一般LR構文解析法におけるエラー処理

斎藤博昭†

文脈自由文法を用いたシフト・還元操作による高速な解析法のひとつである一般LR構文解析法において，入力の誤りを検知・修正する一手法を提案する．この手法においては，従来とまったく同一の文法規則と解析表を使いながら，その表の見方を変えることでエラー処理を行う．すなわち，従来はある時点での状態と入力記号をもとに解析表の動作部を参照し，シフト，還元，受理，エラーの4つの動作のいずれであるかを決定し解析を進めていた．これに対して本手法では，エラー状態にならないように，入力に対して挿入，置換，読み飛ばしの3種の操作を行い，入力誤りを修正しつつ解析を進める．ただこの方法では，入力中の小さな誤りは修正できるが，大きな誤りに対しては文法に適うように無理に修正作業を行うおそれがある．そこで，解析表の行先部をもとにダミーの非終端記号をあてがう操作を行い，過修正を防ぐことにする．このダミーの非終端記号により，入力中の大きな欠落や挿入ノイズを検出できるだけでなく，入力中の不明確な部位が原因で解析全体が失敗することを防ぐことができる．従来の手法に比べて本手法においては探索が増大するが，効率低下を抑えるための工夫を導入している．また，本手法は入力中の未知語に対しても頑健な処理ができる．音声認識における誤り修正の実験を通して，本手法の有効性を確認する．

Robust Error Handling in the Generalized LR Parsing

HIROAKI SAITO†

This paper proposes a method of handling erroneous sentences in the generalized LR parsing. Our proposed method uses the same grammar and its LR table as in the original generalized LR parsing. The new method, however, looks up the action table in different ways in that no error state is encountered during parsing by substituting/inserting/deleting erroneous input symbols. Although this error correcting mechanism works fine against minor errors, the mechanism might over-correct big errors by obeying the grammar too strictly. Thus our method also introduces a 'dummy nonterminal symbol' against a big erroneous part. Dummy nonterminals are placed by consulting the goto table before the reduce action takes place. The dummy nonterminal prevents a noisy portion from collapsing the whole parse. Unknown words are also handled elegantly in our method. Experiments show how the proposed method works successfully in parsing spoken sentences.

1. はじめに

自然言語処理においては，これまで文脈自由文法を用いた解析が広く行われてきた．近年，より大規模かつ実用的な自然言語処理システムへ向けて，すべての文法ルールが書けるかといった問題が提起され，文例による処理，コーパスを用いた処理などの研究がさかんに行われている．たしかに文脈自由文法での処理だけをもって，自然言語を解析していくのは限界が見えてきている．しかしながら，解析時間が$O(n^3)$（nは入力列の長さ）で抑えられる，文法の管理がしやすいといった利点を持った文脈自由文法での解析法をまったく離れてしまうのも惜しまれる．本稿では，文脈自由文法を用いた解析法のうち，一般LR構文解析法でいかに入力のエラーに対処できるかを示し，この解析法に柔軟性および頑健さを付与する．

† 慶應義塾大学理工学部
Department of Mathematics, Keio University

2. 一般LR構文解析法

一般LR構文解析法は，コンパイラなどに広く利用されてきたあいまいでない文脈自由文法に対してのLR構文解析法[1]を，文脈自由文法一般に対して拡張した技法である[8]．LR構文解析法では文脈自由文法から解析表を作成し，その表を引くことで解析を行うが，一般LR構文解析法においても同一の方法で文法規則から解析表を作成し，その表をもとに解析を進める．

図1に英語文を解析する簡単な文法例を示すが，各

```
(1)  S   -->  NP  VP
(2)  S   -->  S   PP
(3)  NP  -->  noun
(4)  NP  -->  det  noun
(5)  NP  -->  NP  PP
(6)  PP  -->  prep  NP
(7)  VP  -->  verb  NP
```

図1 文脈自由文法の例
Fig. 1 Sample context-free grammar.

表1 一般LR構文解析表
Table 1 Generalized LR table.

