるという利点がある[1]．一方，Packrat Parsing は再帰下降構文解析法の一種であるため，左再帰を含む文法に従って解析できないという欠点がある[3]．左再帰とは，文法において，ある非終端記号を置換する文字列の左端の記号がその非終端記号自身になる状態を指す．たとえば，以下の文法は左再帰を含む．

S ← S b a / a

ここで，非終端記号は大文字，終端記号は小文字で表現している．以降，同様の表現を用いる．また，上記の文法で用いている「/」という演算子は，Packrat Parsing で用いられる choice 演算子[6] である．

左再帰を含む文法に従って入力を通常の再帰下降構文解析法で解析すると，無限再帰に陥ってしまう．

左再帰は，直接左再帰と間接左再帰の 2 種類に分類される．直接左再帰は，上記の文法の例のように，非終端記号が直接その非終端記号自身を呼び出す左再帰である．従来は，直接左再帰を含む文法に従って入力を解析する際，左再帰部分を同一の入力を解析可能な右再帰へと変換し，左再帰を除去して解析を行う．Packrat Parsing でも同様である．上記の直接左再帰を含む文法の場合，Pappy [2] と Rats! [3] などの Packrat Parsing の実装では以下のように変換する．

S ← a (b a)*

この変換を用いると，直接左再帰を除去できるため入力を解析可能になるが，解析の結果得られる木の形が変わってしまうという問題点がある．

間接左再帰は，非終端記号が複数の非終端記号を経由し，間接的に自分自身を呼び出す左再帰である．以後，説明の簡略化のため，左再帰において，自分自身を呼び出す非終端記号を head rule，経由される非終端記号を involved rule と呼ぶ．間接左再帰を含む文法の例を以下に示す．

†1 熊本大学大学院自然科学研究科
Kumamoto Graduate School of Science and Technology, Kumamoto University

S ← A
A ← S b a / a

直接左再帰とは異なり，間接左再帰を含む文法に従って解析可能な Packrat パーサの実装はなかった．

Alessandro ら[4] は，Packrat Parsing において，左再帰を含む文法を，右再帰文法への変換なしに入力を解析可能にした．Alessandro らの実装では，直接左再帰を含む文法だけでなく，間接左再帰を含む文法を用いても解析可能である．

しかし，Alessandro らの実装では，同一の入力位置で head rule が異なる左再帰が発生する文法において，特定の入力が与えられたときに解析を失敗するという問題点がある．

そこで本論文では，同一の入力位置で head rule が異なる左再帰が発生する文法に従って入力を解析できる手法について提案し，評価を行う．2 章では，Alessandro らが提案した従来手法とその問題点について述べる．3 章では，従来手法の問題点に対応する提案手法について述べる．4 章では，提案手法の評価およびその考察を述べる．最後に，5 章で本論文のまとめを述べる．

2. 従来手法

本章では，Alessandro らが提案した従来手法とその問題点について述べる．従来手法を用いると，文法の変換なしに左再帰を含む文法を解析できる．

2.1 従来手法

まずは従来手法のアルゴリズムについて説明する．**図 1** の文法と入力を用いて考える．従来手法では，左再帰が発生したとき，呼び出された head rule の解析を保留し，バックトラックして次の選択肢に従って解析を進める．head rule から進めた解析から左再帰以外の解析結果を得るとメモテーブルに記録し，その解析結果を用いて保留にしておいた左再帰の解析を進める．このようにメモを利用し，左再帰の発生に対応する．

従来手法のアルゴリズムの擬似コードを文献 4) より再録し，簡潔に説明する．APPLY-RULE プロシージャを**図 2** に示す．APPLY-RULE プロシージャはルール適用に用いる．メモテーブルに解析結果を記録していない場合は図 2 の 04.〜15. を，ある場合は 16.〜20. をそれぞれ実行する．LRStack は左再帰の情報を得るために用いる．メモテーブルに記録していないルールを評価するたびに LR データをプッシュし，その情報を用いて左再帰の情報を得る．ルールの評価を終えるとそのルールの LR データをスタックからポップする．LR データのデータ型は以下のとおりである．

S ← A
A ← S b a / a
input: “aba”

図 1　文法と入力 1
Fig. 1　Grammar and input 1.

```
01.APPLY-RULE(R, P)
02.   let m = RECALL(R, P)
03.   if m = NIL
04.      then let lr = new LR(FAIL, R, NIL, LRStack)
05.           LRStack ← lr
06.           m ← new MEMOENTRY(R.body, P)
07.           MEMO(R, P) ← m
08.           let ans = EVAL(R.body)
09.           LRStack ← LRStack.next
10.           m.pos ← Pos
11.           if lr.head ≠ NIL
12.              then lr.seed ← ans
13.                   return LR-ANSWER(R, P, m)
14.              else  m.ans ← ans
15.                   return ans
16.      else  Pos ← m.pos
17.           if m.ans is LR
18.              then SETUP-LR(R, m.ans)
19.                   return m.ans.seed
20.              else  return m.ans
```

図 2　APPLY-RULE プロシージャ
Fig. 2　APPLY-RULE procedure.

