プログラミング 7－4
（1996. 5. 31）

正規右辺文法のLALRパーサの新しい実現法

森本 真一
morimoto@ccs.mt.nec.co.jp
日本電気 マイコンソフト開発環境研究所
〒108 港区芝浦2－11－5

本稿では、正規右辺文法に対するLALR解析方法として従来よりも簡単な方法を述べる．本方式ではstack conflictへの対応として，stack shift時に対応する構文規則の記号をスタックにpushするだけであり，従来の方法のようにlook back状態を算出したり，スタックの要素に関するカウンタを操作する必要がない．

本稿では、まず本方式の内容を必要な定義と共に述べ，次に本方式での動作を特徴的な場合を例として説明する．さらに本方式の課題である還元時の動作の複雑さを解消した方式を述べる．

Yet Another Generation of LALR parsers for Regular Right Part Grammars

Shin-ichi MORIMOTO

Microcomputer Software Laboratories NEC Corporation
2−11−5 Shibaura, Minato−ku, Tokyo, 108 Japan

A simple method for building LALR parsers for regular right part grammars is given. No grammar transformation is required. No extra data structures such as counters are required.

At stack shift states, parser pushs the symbol that corresponds to the production shifted in the states. At reduce state parser pops to the corresponding symbol. A method that simplifies the action in the reduce states is also given.

1 はじめに

本稿では、正規右辺文法に対する新しいLR構文解析法を述べる．正規右辺文法とは，文脈自由文法の生成規則の右辺の記述に構文記号の正規表現を許したものである．正規右辺文法のうち，k個の構文記号の先読みにより左から右に構文解析できるものをELR(k)文法という．

ELR文法に対するLR構文解析では還元時の動作が問題になる．つまりELR文法では，通常のLR文法と異なり構文規則の右辺の長さが決まっていないため，そのままでは，還元時にpopするスタックの要素数を決定できない．

本稿の方法は，ELR文法から(他の等価の文法に変換しないで)直接LRパーサを生成するが，パーサを直接生成する方法として従来提案されているものでは，この問題に対して

a) look back 状態という状態までpopする [1]

b) popする要素数に関するカウンタを設け
カウンタの値に基づいてpopする [2][3]

という方法で対処している．

本稿の方法は，スタックのpush時にその構文規則に対応する記号もpushし，還元時には還元すべき構文規則に対応する記号までpopするものである．この方法では，a)と異りlook back状態を算出する必要はなく，またb)と異り，カウンタなどの特別なデータ構造を必要としない．

本稿では，さらにこの方法の課題とそれへの対策を述べる．

2 術語の定義

ここでは、[1]を参考に以後の議論に必要な術語の定義を行なう。

定義 2.1 (右辺正規文法)

右辺正規文法Gは，7つ組$(V_N, V_T, S, Q, \partial, F, P)$で定義される。ただし

V_N は非終端記号の集合、V_T は終端記号の集合
$V = V_N \cup V_T$ とする．

S は開始記号、　　Q は右辺の状態の集合

∂ は状態遷移関数 $(Q \times V \to Q)$

F は終端状態、　　P は構文規則の集合

構文規則を $A \to q(A \in V_N, q \in Q)$ と表す。

定義 2.2 (↓)

$p, q \in Q$ に対して，

$p \downarrow q$ iff $\exists A$ s.t. $\partial(p, A) \in Q$, $(A \to q) \in P$

注意　↓のtransitive closureを$\downarrow^*$で表す．

定義 2.3 (closure)

$R \subseteq Q$ に対して，

closure(R) = $\{q \mid \exists p \in R$ s.t. $p \downarrow^* q\}$

定義 2.4 (LR(0)オートマトン)

右辺正規文法Gに対するLR(0)オートマトンは，以下の初期状態$\mathbf{q_0}$と，遷移関数Nextを持つ．

$\mathbf{q_0}$ = closure($\{q \mid \exists (S \to q) \in P\}$)

Next(q,X) = closure(succ(q,X))

