Lecture 9: Error Handling

- One purpose of the parser is to filter out errors that show up in parsing
- Later stages should not have to deal with possibility of malformed constructs
- Parser must *identify* error so programmer knows what to correct
- Parser should *recover* so that processing can continue (and other errors found).
- Parser might even *correct* error (e.g., PL/C compiler could âœcorrectâ some Fortran programs into equivalent PL/1 programs!)

Identifying Errors

- All of the valid parsers we've seen identify syntax errors as soon as possible.
- *Valid prefix property:* all the input that is shifted or scanned is the beginning of some valid program...
- ... But the rest of the input might not be.
- So in principle, deleting the lookahead (and subsequent symbols) and inserting others will give a valid program.

Automating Recovery

- Unfortunately, best results require using semantic knowledge and hand tuning.
  - E.g., a(i].y = 5 might be turned to a[i].y = 5 if a is statically known to be a list, or a(i).y = 5 if a function.
- Some automatic methods can do an OK job that at least allows parser to catch more than one error.

Bison's Technique

- The special terminal symbol error is never actually returned by the lexer.
- Gets inserted by parser in place of erroneous tokens.
- Parsing then proceeds normally.

Example of Bison's Error Rules

Suppose we want to throw away bad statements and carry on

```
stmt : whileStmt
     | ifStmt
     | ...
     | error NEWLINE
     ;
```

Response to Error

- Consider erroneous text like

```
if x y: ...
```

- When parser gets to the y, will detect error.
- Then pops items off parsing stack until it finds a state that allows a shift or reduction on 'error' terminal
- Does reductions, then shifts 'error'.
- Finally, throws away input until it finds a symbol it can shift after 'error', according to the grammar.

Error Response, contd.

- So with our example:

```
stmt : whileStmt
     | ifStmt
     | ...
     | error NEWLINE
     ;
```

  We see 'y', throw away the 'if x', so as to be back to where a stmt can start.
- Shift 'error' and throw away more symbols to NEWLINE. Then carry on.

Of Course, It's Not Perfect

- "Throw away and punt" is sometimes called "panic-mode error recovery"
- Results are often annoying.
- For example, in our example, there's an INDENT after the NEWLINE, which doesn't fit the grammar and causes another error.
- Bison compensates in this case by not reporting errors that are too close together
- But in general, can get cascade of errors.
- Doing it right takes a lot of work.