状態	動作部					行先部			
	det	noun	verb	prep	$	NP	PP	VP	S
0	s3	s4				2			1
1				s6	acc		5		
2			s7	s6			9	8	
3		s10							
4			r3	r3	r3				
5				r2	r2				
6	s3	s4				11			
7	s3	s4				12			
8				r1	r1				
9			r5	r5	r5				
10			r4	r4	r4				
11			r6	r6,s6	r6		9		
12				r7,s6	r7		9		

ルールは

（ルール番号） 非終端記号 --> [非] 終端記号列

となっており，終端記号は小文字，非終端記号は大文字で記してある．

表1は図1の文法から生成した解析表である．一般LR構文解析表中の動作部（action table）の4種の動作（シフト，還元，受理，エラー）は通常のLR構文解析法のものとまったく同一のものである．この動作部と行先部（goto table）を表引きしながら，スタックを用いたシフト・還元操作で解析を進める[1]．一般LR構文解析法においては，ある状態における動作が1つとは限らないことが大きな違いとなる（表1中の状態11および12におけるprep入力時の動作）．動作が複数ある状態になったときは，解析スタックを分割し，動作をすべて並列に実行することになる．ただ，探索の爆発を防ぐため，並列実行されたスタックを同じ状態点に到達した時点で統合する[8]．

一般LR構文解析法は自然言語処理の分野で広く用いられており，一般LRパーザ生成プログラムも配布されている[2),9)]．また，一般LR構文解析法の考え方を音声認識と組み合わせる研究も行われている[7),10),11)]．

3. 入力誤りへの対処

一般LR構文解析法にせよ，そのもととなったLR構文解析法にせよ，その使われ方は「入力が定義された文法に適っているかどうか」を決定するものであった．プログラミング言語の解析のように文法が厳密に定義できる分野においては確かにそのような判定で十分であった．しかし，文脈自由文法一般にまで適用範囲が拡張された一般LR構文解析法を，たとえば自然言語の処理に適用しようとすると，入力が文法規則に適っているかどうかを調べるというよりは，ある状況における使われうる文を文法で規定し，入力がそのうちのどの文であるかを決定するといった使い方が実際的となる☆．特に入力が音声の場合には，100%の認識率を期待することができないので，後者の使い方にならざるをえない．

エラーの修正操作を記す前に，(一般) LR構文解析法における表引き関数について定義しておく．すなわち，スタックの最上段に積まれている状態をs，現在の入力記号をaとすると，解析表の動作部をもとに動作関数 action(s, a) はその時点での動作（シフト，還元，受理，誤りの4種ある）を返す．一般LR構文解析法を使用するので，動作は複数の場合もありうる．還元動作後，スタックの最上段の状態をs'，還元された非終端記号をAとすると，解析表の行先部をもとに行先関数 goto(s', A) はAの状態を返す．

ここで誤りを含む入力に対応するために，一般LR構文解析法にエラー対処機能を付与する．ここでいうエラー対処機能とは，入力文中に未登録語があったり，入力文が文法外であったときに，解析表動作部の参照に戻れるまで入力語を読み飛ばしたり，他の語を挿入したり，置き換えることで解析を続けるものである．入力記号列 $x_1, x_2, ..., x_n, \$$ （$は記号列の最後を示す）において，記号 x_i に対するスタックの最上段の集まり $\{p_i\}$ ($0 \leq i$, p_0は始状態）で，すべての p_i において

$$action(p_i, x_{i+1}) \Rightarrow error$$

となったとき，次の3つの操作で解析の続行を試みる．

[記号置換] x_{i+1} を

$$\{y_i \mid action(p_i, y_i) \neq error\}$$

となるすべての y_i（終端記号）で置き換え，解析を続ける． □

[記号飛越し] x_{i+1} を無視し x_{i+2} について，解

☆ コンパイラにおいても入力プログラムに対するエラー処理としてこのような考え方はあったが，あくまでも補助的な使われ方であった．

man	·····	noun
I	·····	noun
park	·····	noun
saw	·····	noun
saw	·····	verb
a	·····	det
the	·····	det
in	·····	prep
.		.
.		.

図2 辞書
Fig. 2 Lexicon.

析を進める. □

[記号挿入] x_i と x_{i+1} の間に
$\{y_i \mid \text{action}(p_i, y_i) \neq \text{error}\}$
となるすべての y_i(終端記号) を挿入し，解析を続ける. □

ここで "I saw a man the park" という非文が，図1の文法および表1の解析表を用いた一般LR構文解析法により，どのように解析・修正されるかを示す．この解析にあたっては，実際の単語を文法中の終端記号に変換する**図2**のような辞書が用意されているものとする．さらに，不必要に説明を複雑にしないために，「入力文中に誤りは連続して起こっていない」と仮定する.