LR: ($seed$: AST, $rule$: $RULE$, $head$: $HEAD$, $next$: LR)

RULE 型の rule は評価したルールの情報を保持する．HEAD 型の head は評価したルールが関連する左再帰の情報を保持する．HEAD のデータ型を以下に示す．

$HEAD$: ($rule$: $RULE$, $involvedSet$, $evalSet$: $SET\ OF\ RULE$)

rule は発生した左再帰の head rule を保持する．involvedSet は involved rule の集合を保持する．

SETUP-LR プロシージャを**図 3** に示す．SETUP-LR プロシージャは，発生している左再帰の情報を LRStack から得るためのプロシージャである．得た左再帰の情報は，メモテーブルの HEAD データに記録する．

```
01.SETUP-LR(R, L)
02.   if L.head = NIL
03.      then L.head ← new HEAD(R, {}, {})
04.   let s = LRStack
05.   while s.head ≠ L.head
06.      do s.head ← L.head
07.         L.head.involvedSet ← L.head.involvedSet ∪ {s.rule}
08.         s ← s.next
```

図 3　SETUP-LR プロシージャ
Fig. 3　SETUP-LR procedure.

```
01.LR-ANSWER(R, P, M)
02.   let h = M.ans.head
03.   if h.rule ≠ R
04.      then return M.ans.seed
05.      else  M.ans ← M.ans.seed
06.            if M.ans = FAIL
07.               then return FAIL
08.               else  return GROW-LR(R, P, M, h)
```

図 4　LR-ANSWER プロシージャ
Fig. 4　LR-ANSWER procedure.

LR-ANSWER プロシージャを**図 4** に示す．LR-ANSWER プロシージャは，発生した左再帰の解析を行うために呼び出す．解析保留中の左再帰を再び解析すると判断した場合，GROW-LR プロシージャを呼び出す．

GROW-LR プロシージャを**図 5** に示す．GROW-LR プロシージャは解析を保留していた左再帰の解析を再び行う際に呼び出す．HEADS は解析する左再帰の情報を持つ．HEADS のデータ型を以下に示す．

$$HEADS\colon POSITION \rightarrow HEAD$$

GROW-LR プロシージャでは，入力の読み込み位置が更新されるうちは何度でも評価を行う．できるだけ多くの入力トークンを読み込めるようにする．

RECALL プロシージャを**図 6** に示す．RECALL プロシージャは，解析結果をメモテーブルから取得するために呼び出す．

従来手法を用いて，図 1 の文法と入力を実際に解析する．解析を進めると，**図 7** (a) のように S を head rule とする間接左再帰が起こる．このまま左再帰の解析を続けると無限再帰

```
01.GROW-LR(R, P, M, H)
02.   HEADS(P) ← H
03.   while TRUE
04.      do
05.         Pos ← P
06.         H.evalSet ← COPY(H.involvedSet)
07.         let ans = EVAL(R.body)
08.         if ans = FAIL or Pos ≤ M.pos
09.            then break
10.         M.ans ← ans
11.         M.pos ← Pos
12.   HEADS(P) ← NIL
13.   Pos ← M.pos
14.   return M.ans
```

図 5　GROW-LR プロシージャ
Fig. 5　GROW-LR procedure.

```
01.RECALL(R, P)
02.   let m = MEMO(R, P)
03.   let h = HEADS(P)
04.   if h = NIL
05.      then return m
06.   if m = NIL and R ∉ {h.rule} ∪ h.involvedSet
07.      then return new MEMOENTRY(FAIL, P)
08.   if R ∈ h.evalSet
09.      then h.evalSet ← h.evalSet \ {R}
10.           let ans = EVAL(R.body)
11.           m.ans ← ans
12.           m.pos ← Pos
13.   return m
```

図 6　RECALL プロシージャ
Fig. 6　RECALL procedure.

になるため，左再帰の解析を保留する．このとき，左再帰の情報をメモテーブルに記録する．左再帰の情報がメモテーブルに記録されているときは，その左再帰の解析を保留していると見なす．バックトラックして次の選択肢を解析すると，図 7 (b) のように入力 1 トークン目の “a” を読み進める．このとき，非終端記号 S から解析を進めると “a” を読み進められると判明する．そして，保留していた S を head rule とする左再帰の解析が可能になる．よって，この情報をメモテーブルに記録し，GROW-LR プロシージャを呼び出して S の間

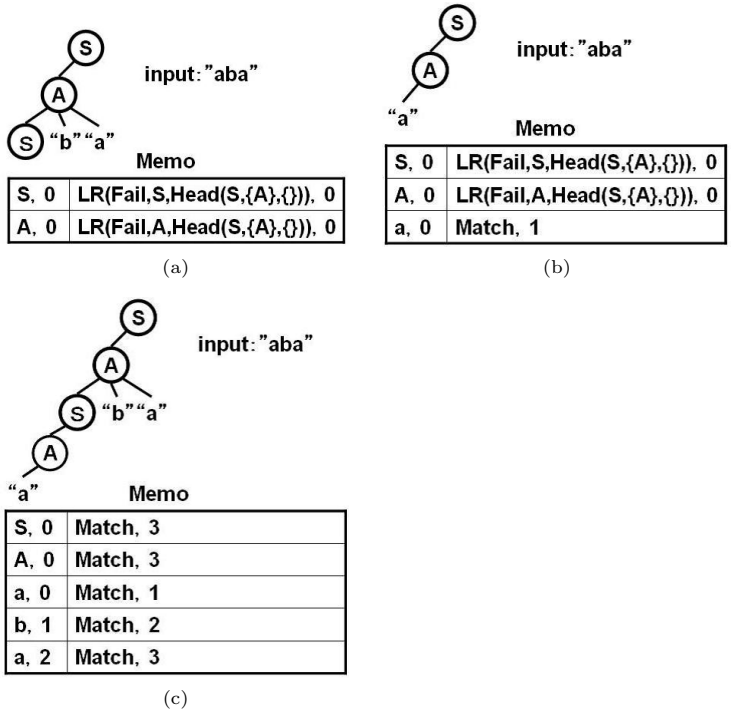

図 7　従来手法の解析成功例
Fig. 7　Case of success with traditional method.

