文脈自由文法の並列構文解析

峯，恒憲
九州大学大学院総合理工学研究科情報システム学専攻

谷口，倫一郎
九州大学大学院総合理工学研究科情報システム学専攻

雨宮，真人
九州大学大学院総合理工学研究科情報システム学専攻

https://doi.org/10.15017/17163

出版情報：九州大学大学院総合理工学報告．11（3），pp.339-348，1989-12-01．九州大学大学院総合理工学研究科
バージョン：
権利関係：

九州大学総合理工学研究科報告
第11巻 第3号 339-348頁 平成元年12月

Engineering Sciences Reports, Kyushu University
(KYUSHU DAIGAKU SOGORIKOGAKU KENKYUKA HOKOKU)
Vol. 11, No. 3 pp. 339 - 348 December 1989

文脈自由文法の並列構文解析

峯　恒憲*・谷口倫一郎**・雨宮真人**

(平成元年8月31日　受理)

Parallel Parsing for Context-Free-Grammar

Tsunenori Mine, Rin-ichiro Taniguchi and Makoto Amamiya

The paper presents an efficiently parallel parsing algorithm for arbitrary contextfree grammars. This algorithm uses an LR transition network which is similar to a shift reduce table used by standard LR parsing algorithm. It runs in linear time for n (where n is a length of the string being parsed); it needs $O(n^2)$ processes, and $O(n^2)$ spaces.

1. はじめに

いままでに並列構文解析アルゴリズムについての報告が多数なされている．並列に解析する目的は，実現の可能性のあるマシン上で実現間処理を行うことにあるが，いままでに提案されているものは次の点でこの目的を満たしていない．

(1) 入力に同期した解析を行うために，解析時間が $O(n^2)$ (n は入力文の長さ) となる．

(2) プロセス数が多くなりすぎる．

本稿で提案するアルゴリズムは一般の文脈自由文法を対象とし，LR 法に類似した手法[1)2)]で作成した状態遷移図[3)4)]を使用して，全ての可能性を並列に試しながら入力に非同期に解析を進める．このアルゴリズムは解析時間が $O(n)$ ですみ，プロセス数と使用メモリ空間をどちらも $O(n^2)$ に抑えている．

2. LR 状態遷移図

2.1 準　備

まず，本稿で使用する用語の定義を行う．

定義1　クロージャ：文法規則の右辺にマーカ"・"を埋め込んだもの ($[A\rightarrow \alpha \cdot \beta]$) をマーカ付規則と呼び，マーカ付規則の集合のクロージャ Closure ( ) を次式のように定義する．ここで｛ ｝はマーカ付規則の集合を表す．

Closure ($\{[A\rightarrow \alpha \cdot B\beta]\}$) =
$\{[A\rightarrow \alpha \cdot B\beta]\} \subset$ Closure ($\{[B\rightarrow \cdot \gamma]\}$)

Closure ($[A\rightarrow \alpha \cdot a\beta]$) = $\{[A\rightarrow \alpha \cdot a\beta]\}$
($|\alpha| \geqq 0, |\beta| \geqq 0, |\gamma| \geqq 0$)

ここで，英大文字は非終端記号を表し，ギリシャ文字は終端記号と非終端記号からなる記号列を表す．また，$|\alpha|$ は記号列 α の長さを表す．但し，英大文字の X と Y に限っては終端記号か非終端記号のどちらか一方を表す（これを単に（非）終端記号と記す）ことにする．

定義2　状態：LR 状態遷移図中のノードを状態と呼ぶ．この状態は LR (0) 項の状態と同じもので，この状態を記号 S_i ($0 \leqq i \leqq C-1$) でラベル付けする．ここで C は作成される状態の数である．

定義3　パケット：4つ組のリスト $\langle S_i, X, k, f\rangle$ をパケットと呼ぶ．ここで S_i は状態遷移図中の状態の番号，X は（非）終端記号，k は入力ポインタ，f はチェーンリストの長さである．

定義4　チェーンリスト：複数のリスト（パケット）をポインタで結んだものとする．

2.2 shift reduce テーブルと LR 状態遷移図

一般の LR 法で使用される shift reduce テーブルには次のような問題があった．[3)4)]

(1) reudce 操作の時，その操作で使用する文法規則の左辺記号についてのクロージャを求めた状態（以下，これを「戻り先」と呼ぶ）へ戻らなければ，次に

*情報システム学専攻博士課程
**情報システム学専攻

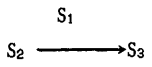
遷移すべき状態へ移ることができない．例えば，**Fig. 1(a)** の S_2 からは，S_3 へ移動する前に S_1 に戻らなければならない．

(2) reduce 操作の時，次にどの状態へ遷移すべきかが明示されていない．これは解析を不透明にする．

我々の LR 状態遷移図では，状態番号を遷移情報（アーク上の記号）として用いることにより(1)，(2)の問題を解決した．LR 法では shift 操作の時に状態番号をスタックに push するため，状態番号を見れば戻り先がわかるからである．**Fig. 1(a)** の例は LR 状態遷移図では **Fig. 1(b)** のようになる．

	element of states
S_1	A→X·Y Z, Y→·a B
S_2	Y→a B·
S_3	A→X Y·Z

(a)



(b)

Fig. 1　An example of an LR transition network for a conventional shift reduce table.

