AN IMPROVEMENT TO

IMMEDIATE ERROR DETECTION IN STRONG LL(1) PARSERS

by

J. Mauney
C. N. Fischer

Computer Sciences Technical Report #392

July 1980

An Improvement to Immediate Error Detection in Strong LL(1) Parsers*

J. Mauney

C. N. Fischer

University of Wisconsin-Madison, Madison, WI 53706, USA

parsing, LL(1), syntactic error detection, error correction

1. Introduction

Many error-correction methods [2,3] require that the associated parser possess the immediate error detection property; that is, that any error is announced as soon as the erroneous symbol is encountered. Strong LL(1) parsers do not have the immediate error detection property and may perform incorrect parsing actions before an error is announced [1].

A simple and efficient algorithm for immediate error detection in a subset of Strong LL(1) grammars has been previously presented [4]. In this paper we extend the technique to provide immediate error detection for any Strong LL(1) grammar.

2. Immediate Error Detection

As in [4], we note that the only incorrect moves a Strong LL(1) parser can make are a series of predictions

*Research supported in part by National Science Foundation Grant MCS78-0257

with the net effect of deriving λ (where λ denotes the empty string). Immediate error detection is provided by checking such predictions before they are performed, to insure that the input symbol will be accepted by some symbol lower on the stack. In [4], grammars were restricted, so that any derivation of λ occurred in one step. We now extend the method to all Strong LL(1) grammars.

Lemma 2.1. In a Strong LL(1) parser, the only moves an error symbol, a, can induce are predictions of the form $A \rightarrow \alpha$ where $\alpha \Rightarrow^{*} \lambda$.

Proof. Clearly, the only moves an error symbol can induce are predictions. By construction of Strong LL(1) parsers, a production of the form $A \rightarrow \alpha$, where $\alpha \not\Rightarrow^{*} \lambda$, can only be predicted for an input symbol, a, if $a \in First(\alpha)$. But this means that $\alpha \Rightarrow^{*} a\beta$, and such a prediction is obviously correct. ☒

Lemma 2.2. In a Strong LL(1) parser, assume an input symbol, a, induces a prediction of a production $A \rightarrow \alpha$, where $\alpha \Rightarrow^{*} \lambda$. Then $a \in Follow(A)$ if and only if $a \notin First(\alpha)$.

Proof. (only if) Assume $a \in Follow(A)$ and $a \in First(\alpha)$. Then for some X, such that $\alpha \Rightarrow^{*} X\alpha'$, there must be two productions $X \rightarrow \gamma$ and $X \rightarrow \delta$, with $\delta\alpha' \Rightarrow^{*} \lambda$ and $\gamma \Rightarrow^{*} a\gamma'$. Therefore $a \in First(\gamma Follow(X)) \sqcup First(\delta Follow(X))$ (since $a \in Follow(X)$), and the grammar cannot be Strong LL(1).

(if) If $a \notin Follow(A)$ and $a \notin First(\alpha)$ then the parse table

entry for (A, a) should be 'error', not 'predict'. ☒

Lemma 2.2 tells us that if A is the current stacktop, and a is the next input symbol, then A can derive λ (in this context) only if $a \in$ Follow(A). Furthermore, if $a \in$ Follow(A) then A must derive λ (directly or indirectly). In the case that a is an error symbol, an incorrect prediction $A \rightarrow \alpha$ can only occur if $\alpha \stackrel{*}{\Rightarrow} \lambda$ (Lemma 2.1) and $a \in$ Follow(A) (Lemma 2.2). Thus if the Strong LL(1) parse table entry for the pair (A,a) is "predict $A \rightarrow \alpha$" and $\alpha \stackrel{*}{\Rightarrow} \lambda$ and $a \in$ Follow(A) then we need to check that the prediction is correct; in all other cases the parse action indicated must be correct. When a Strong LL(1) parse table is constructed, we mark those predictions that require special checking (for example, an entry, +p, can denote a prediction of production p without a mark, and -p can denote a prediction of p with a mark). Algorithm 2.3 is a Strong LL(1) parser, modified to check for marked productions. Before a marked prediction is performed, Algorithm 2.4 is invoked. This algorithm checks that the prediction is correct by verifying that after A (and possibly other nonterminals) derives λ, a will be matched, or derived by, some deeper stack symbol. Both algorithms are adapted from [4].

We note that a single input symbol may induce a number of parse actions before finally being matched. However, after Algorithm 2.4 has been run it is guaranteed that the symbol will be accepted (or that it is an error symbol). We

can exploit this by setting a flag when Algorithm 2.4 is called. As long as the flag remains set, we needn't perform the check again. The flag is cleared whenever a symbol is matched. As noted in [4], this modification allows us to guarantee a linear-time parse.

Algorithm 2.3. (A parser for Strong LL(1) grammars).

{Assume X_m ... $X_\emptyset$ is the parse stack with X_m the top element. Let a be the current input symbol. T is the parser action table}

```
flag := false;
loop { forever }
  case T[X_m,a] of
    POP: Pop X_m from parse stack;
         flag := false;
         read next input symbol;
    ACCEPT: Terminate parser;
    ERROR: Invoke error recovery;
    PREDICT: {assume prediction is X_m --> α}
      if not Marked(T[X_m,a]) cor flag cor CheckPrediction
      then Pop X_m from stack;
           Push α onto stack
      else Invoke error recovery;
      fi;
  end; { case }
end; { loop }
```

Note that we use the conditional or operator, cor, to insure that CheckPrediction is invoked only when necessary.

Algorithm 2.4.

```
(check whether a marked prediction is correct)
function CheckPrediction : boolean;
{X_m ... X_Ø is the parse stack, a is the input,
  X_Ø is the end-marker}
  for i := m-1 downto Ø do
    case T[X_m,a] of
      POP, ACCEPT: flag := true;
        return(true);
      ERROR: return(false);
      PREDICT:
        if not Marked(T[X_m,a]) then
          flag := true;
          return(true)
        { else continue forloop }
        fi;
    end; { case }
  end; { for }
end; { function CheckPrediction }
```

3. Conclusion

We have presented an improvement to the method of [4] which provides immediate error detection for all Strong LL(1) grammars. The method is very simple to use in Strong

LL(1) parsers. In practice it requires no increase in parse table size, and no appreciable decrease in parsing speed.

4. References

[1] A.V. Aho and J.D. Ullman, The Theory of Parsing, Translation and Compiling, Vol. 1 (Prentice Hall, Englewood Cliffs, NJ, 1972) Section 5.1, 334-361.

[2] C.N. Fischer, B.A. Dion and J. Mauney, A Locally Least-cost LR Error-Corrector, TOPLAS, to appear.

[3] C.N. Fischer, D.R. Milton and S.B. Quiring, Efficient LL(1) Error Correction and Recovery Using Only Insertions, Acta Informatica 13 (1980), 141-154

[4] C.N. Fischer, K.C. Tai and D.R. Milton, Immediate Error Detection in Strong LL(1) Parsers, Information Processing Letters 5 (1979), 261-266