A Study of Error Correction and Recovery for SLR(k) Parsers

Kenji KAIJIRI*, Seiichi UCHINAMI** and Yoshikazu TEZUKA**

Abstract

We have proposed the 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 $(T_0 \ldots T_n, a_j \ldots a_m)$ be an error configuration. If $(T_n, a_h \ldots a_{h+k-1})$ is the i-order valid pair for some $\beta \varepsilon \Sigma^i$, we correct the above configuration to $(T_0 \ldots . T_n, \beta a_h \ldots a_m)$. If we extend β in the definition above to $\beta \varepsilon (N \cup \Sigma)^i$, 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) time.

We have shown by simulation that the algorithm corrects60-80% of the programs with errors.

1. Introduction

One of the important functions of parsers is error processing (error correction and recovery). Some theoretical researches about least error correction have been done, but these algorithms reqiure $O(n^3)$ time or require backtracking; so they are not adequate for practical use. Considering from the users' side, the minimum corrected program is not necessarily the program that users intended to make. We consider error processing from a practical point of view, so we suppose the task of error processing is the following:

(1) To correct the parser defined error and reduce the cost of debugging.

(2) To make the eliminated portion by recovery short and detect as many errors as possible.

In this paper, we consider error processing for parser defined errors without backtracking and propose the error correcting and recovering algorithms for SLR(k) parsers. They have the following characterisics:

This paper first appeared in Japanese in Joho-Shori (Journal of the Information Processing Society of Japan), Vol. 18, No. 3 (1977), pp. 230~236.

* Faculty of Engineering, Shinsyu University
** Faculty of Engineering, Osaka University

(1) Error correction and recovery are invoked by procedure call when an error is detected; so the parsing of legal programs is not affected.

(2) They correct and recover within O(n) time.

(3) Elimination part of program by error recovery is smaller than that of ordinarily used methods.

2. Fundamental Concepts

In this section, we define the valid error correction and recovery. The notation of SLR(k) parsers are the same as in [2]. The next two definitions are essential.

[definition 1] Valid Table Sequence

We say that the sequence of LR(0) tables, $T_0 \ldots T_n$, is a valid table sequence if there exists a terminal string w_1 such that $[T_0, w_1 w_2 \$^k] \overset{+}{\vdash} [T_0 \ldots T_n, w_2 \$^k]$, where T_0 is an initial LR(0) table.//

[definition 2] Valid Sequence

We say that the sequence of LR(0) tables followed by a terminal string, $T_0 \ldots T_n a_i \ldots a_m$, is a valid sequence if the following two conditions hold:

(1) $T_0 \ldots T_n$ is a valid table sequence.

(2) $[T_0 \ldots T_n, a_i \ldots a_m w \$^k] \overset{+}{\vdash} [T_0 T_1' \ldots T_p', a_m w \$^k] \vdash$ (not error).//

[definition 3] Valid Error Correction

The transformation from an error configuration $[T_0 \ldots T_n, a_i \ldots a_m]$ to a nonerror configuration $[T_0 \ldots T_n, \alpha a_k \ldots a_m]$ is a valid error correction if $i \le k \le m$, $\alpha \varepsilon \Sigma^*$ and $T_0 \ldots T_n \alpha a_k$ is a valid sequence.//

This correction is a local error correction because $a_1 \ldots a_{i-1}$ is not changed.

3. Error Correction by Valid Pairs

[definition 4] i-order valid pairs

We say that (T,a) is an i-order valid pair for a parser Π if there exist α and γ holding the following condition: for any $\delta \varepsilon \Sigma^*$, $[T_0, \alpha\gamma a\delta] \overset{+}{\vdash}_{\pi} [T_0 \ldots T_n, \gamma a \delta] \overset{+}{\vdash}_{\pi} [T_0 T_1' \ldots T_p', a\delta] \vdash_{\pi}$ (not error), where $T_n = T$, $a \varepsilon \Sigma \cup \{\$\}$, $\alpha, \gamma \varepsilon \Sigma^*$, and $|\gamma| = i$.//

If (T,a) is an i-order valid pair for some $\gamma \varepsilon \Sigma^*$, then there exists a valid table sequence $T_0 \ldots T_n (T_n = T)$ such that $T_0 \ldots T_n \gamma a$ is a valid sequence, that is, i-order validness guarantees that γ can be inserted between T and a. Fig.1 is the error correcting algorithm using i-order valid pairs.