2.3 LR 状態遷移図作成アルゴリズム

状態の作成と遷移情報の作成アルゴリズムを以下に示す．状態の作成アルゴリズムは LR (0)項を求めるアルゴリズムと同じである．

2.3.1 状態の作成アルゴリズム

(1) 初期設定

文法 G に文法規則 "%→S \$" を加える．ここで S，% は非終端記号，\$は終端記号で，S は開始記号，\$は受理記号を表す．

S_0=Closure（{[%→・S \$]}）

(2) $S_i \in R$ $(0 \leqq i \leqq m)$ という集合 R に対して，

もし $GOTO(S_i, X) \notin R$ ならば，$GOTO(S_i, X)$ を R に加える．ここで m は既に生成された状態の数で，$GOTO(S_i, X)$ は以下の様に定義される．

$GOTO(S_i, X)$ ：S_i $(0 \leqq i \leqq m)$ において

$[A \rightarrow \alpha \cdot X\beta] \in S_i$ の時，$[A \rightarrow \alpha X \cdot \beta]$ という形をした全てのマーカ付規則の集合を J とすると

$GOTO(S_i, X)$ =Closure (J)

2.3.2 遷移情報の作成アルゴリズム

(1) もし，$[A \rightarrow \alpha \cdot a\beta] \in S \cap [A \rightarrow \alpha a \cdot \beta] \in S_j$ ならば，S_i から S_j への遷移情報を a とする．

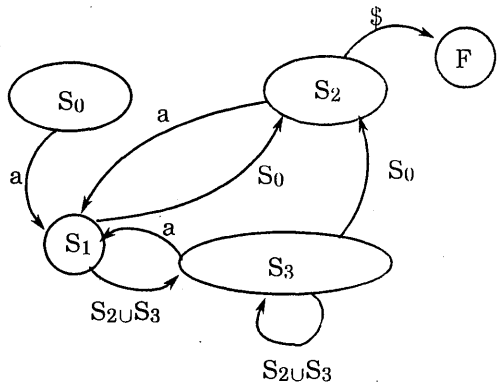
(2) もし，$[B \rightarrow \alpha A \cdot \beta] \in S \cap [A \rightarrow \gamma \cdot] \in S_i$ ならば，$[B \rightarrow \alpha \cdot A\beta] \in S_k$ という全ての S_k を S_i から S_j への遷移情報とする．

3. アルゴリズム

3.1 O(n)時間・O(n^3)プロセス数のアルゴリズム

解析は前章で述べた LR 状態遷移図を使用して行う．一般の文脈自由文法から作成した LR 状態遷移図の中のある状態では，複数の操作（shift-reduce, reduce-reduce）の実行が可能となる場合がある．そのような場合には各操作毎にプロセスを割り当て，それらを並列に実行する．即ち，shift 操作と reduce 操作を並列に実行するという，入力に対して同期をとらない解析を行う．一つのプロセスが一つの構文木を作成するようにプロセスを割り当てると，曖昧さの数（O(C^n)，C は定数，n は入力文の長さ）だけプロセスが増加するため，なんらかの形でプロセスの増加を抑えることが必要となる．

プロセスの増加を抑えるために，入力に対して同期をとる方式[5)6)7)]が提案されてるが，入力に同期させた場合，解析時間が O(n^2)（n は入力の長さ）かかるため，本手法では入力には同期をとらず shift／reduce の各操作に対して同期をとることによりプロセス数を抑え



S0={[% → ·A $],[A→·A A],[A→·a]}

S1={[A→a·]}

S2={[% → A ·$],[A→A ·A],[A→·A A],[A→·a]}

S3={[A→A A·],[A→A ·A],[A→·A A],[A→·a]}

Fig. 2　LR transition network of grammar G (A→AAla)

Table 1　Parsing process of an input sentence "a a a a"

step	state	chain list	inf→state
0	S_0	$<\ >$	$(a) \rightarrow S_1$
1	S_1	$<S_0,a,1,1>$	$(S_0) \rightarrow S_2$
2	S_2	$<S_0,A,1,1>$	$(a) \rightarrow S_1$
3	S_1	$<S_2,a,2,2> \rightarrow <S_0,A,1,1>$	$(S_2) \rightarrow S_3$
4	S_3	$<S_2,A,2,2> \rightarrow <S_0,A,1,1>$	$(a) \rightarrow S_1$ $(S_0) \rightarrow S_2$
5	S_1	$<S_3,a,3,3> \rightarrow <S_2,A,2,2> \rightarrow <S_0,A,1,1>$	$(S_3) \rightarrow S_3$
	S_2	$<S_0,A,2,1>$	$(a) \rightarrow S_1$
6	S_3	$<S_3,A,3,3> \rightarrow <S_2,A,2,2> \rightarrow <S_0,A,1,1>$	$(a) \rightarrow S_1$ $(S_2) \rightarrow S_3$
	S_1	$<S_2,a,3,2> \rightarrow <S_0,A,2,1>$	$(S_2) \rightarrow S_3$
7	S_1	$<S_3,a,4,4> \rightarrow <S_3,A,3,3> \rightarrow <S_2,A,2,2> \rightarrow <S_0,A,1,1>$	$(S_3) \rightarrow S_3$
	S_3	$<S_2,A,3,2> \rightarrow <S_0,A,2,1>$ $<S_2,A,3,2> \rightarrow <S_0,A,1,1>$	$(S_0) \rightarrow S_2$ $(S_0) \rightarrow S_2$ $(a) \rightarrow S_1$
8	S_3	$<S_3,A,4,4> \rightarrow <S_3,A,3,3> \rightarrow <S_2,A,2,2> \rightarrow <S_0,A,1,1>$	$(S_3) \rightarrow S_3$
	S_1	$<S_3,a,4,3> \rightarrow <S_2,A,3,2> \rightarrow <S_0,A,2,1>$ $<S_3,a,4,3> \rightarrow <S_2,A,3,2> \rightarrow <S_0,A,1,1>$	$(S_3) \rightarrow S_3$
	S_2	$<S_0,A,3,1>$	$(a) \rightarrow S_1$
9	S_3	$<S_3,A,4,3> \rightarrow <S_2,A,2,2> \rightarrow <S_0,A,1,1>$ $<S_3,A,4,3> \rightarrow <S_2,A,3,2> \rightarrow <S_0,A,2,1>$ $<S_3,A,4,3> \rightarrow <S_2,A,3,2> \rightarrow <S_0,A,1,1>$	$(S_2) \rightarrow S_3$ $(S_2) \rightarrow S_3$
	S_1	$<S_2,a,4,2> \rightarrow <S_0,A,3,1>$	$(S_2) \rightarrow S_3$
10	S_3	$<S_2,A,4,2> \rightarrow <S_0,A,3,1>$ $<S_2,A,4,2> \rightarrow <S_0,A,2,1>$ $<S_2,A,4,2> \rightarrow <S_0,A,1,1>$	$(S_0) \rightarrow S_2$ $(S_0) \rightarrow S_2$ $(S_0) \rightarrow S_2$
11	S_2	$<S_0,A,4,1>$	$(\$) \rightarrow final$
12	*final state*		*accepted*

