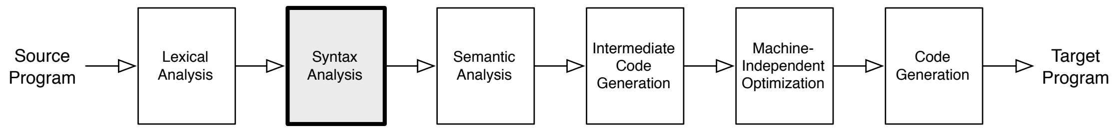
CS 301 Spring 2021 T4
Class Assignment
20 April

Context-Free Grammars

1 Plan



This assignment explores context-free grammars, the formal tool used to specify programming language syntax and the foundation for parsing. There are two major families of parsing techniques: *top-down* and *bottom-up*. This term, we will not cover top-down (a.k.a., *predictive*) parsing (and LL grammars) in depth, but additional readings and exercises are included here if you are curious to learn about them. Our next tutorial will focus on bottom-up parsing and LR/LALR grammars.

2 Readings

- EC 3.1–3.2
  Alternative: Dragon 4.1.1–4.1.2, 4.2–4.2.5, 4.2.7, 4.3–4.3.2
- Skim EC 3.3.0, 3.3.2
  Alternative: Skim Dragon 4.4–4.4.1
- **Extra Depth:**
  - Top-down parsing: EC 3.3 / Dragon 4.3.3–4.3.4, 4.4–4.4.4
  - Additional detail on grammars, top-down parsing, error recovery: all sections Dragon 4.1–4.4

3 Exercises

1. Dragon Exercise 4.2.1
2. Dragon Exercise 4.2.3 (a) — (e)
3. *Tiny Compiler Front End* Sections 5-8 / Exercises 4-14
   `https://cs.wellesley.edu/~cs301/s21/project/tiny/tiny-front.pdf`
4. **Extra Depth:** Dragon Exercise 4.3.1
5. **Extra Depth:** Consider the following grammar:

$$S \quad \rightarrow \quad \mathbf{a}\, S\, \mathbf{b}\, S \;\mid\; \mathbf{b}\, S\, \mathbf{a}\, S \;\mid\; \epsilon$$

   (a) Show that the grammar is ambiguous by constructing two different rightmost derivations for some string.
   (b) Construct the corresponding parse trees for this string.
   (c) **Extra Depth:** Write an unambiguous grammar that describes the same language. (There is no algorithm to remove ambiguiuty from a grammar. I suggest first trying to understand the language for the original grammar and then constructing a new unambiguous CFG from scratch that accepts the same language.)

6. **Extra Depth:** Consider the following grammar:

$$\begin{array}{lcl} S & \rightarrow & B\ C\ \mathbf{z} \\ B & \rightarrow & \mathbf{x}\ B \mid D \\ C & \rightarrow & \mathbf{u}\ \mathbf{v} \mid \mathbf{u} \\ D & \rightarrow & \mathbf{y}\ D \mid \epsilon \end{array}$$

   (a) Is this grammar LL(1)? Explain why (not). If not, modify the grammar to be LL(1) before proceeding.

   (b) Compute the FIRST and FOLLOW sets for the (possibly modified) grammar.

   (c) Construct the LL(1) parsing table.
   NOTE: There is a typo in the Dragon book in the description of how to construct the parsing table. On page 224, step 1 of Algorithm 4.31 should refer to FIRST(α), and *not* FIRST(A). This has been fixed in some printings (international paperback?) but not all.

   (d) Show the steps taken to parse **xxyuz** with your table. (Use Dragon Fig. 4.21 as an example of how to show the parser's progress.)

7. **Extra Depth:** Consider the following grammar for statements:

$$\begin{array}{lcl} \textit{Stmt} & \rightarrow & \texttt{if E then}\ \textit{Stmt}\ \textit{StmtTail} \\ & \mid & \texttt{while E}\ \textit{Stmt} \\ & \mid & \{\ \textit{List}\ \} \\ & \mid & \texttt{S} \\ \\ \textit{StmtTail} & \rightarrow & \texttt{else}\ \textit{Stmt} \\ & \mid & \epsilon \\ \\ \textit{List} & \rightarrow & \textit{Stmt}\ \textit{ListTail} \\ \\ \textit{ListTail} & \rightarrow & ;\ \textit{List} \\ & \mid & \epsilon \end{array}$$

Unlike Java (and like ML), semicolons separate consecutive statements. You can assume `E` and `S` are terminals that represent other expression and statement forms that we do not currently care about. If we resolve the typical conflict regarding expansion of the optional `else` part of an `if` statement by preferring to consume an `else` from the input whenever we see one, we can build a predictive parser for this grammar.

   (a) Build the LL(1) predictive parser table for this grammar.

   (b) Using Figure 4.21 in the Dragon book as a model, show the steps taken by your parser on input:

   `if E then S else while E { S }`

   (c) **Extra Depth:** Use the techniques outlined in Dragon 4.4.5 to add error-correcting rules to your table.

   (d) **Extra Depth:** Describe the behavior of your parser on the following two inputs:

      i. `if E then S ; if E then S }`

      ii. `while E { S ; if E S ; }`