Lecture 15 - Nov. 3

Syntactic Analysis

Identifying Derivations: TDP vs. BUP
Top-Down Parsing: Algorithm
Left-Recursive CFG

Announcements

- **Assignment 2** released

- **Project Milestone 1** next week
  + Source project due at 11:59 PM on Tuesday
  + A simple readme.txt file explains how to run your tool: e.g.,
    java -jar compiler.jar prog.txt test.txt
    (and where to find the output HTML file)
  + Example files you supplied are supposed to work automatically
  + Jackie will share his screen to build, run, and explore your code.

- **Visitor Pattern** source code: Type Casting

Project: Milestone 1

Milestone 1: Show 3 Example Runs [**1%**]

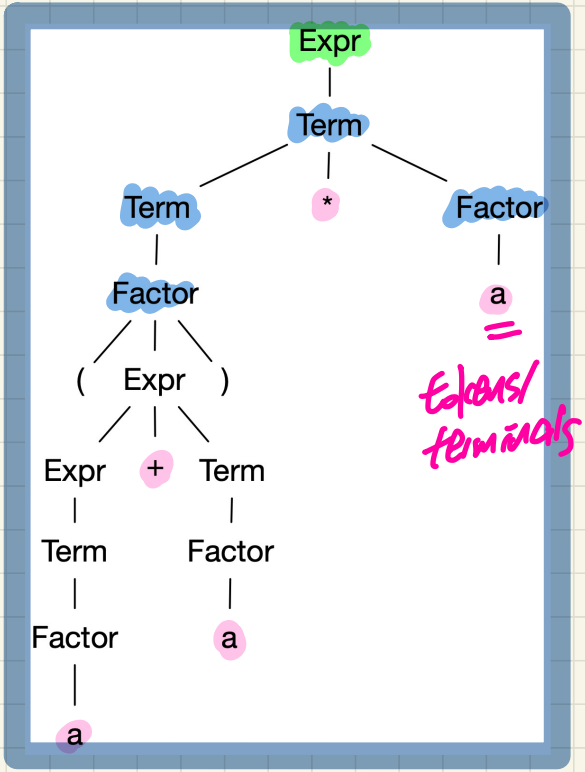
- On the **week of November 7** (about 3 weeks after the project is released), your team is required to meet with Jackie and demonstrate:
  - 3 example runs of your compiler. Each example run consists of the input files and the automatically generated output files.
  - Your example input files should cover (some of the) basic programming features (written in sytax of your own design):
    - ◇ class/module declarations
    - ◇ variable declarations
    - ◇ variable assignments
    - ◇ variable references (i.e., referring to declared variables in expressions)
    - ◇ arithmetic, relational, and logical expressions
    - ◇ conditionals
  - The corresponding produced outputs should cover **at least one** control-flow coverage criterion and **at least one** data-flow coverage criterion.
- **In this meeting, Jackie may suggest specific tasks that your team should complete and will be included in the evaluation of Milestone 2.**

