Course Overview

PART I: overview material
1 Introduction
2 Language processors (tombstone diagrams, bootstrapping)
3 Architecture of a compiler

PART II: inside a compiler
4 Syntax analysis
5 Contextual analysis
6 Runtime organization
7 Code generation

PART III: conclusion
8 Interpretation
9 Review

Nullable, First sets (starter sets), and Follow sets

- A non-terminal is **nullable** if it derives the empty string
- First(N) or starters(N) is the set of all terminals that can begin a sentence derived from N
- Follow(N) is the set of terminals that can follow N in some sentential form

Next we will see algorithms to compute each of these.

Algorithm for computing Nullable

For each terminal t
Nullable(t) = false

For each non-terminal N
Nullable(N) = is there a production N ::= ε?

Repeat
For each production N ::= x₁ x₂ x₃ … xₙ
If Nullable(xᵢ) for all of xᵢ then set Nullable(N) to true
Until nothing new becomes Nullable

Generalizing the definition of Nullable

Define Nullable(x₁ x₂ x₃ … xₙ) as:
if n==0 then
true
else if !Nullable(x₁) then
false
else
Nullable(x₂ x₃ … xₙ)

Algorithm for computing First sets

For each terminal t
First(t) = { t }

For each non-terminal N
First(N) = { }

Repeat
For each production N ::= x₁ x₂ x₃ … xₙ
First(N) = First(N) ∪ First(x₁)
For each i from 2 through n
If Nullable(x₁ … xᵢ₋₁), then
First(N) = First(N) ∪ First(xᵢ)
Until no First set changes

Generalizing the definition of First sets

Define First(x₁ x₂ x₃ … xₙ) as:
if !Nullable(x₁) then
First(x₁)
else
First(x₁) ∪ First(x₂ x₃ … xₙ)

Note: some textbooks add ε (empty string) to First(N) whenever N is nullable, so that First(N) is never { } (empty set)
Algorithm for computing Follow sets

Follow(S) = {$} // the end-of-file symbol
For each non-terminal N other than S
Follow(N) = { }
Repeat
For each production N ::= x1 x2 x3 … xn
    If xi is a non-terminal then
    Follow(xi) = Follow(xi) ∪ First(x1+1 … xn)
For each i from n downto 1
    If xi is a non-terminal and Nullable(x1+1 … xn) then
    Follow(xi) = Follow(xi) ∪ Follow(N)
Until no Follow set changes

Example of computing Nullable, First, Follow

S ::= TUVW | WVUT
T ::= aT | e
U ::= Ub | f
V ::= eV | ε
W ::= Wd | ε

<table>
<thead>
<tr>
<th>Nullable?</th>
<th>First</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>[a, c, d, f]</td>
<td>{$}</td>
</tr>
<tr>
<td>T</td>
<td>[a, e]</td>
<td>[f, {$}]</td>
</tr>
<tr>
<td>U</td>
<td>[f]</td>
<td>{c, d, s, a, e, b}</td>
</tr>
<tr>
<td>V</td>
<td>[c] or [c, e]</td>
<td>[d, s, f]</td>
</tr>
<tr>
<td>W</td>
<td>[d] or [d, ε]</td>
<td>{$, c, f, d}</td>
</tr>
</tbody>
</table>

Parsing

We will now look at parsing.

Topics:
- Some terminology
- Different types of parsing strategies
  - bottom up
  - top down
- Recursive descent parsing
  - What is it
  - How to implement a parser given an EBNF specification

Different kinds of Parsing Algorithms

- Two big groups of algorithms can be distinguished:
  - bottom up strategies
  - top down strategies
- Example: parsing of “Micro-English”

Sentence ::= Subject Verb Object.
Subject ::= I | A Noun | The Noun
Object ::= me | a Noun | the Noun
Noun ::= cat | bat | rat
Verb ::= like | is | see | sees

The cat sees the rat.
The rat likes me.
The rat sees me.
I like a cat.
I see the rat.
I see a rat.

Bottom up parsing

The parse tree “grows” from the bottom (leaves) up to the top (root).
Top-down parsing

The parse tree is constructed starting at the top (root).

Sentence
   Subject Verb Object
       Noun       Verb       Noun
   The  cat     sees    a  rat

Quick review

- Syntactic analysis
  - Lexical analysis
    - Group letters into words (or group characters into tokens)
    - Use regular expressions and deterministic FSM’s
  - Grammar transformations
    - Left-factoring
    - Left-recursion removal
    - Substitution
  - Parsing = structural analysis of program
    - Group words into sentences, paragraphs, and documents
      (or tokens into expressions, commands, and programs)
    - Top-Down and Bottom-Up

Recursive Descent Parsing

- Recursive descent parsing is a straightforward top-down parsing algorithm.
- We will now look at how to develop a recursive descent parser from an EBNF specification.
- Idea: the parse tree structure corresponds to the recursive calling structure of parsing functions that call each other.

Recursive Descent Parsing: Auxiliary Methods

```java
public class MicroEnglishParser {
    private TerminalSymbol currentTerminal;
    // Auxiliary methods will go here
    // Parsing methods will go here
}
```

Define a procedure parseN for each non-terminal N

```java
private void parseSentence( ) {
    private void parseSubject( );
    private void parseObject( );
    private void parseNoun( );
    private void parseVerb( );
}
```

```java
public class MicroEnglishParser {
    private TerminalSymbol currentTerminal;
    // Auxiliary methods will go here
    ...
    // Parsing methods will go here
}
```
Recursive Descent Parsing: Parsing Methods

```java
private void parseSentence() {
    parseSubject();
    parseVerb();
    parseObject();
    accept('.');
}
```

```java
private void parseSentence() {
    parseSubject();
    parseVerb();
    parseObject();
    accept('.');
}
```

```java
Sentence ::= Subject Verb Object

Sentence ::= Subject Verb Object
```

```java
private void parseSubject() {
    if (currentTerminal matches 'I')
        accept('I');
    else if (currentTerminal matches 'A') {
        accept('A');
        parseNoun();
    } else if (currentTerminal matches 'The') {
        accept('The');
        parseNoun();
    } else
        report a syntax error
}
```

```java
private void parseSubject() {
    if (currentTerminal matches 'I')
        accept('I');
    else if (currentTerminal matches 'A') {
        accept('A');
        parseNoun();
    } else if (currentTerminal matches 'The') {
        accept('The');
        parseNoun();
    } else
        report a syntax error
}
```

```java
Subject ::= I | A Noun | The Noun
```

```java
Noun ::= cat | bat | rat
```

```java
private void parseNoun() {
    if (currentTerminal matches 'cat')
        accept('cat');
    else if (currentTerminal matches 'bat')
        accept('bat');
    else if (currentTerminal matches 'rat')
        accept('rat');
    else
        report a syntax error
}
```

```java
private void parseNoun() {
    if (currentTerminal matches 'cat')
        accept('cat');
    else if (currentTerminal matches 'bat')
        accept('bat');
    else if (currentTerminal matches 'rat')
        accept('rat');
    else
        report a syntax error
}
```

```java
Verb ::= like | is | see | sees
```

```java
Object ::= me | a Noun | the Noun
```

```
private void parseObject() {
    ?
}
```

```
private void parseVerb() {
    ?
}
```

Test yourself: Can you complete parseObject() and parseVerb()?