接左再帰の解析を進める．メモテーブルの S の記録を用いて解析すると，図 7 (c) の木が得られる．GROW-LR プロシージャは入力の読み込み位置が更新されなくなるまでループして解析を続ける．そのため，正しい解析木が得られた後，もう一度 S を head rule とする左再帰の解析を行う．そして入力の読み込み位置が更新されないため，解析を終了する．このとき，入力をすべて読み込んでいるので解析成功となる．

2.2　従来手法の問題点

従来手法では，文法の変更なしに左再帰を含む文法に従った解析が可能になる．しかし，従来手法は，同一の入力位置で head rule が異なる左再帰が発生する文法に対応していないという問題点がある．

S ← A b
A ← A a / S a / a
input:“aab”

図 8　文法と入力 2
Fig. 8　Grammar and input 2.

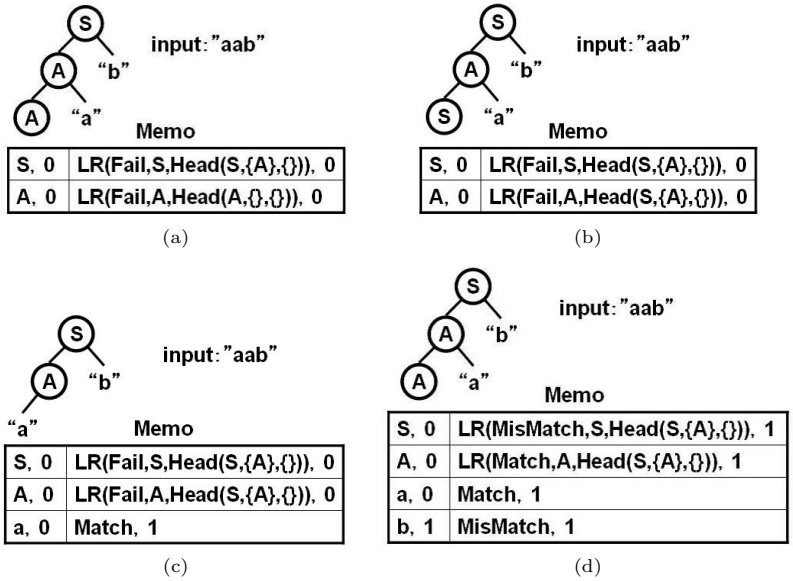

図 9　従来手法の解析失敗例
Fig. 9　Case of failure with traditional method.

例を用いて従来手法の問題点について述べる．従来手法を用いて**図 8** の文法と入力を解析する．解析を始めると，**図 9** (a) のように A が head rule である左再帰が起こる．このとき，左再帰の情報を A の解析結果のメモテーブルに記録する．そしてバックトラックし，次の選択肢に従って解析を進める．すると，図 9 (b) のように S が head rule である左再帰が起こる．そして S が head rule である左再帰の情報を S と A の解析結果のメモテーブルに記録する．このとき，S が head rule である間接左再帰の情報によって，A が head rule である直接左再帰の情報が上書きされる．つまり，A が head rule である左再帰の情報が失われる．その後，図 9 (c) まで解析を進めたとき，S の左再帰の解析が可能になる．そのた

め，GROW-LR プロシージャを呼び出して再度解析を行う．すると，図 9 (d) のように A の左再帰が再び起こる．このとき，A の左再帰が初めて起こったと見なし，SETUP-LR プロシージャを実行して再び A の左再帰の情報を得ようとする．しかしこのとき，LRStack は空となっている．空の LRStack から A の左再帰の情報を得ようとし，図 3 の 08. でエラーが発生して異常終了する．

3 章では，この問題に対応可能な提案手法について述べる．

3. 提 案 手 法

本章では，提案手法について述べる．まずは，Alessandro らが提案した従来手法の問題点に対応した提案手法について述べる．そして，その提案手法の問題点について述べ，さらに改良した提案手法について述べる．

従来手法の解析時に起きた異常終了は，同一の入力位置で head rule が異なる左再帰が発生する文法を用いた解析を想定していないため起こる．従来手法を用いて図 8 を解析したときに確認された問題点は以下の 2 点である．

(1)　左再帰の情報が異なる左再帰の情報によって上書きされる．

(2)　空の LRStack の情報を用いて左再帰の情報を得ようとする．

問題点 (1) を解決するため，左再帰の情報を head rule の非終端記号ごとに，HEADTABLE というメモテーブルに記録する．左再帰の情報が必要になったときはそのメモから取得する．また，問題点 (2) を解決するため，LRStack が空のときに SETUP-LR プロシージャを実行しない．

3.1 提 案 手 法

左再帰の情報が上書きされる，空の LRStack の情報を用いる，という 2 つの問題に対応する提案手法のアルゴリズムについて説明する．変更した行には * をつけて示す．

変更した APPLY-RULE プロシージャを**図 10** に示す．問題点 (1) を解決するために，左再帰情報を，非終端記号ごとに記録するように変更する．左再帰情報は新たに用意した HEADTABLE というメモテーブルに記録する．HEADTABLE のデータ型は以下のとおりである．

$$HEADTABLE\colon (RULE, POSITION) \rightarrow HEAD$$

また，問題点 (2) に対応するために，20. の条件文を満たすときのみ SETUP-LR プロシージャを実行するように変更する．さらに，解析木とメモテーブルに記録した読み込み位置の値が異なる場合があるため，10.，11. 行で修正している．

```
01.APPLY-RULE(R, P)
02.   let m = RECALL(R, P)
03.   if m = NIL
04.   * then let lr = new LR(FAIL, R, HEADTABLE(R, P), LRStack)
05.            LRStack ← lr
06.            m ← new MEMOENTRY(lr, P)
07.            MEMO(R, P) ← m
08.            let ans = EVAL(R.body)
09.            LRStack ← LRStack.next
10.          * if ans ≠ Match or m.pos ≤ Pos
11.            * then m.pos ← Pos
12.            if lr.head ≠ NIL
13.              then lr.seed ← ans
14.                   return LR-ANSWER(R, P, m)
15.            * else m.ans ← ans
17.                 return ans
18.      else  Pos ← m.pos
19.            if m.ans is LR
20.            * then if R = m.ans.rule or m.ans.head = NIL
21.                      then SETUP-LR(R, m.ans, P)
22.                   return m.ans.seed
23.              else return m.ans
```

