SCRIBING REPORT OF COMPILERS CLASS ON 27.08.2013 BY PURELLA VIVEK ADITYA (11CS10034)

ERROR HANDLING IN PREDICTIVE PARSER

Motivation:

- We know that the Predictive parser performs Left most derivative while parsing the given sentence
- Now the given sentence may be a valid sentence or an invalid sentence with respect to the specified grammar
- An error is detected during the predictive parsing when the terminal on top of the stack does not match the next input symbol, or when nonterminal X on top of the stack, a is the next input symbol, and parsing table entry PT[X , a] is empty i.e., X is not in the parsing table.
- Our predictive parser works as follows

```
Stack top X
input symbol a
:
while (...)     {
        if (X is Terminal)
                {
                if(X==a)
                        {
                        pop X
                        move input ptr forward to next symbol
                        }
                else                    // X!=a
                        {               // Error found                }
                }
        else                            // X is NT
                {
                if (X is found in parsing table )
                        {
                                pop X   // X---> Y1 Y2...Yn
                                push Yn... Y2 Y1
                        }
                else                    // X not in parsing table
                        {               // Error found                }
                }
        }
```

- Our code will report only the 1st error, but we want to report all errors , so we need to do error handling
- Specification of a parser
    - Report syntax errors
    - Proceed forward after detecting 1st error $\rightarrow$ Error Recovery (ER)
    - Simple and Fast

Error Recovery Methods and Techniques:

1. Panic Mode Recovery
2. Phrase level Recovery
3. Erroneous Productions

Panic Mode Recovery:

- Let us think that the parser has successfully scanned and created a parse tree till 'a' and next to that it has found an error

  $W= x_1x_2……a……x_n$

- Panic-mode error recovery is based on the idea of skipping symbols on the input until a token in a selected set of synchronizing tokens (a specific set of symbols) appears. From there continue parsing the rest of string.
- After detection of error the parser should be restored to a start state, where it can restart again.
- Its effectiveness depends on the choice of synchronizing set. The sets should be chosen so that the parser recovers quickly from errors that are likely to occur in practice.
- Good example for specific symbol in C is **;** .
- As a starting point, place all symbols in FOLLOW (X) into the synchronizing set for nonterminal X, if we skip tokens until an element of FOLLOW(X) is seen and pop X from the stack, it is likely that parsing can continue.
- We need to add some more symbols in the synchronizing set. But how does this work
- By keeping a special character 'S' in the parsing table at the places where the elements of synchronizing set are present we can achieve it

PT	$SyncSymbol_1$ …………. $SyncSymbol_2$
X	S ………… S …………………………………………….

Justification

- Suppose $S\rightarrow\alpha A\beta$ And $A\rightarrow a\Upsilon$

  $S\rightarrow \alpha a\Upsilon\beta$ ------- a valid sentential form

- a ϵ Follow (X) , if we skip from erroneous symbol to 'a' the rest is probably valid sentential form

Drawbacks

- The above discussion of panic-mode recovery does not address the important issue of error messages.
- The compiler designer must supply informative error messages that not only describe the error, they must draw attention to where the error was discovered.

Phrase level Recovery:

- On discovering an error, perform a local fix to allow the parser to continue.
- Simultaneously report error

PT	T
NT	Now invoke a specific function to modify the string

- Simple cases are exchanging ; with , and = with == , delete an extraneous semicolon, or insert a missing semicolon. Difficulties occur when the real error occurred long before an error was detected.
- The choice of the local correction is left to the compiler designer.

Erroneous Productions

- Include productions for common errors. We can augment the grammar for the language at hand with productions that generate the erroneous constructs.
- A parser constructed from a grammar augmented by these error productions detects the anticipated errors when an error production is used during parsing.
- The parser can then generate appropriate error diagnostics about the erroneous construct that has been recognized in the input.

THE END