Lecture 10
Wednesday February 5

Removing Left-Recursions (1)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G acyclic ∧ with no ε-productions
4    OUTPUT: G' s.t. G' ≡ G, G' has no
5            indirect & direct left-recursions
6  PROCEDURE:
7    impose an order on V: ⟨(A1, A2, ..., An)⟩
8    for i: 1 .. n:
9      for j: 1 .. i−1:
10       if ∃ Ai → Ajγ ∈ R ∧ Aj → δ1 | δ2 | ... | δm ∈ R then
11         replace Ai → Ajγ with Ai → δ1γ | δ2γ | ... | δmγ
12       end
13     for Ai → Aiα | β ∈ R:
14       replace it with: Ai → βA', A' → αA' | ε
```

not executed at all

Directly Left-Recursive CFG:

1	Expr	→	Expr + Term	
		|	Term	
2	Term	→	Term * Factor	
		|	Factor	
3	Factor	→	(Expr)	
		|	a	

$A_2 \to A_1$

I | J
1 | —

F
F | 2 | 1 | Term → Expr
F | 3 | 1 | E → E...
T | 3 | 2 | T → T

I
1 | Expr → Expr
2 | T → T
3 | F

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G acyclic ∧ with no ϵ-productions
4    OUTPUT: G′ s.t. G′ ≡ G, G′ has no
5            indirect & direct left-recursions
6  PROCEDURE:
7    impose an order on V: ⟨(A1, A2, ..., An)⟩
8    for i: 1 .. n:
9      for j: 1 .. i − 1:
10       if ∃ Ai → Ajγ ∈ R ∧ Aj → δ1 | δ2 | ... | δm ∈ R then
11         replace Ai → Ajγ with Ai → δ1γ | δ2γ | ... | δmγ
12     end
13     for Ai → Aiα | β ∈ R:
14       replace it with: Ai → βA′, A′ → αA′ | ϵ
```

D. LR

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

α

β

Expr → Term Expr′

Expr′ → + Term Expr′ | ϵ

Example.

I J

I

|

E → E

Removing Left-Recursions (2)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G acyclic ∧ with no ε-productions
4    OUTPUT: G′ s.t. G′ ≡ G, G′ has no
5            indirect & direct left-recursions
6  PROCEDURE:
7    impose an order on V: ⟨⟨A_1, A_2, ..., A_n⟩⟩
8    for i: 1 .. n:
9      for j: 1 .. i − 1:
10       if ∃ A_i → A_j γ ∈ R ∧ A_j → δ_1 | δ_2 | ... | δ_m ∈ R then
11         replace A_i → A_j γ with A_i → δ_1 γ | δ_2 γ | ... | δ_m γ
12       end
13     for A_i → A_i α | β ∈ R:
14       replace it with: A_i → β A′, A′ → α A′ | ε
```

Directly Left-Recursive CFG:

Expr	→	Expr + Term
	|	Expr − Term
	|	Term
Term	→	Term * Factor
	|	Term / Factor
	|	Factor

Exercise.

Removing Left-Recursions (3)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G acyclic ∧ with no ϵ-productions
4    OUTPUT: G' s.t. G' ≡ G, G' has no
5            indirect & direct left-recursions
6  PROCEDURE:
7    impose an order on V: ⟨A1, A2, ..., An⟩
8    for i: 1 .. n:
9      for j: 1 .. i − 1:
10       if ∃ Ai → Ajγ ∈ R ∧ Aj → δ1 | δ2 | ... | δm ∈ R then
11         replace Ai → Ajγ with Ai → δ1γ | δ2γ | ... | δmγ
12       end
13     for Ai → Aα | β ∈ R:
14       replace it with: Ai → βA', A' → αA' | ϵ
```

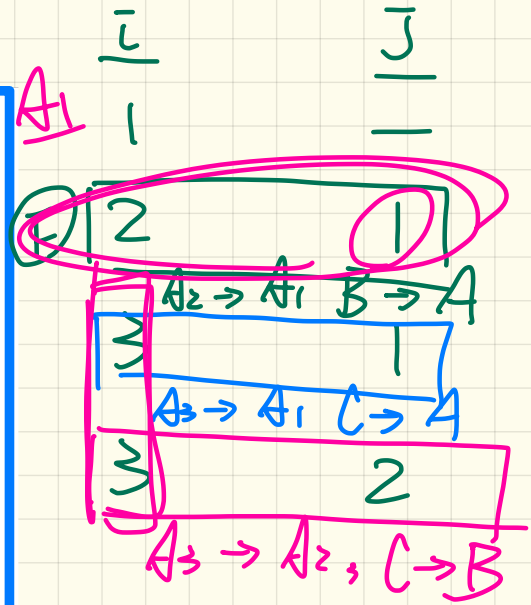
Expect: direct LR resulted

Indirectly Left-Recursive CFG:

