JAIST Repository
https://dspace.jaist.ac.jp/

Title	品詞間接続制約のLR構文解析表への組み込みの局所性の解消
Author(s)	野呂，智哉; 田中，穂積; 橋本，泰一; 白井，清昭
Citation	自然言語処理，16(3): 81-101
Issue Date	2009-07-10
Type	Journal Article
Text version	publisher
URL	http://hdl.handle.net/10119/9166
Rights	Copyright (C) 2009 言語処理学会. 野呂智哉，田中穂積，橋本泰一，白井清昭，自然言語処理，16(3)，2009，81-101.
Description	

JAIST
JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

品詞間接続制約のLR構文解析表への組み込みの局所性の解消

野呂　智哉[†]・田中　穂積[††]・橋本　泰一[†††]・白井　清昭[††]

LR構文解析表（LR表）を作成する際，CFG規則による制約だけでなく品詞（終端記号）間の接続制約も同時に組み込むことによって，LR表中の不要な動作（アクション）を削除することができる．それにより，接続制約に違反する解析結果を受理しないLR表を作成できるだけでなく，LR表のサイズを縮小することも可能であり，構文解析の効率の向上が期待できる．これまでにも接続制約の組み込み手法はいくつか提案されているが，従来手法では，注目する動作の前後に実行され得る動作を局所的に考慮するため，削除しきれない動作が存在する．そこで，本論文では新しい組み込み手法を提案する．提案手法では，初期状態から最終状態までの全体の実行すべき動作列（アクションチェイン）を考慮し，接続制約を組み込む．評価実験の結果，従来手法と比較して，不要な動作をさらに約1.2%削減でき，構文解析所要時間は約2.4%短縮できることが分かった．最後に，提案手法の完全性について考察する．

キーワード：一般化LR構文解析，品詞間接続制約，文脈自由文法，局所性解消，アクションチェイン

Globalization of Incorporating Adjacent Symbol Connection Constraints into an LR Parsing Table

Tomoya Noro[†], Hozumi Tanaka[††], Taiichi Hashimoto[†††] and Kiyoaki Shirai[††]

Adjacent symbol connection constraints (ASCCs) are very useful for not only morphological analysis of non-segmenting language such as Japanese language, but also for continuous speech recognition of any language. By incorporating ASCCs into an LR parsing table, it is possible to reduce the size of the table, as well as reject any locally implausible parsing results. Although several algorithms have been proposed, they cannot remove all of the unnecessary actions because they consider only local context. This paper proposes a new algorithm and show some evaluation results. The proposed algorithm incorporates ASCCs by searching for global action chains from the initial state to the final state. According to the results, the proposed algorithm can remove about 1.2% more actions than a conventional algorithm, and the parsing time can be reduced by about 2.4%. Lastly, we show the completeness of our algorithm.

† 東京工業大学大学院情報理工学研究科, Department of Computer Science, Tokyo Institute of Technology
†† 北陸先端科学技術大学院大学情報科学研究科, School of Information Science, Japan Advanced Institute of Science and Technolgoy
††† 東京工業大学統合研究院, Integrated Research Institute, Tokyo Institute of Technology

Key Words: *Generalized LR Parsing, Adjacent Symbol Connection Constraints, Context-Free Grammar, Globalization, Action Chains*

1　はじめに

インターネットの普及にともない，多種多様な電子情報が至るところに蓄積され，溢れている．我々は，インターネットを介して，時と場所を選ばず，即座にそれらの情報にアクセスすることができるが，その量は非常に膨大である．「情報爆発」というキーワードのもと，わが国でも文部科学省，経済産業省が新しいプロジェクトを立ち上げ，新技術の開発に取り組み始めている．この膨大な量の情報を人手で処理することは，不可能に近い．

情報には文書，画像，音声，動画など様々なものがあるが，自然言語で書かれた文書情報は，その中で最も重要な情報の１つである．文書情報を機械的に処理する技術の研究，言い換えると自然言語処理技術の研究が極めて重要になっているのはそのためである．自然言語処理技術は，2つに大別される．コーパス（統計）ベースの手法とルール（文法規則）ベースの手法である．自然言語処理技術の１つである音声認識の精度のブレイクスルーがあったことにより，最近では，コーパスベースの手法が自然言語処理技術の世界を席巻している．これは網羅性のある文法規則を開発することが困難であったことが主な要因としてあげられる．これに対し，コーパスベースの手法は，そこから得られた統計データに文法規則性が反映されており，コーパスの量を増やすことで，文法規則性をより精密に反映させることができるという考えに基いている．ところが統計データからは陽に文法規則が取り出されるわけではなく，文法規則を取り出し，それをどう改良すべきかは分からない．

文法規則は機械（コンピュータ）で扱うことができる規則でなければならない．多種多様な分野の日本語の文書処理を行う文法規則の数は，およそ数千の規模になると言われている．ところがこのような日本語の文法規則を言語学者ですら作成したという話をまだ聞かない．これに対し，コーパスベースの手法による日本語文の文節係り受け解析の精度は90%に達する (工藤，松本 2002; 内元，関根，井佐原 1999)．これがルールベースの方法が自然言語処理技術の中心ではなくなってきた大きな理由である．

ところが，最近，コーパスベースの自然言語処理法も解析精度に飽和現象が見られる．精度をさらに向上させようとすれば，現存するコーパスの量を１桁以上増やさなくてはならないといわれている．これは，音声認識精度の向上でも問題になりはじめているが，コーパスの量を1桁以上増やすことは容易なことではない．この限界を越える技術として，闇雲にコーパスの量を増やすのではなく，ルールベースの方法を再考すべき時期に来ていると考えている．

本論文では，一般化LR (Generalized LR; GLR) 構文解析 (DeRemer and Pennello 1982; Aho, Sethi, and Ullman 1986; Tomita 1991) に注目する．一般化LR構文解析は，文法 (CFG) 規則を

LR 構文解析表 (LR 表) と呼ばれるオートマトンに変換し，効率的に解析を行う[1]．このLR表には，CFG 規則のほかに品詞（終端記号）間の接続制約 (adjacent symbol connection constraints; ASCCs) を反映させることもできる．品詞間の接続制約を反映させることにより，接続制約に違反する解析結果を受理しない LR 表を作成できるだけでなく，LR 表のサイズ（状態数や動作（アクション）数）を縮小することもでき，その結果，構文解析の使用メモリ量や解析所要時間の削減，統計データを取り入れた場合の解析精度向上の効果の増大が期待できる．品詞間接続制約を CFG 規則に直接反映させることも可能であるが，非終端記号の細分化によって規則数が組み合わせ的に増大し，CFG 作成者への負担や LR 表のサイズの増大を招く．品詞間接続制約の LR 表への組み込み手法は，これまでにも提案されているが (Tanaka, Tokunaga, and Aizawa 1995; Li and Tanaka 1995)，従来の手法では，LR 表中の不要な動作を十分に削除できない問題があった．本論文では新しい組み込み手法を提案し，従来の手法では削除できなかった不要な動作も削除できることを実験により示す．

本論文の構成は以下のとおりである．第 2 節では，まず，一般化 LR 構文解析アルゴリズムを採用している MSLR パーザ (白井，植木，橋本，徳永，田中 2000) について説明し，従来の品詞間接続制約の LR 表への組み込み手法の問題点を述べる．その問題点を踏まえ，第 3 節で新しい組み込み手法を提案し，第 4 節で評価実験を行う．第 5 節では，提案アルゴリズムの完全性について考察を行う．最後に，第 6 節で結論と今後の課題について説明する．

2 MSLR パーザと従来の組み込み手法

本節では，従来の LR 表への接続制約の組み込み手法とその問題点を述べるが，その前に，第 4 節の評価実験で使用する MSLR パーザ (白井他 2000) の原理について概略を説明する．

