

Specifying jcup generated parsers

Exploring problems

CS 202 T

spec example

1

---

---

---

---

---

---

---

---

```
setenv CLASSPATH ~cdamon/cs202/java
```

```
java java_cup.Main < pspec.cup
```

*or*

```
~cdamon/cs202/java/bin/jcup pspec.cup
```

Generates code in parser.java

Can call parser()

Can call debug\_parser()

CS 202 T

running cup

2

---

---

---

---

---

---

---

---

2 potential problems with grammar:

Ambiguous

Wrong language

Relatively easy to fix problems (once you find them)

CS 202 T

identifying ambiguity

3

---

---

---

---

---

---

---

---

Can resolve some ambiguities with  
precedence rules

jcup offers two options for resolving many  
conflicts: *precedence* and *associativity*

CS 202 T

precedence

4

Specify precedence, associativity on  
terminals  
Appears before the grammar

Imagine arithmetic language with negative  
numbers (written

CS 202 T

precedence

5

Lexer will not generate separate unary  
MINUS versus binary MINUS tokens  
They both look the same to lexer

Parser can define extra token, never  
returned by lexer

production declared as  
`exp ::= MINUS exp %prec UMINUS;`

CS 202 T

extra precedence rules

7

---

---

---

---

---

---

---

---

Generally better to rewrite grammar than  
use precedence  
matter of style, not substance

Consider (b, OP1, OP2 terminals):

`a ::= a OP1 a;`

`a ::= a OP2 a;`

`a ::= b;`

Ambiguous grammar

CS 202 T

rewriting instead

8

---

---

---

---

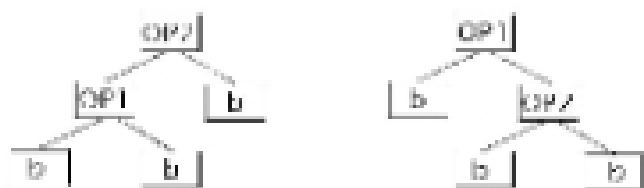
---

---

---

---

Multiple parse trees for (b OP1 b OP2 b):



CS 202 T

rewriting instead

9

---

---

---

---

---

---

---

---