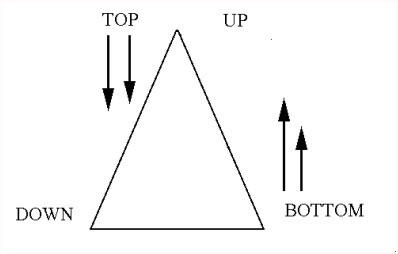
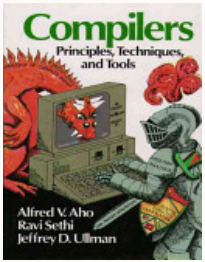
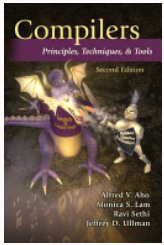
Bottom-Up Parsing

Brian Malloy
Clemson University
School of Computing



1. Resources


Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman

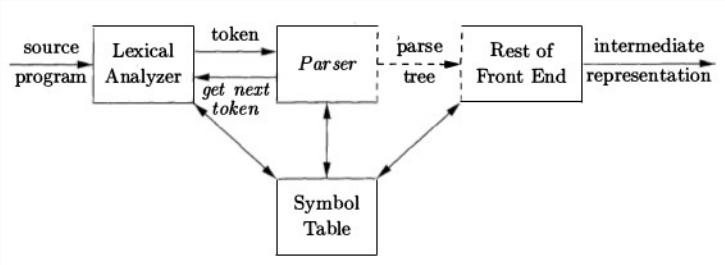

Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman


Unix Text Processing Tools
flex & bison
O'REILLY
John Levine


Bison
The Yacc-compatible Parser Generator
by Charles Donnelly and Richard Stallman

2. Introduction

- The goal of parsing: recognize sentences.

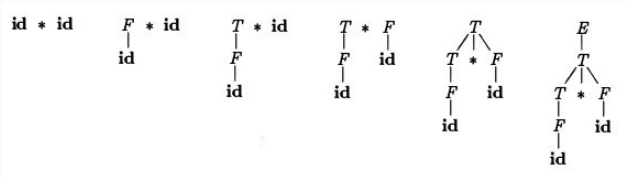


2.1. Bottom-Up Parsing

- A bottom-up parser builds a parse tree by starting at the leaves and works to the root:

  > Left to right scan of input, right-most derivation in reverse. (p. 235, dragon)

- The parse tree is likely implicit.
- Consider a parse of id*id (p. 234 dragon):



3. Shift-Reduce Parsing

- A stack holds grammar symbols
- Input buffer holds rest of string to be parsed
- $ marks bottom of stack & end of input
- handle always appears on top of stack
- rightmost derivation: try to reduce a sub-string (handle) back to a non-terminal.

3.1. What's a handle?

- substring that matches the rhs of a production

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

Right Sentential Form	Handle	Reducing Production
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \rightarrow \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \rightarrow F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \rightarrow \mathbf{id}$
$T * F$	$T * F$	$E \rightarrow T * F$

3.2. Shift-Reduce: How it works

- Initially, stack is empty, w is entire input

STACK	INPUT
$	w $

- During left to right scan of input, parser:
  - shifts zero or more input symbols onto stack,
  - until it can reduce a string β to appropriate non-terminal
  - Repeat until: error or only start symbol on stack

3.3. Example of s/r Parsing

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

STACK	INPUT	ACTION
\$	$\mathbf{id}_1 * \mathbf{id}_2$ \$	shift
\$ $\mathbf{id}_1$	$* \mathbf{id}_2$ \$	reduce by $F \rightarrow \mathbf{id}$
\$ F	$* \mathbf{id}_2$ \$	reduce by $T \rightarrow F$
\$ T	$* \mathbf{id}_2$ \$	shift
\$ $T *$	$\mathbf{id}_2$ \$	shift
\$ $T * \mathbf{id}_2$	\$	reduce by $F \rightarrow \mathbf{id}$
\$ $T * F$	\$	reduce by $T \rightarrow T * F$
\$ T	\$	reduce by $E \rightarrow T$
\$ E	\$	accept

3.4. Shift-Reduce Operations

- shift: shift next input symbol on top of stack; handle is always on top of stack, never inside → stack
- reduce: locate left end of string on stack, replace with non-terminal rhs
- accept: successful completion of parse
- error: Discover a syntax error; call error recovery routine

4. Conflicts

- There are cfgs for which s/r parsing cannot be used
- shift/reduce conflict: The parser can reach a configuration in the grammar where it cannot decide to shift or reduce
- reduce/reduce conflict: The parser can reach a configuration in the grammar where it cannot decide which of several reductions
- An ambiguous grammar is not an LR grammar.