図 10　提案手法の APPLY-RULE プロシージャ
Fig. 10　Proposal APPLY-RULE procedure.

```
01.LR-ANSWER(R, P, M)
02.* let h = HEADTABLE(R,  P)
03.   if h.rule ≠ R
04.      then return M.ans.seed
05.      else  M.ans ← M.ans.seed
06.            if M.ans = FAIL
07.               then return FAIL
08.               else  return GROW-LR(R, P, M, h)
```

図 11　提案手法の LR-ANSWER プロシージャ
Fig. 11　Proposal LR-ANSWER procedure.

次に，変更した LR-ANSWER プロシージャを**図 11** に示す．問題点 (1) に対応するために，h には HEADTABLE から得た値を与えるように変更する．

変更した RECALL プロシージャを**図 12** に示す．HEADS は，従来手法と異なりスタックを用いて HEAD を複数記録可能にしている．GROW-LR プロシージャによる左再帰解

```
01.RECALL(R, P)
02.   let m = MEMO(R, P)
03.   let hs = HEADS(P)
04.   if hs = NIL
05.      then return m
06.* if m = NIL and R ∉ {h.top.rule} ∪ hs.top.involvedSet
07.      then return new MEMOENTRY(FAIL, P)
08.* if R ∈ hs.top.evalSet
09.   * then hs.top.evalSet ← hs.top.evalSet \ {R}
10.           let ans = EVAL(R.body)
11.           m.ans ← ans
12.         * if ans ≠ Match or m.pos ≤ Pos
13.           * then m.pos ← Pos
14.   return m
```

図 12　提案手法の RECALL プロシージャ
Fig. 12　Proposal RECALL procedure.

析中に別の左再帰解析のための GROW-LR プロシージャ呼び出しが起きても対応可能となる．HEADS のデータ型は以下のとおりである．

$HEADS: POSITION \rightarrow Stack(HEAD)$

変更した RECALL プロシージャでも，解析木とメモテーブルに記録した読み込み位置の値が異なる場合があるため，12.，13. 行で修正している．

変更した SETUP-LR プロシージャを**図 13** に，変更した GROW-LR プロシージャを**図 14** にそれぞれ示す．変更した GROW-LR プロシージャでは，解析木とメモテーブルに記録した読み込み位置の値が異なる場合があるため，12.，13. 行で修正している．それ以外では，HEADTABLE への対応と問題点 (2) に対応するための条件文の追加についての変更のみである．

提案手法を用いて，図 8 の文法と入力を解析する．**図 15** (a) の木ができるまで解析を進める．A の左再帰が発生するので，SETUP-LR プロシージャを用いてメモテーブルと HEADTABLE に A の左再帰の情報を記録する．そして図 15 (b) まで解析を進める．S の左再帰が発生するので，メモテーブルと HEADTABLE に S の左再帰の情報を記録する．このとき，左再帰の情報を HEADTABLE に記録しているので，従来手法と異なり，A の直接左再帰の情報が失われない．その後，図 15 (c) まで解析したとき，A の左再帰の解析が可能になるので，GROW-LR プロシージャを実行する．そして，図 15 (d) の木ができ，GROW-LR プロシージャによる A の左再帰の解析を終える．GROW-LR プロシージャは

```
01.SETUP-LR(R, L, P)
02.   if L.head = NIL
03.   * then HEADTABLE(R, P) ← new HEAD(R, {}, {})
04.   let s = LRStack
05.   while s.head ≠ L.head
06.      do s.head ← L.head
07.         L.head.involvedSet ← L.head.involvedSet ∪ {s.rule}
08.         s ← s.next
09.* HEADTABLE(R, P) ← L.head
```

図 13　提案手法の SETUP-LR プロシージャ
Fig. 13　Proposal SETUP-LR procedure.

```
01.GROW-LR(R, P, M, H)
02.   HEADS(P).push(H)
03.   while TRUE
04.      do
05.         Pos ← P
06.         H.evalSet ← COPY(H.involvedSet)
07.       * HEADTABLE(R, P) ← H
08.         let ans = EVAL(R.body)
09.         if ans = FAIL or Pos ≤ M.pos
10.            then break
11.         M.ans ← ans
12.       * if ans ≠ Match or m.pos ≤ Pos
13.         * then M.pos ← Pos
14.* HEADS(P).pop
15.   Pos ← M.pos
16.   return M.ans
```

図 14　提案手法の GROW-LR プロシージャ
Fig. 14　Proposal GROW-LR procedure.