Even if (T_n, a) is an i-order valid pair for γ, $T_0 \ldots T_n \gamma a$ is a valid sequence only for the particular valid table sequence $T_0 \ldots T_{n-1}$. It is necessary to check whether

γ is valid for the current table sequence. TVS does this check and is the most time consuming. We describe in detail TVP (Fig.2) and TVS (Fig.3) for the case of i=0 or 1 (i_n=1). For the case of i_n>1, the algorithms are almost the same as these.

Procedure TVS is dependent on the current table sequence, and is very time consuming. We define strictly restricted valid pairs in order to give more efficient algorithm.

[definition 5] i-order strictly valid pair

We say that (T,a) is an i-order strictly valid pair for a parser Π if there exists at least one terminal string α of length i which satisfies the following conditions;

(1) (T,a) is an i-order valid pair for α.

```
Procedure ERROR CORRECTION
 Begin comment input [T0...Tn,aj...am]
       output [T0...Tn,γap...am];
  For k=j to j+1 do
   For i=0 to in do
    If (Tn,ak) is an i-order valid pair..I
     Then If there exists γ such that T0..
           .TnγaK is a valid sequence...II
           Then Goto SUCCEED;
  error correction fails and "No";
  SUCCEED:correct to [T0...Tn,γak...am]
 End comment procedure I is TVP(T,a,i)
     procedure II is TVS(TS,a,i,γ) and
     TS is T0...Tn;
```

Fig.1 Error correction using i-order valid pairs

```
Procedure TVS(TS,a,i,γ) comment if there
 exists γ such that T0...Tnγa (T0...Tn=
 TS) is a valid pair and |γ|=i then TRUE
 else FALSE;
 Begin T=top of TS;
  S={b∈Σ|(T,b) is 0-order valid pair};
  TVS=FALSE;
  If S≠empty Then
   For all b in S do Begin γ=b;
    SMT(TS,b,T1);
    If f(T1,b)≠error Then Begin
     T2=g(T1,b);
     If f(T2,a)≠error Then TVS=TRUE
    End
   End
 End comment SMT(TS,b,T1) computes the
     following Tp'=T1, [T0...Tn,bα]⊢*[T0
     T1'...Tp',bα]⊢(shift or error);
```

Fig.3 Procedure TVS

```
Procedure TVP(T,a,i) comment if (T,a) is
 an i-order valid pair then TRUE else
 FALSE;
 Begin set S initial empty;
  Case i of
   0:If f(T,a)≠error Then TVP=TRUE Else
      TVP=FALSE;
   1:Begin L1=FALSE;
      For all b in Σ do Begin
       S=NEXT*(T,b);
       If S≠empty Then Begin L2=FALSE;
        For all T1 in S do Begin
         T2=g(T1,b);
         If f(T2,a)≠error Then L2=TRUE
        End
        If L2 Then L1=TRUE
       End
      End;
      If L1 Then TVP=TRUE Else TVP=False
     End
 End comment this procedure is a test
     whether (T,a) is an i-order valid
     pair for some b. f is an action and
     g is a goto function;

Procedure NEXT*(T,b)
 Begin set S initial empty;
  Case i of
   shift:S={T};
   error:S=S;
   reduce:Begin U=NEXT(T,b);
           For all T1 in U except T do
            Case f(T1,b) of
             error:S=S;
             shift:S=S {T1};
             reduce:S=S NEXT*(T1,b);
          End;
  NEXT*=S
 End comment NEXT(T,b)={T1|there exists
     T2 such that f(T,b)=reduce i, Pi:A
     →α, g(T2,α)=T, and g(T2,A)=T1};
```

Fig.2 Procedure TVP

(2) For any $T_0 \ldots T_{n-1}$ such that $T_0 \ldots T_n$ is a valid table sequence ($T=T_n$), $[T_0 \ldots T_{n-1} T, \alpha a\beta] \vdash^{+}_{\pi} [T_0 T'_1 \ldots T'_p, a\beta] \vdash_{\pi}$ (not error).//