2.1 MSLR パーザの原理

MSLR (Morpho-Syntactic LR) パーザは，GLR 構文解析アルゴリズムを拡張し，日本語などの分かち書きされていない文の形態素解析と構文解析を同時に行うことのできるパーザである．図 1 に示すように，MSLR パーザは，文法 (CFG) から LR 表を生成し，それを参照しながら入力文の解析を行う．LR 表を生成する段階では，文法のほかに品詞間接続制約を組み込むことも可能である．品詞間接続制約を組み込むことにより，LR 表のサイズを小さくし，解析効率を向上させることができる．また，MSLR パーザは，平文を入力とすることで形態素解析と構文解析を同時に行うことができるが，形態素区切りや品詞，係り受けなどの部分的な制約を入力に加

[1] 一般化 LR 構文解析は，構文解析結果の順序付けに確率一般化 LR モデル (Inui, Sornlertlamvanich, Tanaka, and Tokunaga 2000; Briscoe and Carroll 1993; Charniak 1996; Jelinek 1998) を用いることができるので，ルールベース手法にコーパスベース手法を融合したハイブリッドな方法であるといえる．

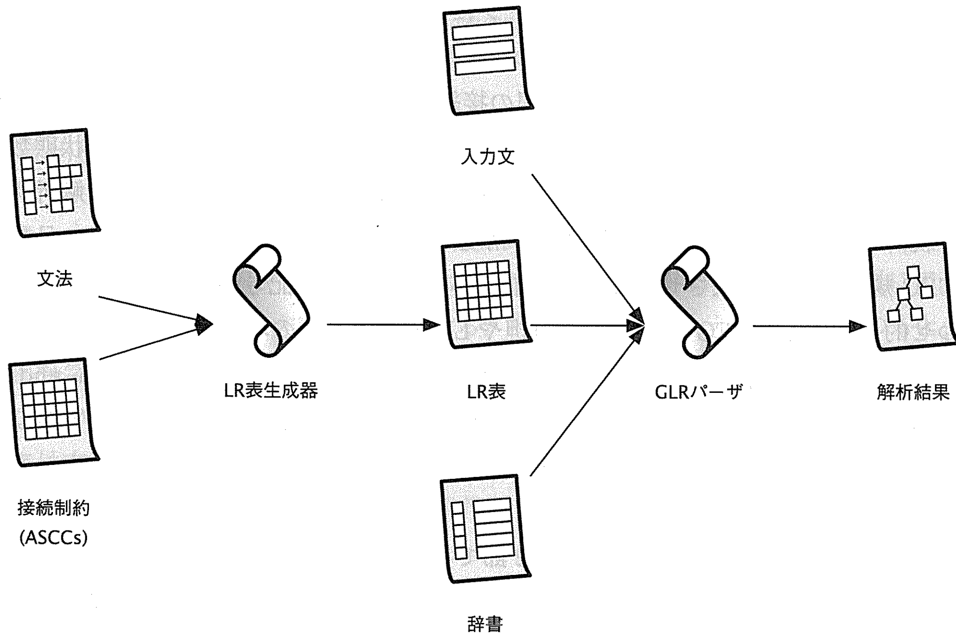


図 1　MSLR パーザの動作の流れ

えて解析を行うこともできる．さらに，確率一般化 LR (Probabilistic Generalized LR; PGLR) モデル (Inui et al. 2000) により，GLR アルゴリズムの枠組みにおいて構文木の生成確率を求めることもできる．

MSLR パーザでは，ε 規則（右辺の記号列長が 0 の規則）を含む文法は扱えない．文法が大規模化するにつれ，文法作成者が予期しない ε 規則の適用や，それによる解析結果の曖昧性の増大が起きるため，MSLR パーザの仕様として，文法に ε 規則は含まれないことを前提としている．本論文でも，ε 規則を含まない文法を前提とする．

2.2　接続制約と接続表

終端記号と文末記号 \$ の集合 $\{t_1, t_2, \ldots, t_n, t_{n+1}(=\$)\}$ の接続制約は，n 行 $n+1$ 列の表（接続表）で表現できる．

$$\mathrm{connect}[t_i, t_j] = \begin{cases} 1 & t_i t_j \text{ の順で接続可能な場合} \\ 0 & t_i t_j \text{ の順で接続不可能な場合} \end{cases}$$

ただし，$1 \leq i \leq n$，$1 \leq j \leq n+1$ である．また，終端記号または非終端記号 X の直後に接続可能な終端記号の集合を返す関数 Connect を以下のように定義する．

$$\mathrm{Connect}(X) = \begin{cases} \{t|\mathrm{connect}[X,t] = 1 \wedge t \in \mathrm{Follow}(X)\} & X\ \text{が終端記号の場合} \\ \bigcup\{\mathrm{Connect}(t) \cap \mathrm{Follow}(X)|t \in \mathrm{Last}(X)\} & X\ \text{が非終端記号の場合} \end{cases}$$

ただし，Follow(X) と Last(X) は，それぞれ CFG の開始記号から展開した場合に非終端記号 X の直後に出現し得る終端記号の集合，X を展開した場合に末尾に出現し得る終端記号の集合を表す．さらに，終端記号または非終端記号列 $\alpha(=\beta Y)$ の場合や，終端記号または非終端記号の集合 Σ の場合は，関数 Connect を以下のように定義する（Y は終端記号または非終端記号）．

$$\mathrm{Connect}(\alpha) = \mathrm{Connect}(Y)$$
$$\mathrm{Connect}(\Sigma) = \bigcup_{X\in\Sigma} \mathrm{Connect}(X)$$

2.3 従来の接続制約組み込み手法

LR 表への品詞間接続制約の組み込み手法には，まず接続制約を考慮しない LR 表を作成してから不要な動作を削除する手法 (Tanaka et al. 1995)，LR 表作成前と作成後の両方で不要動作を削除する手法 (Li and Tanaka 1995) などがある．ここでは，MSLR パーザの LR 表生成器で採用されている 2 つ目の LR 表作成前と作成後の両方で不要動作を削除する手法（Li の手法）について述べる．

LR 構文解析では，LR アイテムを利用して CFG から状態遷移図（goto グラフ）を作成する．Li らは，goto グラフを作成する段階で，接続制約を利用してアイテムの生成を抑制することにより，接続制約を組み込んだ goto グラフを作成する．さらに，接続制約を組み込んだ goto グラフから LR 表を作成した後，接続制約を伝播させることにより，LR 表作成前に削除できなかった動作を削除する．

接続制約を利用した LR(0) アイテムの生成の抑制は，核アイテム $[X \rightarrow \alpha \cdot \beta] \in \mathrm{Goto}(I, Z)$ を closure 展開する際，以下の 2 つの条件を満たす LR(0) アイテムのみを生成することにより行う[2]．

$$\mathrm{Connect}(Z) \cap \mathrm{First}(\beta) \neq \emptyset$$
$$\mathrm{Follow}(X) \cap \mathrm{Connect}(\beta) \neq \emptyset$$

ただし，Goto(I, Z) は，goto グラフにおいて状態 I から終端記号または非終端記号 Z で遷移した先の状態を表す．また，First(β) は，β を展開した場合に先頭に出現し得る終端記号の集合を表す．

接続制約を組み込んだ goto グラフを作成したら，それをもとに LR 表を作成する．この時点

[2] LR(1) アイテム $[X \rightarrow \alpha \cdot \beta; t] \in \mathrm{Goto}(I, Z)$ の場合は，第 2 条件を $t \in \mathrm{Connect}(\beta)$ に置き換える．

で既にいくらかの不要な動作は削除されているが，削除できずに残っている動作もあるため，LR 表作成後に接続制約を伝播させることにより，さらに不要な動作を削除する．具体的には，LR 表中の各動作について，その直前に実行すべき動作が存在しない場合，または直後に実行すべき動作が存在しない場合，その動作を削除する．