読み込んだトークン数の値が更新されなくなるまで何度も評価を行うため，GROW-LR プロシージャによる A の左再帰の解析を終えた後，GROW-LR プロシージャによる S の左再帰の解析を行う．そして図 15 (e) の解析結果を得て解析成功となる．このとき図 15 (e) のメモテーブルを観察すると，入力をすべて読み込んで解析には成功するが，メモテーブルの記録と解析木の形が一致していない．つまり，メモテーブルから解析木を再現できない．これは，同一の入力位置で head rule が異なる左再帰を含むため起こる．今回の解析の場合，図 15 (d) のようになるまで GROW-LR プロシージャを呼び出して A が head rule の左再帰を解析し，その後，保留していた S の解析中に (A,0) のメモテーブルにおいて，(Match,

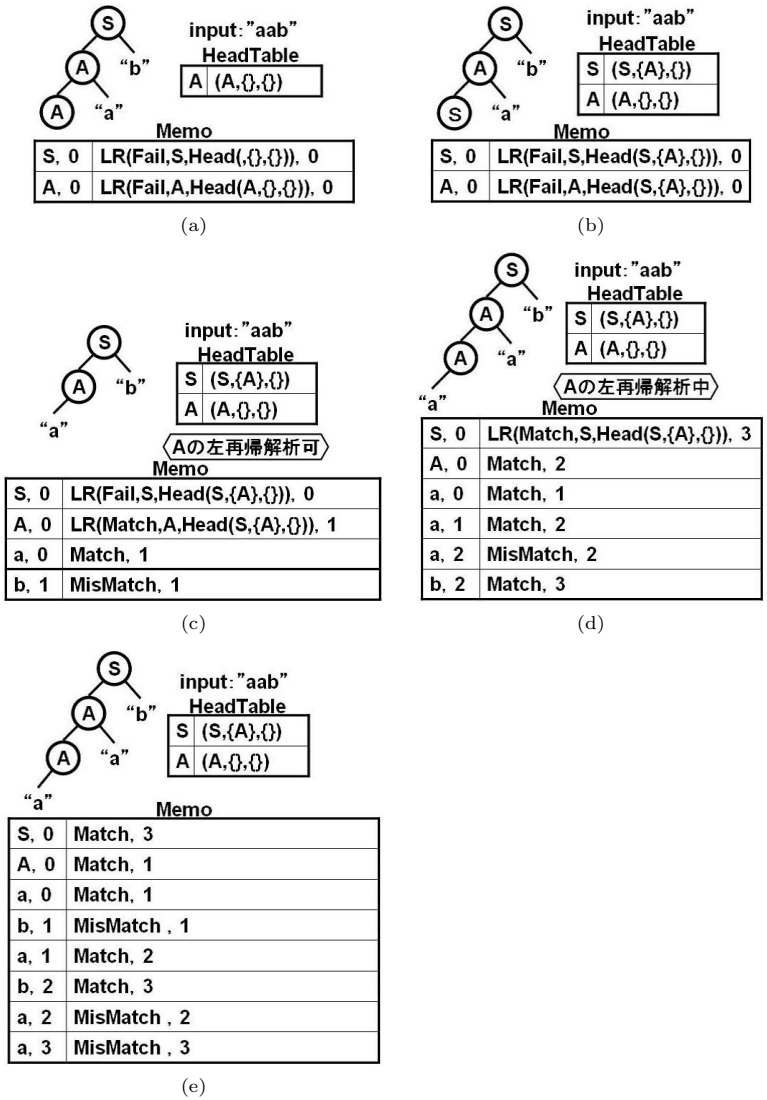

図 15　提案手法の解析成功例
Fig. 15　Case of success with proposal method.

S ← A b / b
A ← A a / S a
input:“baab”

図 16　文法と入力 3
Fig. 16　Grammar and input 3.

2) に (Match, 1) を上書きしてしまう．この問題点を解決するため，POSITION の値が減少するようなメモの更新は行わないようにする．

これを実現させるため，図 10 の 10.，図 6 の 12.，図 14 の 12.，では

M.pos ← Pos

を

* if ans ≠ Match or m.pos ≤ Pos
  * then M.pos ← Pos

に置き換えてある．

しかし，この提案手法を用いても解析できない文法がある．**図 16** の文法と入力を解析する．Alessandro らの従来手法を用いて解析すると，図 8 の文法と入力を用いた場合と同様，異常終了する．次に，上記の提案手法を用いて図 16 を解析する．解析を始めると，まず A が head rule である直接左再帰が発生する．解析を保留してバックトラックし，次に S が head rule である間接左再帰が発生する．また解析を保留してバックトラックすると**図 17** (a) の木ができる．入力を読み進めたので S の左再帰が解析可能となる．GROW-LR プロシージャを用いて解析を進める．すると図 17 (b) のように，A を head rule とする直接左再帰が再度発生する．このときも A から読み込み可能な終端記号が分からないので，A の解析を保留してバックトラックする．次に S がループする間接左再帰が発生し，図 17 (c) のようにメモを用いて入力を読み進める．そして入力を最後まで読み進められず，バックトラックして次の選択肢を読み進めた時点で図 17 (d) のようになり，解析に失敗して終了する．図 17 (c) では，非終端記号 A から読み進められる終端記号が新たに判明し，図 17 (b) で保留した A の解析を行えるようになっている．しかし，S からは入力を読み進めた位置が更新されていないため，A の解析を行わないまま解析を終えている．

3.2　改良した提案手法

上記の提案手法では図 16 の文法と入力の解析に失敗した．しかし，解析終了時，非終端記号 A から読み進められる終端記号が判明しているにもかかわらず，A を head rule とする左再帰の解析を保留し，解析せずに終了している．このとき，A の左再帰を解析すると

