# Bottom-Up Syntax Analysis

Mooly Sagiv
Tel-Aviv University
Schriber 317
Phone (03)640-7606
sagiv@math.tau.ac.il

November 19, 2000

"http://www.math.tau.ac.il/~ sagiv/courses/wcc.html"

## 1 Bottom-Up Syntax Analysis

**input**

- A context free grammar
- A stream of tokens

**output** A syntax tree or error

**method**

- Construct the syntax tree in a bottom-up manner
- Find the right most derivation (in reversed order)
- Decide which production to "reduce" scanning one token forward
- Report error as soon as the input is not a prefix of a valid program
- After an error ignore tokens until a"synchronization token" occurs

## 2 Parser Actions

**shift**

1. Push current input symbol on the stack
2. Advance input symbol

**reduce** $A \to \alpha$

1. Pop $|\alpha|$ symbols from the stack

2. Create abstract syntax tree node
3. Push $A$ on the stack

**Accept** Parsing is complete

**Error** Report an error

# 3 Parsing $aabb$ in the grammar $S \to aSb|\epsilon$

| Stack | Input | Action |
|---|---|---|
| $ | $aabb$$ | **shift** |
| $$a$ | $abb$$ | **shift** |
| $$aa$ | $bb$$ | **reduce** $S \to \epsilon$ |
| $$aaS$ | $bb$$ | **shift** |
| $$aaSb$ | $b$$ | **reduce** $S \to aSb$ |
| $$aS$ | $b$$ | **shift** |
| $$aSb$ | $ | **reduce** $S \to aSb$ |
| $$S$ | $ | **accept** |

Issues:

- Shift vs. Reduce
- Reduce $A \to \alpha$, Reduce $B \to \alpha\beta$

# 4 Parsing $aa$ in the grammar $S \to AB, S \to A, A \to a, B \to a$

| Stack | Input | Action |
|---|---|---|
| $ | $aa$$ | **shift** |
| $$a$ | $a$$ | **reduce**$A \to a$ |
| $$A$ | $a$$ | **shift** |
| $$Aa$ | $ | **reduce**$B \to a$ |
| $$AB$ | $ | **reduce**$S \to AB$ |
| $$S$ | $ | **accept** |

Issues:

- Shift vs. Reduce
- Reduce $A \to \alpha$, Reduce $B \to \alpha\beta$

## 5 Modified Parser Actions

**shift** $i$

1. Push current input symbol on the stack
2. Push $i$ on the stack
3. Advance input symbol

**reduce** $A \to \alpha$

1. Pop 2 $|\alpha|$ symbols from the stack
2. Create abstract syntax tree node
3. Push $A$ on the stack

**Accept** Parsing is complete

**Error** Report an error

## 6 A Parser table for $S \to aSb|\epsilon$

| **state** | **state-description** | **tokens** | | | **goto** |
|---|---|---|---|---|---|
| | | $a$ | $b$ | $ | $S$ |
| 0 | initial | $s1$ | | $rS \to \epsilon$ | 4 |
| 1 | parsed $a$ | $s1$ | $rS \to \epsilon$ | | 2 |
| 2 | after $S$ | | $s3$ | | |
| 3 | parsed $aSb$ | | $rS \to aSb$ | $rS \to aSb$ | |
| 4 | final | | | $a$ | |

Parsing $aabb$

| **Stack** | **Input** | **Action** |
|---|---|---|
| $ 0 | $aabb$$ | **shift** 1 |
| $ 0 $a$ 1 | $abb$$ | **shift** 1 |
| $ 0 $a$ 1 $a$ 1 | $bb$$ | **reduce** $S \to \epsilon$ |
| $ 0 $a$ 1 $a$ 1 $S$ 2 | $bb$$ | **shift 3** |
| $ 0 $a$ 1 $a$ 1 $S$ 2 $b$ 3 | $b$$ | **reduce** $S \to aSb$ |
| $ 0 $a$ 1 $S$ 2 | $b$$ | **shift 3** |
| $ 0 $a$ 1 $S$ 2 $b$3 | $ | **reduce** $S \to aSb$ |
| $ 0 $S$ 4 | $ | **accept** |

## 7 A Parser table for $S \to AB, S \to A, A \to a, B \to a$

| s | state-description | tokens | | | goto | | |
|---|---|---|---|---|---|---|---|
| | | $a$ | $b$ | $\$$ | $A$ | $B$ | $S$ |
| 0 | initial | $s1$ | | | 2 | | 5 |
| 1 | parsed $a$ from $A$ | $rA \to a$ | | $rA \to a$ | | | |
| 2 | parsed $A$ | $s3$ | | $rS \to A$ | | 4 | |
| 3 | parsed $a$ from $B$ | | | $rB \to a$ | | | |
| 4 | parsed $AB$ | | | $rS \to AB$ | | | |
| 5 | final | | | $a$ | | | |