ることを考える．即ち全てのプロセスの1回の shift／reduce 操作をある一定時間内で行うことにする．これはマクロ SIMD 型の実行に等しくパイプライン処理の効果を生む．

解析にはスタックの代わりにパケットをポインタで結んだチェーンリストを使用し，このパケット1つに対して1つのプロセスを割り当てる．そして各操作を行うある時間帯に，同じ状態 S_i にあり同じパケットをもつ複数のプロセスを1つにまとめる．1つにまとめたプロセスに，他のプロセスがもっていたパケットのポインタを持たせる．この時，複数のポインタに対する reduce 操作を独立動作にするために，ポインタ1つ1つに対してもプロセスを割り当てる．そして reduce 操作が起こるまで，これらのプロセスを活性状態のままで待機させておく．

文法 G（A→AA|a）を例にとり説明する．文法Gから作成した LR 状態遷移図は **Fig. 2** のようになる．

この状態遷移図を使用して文 "aaaa $" を解析する

時，解析は **Table 1** のように進む．**Table 1** の inf→state の欄の中で，活弧でくくった遷移情報を→の左に記し，遷移先状態番号を→の右に記す．

Table 1 のステップ4では，S_3 で shift 操作と reduce 操作の両操作の実行が可能である．そこで両操作を並列に実行した結果，shift／reduce 操作を行うプロセスが2つに増えていることがステップ5に示されてる．

ステップ6では，S_3 で規則 A→AA を使用して reduce 操作を行うプロセスと，S_1 で A→a を使用して reduce 操作を行うプロセスが同じS_3 へ遷移し，しかも同じパケット $<S_2, A, 3, 2>$ を生成するので，ステップ7では2つのプロセスを1つにまとめている．この時ステップ7の S_3 では，ステップ2の $<S_0, A, 1, 1>$ とステップ5の $<S_0, A, 2, 1>$ という2つのパケットを指すポインタを持つ．この時 $<S_2, A, 3, 2>$ では **Fig. 3** のような2つの異なった木 (treel, tree2) を持つ．

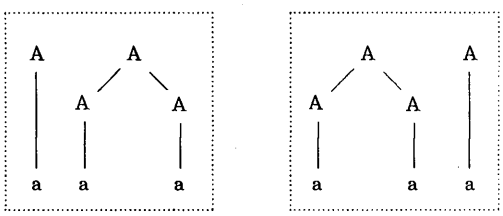


Fig. 3　Parsing trees stored at $<S_2, A, 3, 2>$

ステップ7の S_3 の $<S_2, A, 3, 2>$ は2つのポインタを持つために，reduce 操作によって shift／reduce 操作を行うプロセスが2つ生成されるが，2つとも同じ S_2 へ行き，$<S_0, A, 3, 1>$ を生成するので1つにまとめられる．

さて，上の方法に対して入力文の長さ n を使って，時間とプロセス数に関する簡単な評価を行う．まず解析時間を評価すると $O(n)$ である．なぜなら，入力に同期しないために解析時間は一つの構文木を作成する時間ですむ．一つの構文木作成時間は木の節の数に比例し，その数は $O(n)$ である．木の節の作成は shift 操作と reduce 操作によって行われるが，shift 操作はチェーンリストに対して新たなパケットを接続するに過ぎないので明らかに $O(1)$ であり，またパケットのもつポインタに対してプロセスを割り当てて reduce 操作を独立動作にしているので reduce 操作にかかる時間も明らかに $O(1)$ である．以上から解析時間は $O(n)$ である．

次にプロセス数を評価すると $O(n^3)$ である．なぜなら，あるパケット $<S_i, X, k, f>$ がポインタで指すパケットを $<S_j, Y, k', f'>$ とする．この時 S_j の個数は高々状態数Cであり，Y は S_j によって決まり，$1 \leqq k' \leqq k$ で，$f'=f-1$ で，f' は f が決まれば一意に決まる．従って reduce 操作を独立動作にするために一つのパケット当りに用意すべきプロセスの個数は $O(n)$ である．パケットの数を求めると，S_i は高々状態数C個であり，k，f に依存するのでパケットの数は $O(n^2)$ である．以上よりプロセス数は $O(n^3)$ である．しかしこの方法は，逐次処理で $O(n^3)$ である[3]ことを考えると，$O(n) \times O(n^3) = O(n^4)$ となるためあまり望ましいとはいえない．そこで時間のオーダーを変えずにプロセス数をさらに削減する方法を次節で述べる．

