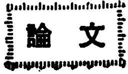

SLR (k) パーザにおける誤り訂正，回復について*

海 尻 賢 二** 打 浪 清 一** 手 塚 慶 一**

Abstract

We have proposed practical error correcting and recovering algorithms for the SLR (k) parsers. First we define the i-order valid pair for a LR (0) table T and a k-terminal string w. Let $(T_0 \cdots T_n, a_l \cdots a_m)$ be an error configuration. If $(T_j, a_h \cdots a_{h+k-1})$ is the i-order valid pair for some $\beta \in V_T{}^i$, we correct above configuration to $(T_0 \cdots T_j, \beta a_h \cdots a_m)$.

If we extend β in the definition above to $\beta \in (V_T \cup V_N)^i$, then we can make error recovery in the same way. Most useful is the case $i=0$ or 1. In these cases the i-order valid pairs can be stored in the SLR (k) parsing table. The SLR (k) parser with these algorithms can parse and correct an input with length n within O(n) times.

We have shown by simulation that these algorithms correct 60~80% of the programs with errors.

1. まえがき

パーザの重要な機能の一つに誤り処理がある．誤り処理に関する研究は理論的な面からは種々の研究が行われ，最小誤り訂正のアルゴリズムもいくつか提案されている[1]．しかしそれらはいずれも $O(n^2)$ 以上またはある程度の backtrack を必要として実用上問題がある．またユーザサイドから考えると必ずしも最小訂正されたプログラムがユーザの意図していたものとは限らない．そこで誤りを構文誤りに限定するならば，パーザの行うべき処理としては次のもので十分であると考える．①パーザが発見した誤りを訂正し，ユーザのデバッキングに要する労力を軽減する．②最小の読みとばしで後の処理を進めることにより，できるだけ多くの誤りをみつける．

以上の点より本論文ではパーザの発見した誤りを，backtrack なしに訂正するシステムを作成することを目的とし，このための SLR (k) パーザに対する誤り訂正アルゴリズム，及び誤り回復アルゴリズムを示す．この2つのアルゴリズムの特徴は，①時間的にも空間的にもパーザに余分な負担をかけないこと，②誤り訂正，回復を含めて全体の処理時間は入力の長さ n に対して $O(n)$ であること，③誤り訂正，回復のための読み飛ばしがないこと，の3点にある．

さらにこれらのアルゴリズムの有効性を示すためにシミュレーションにより擬似的に誤りを持つプログラムを発生させ，それをこれら2つのアルゴリズムに通すことにより誤り処理能力の評価を行い，誤り訂正については 70~80% の訂正率，誤り回復についてはほぼ 100% の回復率を得た．

2. 用語及び諸定義

本章では SLR (k) パーザの概要及び誤り訂正，回復機能について述べる．なお本論文での記法は文献 2) の記法に従い，またパーザの詳細は上記に準ずるものとする．

〔定義 2-1〕

文法Gの任意の生成規則 $A \to \alpha\beta$ に対して $[A \to \alpha \cdot \beta]$ を LR (0) 項目と呼ぶ．LR (0) 項目 $[A \to \alpha \cdot \beta]$ がGのある文型 $\gamma_1\gamma_2$ の prefix γ_1 に対して valid であるとは $\gamma_1 = \gamma_1{}'\alpha$ となることである．ある γ_1 に対して valid な LR (0) 項目の集合を γ_1 に対する LR (0)

---

* A Study of Error Correction and Recovery for SLR (k) parsers by Kenji KAIJIRI, Seiichi UCHINAMI and Yoshikazu TEZUKA (Department of Communication Engineering, Faculty of Engineering, Osaka University).

** 大阪大学工学部通信工学科

表と呼び，T であらわす．また T の集合，即ち文法Gの各文型の prefix に対して valid な LR (0) 表の集合を$\mathcal{T}$であらわす．T及び$\mathcal{T}$の個数は有限である．☒