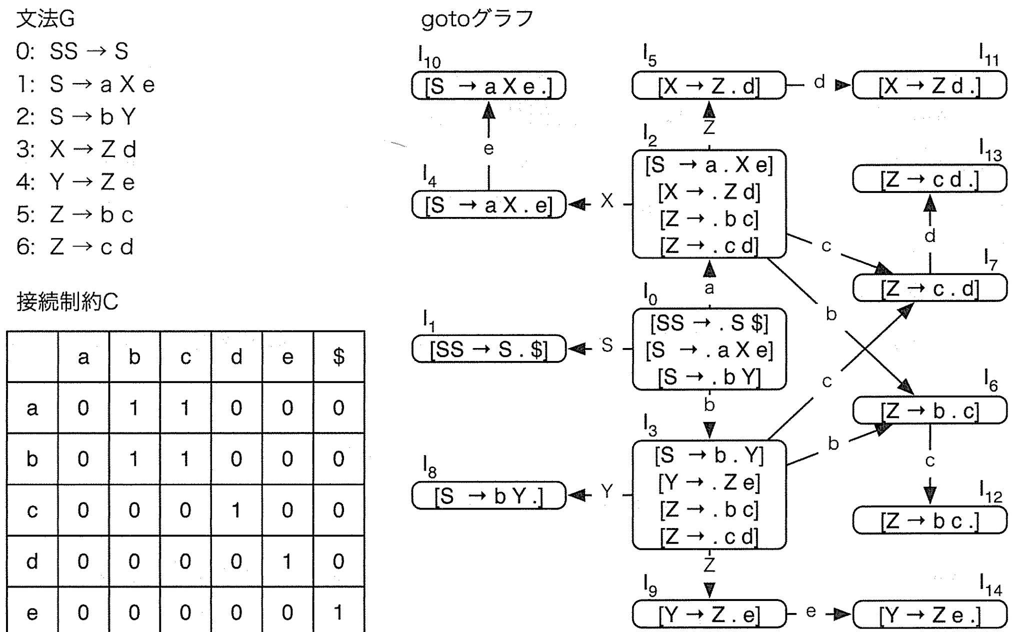
2.4 従来手法の問題点

図 2 に示すような文法 G [3]と接続制約 C（と文法 G から作成される goto グラフ）を例に，従来手法（Li の手法）の問題点を述べる．

Li の手法により作成される LR 表を表 1 に示す．ただし，括弧で囲まれた動作は，接続制約により削除されたものである．ここで，状態 2，先読み c における移動 (shift) 動作 sh_7 に注目する．この動作は，Li の手法では削除されない．

この shift 動作に関連する動作実行列として，以下のような場合が想定される（$(2, c, \mathrm{sh}_7)$ は，状態 2，先読み c における shift 動作 sh_7 を表す）．

$$(2, c, \mathrm{sh}_7) \rightarrow (7, d, \mathrm{sh}_{13}) \rightarrow (13, d, \mathrm{re}_6) \rightarrow (2, Z, \mathrm{goto}_5) \rightarrow (5, d, \mathrm{sh}_{11})$$



	a	b	c	d	e	$
a	0	1	1	0	0	0
b	0	1	1	0	0	0
c	0	0	0	1	0	0
d	0	0	0	0	1	0
e	0	0	0	0	0	1

図 2　CFG と接続制約の例

[3] LR 構文解析では，与えられた文法 G から LR 表を作成する際，便宜的に，非終端記号 SS を G の非終端記号集合に，文末を表す終端記号 $\$$ を終端記号集合に追加し，$SS \rightarrow S\$$ を G に追加する（S は元の G の開始記号）．本論文では，新たに追加する CFG 規則の番号を常に 0 番とする．

表 1　Li の手法により作成される LR 表

	a	b	c	d	e	$	S	X	Y	Z
0	sh_2	sh_3					1			
1						acc				
2		sh_6	sh_7					4		5
3		sh_6	sh_7						8	9
4					sh_{10}					
5				sh_{11}						
6			sh_{12}							
7				sh_{13}						
8						re_2				
9					sh_{14}					
10						re_1				
11					re_3					
12				re_5	(re_5)					
13				(re_6)	re_6					
14						re_4				

一方，以下のような動作実行列も存在する．

$$(3, c, \mathrm{sh}_7) \rightarrow (7, d, \mathrm{sh}_{13}) \rightarrow (13, e, \mathrm{re}_6) \rightarrow (3, Z, \mathrm{goto}_9) \rightarrow (9, e, \mathrm{sh}_{14})$$

接続制約より，終端記号 d は終端記号 e と接続するが，終端記号 d とは接続しないため，前者の実行列は制約に違反する．その結果，$(13, d, \mathrm{re}_6)$ は削除される．しかし，$(7, d, \mathrm{sh}_{13})$ は，もう一方の接続制約を満たす動作実行列に含まれるため，残される．$(2, c, \mathrm{sh}_7)$ は，接続制約を満たすどのような動作実行列にも含まれず，削除すべき動作であるが，次の $(7, d, \mathrm{sh}_{13})$ が残されるため，Li の手法では削除できない．

従来手法では，1 つ先または 1 つ前の動作が存在しないことが判明した場合に，その動作を削除する．この例では，2 つ先の動作が存在するか否かを調べなければ，削除可能かどうかを判断できない．これを一般化すると，1 つ先や 2 つ先だけでなく，n 個先の動作が存在するか否かを調べる必要があり，連続する動作の存在を局所的に調べるだけでは，接続制約に違反する動作を完全に削除することはできない．このような例でも動作を削除できるようにするためには，その動作実行列が最終的に acc 動作に到達可能であるか否かを調べる必要がある[4]．

[4] 動作実行列が（acc 動作に到達できない）無限ループを形成するような文法と接続制約の例も存在する．これは n をどれだけ大きくしても，無限ループ内の動作が削除可能であることを発見できない究極の例である．

3 提案アルゴリズム

初期状態から実行すべき動作を順番に決めていくと，動作の実行列（アクションチェイン）ができる．このアクションチェインがacc動作に到達すれば，解析が成功することになる．一方，実行すべき動作がLR表から決まらないときには，解析が失敗することになる．このアクションチェインは有向グラフ（アクションチェイングラフ）として表現できる．

初期状態からacc動作に至るアクションチェインを成功パスと呼ぶ．成功パス上の動作は，必要な動作としてLR表に残す．提案アルゴリズムでは，アクションチェインを最終状態（acc動作）から逆向きに横型探索によりたどることにより，成功パスを探索する．すなわち開始記号を左辺に持つCFG規則について，その右辺の末尾の記号から順番に展開しながら（最右導出を行いながら）接続制約を満たすか否かをチェックする．

開始記号Sを左辺に持つ$S \rightarrow X_1X_2\ldots X_n$というCFG規則（規則番号を$m$とする）があったとする．gotoグラフには図3(a)に示すような状態とリンクが存在する（開始状態を0とする）．このCFG規則の展開に対応するLR表中の動作は，状態s_n，先読み\$におけるreduce動作re_mとその後の状態0，非終端記号Sにおける状態s_0への遷移であり，この動作をアクションチェインに追加する．そして，右辺の各終端記号または非終端記号について，X_n，X_{n-1}，$\cdots X_1$の順に接続制約を満たすか否かをチェックする．X_nが終端記号の場合，X_nと\$の間の接続制約をチェックする．接続制約を満たすならば，状態s_{n-1}，先読みX_nにおけるshift動作sh_{s_n}をアクションチェインに追加し，X_{n-1}のチェックに移る（先読みはX_nとなる）．X_nが非終端記号の場合は，X_nを左辺とするCFG規則で展開する．このCFG規則が$X_n \rightarrow Y_1Y_2\ldots Y_{n'}$（規則番号$m'$）であるとすると，gotoグラフ中では図3(b)に示すような状態とリンクが存在する．このCFG規則の展開に対応する，状態$s'_{n'}$，先読み\$におけるreduce動作$\mathrm{re}_{m'}$と状態s_{n-1}，記号X_nにおける状態s_nへの遷移をアクションチェインに追加し，$Y_{n'}$，$Y_{n'-1}$，$\cdots Y_1$の順に接

