COEN 175

Lecture 2: Syntax Analysis and Grammars

Syntax Analysis

- Syntax analysis (i.e., parsing) is the process of taking the sequence of tokens and grouping them **structurally** into meaningful sentences.
- Syntax refers to structure, whereas semantics refers to meaning.
- Are regular expressions sufficient for describing the structure of a programming language?
- Consider matching balanced parentheses:
  - Actually, for clarity, we'll match balanced "*a*"s and "*b*"s.
  - Is the language $\{a^n b^n \mid n \geq 0\}$ regular?

Regular Languages

- Every regular expression describes a regular language, and vice versa.
- However, not all languages are regular:
  - $\{a^m b^n \mid m, n \geq 0\}$ is regular.
  - $\{a^n b^n \mid k \geq n \geq 0\}$ is regular.
  - $\{a^n b^n \mid n \geq 0\}$ is **not** regular.
- Informally, a regular expression can only "remember" a finite amount of information.
  - We cannot remember an arbitrary number of "*a*"s in order to make sure we match the same number of "*b*"s.

Context-Free Languages

- A context-free language is more powerful than a regular language.
- Each context-free language is described by a **context-free grammar** (CFG), and vice versa.
- A grammar consists of four parts:
  - A set, T, of **terminal** symbols.
  - A set, NT, of **nonterminal** symbols.
  - A special nonterminal, S, called the **start symbol**.
  - A set of rules or **productions** of the form $NT \rightarrow (T \cup NT)^*$.

Context-Free Grammars

- General typesetting conventions:
  - Nonterminals are uppercase letters and words.
  - Terminals are operators, lowercase letters, and bold words like **id** (often written `ID` on the board).
  - Greek letters are strings of terminals or nonterminals.
- Our first grammar:

  $L \rightarrow$ **id**
  $L \rightarrow L$ **,** **id**

  - OR -

  $L \rightarrow$ **id**
  | L **,** **id**

- Literally, a list, *L*, is either an identifier, or is another list followed by a comma and an identifier.

Derivations

- We say that $\alpha A \beta \Rightarrow \alpha\gamma\beta$ is a **derivation** if there is a production $A \rightarrow \gamma$.
  - In other words, we can replace a nonterminal *A* with the right-hand side of one of its productions.
- What sentences can we derive from our grammar?
  - $L \Rightarrow$ **id**
  - $L \Rightarrow L$ **, id** $\Rightarrow$ **id , id**
  - $L \Rightarrow L$ **, id** $\Rightarrow L$ **, id , id** $\Rightarrow$ **id , id , id**
- This grammar generates a non-empty, comma-separated list of identifiers.

Specific Derivations

- Usually, there is more than one possible derivation for the same sentence. Consider this grammar:

  $S \rightarrow A\ B$
  $A \rightarrow a$
  $B \rightarrow b$

- This grammar only generates one sentence, but can do so in two different ways:
  - $S \Rightarrow A\ B \Rightarrow a\ B \Rightarrow ab$.
  - $S \Rightarrow A\ B \Rightarrow A\ b \Rightarrow ab$.

- In a **leftmost** derivation, we always replace the leftmost nonterminal (and similarly for **rightmost**).

Expression Grammars

- Consider the following grammar for expressions:
  - $E \rightarrow E + E \mid E * E \mid \mathbf{id}$.
- Derive the input sentence **id** + **id** $*$ **id** using a leftmost derivation.
  - $E \Rightarrow E + E \Rightarrow \mathbf{id} + E \Rightarrow \mathbf{id} + E * E \Rightarrow \mathbf{id} + \mathbf{id} * E \Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}$.
  - $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow \mathbf{id} + E * E \Rightarrow \mathbf{id} + \mathbf{id} * E \Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}$.
- Which derivation should we use? It depends.
  - Show that the input **id** + **id** + **id** also has two different leftmost derivations.
  - Again, which should we choose? It depends.

Parse Trees

- A **parse tree** is a graphical representation of a derivation sequence.
  - The root of the tree is the start symbol.
  - Each nonleaf node is a nonterminal.
  - The leaves are the terminals of the input sentence.
  - If a nonleaf node, X, has children Y_1, Y_2, ..., Y_n (from left to right), then $X \rightarrow Y_1\ Y_2\ \ldots\ Y_n$ is a production.
- Top-down parsers construct parse trees from the root to the leaves.
- Bottom-up parsers construct parse trees from the leaves to the root.

Ambiguous Grammars

- A grammar is said to be **ambiguous** if, for **some** input sentence, **any** of the following are true:
  - There is more than one leftmost derivation.
  - There is more than one rightmost derivation.
  - There is more than one parse tree.
- Our compiler cannot use an ambiguous grammar.
  - How would you like it if the compiler decided arbitrarily the order of multiplication and addition?
- Therefore, we must first **disambiguate** the grammar without changing the language being defined.

Undecidability

- It would be nice if we had a tool that could tell us whether a grammar was ambiguous.
- Also, after disambiguation, we must be sure we have not changed the language being defined.
- It would be nice if we had a tool to tell us whether two grammars define the same language.
  - Unfortunately, these tools cannot exist.
- These two problems are **undecidable**: it is impossible to construct such an algorithm.

Recursive Grammars

- Draw parse trees for the input sentence **id , id , id** using each of the following grammars:
  - $L \rightarrow$ **id** | L , **id**
  - $L \rightarrow$ **id** | **id** , L
- The first grammar is said to be **left recursive**.
  - Left recursive grammars yield **left-heavy** parse trees.
  - We want left-heavy parse trees for **left associativity**.
- The second grammar is said to be **right recursive**.
  - Right recursive grammars yield **right-heavy** parse trees.
  - We want right-heavy parse trees for **right associativity**.