Introduction to Grammars

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Objectives

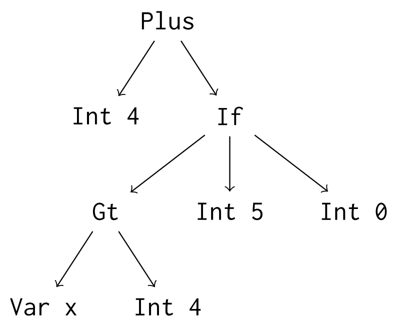
- Identify and explain the parts of a grammar.
- Define *terminal*, *nonterminal*, *production*, *sentence*, *parse tree*, *left-recursive*, *ambiguous*.
- Use a grammar to draw the parse tree of a sentence.
- Identify a grammar that is *left-recursive*.
- Know about *ambiguous grammars*:
  - Be able to identify, demonstrate, and eliminate ambiguity.

Reminder: The Problem

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

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

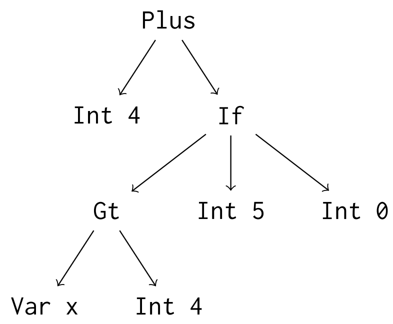
- We want to convert them into an *Abstract Syntax Tree*



Haskell Code

Code

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



Reminder: The Solution

Characters ⟶ Lexer ⟶ Tokens ⟶ Parser ⟶ Tree

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 did this in the last two lectures!
- Second, convert the stream of tokens into an abstract syntax tree.
  - This is called *parsing*.
  - Turns words into *sentences*.

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

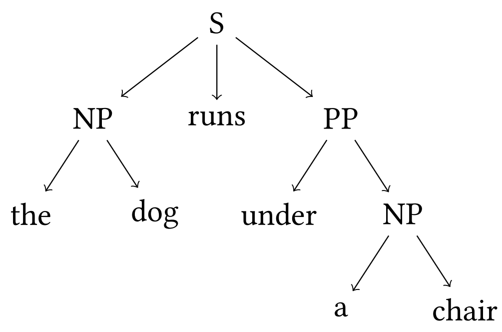
$$\begin{aligned} S &\rightarrow N \text{ verb } P \\ N &\rightarrow \text{det noun} \\ P &\rightarrow \text{prep } N \end{aligned}$$

- 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 ⟶ NP verb PP)

Grammars specify trees...

“The dog runs under a chair.”

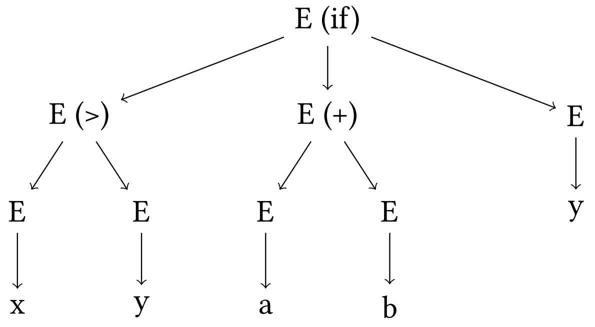
$$\begin{aligned} S &\rightarrow NP \text{ verb } PP \\ NP &\rightarrow \text{det noun} \\ PP &\rightarrow \text{prep } NP \end{aligned}$$



Another Example...

E → E + E
  → v
  → E > E
  → if E then E else E

`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 → ϵ", where ϵ represents the empty string.

Right Linear Grammars where all the productions have the form "E → x F" or "E → x".

Left-Recursive a production like "E → 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 → ϵ"
- Another example:
  S → NP verb PP
  NP → det A noun
  PP → prep NP
  A → adjective A
  A → ϵ
  This says that adjectives are an optional part of noun phrases.

Right Linear Grammars

- A *right linear* grammars 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:
  $A_0 \rightarrow 1A_1 \mid 2A_2$
  $A_1 \rightarrow 0A_0$
  $A_2 \rightarrow 3A_3$
  $A_3 \rightarrow \epsilon$

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: $A \longrightarrow Bx$
  $B \longrightarrow Ay$

Properties of Grammars

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)"?

Properties of Grammars

Fixing Ambiguity

- Ambiguity can often be eliminated by thinking more carefully about what you are trying to express with your grammar.
- "Dangling else" usually matches with the nearest `if`. This can be encoded in the grammar.

Properties of Grammars

Fixing Ambiguity

- The "double-ended recursion" form usually reveals a lack of precedence and associativity information. A technique called *stratification* often fixes this.
  - Left-recursive means "associates to the left", similarly right-recursive.
  - Higher precedence rules occur lower in the grammar.

E$\longrightarrow$ F + E
E$\longrightarrow$ F
F$\longrightarrow$ T * E
F$\longrightarrow$ T
T$\longrightarrow$ (E)
T$\longrightarrow$ 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?