〔定義 2-2〕 SLR (k) パーザ π

文脈自由文法 $G=(\Sigma, N, P, S)$ に対する SLR (k) パーザは次の 7 つ組で定義する．

$$\pi=(\Sigma_1, Z, O, T_0, \$, f, g)$$

ここで Σ_1 は入力記号の集合で $\Sigma_1=\Sigma\cup\{\$\}$，$Z$ はスタック記号の集合で LR (0) 表の集合$\mathcal{T}$, O は Action 集合で {shift, reduce i, error, accept}，T_0, \$ はそれぞれスタックの初期記号，入力の最終記号，f, g は次のような関数である．

f: 動作関数 $Z\times\Sigma_1{}^K\to O$

g: 行先関数 $Z\times(\Sigma\cup N)\to Z$ ☒

SLR (k) パーザは各 LR (0) 項目集合 T ごとに，T より次のようにして求められる．

（1） 動 作 関 数

$f(T, u)=\text{shift}\longleftrightarrow[A\to\alpha\cdot\beta]\in T$

$u\in EFF_K(\beta\ \text{Follow}_K(A))$

$f(T, u)=\text{reduce } i\longleftrightarrow[A\to\alpha\cdot]\in T$

$A\to\alpha$ が i 番目の生成規則 かつ $u\in\text{Follow}_K(A)$

$f(T, \$^K)=\text{accept}\longleftrightarrow[S'\to S\cdot]\in T$

$f(T, u)=\text{error}\longleftrightarrow$その他の場合

（2） 行 先 関 数

$g(T, X)=T'\longleftrightarrow[A\to\alpha\cdot X\beta]\in T$

かつ $[A\to\alpha X\cdot\beta]\in T'$

ここで$\text{Follow}_K(\beta)=\{w\,|\,S\overset{*}{\Rightarrow}\alpha\beta\gamma$ かつ$w\in\text{First}_K(\gamma)\}$

$EFF_K(\alpha)=\{w\,|\,w\in\text{First}_K(\alpha)$ かつ $\alpha\underset{rm}{\overset{*}{\Rightarrow}}\beta\underset{rm}{\Rightarrow}wx$ なる最右導出があり，β はいかなる非終端語 A に対しても $\beta=Awx$ ではない$\}$，$\text{First}_K(\alpha)=\{w\,|\,w\in\Sigma^*, \alpha\overset{*}{\Rightarrow}w$, $|w|<K$ もしくは $\alpha\overset{*}{\Rightarrow}wx$ かつ $|w|=K\}$ である．各々は Follow 関数，ϵ-free first 関数，first 関数と呼ばれる．

パーザの姿態は表列 $\alpha\in\mathcal{T}^+$ と終端語列 $w\in\Sigma_1{}^+$ により $[\alpha, w]$ と表わす．

〔定義 2-3〕 正当な表列 (Valid table sequence)

LR (0) 表の列 $T_0\ T_1\cdots\ T_n$ が正当であるとは，$[T_0, w_1w_2\$^K]\ |\!\overset{+}{-}\ [T_0\ T_1\ \cdots\ T_n, w_2\$^K]$ なる遷移が存在するような終端語列 w_1 が存在することをいう．但し T_0 は初期 LR (0) 表である． ☒

〔定義 2-4〕 正当な列 (Valid sequence)

LR (0) 表と入力記号の次のような列，

$$T_0\ T_1\ \cdots\ T_n, a_l\ \cdots\ a_m$$

が正当であるとは，次の 2 つの条件が成立することである．

（1） $T_0\ T_1\ \cdots\ T_n$ が正当な表列である．

（2） $[T_0\ \cdots\ T_n, a_l\ \cdots\ a_mw\$^K]\ |\!\overset{+}{-}\ [T_0\ T_1'\ \cdots\ T_p', a_mw\$^K]\vdash$（エラーでない）である． ☒

誤り訂正には種々の考え方があるが本論文では主に使用者にどのような個所が間違っており，かつパーザがどのように誤り個所を処理し解析を進めたか，更に訂正の一方法を与えることを目的とする．そのために誤り訂正アルゴリズムは誤り発見時のみに動作し，正しいプログラムの解析には何の影響も及ぼさないようにする．そのような目的から誤りとしてはパーザが発見する誤りのみに注目し，誤り訂正のための backtrack は行わないこととする．そこで正当な表列と，正当な列を使って正当な誤り訂正と，正当な誤り回復を次のように定義する．

〔定義 2-5〕 正当な誤り訂正

誤り姿態* $[T_0\ T_1\ \cdots\ T_n, a_l\ \cdots\ a_m]$ におけるパーザ π の正当な誤り訂正とは $[T_0\ T_1\ \cdots\ T_n, \alpha a_K\ \cdots\ a_m]$ なる姿態への復帰という．ここで $l\leq K\leq m$ $\alpha\in V_T{}^*$ であり, $T_0\ T_1\ \cdots\ T_n\alpha a_K$ は正当な列である．☒

定義からも明らかなように本論文での誤り訂正は誤った姿態の局所的な訂正であり，$a_l\ \cdots\ a_{K-1}$ と α の置換により訂正を行う．a_1 から a_{l-1} については訂正しない．それゆえこの訂正は最適訂正ではない．