If (T,a) is an i-order strictly valid pair for α, then $T_0 \ldots T_n \alpha a$ ($T_n=T$) is a valid sequence whenever $T_0 \ldots T_n$ is a valid table sequence. If we use this pair, we may look only at the topmost table (T_n). The error correcting algorithm by i-order strictly valid pairs is in Fig.4. Whether (T,a) is an i-order valid pair is determined in advance only by (T,a,i), so the test in III (Fig.4) is done by table look up. Procedure TSVP (Fig.5) tests i-order validness for i=0 or 1. This information can be stored in f-function of SLR(k) parsing table.

Example. Error correction in a SLR(1) parser

Consider the SLR(1) grammar G as follows: G=<{E,T,F},{a,+,*,(,)},P,E>

P: 1) E → E+T 2) E → T 3) T → T*F 4) T → F 5) F → (E) 6) F → a

We show the SLR(1) parsing table with error correcting entries in Fig.6. In Fig.6,

M[i,B]=j means $f(T_i,B)$=shift and $g(T_i,B)=T_j$

M[i,B]=R_k means $f(T_i,B)$=reduce k

M[i,B]=A means $f(T_i,B)$=accept

M[i,B]=a means (T_i,B) is an i-order valid pair for a.//

γ in definition 4 can be extended to the element in $(\Sigma \cup N)^i$ easily. In this case, the former algorithm can be used as an error recovery algorithm with some modifications.

5. Evaluation and Conclusion

We have evaluated these algorithms by simulation. We have chosen three factors for this evaluation: 1) programs' length, 2) the number of errors (this is determined randomly and three kinds of upper bounds are

```
Procedure ERROR CORRECTION
 Begin For k=j to j+1 do
   For i=0 to i_n do
     If (T_n,a_k) is an i-order strictly
      valid pair for some α.........III
      Then Goto SUCCEED;
  error correction fails and "No";
  SUCCEED:correct to [T0...T_n,αa_k...a_m]
 End comment procedure III is
      TSVP(T,a,i,α);
```

Fig.4 Error correction using i-order strictly valid pairs

```
Procedure TSVP(T,a,i,α) comment if (T,a)
 is an i-order valid pair for some α
 then TRUE else FALSE;
 Begin set S initial empty;
  Case i of
   0:If f(T,a)≠error Then TSVP=TRUE
                     Else TSVP=FALSE;
   1:Begin L1=FALSE;
      For all b in Σ do
       Begin S=NEXT*(T,b);
        If S≠empty Then Begin L2=FALSE;
         For all T1 in S do Begin
          T2=g(T1,b);
          If f(T2,a)=error Then L2=FALSE
         End;
         If L2 Then Begin L1=TRUE; α=b
                    End
        End
       End;
       If L1 Then TSVP=TRUE Else TSVP=FALSE
     End
 End
```

Fig.5 Procedure TSVP

	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		8	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.6 SLR(1)parsing table for G

<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<Statement>|IF<Exp.> THEN<Simplestate.>ELSE<Statement>
<Exp.> →<Term>|<Term>+<Exp.>
<Term> →id|(<Exp.>)

Fig.7 Test grammar

chosen), 3)an error probability for each terminal symbol. We made a program which produces an illegal program according to the above three factors. The test grammar is shown in Fig.7 and the results in table.1. Each value is the number of corrected programs for 100 illegal programs. The error boundary 1/20 is the most practical case. In this case, the correcting ratios are 80-90% and decrease of these ratios accompanying with increase of the programs' length is small compared with other cases. The remainder which can not be corrected by this algorithm can be recovered by the above mentioned recovering algorithm.

We have shown the error correcting and recovering algorithms for SLR(k) parsers. They reqiure no extra memory and parse an illegal program with length n within O(n) time.

Table 1 Simulation results of error correction

A	I			II			III		
B / C	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$
1(length=33)	56	82	86	65	77	90	67	79	89
2(length=47)	59	80	85	53	75	83	56	77	83
3(length=58)	57	69	82	53	77	80	55	75	85
4(length=75)	41	65	80	52	65	78	46	62	81

A..error probabilities
B..error bound
C..input program

REFERENCES

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