3.2　$O(n^2)$ にプロセス数を削減するアルゴリズム

3.2.1　チェーンリスト内の入力数の統一

前節の方法では，一つのパケットが持つポインタの数が $O(n)$ となった．そこで時間のオーダーを変えずにポインタ数を定数個に抑えることができれば，プロセス数を $O(n^2)$ で抑えることができる．これを実現するために，チェーンリスト内のパケットの入力数を統一することを考える．reduce 操作の時に必要になるのはパケットの入力数ではなく状態番号であるので入力数を統一しても解析に影響はでない．この入力数の統一は，shift 操作にプロセスからプロセスへのポインタのコピーによって行う．

再び文法Gを例にとる．前節の方法では状態 S_2 から S_1 へ shift 操作によって遷移する時，

$<S_0, A, 1, 1> \leftarrow <S_2, a, 2, 2>$

を作成していたが，この時パケット $<S_0, A, 2, 1>$ を生成し，

$<S_0, A, 2, 1> \leftarrow <S_2, a, 2, 2>$

を作成する．また S_1 から S_3 へ遷移して

$<S_0, A, 2, 1> \leftarrow <S2, A, 2, 2>$

を作成した後，S_3 から S_1 への遷移時に shift 操作によってパケット $<S_3, a, 3, 3>$ を生成する時，まず

$<S_2, A, 3, 2> \leftarrow <S_3, a, 3, 3>$

を作成し，そして $<S_2, A, 2, 2>$ が $<S_0, A, 3, 1>$ を

指すポインタを $<S_2, A, 3, 2>$ に渡して

$<S_0, A, 3, 1> \leftarrow <S_2, A, 3, 2> \leftarrow <S_3, a, 3, 3>$

を作成する．

shift 操作時のポインタのコピーはチェーンリストの先頭から順々にするのではなく，ポインタをもつパケット $<S_i, X, k, f>$ のプロセスが，対応するパケット $<S_i, X, k+1, f>$ のプロセスに対してそれぞれ並列に渡す（**Fig. 4** 参照）．これを行うために，ポインタを持ったパケットに対応するプロセスを常に活性状態にしておく．このようにすればポインタのコピーは O(1) で実現でき，しかも一つのパケットがもつポインタの数を高々状態数Cに抑えることができる．

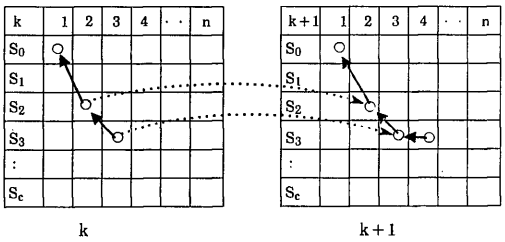


Fig. 4　Pointer copying in shift operation

このポインタのコピーの手続きは以下のようになる．

(1) パケット $<S_i, X, k, f>$ のプロセスが shift 操作時に，入力数 k のパケットのプロセス（以後入力数 k のプロセスと略記）の全てを管理する親プロセスにコピー指示を送る．コピー指示を受け取った親プロセスは入力数 k のプロセス全てにコピー指示を送る．

(2) コピー指示を受け取った入力数 k のプロセスは，自分のパケット $<S_i, X, k, f>$ に対応する $<S_i, X, k+1, f>$ のプロセスにポインタを渡す．この時渡すポインタは，$<S_i, X, k, f>$ のポインタが $<S_j, Y, k, f-1>$ を指している時，$<S_j, Y, k+1, f-1>$ を指すように変えたものである．

複数のプロセスを一つにまとめること，及び複数のチェーンリストによるパケットの共有を許すことのため，reduce 操作時に必要のないポインタをアクセスすることが心配される．その対策として生成規則を状態番号の列で表すことにする．これにより reduce 操作で使うべきポインタを知ることができる．

例えばパケット $<S_4, D, 5, 5>$ のプロセスが，文法規則 A→B C D を使用して reduce 操作を行う時，チェーンリストが **Fig. 5** のようになっていたとしても，

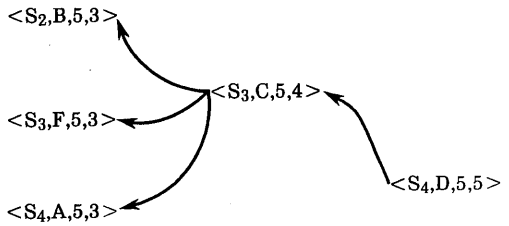


Fig. 5　An example of a packet having some pointer at a chain list.

