Today

- Formal Grammars (cont)

Assignments

- HW2 out (due Thur)
- Quiz 2 on Thurs
**Grammar Rules (Review)**

Grammar rules define “productions” ("rewritings")

\[ s \rightarrow a \]

- we say \( s \) “produces” (or “yields”) \( a \)

**Concatenation**

\[ s \rightarrow ab \]

- \( s \) yields the string \( a \) followed by the string \( b \)

**Alternation**

\[ s \rightarrow a \mid b \]

- \( s \) yields the string \( a \) or \( b \)

**The empty string**

\[ s \rightarrow a \mid \epsilon \]

- \( \epsilon \) denotes the special “empty” terminal
- \( s \) yields either the string \( a \) or "" (empty string)

**Kleene Star (Closure)**

\[ s \rightarrow a^* \]

- \( s \) yields the strings with zero or more a’s
Recursion

\[ s \rightarrow asb \mid ab \]

- \( s \) yields the strings with \( n \) a’s followed by \( n \) b’s

Exercise: Try to define some languages ...

(1). The set of strings consisting of (only) zero or more a’s followed by a semi-colon.

(2). The set of strings that consist of an odd number of a’s followed by the same number of b’s.

(3). The set of strings for performing zero or more sequences of addition operations over the values 0 and 1, where \( 1 + 1 \), \( 1 + 0 + 1 \), \( 0 + 0 + 1 + 0 \) are all in the language.
Parsing: An example grammar

Simple list of assignment statements

\[
\begin{align*}
\text{stmt}\_\text{list} & \rightarrow \text{stmt} \mid \text{stmt} \ ';;' \ \text{stmt}\_\text{list} \\
\text{stmt} & \rightarrow \text{var} \ '=' \ \text{expr} \\
\text{var} & \rightarrow \ 'A' \mid 'B' \mid 'C' \\
\text{expr} & \rightarrow \text{var} \mid \text{var} \ '+' \ \text{var} \mid \text{var} \ '-' \ \text{var}
\end{align*}
\]

– Note: many possible grammars for our language!

We can use grammars to generate strings (derivations)

1. choose a production (start symbol on left-hand side)
2. replace with right-hand side
3. pick a non-terminal $N$ and production with $N$ on left side
4. replace $N$ with production's right-hand side
5. continue until only terminals remain
Example derivation of "A = B + C; B = A"

\[
\text{stmt\_list} \Rightarrow \text{stmt} ; \text{stmt\_list} \\
\Rightarrow \text{var} = \text{expr} ; \text{stmt\_list} \\
\Rightarrow \text{A} = \text{expr} ; \text{stmt\_list} \\
\Rightarrow \text{A} = \text{var} + \text{var} ; \text{stmt} \\
\Rightarrow \text{A} = \text{B} + \text{var} ; \text{stmt\_list} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{stmt\_list} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{stmt} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{var} = \text{expr} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{B} = \text{expr} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{B} = \text{var} \\
\Rightarrow \text{A} = \text{B} + \text{C} ; \text{B} = \text{C}
\]

• This is a "left-most" derivation
  - derived the string by replacing left-most non-terminals

• The opposite is a "right-most" derivation
  \[
\text{stmt\_list} \Rightarrow \text{stmt} ; \text{stmt\_list} \\
\Rightarrow \text{stmt} ; \text{stmt} \\
\Rightarrow \text{stmt} ; \text{var} = \text{expr} \\
\Rightarrow \text{stmt} ; \text{var} = \text{var} \\
\Rightarrow \text{stmt} ; \text{var} = \text{B} \\
\Rightarrow \ldots
\]
Derivations can also be written as “parse trees”

- Using the previous example derivation of “A = B + C; B = A”
Parsing

– A context free grammar is a "generator"
– Whereas a parser is a "recognizer"
  * given a token stream
  * determine if the stream is a derivation of the grammar
– Typically a parser also builds an Abstract Syntax Tree (AST)

We’ll look at $LL(k)$ parsers

– read from left-to-right, performing a left-most derivation
– parses top down (parse tree from the root down)
– at most $k$ look ahead symbols (more later)

Consider this (modified) rule:

$$stmt \rightarrow ( 'A' \mid 'B' \mid 'C' ) \ '=' \ expr$$

Assuming the parser knows this rule is to be applied ...

1. it calls lexer’s next_token
2. it checks if it is a literal "A", "B", or "C"
3. it calls lexer’s next_token
4. it checks that it is an ASSIGN token
5. and so on until it finishes the $stmt$ rule

– the parser produces an error if it finds a token it isn’t expecting