CS443: Compiler Construction

Lecture 1: Lexing and Parsing
Compilers translate code in phases

```
Source Code
```

```
Lexical Analyzer
```

```
Tokens
```

```
Parser
```

```
Abstract Syntax
```

```
Analysis
```

```
```

```
Lowering
```

```
Code Gen.
```

```
Target Code
```

```
“Front End”
```

```
“Back End”
```

```
a = b + c - 1
```

```
VAR a
EQUAL
VAR b
OP +
VAR C
OP -
CONST 1
```

```
Assign
```

```
temp = c - 1
a = b + temp
```

```
subl %rax, 1
addl %rax, %rbx
```

```
“Front End”
```

```
“Back End”
```

```
Analysis
```

```
Optimization
```

```
Code Gen.
```

```
Target Code
```

```
Assign
```

```
temp = c - 1
a = b + temp
```

```
subl %rax, 1
addl %rax, %rbx
```
Terminology

• *Lexical analysis* “lexing”
• Performed by *lexical analyzer* “lexer”
• Produces stream of *tokens*
Tokens are specified using a \textit{regular} grammar

- Regular expressions $R$:
  - $\varepsilon$ Empty string
  - $abc$ Exactly the string $abc$ \textit{Literal}
  - $R_1R_2$ $R_1$ followed by $R_2$ \textit{Concatenation}
  - $R_1 \mid R_2$ $R_1$ or $R_2$ \textit{Alternation}
  - $R^*$ Zero or more $R$ \textit{Kleene Star}

- $R^+$ One or more $R$
- $R?$ Optional $R$
- $[a-z]$ a, b, c, d, ..., z
Tokens are specified using a regular grammar

digit ::= [0-9]
alpha ::= [a-z]
ident ::= alpha (alpha | digit)*
um ::= digit+

ident → IDENT s
num → NUM s
“while” -> WHILE
“+” -> PLUS...
Lexing examples

• while (i < 5)  
  WHILE; LPAREN; IDENT "i"; LT; NUM 5; RPAREN
• while i < 5)  
  WHILE; IDENT "i"; LT; NUM 5; RPAREN
• whole (i < 5)  
  IDENT "whole"; LPAREN; IDENT i; LT; NUM 5; RPAREN

Might be syntax errors during parsing. Not errors during lexing.
Regex matching can be done by finite state machines (FSMs)

- Deterministic Finite Automaton (DFA)
- *States + Transition function + Start state + Set of Accepting states*

![Diagram of a Deterministic Finite Automaton](image)
Can convert regexes to DFAs

• Full algorithm in Appel, PDB. General idea:

    NFA “widgets” for each RE construction. Connect w/ transitions

    NFA (Nondeterministic Finite Automaton)
    Allow multiple transitions on each symbol, empty transitions

    Use one DFA state to represent sets of NFA states we might be in
• Draw DFA example
BNF grammars are “context-free”

\[ e ::= x \mid \text{num} \mid e \text{ bop } e \mid uop e \]
Derivation: Expand one nonterminal at a time using a production

\[ e ::= n \mid e + e \mid (e) \]

Input: \(1 + (2 + 3)\)

Leftmost Derivation

<table>
<thead>
<tr>
<th>Leftmost Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
</tr>
<tr>
<td>e + e</td>
</tr>
<tr>
<td>1 + e</td>
</tr>
<tr>
<td>1 + (e)</td>
</tr>
<tr>
<td>1 + (e + e)</td>
</tr>
<tr>
<td>1 + (2 + e)</td>
</tr>
<tr>
<td>1 + (2 + 3)</td>
</tr>
</tbody>
</table>

Rightmost Derivation

<table>
<thead>
<tr>
<th>Rightmost Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
</tr>
<tr>
<td>e + e</td>
</tr>
<tr>
<td>e + (e)</td>
</tr>
<tr>
<td>e + (e + e)</td>
</tr>
<tr>
<td>e + (e + 3)</td>
</tr>
<tr>
<td>e + (2 + 3)</td>
</tr>
<tr>
<td>1 + (2 + 3)</td>
</tr>
</tbody>
</table>
Parsing: Produce a *parse tree* from a stream of tokens

```
nonterminal N
  terminal
  nonterminal M
  terminal

production for N

terminal
terminal

production for M

sentence
```
Ambiguous grammars allow multiple correct parse trees

• “Fruit flies like a banana”
Associativity is one source of ambiguity

• $e ::= \text{num} \mid e + e \mid e - e$

• $1 + 2 + 3 + 4 \rightarrow ((1 + 2) + 3) + 4, (1 + 2) + (3 + 4), \ldots$

• Solution:
  $e ::= \text{num} \mid e + \text{num} \mid e - \text{num} \mid e + (e) \mid e - (e)$
Precedence is one possible source of ambiguity

• Abstract syntax: \( e ::= e + e \mid e - e \mid e \ast e \mid e \div e \)
• \( 1 + 2 \ast 3 - 4 \) ????

• Solution: Factoring out productions

• \( f ::= \text{num} \mid (e) \)
  \( t ::= f \mid t \ast f \mid t \div f \)
  \( e ::= t \mid e + t \mid e - t \)
Classic example: “dangling else”

• $s ::= \text{if } e\ s \mid \text{if } e\ s\ \text{else } s$

• if $e_1$ if $e_2\ s_1$ else $s_2$
  • By convention: if $e_1$ (if $e_2\ s_1$ else $s_2$)

• Solution:
  
  $\text{closedstmt ::= if } e\ \text{closedstmt else closedstmt}$
  $\mid \ldots \text{ (non-if stmts not ending with an openstmt)}$
  
  $\text{openstmt ::= if } e\ \text{closedstmt else openstmt} \mid \text{if } e\ \text{stmt}$
  $\mid \ldots \text{ (non-if stmts ending with an openstmt)}$
  
  $\text{stmt ::= openstmt} \mid \text{closedstmt}$