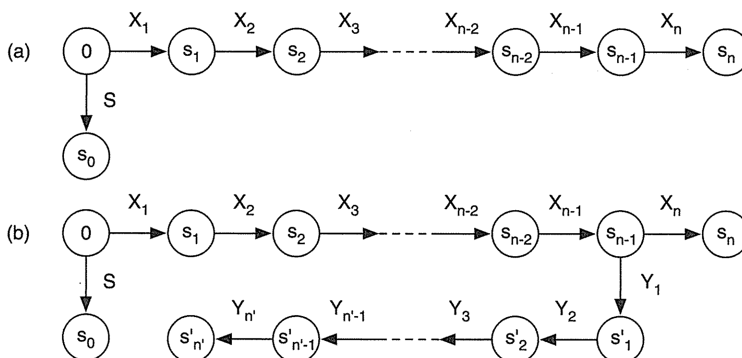


図3　gotoグラフ

続制約を満たすか否かを同様にチェックする．すべてのチェックが完了したら，X_{n-1} のチェックに移る（先読みは Y_1 のチェックで最後にアクションチェインに追加した shift 動作の先読みとなる）．以下，同様に続け，最終的に状態 0 における shift 動作がアクションチェインに追加されたら，それが成功パスとなる．

提案アルゴリズムの概要を図 4 に示す．図中の記法については，以下のとおりである．

$[s, \mathrm{re}_n, la, status]$: 状態 LastState$(s, n)$，先読み la で実行される n 番目の CFG 規則による reduce 動作を表すアクションチェインの要素．reduce 後，状態 s，非終端記号 LHS(n) で状態 Goto$(s, \mathrm{LHS}(n))$ へ遷移する．ただし，$n = 0$ の場合は，reduce 動作ではなく acc 動作を表す要素となる．*status* は要素の処理状態を表す．要素の処理状態には，init（初期状態），wait（待機状態），check（調査中），pass（調査済），end（最終状態）があり，この順番で遷移する（init は飛ばされることもある）．

init: 要素を作成しただけの状態

wait: 次にアクションチェインに追加可能であることを表す状態

check: アクションチェインに追加され，その後，解析開始状態（goto グラフにおける状態 0）に到達可能かどうか（最終的に接続制約を満たすかどうか）を調査中であることを表す状態

pass: 解析開始状態に到達可能であることが判明したことを表す状態

end: 成功パスの要素であることを表す状態

$[s, \mathrm{sh}, la, status]$: 状態 s，先読み la で実行される shift 動作を表すアクションチェインの要素．

Length(n): n 番目の CFG 規則の右辺の長さ．

LHS(n): n 番目の CFG 規則の左辺の非終端記号．

RHS(n, i): n 番目の CFG 規則の右辺の i 番目の終端記号または非終端記号．$1 \leq i \leq \mathrm{Length}(n)$

Rule(A): 非終端記号 A を左辺に持つ規則番号の集合．$\mathrm{Rule}(A) = \{n | \mathrm{LHS}(n) = A\}$

PrevAction(a): reduce 動作または shift 動作 a に続く動作の集合．

State(s, n, i): n 番目の CFG 規則について，状態 s から RHS$(n, 1)$，$\cdots$RHS$(n, i-1)$ を遷移した後の状態．

LastState(s, n): 状態 s から n 番目の CFG 規則の右辺の終端記号または非終端記号列すべてを遷移した後の状態．

LA(s, n): 状態 LastState(s, n) における n 番目の CFG 規則による reduce 動作の先読みの集合．

PrevSym(s): 状態 s への遷移記号の集合．$\mathrm{PrevSym}(s) = \{sym | \mathrm{Goto}(s', sym) = s\}$

PrevState(s, sym): 記号 sym によって状態 s に遷移する状態の集合．$\mathrm{PrevState}(s, sym) = \{s' | \mathrm{Goto}(s', sym) = s\}$

図 4 の（2）では，wait 状態の reduce 動作要素について，その状態を check として，対象となる動作の実行後に解析開始状態まで接続制約に違反することなく到達可能かどうかのチェックを

(1) $[0, \mathrm{re}_0, \$, \mathrm{wait}]$ を作成し，$\mathrm{PrevAction}([0, \mathrm{re}_0, \$, \mathrm{wait}]) = \emptyset$ とする．
(2) 処理状態が wait の reduce 動作要素または shift 動作要素があれば，その中から 1 つを横型探索で選択し，処理状態を check として以下の処理を行う (これを $a = [s, act, la, \mathrm{check}]$ とする)．
- reduce 動作要素 ($act = \mathrm{re}_n$) の場合
  - $1 \leq i \leq \mathrm{Length}(n) - 1$ である各 i について
    - * $sym = \mathrm{RHS}(n, i)$ が終端記号の場合，shift 動作要素 $a' = [\mathrm{State}(s, n, i), \mathrm{sh}, sym, \mathrm{init}]$ を生成．
    - * $sym = \mathrm{RHS}(n, i)$ が非終端記号の場合，各 $n' \in \mathrm{Rule}(sym)$，各 $l \in \mathrm{LA}(\mathrm{State}(s, n, i), n')$ について，reduce 動作要素 $a' = [\mathrm{State}(s, n, i), \mathrm{re}_{n'}, l, \mathrm{init}]$ を生成．
  - $i = \mathrm{Length}(n)$ について
    - * $sym = \mathrm{RHS}(n, i)$ が終端記号の場合，$\mathrm{connect}[\mathrm{RHS}(n, i), la] = 1$ ならば，
      (a) shift 動作要素 $a' = [\mathrm{State}(s, n, i), \mathrm{sh}, sym, \mathrm{wait}]$ を生成．
      (b) $\mathrm{PrevAction}(a') = \{a\}$ とする．
    - * $sym = \mathrm{RHS}(n, i)$ が非終端記号の場合，各 $n' \in \mathrm{Rule}(sym)$ について，
      (a) reduce 動作要素 $a' = [\mathrm{State}(s, n, i), \mathrm{re}_{n'}, la, \mathrm{wait}]$ を生成．
      (b) $\mathrm{PrevAction}(a') = \{a\}$ とする．
- shift 動作要素 ($act = \mathrm{sh}$) の場合
  - $s = 0$ (開始状態) の場合，処理状態を pass とする．
  - $s \neq 0$ の場合，各 $sym \in \mathrm{PrevSym}(s)$ について
    - * sym が終端記号の場合，$\mathrm{connect}[sym, la] = 1$ ならば，各 $s' \in \mathrm{PrevState}(s, sym)$ について，shift 動作要素 $a' = [s', \mathrm{sh}, sym, \mathrm{init}]$ があれば，
      (a) 処理状態を wait とする
      (b) $\mathrm{PrevAction}(a') := \mathrm{PrevAction}(a') \cup \{a\}$ とする．
    - * sym が非終端記号の場合，各 $s' \in \mathrm{PrevState}(s, sym)$，各 $n' \in \mathrm{Rule}(sym)$ について，reduce 動作要素 $a' = [s', \mathrm{re}_{n'}, la, \mathrm{init}]$ があれば，
      (a) 処理状態を wait とする．
      (b) $\mathrm{PrevAction}(a') := \mathrm{PrevAction}(a') \cup \{a\}$ とする．