Parsing $aa$

| Stack | Input | Action |
|---|---|---|
| $\$\,0$ | $aa\$$ | **shift** 1 |
| $\$\,0\,a\,1$ | $a\$$ | **reduce** $A \to a$ |
| $\$\,0A\,2$ | $a\$$ | **shift 3** |
| $\$\,0A\,2\,a\,3$ | $\$$ | **reduce** $B \to a$ |
| $\$\,0A\,2\,B\,4$ | $\$$ | **reduce** $S \to AB$ |
| $\$\,0\,S\,5$ | $\$$ | **accept** |

## 8 $LR(1)$ Grammars

- Add rule $S' \to S\$$ where $S$ is the start non-terminal
- Scan the input from left to right
- Yield the right most derivations in the reverse order
- Use one token to decide:
  - To shift or to reduce
  - According to which rule to reduce
- Subclasses of $LR(1)$

  **SLR(1)** Simple LR

  **General** May require a lot of states

  $LALR(1)$ Intermediate in expressive power (Yacc, Bison, CUP)

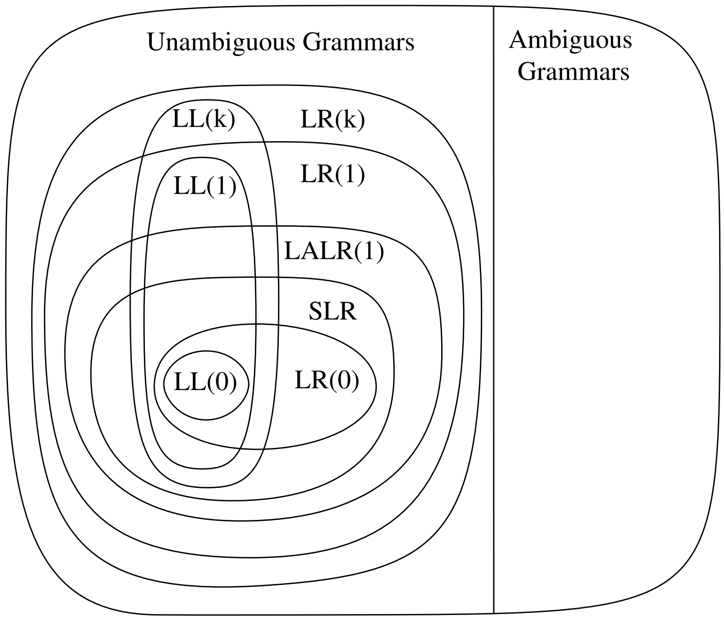


**FIGURE 3.30.** A hierarchy of grammar classes.
From *Modern Compiler Implementation in ML,*
Cambridge University Press, ©1998 Andrew W. Appel

# 9 Creating the states in Parser Generation

- A state remembers the possible prefixes of right hand sides "handles"
- There is a regular language of valid prefixes over terminals and nonterminals
- The state transition under terminal — **shift**
- The state transition under non-terminal — **goto**

# 10 SLR(1) Parser Generator

**item** $A \to \alpha.\beta$ –

- "In attempt to recognize $A$, $\alpha$ has already been recognized"
- "$\alpha$ is at the top of the stack and we are expecting $\beta$"

**SLR(1)-State** Set of items

## 11 Closure

**Closure**(I)
    **repeat**
        **for** any item $A \to \alpha.X\beta$ in $I$
            **for** any production $X \to \gamma$
                add $X \to .\gamma$ to $I$
    **until** no more changes
    **return** (I)

## 12 Goto

**Goto**(I, X)
    set $J := \phi$
    **for** any item $A \to \alpha.X\beta$ in $I$
        add $A \to \alpha X.\beta$ to $J$
    **return** closure(J)

## 13 SLR(1) States

- Initial state — $I = Closure(\{S' \to .S\$\})$
- Repeat adding states $J = Goto(I, X)$ for every state $I$ and grammar symbol $X$

## 14 Generating the SLR(1) Parsing Table

- For every token $t$, if $Goto(I, t) = J$, set the entry of $s_I$ under $T$ to "shift $s_J$"
- For every nonterminal $X$, if $Goto(I, X) = J$, set the entry of $s_I$ under $X$ to "goto $s_J$"
- For every item $A \to \alpha.$ in $I$ and for every $t \in \mathbf{FOLLOW}(A)$, set the entry of $s_I$ under $t$ to "reduce $A \to \alpha$"
- If $S \to S.\$$ in $I$, set the entry of $s_I$ under \$ to "accept"

