Lecture 8
Wednesday January 29

- Quizzes will be on Wednesdays
- Office Hours this Friday: 2:30pm

Quiz 1

Context-Free Grammar (CFG): Exercise (1)

Is the following CFG ambiguous?

$$Expr \rightarrow Expr \ + \ Expr \mid Expr \ * \ Expr \mid (\ Expr \ ) \mid a$$

Context-Free Grammar (CFG): Exercise (2.1)

Is the following CFG ambiguous?

```
Statement → if Expr then Statement
          | if Expr then Statement else Statement
          | Assignment
          ...
```

if Expr1 then if Expr2 then Assignment1 else Assignment2

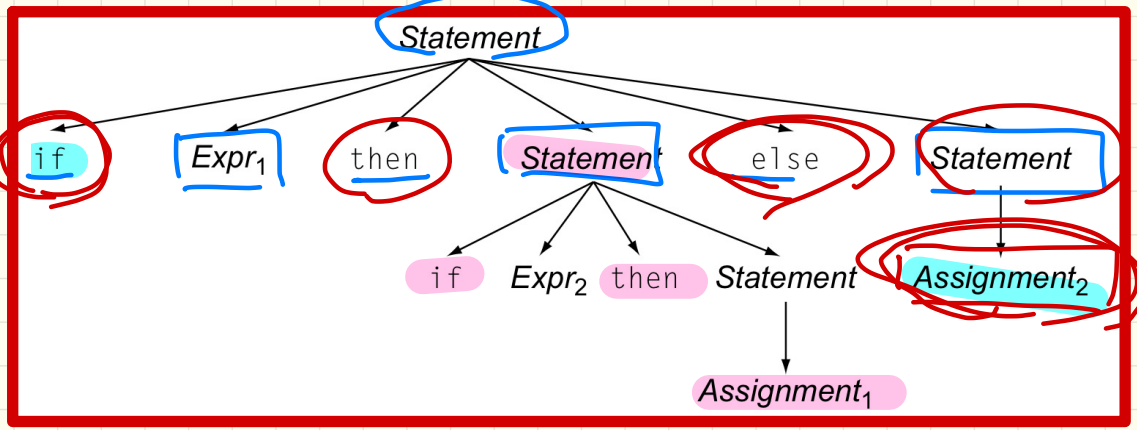
Context-Free Grammar (CFG): Exercise (2.2.1)

Is the following CFG ambiguous?

```
Statement  →  if Expr then Statement
           |  if Expr then Statement else Statement
           |  Assignment
           ...
```

Example:

if *Expr1* **then if** *Expr2* **then** *Assignment1* **else** *Assignment2*



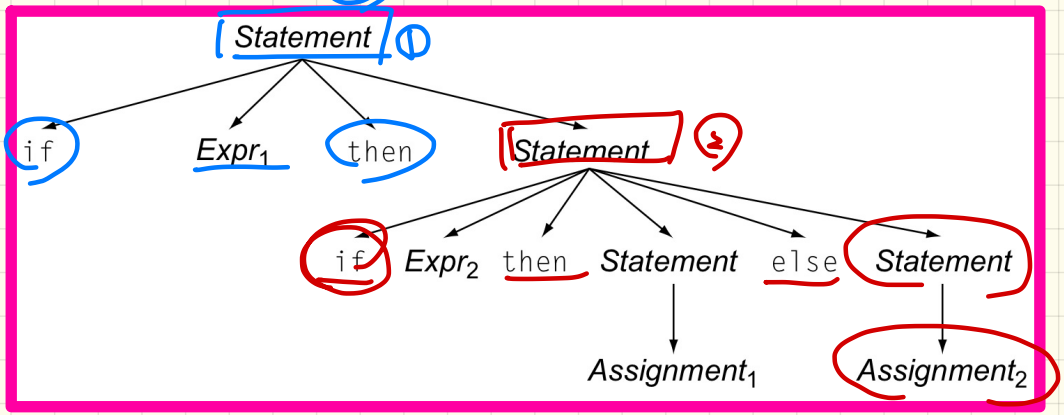
Context-Free Grammar (CFG): Exercise (2.2.2)

Is the following CFG ambiguous?

Statement → ① `if` *Expr* `then` *Statement*
| ② `if` *Expr* `then` *Statement* `else` *Statement*
| *Assignment*
...