まず，たとえ最初の語 'I⟨noun⟩' が始状態0で受け入れられるとしても，その語の前に記号挿入の操作により1語を挿入するのだが，
$\{y_0 \mid \text{action}(0, y_0) \neq \text{error}\} = \{\text{det}, \text{noun}\}$ より状態0で受理されうる語は det と noun であることが分かる．ここで，「入力中に誤りは連続して起こらない」という仮定と，図1の文法がある特定の終端記号が2つ続けて起こることを許していないということから，ここでの noun の挿入の可能性は考えなくてよい☆. $\text{action}(0, \text{det}) \Rightarrow$ s3 により '(det)' がシフトされ，新たな状態3が決定される.(**図3**：図中の点線で示したパスが1つのスタックを表している．数字はスタックの状態を表す．( ) で囲まれた終端記号は3つのエラー対処操作により作成されたものであることを示す.)

最初の語 'I⟨noun⟩' は始状態0からも挿入語 '(det)' からも受け入れられるので，両方ともシフト操作を行う. $\text{action}(3, \text{noun}) \Rightarrow$ s10 により 'I' がシフトされ，新たな状態10が決定される．状態10では $\text{action}(10, *) \Rightarrow$ r4 により，ルール4を使って還元が行われ☆☆，非終端記号NPが作成される. $\text{goto}(0, \text{NP}) \Rightarrow$ 2 より，そのNPの状態は2となる.

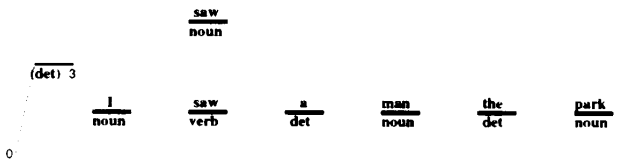

図3 解析トレース
Fig. 3 Parse trace.

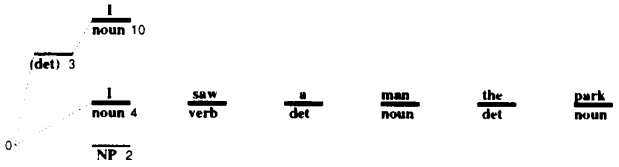

図4 解析トレース（続き）
Fig. 4 Parse trace (cont'd).

同様に，$\text{action}(0, \text{noun}) \Rightarrow$ s4 により 'I' がシフトされ，新たな状態4が決定される．状態4では $\text{action}(2, *) \Rightarrow$ r3 により，ルール3を使い還元が行われ，非終端記号NPが作成されるのだが，ここで次の結束操作により，"(det) noun" によるNPと統合する（**図4**）.

[非終端記号の結束]
ある状態点から伸びている複数のスタックが，同一の非終端記号と状態番号と終了地点を持つとき，これを1つの非終端記号に束ねる. □

すなわちこのNPは，"(det) I" から生成されたNPと 'I' のみから生成されたNPの2つをまとめたものである．この結束操作はスタック数の急増を防ぐとともに，この結束時に枝刈りをすることを可能にする☆☆☆（この例でいえば '(det)' をペナルティと考えて，NPを 'I' のみから生成されたとする），実用性が高い操作である.

'I⟨noun⟩' については，これの置換候補を記号置換操作により考えなければならないが，この状況では始状態0につながる (det) のみである．また，'I⟨noun⟩' を挿入誤りと見なして，記号飛越しの操作により次の 'saw⟨noun⟩' および 'saw⟨verb⟩' を始状態0につなげることも試みなければならない（**図5**）.

このように解析は続き，'man⟨noun⟩' までの解析を

☆ 実際には，'I' のような代名詞の前に冠詞が付くことはないので，語彙接続表といったものを利用すれば "(det) I" という接続も考えなくてよい.

☆☆ ここでは入力列の先読みをしないことにするので，還元動作がありうる状態点に到達したときには，還元動作を行うことにする.

☆☆☆ 詳しくは5章で述べる.

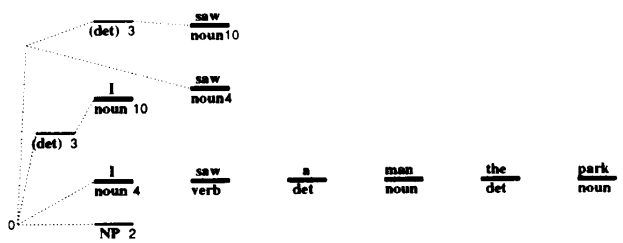

図5 解析トレース（続き）
Fig. 5 Parse trace (cont'd).

