Class 8: Parsing: Top-down and Bottom-up

SI 413 - Programming Languages and Implementation

Dr. Daniel S. Roche

United States Naval Academy

Fall 2011

Structure of a Scanner

How does a scanner generation tool like `flex` actually work?

Structure of a Scanner

How does a scanner generation tool like `flex` actually work?

1. An NDFA is generated from each regular expression.
   Final states are marked according to which rule is used.
2. These NDFAs are combined into a single NDFA.
3. The big NDFA is converted into a DFA. *How are final states marked?*
4. The final DFA is minimized for efficiency.
   The DFA is usually represented in code with a *state-character array*.

Look-ahead in scanners

The "maximal munch" rule says to always return the longest possible token.

But how can the DFA tell if it has the maximal munch?

Look-ahead in scanners

The "maximal munch" rule says to always return the longest possible token.

But how can the DFA tell if it has the maximal munch?

Usually, just stop at a transition from accepting to non-accepting state. This requires one character of *look-ahead*.

Is this good enough for any set of tokens?

Parsing

Parsing is the second part of syntax analysis.

We use grammars to specify *how tokens can combine*.
A parser uses the grammar to construct a parse tree
with tokens at the leaves.

Scanner: Specified with regular expressions, generates a DFA
Parser: Specified with context-free grammar, generates a ...

Parsing

Parsing is the second part of syntax analysis.

We use grammars to specify *how tokens can combine*.
A parser uses the grammar to construct a parse tree
with tokens at the leaves.

Scanner: Specified with regular expressions, generates a DFA
Parser: Specified with context-free grammar, generates a PDA

Generalize or Specialize?

Parsing a CFG *deterministically* is hard:
requires lots of computing time and space.

By (somewhat) restricting the class of CFGs, we can parse much faster.

For a program consisting of n tokens, we want $O(n)$ time,
using a single stack, and not too much look-ahead.

Parsing Strategies

Top-Down Parsing:

- Constructs parse tree starting at the root
- "Follow the arrows" — carry production rules forward
- Requires *predicting* which rule to apply for a given nonterminal.
- LL: Left-to-right, Leftmost derivation

Parsing Strategies

Top-Down Parsing:

- Constructs parse tree starting at the root
- "Follow the arrows" — carry production rules forward
- Requires *predicting* which rule to apply for a given nonterminal.
- LL: Left-to-right, Leftmost derivation

Bottom-Up Parsing:

- Constructs parse tree starting at the leaves
- "Go against the flow" — apply reduction rules *backwards*
- Requires
- LR: Left-to-right, Rightmost defivation

Parsing example

Simple grammar

$S \rightarrow T\ T$
$T \rightarrow \texttt{aa}$
$T \rightarrow \texttt{bb}$

Parse the string aabb, top-down and bottom-up.

Handling Errors

How do scanning errors occur?
How can we handle them?

How do parsing errors occur?
How can we handle them?

"Real" scanners/parsers also tag *everything* with filename & line number to give programmers extra help.