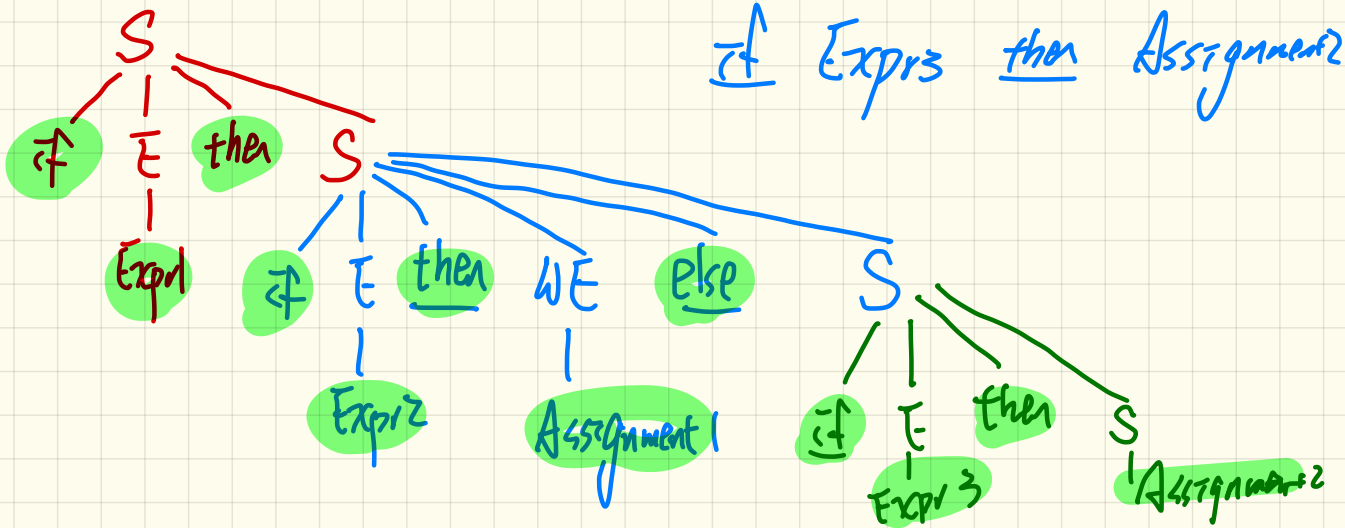
Example:

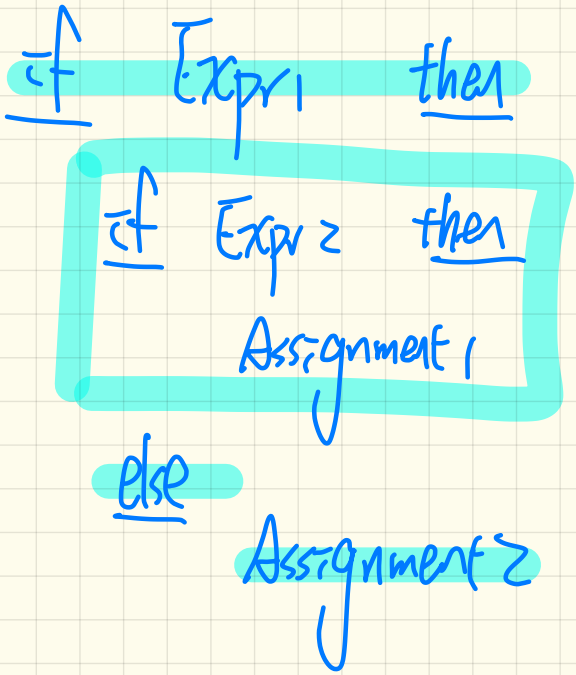
if *Expr1* **then** **if** *Expr2* **then** *Assignment1* **else** *Assignment2*

