Lecture 16 - Nov. 10

Syntactic Analysis

Removing Left-Recursion from CFG
Computing the FIRST Set

Announcements

- **ProgTest** marks and results released

- **Assignment 2** due next Monday

- **Quiz2** and **Quiz3** papers ready for pick-up on Monday

Removing Left-Recursions: Algorithm

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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 → Aj γ ∈ R ∧ Aj → δ1 | δ2 | ... | δm ∈ R then
11         replace Ai → Aj γ with Ai → δ1 γ | δ2 γ | ... | δm γ
12       end
13     for Ai → Ai α | β ∈ R:
14       replace it with: Ai → β Ai′, Ai′ → α Ai′ | ϵ
```

diff. variables

eliminate indirect left LR

same variable ⇒ direct left recursion

$A \to \epsilon$

ϵ-production

$A_i \to A_i\alpha \mid \beta$

left-recursion

① A_i ⇒ β

② A_i ⇒ $A_i\alpha$ ⇒ $A_i\alpha\alpha$ ⇒ $\beta\alpha\alpha$

right-recursion

$A_i \to \beta A_i'$

$A_i' \to \alpha A_i' \mid \epsilon$

Removing Left-Recursions (1a)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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′i, A′i → αA′i | ϵ
```

P1

P2

i j

1 Expr — X

2 Term 1 Expr X

3 Factor 1 → 2

Expr Term

i, j = 2, 1

Ai → Aj

P1 Term → Expr ? no

↳ do nothing.

P2 Ai Ai α yes

Term → Term * Factor ?

| Factor β

Directly Left-Recursive CFG:

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

Term → Factor Term′

Term′ → * Factor Term′ | ϵ

Removing Left-Recursions (1b)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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_i′, A_i′ → α A_i′ | ϵ
```

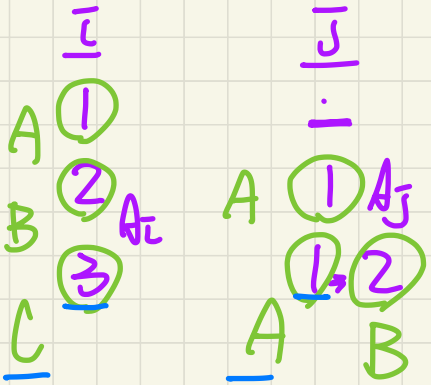
Directly Left-Recursive CFG:

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

Exercise

Removing Left-Recursions (2a)

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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_i', A_i' → α A_i' | ϵ
```



i, j = 2, 1

B → A ? No

↳ do nothing.

Indirectly Left-Recursive CFG:

① A → Br

② B → Cd

③ C → At

C → Brt

i, j = 3, 2

↳ exercise.

i, j = 3, 1

C → At

Ai Aj γ

Aj A → Br δ1

→ C → Brt

Ai δ1 γ

Removing Left-Recursions (2b)

Does the order of variables matter?

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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_i′, A_i′ → α A_i′ | ϵ
```

Indirectly Left-Recursive CFG:

① C → At

② B → Cd

③ A → Br

Exercise!

Removing Left-Recursions (2c)

Exercise

```
1  ALGORITHM: RemoveLR
2    INPUT: CFG G = (V, Σ, R, S)
3    ASSUME: G has 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_i′, A_i′ → α A_i′ | ϵ
```

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

$A \to B_1 B_2 B_3$

B_1 is nullable $\quad B_1 \to b \mid \varepsilon$

B_2, B_3 are nullable $\quad B_2 \to c \mid \varepsilon$

$B_3 \to d \mid \varepsilon$

$B \to C \boxed{A} D$

nullable

$C \to c$

$D \to d$

$A \to a \mid \varepsilon$

↓

$B \to CD$

$\mid CaD$

$A \to a$

$A \to X_1 X_2 \cdots X_{10}$

↳ What if all 10 variables nullable

What if X_2, X_3, X_4 are nullable.

2^0 — 1 ↓ when all variables produce ε

How many versions of A to produce?

2^3

$A \to X_1 \_ \_ \_ X_5 \cdots X_{10}$

X_3 X_4

X_2 X_4

X_2 X_3

⋮

Eliminating epsilon-Productions

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aAA \mid \epsilon \\
B &\rightarrow bBB \mid \epsilon
\end{aligned}
$$

Q: Nullable variables?

↳ $S \rightarrow B \mid A \mid AB$

$A \rightarrow aAA \mid aA \mid a$

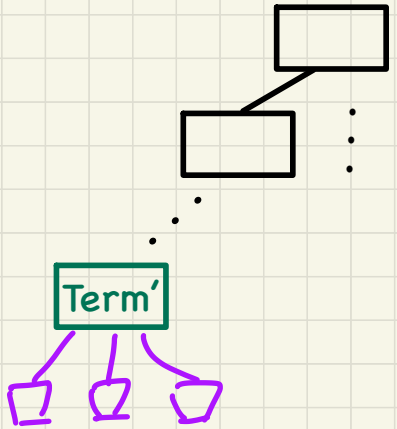
$B \rightarrow bBB \mid bB \mid b$

Top-Down Parsing: Backtrack

```
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

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 Set

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

Right-Recursive CFG:

0	*Goal*	→	*Expr*	6	*Term′*	→	x *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

→ what if: Factor → ε

	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