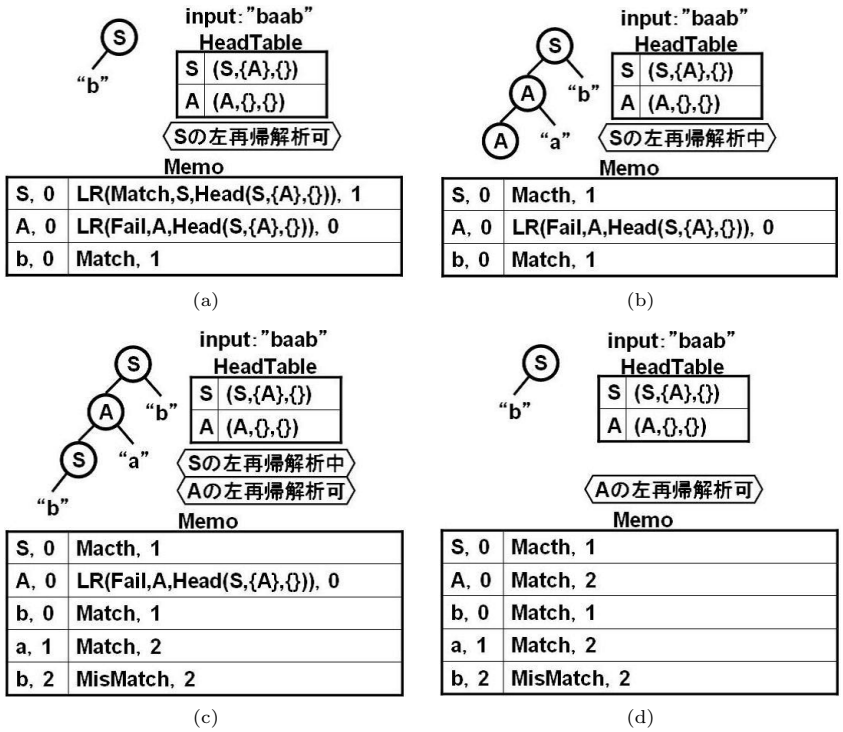

図 17　提案手法の解析失敗例
Fig. 17　Case of failure with proposal method.

入力をさらに読み進められる可能性がある．よって，GROW-LR プロシージャの実行を終える前に A の左再帰を解析しなければならない．これに対応した新しい手法を提案する．GROW-LR プロシージャ実行中に，新たに解析保留中の非終端記号を解析できるようになったとき，GROW-LR プロシージャのループを繰り返すようにする．**図 18** に新しい提案手法を加えた GROW-LR プロシージャのアルゴリズムを示す．** を付けた行が変更した箇所である．lrSet は，involvedSet の中で，読み進められる終端記号が判明していない非終端記号の集合である．lrSet は GROW-LR プロシージャのループ部分の開始時に更新する．そして EVAL 実行後に同様の集合を再度探し，lrSet と比較する．そして解析可能な非終端記号が増えていれば再度ループする．

```
01.GROW-LR(R, P, M, H)
02.* HEADS(P).push(H)
03.   while TRUE
04.     do
05.        Pos ← P
06.        H.evalSet ← COPY(H.involvedSet)
07.     ** let lrSet = H.involvedSet.filter(Memo(_ , p) is LR)
08.      * HEADTABLE(R, P) ← H
09.        let ans = EVAL(R.body)
10.        if ans = FAIL or Pos ≤ M.pos
11.       ** then if H.involvedSet.filter(Memo(_ , p) is LR) = lrSet
12.                then break
13.        M.ans ← ans
14.      * if ans ≠ Match or m.pos ≤ Pos
15.        * then M.pos ← Pos
16.   HEADS(P).pop
17.   Pos ← M.pos
18.   return M.ans
```

図 18　改良した提案手法の GROW-LR プロシージャ
Fig. 18　Improved proposal GROW-LR procedure.

新たな提案手法を用いて，図 16 の文法と入力を解析する．**図 19** (a) までは図 17 の場合と同様に解析する．ここで，A の左再帰は読み込める終端記号が分からず解析を進められないので，lrSet に A を記録する．そして S の左再帰の解析を進める．解析を進めると，図 19 (b) の木ができたときに，非終端記号 A からトークン “ba” を読み進められると分かる．このとき，lrSet の中の非終端記号 A から読み進められる終端記号が分かる．よって，図 19 (c) となるまで解析を進めたときに lrSet から A を消去し，再度 GROW-LR プロシージャのループを行う．そして図 19 (d) のように解析に成功する．