文法規則が状態番号列 $(S_2 \rightarrow S_2\ S_3\ S_4)$ に変換されていれば，$<S_3, C, 5, 4>$ から $<S_3, F, 5, 3>$ や $<S_4, A, 5, 3>$ へでているポインタをアクセスすることがなくなるので，間違った木を作成する可能性が消える．

ここで，生成規則の列を状態番号の列で書き直す手続きは以下のようになる．

(1) $A \rightarrow X$ で，$[A \rightarrow \cdot X] \in S_i$ ならば $S_i \rightarrow S_i$ とする．

(2) $A \rightarrow \alpha XY\ \beta$ で，$[A \rightarrow \alpha \cdot XY\ \beta] \in S_i$ かつ $[A \rightarrow \alpha X \cdot Y\ \beta] \in S_j$ かつ $[A \rightarrow \alpha\ XY \cdot \beta] \in S_k$ かつ

$GOTO(S_i, X) = S_j$ かつ $GOTO(S_j, Y) = S_k$ ならば $A \rightarrow \alpha S_i S_j\ \beta$ とする．ここでもし $\alpha = \epsilon$ ならば，$S_i \rightarrow S_i S_j\ \beta$ とする．

例）文法Gから状態番号列を作成する

$[A \rightarrow \cdot A\ A] \in S_0, S_2, S_3$ …(1)

$[A \rightarrow \cdot A\ A] \in S_2, S_3$ …(2)

$[A \rightarrow \cdot a] \in S_0, S_2, S_3$ …(3)

$GOTO(S_0, A) = S_2,\ GOTO(S_0, a) = S_1$ …(4)

$GOTO(S_2, A) = S_3,\ GOTO(S_2, a) = S_1$ …(5)

$GOTO(S_3, A) = S_3,\ GOTO(S_3, a) = S_1$ …(6)

(1)，(2)，(4)より，$A \rightarrow A\ A ==> S_0 \rightarrow S_0\ S_2$

(1)，(2)，(5)より，$A \rightarrow A\ A ==> S_2 \rightarrow S2\ S_3$

(1)，(2)，(6)より，$A \rightarrow A\ A ==> S_3 \rightarrow S_3\ S_3$

(3)より　　$A \rightarrow a\ ==> S_0 \rightarrow S_0,\ S_2 \rightarrow S_2,\ S_3 \rightarrow S_3$

よって $A \rightarrow A\ A = \{S_0 \rightarrow S_0\ S_2, S_2 \rightarrow S_2\ S_3, S_3 \rightarrow S_3\ S_3\}$

$A \rightarrow a\ = \{S_0 \rightarrow S_0, S_2 \rightarrow S_2, S_3 \rightarrow S_3\}$

このように文法規則を状態番号の列に変換することによってパケットの（非）終端記号成分は不用となるので，3.4節以降ではパケットを $<S_i, k, f>$ のように変える．

3.3　構文木の作成

shift 操作の時にポインタのコピーを行ったが，そ

の時に構文木を蓄えているメモリアドレスを指すポインタを渡す．メモリは同時読みだし可能，同時書き込み不可の共有メモリで，記号を格納する箱（S部と略す）とポインタを格納する箱（P部と略す）の2つの箱を持っており，パケット自体がメモリの絶対アドレス

Table 2　A configuration of CREW (Concurrent-Read, Exclusive-Write) memory

address　　：　　memory
$<S_i, X, k, f>$　：[Symbol | Pointer]

Table 3　Parsing process of an input sentence "aaaa", and contents of CREW memory

step		address：[Symbol | pointer]
0	S_0：< >	< > : [|]
1	S_1：$<S_0,1,1>$	$1<S_0,a,1,1>$: [a(0,1) |]
2	S_2：$<S_0,1,1>$	$2<S_0,A,1,1>$: [A(1S) |]
3	S_1：$<S_2,2,2>\rightarrow<S_0,2,1>$	$3<S_2,a,2,2>$: [a(1,2) |] $4<S_0,A,2,1>$: [| 2] $5<S_0,a,2,1>$: [| 1]
4	S_3：$<S_2,2,2>\rightarrow<S_0,2,1>$	$6<S_2,A,2,2>$: [A(3S) |]
5	S_1：$<S_3,3,3>\rightarrow<S_2,3,2>\rightarrow<S_0,3,1>$ S_2：$<S_0,2,1>$	$7<S_3,a,3,3>$: [a(2,3) |] $8<S_2,A,3,2>$: [| 6] $9<S_2,a,3,2>$: [| 3] $10<S_0,A,3,1>$: [| 4] $11<S_0,a,3,1>$: [| 5] $4<S_0,A,2,1>$: [A(4P6S) | 2]
6	S_3：$<S_3,3,3>\rightarrow S_1$：$<S_2,3,2>\rightarrow<S_0,3,1>$	$12<S_3,A,3,3>$: [A(7S) |] $9<S_2,a,3,2>$: [a(2,3) [| 3] $10<S_0,A,3,1>$: [| 4] $11<S_0,a,3,1>$: [| 5]
7	S_1：$<S_3,4,4>\rightarrow<S_3,4,3>\rightarrow<S_2,4,2>\rightarrow<S_0,4,1>$ S_3：$<S_2,3,2>\rightarrow<S_0,3,1>$	$13<S_3,a,4,4>$: [a(3,4) |] $14<S_3,A,4,3>$: [| 12] $15<S_3,a,4,3>$: [| 7] $16<S_2,A,4,2>$: [| 8] $17<S_2,a,4,2>$: [| 9] $18<S_0,A,4,1>$: [| 10] $19<S_0,a,4,1>$: [| 11] $8<S_2,A,3,2>$: [A(9S), A(8P12S) | 6]
8	S_3：$<S_3,4,4>\rightarrow S_1$：$<S_3,4,3>\rightarrow<S_2,4,2>\rightarrow<S_0,4,1>$ S_2：$<S_0,3,1>$	$20<S_3,A,4,4>$: [A(13S) |] $15<S_3,a,4,3>$: [a(3,4) | 7] $16<S_2,A,4,2>$: [| 8] $17<S_2,a,4,2>$: [| 9] $18<S_0,A,4,1>$: [| 10] $19<S_0,a,4,1>$: [| 11] $10<S_0,A,3,1>$: [A(10P8S) | 4]
9	S_3：$<S_3,4,3>\rightarrow S_1$：$<S_2,4,2>\rightarrow<S_0,4,1>$	$14<S_3,A,4,3>$: [A(15S), A(14P20S) | 12] $17<S_2,a,4,2>$: [a(3,4) | 9] $16<S_2,A,4,2>$: [| 8] $18<S_0,A,4,1>$: [| 10] $19<S_0,a,4,1>$: [| 11]
10	S_3：$<S_2,4,2>\rightarrow<S_0,4,1>$	$16<S_2,A,4,2>$: [A(17S), A(16P14S) | 8]
11	S_2,：$<S_0,4,1>$	$18<S_0,A,4,1>$: [A(18P16S) | 10]

