Compilers

Handles
Bottom-up parsing uses two actions:

**Shift**

\[ ABC | xyz \Rightarrow ABCx | yz \]

**Reduce**

\[ Cbxy | ijk \Rightarrow CbA | ijk \]

\[ A \Rightarrow xy \]
• Left string can be implemented by a stack
  – Top of the stack is the |

• Shift pushes a terminal on the stack

• Reduce
  – pops 0 or more symbols off of the stack
    • production rhs
  – pushes a non-terminal on the stack
    • production lhs
• How do we decide when to shift or reduce?

• Example grammar:
  
  \[
  E \rightarrow T + E \mid T \\
  T \rightarrow \text{int} \ast T \mid \text{int} \mid (E)
  \]

• Consider step \( \text{int} \mid \ast \text{int} + \text{int} \)
  
  – We could reduce by \( T \rightarrow \text{int} \) giving \( T \mid \ast \text{int} + \text{int} \)
  
  – A fatal mistake!
    
    • No way to reduce to the start symbol \( E \)
• Intuition: Want to reduce only if the result can still be reduced to the start symbol

• Assume a rightmost derivation

\[ S \rightarrow^* \alpha X \omega \rightarrow \alpha \beta \omega \]

• Then \( \alpha \beta \) is a handle of \( \alpha \beta \omega \)
• Handles formalize the intuition
  – A handle is a reduction that also allows further reductions back to the start symbol

• We only want to reduce at handles

• Note: We have said what a handle is, not how to find handles
Given the grammar at right, identify the handle for the following shift-reduce parse state: \( E' + -id | + -(id + id) \)

- \( E' + -id \)
- \( id \)
- \( -id \)
- \( E' + -E' \)
Important Fact #2 about bottom-up parsing:

In shift-reduce parsing, handles appear only at the top of the stack, never inside.
• Informal induction on # of reduce moves:
  • True initially, stack is empty
  • Immediately after reducing a handle
    – right-most non-terminal on top of the stack
    – next handle must be to right of right-most non-terminal, because this is a right-most derivation
    – Sequence of shift moves reaches next handle
• In shift-reduce parsing, handles always appear at the top of the stack

• Handles are never to the left of the rightmost non-terminal
  – Therefore, shift-reduce moves are sufficient; the need never move left

• Bottom-up parsing algorithms are based on recognizing handles