〔定義 2-6〕 正当な誤り回復

誤り姿態 $[T_0\ T_1\ \cdots\ T_n, a_l\ \cdots\ a_m]$ におけるパーザ π の正当な誤り回復とは，$[T_0\ T_1\ \cdots\ T_q\ T'_{q+1}\ \cdots\ T_p', \alpha a_K\ \cdots\ a_m]$ なる姿態への復帰をいう．ここで $l\leq K\leq m, a\in V_T{}^*$ で $0<q<n$ であり，かつ $T_0\ T_1\ \cdots\ T_q\ T'_{q+1}\ \cdots\ T_p'\alpha a_k$ は正当な列である． ☒

以下では実用的見地より SLR (1) パーザに対する誤り訂正，回復について考察する．$k\geq 2$ の場合でも方法は全くかわらない．

3. valid pair に基づく誤り訂正

本章では終端語による valid pair と strictly valid pair の定義を行い，それぞれに基づく誤り訂正法について述べ，合わせてその正当性をも証明する．

〔定義 3-1〕 i-order valid pair

(T, a) に対して次の条件を満足する α 及び γ が存在するとき，(T, a) をパーザ π における i-order valid

---

* $T_0\ \cdots\ T_n$ が正当な表列で，$f(T_n, a_l)=\text{error}$ のとき，$[T_0\ \cdots\ T_n, a_l, \cdots a_m]$ を誤り姿態と呼ぶ．

pair と呼ぶ．但し $T\in\mathcal{T}$，$a\in V_T\cup\{\$\}$，$\alpha,\gamma\in V_T{}^*$，$|\gamma|=i$ である．

任意の $\delta\in V_T{}^*$ に対して

$[T_0,\alpha\gamma a\delta]\vdash_{\pi}^{+}[T_0\ T_1\ \cdots\ T_n,\gamma a\delta]\vdash_{\pi}^{*}[T_0\ T_1'\cdots T_p',a\delta]\vdash_{\overline{\pi}}$（エラーでない）但し $T_n=T$　⊠

(T,a) が γ に対して i-order valid pair であれば，$T_0\cdots T_n\gamma a$ が正当な列となるような正当な表列 $T_0\cdots T_n(T_n=T)$ が存在する．即ち i-order valid pair とはテーブルTと終端語 a の対に対して，その間に入っても矛盾しない終端語列 $\gamma,|\gamma|=i$ が存在することを保証したものである．これを使うことにより次のような訂正法が可能である．

〔アルゴリズム 1〕

i-order valid pair に基づく誤り訂正法

（入力）　誤り姿態 $[T_0\cdots T_n,a_l\cdots a_m]$

（出力）　局所的に訂正された姿態 $[T_0\cdots T_n,\gamma a_p\cdots a_m]$ 但し $p=l$ もしくは $l+1$*

（方法）　① $k=l$ or $l+1,i=0\sim i_n$ の各 k,i について次のことを調べる．すべて No であれば訂正できない．但し i_n は適宜前もって決定しておく．②(T_n,a_k) が i-order valid pair かどうか調べる．Yes ならば③へ，No ならば新しい k,i について調べる．③$T_0\cdots T_n\gamma a_k$ を正当な列とするような長さ i の終端語列 γ が存在するか，存在すれば④へ，存在しなければ別の k,i について調べる．④ $[T_0\cdots T_n,\gamma a_k\cdots a_m]$ に訂正する．　⊠

i-order valid pair ではある正当な表列 $T_0\cdots T_n$ に対して $T_0\cdots T_n\gamma a$ が正当な列となるのであるから，単に (T_n,a) が γ に対して正当であるといってもステップ③でそれが現在の $T_0\cdots T_n$ に対して正当かどうかのテストを行う必要がある．③のステップは①〜④の中でも最も時間のかかるものであり，その回数を減らすのがステップ②の役割である．またこのアルゴリズムは i が大きくなるに従い指数的に計算量が増大するので，ここでは $i=0$ or 1 の場合について本アルゴリズムのステップ②及び③を詳述する．$i>1$ の場合はこれに準ずる．