を表している（**Table 2** 参照）.

shift 操作時に終端記号を格納する時，それと一緒に入力文に於ける位置情報も格納する．reduce 操作時の木の作成では記号の代わりにポインタを取り込むが，その時 reduce を始めたパケットからは S 部を指すポインタを取り込み，それ以外の（$A\rightarrow\alpha$，$|\alpha|\geqq 2$ の規則を使用して reduce をする）場合には P 部を指すポインタを取り込む．構文木作成のための記録形式を次のように定義する．

(1)　アドレスNの S 部を $N.S$, P 部を $N.P$ で表し，N だけの時には $N.S$ と $N.P$ の両方を示す．

(2)　$A\ (N_1.\ P\cdots N_{m-1}.\ S)$ は，記号Aが m＋1 個の記号を reduce して生成されたことを示す．

(3)　shift 操作の時，$<S_i, X, k, f>$ のアドレスNを $<S_i, X, k+1, f>$ のアドレスN'の P 部に格納する．

(4)　複数のプロセスが同じパケット $<S_i, X, k, f>$ をアクセスする時，同じアドレスに複数の記号を書き込み，コンマ（,）で結ぶ．

以上の定義にしたがって解析終了時に作成されたものは全ての構文木の圧縮された形である．それから，全構文木を取り出す時には，まず木の根の部分から葉の方へ展開し，終端記号の入力文における位置情報を取り出す．そして，その情報から他の部分解析木との適合しうる組合せを全て計算して作成する．次節では入力文を解析し，全ての構文木を取り出す例を示す．

3.4　解　析　例

3章で使用した文法 G を使って文" aaaa $ " を解析する時のチェーンリストと構文木の記録状況を **Table 3** に示す．**Table 4** に解析終了時のメモリの中身を示す．また，全ての構文木を取り出したものを **Fig. 6** に示す．

3.5　不要になったプロセスの消去

チェーンリスト内のパケットの入力数を統一するために，ポインタを持つパケットは全て活性状態にしておかなければならないが，この中には解析には利用されなくなった無駄なプロセスも存在する．例えば Table 3 に示した解析例のステップ6では，2つのチェーンリストの入力数は全て3であるから，入力数が2以下のプロセスは必要ない．このような不要なプロセスを消去することは，Lisp などの言語で行われているゴミ集めと同じで，実際上必要なものである．こ

Table 4　Contents of CREW after parsing

$1<S_0,a,1,1>$: $[a(0,1)$	]
$2<S_0,A,1,1>$: $[A(1S)$ [	]
$3<S_2,a,2,2>$: $[a(1,2)$	]
$4<S_0,A,2,1>$: $[A(4P6S)$	2]
$5<S_0,a,2,1>$: [	1]
$6<S_2,A,2,2>$: $[A(3S)$	]
$7<S_3,a,3,3>$: $[a(2,3)$	]
$8<S_2,A,3,2>$: $[A(9S), A(8P12S)$	6]
$9<S_2,a,3,2>$: $[a(2,3)$	3]
$10<S_0,A,3,1>$: $[A(10P8S)$	4]
$11<S_0,a,3,1>$: [	5]
$12<S_3,A,3,3>$: $[A(7S)$	]
$13<S_3,a,4,4>$: $[a(3,4)$	]
$14<S_3,A,4,3>$: $[A(15S), A(14P20S)$	12]
$15<S_3,a,4,3>$: $[a(3,4)$	7]
$16<S_2,A,4,2>$: $[A(17S), A(16P14S)$	8]
$17<S_2,a,4,2>$: $[a(3,4)$	9]
$18>S_0,A,4,1>$: $[A(18P16S)$	10]
$19<S_0,a,4,1>$: [	11]
$20<S_3,A,4,4>$: $[A(13S)$	]

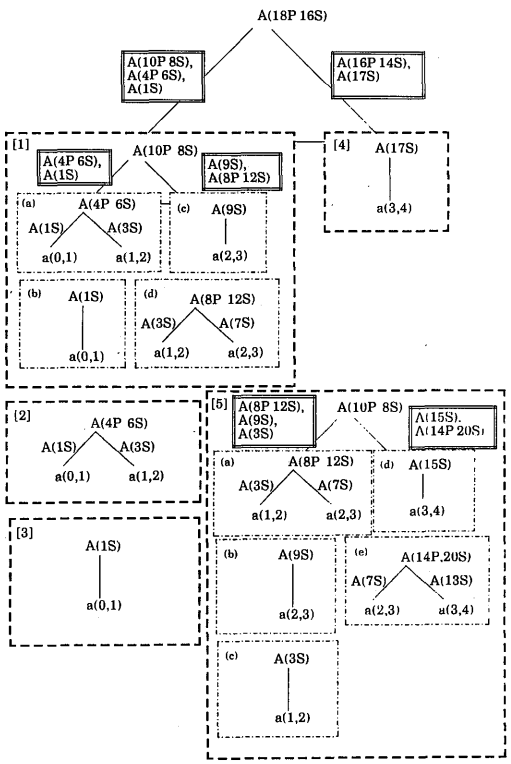


Fig. 6　All parsing trees generated from [1] {(a)-(c), (b)-(d)}-[4], [2]-[5] {(b)-(d)}, [3]-[5] {(a)-(d), (c)-(e)}