(3) 処理状態が wait の reduce 動作要素または shift 動作要素があれば，(2) へ戻る．
(4) 処理状態が pass の reduce 動作要素または shift 動作要素があれば，その中から 1 つを選択して処理状態を end とし (これを a とする)，各 $a' \in \mathrm{PrevAction}(a)$ について以下の処理を行う．
- reduce 動作要素 $a' = [s, \mathrm{re}_n, la, \mathrm{check}]$ の場合，$i = Length(n)$ について，
  - $sym = \mathrm{RHS}(n, i)$ が終端記号の場合，$\mathrm{connect}[sym, la] = 1$，かつ，$[\mathrm{State}(s, n, i), \mathrm{sh}, sym, \mathrm{end}]$ があれば，a' の処理状態を pass にする．
  - $sym = \mathrm{RHS}(n, i)$ が非終端記号の場合，$[\mathrm{State}(s, n, i), \mathrm{re}_{n'}, la, \mathrm{end}]$ ($n' \in \mathrm{Rule}(sym)$) があれば，$a'$ の処理状態を pass にする．
- shift 動作要素 $a' = [s, \mathrm{sh}, la, \mathrm{check}]$ の場合，各 $sym \in \mathrm{PrevSym}(s)$ について
  - sym が終端記号の場合，$\mathrm{connect}[sym, la] = 1$ ならば，各 $s' \in \mathrm{PrevState}(s, sym)$ について，$[s', \mathrm{sh}, sym, \mathrm{end}]$ があれば，a' の処理状態を pass にする．
  - sym が非終端記号の場合，各 $s' \in \mathrm{PrevState}(s, sym)$，各 $n' \in \mathrm{Rule}(Sym)$ について，$[s', \mathrm{re}_{n'}, la, \mathrm{end}]$ があれば，a' の処理状態を pass にする．

(5) 処理状態が pass の reduce 動作要素または shift 動作要素があれば，(4) へ戻る．

図 4 アルゴリズム概略

行う．wait 状態の shift 動作要素ならば，その状態を check として，それに先行する init 状態の要素について，その状態を wait とする．ただし，先行する要素が shift 動作要素の場合は，両者の先読み記号の間の接続制約をチェックする．また，goto グラフにおける状態 0 での shift 動作要素の場合は，解析開始状態まで到達可能であることが判明したので，要素の状態を pass とする．図 4 の (4) では，pass 状態の要素について，その状態を end とし，そこから (2) のと

きとは逆に要素をたどり，check 状態の要素が解析開始状態まで到達可能であることを伝えていく（状態を check から pass にする）．最終的に状態が end となった要素の列が成功パスとなる．

図 2 に示す文法 G と接続制約 C に対し，上述のアルゴリズムを適用すると，以下のような手順で処理が進行する．

(1) $[0, \mathrm{re}_0, \$, \mathrm{wait}]$ を作成．

(2) $[0, \mathrm{re}_0, \$, \mathrm{wait}]$ について，処理状態を check に変更し，
$[0, \mathrm{re}_1, \$, \mathrm{wait}]$，$[0, \mathrm{re}_2, \$, \mathrm{wait}]$ を作成．

(3) $[0, \mathrm{re}_1, \$, \mathrm{wait}]$ について，処理状態を check に変更し，
$[0, \mathrm{sh}, a, \mathrm{init}]$，$[2, \mathrm{re}_3, e, \mathrm{init}]$，$[4, \mathrm{sh}, e, \mathrm{wait}]$ を作成．

(4) $[0, \mathrm{re}_2, \$, \mathrm{wait}]$ について，処理状態を check に変更し，
$[0, \mathrm{sh}, b, \mathrm{init}]$，$[3, \mathrm{re}_4, \$, \mathrm{wait}]$ を作成（図 5（1））．

(5) $[4, \mathrm{sh}, e, \mathrm{wait}]$ について，処理状態を check に変更し，
$[2, \mathrm{re}_3, a, \mathrm{init}]$ の処理状態を wait に変更．

(6) $[3, \mathrm{re}_4, e, \mathrm{wait}]$ について，処理状態を check に変更し，
$[3, \mathrm{re}_5, e, \mathrm{init}]$，$[3, \mathrm{re}_6, e, \mathrm{init}]$，$[9, \mathrm{sh}, e, \mathrm{wait}]$ を作成．

(7) $[2, \mathrm{re}_3, e, \mathrm{wait}]$ について，処理状態を check に変更し，
$[2, \mathrm{re}_5, d, \mathrm{init}]$，$[2, \mathrm{re}_6, d, \mathrm{init}]$，$[5, \mathrm{sh}, d, \mathrm{wait}]$ を作成（図 5（2））．

(8) $[5, \mathrm{sh}, d, \mathrm{wait}]$ について，処理状態を check に変更し，
$[2, \mathrm{re}_5, d, \mathrm{init}]$，$[2, \mathrm{re}_6, d, \mathrm{init}]$ の処理状態を wait に変更．

(9) $[9, \mathrm{sh}, e, \mathrm{wait}]$ について，処理状態を check に変更し，
$[3, \mathrm{re}_5, e, \mathrm{init}]$，$[3, \mathrm{re}_6, e, \mathrm{init}]$ の処理状態を wait に変更．

(10) $[2, \mathrm{re}_5, d, \mathrm{wait}]$ について，処理状態を check に変更し，$[2, \mathrm{sh}, b, \mathrm{init}]$，$[6, \mathrm{sh}, c, \mathrm{wait}]$ を作成．

(11) $[2, \mathrm{re}_6, d, \mathrm{wait}]$ について，処理状態を check に変更し，$[2, \mathrm{sh}, c, \mathrm{init}]$ を作成
（$[6, \mathrm{sh}, d, \mathrm{wait}]$ は，$\mathrm{connect}(d, d) = 0$ より作成しない）．

(12) $[3, \mathrm{re}_5, e, \mathrm{wait}]$ について，処理状態を check に変更し，$[3, \mathrm{sh}, b, \mathrm{init}]$ を作成
（$[7, \mathrm{sh}, c, \mathrm{wait}]$ は，$\mathrm{connect}(c, e) = 0$ より作成しない）．

(13) $[3, \mathrm{re}_6, e, \mathrm{wait}]$ について，処理状態を check に変更し，$[3, \mathrm{sh}, c, \mathrm{init}]$，$[7, \mathrm{sh}, d, \mathrm{wait}]$ を作成．

(14) $[6, \mathrm{sh}, c, \mathrm{wait}]$ について，処理状態を check に変更し，
$[2, \mathrm{sh}, b, \mathrm{init}]$ の処理状態を wait に変更．

(15) $[7, \mathrm{sh}, d, \mathrm{wait}]$ について，処理状態を check に変更し，
$[3, \mathrm{sh}, c, \mathrm{init}]$ の処理状態を wait に変更．

(16) $[2, \mathrm{sh}, b, \mathrm{wait}]$ について，処理状態を check に変更し，
$[0, \mathrm{sh}, a, \mathrm{init}]$ の処理状態を wait に変更．