4. 評　　価

本章では，提案手法の評価と考察について述べる．

4.1 評 価 手 法

提案手法に関して以下の 4 点の評価を行う．

(1)　従来手法が対応している文法の解析が可能であるか．
(2)　従来手法に比べて解析時間が遅くなっていないか．
(3)　解析時間が線形性を保っているか．

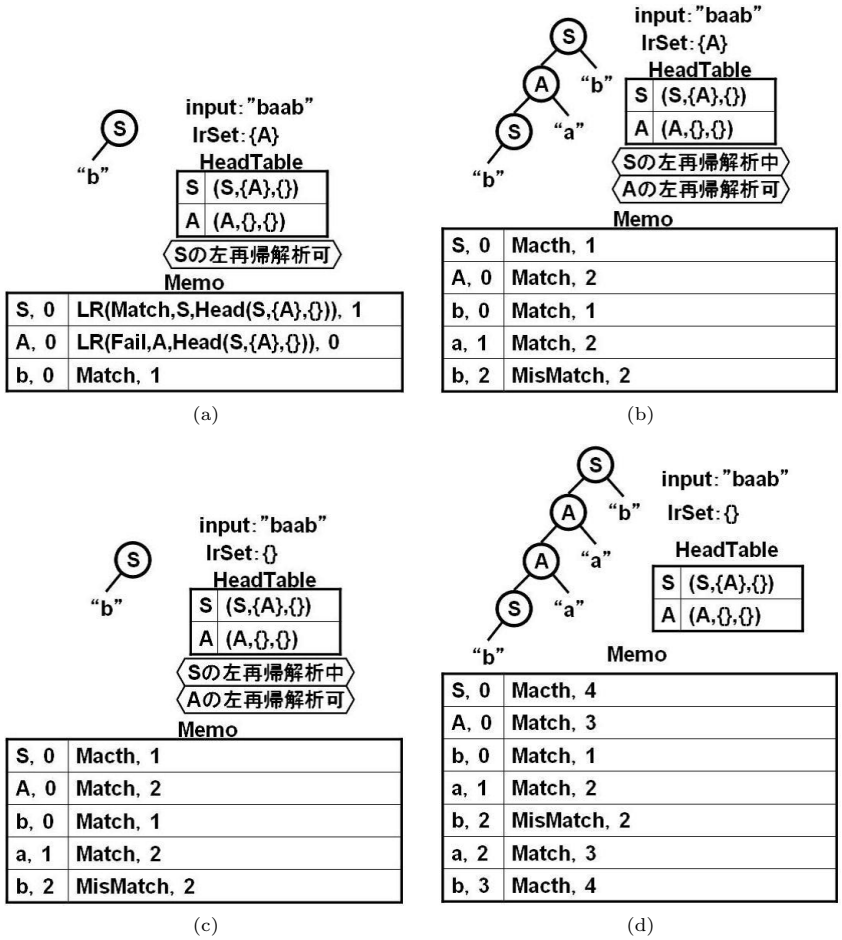

図 19　改良した提案手法の成功例
Fig. 19　Case of success with improved proposal method.

(4)　メモリ使用量が過度に増加しないか．

(1) の評価に関しては，従来手法の論文[4] で評価に用いられた文法と入力を利用する．文法を**図 21** に，入力を**図 20** にそれぞれ示す．図 21 の文法は Java 言語のプライマリ表現[5]

“this”
“this.x”
“this.x.y”
“this.x.m()”
“x[i][j].y”

図 20　Java プライマリの入力
Fig. 20　Java Primary expressions.

```
Primary                          ←   <PrimaryNoNewArray>
PrimaryNoNewArray                ←   <ClassInstanceCreationExpression>
                                 /   <MethodInvocation>
                                 /   <FieldAccess>
                                 /   <ArrayAccess>
                                 /   this
ClassInstanceCreationExpression  ←   new <ClassOrInterfaceType> ()
                                 /   <Primary> . new <Identifier> ()
MethodInvocation                 ←   <Primary> . <Identifier> ()
                                 /   <MethodName> ()
FieldAccess                      ←   <Primary> . <Identifier>
                                 /   super . <Identifier>
ArrayAccess                      ←   <Primary> [ <Expression> ]
                                 /   <ExpressionName> [ <Expression> ]
ClassOrInterfaceType             ←   <ClassName> / <InterfaceTypeName>
ClassName                        ←   C / D
InterfaceTypeName                ←   I / J
Identifier                       ←   x / y / <ClassOrInterfaceType>
MethodName                       ←   m / n
ExpressionName                   ←   <Identifier>
Expression                       ←   i / j
```

図 21　Java プライマリ表現
Fig. 21　Java's Primary expressions.

を簡略化した文法である．左再帰を含む文法であるため今回の評価に用いる．図 21 の文法と図 20 の入力を従来手法と提案手法の両方を用いて解析を行い，その結果と解析木を比較する．なお，図 21 の文法において，非終端記号は < > で囲んで表している．(2) の評価に関しては，**図 22** の文法を従来手法と提案手法の両方を用いて解析を行い，実行時間を比較する．入力は “a” を繰り返したトークン列を与える．(3) に関しては，**図 23** の文法を提案手法を用いて解析を行い，解析時間の線形性を調べる．入力は “ba” を繰り返して最後に “b” を付け加えたトークン列を与える．この文法と入力を従来手法を用いて解析すると異常

```
S ← A a / a
A ← S
```

図 22　性能比較に用いる文法
Fig. 22　Grammar using for performance comparison.

```
S ← A b / b
A ← A a / S a
```

図 23　線形性の評価に用いる文法
Fig. 23　Grammar using for evaluating linearity.

表 1　評価環境
Table 1　Evaluation environment.

カーネル	Linux 2.6.25
CPU	Intel(R) Xeon(TM) CPU 2.66 GHz
scala	scala version 2.8.0

終了する．(4) の評価に関しては，提案手法のメモリ使用量をアルゴリズムより見積もり，考察する．

従来手法と提案手法のアルゴリズムは Scala を用いて実装した．評価環境を**表 1** に示す．

4.2　評 価 結 果

本節では，従来手法と提案手法の結果の比較，従来手法と提案手法の性能比較，提案手法の線形性，提案手法のメモリ使用量の 4 点についての評価結果を述べる．

4.2.1　従来手法と提案手法の結果の比較

図 21 の文法と図 20 の入力を従来手法と提案手法の両方を用いて解析を行った．従来手法と提案手法の結果は一致した．さらに，結果の解析木も一致した．よって，従来手法で解析可能な文法は提案手法を用いても解析できると考えられる．提案したアルゴリズムは，従来手法で解析可能な文法の解析時において，HEADTABLE を用いた点以外の変更箇所は影響がないように設計した．この設計が成功していると考えられる．

4.2.2　従来手法と提案手法の性能比較

図 22 の文法を従来手法と提案手法の両方を用いて解析を行った．入力は “a” を繰り返したトークン列を与えている．**図 24** に評価結果を示す．簡単な間接左再帰の文法の場合，従来手法を用いたとき，提案手法を用いたときのどちらでも，入力文字数が増えると，それにほぼ比例して解析時間が増えるといえる．また，同じ文法の解析をしているにもかかわらず，従来手法を用いたときの方が，提案手法を用いたときより短い時間で解析を行っている．これは，提案手法では従来手法より条件文が増える，HEADRULE を用いてメモの操作を行う，の 2 点が原因であると考えられる．

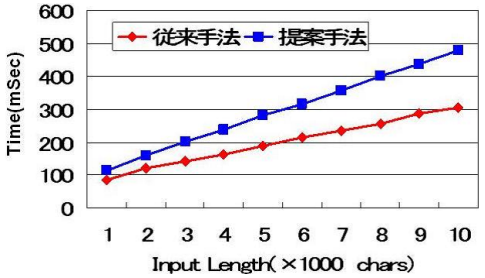

図 24　従来手法と提案手法による左再帰を含む文法の解析時間
Fig. 24　Analysis time of evaluating grammar containig left recursion with traditional method and proposal method.