但し，succ(q,X) = $\{\partial$(q,X) | q∈q$\}$

注意　LR(0)オートマトンの状態は，太字で表す．

定義 2.5 (kernel, nonkernel)

LR(0)オートマトンの状態qに対して，kernel(q),nonkernel(q)を次のように定める．

kernel($\mathbf{q_0}$) = ϕ

kernel(Next(q,X)) = succ(q,X)

nonkernel(q) = {q|∃ p∈kernel(q) s.t. p↓*q}

本稿は，LALR(1)パーザで解析できる正規右辺文法を対象とする．つまり，次の条件を満たすとする．

定義 2.6 (ELALR(1)文法)

次の条件を満たす正規右辺文法をELALR(1)文法という．

1) LR(0)オートマトンにおけるコンフリクトは先読み記号を用いる事により解決できる．

2) 還元時のhandleは，一意に決定できる．

定義 2.7 (→)

$q_1 \in \mathbf{q_1}$, $q_2 \in \mathbf{q_2}$ に対して，

$\mathbf{q_2}$=Next($\mathbf{q_1}$,X), $q_2 = \partial(q_1, X)$ の時，$(\mathbf{q_1}, q_1) \to^X (\mathbf{q_2}, q_2)$

とする．

このとき

$q_1 \in$kernel($\mathbf{q_1}$) ならば，　$\mathbf{q_1} \overset{K}{\to} \mathbf{q_2}$

$q_1 \in$nonkernel($\mathbf{q_1}$) ならば，　$\mathbf{q_1} \overset{N}{\to} \mathbf{q_2}$

と表す．

定義 2.8 (configuration)

構文解析の configuration とは,

スタックの内容, 現在の状態, 現在の入力列から構成される. つまり configuration は,

$(q_0 P_0 \beta_0\ q_1 \ldots \beta_{n-1}\ q_n P_n \beta_n,\ q,\ Xz)$

と表せる.

但し,

$q_i \in Q,\ P_i \subseteq P,\ \beta_i \in V^*\ \ (0 \leq i \leq n)$

$q \in Q,\ X \in V,\ z \in V_T^*$

定義 2.9 (パーサの動作)

configuration が,

$(q_0 P_0 \beta_0\ q_1 \ldots \beta_{n-1}\ q_n P_n \beta_n,\ q,\ Xz)$ の場合にパーザは, 現在の状態 q と入力記号 X に応じて, 次の 3 つの動作のいずれかを行なう.

各動作を行なう条件と動作後の configuration は, 次の通りである

1) shift(X) ($q^K \rightarrow^X q'$ の場合)

$(q_0 P_0 \beta_0 q_1 \ldots \beta_n X,\ q',\ z)$

2) stack shift(X) ($q^N \rightarrow^X q'$ の場合)

$(q_0 P_0 \beta_0 q_1 \ldots \beta_n q P_{n+1} X,\ q',\ z)$

但し, P_{n+1}={p|∃q∈q, q'∈q' s.t. p は A→q, q∈nonkernel(q), (q,q)→^{X}(q',q')}

3) reduce (q は,p(=A→q") の final 状態を含み, X∈look ahead(p,q) の場合)

$(q_0 P_0 \beta_0 q_1 \ldots \beta_{k-1} q_k P_k \beta_k,\ q_{k+1},\ Az)$

但し, P_{k+1} は, $\beta_{k+1} \ldots \beta_n$ を handle とする構文規則を含み, 最も上にある (stack top に近い) もの

なお, 1) と 2) の条件が共に成立する場合を stack conflict という. その場合は, 2) の動作を行なうとする (これを stack conflict shift という).

本方式は, ELALR(1) 文法に対しては正しく解析できる. というのは, 定義 2.6 の 1) の条件により, shift と stack shift の際のオートマトンの動作は一意に決まり, 2) の条件により還元の際の構文規則とその handle は一意に決まるためスタックのどの要素まで pop すべきかを決定できるからである.

3 本方式による解析例

ここでは, 図 1 で表される文法 G1 に対する, 本方式による解析例を示す.