示したものが，**図6**である．

すべての解析が終了したときの状況を示したのが**図7**であり，結果として受理されたSは，

```
1. [(det) noun] [verb [[det noun] [(prep) [det noun]]]]
2.      [noun] [verb [[det noun] [(prep) [det noun]]]]
3. [[(det) noun] [verb [det noun]]] [(prep) [det noun]]
4.      [[noun] [verb [det noun]]] [(prep) [det noun]]
5. [(det) noun] [(verb) [[det noun] [(prep) [det noun]]]]
6.      [noun] [(verb) [[det noun] [(prep) [det noun]]]]
7. [[(det) noun] [(verb) [det noun]]] [(prep) [det noun]]
8.      [[noun] [(verb) [det noun]]] [(prep) [det noun]]
```

の8つのパスを含んでいる．このうち，1～4はverbとしてのsaw，5～8はnounとしてのsawを使っている．もし，エラー対処操作によって挿入や置換された語をペナルティと考えると，2の解析木が一番もっともらしいと判定できる．

次に，入力が未知語を含んでいる場合を考えてみる．この場合の未知語とは図2のような辞書に入っていない未登録語のことである．解析は未知語のところで続けられなくなるが，記号置換の操作により未知語の品詞が推定できるので，解析を続けることができる．もし未知語が2つ以上続いている場合には，1語置換では解決できないが，これへの対処については次章で述べる．

4. 大きなエラーに対する処置

前章の3つの操作は，規定した文法に適うように入力を修正するものであった．文法が厳格に規定することができ，しかもその規定された文法にあてはまらない文はありえないという状況においては，この3つの操作で十分であろう．しかしながら実際的な自然言語処理においては，厳格な文法を作ることは難しく，文法にあてはまらない予想もしないような入力への対応が必要となる．このような場合に，前章で述べた3つのエラー回復操作だけで処理すると，文法に適うように無理に修正することになりかねない．そこで，本章では「ギャップ埋め」という操作を導入し，不明確な部位はダミーの非終端記号で置き換えることを提案する．

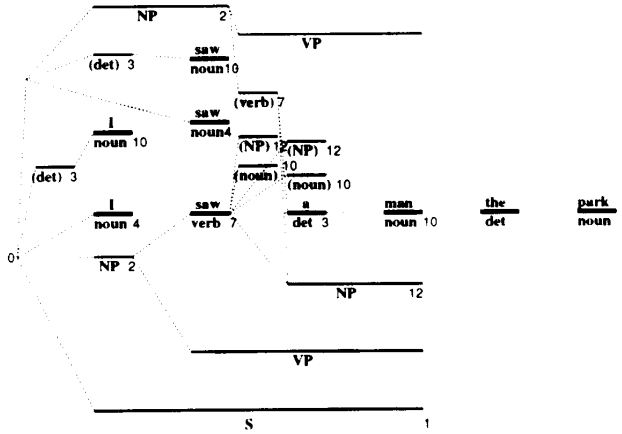

図6 解析トレース（続き）
Fig. 6 Parse trace (cont'd).

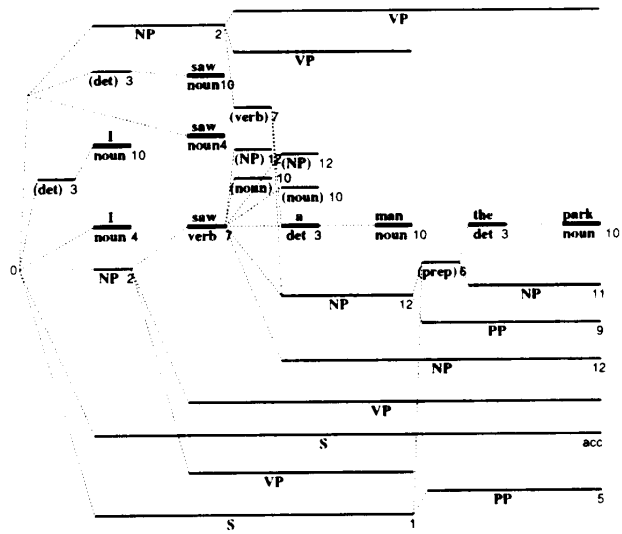

図7 解析トレース（終了図）
Fig. 7 Parse trace (complete).

