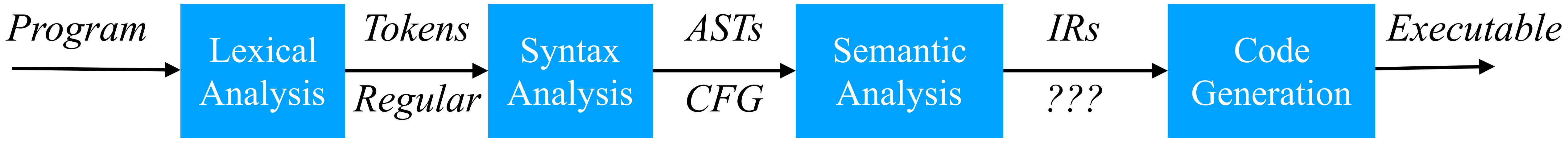
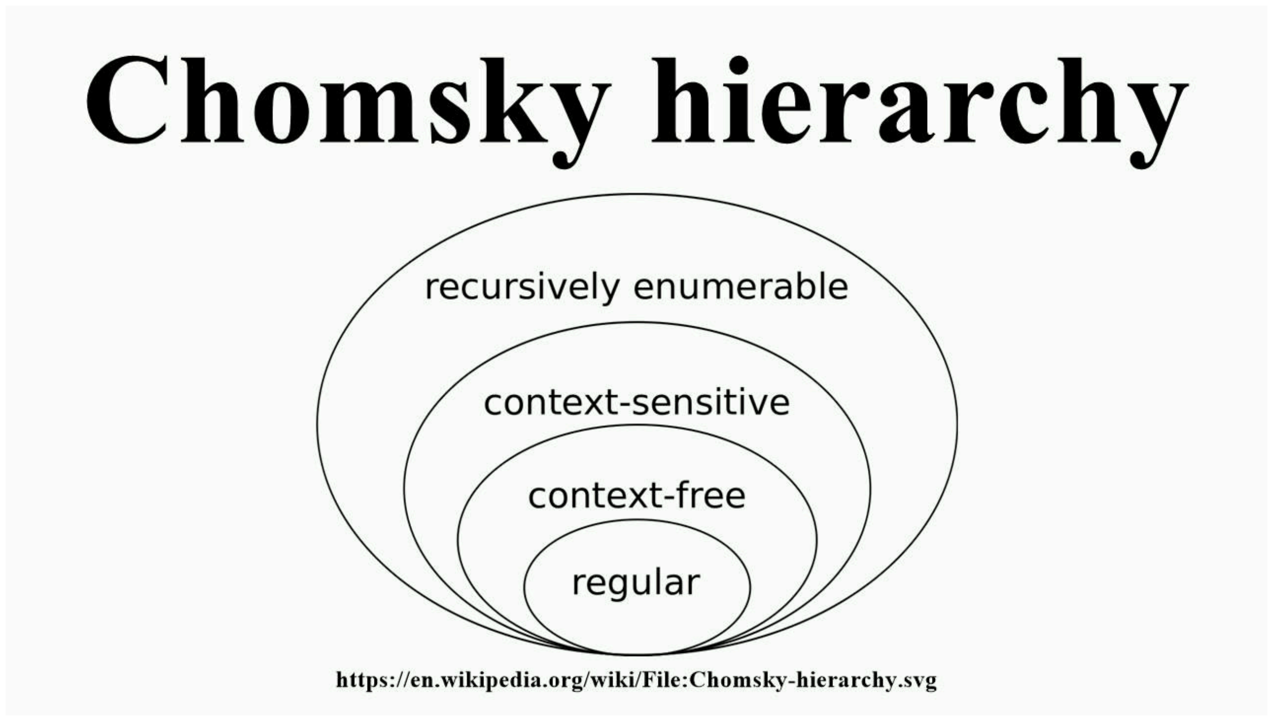
CS 160 Compilers

Lecture 8: Introduction to Parsing

Yu Feng
Fall 2021

A typical flow of a compiler





Lexical analysis

- Consider the following λ$^+$ program:

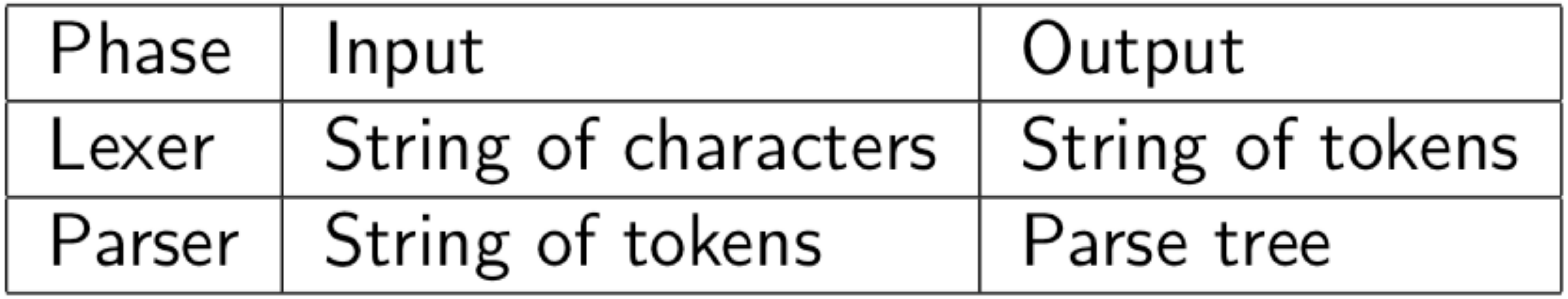
```
if x > y

then 10

else 8
```

- This program is just a string of characters

```
    if x > y\nthen\t10\nelse\t8
```

- Goal: Portion the input string into substrings where the substrings are ***tokens***

The role of a parser

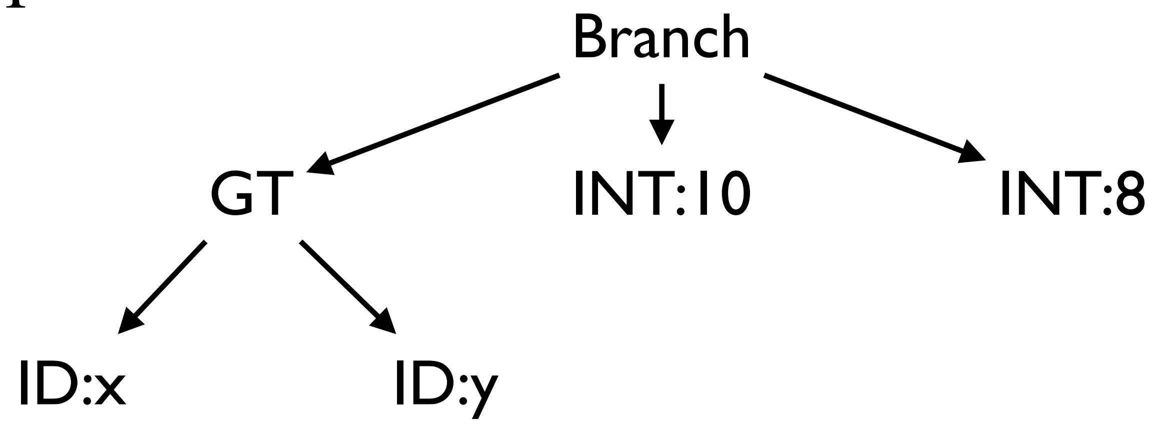


Phase	Input	Output
Lexer	String of characters	String of tokens
Parser	String of tokens	Parse tree

- Input: sequence of tokens from the lexer
- Output: parse tree (Abstract Syntax Tree) of the program

Example

- Input: Consider the previous Patina expression: **if** x>y **then** 10 **else** 8
- Parse Input: TOKEN_IF TOKEN_ID("x") TOKEN_GT TOKEN_ID("y") TOKEN_THEN TOKEN_INT(10) TOKEN_ELSE TOKEN_INT(8)
- Parser Output:


$$\begin{tikzcd}
 & & \text{Branch} \arrow[dll] \arrow[d] \arrow[dr] & \\
 & \text{GT} \arrow[dl] \arrow[dr] & \text{INT:10} & \text{INT:8} \\
\text{ID:x} & & \text{ID:y} &
\end{tikzcd}$$

The role of a parser

- Not all strings of tokens are programs...
- Parser must distinguish between valid and invalid strings of tokens
- What we need:
  - A language for describing valid strings of tokens
  - A method for recognizing if a string of tokens is in this language or not

Context free grammar (CFGs)

- Programming language constructs have ***recursive*** structure
- Example: A Patina expression is
  - *expression* + *expression*,
  - **if** *expression* **then** *expression* **else** expression, ...
- Context free grammars are a natural notation for this recursive structure

CFGs in more detail

- A CFG consists of:
  - A set of terminals T
  - A set of non-terminals N
  - A start symbol S (non-terminal)
  - A set of productions: $X \rightarrow Y_1 Y_2 \ldots Y_n$

where $X \in N$ and $Y_i \in (T \cup N \cup \{\varepsilon\})$

CFGs example

- Recall the earlier fragment of Patina:

