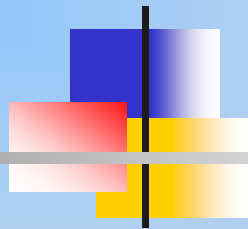
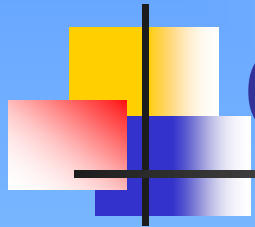


CSE302: Compiler Design



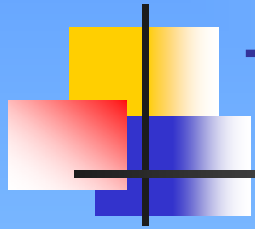
Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

March 15, 2007



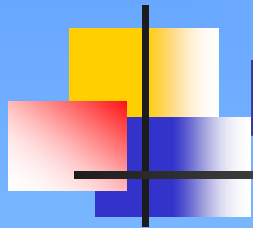
Outline

- 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

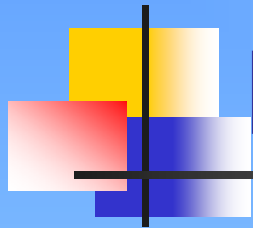
InputString\$

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

- $S \rightarrow (S) S \mid \epsilon$

- 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
 - $E' \rightarrow E$
 - $E \rightarrow E + n \mid n$
 - 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 string 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
 - $A \rightarrow \alpha$
 - $A \rightarrow \cdot \alpha$ is an item (initial item)
 - We may want to recognize A by $A \rightarrow \alpha$
 - $A \rightarrow \alpha \cdot$ is also an item (complete item)
 - α may be a handle for reduction
 - $A \rightarrow \beta\gamma$
 - $A \rightarrow \beta \cdot \gamma$, $A \rightarrow \cdot \beta\gamma$, and $A \rightarrow \beta\gamma \cdot$ are LR(0) items
 - **Examples**
 - NFA construction
 - Parsing based on item1 $\xrightarrow{?}$ parsing based on item2



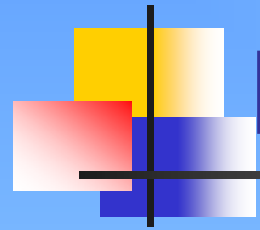
LR(0) Parser NFA Construction

- $A \rightarrow \alpha.X\beta \xrightarrow{X} A \rightarrow \alpha X.\beta$

- Shift action if X is a terminal

- $A \rightarrow \alpha.X\beta \xrightarrow{\epsilon} X \rightarrow .\gamma$

- 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 D_{tran} corresponding to D

Initially, $\epsilon\text{-closure}(s_0)$ is the only state in D_{states} , and it is unmarked;
while (there is an unmarked state T in D_{states}) {
 mark T ;
 for (each input symbol a) {
 $U = \epsilon\text{-closure}(\text{move}(T, a))$;
 if (U is not in D_{states}) add U as an unmarked state to D_{states} ;
 $D_{\text{tran}}[T, a] = U$;
 }
}



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
 - $\cup_{s \in T} \epsilon$ -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;

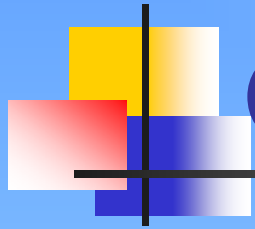
 }

 }



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 $B \rightarrow \lambda A.\eta$



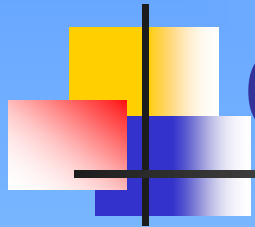
The LR(0) Parsing Algorithm

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



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