190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 12

Dr. Öğr. Üyesi İsmail Hakkı Parlak

Parsing (Ayrıştırma)

Top-Down Parsing (Tepeden Alta Ayrıştırma)

Start sembolü ile başlanır ve hedef dize (string) elde edilene kadar üretim kuralları (production rules) uygulanır.

G = ({S, A}, {a, b, c, d}, R, S)

R = {

S -> cAd

A -> ab | a

}

w = cad

w, G içinde geçerli midir?

Tepeden Alta Ayrıştırma

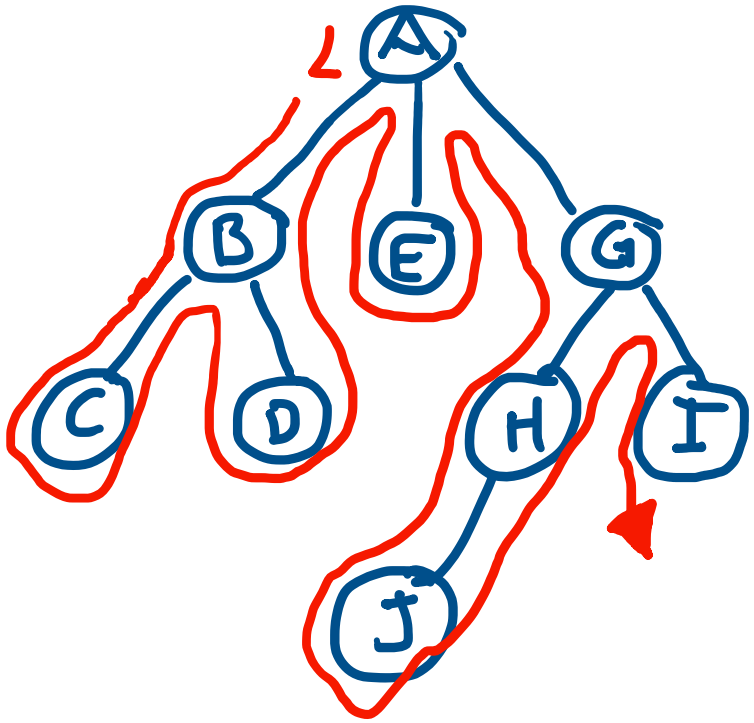
- Tepeden alta ayrıştırmada soldan türetim (left-most derivation) uygulanır.
- Ayrıştırma ağacı (parse tree) pre-order (önden sıralı) şekilde oluşturulur.

```
def pre_order(node):
    if node == None:
        return

    print(node.data)
    pre_order(node.left)
    pre_order(node.right)
```

- Başarısızlık durumunda geri izleme (backtracking) uygulanır.