〔アルゴリズム 1-1〕　(T,a) が i-order valid pair かどうかのテスト（ステップ②）

I．$i=0$ の場合．

$f(T,a)\neq$error ならば "Yes"，$f(T,a)=$error ならば "No" である．

II．$i=1$ の場合

①すべての終端語 b について以下の②〜⑥を実行する．すべての終端語について "No" ならば "No"，"Yes" となるものが1つでもあれば "Yes" をそれぞれ出力する．②NEXT*(T,b) を求め，S とする．③S が空ならば "No" である．④S のすべての要素 T' について以下の⑤，⑥を実行する．すべての要素について "No" であれば "No" を，"Yes" となるものが1つでもあれば "Yes" をそれぞれ出力する．⑤$T''=g(T',b)$ とする．⑥$f(T'',a)\neq$error ならば "Yes" を，$f(T'',a)=$error ならば "No" をそれぞれ出力する．

㊟NEXT*(T,b) の計算法

①$S=\phi$（空）とする．②$f(T,b)=$shift ならば $S=\{T\}$ として stop，error ならばそのまま stop，それ以外は③へ行く．③NEXT (T,b) を求め S_1 とする．但し NEXT$(T,b)=\{T'\,|\,f(T,b)=$reduce i かつ $P_i:A\to\alpha$ であり，ある T'' に対して GOTO(T'',α)**$=T$ かつ GOTO$(T'',A)=T'\}$，④S_1 のすべての要素について次の⑤，⑥を実行する．⑤$f(T',b)=$error ならば $S=S$，$f(T',b)=$shift ならば $S=S\cup\{T'\}$，$f(T',b)=$reduce i ならば⑥へ行く．⑥$S=S\cup$NEXT*(T',b)，但し S_1 が T を要素として持つときには T について⑤，⑥は行わない．　⊠

一般に (T,a) が valid であるような終端語は複数個存在する．そのためアルゴリズム1ではステップ②で valid かどうかを判定した後に，ステップ③であらためてそのための終端語を捜し，正当な列になるかどうかのテストを行う．(T,a) が valid かどうかの判定は上記のように前もって計算できるので表として記憶しておくことができる．

〔アルゴリズム 1-2〕（ステップ③）

①T に対して (T,b) が 0-order valid pair となるような終端語 b を求め，S とする．S のすべての要素について以下の手続き（②〜③）を実行する．"Yes" となるものがあればそれが求める語である．すべて "No" であれば "No" である．②$T_0\cdots T_n$ と先読み記号 b に対するパーザの動作をシミュレートする．b をシフトする前にエラーとなれば "No"，シフトになるとそのときの表を T' とする．③$f(T',a)\neq$error ならば "Yes" を，$f(T',a)=$error ならば "No" を出力する．　⊠

アルゴリズム1では i-order valid pair に基づく誤り訂正法を与えたが，i-order valid pair の性質から

---

* 読み飛しを最小にし，かつアルゴリズムをコンパクトにするために p を l もしくは $l+1$ に限る．

** 行先関数の次のような拡張である．$Z\times(V_N\cup V_T)^+\to Z$

ステップ③が必要となり，時間も長くかかる．そこで次にこのステップ③の操作が必要でないような訂正法を与えるための valid pair を導入する．

〔**定義 3-2**〕 i-order strictly valid pair

(T, a) を i-order valid pair とするような長さ i の終端語列の集合の中に次の条件を満たすものが少なくとも1つ（たとえば α）あれば (T, a) を i-order strictly valid pair という．T_0 と T を両端とする任意の正しい表列を $T_0\ T_1 \cdots T_n\ (T=T_n)$ とするとき，

$$[T_0 \cdots T_n, \alpha a\beta] \vdash_{\pi}^{+} [T_0\ T_1' \cdots T_p', a\beta]$$

$$\vdash_{\pi} (\text{エラーでない}) \qquad \boxtimes$$

(T, a) が α に対して i-order strictly valid Pair であるとは，$T_0 \cdots T_n\ (T_n=T)$ が正当な表列である限り，$T_1 \cdots T_{n-1}$ には関係なく $T_0\ T_1 \cdots T_n\alpha a$ が正当な列であることを示している．即ちテーブルの先頭だけで valid であることがきまる．そのため訂正アルゴリズムが次のように簡単になる．

〔**アルゴリズム 2**〕 i-order strictly valid pair に基づく誤り訂正法

入力，出力はアルゴリズム1と同じである．

（方法） ①$k=l$ or $l+1$，$i=0\sim i_n$ の各 k, i について次のことを調べる．すべて "No" であれば訂正できない．"Yes" となるものが1つでもあればそれにより訂正を行う．但し i_n は前もってきめておく．②(T_n, a_k) が i-order strictly valid pair かどうかを調べる．"No" ならばくりかえし，"Yes" ならば長さ i の終端語列 α を出す． ⊠

