1. Error Recovery

This is an extract from °*http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html#errors*

- A important aspect of building parsers with CUP is support for syntactic error recovery.
- CUP uses the same error recovery mechanisms as YACC. In particular, it supports a special error symbol (denoted simply as error).
- This symbol plays the role of a special non-terminal which, instead of being defined by productions, instead matches an erroneous input sequence.
- The error symbol only comes into play if a syntax error is detected.
- If a syntax error is detected then the parser tries to replace some portion of the input token stream with error and then continue parsing.

  For example, we might have productions such as:

```
stmt ::= expr SEMI
       | while_stmt SEMI
       | if_stmt SEMI
       | ...
       | error SEMI
         ;
```

  This indicates that if none of the normal productions for stmt can be matched by the input, then a syntax error should be declared, and recovery should be made by skipping erroneous tokens (equivalent to matching and replacing them with error) up to a point at which the parse can be continued with a semicolon (and additional context that legally follows a statement).

- An error is considered to be recovered from if and only if a sufficient number of tokens past the error symbol can be successfully parsed. (The number of tokens required is determined by the error_sync_size() method of the parser and defaults to 3).

- Specifically, the parser first looks for the closest state to the top of the parse stack that has an outgoing transition under error.
- This generally corresponds to working from productions that represent more detailed constructs (such as a specific kind of statement) up to productions that represent more general or enclosing constructs (such as the general production for all statements or a production representing a whole section of declarations) until we get to a place where an error recovery production has been provided for.
- Once the parser is placed into a configuration that has an immediate error recovery (by popping the stack to the first such state), the parser begins skipping tokens to find a point at which the parse can be continued.
- After discarding each token, the parser attempts to parse ahead in the input (without executing any embedded semantic actions).
- If the parser can successfully parse past the required number of tokens, then the input is backed up to the point of recovery and the parse is resumed normally (executing all actions).
- If the parse cannot be continued far enough, then another token is discarded and the parser again tries to parse ahead.
- If the end of input is reached without making a successful recovery (or there was no suitable error recovery state found on the parse stack to begin with) then error recovery fails.

1.1. Cup Error Methods

public void report_error(String message, Object info)

This method should be called whenever an error message is to be issued. In the default implementation of this method, the first parameter provides the text of a message which is printed on System.err and the second parameter is simply ignored. It is very typical to override this method in order to provide a more sophisticated error reporting mechanism.

public void report_fatal_error(String message, Object info)

This method should be called whenever a non-recoverable error occurs. It responds by calling report_error(), then aborts parsing by calling the parser method done_parsing(), and finally throws an exception. (In general done_parsing() should be called at any point that parsing needs to be terminated early).

public void syntax_error(Symbol cur_token)

This method is called by the parser as soon as a syntax error is detected (but before error recovery is attempted). In the default implementation it calls: report_error("Syntax error", null);.

public void unrecovered_syntax_error(Symbol cur_token)

This method is called by the parser if it is unable to recover from a syntax error. In the default implementation it calls: report_fatal_error("Couldn’t repair and continue parse", null);.

protected int error_sync_size()

This method is called by the parser to determine how many tokens it must successfully parse in order to consider an error recovery successful. The default implementation returns 3. Values below 2 are not recommended.

1.2. Example

- Parser.cup:

```
1       // JavaCup specification for a simple expression evaluator (w/ a
2
3       import java_cup.runtime.*;
4
5       parser code {:
6
7         protected int error_sync_size () {
8           System.out.println(":error_sync_size was called.");
9           return 1;   // not recommended by the CUP manual
```

```
10                    // but we need recovery after the next successfu
11        }
12        public void syntax_error(Symbol cur_token)    {
13          System.out.println("I'm sorry, but I have to ");
14          System.out.println("report a syntax error.");
15          System.out.println("The last symbol was: "+ Scanner.lastSymb
16                             " (See sym.java)");
17          System.out.println("at line number " + Main.s.getLineNumber(
18
19          report_error("Syntax error", null);
20        }
21
22      :}
23
24      /* Terminals (tokens returned by the scanner). */
25      terminal           SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
26      terminal           LPAREN, RPAREN;
27      terminal Integer   NUMBER;
28
29      /* Non terminals */
30      non terminal Object     expr_list, expr_part;
31      non terminal Integer    expr;
32
33      /* Precedences */
34      precedence left PLUS, MINUS;
35      precedence left TIMES, DIVIDE, MOD; precedence left LPAREN; /* T
36                    | expr_part
37                    ;
38
39      expr_part ::= expr:e
40                        {: System.out.println("= " + e); :}
41                      SEMI
42      /*
43                    | error:e
44                        {: System.out.println("error ");
45                           parser.report_error("this input is not cor
```

```
46                          System.out.println("e = " + e);
47                        :}
48                      SEMI
49      */
50                    ;
51
52      expr      ::= expr:e1 PLUS expr:e2
53                        {: RESULT = new Integer(e1.intValue() + e2.in
54                    | expr:e1 MINUS expr:e2
55                        {: RESULT = new Integer(e1.intValue() - e2.in
56                    | expr:e1 TIMES expr:e2
57                        {: RESULT = new Integer(e1.intValue() * e2.in
58                    | expr:e1 DIVIDE expr:e2
59                        {: if ( e2.intValue() != 0 )
60                             RESULT = new Integer(e1.intValue() / e2.
61                           else {
62                             System.out.println("Zero Division");
63                             RESULT = new Integer(0);
64                           }
65                        :}
66                    | expr:e1 MOD expr:e2
67                        {: if ( e2.intValue() != 0 )
68                             RESULT = new Integer(e1.intValue() % e2.
69                           else {
70                             System.out.println("Zero Mod Operation")
71                             RESULT = new Integer(0);
72                           }
73                        :}
74                    | NUMBER:n
75                        {: RESULT = n; :}
76                    | LPAREN expr:e RPAREN
77                        {: RESULT = e; :}
78
79                    | error:e
80                        {:
81                           parser.report_error("Operand missing", nul
```

```
82                              System.out.println("e = " + e);
83                              RESULT = new Integer(42);
84                            :}
85
86                      ;
```

Source Code: °*Src/11/Parser.cup*

- Execution

```
java Main
2 - ;
I’m sorry, but I have to
report a syntax error.
The last symbol was: #2 (See sym.java)
at line number 1
Syntax error
:error_sync_size was called.
:error_sync_size was called.
2 + 3;
:error_sync_size was called.
:error_sync_size was called.
Operand missing
e = null
= -40
:error_sync_size was called.
= 5
2 + 3;
= 5
```

1.3. CUP Manual: Appendix D: Bugs

In this version of CUP it’s difficult for the semantic action phrases (Java code attached to productions) to access the report_error method and other similar methods and objects defined in the parser code directive.