[ギャップ埋め]

入力記号列 $x_1, x_2, \ldots, x_n, \$$ において，記号 x_i に対するスタックの最上段の集まり $\{p_i\}$ $(0 \leq i,\ p_0$は始状態）で，次の操作を行う．

$\{X \mid \mathrm{goto}(p_i, X) \neq \mathrm{error}\}$

となるすべての非終端記号 X を見つけ，これをダミーの非終端記号として生成する．この結果，x_i 以降のある長さの部位を生成したダミーの非終端記号で置き換えることになる． □

では，ここで“we cut sad with a knife”という非文を図1の文法および表1の解析表を使い，どのようにギャップ埋めの操作が行われるかを示す．この文は，前章に導入した記号置換操作により‘sad’の代わりに‘(noun)’を置くことで解析できるのであるが，ここではギャップ埋めの操作を分かりやすくするため，あえて前章の3つの操作は行わないことにする．

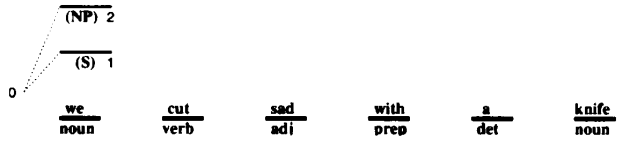

図8 解析トレース
Fig. 8 Parse trace.

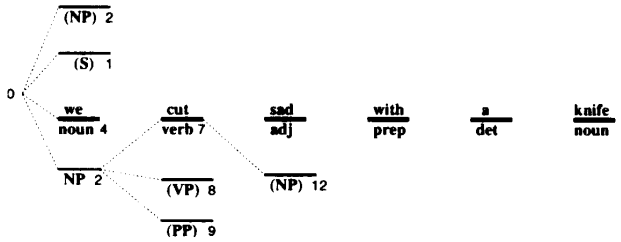

図9 解析トレース（続き）
Fig. 9 Parse trace (cont'd).

始状態0において，goto(0, X) ⇒ error とならないXはNPとSで，ギャップ埋め操作により，これらの非終端記号がダミーの非終端記号として作られうることが分かる．このとき，最初の語の 'we(noun)' が始状態0においてシフト可能であることから，ダミーの非終端記号を作る必要がないと考えることもできるが，'we' が入力誤りである可能性もあるので，これら2つのダミー非終端記号を作ることにする．NPとSのダミー非終端記号の状態はそれぞれ2と1である（図8）．NPの新状態2において，goto(2, X) ⇒ error とならないXはPPとVPだが，この2つのダミー非終端記号の可能性は考えない．なぜなら，2つ以上のダミーの非終端記号が続くことは実用上まずありえず，そのような可能性まで考えると探索が急増するからである．同様の理由でダミーの非終端記号Sに続く他のダミー非終端記号の可能性も考えない．

action(0, noun) ⇒ s4 により最初の語 'we' がシフトされ，新たな状態4が決定される．状態4では，action(4, *) ⇒ r3 によりルール3を用いて還元が行われ，非終端記号NPが作成される．goto(0, NP) ⇒ 2 より，そのNPの状態は2となる．状態2では，{X | goto(2, X) ≠ error} = {VP, PP} より，ダミーの非終端記号VPとPPを作成する．次いで，action(2, verb) ⇒ s7 より，2番目の語 'cut' はシフトされ，状態は7となる．状態7では，{X | goto(7, X) ≠ error} = {NP} より，ダミーの非終端記号NPが作られる（図9）．

次の語 'sad(adj)' の解析においては，この語がどの状態にもつながらないので，無視する．

このように解析は進み，最終状態は図10のようになり，Sとして受理された解析は次の2つである．

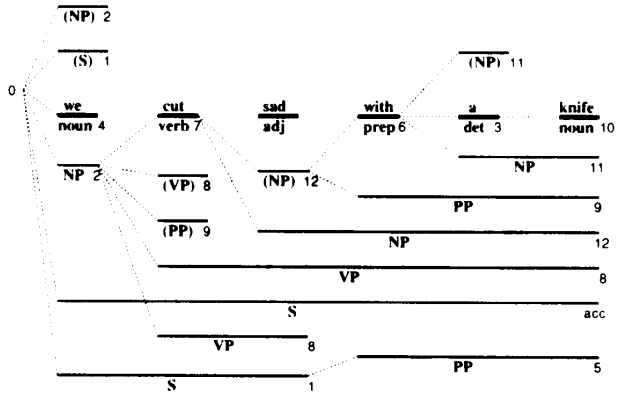

図10 解析トレース（終了図）
Fig. 10 Parse trace (complete).