の手続きは紙面の都合上省略し，別の機会に譲る．

4. 評　　価

評価には，入力文の長さnを使用して行う．

(1) 時間 $O(n)$

(a) shift 操作：shift 操作は $<S_i, k+1, f>$ と $<S_j, k+1, f+1>(1 \leqq k)$ をポインタで結び，入力数 k のポインタをもつプロセスが，ポインタを入力数 $k+1$ の対応するパケットのプロセスに渡すだけである．入力数kのプロセスを管理する親プロセスにポインタのコピーを指示する時間は $O(1)$ で，この親プロセスから入力数kの各プロセスがコピー指示を受け取る時間も $O(1)$ 時間である．また一つの $<S_i, k, f>$ がもつポインタの数は高々状態数C個なので，ポインタのコピーにかかる時間は高々 $O(1)$ である．以上から，shift 操作にかかる時間は $O(1)$ ですむ．

(b) reduce 操作：shift 操作の時にポインタを全てコピーするため，たどっていく $<S_i, k, f>$ の入力数は全て同じであることが保証される．reduce 操作は $<S_i, k, f>$ から文法規則 $A \rightarrow \alpha$ を使ってポインタをたとる操作である．$<S_i, k, f>$ のもつポインタの数は高々C個である．$<S_i, k, f>$ のプロセスは複数の状態における shift／reduce 操作の要求を受け付けるが，この要求の数も高々状態数C個である．$|\alpha|$ の大きさは文法に依存し，その最大のものを C' とすれば，以上から reduce 操作にかかる時間は高々 $C^2 \times C'$ である．以上より reduce 操作にかかる時間も $O(1)$ ですむ．

(a)，(b)より shift／reduce の各操作とも $O(1)$ である．また，本手法は入力に非同期な解析を行うので，他のプロセスが reduce し終えるのを待つ必要がない．以上より解析時間は $O(n)$ である．

(2) プロセス数 $O(n^2)$

プロセスは1つのパケット $<S_i, k, f>$ に1つ割り当てている．S_i は状態数Cで制限され，k, f はそれぞれ $O(n)$ である．よってプロセス数は $O(n^2)$ である．

(3) メモリ領域 $O(n^2)$

1つのパケットに割り当てられるメモリの大きさは，状態の要素であるマーカ付規則によって決まるもので入力文の長さとは無関係であり，高々（非）終端記号数である．よって，メモリ領域はパケットの数に比例

Table 5　Sample grammars $G_1 \sim G_{10}$

G_1	S→A b, A→A b | a
G_2	S→a B, B→a B | b
G_3	S→a b | a S b
G_4	S→A B, A→A b | a, B→b B | B d | b c
G_5	X→a | X b | Y a, Y→e | Y d Y
G_6	A→x A x | y A y | x x | y y
G_7	S→JP VP, VP→v, JP→NP j, NP→a n | d n | n
G_8	S→A B | C X B, A→D X, C→c, X→a A b | a C X b | x, D→d, B→b
G_9	S→A B | C X B, A→C X, C→c, X→a A b | a C X b | a C X | x, B→b
G_{10}	A→A A | a

Table 6　The number of processes and steps for $G_1 \sim G_{10}$

	input	Processes (pn^2+qn+r)						steps $(sn+t)$	
		NGC			GC				
		p	q	r	p	q	r	s	t
G_1	$a^{n-1}b$	0	2.0	−1.0	0	0	2.0	2.0	0
G_2	$a^{n-1}b$	0.50	0.50	0	0	1.0	0	2.0	0
G_3	$a^{\frac{n}{2}}b^{\frac{n}{2}}$	0.50	0.50	0	0	0.50	2.0	1.5	0
G_4	$ab^{n-3}cd$	0.50	1.50	−4.0	0.50	−2.0	4.0	2.0	0
G_5	$(ed)^{\frac{n-3}{2}}ab^2$	0.50	1.0	1.0	0.17	0.33	0	2.0	2.0
	$(ed)^2ab^{n-4}$	6.0	0	−4.0	0	0	7.0	2.0	4.0
G_6	$(xx)^{\frac{n}{2}}$	1.0	−2.0	3.0	0.50	−0.50	−1.0	1.5	0
G_7	$(nj)^{\frac{n-1}{2}}v$	0.50	0.33	0	0.33	−3.0	0	2.5	−0.50
G_8	$(ca)^{\frac{n}{3}-1}cxb^{\frac{n}{3}-1}b$	0.50	0.50	0	0	0.67	1.0	1.7	2.0
G_9	$(ca)^{\frac{n}{3}-1}cxb^{\frac{n}{3}-1}b$	0.56	0.67	0	0.17	1.0	0	1.7	3.0
G_{10}	a^n	0.50	0.50	0	0.25	0.50	0	3.0	−1.0