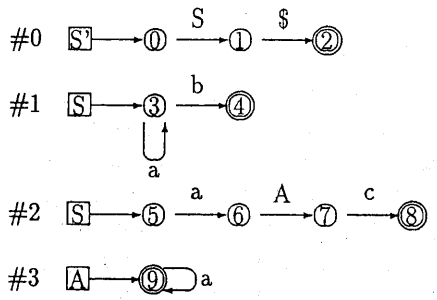

図 1 正規右辺文法 G1 の構文規則

G1 に対する LR オートマトンは, 図 2 で表される [1].

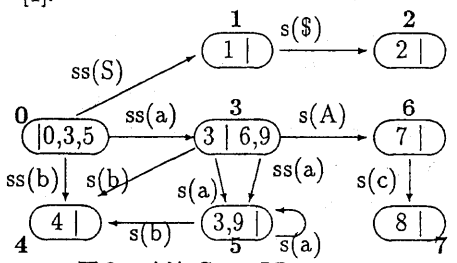

図 2 文法 G1 の LR オートマトン

図 2 で, | の左側の数字は kernel の要素, | の右側の数字は nonkernel の要素 を表す.

ss(x) は stack_shift(x) を, s(x) は stack(x) を表す.

状態 3 で入力が a の場合に, stack conflict が生じる.

G1 の記号列 aaac と記号列 aaab に対する本方式による解析は, 次のようになる.

○ a a a c に対する解析動作

(0{#1,#2}a 3{#3}a a,5,c $)

↓ reduce for #3 (A: a a)

(0{#1,#2}a , 3, Ac $)

↓ s(A)

(0{#1,#2}a A , 6, c $)

↓ s(c)

(0{#1,#2}a A c , 7, $)

↓ reduce for #2 (S: a A c)

(, 0, S $)

○ a a a b に対する解析動作

(0{#1,#2}a 3{#3}a a , 5, b $)

↓ s(b)

(0{#1,#2}a 3{#3}a a b, 4, $)

↓ reduce for #1 (S: a a a b)

(, 0, S $)

4 本方式の課題

本方式の課題は，還元時の動作にある．本方式では([1]の方式でも同じであるが)，還元時にスタック上の構文規則記号pの位置より上にある記号列がpのhandleとなるか(pの右辺のオートマトンで受理されるか)を判定する必要がある．この動作は，構文解析における他の動作(スタックに要素をpushする等)に比べると非常に複雑である．解析の対象がELALR(1)文法に制限を加えた文法，例えば，G1のように同じ構文規則の間でstack conflictが発生しないELALR(1)文法の場合は，還元時の動作をより簡単な動作，例えば，

求める構文規則記号pでスタックの
最も上にあるものまでpopする

に変更できる．しかし一般のELALR(1)文法では，この制限を満たさない場合も存在するので，この動作に置き換えることはできない．

同じ構文規則の間でstack conflictが発生する例として，次の文法G2を考える．

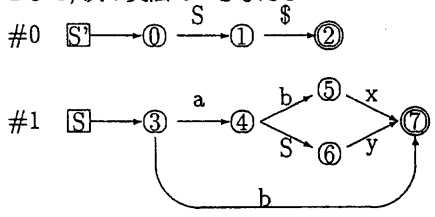

図3　正規右辺文法 G2 の構文規則

G2に対するLRオートマトンは，図4で表される．状態3で入力がbの場合に，stack conflictが生じる．

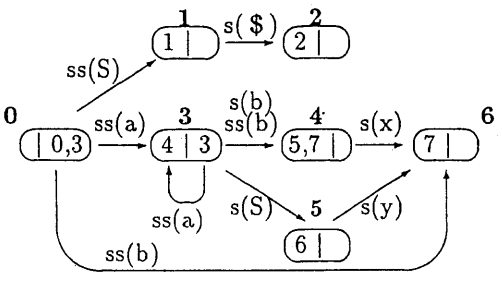

図4　文法 G2 のLRオートマトン

G2の記号列abxに対する解析は，次のようになる．

(0{#1}a, 3, bx $)
　↓ ss(b)
(0{#1}a3{#1}b, 4, x $)
　↓ s(x)
(0{#1}a3{#1}bx, 6, $)
　↓ reduce for #1 (S: a b x)

