COMPILER DESIGN

Bottom up parser

Session Objectives

- Learn the concepts of Bottom up parsing
- Learn the concepts of Shift reduce parsing

Session Outcomes

- At the end of this session, participants will be able to
  - Understand bottom up parser
  - Understand shift reduce parser

Agenda

- Bottom up parser
- Shift reduce parser
- Handle
- Handle pruning
- Conflicts during shift reduce parsing

Bottom-Up Parsing

- A **bottom-up parser** creates the parse tree of the given input starting from leaves towards the root.
- A bottom-up parser tries to find the right-most derivation of the given input in the reverse order.

  $S \Rightarrow ... \Rightarrow \omega$ (the right-most derivation of ω)

  $\leftarrow$ (the bottom-up parser finds the right-most derivation in the reverse order)
- Bottom-up parsing is also known as **shift-reduce parsing** because its two main actions are shift and reduce.
  - At each shift action, the current symbol in the input string is pushed to a stack.
  - At each reduction step, the symbols at the top of the stack (this symbol sequence is the right side of a production) will replaced by the non-terminal at the left side of that production.
  - There are also two more actions: accept and error.

Bottom-Up Parsing

- Two Types:
  - Shift-reduce parsing
  - Operator-precedence parsing
- Efficient Method

  →LR methods (Left-to-right, Rightmost derivation in Reverse)
  - SLR, Canonical LR, LALR

Shift-Reduce Parsing

- A shift-reduce parser tries to reduce the given input string into the starting symbol.

  a string ➔ the starting symbol

  reduced to

- At each reduction step, a substring of the input matching to the right side of a production rule is replaced by the non-terminal at the left side of that production rule.
- If the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

Rightmost Derivation: $S \overset{*}{\underset{rm}{\Rightarrow}} \omega$

Shift-Reduce Parser finds: $\omega \underset{rm}{\Leftarrow} \ldots \underset{rm}{\Leftarrow} S$

Shift-Reduce Parsing

Grammar:

$S \rightarrow \mathbf{a}\; A\; B\; \mathbf{e}$

$A \rightarrow A\; \mathbf{b}\; \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

These match production's right-hand sides

Reducing a sentence:

$\mathbf{a}\; \underline{\mathbf{b}}\; \mathbf{b}\; \mathbf{c}\; \mathbf{d}\; \mathbf{e}$

$\mathbf{a}\; \underline{A\; \mathbf{b}\; \mathbf{c}}\; \mathbf{d}\; \mathbf{e}$

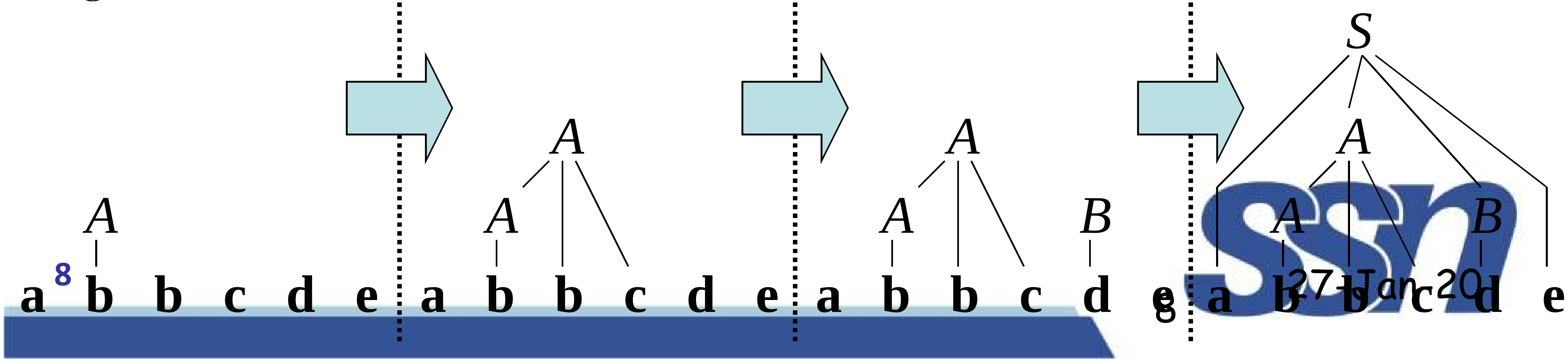
$\mathbf{a}\; A\; \underline{\mathbf{d}}\; \mathbf{e}$

