CSE302: Compiler Design

Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

March 15, 2007

Outline

- Recap
  - Top-down parsing (Section 4.4)
- Bottom-up parsing (Section 4.5)
- Summary and homework

Top-Down Parsing

- Finding a leftmost derivation for an input string
  - Recursive-descent parsing
    - Predictive parsing for LL(1) grammars
      - Non-recursive version

LL(1) Parsing: A Schematic View

- Top-down parsing
  - Leftmost derivations and left-sentential forms

Parsing stack	Input buffer	Actions
$StartSymbol	InputString$	lookahead one token, decide A-production
...	...	...
$	$	accept

LR Parsing: A Schematic View

- Bottom-up parsing
  - Rightmost derivations and right-sentential forms

Parsing stack	Input buffer	Actions
$	InputString$	lookahead zero or one token, decide S/R
...	...	...
$StartSymbol	$	accept

An Example

- Balanced parentheses
  - $S \rightarrow (S) S \mid \varepsilon$
  - Input string: ( )
  - Parsing stack Input buffer Action

    ... ...
- This process reflects the rightmost derivation but in a reverse order
  - Right-sentential forms
  - Grammars are always augmented with a new start symbol
- When to shift and when to reduce depend on the parsing states

Another Example

- Expressions of numeric additions
  - $E' \rightarrow E$
  - $E \rightarrow E$ + **n** | **n**
  - Input string: **n** + **n**
  - Parsing stack Input buffer Action

    ... ...
- This process reflects the rightmost derivation but in a reverse order
  - Right-sentential forms
- When to shift and when to reduce: parsing states
  - Shift until it is possible for reduction
  - Reduce when the strong of symbols on the top of the stack matches a production body & the reduced result is a next right-sentential form
    - **Handle** of a right-sentential form

Finite Automata of Parsing States

- Finite automata for LR(0) parsers
  - LR(0) items are used to identify the parsing states
    - $A \rightarrow \alpha$
      - $A \rightarrow .\alpha$ is an item (initial item)
        - We may about to recognize A by $A \rightarrow \alpha$
      - $A \rightarrow \alpha.$ is also an item (complete item)
        - α may be a handle for reduction
    - $A \rightarrow \beta\gamma$
      - $A \rightarrow \beta.\gamma$, $A \rightarrow .\beta\gamma$, and $A \rightarrow \beta\gamma.$ are LR(0) items
    - Examples
  - NFA construction
    - Parsing based on item1 $\xrightarrow{?}$ parsing based on item2

LR(0) Parser NFA Construction

- $A \rightarrow \alpha.X\beta \xrightarrow{X} A \rightarrow \alpha X.\beta$
  - Shift action if *X* is a terminal
- $A \rightarrow \alpha.X\beta \xrightarrow{\varepsilon} X \rightarrow .\gamma$
  - Reduction action if *X* is a non-terminal

NFA Construction Examples

Conversion of NFA to DFA (2/15)

- Subset construction algorithm
  - Input: An NFA *N*
  - Output: A DFA *D* accepting the same language as *N*
  - Algorithm: construct a transition table Dtran corresponding to D

```
Initially, ε-closure(s0) is the only state in Dstates, and it is unmarked;
while ( there is an unmarked state T in Dstates ) {
    mark T;
    for ( each input symbol a) {
      U = ε-closure(move(T,a));
      if ( U is not in Dstates ) add U as an unmarked state to Dstates;
      Dtran[T,a] = U;
    }
}
```

ε-closure(*s*) and ε-closure(T)

- ε-closure(*s*): a set of NFA states reachable from NFA state *s* on ε-transitions alone
- ε-closure(T): a set of NFA states reachable from some NFA state s in the set T on ε-transitions alone
  - $\cup_{s \text{ in } T}$ ε-closure(*s*)

```
push all states of T onto stack;
initialize ε-closure(T) to T;
while ( stack is not empty ) {
   pop t, the top element, off stack;
   for ( each state u with an edge from t to u labeled ε)
     if ( u is not in ε-closure(T) ) {
            add u to ε-closure(T);    push u onto stack;
     }
}
```

move(T,*a*)

- A set of NFA states to which there is a transition on input symbol a from some state *s* in T

Converting NFA Examples to DFA

The LR(0) Parsing Algorithm

- LR(0) parsing
  - If state s contains $A \rightarrow \alpha.X\beta$ where X is a terminal, then shift and the state changes to s' containing $A \rightarrow \alpha X.\beta$
  - If state s contains $A \rightarrow \gamma.$, then reduce by $A \rightarrow \gamma$ (stack ops) and the state changes to s' containing $B \rightarrow \lambda A.\eta$

The LR(0) Parsing Algorithm

- LR(0) parsing cannot handle a grammar that in its DFA there is a state *s*
  - *s* contains a shift item $A \to \alpha.X\beta$ and a complete item $B \to \delta.$
  - *s* contains two complete items $A \to \gamma.$ and $B \to \delta.$

Another Example

- $A \rightarrow (A) \mid a$
- LR(0) items, NFA, and DFA
- Schematic view for parsing ((a))

Outline

- Recap
- Bottom-up parsing (Section 4.5)
- Summary and homework
  - Homework posted at the Blackboard