この時，スタックの上にある#1ではなく，その下の#1までpopしなければならない．

5 改良した方法

本章では，3章で述べた方法を改良し一般のELALR(1)文法に対しても，還元時に複雑な動作を行なわない方法を述べる．この方法の基本的な考え方は，次の通りである．

- 構文規則pをpushしたstack conflict shiftの後にpが還元の対象でないと判明した時点で，pに対応した記号($\overline{p}$とする)をスタックにpushする．
- pで還元する場合は，それより上にあるpの個数と$\overline{p}$の個数が等しい(それより上にあるpと$\overline{p}$がmatchする)pで最も上にあるものまでスタックをpopする．

以下に，この方法を必要な定義と共に主に述べる

定義 5.1 (closure)
$R \subseteq Q^*$ に対して，
$\mathrm{closure}(R) = \{q \mid \exists p \in R \text{ s.t. } \mathrm{tail}(p)\downarrow^* q\}$

注意　この方法では，LR(0)オートマトンの状態は，3章の方法のような構文規則の右辺の状態の集合(Qの部分集合)ではなく，右辺の状態の列の集合(Q^*の部分集合)である．

定義 5.2 (LR(0)オートマトン)
本方法で定義されるLR(0)オートマトンは，以下の初期状態$\mathbf{q_0}$と，遷移関数Nextを持つ．

$\mathbf{q_0} = \mathrm{closure}(\{q \mid \exists(S \to q) \in P\})$
$\mathrm{Next}(\mathbf{q},X) = \mathrm{closure}(\mathrm{succ}(\mathbf{q},X))$
　但し，$\mathrm{succ}(\mathbf{q},X)=\{\mathrm{next}(q, \partial(q,X)) \mid q \in \mathbf{q}\}$
　　$\mathrm{next}(q,r)=q \cdot r$　(rはq上にない)
　　$=q'$　($q=q' \cdot q''$, $r=\mathrm{tail}(q')$)

定義 5.3 (kernel, nonkernel)

LR オートマトンの状態 $\mathbf{q}$ に対して,
kernel($\mathbf{q}$),nonkernel($\mathbf{q}$) を次のように定める.

$\mathrm{kernel}(\mathbf{q_0}) = \phi$

$\mathrm{kernel}(\mathrm{Next}(\mathbf{q},\mathrm{X})) = \mathrm{succ}(\mathbf{q},\mathrm{X})$

$\mathrm{nonkernel}(\mathbf{q}) = \{\mathrm{q} | \exists\ \mathrm{p}\in\mathrm{kernel}(\mathbf{q})\ \mathrm{s.t.}\ \mathrm{tail}(\mathrm{p})\downarrow^{*}\mathrm{q}\}$

定義 5.2 で定義したオートマトンや定義 5.3 で定義した kernel, nonkernel に対して, 定義 2.7 と同じく $\rightarrow$, $^{K}\rightarrow$, $^{N}\rightarrow$ を定義する.

定義 5.4 (分離状態)

$q_1, q_1' \in \mathbf{q_1}$, $q_2, q_2' \in \mathbf{q_2}$, $q_3 \in \mathbf{q_3}$ に対して,

- $(\mathbf{q_1}, q_1)^{K}\rightarrow^{X}(\mathbf{q_2}, q_2)^{K}\rightarrow^{X}(\mathbf{q_3}, q_3)$
- $(\mathbf{q_1}, q_1')^{N}\rightarrow^{X}(\mathbf{q_2'}, q_2)$
- $\partial(q_2', \mathrm{X})$ は, 存在しない.

が成り立つ時, $\mathbf{q_3}$ を分離状態という.

また, q_1' を右辺に持つ構文規則を脱落規則という.