1	A	→	Br
2	B	→	Cd
3	C	→	At

C → Brt → Cdrt

C → C'
C' → drt C' | ϵ



Removing Left-Recursions (4)

```
1   ALGORITHM: RemoveLR
2     INPUT: CFG G = (V, Σ, R, S)
3     ASSUME: G acyclic ∧ with no ϵ-productions
4     OUTPUT: G′ s.t. G′ ≡ G, G′ has no
5             indirect & direct left-recursions
6   PROCEDURE:
7     impose an order on V: ⟨(A1, A2, ..., An)⟩
8     for i: 1 .. n:
9       for j: 1 .. i − 1:
10        if ∃ Ai → Ajγ ∈ R ∧ Aj → δ1 | δ2 | ... | δm ∈ R then
11          replace Ai → Ajγ with Ai → δ1γ | δ2γ | ... | δmγ
12      end
13      for Ai → Aiα | β ∈ R:
14        replace it with: Ai → βA′, A′ → αA′ | ϵ
```

assert L.I. $i=3$

Indirectly Left-Recursive CFG:

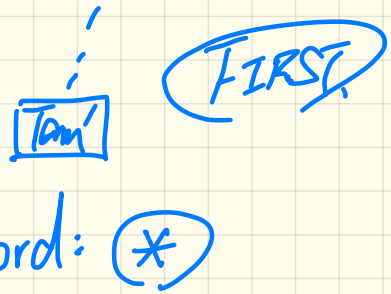
A	$\rightarrow$	Ba	$\mid$	b	
B	$\rightarrow$	Cd	$\mid$	e	
C	$\rightarrow$	Df	$\mid$	g	
D	$\rightarrow$	f	$\mid$	Aa	$\mid$ Cg