EXPR → **if** *EXPR* **then** *EXPR* **else** *EXPR*

| *EXPR* + *EXPR*

| *ID*

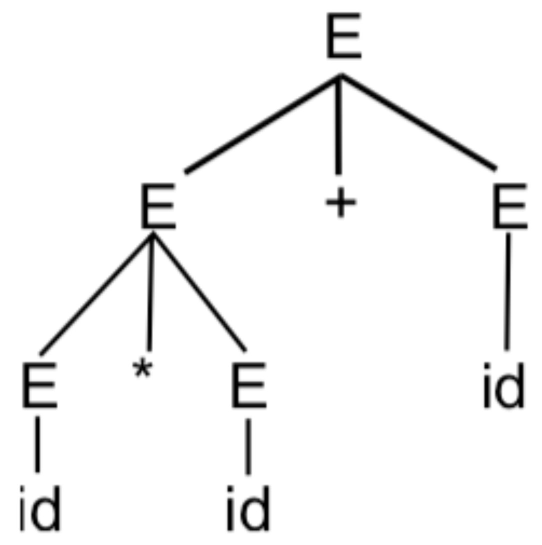
- Some strings in this language:

ID
IF ID THEN ID ELSE ID
ID + ID
IF ID THEN ID+ID ELSE ID
IF IF ID THEN ID ELSE IF THEN ID ELSE ID

From derivations to parse trees

- A derivation is a sequence of productions: $\boldsymbol{S} \to \ldots \to \ldots \to \ldots$
- A derivation can be drawn as a tree
    - Start symbol is the tree's root
- For a production $X \to Y_1 \ldots Y_n$ add children $Y_1 \ldots Y_n$ to node X

```
  E
→ E+E
→ E*E+E
→ id*E+E
→ id*id + E
→ id*id + id
```



Ambiguity

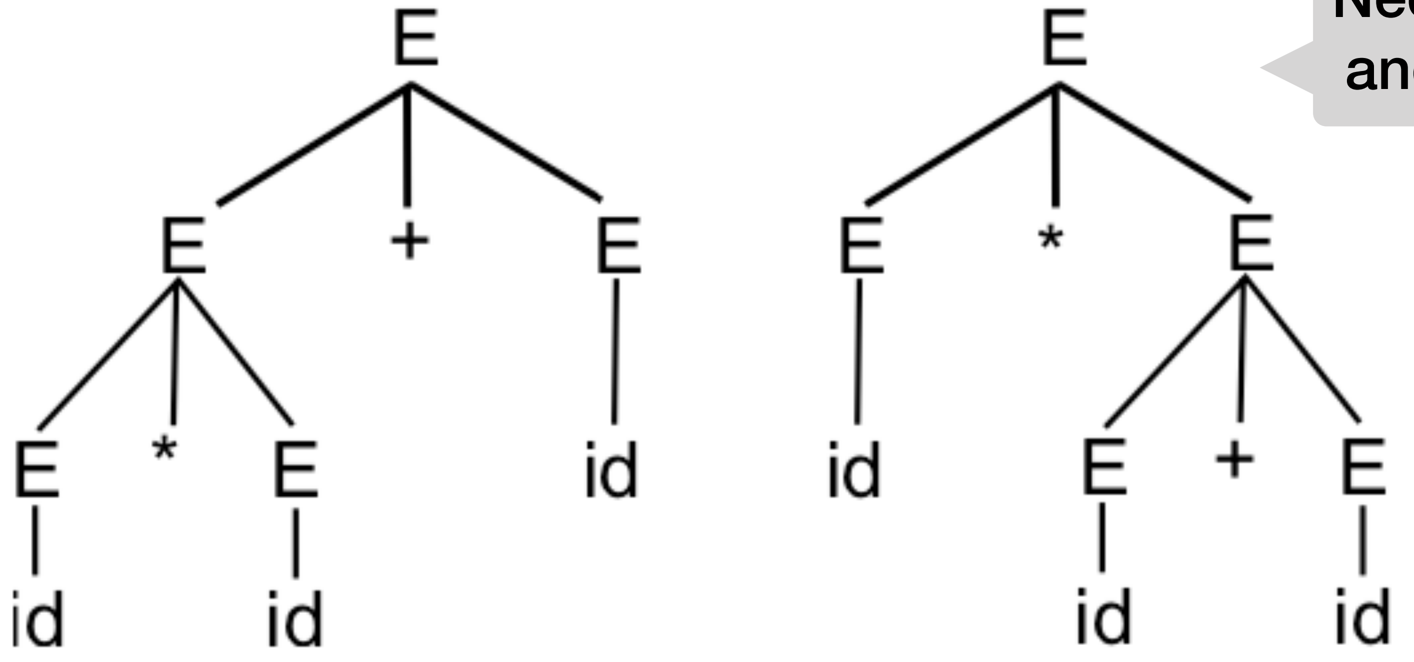
- Consider this grammar:

$EXPR \rightarrow E * E$

$\mid E+E \mid (E)$

$\mid id$

- Now, this string *id*id+id* has two parse trees!



Need Precedence and Associativity

TODOs by next lecture

- Hw2 is out. Please start early!
- Come to the discussion session if you have questions