1 Predictive Parsing


2 Using Parse Tables

Use the parse tables in Handout # 3 to parse the following program:

\[ A = B \times X + 1; \]

Your answer should be similar to the form shown in Figure 2.29 on page 98 (page 91 of the 2nd edition).

In the grammar, we used precedence declarations to resolve some shift-reduce conflicts. Suppose we hadn’t used the precedence declarations. Then the shift-reduce conflicts would be resolved in favor of the shift rules. So in states 20 and 21, on ‘+’ the parser would shift and go to state 15, and in state 21, on ‘*’ the parser would shift and go to state 16.

Supposing this case, how would your answer to the first part of this question differ? Please give the part of the trace that differs.

What is the problem with this result?

3 Testing

In your homework2 directory, create two files: good.cool and bad.cool. The first should be syntactically correct, but not semantically correct (it will not be run).

At the very least, you should test every grammatical construct and every AST possibility. It should also test 0, 1 and multiple cases where repetition occurs (e.g., in formal parameter lists). You must also test each precedence level against neighboring precedence levels.

Your files should not use the “native” keyword. The file basic.cool has sufficient good tests of native, and since you can assume that random people aren’t editing it, we don’t need bad tests for “native.”

In the file bad.cool, it is most important to test that the parser correctly rejects and recovers from errors that the user is likely to make, such as incorrect semicolon use or having an empty repetition in a situation where a repeated entity must occur at least once.

You should also test recovery at all three levels of required error recovery: classes, features, and expressions in a block.

You can check out your test coverage using the test3 script:

```
test3 [-v] good.cool bad.cool
```

(Our solution only scores 125/126.)
1 Theoretical Results

Please read Section 2.4.3 (on the CD). Please also look at the figures that go along with this section.

(a) The textbook claims (correctly) that every LR language has an SLR(1) grammar. Give an example grammar that is LL(1) but not SLR(1). (Hint: one of the figures has such a grammar, but it is not labeled as such.) Explain why it is not. (Construct the tables and explain.)

(b) What is the significance of the textbook claim in the previous question? (For compiler writers.)

2 Tool history

Read the following papers:

- C Donnelly and R Stallman, Bison: the Yacc-Compatible Parser Generator, (various editions).

In this family, what remains the same and what changes? Please explain!

3 Alternatives

The family of tools including ANTLR and JavaCC have a very different philosophy. Compare these tools to the ‘yacc’ family; what are the advantages and disadvantages. Cite published work.