(17) $[3, \mathrm{sh}, c, \mathrm{wait}]$ について，処理状態を check に変更し，

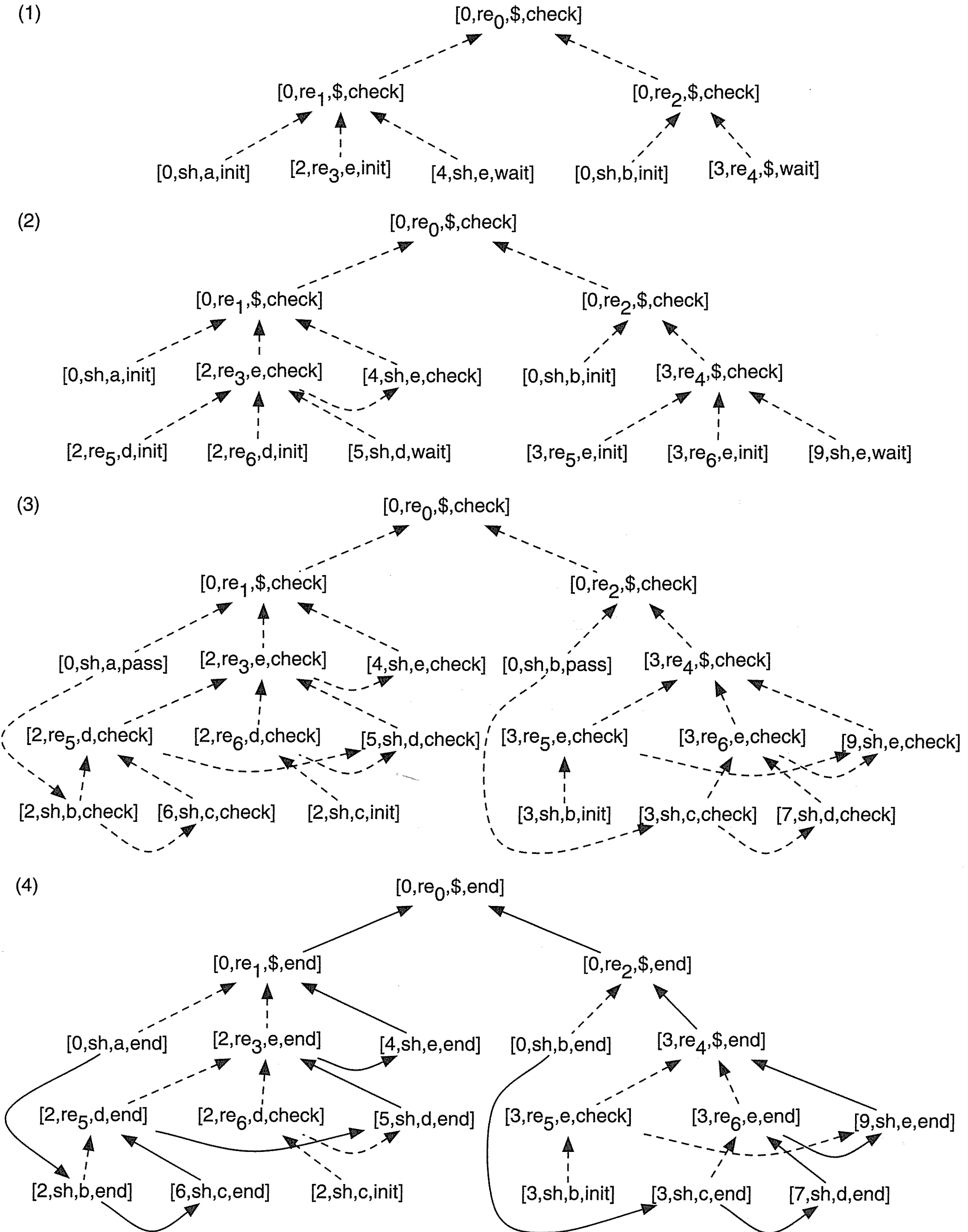


図 5　アクションチェイングラフ作成の経過

[0, sh, b, init] の処理状態を wait に変更．

(18) [0, sh, a, wait] について，処理状態を pass に変更．

(19) [0, sh, b, wait] について，処理状態を pass に変更（図 5（3））．

(20) [0, sh, a, pass] について，処理状態を end に変更し，
[2, sh, b, check] の処理状態を pass に変更．

(21) [0, sh, b, pass] について，処理状態を end に変更し，
[3, sh, c, check] の処理状態を pass に変更．

(22) [2, sh, b, pass] について，処理状態を end に変更し，
[6, sh, c, check] の処理状態を pass に変更．

(23) [3, sh, c, pass] について，処理状態を end に変更し，
[7, sh, d, check] の処理状態を pass に変更．

(24) 以下，同様に処理を続け，処理状態が pass の要素がなくなったら終了（図 5（4））．

アルゴリズムを適用後，処理状態が end である動作要素をたどることにより，成功パスを抽出できる（図 5（4）の実線のリンクが成功パスである）．作成される LR 表を表 2 に示す．また，表 2 において，括弧で囲まれた動作は，Li の手法で削除できず，提案手法により削除されたものを表す．

表 2　提案手法により作成される LR 表

	a	b	c	d	e	$	S	X	Y	Z
0	sh_2	sh_3					1			
1						acc				
2		sh_6	(sh_7)					4		5
3		(sh_6)	sh_7						8	9
4					sh_{10}					
5				sh_{11}						
6			sh_{12}							
7				sh_{13}						
8						re_2				
9					sh_{14}					
10						re_1				
11					re_3					
12				re_5						
13					re_6					
14						re_4				

4　実験と評価

提案手法の効果を調べるため，従来手法との比較実験を行った．コーパスは東工大コーパス 20,190 文（1 文あたり約 23 形態素）(Noro, Koike, Hashimoto, Tokunaga, and Tanaka 2005) を利用する．20,190 文すべてから抽出した文法 G_{all} を使用し，MSLR パーザで構文解析を行う．入力は平文とする．解析結果の順位付けは PGLR モデルにより行う．比較は，提案手法により生成される LR 表と，Li の手法により生成される LR 表，接続制約を組み込まない LR 表の 3 つで行う．

抽出した CFG 規則数は 2,722 規則（非終端記号 294 個，終端記号 412 個）である．各手法により生成された LR 表中の状態数と動作数を表 3 に示す．状態数において，「shift 後」，「reduce 後」とは，それぞれ shift/reduce を実行した直後に到達する状態を指す．PGLR モデルによる確率計算ではこの 2 種類の状態を区別する必要があるため，参考として内訳を示している．また，動作数において，丸括弧，角括弧で囲まれた数字は，それぞれ，コンフリクトが生じる動作の数，PGLR モデルによる確率が 1 ではない動作の数を表す．前者は解析途中での曖昧性の大小の目安に，後者は PGLR モデルによる確率計算の影響の大小（パラメータ数の大小）の目安になる．この表より，状態数にはそれほど大きな差は生じないが，総動作数については，接続制約を組み込まない場合と比較して約 64%削減できていることが分かる．コンフリクトが生じる動作数，PGLR モデルによる確率が 1 にならない動作数は，それぞれ約 56%，71%削減できている．一方，Li の手法と比較すると，総動作数では約 1.2%，コンフリクトが生じる動作数と PGLR モデルによる確率が 1 ではない動作数はどちらも約 1.4%削減できている．

表 3　各 LR 表中の状態数と動作数

		提案手法	Li の手法	接続制約無
状態数	shift 後	414	414	414
	reduce 後	3,359	3,363	3,505
	合計	3,773	3,777	3,919
動作数	shift	38,020 (11,026) [11,206]	38,355 (11,077) [11,261]	77,900 (18,325) [18,509]
	reduce	213,163 (66,706) [74,993]	216,296 (67,767) [76,054]	675,575 (158,198) [279,222]
	goto	41,208	41,379	50,799
	acc	1	1	1
	合計	292,392 (77,732) [86,119]	296,031 (78,844) [87,315]	804,275 (176,523) [297,731]

次に，全 20,190 文を構文解析する際の所要時間（ユーザ CPU 時間）を計測した．結果を表 4 に示す．ただし，計測は Dual-Core Intel Xeon 3 GHz，メモリ 4 GB の環境で行った．結果より，接続制約を組み込まない場合と比較して約 52%，Li の手法と比較して約 2.4%短縮された．

