Syntax and Semantics

Alark Joshi
TBL quizzes

• Individual quiz
  o Divide your 4 points accordingly
  o 4 on 1 answer if you are confident
  o 2-2 on 2 answers if you are a bit unsure
  o 1 on all if you want to get at least 1 point

• Team quiz
  o 4 points for getting it correct on first scratch off
  o 2 for 2\textsuperscript{nd} scratch off
  o 1 for 3\textsuperscript{rd} and 0 for last scratch off
Introduction

• **Syntax**: the form or structure of the expressions, statements, and program units

• **Semantics**: the meaning of the expressions, statements, and program units
Syntax and Semantics

• Syntax and semantics provide a language’s definition
  o Users of a language definition
    • Other language designers
    • Implementers
    • Programmers (the users of the language)
The General Problem of Describing Syntax: Terminology

• What is a sentence?
  o A *sentence* is a string of characters over some alphabet

• What is a language?
  o A *language* is a set of sentences
The General Problem of Describing Syntax: Terminology

- A **lexeme** is the lowest level syntactic unit of a language (e.g., *, sum, begin)

- A **token** is a category of lexemes (e.g., identifier)

New terminology alert!!
Formal Definition of Languages

• **Recognizers**
  o A recognition device reads input strings over the alphabet of the language and decides whether the input strings belong to the language
  o Example: syntax analysis part of a compiler

• **Generators**
  o A device that generates sentences of a language
  o One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator
Grade distribution for Q1

- Individual Test
  - Minimum score = 14
  - Maximum score = 40
  - Average score = 31.25

- Team test
  - Minimum score = 38
  - Maximum score = 40
  - Average score = 39.7
BNF and Context-Free Grammars

• Context-Free Grammars (CFG)
  o Developed by Noam Chomsky in the mid-1950s
  o Language generators, meant to describe the syntax of natural languages
  o Define a class of languages called context-free languages

• Backus-Naur Form (1959)
  o Invented by John Backus to describe Algol 58
  o The notation for CFG is often called Backus-Naur Form (BNF)
Context-Free Grammars

• A CFG consists of
  o A set of terminals $T$
  o A set of non-terminals $N$
  o A start symbol $S$ (a non-terminal)
  o A set of productions/rules

<assign> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
Regular Expressions

- A regular expression is one of the following:
  - A character
  - The empty string, denoted by $\varepsilon$
  - Two regular expressions concatenated
  - Two regular expressions separated by $|$ (i.e., or)
  - A regular expression followed by the Kleene star $^*$ (concatenation of zero or more strings)
Numerical literals in Pascal may be generated by the following:

\[
\begin{align*}
digit & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
unsigned\_integer & \rightarrow digit\ digit^* \\
unsigned\_number & \rightarrow unsigned\_integer\ (( .\ unsigned\_integer) \mid \epsilon) \\
               & \mid ((( e \mid E) ( + \mid - \mid \epsilon) unsigned\_integer) \mid \epsilon)
\end{align*}
\]
BNF Fundamentals

• Abstractions are used to represent classes of syntactic structures
  o Like syntactic variables (also called nonterminal symbols, or just non-terminals)

• Terminals are lexemes or tokens

• A rule has
  o LHS - which is a nonterminal,
  o RHS - which is a string of terminals and/or nonterminals

```
digit      →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```
BNF Fundamentals

- Nonterminals are often enclosed in angle brackets

- A simple rule
  
  `<if_stmt> → if <logic_expr> then <stmt>`

- Grammar: a finite non-empty set of rules

- A start symbol is a special element of the nonterminals of a grammar
BNF Rules

- An abstraction (or nonterminal symbol) can have more than one RHS

\[
<\text{stmt}> \rightarrow <\text{single\_stmt}>
\]
\[
| \text{begin } <\text{stmt\_list}>, \text{end}
\]

\[
digit \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
\]
Rules can be recursive

• Syntactic lists are described using recursion
  \[ \text{ident_list} \rightarrow \text{identifier} \mid \text{identifier, ident_list} \]

• A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)
In-class activity

Modify the following grammar to accept a sentence $A = A + B + C$

$$
\text{<assign>} \rightarrow \text{<var>} = \text{<expression>}
\text{<var>} \rightarrow A \mid B \mid C
\text{<expression>} \rightarrow \text{<var>} + \text{<var>}
$$
Alternate Grammar

<assign> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
<expression> → <expression> + <var> | <var>
    → <expression> + <var> + <var>
    → <var> + <var> + <var>
    → A + B + C
An Example Grammar

\[
\begin{align*}
\langle \text{program} \rangle & \rightarrow \langle \text{stmts} \rangle \\
\langle \text{stmts} \rangle & \rightarrow \langle \text{stmt} \rangle \mid \langle \text{stmt} \rangle ; \langle \text{stmts} \rangle \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle \\
\langle \text{var} \rangle & \rightarrow a \mid b \mid c \mid d \\
\langle \text{expr} \rangle & \rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle - \langle \text{term} \rangle \\
\langle \text{term} \rangle & \rightarrow \langle \text{var} \rangle \mid \text{const}
\end{align*}
\]
An Example Derivation

\[ \langle \text{program} \rangle \Rightarrow \langle \text{stmts} \rangle \Rightarrow \langle \text{stmt} \rangle \]

\[ \Rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle \]

\[ \Rightarrow \quad a = \langle \text{expr} \rangle \]

\[ \Rightarrow \quad a = \langle \text{term} \rangle + \langle \text{term} \rangle \]

\[ \Rightarrow \quad a = \langle \text{var} \rangle + \langle \text{term} \rangle \]

\[ \Rightarrow \quad a = b + \langle \text{term} \rangle \]

\[ \Rightarrow \quad a = b + \text{const} \]
Another Valid Derivation

• Work out a valid derivation for the following grammar

<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | const

• Get your neighbor to verify its validity
Derivations

• A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)

• BNF is a generator
  o Use the grammar to generate sentences that belong to the language described by that grammar
Derivations

• Every string of symbols in a derivation is a *sentential form*
• A *sentence* is a sentential form that has only terminal symbols
• A *leftmost derivation* is one in which the leftmost nonterminal in each sentential form is the one that is expanded
• A derivation may be neither leftmost nor rightmost
Parse Tree

• A hierarchical representation of a derivation