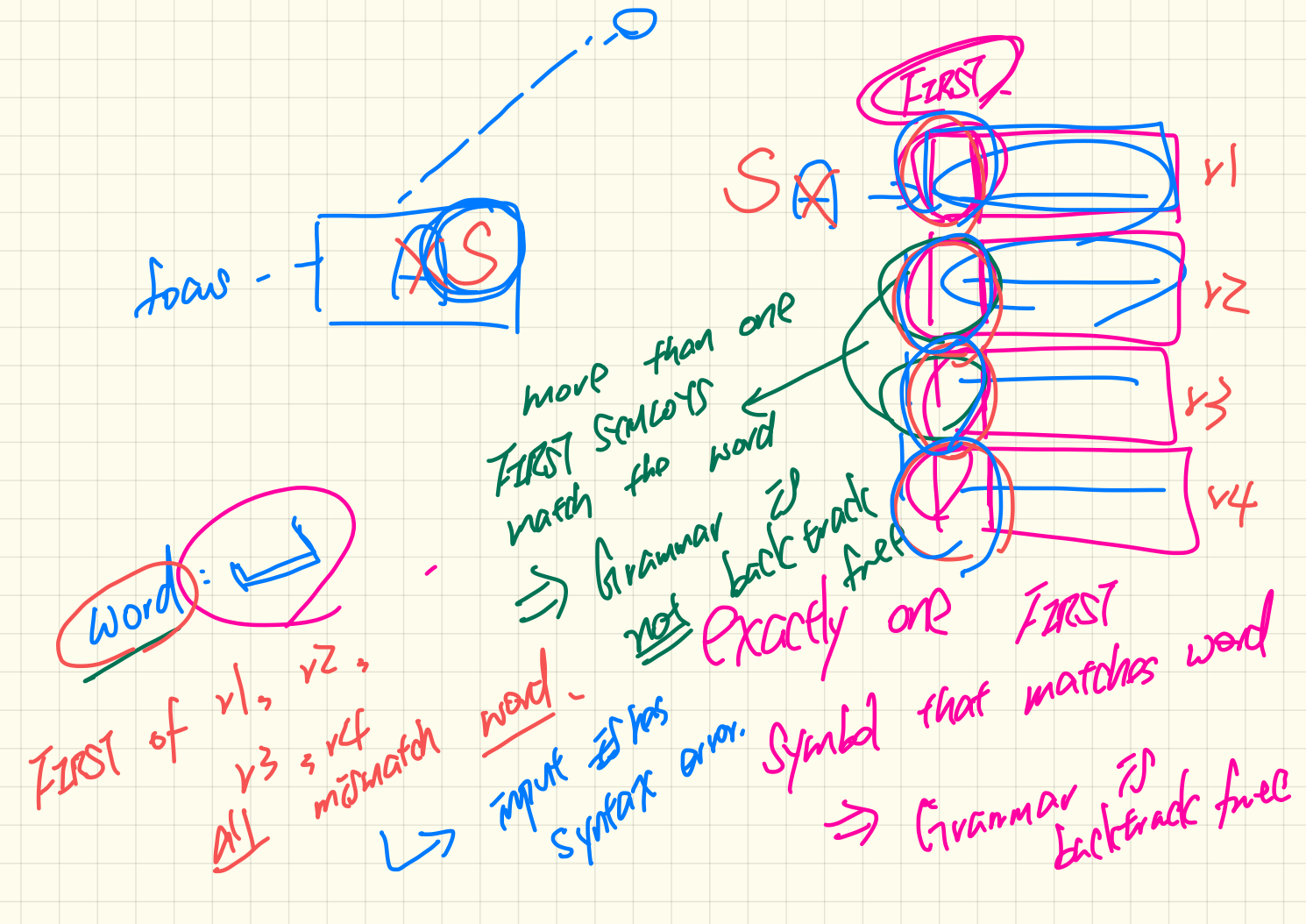
Top-Down Parsing: Algorithm

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
```

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr′*
2	*Expr′*	→	+ *Term Expr′*
3		|	- *Term Expr′*
4		|	ϵ
5	*Term*	→	*Factor Term′*
6	*Term′*	→	× *Factor Term′*
7		|	÷ *Factor Term′*
8		|	ϵ
9	*Factor*	→	(*Expr*)
10		|	num
11		|	name




FIRST
S
v1
v2
v3
v4
more than one FIRST sometimes match the word
⇒ Grammar is not backtrack free
not exactly one FIRST symbol that matches word
⇒ Grammar is backtrack free
word
FIRST of v1, v2, v3, v4 all mismatch word
input is has syntax error.

FIRST Set

$$\textsc{First}(\alpha) = \begin{cases} \{\alpha\} & \text{if } \alpha \in T \\ \{w \mid w \in \Sigma^* \wedge \alpha \overset{*}{\Rightarrow} w\beta \wedge \beta \in (V \cup \Sigma)^*\} & \text{if } \alpha \in V \end{cases}$$

first word

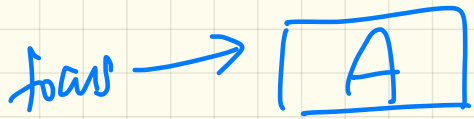
Right-Recursive CFG:

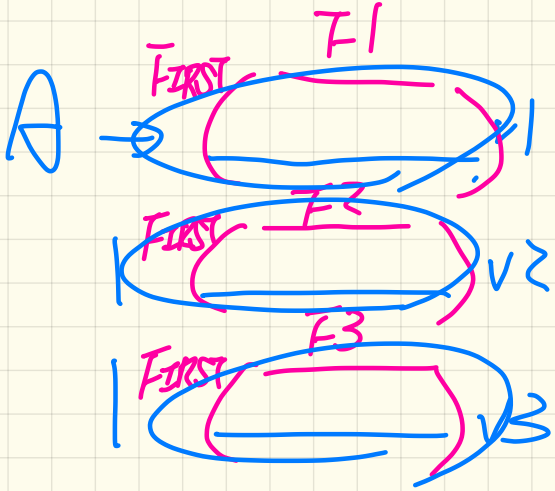
0	*Goal*	→	*Expr*	6	*Term′*	→	× *Factor Term′*
1	*Expr*	→	*Term Expr′*	7		|	÷ *Factor Term′*
2	*Expr′*	→	+ *Term Expr′*	8		|	ϵ
3		|	- *Term Expr′*	9	*Factor*	→	(*Expr*)
4		|	ϵ	10		|	num
5	*Term*	→	*Factor Term′*	11		|	name

1, 2, 3 alon

	num	name	+	-	×	÷	(	)	eof	ϵ
FIRST	num	name	+	-	x	÷	(	)	eof	ϵ

	Expr	Expr′	Term	Term′	Factor
FIRST	(, name, num	+, -, ϵ	(, name, num	x, ÷, ϵ	(, name, num





word: x

$x \in F_1$

$x \in F_2$

$\Rightarrow$ $x \in F_3$

G is not backtrack free.

$F_1 \cap F_2 \cap F_3 = \emptyset$.