1. [noun [verb [(NP) [prep [det noun]]]]]
2. [[noun [verb (NP)]] [prep [det noun]]]

すなわち，'sad' という語の部位にダミーの非終端記号NPがあてがわれたものが得られる．この例では，ダミーの非終端記号が1語をカバーしていたが，一般に非終端記号は1個以上の［非］終端記号から成り立つので，ダミーの非終端記号の長さは個々の場合で調整することになる．

このように，ダミーの非終端記号を導入することで次のような利点がある．

- 一般LR構文解析法は，入力列の最初の記号から最後の記号へバックトラックなしで解析が進む．したがって，誤りが前章で導入した1記号単位の3つの回復操作では修正できなかった場合には，入力列全体の解析に失敗することになる．ダミーの非終端記号が，そのような大きな誤りを含む部位をカバーすることによって，解析全体の失敗を防止することができる．
- 入力中に大きな欠落があるような場合，その部位をダミーの非終端記号で覆うことができる．
- 未知語が2つ以上続いているようなときには，その未知語列をカバーするようにダミーの非終端記号が作られる．
- 音声のように入力記号に何らかの確からしさの度合いが付いているときは，還元動作の際，確かなときのみその動作を行うことで挿入ノイズが検出できる．

逆にダミーの非終端記号導入による欠点として探索の増大があるが，この点については次章で述べる．

5. エラー処理による探索増大の制御

3章と4章で導入したエラー処理を解析が先に進めなくなったときの操作として行うとすると，どこが

原因だったかを特定するために，解析のバックトラックをすることになる．もちろん，このようなバックトラックの操作を行ってもよいのだが，一般LR構文解析法そのもののバックトラックなしで複数の解析木を並列に扱えるという特長を利用すれば，エラー状態に到達する前から入力中に誤りがあることを想定してエラー処理を並列的に行うことができるし，実装上も容易である☆．ただ，こうするとエラー処理により通常のLR構文解析法に比べて探索空間が爆発的に拡大することは明らかであり，ここではその増大する探索をいかに制御するかを記す．

まず，エラー処理の操作自体の制御について述べる．3章で導入したエラーに対する処置では，LR構文解析表の動作部を「次には…がくるはずである」といった予測の道具として使用しているのが特徴である．しかもこの「次」ということは必ずしも「1つ先だけ」ということに限られず，「任意個分の先」に対して適用できる．しかしながら，原理としてはそうであっても，実際の自然言語のアプリケーションにおいては，エラーが連続して発生することは少なく，3つのエラー回復操作を2記号以上の誤りまで考慮して解析を進めると，探索が増大するだけで効果が少ないことがしばしばである．

4章で導入したダミーの非終端記号においては，これを無制限に作成すると解析効率が落ちることは明らかであり，4章でも紹介した「ダミーの非終端記号が2つ以上連続することはない」というヒューリスティックスを用いることは自然であろう．

次に，解析中での枝刈りについて述べる．これまでにも記したように，解析は入力列を左から右にバックトラックなしに進むので，エラー処理による増大する探索数を枝刈りによって制御することができる．枝刈りに用いる評価スコアには，入力シンボルに付随するものと，エラー処理に付随するものと2種類ある．前者は，たとえば，音声認識や文字認識の結果であればその認識スコアである．後者のスコアにおいても，認識結果がラティス形式であれば，あたかもその認識結果を挿入，飛越し，置換の操作で生じた候補と見なすことで，付随する認識スコアを使うことができる．認識結果がラティス形式でない場合には，3つの誤り操作をペナルティとするようなスコア付けをしなければならない．この際，たとえばキーボードからの入力の場合，タイプミスの起こりやすさは決して一様ではなく，なんらかの傾向があり，その傾向をスコア付けに反映させるべきである．文字認識においても，どの文字がどの程度他の文字に誤って認識されるかといった統計情報から，スコア付けが可能となる．

