Introduction to Grammars

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

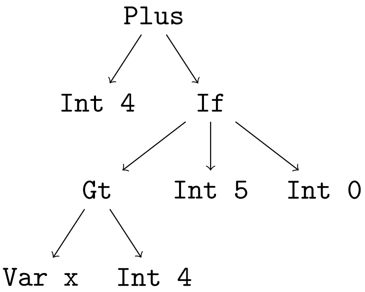
- Identify and explain the parts of a grammar.
- Define *terminal*, *nonterminal*, *production*, *sentence*, *parse tree*, *left-recursive*, *ambiguous*.
- Use a grammar to draw the parse tree of a sentence.
- Identify a grammar that is *left-recursive*.
- Identify, demonstrate, and eliminate ambiguity in a grammar.

The Problem We are Trying to Solve

- Computer programs are entered as a stream of ASCII (usually) characters.

```
4 + if x > 4 then 5 else 0
```

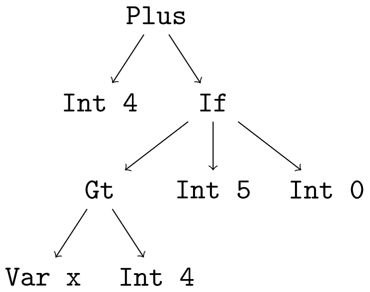
- We want to convert them into an *abstract syntax tree* (AST).



Haskell Code

Code

```
PlusExp (IntExp 4)
  (IfExp (GtExp (VarExp "X") (IntExp 4))
         (IntExp 5)
         (IntExp 0))
```



The Solution

The conversion from strings to trees is accomplished in two steps.

- First, convert the stream of characters into a stream of *tokens*.
  - This is called *lexing* or *scanning*.
  - Turns characters into words and categorizes them.
  - We will cover this in the next lecture.
- Second, convert the stream of tokens into an abstract syntax tree.
  - This is called *parsing*.
  - Turns words into *sentences*.

Definition of Grammar

A context free grammar G has four components:

- A set of terminal symbols representing individual tokens,
- A set of non terminal symbols representing syntax trees,
- A set of productions, each mapping a non terminal symbol to a string of terminal and non terminal symbols, and
- A designated non terminal symbol called the *start symbol*.

What Is In a Sentence?

When we specify a sentence, we talk about two things that could be in them.

1. *Terminals*: tokens that are atomic – they have no smaller parts (e.g., "nouns," "verbs," "articles")
2. *Non terminals*: clauses that are not atomic – they are broken into smaller parts (e.g., "prepositional phrase," "independent clause," "predicate")

Examples: (Identify the terminals and the non terminals.)

- A sentence is a noun phrase, a verb, and a prepositional phrase.
- A noun phrase is a determinant, and a noun.
- A prepositional phrase is a preposition and a noun phrase.

Notation

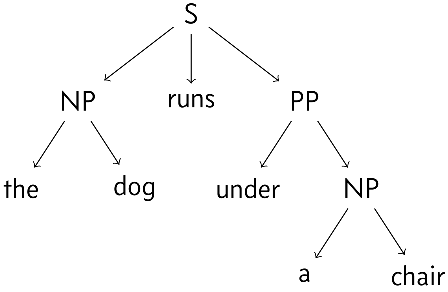
$$S \longrightarrow N \text{ verb } P$$
$$N \longrightarrow \text{det noun}$$
$$P \longrightarrow \text{prep } N$$

- Each of the above lines is called a *production*.
  The *symbol* on the left-hand side can be *produced* by collecting the symbols on the right-hand side.
- The capital identifiers are *non terminal* symbols.
- The lower case identifiers are *terminal* symbols.
- Because the left-hand side is only a single non terminal, the rules are *context free*.
  (Contrast: x $S \longrightarrow NP$ verb PP)

We Use Grammars to Make Trees

"The dog runs under a chair."