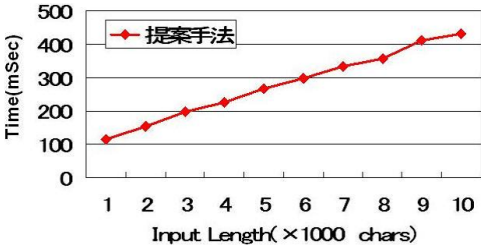

図 25　提案手法で対応した文法の解析時間
Fig. 25　Analysis time of evaluating grammar supporting proposal method.

4.2.3　提案手法の線形性の評価

図 23 の文法を提案手法を用いて解析を行った．入力は “ba” を繰り返して最後に “b” を付け加えたトークン列を与える．この文法と入力の解析は，従来手法を用いて解析すると異常終了する．**図 25** に解析結果を示す．図 25 より，提案手法を用いて新たに解析可能になった文法の解析においても，実行時間は線形性を保っていると分かる．これは，提案手法は従来手法と同様にメモ化を利用しているためであると考えられる．

4.2.4　提案手法におけるメモリ使用量増加についての考察

本項では，提案手法で増加するメモリ使用量をアルゴリズムより見積もり，考察する．提案手法では，HEADTABLE と lrSet という従来手法にはないデータが 2 つある．そのため，この 2 つのデータについて考察を行う．

HEADTABLE によるメモリ使用量の増加について考察する．HEADTABLE によるメ

モリ使用量は，最大の場合でも従来手法で用いているメモテーブルによるメモリ使用量を超えない．なぜなら，図 13 の 9. 行から分かるように，HEADTABLE はメモテーブルに記録される情報の一部である左再帰情報のみを記録するためである．

次に，lrSet によるメモリ使用量の増加について考察する．lrSet のサイズは，最大でも解析する文法に含まれるすべての非終端記号分である．なぜなら，図 18 の 7. 行から分かるように，lrSet は involvedSet の中にある条件にあった非終端記号のみを記録するためである．よって，lrSet によるメモリ使用量は，従来手法で用いているメモテーブルによるメモリ使用量を超えない．

以上の考察から，HEADTABLE と lrSet の追加によるメモリ使用量の増加は，最大の場合でも，従来手法で用いるメモテーブルによるメモリ使用量のたかだか 3 倍であると考えられる．

5. 結論と今後の課題

本論文では，同一入力位置で head rule が異なる左再帰が発生する文法を解析できる手法を提案した．結果，従来手法では解析できない文法を解析できるようになった．また，提案手法を用いての解析は線形時間で行える．

今後の課題として，新たにより多くの文法を解析可能な手法の提案があげられる．ここで，提案手法を用いても解析できないと現在判明している文法と入力の例を以下に示す．

S ← A a

A ← A a / a

input:“aaa”

この例は，間接左再帰を含まないが，従来手法，提案手法のどちらを用いても解析できない．

現段階では，解析不可能な文法と入力のパターンの法則性を見つけるまでには至っていない．そのため，他に解析不可能な文法と入力の例があるかもしれない．

参 考 文 献

1) Ford, B.: Packrat Parsing: Simple, Powerful, Lazy, Linear Time, *ICFP'02: Proc. 7th ACM SIGPLAN International Conference on Functional Programming*, New York, NY, USA, pp.36–47, ACM Press (2002).
2) Ford, B.: Packrat Parsing: a practical linear-time algorithm with backtracking, Master's thesis, Massachusetts Institute of Technology (Sept. 2002).
3) Grimm, R.: Better Extensibility Through Modular Syntax, *PLDI'06: Proc. 2006 ACM SIGPLAN conference on Programming Language Design and Implementation*, New York, NY, USA, pp.38–51, ACM Press (2006).
4) Alessandro, W., James, D. and Todd, M.: Packrat Parsers Can Support Left Recursion, *ACM SIGPLAN 2008 Workshop on Partial Evaluation and Program Manipulation*, January 7–8, 2008, San Francisco, California, USA, pp.103–110 (2008).
5) Gosling, J., Joy, B., Steele, G. and Bracha, G.: *The Java Language Specification, 3rd Edition*, Addison-Wesley (2005).
6) Ford, B.: Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, *POPL'04: Proc. 31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, New York, NY, USA, pp.111–122, ACM Press (2004).

(平成 22 年 10 月 6 日受付)
(平成 23 年 1 月 15 日採録)

白田　佳章

昭和 60 年生．平成 20 年熊本大学工学部数理情報システム工学科卒業．同年熊本大学大学院自然科学研究科情報電気電子工学専攻入学．現在に至る．

木山　真人（正会員）

昭和 51 年生．平成 11 年広島市立大学情報科学部情報工学科卒業．平成 15 年広島市立大学大学院情報科学研究科情報科学専攻博士後期課程修了．同年熊本大学工学部数理情報システム工学科助手．現在，熊本大学大学院自然科学研究科情報電気電子工学専攻助教．博士（情報工学）．プログラミング言語，言語処理系，オブジェクト指向言語の高速化手法に興味を持つ．

芦原　　評（正会員）

昭和 39 年生．昭和 62 年東京大学理学部情報科学科卒業．平成 4 年同大学院理学系研究科博士課程修了．博士（理学）．同年電気通信大学情報工学科助手．平成 11 年熊本大学工学部数理情報システム工学科助教授．現在，熊本大学大学院自然科学研究科情報電気電子工学専攻准教授．