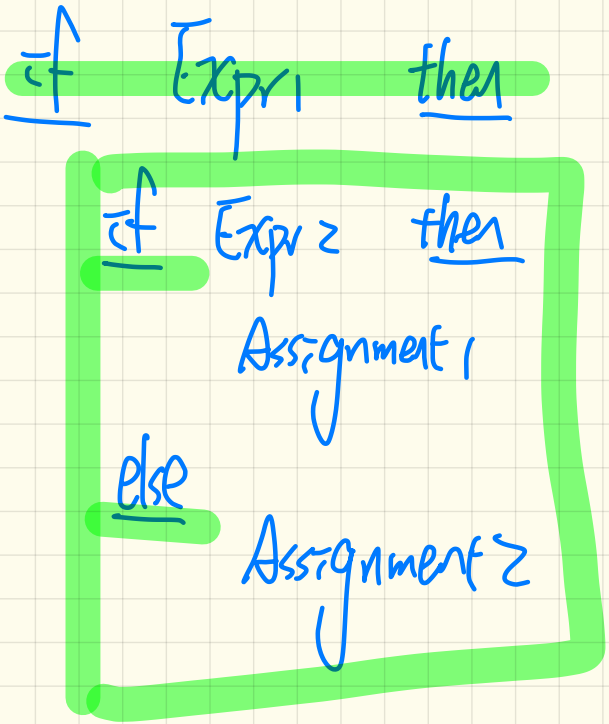


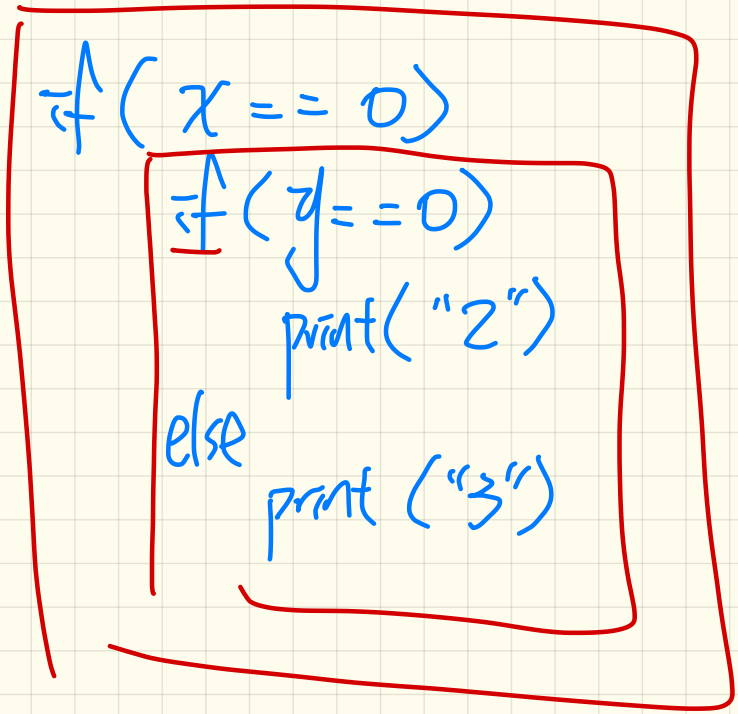
Statement (S)	→	`if` *Expr* (E) `then` *Statement*
	|	`if` *Expr* `then` *WithElse* `else` *Statement*
	|	*Assignment*
WithElse (WE)	→	`if` *Expr* `then` *WithElse* `else` *WithElse*
	|	*Assignment*