アルゴリズム2では任意の表と終端語の対 (T, a) が strictly valid pair であるかどうかを次のアルゴリズム3で前もって計算しておく．するとステップ②は表のルックアップのみでよい．ここでもアルゴリズム1の場合と同様に $i=0$ or 1 の場合についてのみ述べる．$i>1$ の場合はこれに準ずる．

〔**アルゴリズム 3**〕 (T, a) が i-order strictly valid pair かどうかの判定．

アルゴリズム3はアルゴリズム 1-1 とほぼ同じであるのでここでは異なる個所のみを述べる．

ステップ④ S のすべての要素 T' に対して以下の手続き（⑤，⑥）を実行する．"No" となるものが1つでもあれば "No" を，すべて "Yes" ならば "Yes" と b を出力する． ⊠

i-order valid pair 及び i-order strictly valid pair の定義より次の定理が導かれる．

〔**定理 1**〕 i-order valid pair による誤り訂正は局所的に正当である． ⊠

〔**定理 2**〕 i-order strictly valid pair による誤り訂正は局所的に正当である． ⊠

アルゴリズム 1,2 ともに (T, a) が valid pair か，また strictly valid pair かどうかは，$i=1$ の場合は f 関数に貯えることができる．即ち 0-order valid pair を優先し，0-order valid でない 1-order valid pair のみ記憶するとすれば，f 関数のエラーエントリに貯えることが可能である．次にそのようにして表わした parsing table の例を示す．

例 1. 誤り訂正のための情報を貯えた SLR (1) parsing table

$G_1=\langle \{E, T, F\}, \{a, +, *, (,)\}, P, E\rangle$

P: $E\to E+T^{(1)}$ $E\to T^{(2)}$

$T\to T*F^{(3)}$ $T\to F^{(4)}$

$F\to (E)^{(5)}$ $F\to a^{(6)}$

G_1 の SLR (1) parsing table を **Fig. 1** に示す．Fig. 1 において，数字はシフト動作とその行先の表，R_i は reduce i, A は accept を表わす．また終端語はその行の表と列の終端語に対して 1-order (strictly) valid pair となることを表わす．空白は valid pair とならない部分である．なお G_1 では valid pair と strictly valid pair が一致するので区別しない．

例えば $(T_5, *)$ について調べると，

$\mathrm{NEXT}^*(T_5, a)=\{T_4\}$ $\mathrm{NEXT}^*(T_5, (\ )=\{T_5\}$

$g(T_5, a)=T_4$ $g(T_5, (\ )=T_5$

かつ $f(T_4, *)=\text{reduce } 6$，$f(T_5, *)=\text{error}$ である．故に $(T_5, *)$ は 'a' に対して 1-order strictly valid pair である．しかし '(' に対しては valid でない．また $(T_1, *)$ と $(T_1,))$ は Fig. 1 よりわかるように 0-order valid でもないし，1-order valid でもない．そこで 2-order valid pair かどうか調べると

	E	T	F	a	+	*	(	)	$
0	1	2	3	4	a	a	5	a	a
1				+	6		+		A
2				*	$R2$	7	*	$R2$	$R2$
3				+	$R4$	$R4$	*	$R4$	$R4$
4				+	$R6$	$R6$	*	$R6$	$R6$
5	8	2	3	4	a	a	5	a	a
6		9	3	4	a	a	5	a	a
7			10	4	a	a	5	a	a
8				+	6	)	+	11	)
9				*	$R1$	7	*	$R1$	$R1$
10				+	$R3$	$R3$	*	$R3$	$R3$
11				+	$R5$	$R5$	*	$R5$	$R5$

Fig. 1 SLR (1) parsing table with error correct entries

"$+a$" なる終端語列に対して valid である. ☒

4. valid pair に基づく誤り回復

3. では終端語列を対象とした valid pair を定義し，それに基づく誤り訂正法を示した．本章では更に非終端語まで含めた valid pair を定義し，それに基づく誤り回復法について述べる．

〔**定義 4-1**〕 i-order valid pair

(T, a) に対して次の条件を満足する γ が存在するとき，(T, a) をパーザ π における i-order valid pair と呼ぶ．但し $T \in \mathcal{T}$, $a \in V_T \cup \{\$\}$, $\gamma = (V_N \cup V_T)^i$ である．T を先頭とする任意の正しい表列 $T_0\ T_1 \cdots T_n$ $(T_N = T)$ に対して，

$$[T_0\ T_1 \cdots T_n, \gamma a \delta] \vdash_{\pi}^{*} [T_0\ T_1' \cdots T_p', a\delta] \vdash_{\pi} (\text{エラーでない})$$ ☒