し $O(n^2)$ である．

5. 実験結果

実験に使用した文法 G_1～G_{10} を **Table 5** に示す．このうち G_1～G_5 は文献 (2) から引用した．G_1～G_4 の文法は逐次処理[2]で，解析時間が $O(n)$ のものである．これらに対してプロセス数がどの程度増えるかを実験によって調べた結果を **Table 6** に示す．ここで n は入力文の長さを表す．GC はゴミ集めを行った結果で，NGC の左はゴミ集めをしなかったものを表している．

表の中の式は，各文法に対してシミュレーションしてデータを10個以上求め，それらから数式を予想したものである．ステップ数に関する式は，shift／reduce の各操作を $O(1)$ として計算して求めたものである．プロセス数のデータは解析途中でプロセスが最大に膨れた時にカウントしたものである．G_1 についてはゴミ集めをした場合，プロセス数が常に2という結果を得たが，他の G_2, G_3, G_4 に対しては $O(n)$ から $O(n^2)$ という結果を得た．またゴミ集めをしなかった場合，プロセス数は全て $O(n^2)$ となり，曖昧さのない文法も曖昧さの多い文法もオーダー評価では同じ結果になった．

ステップ数では文献 (2) の手法より定数倍速いという結果を得た．これは *LR* 法の特性をからきたもので，文献 (5) などもそのことを示している．しかしプロセス数が曖昧さのない文法を処理する時でさえ $O(n)$ ～ $O(n^2)$ 必要になることから，この方法をそのまま逐次処理の手法に落とすのは問題がある．また入力に同期しない解析を行っているために，同じ解析を時間をずらして行うようなプロセスが存在することがある．例えば文法 G_9 でそのようなことが起こった．しかし shift／rebuce の操作ごとに同期をとっているので，必要となるプロセス数のオーダーが増えることはない．

6. 他の手法との比較

ここでは，Tomita の手法に類似した手法[6][7]，及び PAX[8] と本手法について比較を行う．Tomita の手法[5] に対して並列化したものに文献 (6,7) などがあるが，これらの解析時間は全て $O(n^2)$ である．またプロセス数は $O(n^2)$ となる．なぜなら，各状態を $<S_i, k>$ (S_i は状態番号，k は入力数) で表すと，S_i は文法に依存する定数であり，k は入力数に依存するため，各状態は最悪 $O(k)$ 個の状態 $<S_j, k\text{-}i>$ $(i \geqq 1)$ を指すポインタを持つ．そのポインタの1つ1つにプロセスを割り当てて reduce 操作を独立動作にするとプロセス数は $O(n^2)$ となる．即ち，3.1節で述べた手法の場合と同様なことがおこる．また必要となるメモリも $O(n^2)$ である．ここで，文献 (6,7) ではグラフ構造スタッフを使用しているが，チェーンリストも基本的には同じであるので，証明はチェーンリストを使用して行った．

PAX[8] の場合，各（非）終端記号が独立に動作するが，左から右へ識別子の受渡しをしているので解析時間は $O(n)$ である．またプロセス数は全ての構文木を作成するために $O(C^n)$ (C は定数) である．

Table 7 Comparison of our method with Tomita's and PAX

	time	process	memory
Tomita's	$O(n^2)$	$O(n^2)$	$O(n^2)$
PAX	$O(n)$	$O(C^n)$	—
Ours	$O(n)$	$O(n^2)$	$O(n^2)$

これらの比較結果を **Table 7** に示す．ここで Tomita のパーサに対して並列化したものをまとめて Tomita's と略記する．また比較の要素として入力文の長さ n を使用している．

7. まとめ

本稿では解析時間が $O(n)$，プロセス数が $O(n^2)$ のアルゴリズムについて述べた．このアルゴリズムは各操作を同期させて解析を行うことのできる SIMD 型の並列マシン上で動作する．

評価では入力文の長さを使用し，時間，プロセス数，メモリ領域とも定数成分をかなり大きめに求めた．実際にはこれらの数よりかなり小さいことが予想される．定数部分に対する評価を行うことは並列アルゴリズムとしては重要である[9]ため，reduce にかかる時間などに関するより精密な数を求めることは今後の課題である．

謝辞　貴重な御助言を頂いた九州大学理学部の宮野助教授に，また情報認識研究室の各位に感謝します．

参 考 文 献

1) D. E. Knuth：On the Translation of Languages from Left to Right, Information and Control, **Vol. 8**, no. 6, pp. 607-639 (1965)

2) J. Earley：An Efficient Context-Free Parsing Algorithm, Comm. ACM, **Vol. 13**, No. 2, pp. 95-102 (1970)

3) 峯，谷口，雨宮：文脈自由文法の並列構文解析，信学技報 (NLC)，89-113, pp. 1-8 (1989.6)

4) 峯，谷口，雨宮：$O(n)$ 時間・$O(n^2)$ プロセス数の並列構文解析，情報処理学会第39回全国大会，(発表予定)

5) M. Tomita：An Efficient Augumented-Context-Free Parsing Algorithm, Computational Linguistic, **Vol. 13**, No. 1-2 (Jan-Jun. 1987)

6) 安留，青江：自然言語の曖昧構文解析に対する並列処理情報研報 (SF, PL)，**27-12**, pp. 109-118 (1988.12)

7) 沼崎，田村，田中：並列論理型言語を用いた自然言語処理のための LR 構文解析アルゴリズムの実現，Proceeding of the Logic Programming Conference'89, pp. 183-192 (1989.7)

8) Y. Matsumoto：A Parallel Parsing System for Natural Language Analysis, New Generation Computing, pp. 63-78 (May-1987)

9) 宮野：並列化と限界，Proceeding of the Logic Programming Conference '89, pp. 11-20, (1989.7)