Discovering Derivations

Input Grammar G

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

AST: (a + a) * a

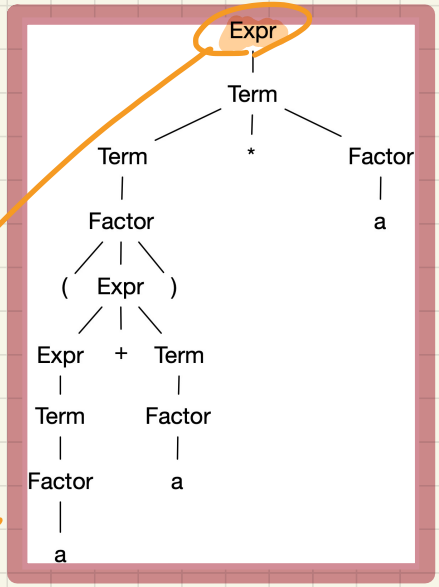


Discovering Derivations: Top-Down vs. Bottom-Up

Input Grammar G

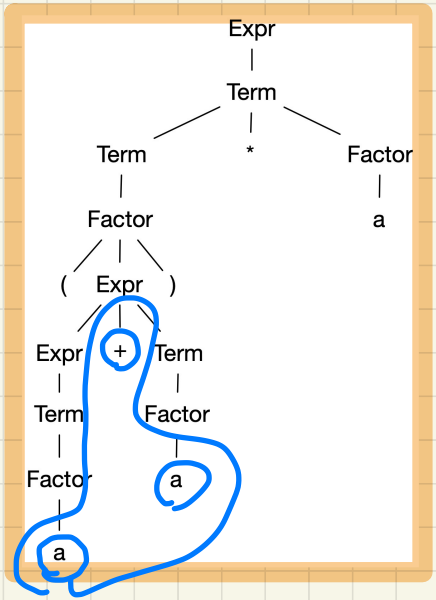
Expr	→	*Expr* + *Term*
	|	*Term*
Term	→	*Term* * *Factor*
	|	*Factor*
Factor	→	(*Expr*)
	|	a

TDP: (a + a) * a



possibility of backtrack.

BUP: (a + a) * a



Top-Down Parsing: Algorithm

```
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
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then return root
    else backtrack
```

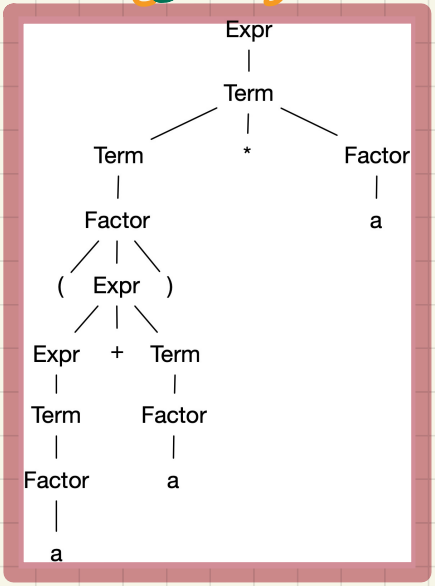
backtrack ≜ pop *focus*.siblings; *focus* := *focus*.parent; *focus*.resetChildren

transforming a linear input token seq. into a non-linear AST.

Input Grammar G

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

TDP: (a + a) * a



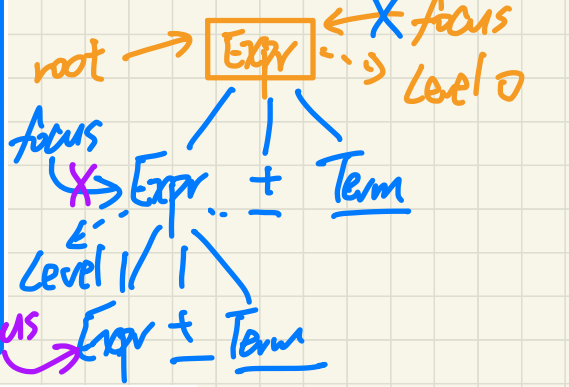
Top-Down Parsing: Discovering Leftmost Derivations (1)

```
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
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then return root
    else backtrack
```

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

Parse: a + a * a

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



word: "a"

non-terminating

Left-Recursions (LRs): Direct vs. Indirect

Direct Left-Recursions:

$$
\begin{array}{lcl}
Expr & \rightarrow & Expr + Term \\
 & | & Term \\
Term & \rightarrow & Term * Factor \\
 & | & Factor \\
Factor & \rightarrow & (Expr) \\
 & | & a
\end{array}
$$

$$
\begin{array}{lcl}
Expr & \rightarrow & Expr + Term \\
 & | & Expr - Term \\
 & | & Term \\
Term & \rightarrow & Term * Factor \\
 & | & Term / Factor \\
 & | & Factor
\end{array}
$$

Indirect Left-Recursions:

$$
\begin{array}{lcl}
A & \rightarrow & Br \\
B & \rightarrow & Cd \\
C & \rightarrow & At
\end{array}
$$