```
<program>
    |
<stmts>
    |
<stmt>
    |
<var> = <expr>
    |
    a <term> + <term>
    |
    <var> const
    |
    b
```
Consider this rule

\[
\begin{align*}
\text{stmt} & \rightarrow \text{if-stmt} \mid \ldots \\
\text{if-stmt} & \rightarrow \\
& \quad \text{if} \ \text{bool-expr} \ \text{then} \ \text{stmt} \\
& \quad \mid \ \text{if} \ \text{bool-expr} \ \text{then} \ \text{stmt} \ \text{else} \ \text{stmt} \\
\text{bool-expr} & \rightarrow \ p \mid q
\end{align*}
\]

- Draw the parse tree for
  
  \[
  \text{if p then} \\
  \qquad \text{if q then stmt2} \\
  \text{else stmt3}
  \]
Ambiguity in Grammars

- A grammar is *ambiguous* if and only if it generates a sentential form that has *two or more* distinct parse trees.
An Ambiguous Expression Grammar

\[
\begin{align*}
<\text{expr}> & \rightarrow <\text{expr}> \ <\text{op}> \ <\text{expr}> \ | \ \text{const} \\
<\text{op}> & \rightarrow / \ | \ -
\end{align*}
\]
An Unambiguous Expression Grammar

- If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity.

\[
\text{<expr>} \rightarrow \text{<expr>} - \text{<term>} \mid \text{<term>}
\]

\[
\text{<term>} \rightarrow \text{<term>} / \text{const} \mid \text{const}
\]

![parse tree diagram]
Let us draw the parse tree for $A = B * (C + B)$.
Derive

- Derive the following sentence for the grammar:
  - $C = B + (B * (A + C))$
Parse tree

• Draw the parse tree for:

  o  C = B + ( B * ( A + C))

<assign>  →  <id> = <expr>
{id>  →  A | B | C
<expr>  →  <id> + <expr> | <id> * <expr> | ( <expr> ) | <id>
Ambiguous grammar?

- Parse tree for \( A = B + C \times A \)

- Two distinct parse trees possible
  1. \(<\text{assign}> \rightarrow <\text{id}> = <\text{expr}>\)
     \(<\text{id}> = <\text{expr}> + <\text{expr}>\)
     ...
  2. \(<\text{assign}> \rightarrow <\text{id}> = <\text{expr}>\)
     \(<\text{id}> = <\text{expr}> \times <\text{expr}>\)
     ...

Operator Precedence

• Operator precedence implemented in most languages to avoid ambiguity
• In math as well as most programming language, multiplication takes higher precedence over addition/subtraction
• $5 + 2 \times 3 = ?$ 21 or 11?
• What would you do if you wanted the answer to be 21?
Operator Precedence

• Refresher

• Brackets first
• Orders (i.e. Powers and Square Roots, etc.)
• Division and Multiplication (left-to-right)
• Addition and Subtraction (left-to-right)

• BODMAS
Operator Precedence

• Operator precedence rules help to make the previous grammar unambiguous

\[\text{assign} \rightarrow \text{id} = \text{expr}\]
\[\text{id} \rightarrow A \mid B \mid C\]
\[\text{expr} \rightarrow \text{expr} + \text{term} \mid \text{term}\]
\[\text{term} \rightarrow \text{term} * \text{factor} \mid \text{factor}\]
\[\text{factor} \rightarrow ( \text{expr} ) \mid \text{id}\]
Operator Precedence

- Drawing a parse tree useful to determine operator precedence
- \( C = B + (A * C) \)
Associativity of Operators

- Operator associativity can also be indicated by a grammar

\[
<\text{expr}> \rightarrow <\text{expr}> + <\text{expr}> \mid \text{const} \quad \text{(ambiguous)}
\]

\[
<\text{expr}> \rightarrow <\text{expr}> + \text{const} \mid \text{const} \quad \text{(unambiguous)}
\]

Associativity example: \((5 + 2) + 1 = 5 + (2 + 1) = 8\)
Monkey Language

<sentence> -> <word>|<sentence>#<word>
<word> -> <syllable> | <syllable><word><syllable>
<syllable> -> <plosive> | <plosive><stop> | a<plosive> | a<stop>
<plosive> -> <stop>a
<stop> -> b | d

• Which are valid?
• ba#ababadada#bad#dabbada
• abdabaadab#ada
• dad#ad#abaadad#badadbaad