接続制約を組み込まない場合，接続制約を満たさない構文木も解析結果として出力される．速度向上の要因は，接続制約を組み込んだことによる曖昧性の減少にあると考えられる．一方，Li の手法では，接続制約が組み込まれているため，最終的に出力される解析結果は提案手法の場合と同じである．しかし，不要な動作が残っているため，解析途中での無駄な曖昧性（最終的に acc に到達できない解析途中状態）が多く存在する．例えば，第 2.4 節で示した動作実行列の場合，提案手法では，状態 2，先読み記号 c における動作が LR 表中に存在しないことが分かった時点で解析を終了するが，Li の手法では，状態 13，先読み記号 d となるまで解析が継続する．提案手法と Li の手法の解析所要時間の差は，ここで生じる．

最後に，PGLR モデルによる順位付けの評価を 10 分割交差検定により行った．すなわち，全体の 10 分の 9 にあたる 18,171 文を利用してモデルの学習を行い，残りの 2,019 文で評価を行った（文法は G_{all} を使用した）[5]．解析精度は，文正解率により比較した．文正解率は以下のように定義する．

$$\text{文正解率} = \frac{\text{上位 } n \text{ 位までに正解が含まれる文の数}}{\text{解析した文の総数}}$$

ここで「正解」とは，出力された解析木が正解とすべき構文木と完全に一致する場合を指す．結果を表 5 に示す．提案手法では，PGLR モデルによる順位が 1 位の解析木のみを見た場合，接続制約を組み込まない場合と比較して 0.74%向上している．一方，Li の手法と比較すると，1 位の解析木のみでは 0.16%向上しているが，上位 10 位までを見るとほとんど差がなく，LR 表中の不要動作の削除が解析精度に与える影響は大きくないことが分かる．

表 4　構文解析所要時間（ユーザ CPU 時間）

	提案手法	Li の手法	接続制約無
所要時間（秒）	1744.0	1786.5	3615.5

表 5　各順位における文正解率 (%)

順位 (n)	1	2	3	4	5	6	7	8	9	10
提案手法	15.60	22.67	26.77	29.84	31.94	33.73	35.14	36.32	37.27	38.37
Li の手法	15.44	22.66	26.74	29.87	31.92	33.74	35.12	36.30	37.28	38.37
接続制約無	14.86	21.89	25.78	28.93	31.07	32.84	34.24	35.42	36.40	37.53

[5] 20,190 文中 93 文について，確率計算の段階でメモリ不足となり，順位付けができなかった．この 93 文は，今回の評価対象からは除外した（PGLR モデルの学習には利用した）．

解析所要時間の差と同様，解析精度の差についても，提案手法と接続制約を組み込まない場合との間では，最終的に出力される解析木の数の違いが要因と考えられる．一方，Liの手法によるLR表での最終的な解析結果の曖昧性は提案手法の場合と同じである．また，提案手法でのみ削除可能な動作は，どのような動作実行列をたどっても，最終的にaccに到達することのないものであるため，学習データ中にも存在しない．PGLRモデルによるLR表中の各動作の確率は，学習データに付与された構文木を生成する際に実行する動作の使用回数をもとに計算されるが，最終的にaccに到達できない動作に対する確率は0となり，最終的に出力される各解析木の確率は提案手法の場合と同じになるはずである．しかし，MSLRパーザでは，確率計算の平滑化のため，全ての動作の実行回数に一定数（初期設定では0.5）を加えている．その結果，学習データ中で使用されない動作についても0ではない確率が与えられ，最終的に出力される各解析木の確率が提案手法の場合とLiの手法の場合との間で異なる場合があり，それが，解析精度に差が生じる要因になる．平滑化を行わなければ同じ結果になるが，その場合，accに到達可能であり，かつ，妥当な動作であるにもかかわらず学習データに偶然出現しなかった動作に対する確率も0となる．確率が0である動作が，接続制約を組み込んだことによってaccに到達不可能となった動作であるか，偶然学習データに出現しなかった動作であるかを，学習の段階で区別することは困難である．LR表を作成する段階でaccに到達不可能な動作を削除しておけば，この問題を回避することが可能であり，その点においても提案手法が有効であることが分かる．

5　提案アルゴリズムの完全性の証明

本節では，提案アルゴリズムの完全性について考察する．ここで，完全性とは，作成されるLR表に不要なアクションが存在しないことである．これを示すためには，LR表が以下の2つの性質を満たすことを示せばよい．

- 妥当性
  任意の構文木 tr に対し，以下が成り立つ．

$$\mathrm{Generate}(tr, G, C) = \mathrm{GenerateLR}(tr, T)$$

  ただし，
  G, C, T:　CFG，接続制約，LR表
  $\mathrm{Generate}(tr, G, C)$:　文法 G，接続制約 C から構文木 tr を生成可能ならば1，不可能ならば0
  $\mathrm{GenerateLR}(tr, T)$:　LR表 T から構文木 tr を生成可能ならば1，不可能ならば0
- 最小性

妥当性を満たすLR表中の任意の要素（動作）aに対し，以下が成り立つような構文木 trが存在する．

$$\mathrm{GenerateLR}(tr, T) = 1 \wedge \mathrm{GenerateLR}(tr, T_a) = 0$$

ただし，

T_a: LR表Tから要素aを除いたLR表

文法Gは，第2.1節で述べた，ε規則を含まないという条件のほかに，以下の条件を満たすことを前提とする．

(1) 文法規則は重複しない．すなわち，文法G中の任意の2つの文法規則$A \to \alpha$，$B \to \beta$について，$A \neq B \vee \alpha \neq \beta$

(2) 循環する導出は存在しない．すなわち，文法G中の任意の非終端記号Aについて，$A \xrightarrow{*} A$となるような導出は存在しない

5.1 妥当性の証明

提案アルゴリズムによって作成されるLR表が妥当性を満たすことを示すためには，以下の2つを示せばよい．

(1) $\mathrm{Generate}(tr, G, C) = 1$ ならば $\mathrm{GenerateLR}(tr, \mathrm{Table}(ACG)) = 1$

(2) $\mathrm{GenerateLR}(tr, \mathrm{Table}(ACG)) = 1$ ならば $\mathrm{Generate}(tr, G, C) = 1$

ただし，

ACG: 提案アルゴリズムによって生成されるアクションチェイングラフ

$\mathrm{Table}(ACG)$: ACGから生成されるLR表

提案アルゴリズムでは，開始記号から最右導出を行いながらアクションチェイングラフを生成し，その中に含まれる成功パスからLR表を生成する．ここで，$\mathrm{Generate}(tr, G, C) = 1$を満たす構文木$tr$に相当する最右導出の際に，提案アルゴリズムによって生成されるアクションチェインは，成功パスである．この成功パス中の要素に対応する動作は，このアクションチェイングラフから生成されるLR表に含まれるので，trは$\mathrm{Table}(ACG)$から生成可能である．すなわち，(1) が成り立つ．

一方，ある構文木trが$\mathrm{GenerateLR}(tr, \mathrm{Table}(ACG)) = 1$を満たすと仮定する．このとき，$\mathrm{Table}(ACG)$から$tr$を生成する際の実行動作列について，先頭の実行動作から順に，以下の法則に従ってACG中のアクションチェイン要素をたどることにより，成功パスを得ることができる．

- 注目する実行動作がacc動作の場合，$[0, \mathrm{re}_0, \$, \mathrm{end}]$をたどる．
- 注目する実行動作が状態s，先読みlaにおけるshift動作の場合，$[s, \mathrm{sh}, la, \mathrm{end}]$をたどる．

- 注目する実行動作が状態 s, 先読み la における規則番号 n による reduce 動作, さらにその次の動作が状態 s', 非終端記号 LHS(n) における状態 s'' への goto 動作の場合, $[s', \mathrm{re}_n, la, \mathrm{end}]$ をたどる.