$\underline{\mathbf{a}\; A\; B\; \mathbf{e}}$

S

Shift-reduce corresponds to a rightmost derivation:

$$
\begin{aligned}
S &\Rightarrow_{rm} \mathbf{a}\; A\; B\; \mathbf{e} \\
&\Rightarrow_{rm} \mathbf{a}\; A\; \mathbf{d}\; \mathbf{e} \\
&\Rightarrow_{rm} \mathbf{a}\; A\; \mathbf{b}\; \mathbf{c}\; \mathbf{d}\; \mathbf{e} \\
&\Rightarrow_{rm} \mathbf{a}\; \mathbf{b}\; \mathbf{b}\; \mathbf{c}\; \mathbf{d}\; \mathbf{e}
\end{aligned}
$$



Handle

- Informally, a **handle** of a string is a substring that matches the right side of a production rule.
  - But not every substring matches the right side of a production rule is handle
- A **handle** of a right sentential form γ ($\equiv \alpha\beta\omega$) is
  a production rule $A \to \beta$ and a position of γ
  where the string β may be found and replaced by A to produce
  the previous right-sentential form in a rightmost derivation of γ.

$$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A \omega \underset{rm}{\Rightarrow} \alpha\beta\omega$$

- If the grammar is unambiguous, then every right-sentential form of the grammar has exactly one handle.
- We will see that ω is a string of terminals.

Handle Pruning

- A right-most derivation in reverse can be obtained by **handle-pruning**.

$$S=\gamma_0 \underset{rm}{\Rightarrow} \gamma_1 \underset{rm}{\Rightarrow} \gamma_2 \underset{rm}{\Rightarrow} \cdots \underset{rm}{\Rightarrow} \gamma_{n-1} \underset{rm}{\Rightarrow} \gamma_n = \omega$$

input string

- Start from γ_n, find a handle $A_n \rightarrow \beta_n$ in γ_n, and replace β_n in by A_n to get γ_{n-1}.
- Then find a handle $A_{n-1} \rightarrow \beta_{n-1}$ in γ_{n-1}, and replace β_{n-1} in by A_{n-1} to get γ_{n-2}.
- Repeat this, until we reach S.

Handle Example

Grammar:

$S \rightarrow \mathbf{a}\ A\ B\ \mathbf{e}$

$A \rightarrow A\ \mathbf{b}\ \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ \underline{A\ \mathbf{b}\ \mathbf{c}}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{d}}\ \mathbf{e}$

$\underline{\mathbf{a}\ A\ B\ \mathbf{e}}$

S

Handle

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{b}}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ A\ \mathbf{e}$

… ?

NOT a handle, because further reductions will fail (result is not a sentential form)

A Shift-Reduce Parser

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id$

Right-Most Derivation of id+id*id

$E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow E+T*id \Rightarrow E+F*id$
$\Rightarrow E+id*id \Rightarrow T+id*id \Rightarrow F+id*id \Rightarrow id+id*id$

Right-Most Sentential Form	Reducing Production
$\underline{id}+id*id$	$F \rightarrow id$
$\underline{F}+id*id$	$T \rightarrow F$
$\underline{T}+id*id$	$E \rightarrow T$
$E+\underline{id}*id$	$F \rightarrow id$
$E+\underline{F}*id$	$T \rightarrow F$
$E+T*\underline{id}$	$F \rightarrow id$
$E+\underline{T*F}$	$T \rightarrow T*F$
$\underline{E+T}$	$E \rightarrow E+T$
E	

Handles are red and underlined in the right-sentential forms.