① if Expr1 then if Expr2 then Assignment1 else if Expr3 then Assignment2



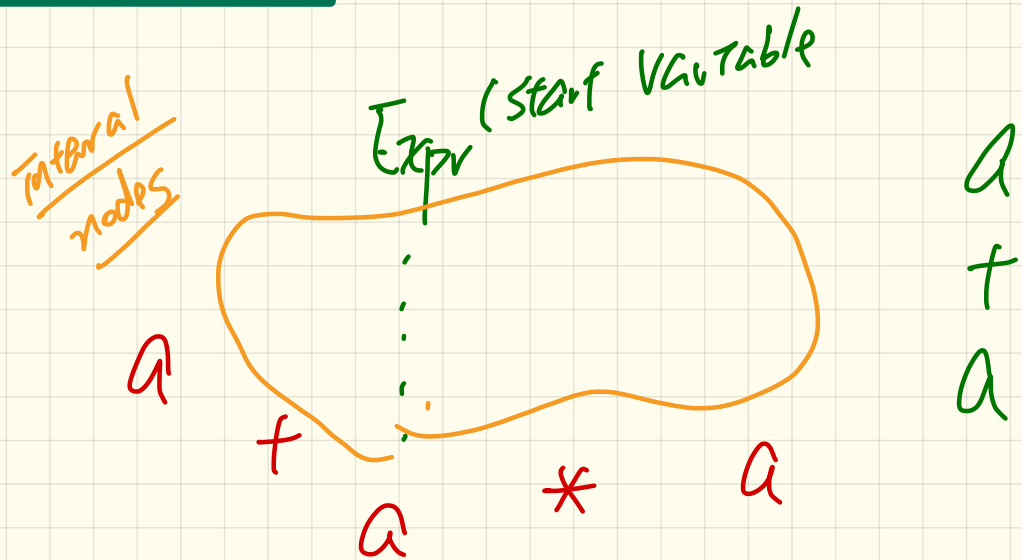

if Expr1 then
if Expr2 then
Assignment1
else
Assignment2


if Expr1 then
if Expr2 then
Assignment1
else
Assignment2

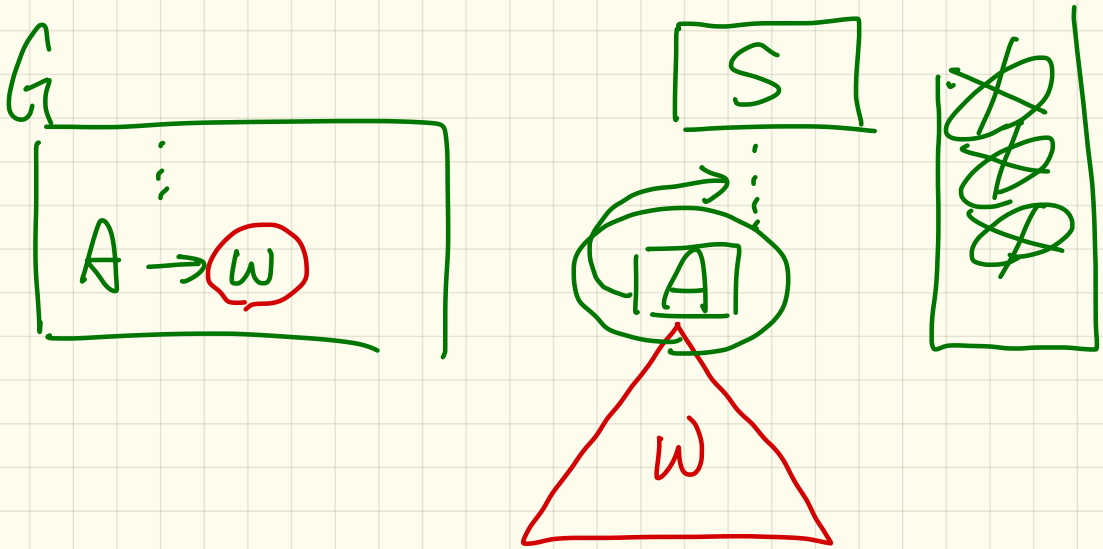

if (x == 0)
if (y == 0)
print("2")
else
print ("3")

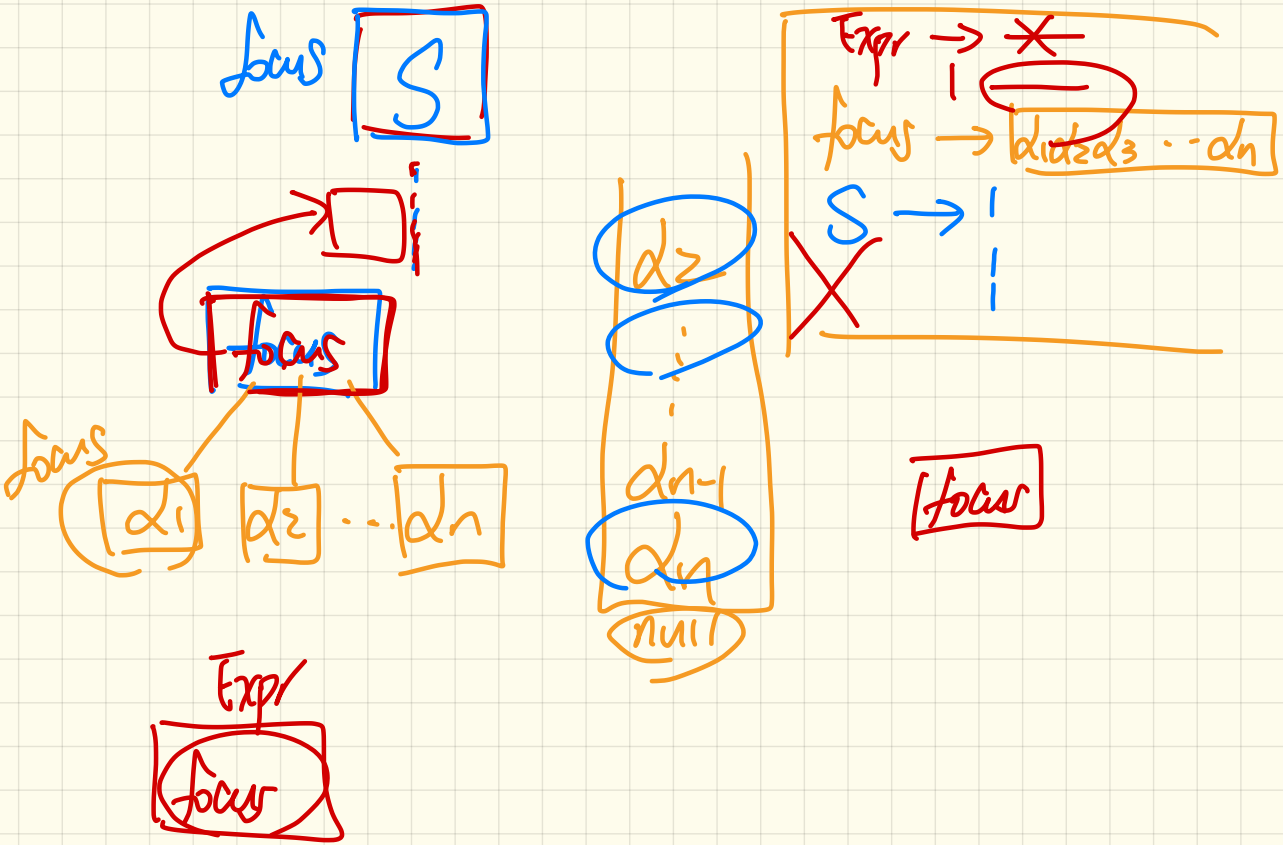
Expr	→	*Expr* + *Term*
	|	*Term*
Term	→	*Term* * *Factor*
	|	*Factor*
Factor	→	(*Expr*)
	|	a

