Plan for Today

Predictive parser for MiniSVG (should be review)

Error recovery for predictive parsers

Predictive parsing as a specific subclass of recursive descent parsing
  – necessary to remove left-recursion (MiniSVG modification as example)
  – might have to left-factor
  – complexity comparisons with general parsing

MeggyJava language intro

FIRST and FOLLOW sets (for reference)

nullable(X)
  – X is a nonterminal
  – nullable(X) is true if X can derive the empty string

FIRST
  - FIRST(z) = \{z\}, where z is a terminal
  - FIRST(X) = union of all FIRST(rhs_i), where X is a nonterminal and X \rightarrow rhs_i
  - FIRST(rhs) = union all of FIRST(sym) on rhs up to and including first nonnullable

FOLLOW(Y), only relevant when Y is a nonterminal
  - look for Y in rhs of rules (lhs \rightarrow rhs) and union all FIRST sets for symbols
    after Y up to and including first nonnullable
  - if all symbols after Y are nullable then also union in FOLLOW(lhs)
Constructing the Predictive Parser Table

Algorithm
for each X -> gamma
    for each T in FIRST(gamma)
        table[X,T] = X->gamma
    if gamma is nullable
        for each T in FOLLOW(X)
            table[X,T] = X->gamma

(1) start -> mesh EOF
(2) mesh -> NUM nodelist NUM elemlist
(3a & b) nodelist -> ϵ | node nodelist
(4) node -> NODE NUM REAL REAL // node_id, x, y
(5a & b) elemlist -> ϵ | elem elemlist
(6a) elem -> TRI NUM NUM NUM NUM // elem_id, 3 node ids
(6b) elem -> SQR NUM NUM NUM NUM NUM //elem_id,4 node ids

Error Recovery

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
– Add minimal overhead to the processing of correct programs

Approaches
– Stop after first error
– Panic mode
– Phrase-level recovery
Predictive parser with panic mode error recovery

```c
// Float assignment grammar.
void S() { switch (m_lookahead) {
    case ID:
        case EOF:// the 2 characters in the FIRST(StmList EOF)
        try { StmList(); match(EOF); } catch { panic_S(); } break;
    default: panic_S(); break;
}}
void StmList() { switch (m_lookahead) {
    case ID: // FIRST( Stm StmList ) = { ID }
        Stm(); StmList(); break;
    case EOF: // FOLLOW(StmList) = { EOF }
        break;
    default: panic_StmList(); break;
}}
void Stm() { switch (m_lookahead) {
    case ID:  try { match(ID); match(ASSIGN); match(FLOAT);
        } catch { panic_Stm(); } break;
    default: panic_Stm(); break;
}}
```

Predictive Parser Table for Modified MiniSVG Grammar

**Algorithm**

for each X -> gamma
    for each T in FIRST(gamma)
        table[X,T] = X->gamma
    if gamma is nullable
        for each T in FOLLOW(X)
            table[X,T] = X->gamma

(1) svg -> SVG_START elem_list SVG_END EOF
(2a & b) elem_list -> elem_list elem | epsilon
(3) elem -> RECT_START ... ELEM_END
(4) elem -> CIRCLE_START ... ELEM_END
(5) elem -> LINE_START ... ELEM_END
Example Parse Tree for Modified MiniSVG (PROBLEM!)

Predictive Parsing Complexity

LL(k) grammar classes
- Left-to-right scan
- Left-most derivation
- k tokens of lookahead

Comparing complexity
- $O(N^3)$ for general case algorithms, where $N$ is the number of tokens in the stream
- $O(N)$ for predictive parsing

Requirements for LL(1), for all productions of nonterminal A
- None of the FIRST(rhs) for A production rules can overlap
- If nullable(A) then FOLLOW(A) must not overlap with FIRST(rhs) for any A->rhs