この一般化した valid pair に基づく誤り処理は非終端語を使うために誤り訂正とはならない．ここで特に有用なのは $\gamma \in V_N$ の場合である．$\gamma \in V_N$ の場合には valid pair と strictly valid pair が一致する．

〔**アルゴリズム 4**〕 非終端語に対する 1-order valid pair の計算

(方法) ① T に対して $g(T, A)$ が値を持つような非終端語 A が存在すれば②へいく，存在しなければ "No" である．② $g(T, A) = T'$ として，$f(T', a) = \text{error}$ ならば "No" であり，$\neq \text{error}$ であれば③へ行く．③ ①, ②で求めた非終端語に対して (T, a) は 1-order valid pair である． ☒

次にこの valid pair に基づく誤り回復アルゴリズムを述べる．これは文献 2) の練習問題 7-4-28 での回復法を SLR (K) パーザ用に変更したものになっている．

〔**アルゴリズム 5**〕 valid pair に基づく誤り回復

(入力) 誤り姿態 $[T_0\ T_1 \cdots T_n, a_l \cdots a_m]$

(出力) 局所的に正当な姿態 $[T_0 \cdots T_p\ T, a_k \cdots a_m]$

(方法) ① $i = l \sim m$ まで以下の手続きをくりかえす．② $j = n \sim 0$ まで以下の手続きをくりかえす．③ $[T_j, a_i]$ が 1-order valid pair か，"Yes" ならば④へ．"No" ならば新たな i, j についてくり返す．④ $[T_j, a_i]$ を valid とするような非終端語を A とする．⑤ $g(T_j, A) = T$ として $[T_0 \cdots T_j\ T, a_i \cdots a_m]$ に回復する． ☒

次の定理の成立は明らかである．

〔**定理 3**〕 アルゴリズム 5 に基づく誤り回復は正当である． ☒

アルゴリズム 4 によって求めた 1-order valid pair による誤り回復では場合により誤り回復がループに陥る．即ち $[T_0\ T_1 \cdots T_n, a_l \cdots a_m] \vdash \text{error}$ としたとき，(T_{n-1}, a_l) で回復するとする．(T_{n-1}, a_l) が A に対して valid で $g(T_{n-1}, A) = T$ かつ $f(T, a_l) = \text{reduce } i$, P_i: $B \rightarrow A$ $g(T_{n-1}, B) = T_n$ となるなら同じ回復をくり返し，解析は前へ進まない．この点を改善するためにアルゴリズム 4 を一部変更する．①, ②はそのままで③のみ次のように変更する．

③′ $f(T', a) = \text{shift}$ ならば，(T, a) は 1-order valid pair である．$f(T', a) = \text{reduce } i$ ならば対応する規則により 2 つの場合にわける，但し P_i: $A \rightarrow \alpha_i$ とする．

③′-1 $|\alpha_i| = 1$ このときには NEXT (T', a) を求め，その各要素について③′を再び実行する．そしてすべて "Yes" ならば "Yes" である．

③′-2 $|\alpha_i| \neq 1$ このときは "Yes" である．

③′-1 の場合にはアルゴリズム 4 の①, ②を満す項目が複数個あり，$[A_i \rightarrow \cdot A_{i-1}]$ なる形をしていると考えられる．この場合には最上位の A_i, 即ち $(A_n \rightarrow A_{n-1} \rightarrow \cdots \rightarrow A_1)$ なる A_n を選んでおけばよい．これを行っているのが③′である．

例 2. G_1 の SLR (1) パーザにおける，非終端語に対する 1-order valid pair

$(T_0, +)$ について考える．Fig. 1 より $g(T_0, A)$ が値を持つ非終端語 A は E, T, F であり，それぞれ値は T_1, T_2, T_3 である．$f(T_1, +) = \text{shift}$, $f(T_2, +) = \text{reduce } 2$, $f(T_3, +) = \text{reduce } 4$ であるので E を選ぶ．また $(T_5, \$)$ についてみるとやはり E, T, F に対して値を持ち，それぞれ T_8, T_2, T_3 である．$f(T_8, s) = \text{error}$ より E はだめ，$f(T_2, \$) = \text{reduce } 2$ であるので $\text{NEXT}^*(T_2, \$) = \{T_1, T_8\}$, $f(T_1, \$) = \text{Accept}$ であるが

	a	$+$	$*$	(	)	\$
0		E	T			E
1						
2						
3						
4						
5		E	T		E	
6		T	T		T	T
7		F	F		F	F
8						
9						
10						
11						