a + a * a



$w \in (V \cup \Sigma)^*$ $A \in V$ $A \Rightarrow w \in R$




focus
S
focus
focus
α1
α2
α3
αn
Expr
focus
α2
αn-1
αn
null
Expr → *
focus → α1α2α3 ··αn
S →
focus

Top-Down Parsing: Algorithm

backtrack ≜ pop *focus*.children; *focus* := *focus*.parent; *focus*.resetChildren

```
ALGORITHM: TDParse
  INPUT: CFG G = (V, Σ, R, S)
  OUTPUT: Root of a Parse Tree or Syntax Error
PROCEDURE:
  root := a new node for the start symbol S
  focus := root
  initialize an empty stack trace
  trace.push(null)
  word := NextWord()
  while (true):
    if focus ∈ V then
      if ∃ unvisited rule focus → β1 β2 ... βn ∈ R then
        create β1, β2 ... βn as children of focus
        trace.push(βn βn−1 ... β2)
        focus := β1
      else
        if focus = S then report syntax error
        else backtrack
        end
      end
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then  return root
    else backtrack
    end
```

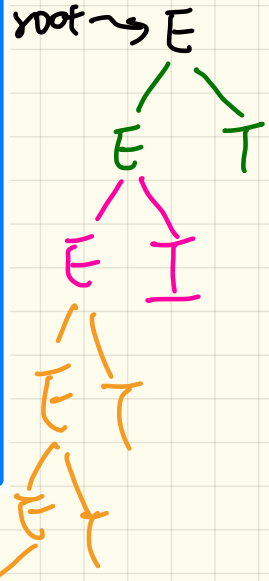
Top-Down Parsing: Discovering Leftmost Derivations (1)

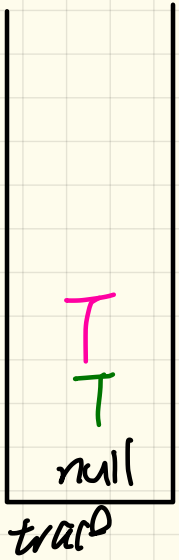
backtrack ≜ pop *focus*.children; *focus* := *focus*.parent; *focus*.resetChildren

Parse: a + a * a

```
ALGORITHM: TDParse
  INPUT: CFG G = (V, Σ, R, S)
  OUTPUT: Root of a Parse Tree or Syntax Error
PROCEDURE:
  root := a new node for the start symbol S
  focus := root
  initialize an empty stack trace
  trace.push(null)
  word := NextWord()
  while (true):
    if focus ∈ V then
      if ∃ unvisited rule focus → β1 β2 ... βn ∈ R then
        create β1, β2 ... βn as children of focus
        trace.push(βn βn−1 ... β2)
        focus := β1
      else
        if focus = S then report syntax error
        else backtrack
        end
      end
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then return root
    else backtrack
    end
```

Expr E	→	Expr + Term
	|	Term
Term T	→	Term * Factor
	|	Factor
Factor F	→	(Expr)
	|	a





word : a

focus : E E E