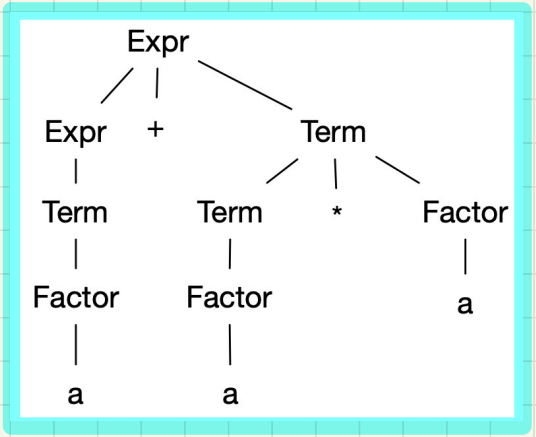
$$
\begin{array}{lcl}
A & \rightarrow & Ba \mid b \\
B & \rightarrow & Cd \mid e \\
C & \rightarrow & Df \mid g \\
D & \rightarrow & f \mid Aa \mid Cg
\end{array}
$$

CFGs: Left-Recursive vs. Right-Recursive

Example: a + a * a

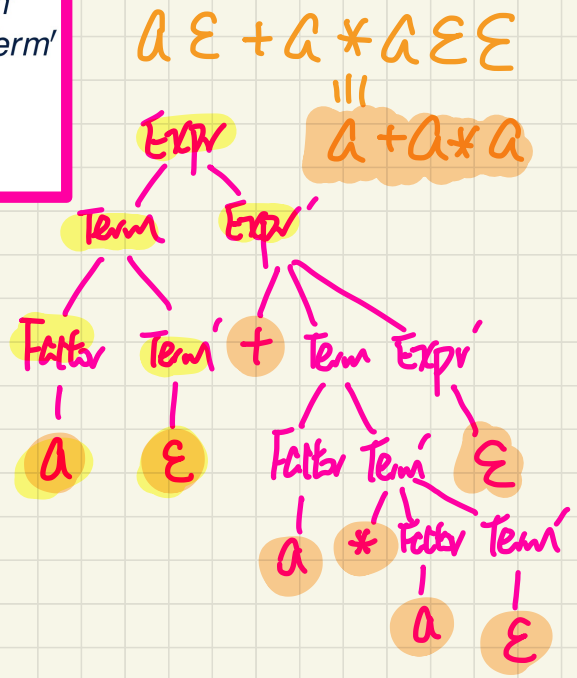
CFG with Left Recursions

$$
\begin{aligned}
Expr &\rightarrow Expr + Term \\
&\mid Term \\
Term &\rightarrow Term * Factor \\
&\mid Factor \\
Factor &\rightarrow (Expr) \\
&\mid a
\end{aligned}
$$



CFG with Right Recursions

$$
\begin{aligned}
Expr &\rightarrow Term\ \ Expr' \\
Expr' &\rightarrow + \ Term\ \ Expr' \\
&\mid \epsilon \\
Term &\rightarrow Factor\ \ Term' \\
Term' &\rightarrow * \ Factor\ \ Term' \\
&\mid \epsilon \\
Factor &\rightarrow (Expr) \\
&\mid a
\end{aligned}
$$



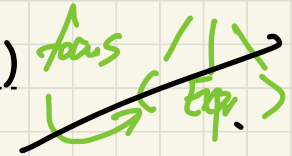
Top-Down Parsing: Discovering Leftmost Derivations (2)

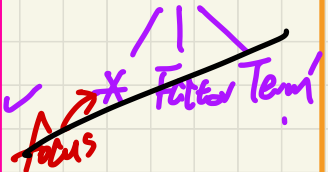
```
ALGORITHM: TDParse
  INPUT: CFG G = (V, Σ, R, S)
  OUTPUT: Root of a Parse Tree or Syntax Error
PROCEDURE:
  root := a new node for the start symbol S
  focus := root
  initialize an empty stack trace
  trace.push(null)
  word := NextWord()
  while (true):
    if focus ∈ V then
      if ∃ unvisited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

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

Parse: a + a * a

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





word: "a" "+"

doesn't match "+" but ϵ has no effect on concat ⇒ the next symbol will match

Top-Down Parsing: Discovering Leftmost Derivations (3)

```
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
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then return root
    else backtrack
```

Parse: (a + a) * a

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

Exercise!

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