Fig. 2 1-order valid pairs for nonterminal symbols

$f(T_8, \$)$=error であるから T もだめである．T_3 についても同様にだめなことがわかる．よって $(T_5, \$)$ は 1-order valid pair とならない．　⊠

5. シミュレーションによるアルゴリズムの評価

本章では誤り処理アルゴリズムの評価のために行ったシミュレーションについて述べ，それによりアルゴリズムの評価を行う．誤り処理アルゴリズムの評価には多くの誤りプログラムが必要であるが，例題として選んだ文法が小さいので非常に困難である．そこで擬似的にエラープログラムを **Fig. 3** の方式で発生させることとした．Fig. 3 において誤りの個数，誤りの種類，誤りの位置及び誤った終端語はそれぞれ乱数により決定した．実際の誤りは文脈にある程度依存するがここでは簡単のため文脈は考えずに，誤りの終端語の発生確率のみいろいろ変えて実験を行った．また誤りの個数はプログラムの長さによる上限 (1/5, 1/10, 1/20) を設け，その範囲内で一様に分布させた．一番最初のプログラムは正しいプログラムとし，誤りの種類は単一の終端語の消去，挿入，置換の 3 つとし，それぞれ元のプログラムを次のように変更する．

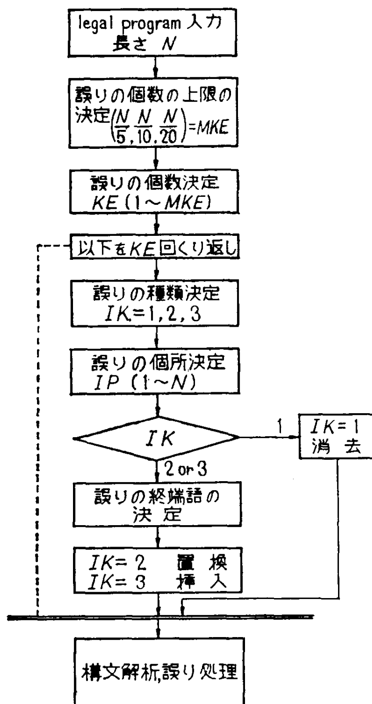

Fig. 3 Flow of the simulation for error correction

```
G₂: <Program>   →<Block>
    <Block>     →<Blockhead><Blockbody>END
    <Blockhead> → BEGIN|<Blockhead><Decl.>;
    <Decl.>     → TYPE id|<Decl.>,id
    <Blockbody> →<Statement>|<Blockbody>;<Statement>
    <Statement> →<Simplestate.>|<Ifstate.>
    <Simplestate.>→id=<Exp.>|<Block>
    <Ifstate.>  → If<Exp.>then<Simplestate.>else<Statement>
    <Ifstate.>  → If<Exp.>then<Statement>
    <Exp.>      →<Term>|<Term>+<Exp.>
    <Term>      → id|(<Exp.>)
```

Fig. 4 Test grammar

```
PROGRAM 1.
 Begin Type a,a; a=a;
       If a+a then Begin a=a+a; a=a End
              else a=a+a
 End

PROGRAM 3.
 Begin Type a,a; a=a+a;
       If a then Begin Type a; a=a+a; a=a+(a+a) End
            else Begin a=a+a+(a+a); a=a End;
       a=a
 End

PROGRAM 2.
 Begin Type a,a; Type a; a=(a+a);
       Begin Type a; a=a+a+a End;
       If a then a=a else a=a+a; a=a
 End

PROGRAM 4.
 Begin Type a,a; a=a+(a+a);
       If a+a then Begin a=a; a=(a+a) End
              else Begin a=(a+a); a=a End;
       Begin Type a;
             If a then a=a else a=a+a;
             a=a+a
       End
 End
```

Fig. 5 Four test programs

Table 1 Error probabilities for each terminal symbol

	⊥	End	Beign	;	Type	a	,	=	If	then	else	+	(	)
1.	1	2	2	5	2	6	5	5	2	2	2	5	6	6
2.	0	1	1	5	2	6	5	5	2	2	2	5	5	6
3.	1	1	1	10	1	10	10	10	1	1	1	10	10	10

消去　$(a_1 \cdots a_{l-1}\ a_l\ a_{l+1} \cdots a_m) \rightarrow$

$$(a_1 \cdots a_{l-1}\ a_{l+1} \cdots a_m)$$

置換　$(a_1 \cdots a_{l-1}\ a_l\ a_{l+1} \cdots a_m) \rightarrow$

$$(a_1 \cdots a_{l-1}\ a_{l-1}\ a\ a_{l+1} \cdots a_m) \quad a_l \fallingdotseq a$$

