

Implementation Plan

Test-driven approach

- Write test cases for
 - one AST node at a time
 - one type check at a time
 - one possible type at a time (start with atomic types)
- Set up a regression testing script
 - capture your compiler output on test case to a temp file
 - compare output to a handwritten output for test case
- Implement
 - one AST node at a time
 - one type check at a time
 - one possible type at a time (start with atomic types)

Advantages

- turn in your program at any point to get partial credit
- separate two most difficult pieces: understanding MiniJava typing and implementing the typecheck with the provided data structures

CS453 Lecture

Error Recovery

1

Total points so far (32% of the final grade)

Distribution for TotalPoints

Statistics: TotalPoints
Graded out of: 0.00 Highest grade: 3200.00 Mean grade: 2487.55 Standard deviation: 858.19
Number of records: 22 Lowest grade: 0.00 Median grade: 2577.00

Score Range	Frequency
[0, 320)	2
[320, 640)	
[640, 960)	
[960, 1280)	
[1280, 1600)	
[1600, 1920)	
[1920, 2240)	1
[2240, 2560)	6
[2560, 2880)	6
[2880, 3200)	6
[3200]	1

Fewer Bars More Bars

CS453 Lecture

Error Recovery

2

Plan for Today

PA5 status

- read emails going to mailing list
 - specification for lines and positions was fixed to match example output
 - isSameOrExtends implementation was buggy
- consider working with a partner
- implement and test one semantic error at a time

Error Recovery during

- lexical analysis
- syntax analysis
- semantic analysis

CS453 Lecture

Error Recovery

3

Implementing type checking for MiniJava (Slide 3)

Syntax

AST production AST node

new id ()

exp = {new} id

Errors

[LINENUM,POSNUM] Class CLASSNAME does not exist

Exp . id (ExpList) exp = {call} exp id [args]:exp*

[LINENUM,POSNUM] Receiver of method call must be a class type

[LINENUM,POSNUM] Method METHODNAME does not exist in class type CLASSNAME

[LINENUM,POSNUM] Method METHODNAME requires exactly NUM arguments

[LINENUM,POSNUM] Invalid argument type for method METHODNAME

CS453 Lecture

Error Recovery

4

Error Handling Goals

Provide program with a list of as many errors as possible

Provide USEFUL error messages

- appropriate line and position information
- guidance for fixing the error

Avoid infinite loops or recursion

Find "all" errors in program before translation begins

Error handling in the SableCC lexer

Input

```
3_id454 @
```

Lexer only

```
> java MainLexer t < temp.java  
minijava.node.TTIntegerLiteral 3  
[1,2] Unknown token: _
```

Parser only

```
minijava.parser.ParserException: [1,1] expecting: 'class'  
at minijava.parser.Parser.parse(Parser.java:599)  
at SemanticAnalysis.main(SemanticAnalysis.java:64)
```

Predictive parser for midterm review example

```
void S() { switch (tok) {  
  case ID:  
    case EOF:// the 2 characters in the FIRST(StmList EOF)  
      StmList(); eat(EOF); break;  
  default: print "error"; error(S); break;  
}}  
void StmList() { switch (tok) {  
  case ID: // FIRST( Stm StmList ) = { ID }  
    Stm(); StmList(); break;  
  case EOF: // FOLLOW(StmList) = { EOF }  
    break;  
  default:print "error"; error(StmList); break;  
}}  
void Stm() { switch (tok) {  
  case ID: eat(id,Stm); eat(assign,Stm); eat(float,Stm);  
    break;  
  default:print "error"; error(Stm); break;  
}}
```