定義 5.5 (configuration)

構文解析の configuration とは,

スタックの内容, 現在の状態, 現在の入力列から構成される. つまり configuration は,

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_{n-1}\ \mathbf{q_n}\overline{P}_n P_n \beta_n,\ \mathbf{q},\ \mathrm{Xz})$

と表せる.

但し,

$\mathbf{q_i} \in \mathbf{Q}$, $P_i \subseteq P$, $\overline{P}_i$ は脱落規則の集合,

$\beta_i \in V^{*}$ $(0 \le i \le n)$, $\mathbf{q} \in \mathbf{Q}$, $\mathrm{X}\in V$, $z \in V_T{}^{*}$

定義 5.6 (パーサの動作)

configuration が,

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_{n-1}\ \mathbf{q_n}\overline{P}_n P_n \beta_n,\ \mathbf{q},\ \mathrm{Xz})$

の場合に, パーザは現在の状態 $\mathbf{q}$ と入力記号 X に応じて, 次の 5 つの動作のいずれかを行なう.

各動作を行なう条件と動作後の configuration は, 次の通りである

1) shift(X) $(\mathbf{q}^{\mathrm{K}}\rightarrow^{\mathrm{X}}\mathbf{q}'$ で, $\mathbf{q}'$ は非分離状態)

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_n X,\ \mathbf{q}',\ z)$

2) cancel(X) $(\mathbf{q}^{\mathrm{K}}\rightarrow^{\mathrm{X}}\mathbf{q}'$ で,$\mathbf{q}'$ は分離状態)

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_n\ \overline{P}_{n+1}\{\}X,\ \mathbf{q}',\ z)$

但し, $\overline{P}_{n+1}$ は, q' の脱落規則の集合

3) stack shift(X) $(\mathbf{q}^{\mathrm{N}}\rightarrow^{\mathrm{X}}\mathbf{q}'$ で,$\mathbf{q}'$ は非分離状態)

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_n \mathbf{q}\{\}P_{n+1}X,\ \mathbf{q}',\ z)$

但し,P_{n+1}={p|$\exists$q$\in$**q**, q'$\in$**q**' s.t. p は A→q,
q$\in$nonkernel(**q**), (**q**,q)$\rightarrow^{X}$(**q**',q')}

4) stack cancel(X) $(\mathbf{q}^{\mathrm{N}}\rightarrow^{\mathrm{X}}\mathbf{q}'$ で, $\mathbf{q}'$ は分離状態)

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_n \mathbf{q}\overline{P}_{n+1} P_{n+1} X,\ \mathbf{q}',\ z)$

但し,P_{n+1}={p|$\exists$q$\in$**q**, q'$\in$**q**' s.t. p は A→q,
q$\in$nonkernel(**q**), (**q**,q)$\rightarrow^{X}$(**q**',q')}

$\overline{P}_{n+1}$ は, q' の脱落規則の集合

5) reduce (**q** は, p(=A→q") の final 状態を含み, X$\in$look ahead(p,q))

$(\mathbf{q_0}\overline{P}_0 P_0 \beta_0 \mathbf{q_1} \ldots \beta_{k-1}\mathbf{q_k}\ \overline{P}_k P_k \beta_k,\ \mathbf{q_{k+1}},\ \mathrm{Az})$

但し, P_{k+1} は, 次の条件を満たし最も上にある

- p を含む
- $\sum_{j=k+2}^{n} P_j$中のpの個数
  $=\sum_{j=k+2}^{n} \overline{P}_j$中の$\overline{p}$の個数

G2 に対する本方法での LR オートマトンは, 図 5 で表される. 状態 6 は, 分離状態である

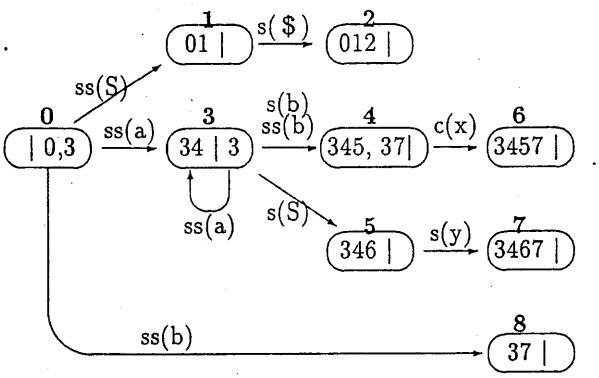

図 5 文法 G2 の LR オートマトン

c(x) は cancel(x) を表す.