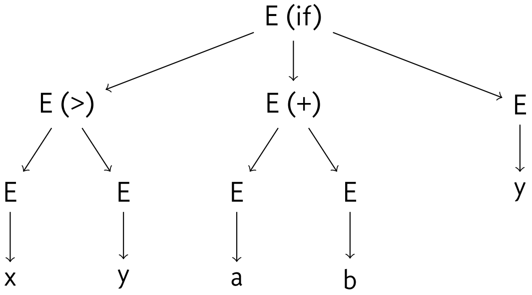
$$
\begin{aligned}
\text{S} &\rightarrow \text{NP verb PP}\\
\text{NP} &\rightarrow \text{det noun}\\
\text{PP} &\rightarrow \text{prep NP}
\end{aligned}
$$



Another Example ...

$$
\begin{array}{lll}
E \rightarrow & & E + E \\
& | & v \\
& | & E > E \\
& | & \text{if } E \text{ then } E \text{ else } E
\end{array}
$$

`if x > y then a + b else y`



Properties of Grammars

It is important to be able to say what properties a grammar has.

Epsilon Productions A production of the form "E $\rightarrow \epsilon$"
where ϵ represents the empty string

Right Linear Grammars where all the productions have the form
"E $\rightarrow$ x F" or "E $\rightarrow$ x"

Left-Recursive A production like "E $\rightarrow$ E + X"

Ambiguous More than one parse tree is possible for a specific sentence.

Epsilon Productions

- Sometimes we want to specify that a symbol can become nothing.
- Example: “E $\rightarrow \epsilon$”
- Another example:

  S $\rightarrow$ NP verb PP
  NP $\rightarrow$ det A noun
  PP $\rightarrow$ prep NP
  A $\rightarrow$ adjective A
  A $\rightarrow \epsilon$

  This says that adjectives are an optional part of noun phrases.

Right Linear Grammars

- A *right linear* grammar is one in which all the productions have the form "E → x A" or "E → x."
- This corresponds to the *regular languages*.
- Example: Regular expression `(10)*23` describes same language as this grammar:
  $A_0 \rightarrow 1A_1 \mid 2A_2$
  $A_1 \rightarrow 0A_0$
  $A_2 \rightarrow 3A_3$
  $A_3 \rightarrow \epsilon$
- The trick: Each node in your NFA is a non terminal symbol in the grammar. The terminal symbol represents an input, and the following nonterminal is the destination state.

Left-Recursive

- A grammar is *recursive* if the symbol being produced (the one on the left-hand side) also appears in the right-hand side.

  Example: "E $\longrightarrow$ if E then E else E"

- A grammar is *left-recursive* if the production symbol appears as the first symbol on the right-hand side.

  Example: "E $\longrightarrow$ E + F"

- ... or if is produced by a chain of left recursions ...

  Example:
  $$\begin{aligned} A &\longrightarrow Bx \\ B &\longrightarrow Ay \end{aligned}$$

Ambiguous Grammars

- A grammar is *ambiguous* if it can produce more than one parse tree for a single sentence.
- There are two common forms of ambiguity:
  - The "dangling else" form:
    E$\longrightarrow$ `if` E `then` E `else` E
    E$\longrightarrow$ `if` E `then` E
    E$\longrightarrow$ whatever
    Example: `if a then if x then y else z` … to which `if` does the `else` belong?
  - The "double-ended recursion" form: E$\longrightarrow$ E + E
    E$\longrightarrow$ E * E
    Example "3 + 4 * 5" … is it "(3 + 4) * 5" or "3 + (4 * 5)"?

Fixing Ambiguity

- The “double-ended recursion” form usually reveals a lack of precedence and associativity information. A technique called *stratification* often fixes this. To stratify your grammar:
  - Use recursion on only one side. Left-recursive means “associates to the left,” similarly right-recursive.
  - Put your highest precedence rules ‘‘lower” in the grammar.

E→ F + E

E→ F

F→ T * F

F→ T

T→ (E)

T→ integer

Next Up

- Parsing is hard! Let's break it up into parts.
- Compute FIRST sets:
  - What is the first symbol I could see when parsing a given non terminal?
- Compute FOLLOW sets:
  - What is the first symbol I could see *after* parsing a given non terminal?