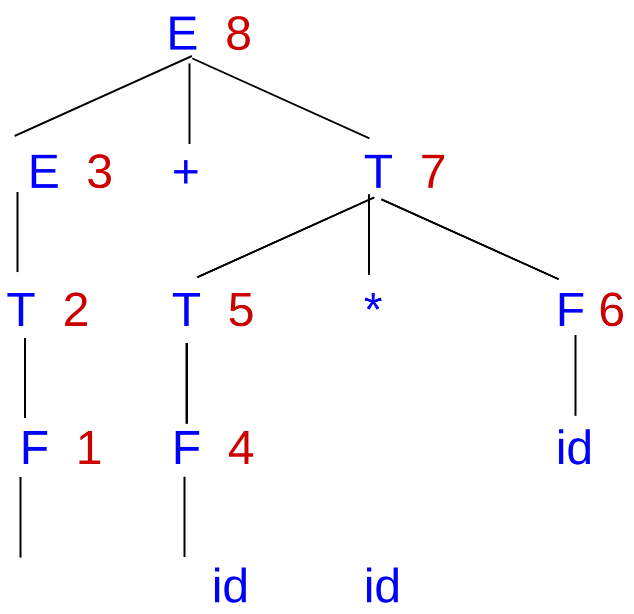
A Stack Implementation - Shift-Reduce Parser

- There are four possible actions of a shift-parser action:
  1. **Shift** : The next input symbol is shifted onto the top of the stack.
  2. **Reduce**: Replace the handle on the top of the stack by the non-terminal.
  3. **Accept**: Successful completion of parsing.
  4. **Error**: Parser discovers a syntax error, and calls an error recovery routine.
- Initial stack just contains only the end-marker $.
- The end of the input string is marked by the end-marker $.

A Stack Implementation - Shift-Reduce Parser

Stack	**Input**	**Action**
$	id+id*id$	shift
$id	+id*id$	reduce by F → id
$F	+id*id$	reduce by T → F
$T	+id*id$	reduce by E → T
$E	+id*id$	shift
$E+	id*id$	shift
$E+id	*id$	reduce by F → id
$E+F	*id$	reduce by T → F
$E+T	*id$	shift
$E+T*	id$	shift
$E+T*id	$	reduce by F → id
$E+T*F	$	reduce by T → T*F
$E+T	$	reduce by E → E+T
$E	$	accept

Parse Tree



Conflicts During Shift-Reduce Parsing

- There are context-free grammars for which shift-reduce parsers cannot be used.
- Stack contents and the next input symbol may not decide action:
  - **shift/reduce conflict**: Whether make a shift operation or a reduction.
  - **reduce/reduce conflict**: The parser cannot decide which of several reductions to make.

Shift-Reduce Conflicts

Ambiguous grammar:

$S \rightarrow$ **if** E **then** S

$\quad\mid$ **if** E **then** S **else** S

$\quad\mid$ **other**

Stack	Input	Action
\$…	…**\$**	…
\$…**if** E **then** S	**else**…**\$**	shift or reduce?

Resolve in favor of shift, so **else** matches closest **if**

Reduce-Reduce Conflicts

Grammar:

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$

Stack	Input	Action
$	**aa$**	shift
$<u>a</u>	**a$**	reduce $A \rightarrow \mathbf{a}$ <u>or</u> $B \rightarrow \mathbf{a}$?

Resolve in favor
of reduce $A \rightarrow \mathbf{a}$,
otherwise we're stuck!

Summary

- Bottom up parser
- Shift reduce parsing
- Handle
- Handle pruning

Check your understanding?

1. What is handle pruning?
2. List the four actions in shift reduce parser.
3. List the conflicts during shift reduce parsing.