ダミーの非終端記号のスコアに関しては，入力からその起こりやすさを推測することは難しい．そこで，「その長さとともに文法と解析状況により決定する」といったヒューリスティックにならざるをえない．また，入力がある程度正しいものでないと，解析がダミーの非終端記号のみになってしまい，探索が増すだけで実用的な効果が少なくなってしまうこともある．ただ，確かにギャップ埋めの操作により探索は増大するものの，ダミーの非終端記号を解析途中の一種の保険と考え，たとえダミーの非終端記号を置いても，非終端記号の結束の操作の際に，ダミーの非終端記号を含まない解析が得られていれば，ダミーの方を刈ってしまったり，通常のシフト動作の際にダミーの非終端記号を含まないパスが得られていればダミーの非終端記号を含むパスを刈るといったことで，探索がいたずらに増大することは防げる．次章での音声認識結果の修正実験においてもこの工夫を採り入れている．

6. 音声認識への適用例

これまで述べてきた一般LR構文解析法におけるエラー処理の諸方法を音声認識に適用した例をここで示す．音声認識においてはその認識率が100%になることはなく，どうしても認識誤りを生じる．この認識誤りをエラー対処機能付き一般LR構文解析法で処理するのである．

一般LR構文解析法を音声認識のどの段階に適用するかという問題だが，認識の最終段階である複数の文候補に対して順次行ういわゆる“Nベストチェック”と呼ばれる方法から，音声認識のエンジンに組み入れる方法まで，さまざまな可能性が考えられるが，ここでは音声認識デバイス[3]が生成した文発話の音素列（使用したデバイスではローマ字列と同じようなものが得られる）に対して適用することを考える．“otomagaitai（発声は「頭が痛い」）”といった音素列を解析することになるので，文法の終端記号は音素である．その結果として，**図11**のように文法が語彙を包含することになる☆☆．このデバイスは認識結果として単に音素列を生成するだけで，各音素の確からしさ（スコア）は提供しない．そこで，このデバイスを製作した際に，大量の認識データから“ある音素がどれほどの確率で

☆ 3章と4章の解析例においてもこのような方針をとっている．

☆☆ このような音素表記そのものを‘隠れマルコフモデル（Hidden Markov Model）’に基づく音声認識部のエンジンとして用いたのがHMM-LR法と呼ばれるものである[11]．

```
S    -->  NP  VP
NP   -->  NOUN
NP   -->  NP  PP
NOUN -->  a t a m a
NOUN -->  k u b i
PP   -->  g a
PP   -->  h a
VP   -->  i t a i
VP   -->  i t a m u
```

図11 音素ベースの文脈自由文法
Fig. 11 Phoneme-based context-free grammar.

ある音素に置き換わったか，抜け落ちたか，挿入されたか”という情報を混同表（confusion matrix）という形で作成しているので，その混同表に基づいて認識結果およびエラー処理にともなう候補のスコア付けをしている．5章に記した探索の制御法は，入力の確からしさによりその適用の厳しさを自由に調節できるものであり，この混同表によるスコア付けにより，各音素は0から33までの確からしさを持ち，ダミーの非終端記号は長さ2～7音素分，確からしさ−10として扱い，枝刈りはその時点の最良スコアからのある範囲外のものを刈るということにしている．この範囲の幅は，解析の初めは広く，だんだんと狭めるように動的に変えている．これは，発声開始時の曖昧さを吸収するとともに，解析時間を短縮するためである．また，この実験においても4章と5章で述べた「ダミーの非終端記号が2つ以上連続することはない」という制限を設けている．

実験は，医者と患者の間に交わされる会話文（日本語）をドメインとした小規模の文法（単語数150）を使い，5人の話者が25文ずつ発話した計125文を解析することで行った．記号挿入，記号飛越し，記号置換（この実験の場合の「記号」は「音素」になる）の3つのエラー修正操作により111文（88.8%）が正しく認識できた．この場合の「正しい認識」とは，修正操作の結果，正解が最高のスコアをとることである．

修正しきれなかった14文は以下の3つのカテゴリーに分類される．

(i) 4文については，ダミーの非終端記号が入ったものが最高のスコアをとっている．たとえば，“hideriesjakiihjisuru（発声は「左足がヒリヒリする」）”という発話に対して，解析の結果“ひだりあしが（pain-spec）する”となり，ダミーの非終端記号pain-specが挿入されたものが得られた．