## 15 Example

- Grammar $S \to aSb, S \to \epsilon$
- States
  - $Closure(\{S' \to .S\$\}) = \{S' \to .S\$, S \to .aSb, S \to .\} = 0$

- $Goto(0, a) = Closure(\{S \rightarrow a.Sb\}) = \{S \rightarrow a.Sb, S \rightarrow ., S \rightarrow .aSb\} = 1$
- $Goto(0, b) = Closure(\emptyset)$
- $Goto(0, S) = Closure(\{S' \rightarrow S.\$\}) = \{S' \rightarrow S.\$\} = 4$
- $Goto(1, a) = Closure(\{S \rightarrow a.Sb\}) = \{S \rightarrow a.Sb, S \rightarrow ., S \rightarrow .aSb\} = 1$
- $Goto(1, b) = Closure(\emptyset)$
- $Goto(1, S) = Closure(\{S \rightarrow aS.b\}) = \{S \rightarrow aS.b\} = 2$
- $Goto(2, a) = Closure(\emptyset)$
- $Goto(2, b) = Closure(\{S \rightarrow aSb.\}) = \{S \rightarrow aSb.\} = 3$
- $Goto(2, S) = Closure(\emptyset)$
- $Goto(3, a) = Closure(\emptyset)$
- $Goto(3, b) = Closure(\emptyset)$
- $Goto(3, S) = Closure(\emptyset)$
- $Goto(4, a) = Closure(\emptyset)$
- $Goto(4, b) = Closure(\emptyset)$
- $Goto(4, S) = Closure(\emptyset)$

- **FOLLOW**$(S) = \{b, \$\}$
- Parser Table

| **st** | **state-description** | **tokens** | | | **goto** |
|---|---|---|---|---|---|
| | | $a$ | $b$ | $\$$ | $S$ |
| 0 | $\{S' \rightarrow .S\$, S \rightarrow .aSb, S \rightarrow .\}$ | $s1$ | | $rS \rightarrow \epsilon$ | 4 |
| 1 | $\{S \rightarrow a.Sb, S \rightarrow ., S \rightarrow .aSb\}$ | $s1$ | $rS \rightarrow \epsilon$ | $rS \rightarrow \epsilon$ | 2 |
| 2 | $\{S \rightarrow aS.b\}$ | | $s3$ | | |
| 3 | $\{S \rightarrow aSb.\}$ | | $rS \rightarrow aSb$ | $rS \rightarrow aSb$ | |
| 4 | $\{S' \rightarrow S.\$\}$ | | | $a$ | |

# 16 Ambiguous Arithmetic Example

$$\begin{array}{ll} 1 & E \rightarrow E - E \\ 2 & E \rightarrow E * E \\ 3 & E \rightarrow -E \\ 4 & E \rightarrow id \\ 5 & E \rightarrow (E) \end{array}$$

## 17 An Interesting Non SLR(1) Grammar

**Grammar**

$$\begin{array}{llll} 1 & S' \to S\$ & & \\ 2 & S \to L = R & 3 & S \to R \\ 4 & L \to *R & 5 & L \to id \\ 6 & R \to L & & \end{array}$$

**SLR(1) states**

$$\begin{array}{lcl} S_0 & = & \left\{ \begin{array}{l} S' \to .S\$, \\ S \to .L = R, \\ S \to .R, \\ L \to .*R, \\ L \to .id, \\ R \to .L \end{array} \right\} \\ S_1 & = & \left\{ \begin{array}{l} S \to L. = R, \\ R \to L. \end{array} \right\} \\ S_2 & = & \left\{ \begin{array}{l} S \to L = .R, \\ R \to .L, \\ L \to .*R, \\ L \to .id \end{array} \right\} \end{array}$$

**FOLLOW(R)** $\{\$, =\}$

## 18 LR(1) Parser Generator

**item** $(A \to \alpha.\beta, t)$ –

- "$\alpha$ is at the top of the stack and we are expecting $\beta t$"

**LR(1)-State** Set of items

## 19 LALR(1) Parser Generator

Merge items with the same look-ahead

## 20 Interesting Non $LR(1)$ Grammars

- Ambiguous:
  - Ambiguous arithmetic expressions
  - Dangling-else

- Common "derived" prefix

$$
\begin{array}{rcl}
A & \rightarrow & B_1 ab \\
A & \rightarrow & B_2 ac \\
B_1 & \rightarrow & \epsilon \\
B_2 & \rightarrow & \epsilon
\end{array}
$$

- Optional non-terminals

$$
\begin{array}{rcl}
St & \rightarrow & OptLab\, St' \\
OptLab & \rightarrow & id : \\
OPtlab & \rightarrow & \epsilon \\
St' & \rightarrow & id := Exp
\end{array}
$$