挿入　$(a_1 \cdots a_{l-1}\ a_l\ a_{l+1} \cdots a_m) \rightarrow$

$$(a_1 \cdots a_{l-1}\ a\ a_l\ a_{l+1} \cdots a_m)$$

但し置換及び挿入における a の決定は **Table 1** の 3 種類の確率分布を想定して実験を行った．例として用いた文法及びその文法に基づく 4 つのプログラムを **Fig. 4**, **Fig. 5** に示す．

Table 2 Simulation result for error correction

誤り確率 \ 上限 \ プログラム	I			II			III		
	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$
1	56	82	86	65	77	90	67	79	89
2	59	80	85	53	75	83	56	77	83
3	57	69	82	53	77	80	55	75	85
4	41	65	80	52	65	78	46	62	81

実験での誤り訂正アルゴリズムは次のような優先度で valid pair を SLR (1) parsing table に記憶させた.

① 0-order valid pair, ② 1-order valid pair, ③ 1-order valid pair (非終端語)

③は誤り回復のためのものである. parsing table の大きさは 36×15, その内上記 3 つのどの pair にもなっていない要素は 295 個であり, 約半数である. 誤り訂正は次の 3 つの形でのみ行った.

誤り姿態 ($T_0\ T_1 \cdots T_n, a_l\ a_{l+1} \cdots a_m$)

1. 消去による訂正 ($T_0\ T_1 \cdots T_n, a_{l+1} \cdots a_m$)
2. 置換による訂正 ($T_0\ T_1 \cdots T_n, a\ a_{l+1} \cdots a_m$)
3. 挿入による訂正 ($T_0\ T_1 \cdots T_n, a\ a_l\ a_{l+1} \cdots a_m$)

Fig. 5 の 4 種類のプログラム, Table 1 の 3 種の誤り確率についてそれぞれ 100 個の誤りプログラムを発生させ, 誤り訂正を試みた結果を **Table 2** に示す.

Table 2 はプログラムの長さが長くなるにつれて, また誤りの個数の上限を上げるにつれて訂正率が下がる傾向を示している. この一つの原因は実験方法にあると考えられる. 即ち誤りの発生をランダムに文脈を無視して行うために誤りの絶対数の増大に伴い, 訂正不可な誤りがふえてくる. たとえば Begin の消去, End の挿入等によるブロック構造の完結がある. また文脈を考慮していないために誤り確率の相違は訂正率にさほど影響を与えていない. 現実のプログラムでの誤りを 100 ステートメント中, 10 個程度と考えると, 上限は 1/20 で十分である. この場合には訂正率も 80〜90% になり, またプログラムの長さに伴う訂正率の低下もわずかである. このことから誤り訂正に関してはアルゴリズムは満足のゆくものである. 本例はすべて 100 個のプログラムについての値であるが 1,000 個について行っても大きく値は変わらない.

訂正不能な 20% については誤り回復を合せて行うことで解析を進めることができる. 誤り回復のために非終端語に対する 1-order valid pair を組み込んで実験を行った結果 100% 回復可能であった. また回復のための読み飛ばしは, 最後まで読み飛ばす場合 (73 回中 22 回) を除くと平均 2.7 語であった.

実験で用いた文法は小さいものであるがアルゴルの基本的な構造は持っている. このことから上記の結果は実用の文法に対してもあてはまるものと考える.

6. むすび

本論文では LR (0) 表と終端語の対に対して valid pair を定義し, それを利用した SLR (k) パーザにおける誤り訂正及び回復のアルゴリズムを示した. パーザが実用可能な SLR (k) パーザであること, 余分な空間を必要としないこと, また誤り処理を含めた処理時間が入力に対して線型であることから十分実用になるものと考える.

本論文での実験では 1-order valid pair までしか使用しなかったが, 2 以上の valid pair を使用するならば更に訂正率は改善できる. しかしその際には記憶量の増大はさけられない.

参考文献

1) A. V. Aho, T. G. Peterson: A Minimum Distance Error Correcting Parsing for Context-Free Languages, SIAM J. Computer, Vol. 1, No. 4, pp. 305〜312 (1972).
2) A. V. Aho, J. D. Ullman: The Theory of Parsing, Translation, and Compiling, Prentice-Hall ch 5, 7 (1972).
3) 海尻, 打浪, 手塚: LR (k) パーザにおける誤り回復について, 49 年度情報処理全大 15.

(昭和 51 年 4 月 12 日受付)
(昭和 51 年 7 月 22 日再受付)