(ii) 6文については，正解が最高のスコアを得られなかったというものである．

(iii) 4文については，発話全体がひどく修正を施しても一定のスコアに到達しなかった．

(i)にあるようにダミーの非終端記号を導入することによって，発話の一部の不明確な部位のために発話全体が解析不能になることを防いでいる．しかも，発話のどこが不明確かを明示することができる．さらに(ii)においても，たとえば“hasirutoguregasazyoogugitamo（発声は「走ると腕が刺すように痛む」）”という発話に対して，

1（622点） HASIRUTO MUNE GA SASUYOONI ITAMU
2（612点） HASIRUTO UDE GA SASUYOONI ITAMU
3（593点） HASIRUTO KUBI GA SASUYOONI ITAMU

のように，特定の部位が不明確のために正解が第1位にならなかったという場合がほとんどである．

このように発話中の明確に認識できる部位とできない部位を区別することができると，明確に認識できない部位だけを再発話してもらうといった，単に言いっぱなしの一方通行の音声認識ではない双方向の音声認識[5]への道を開くことができる．

この実験とは別に，未知語を含む発話文の解析を行った．使用した文法の終端記号が音素なので，終端記号レベルの未知語は存在せず，ダミーの非終端記号が音素の集まりとしての未知単語を検出できた．

7. まとめ

本稿では，一般LR構文解析法におけるエラー処理について述べた．重要なことは，既存のLR構文解析表とまったく同一の表を使いながら，その表の見方を変えることでさまざまなエラー処理を行っていることである．すなわち，文法にはいっさい手を加えることなくエラー処理ができるということである．自然言語処理において，文脈自由文法に基づく解析には，実用性において限界があることが近年いわれているが，本稿で述べたような操作を採り入れることで，まだまだ有用な手法であると考える．さらに，本稿では文法中の各規則が同じ重みを持っていたが，この手法に確率あるいは統計による手法を組み入れることで，より頑健な解析法へ発展させることができるだろう．

参考文献

1) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers*, Addison Wesley (1986).
2) Ishii, M., Ohta, K. and Saito, H.: An Efficient Parser Generator for Natural Language,

Proc. 15th International Conference on Computational Linguistics (COLING), pp.417–420 (1994).
3) Morii, S., Niyada, K., Fujii, S. and Hoshimi, M.: Large Vocabulary Speaker-independent Japanese Speech Recognition System, *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (1985).
4) Saito, H.: Gap-filling LR Parsing for Noisy Spoken Input: Towards Interactive Speech Recognition, *Proc. International Conference on Spoken Language Processing (ICSLP)*, Kobe, Japan, pp.909–912 (1990).
5) Saito, H.: Interactive Speech Understanding, *Proc. 14th International Conference on Computational Linguistics (COLING)*, Nantes, France, pp.1053–1057 (1992).
6) Saito, H. and Tomita, M.: Parsing Noisy Sentences, *Proc. 12th International Conference on Computational Linguistics (COLING)*, Budapest, Hungary, pp.561–566 (1988).
7) Tanaka, H., Hui, L. and Tokunaga, T.: Incorporation of Phoneme-Context-Dependence in LR Table through Constraint Propagation Method, *Proc. AAAI-94 Workshop Integration of Natural Language and Speech Processing*, pp.15–22 (1994).
8) Tomita, M.: *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, MA (1985).
9) Tomita, M. and Carbonell, J.G.: The Universal Parser Architecture for Knowledge-Based Machine Translation, Technical Report, Center for Machine Translation, Carnegie Mellon University, Pittsburgh (1987).
10) 相澤道雄，徳永健伸，田中穂積：一般化LR法を用いた形態素解析と統語解析の統合，信学技報 (1993).
11) 北 研二，川端 豪，斎藤博昭：HMM音韻認識と拡張LR構文解析法を用いた連続音声認識，情報処理学会論文誌，Vol.31, No.3, pp.472–480 (1990).

(平成7年10月9日受付)
(平成8年5月10日採録)

斎藤 博昭（正会員）

昭和35年生．昭和58年慶應義塾大学工学部数理工学科卒業．平成3年同大学院理工学研究科後期博士課程修了．工学博士．同年同大理工学部数理科学科助手．昭和59年よりカーネギーメロン大学に訪問研究員として滞在し，機械翻訳および音声認識の研究に従事．言語処理学会，ACL各会員．