G2 の記号列 abx に対するこの方式による解析は, 次のようになる.

(**0**{}{#1}a, **3**, bx $)

↓ ss(b)

(**0**{}{#1}a**3**{}{#1}b, **4**, x $)

↓ c(x)

(**0**{}{#1}a**3**{}{#1}b{#1}{}x, **6**, $)

↓ reduce for #1 (S: a b x)

(, **0**, S $).

以下に, この方法で ELALR(1) 文法を正しく解析できる事の証明を述べる.

stack conflict shift を含む状態を stack conflict shift

状態という.

命題1

stack conflict shift 状態を含むループには，必ず push された構文規則を脱落規則とする分離状態を含む

証) stack conflict shift 状態を含み，分離状態を含まないループが存在したとする.

このループを

$(q_1,q_1)\rightarrow^{\beta_1}(q_2,q_2)\rightarrow^{X}\ q_3,q_3)\rightarrow^{\beta_2}(q_1,q_1)$

とする (状態 q_2 で，入力記号が X の時に，構文規則 p との stack conflict が生じるとする). p で stack shift した後も，このループを実行できるので，p の handle(の1つ) は $(X\beta_2\beta_1)^*\cdot\beta_3$ と表せる.

故に，p の左辺の構文記号を A とすると，

$\beta\beta_1X\beta_2\beta_1X\beta_2\beta_1\beta_3$ という記号列は，

$\beta\beta_1A$, $\beta\beta_1X\beta_2\beta_1A$ のいずれにも還元できる事になり，定義 2.6 の条件に反する.

また，定義 5.2 のオートマトンの構成法により，次の命題が成り立つ.

命題2

初期状態から分離状態へのパス上には，必ず対応する stack conflict shift 状態が存在する.

命題1，命題2 から，分離状態でスタックに push される構文規則は，(stack conflict shift 状態で push された) 脱落した構文規則に 1：1 に対応している. つまり，還元時にスタックの top から脱落規則の数だけ余分に構文規則の記号を pop することにより，正しい位置に pop できる.

6 まとめ

本稿の方法は，解析時にカウンタ等のデータ構造を必要としない点は [1] の方法と同じである. [1] の方法に比べると，3 章の方式は次の特徴がある.

1. stack conflict が生じない場合は，[1] の方法の方が解析時に構文規則の集合を push する必要がないので簡単である.
2. stack conflict が生じる場合は，3 章の方法の方が look back 状態を算出する必要がないので簡単である.

   同じ構文規則間で stack conflict が生じない (異なる構文規則間で stack conflict が生じる) 場合は，さらに，3 章の方式の還元時の動作を，

   求める構文規則の記号で stack の
   最も上にあるものまで pop する

   という，簡単な動作に置き換える事ができる.

   同じ構文規則間で stack conflict が生じる場合でも，5 章の方法では還元時の動作をより簡単な動作に置き換える事ができる.

参考文献

[1] Ikuo Nakata, Masataka Sassa : Generation of Efficient LALR Parsers for Regular Right Part Grammars, Acta Informatica, Vol.23, pp.149-162, 1986.

[2] 佐々政孝, 中田育男：正規右辺文法の LR パーサの簡単な実現法, 情報処理学会論文誌, Vol.27, No1, pp.124-127, 1986.

[3] 張又普, 中田育男, 佐々政孝：正規右辺文法の効率の良い LR パーサの簡単な実現法, 情報処理学会論文誌, Vol.28, No11, pp.1162-1167, 1987.