ACG 中の成功パスに対応する構文木は文法 G, 接続制約 C を満たすので, (2) が成り立つ.

以上より, 提案アルゴリズムによって作成される LR 表は妥当性を満たす.

5.2 最小性の証明

$T = \mathrm{Table}(ACG)$ が最小性を満たさないと仮定すると, 次を満たす要素 a が T 中に少なくとも 1 つ存在する.

任意の $tr \in \{tr | \mathrm{GenerateLR}(tr, T) = 1\}$ に対して, $\mathrm{GenerateLR}(tr, T_a) = 1$

このとき, $\{tr | \mathrm{GenerateLR}(tr, T) = 1\} \equiv \{tr | \mathrm{GenerateLR}(tr, T_a) = 1\}$ となり, a に対応する ACG 中の要素を e とすると, $\{tr | \mathrm{GenerateLR}(tr, T_a) = 1\}$ 中の任意の構文木を生成する際の実行動作列に対応する ACG 中の成功パスは, e を含まない.

一方, T 中に a が存在することから, ACG 中には e を含む成功パスが存在する. その成功パスに対応する実行動作列は a を含み, その実行動作列で生成される構文木を tr' とすると, 以下が成り立つ.

$$\mathrm{GenerateLR}(tr', T) = 1 \wedge \mathrm{GenerateLR}(tr', T_a) = 0$$

これは T が最小性を満たさないという仮定に矛盾する.

以上より, 提案アルゴリズムによって作成される LR 表は最小性を満たす.

6 結論と今後の課題

コーパスベースの自然言語処理技術は, 音声認識などにおいて, 精度向上のブレイクスルーを持たらした. これは, コーパスの量を増やすことによって精度が向上したからであるが, それには限界が見えはじめている. この限界を越える技術として, コーパスの量を増やすのではなく, ルールベースの手法を再考すべき時期に来ていると考えている.

本論文では, ルールベースの構文解析の 1 つである一般化 LR 構文解析に注目し, 品詞間接続制約を LR 表に組み込み, 不要な動作を削除する手法を提案した. 提案手法により, 接続制約による削除を行わない場合と比較して約 64%の不要動作を削除でき, 従来手法と比較するとさらに約 1.2%の不要動作を削減できた. 提案手法により作成した LR 表で構文解析を行った場合, 解析所要時間は, 接続制約を組み込まない LR 表で構文解析を行った場合と比較して約 52%, 従来手法と比較して約 2.4%短縮された. 解析精度 (文正解率) は, 接続制約を組み込まない場合と比較すると向上が見られたが, 従来手法と比較すると大きな差は見られなかった. しかし,

PGLR モデルによる確率計算の平滑化における問題を回避するためにも，不要な動作を削除することは有効であり，今後，コーパスベースの手法を取り入れた場合の精度向上の効果が大きくなると考えている．

実験で示した解析精度（文正解率）はコーパスベースの解析と比較すると低いと思われるかもしれない．しかし，MSLR パーザは品詞間の接続制約と CFG のみを利用して構文解析を行う．この結果に共起データ等の情報を加えれば，コーパスベースの解析と同程度の正解率が得られるものと期待される[6]．筆者らはルールベースの自然言語処理にはまだ検討の余地があると考えている．

参考文献

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools.* Addison Wesley.

Briscoe, T. and Carroll, J. (1993). "Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars." *Computational Linguistics*, **19** (1), pp. 25–59.

Charniak, E. (1996). *Statistical Language Learning.* The MIT Press.

DeRemer, F. and Pennello, T. (1982). "Efficient Computation of LALR(1) Look-Ahead Sets." *ACM Transactions on Programming Languages and Systems*, **4** (4), pp. 615–649.

Inui, K., Sornlertlamvanich, V., Tanaka, H., and Tokunaga, T. (2000). "Probabilistic GLR Parsing." In Bunt, H. and Nijholt, A. (Eds.), *Advances in Probabilistic and Other Parsing Technologies*, chap. 5, pp. 85–104. Kluwer Academic Publishers.

Jelinek, F. (1998). *Statistical Methods for Speech Recognition.* The MIT Press.

工藤拓，松本裕治 (2002). "チャンキングの段階適用による日本語係り受け解析." 情報処理学会論文誌, **43** (6), pp. 1834–1842.

Li, H. and Tanaka, H. (1995). "A Method for Integrating the Connection Constraints into an LR Table." In *Natural Language Processing Pacific Rim Symposium*, pp. 703–708.

Noro, T., Koike, C., Hashimoto, T., Tokunaga, T., and Tanaka, H. (2005). "Evaluation for a Japanese CFG Grammar Derived from Syntactically Annotated Corpus with Respect to Dependency Measures." In *the 5th Workshop on Asian Language Resources*, pp. 9–16.

[6] 日本語文節係り受け解析では，文節係り受け精度は 90%を超えるが，1 文中の全ての係り受けが正解となる割合は 60〜65%程度である (Noro et al. 2005)．文節区切りや形態素解析の誤りを考慮すると，文全体としての精度はさらに下がるものと考えられる．

白井清昭，植木正裕，橋本泰一，徳永健伸，田中穂積 (2000). “自然言語解析のための MSLR パーザ・ツールキット.” 自然言語処理, **7** (5), pp. 93–112.

Tanaka, H., Tokunaga, T., and Aizawa, M. (1995). “Integration of Morphological and Syntactic Analysis Based on LR Parsing Algorithm.” 自然言語処理, **2** (2), pp. 59–74.

Tomita, M. (1991). *Generalized LR Parsing.* Kluwer Academic Publishers.

内元清貴，関根聡，井佐原均 (1999). “最大エントロピー法に基づくモデルを用いた日本語係り受け解析.” 情報処理学会論文誌, **40** (9), pp. 3397–3407.

略歴

野呂　智哉：2000 年東京工業大学工学部情報工学科卒業．2005 年同大学大学院情報理工学研究科博士課程修了．同年同大学同研究科計算工学専攻助手．現在，同大学同研究科計算工学専攻助教．博士（工学）．自然言語処理，Web 情報処理等の研究に従事．言語処理学会，情報処理学会，日本ソフトウェア科学会各会員．

田中　穂積：1964 年東京工業大学工学部情報工学科卒業．1966 年同大学院理工学研究科修士課程修了．同年電気試験所（現産業技術総合研究所）入所．1980 年東京工業大学工学部情報工学科助教授．1983 年同教授．1994 年東京工業大学大学院情報理工学研究科教授．2005 年中京大学情報科学部認知科学科教授．2006 年東京工業大学先進研究機構機構長．2009 年北陸先端科学技術大学院大学情報科学研究科特任教授，現在に至る．工学博士．人工知能，自然言語処理に関する研究に従事．情報処理学会，電子情報通信学会，認知科学会，人工知能学会，計量国語学会，Association for Computational Linguistics，各会員．

橋本　泰一：1997 年東京工業大学工学部情報工学科卒業．2002 年同大学大学院情報理工学研究科博士課程修了．同年同大学同研究科計算工学専攻助手．2006 年同大学統合研究院特任助教授．現在，同大学統合研究院特任准教授．博士（工学）．自然言語処理，情報検索に関する研究に従事．言語処理学会，情報処理学会，人工知能学会，科学技術社会論学会，各会員．

白井　清昭：1993 年東京工業大学工学部情報工学科卒業．1998 年同大学院情報理工学研究科博士後期課程修了．同年同大学院情報理工学研究科計算工学専攻助手．2001 年北陸先端科学技術大学院大学情報科学研究科助教授．現在

同准教授．博士（工学）．自然言語処理に関する研究に従事．言語処理学会，情報処理学会，人工知能学会，Association for Computational Linguistics，各会員．

（2009 年 2 月 4 日　受付）
（2009 年 4 月 6 日　再受付）
（2009 年 6 月 8 日　採録）