Pre-order (Önden Sıralı) Gezme



A B C D E G H J I

Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

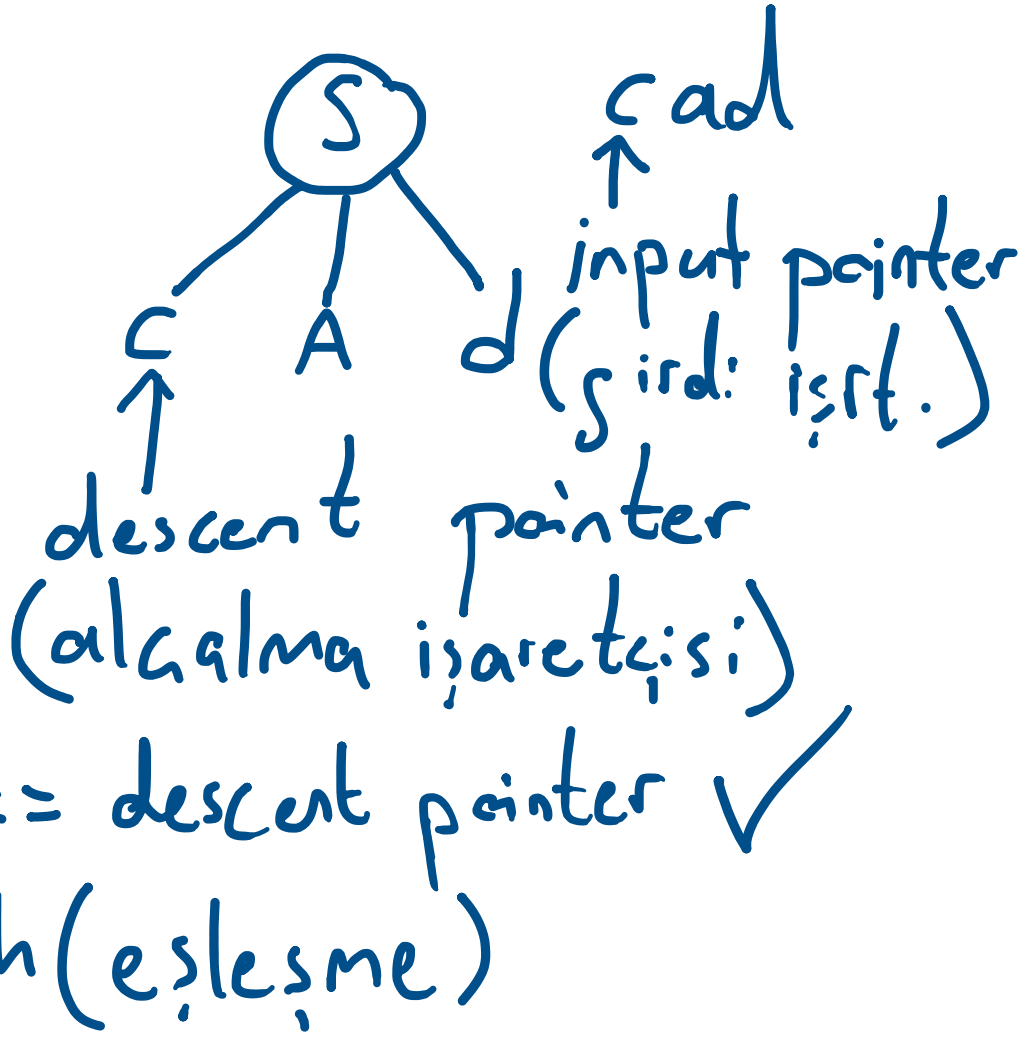
R = {

S -> cAd

A -> ab | a

}

w = cad



Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

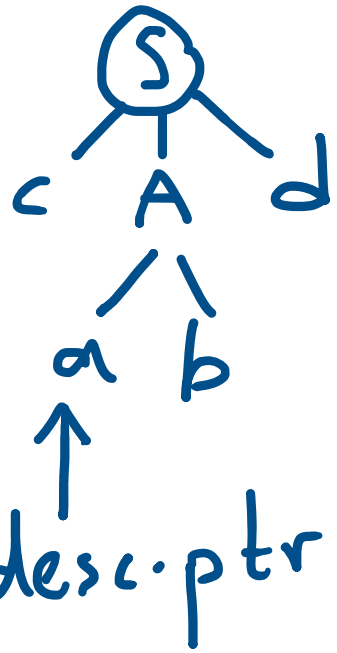
R = {

S -> cAd

A -> ab | a

}

w = cad



cad

↑

input ptr.

inp. ptr == desc. ptr ✓

(eşleşme)

Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

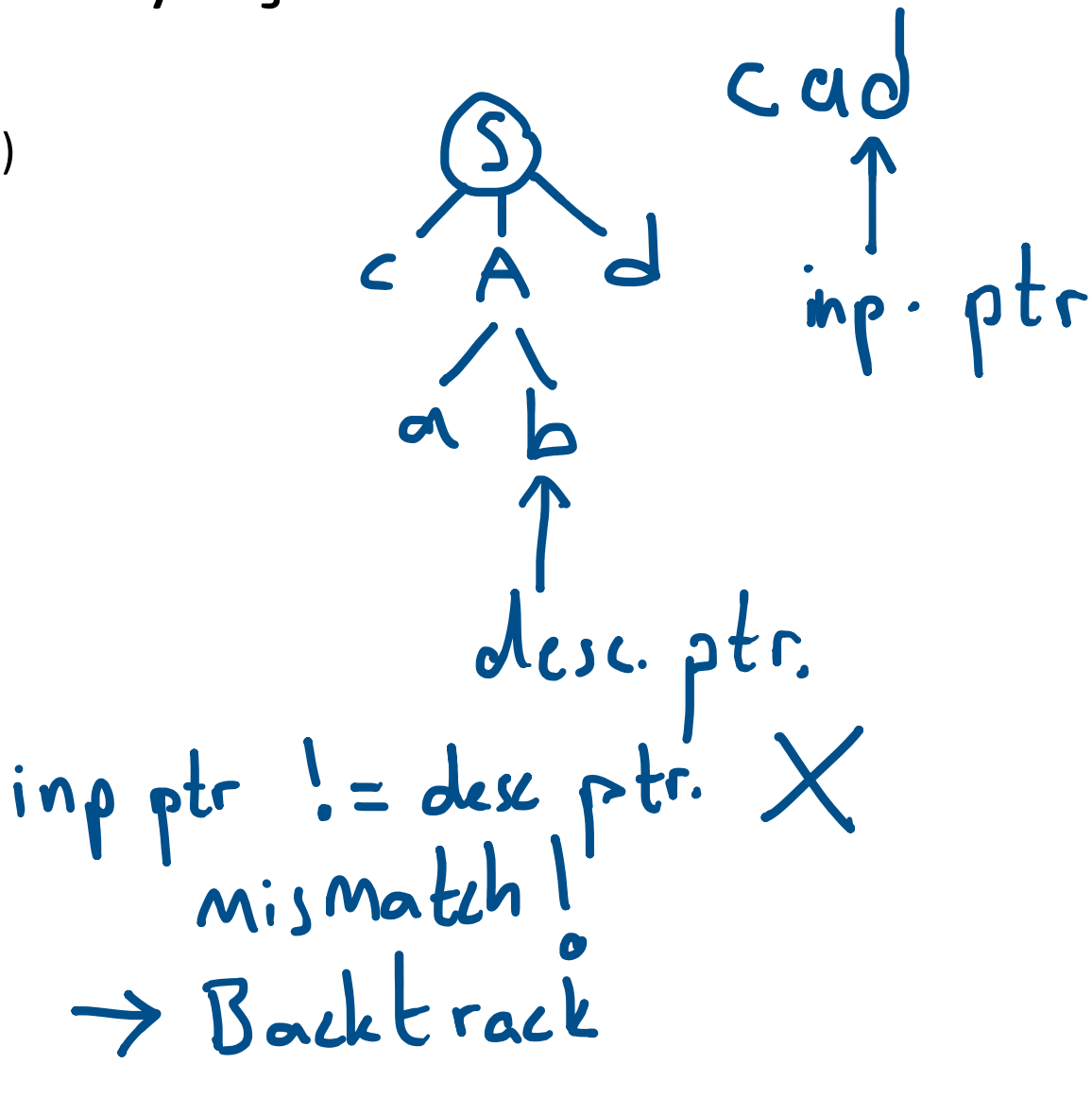
R = {

S -> cAd

A -> ab | a

}

w = cad



Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

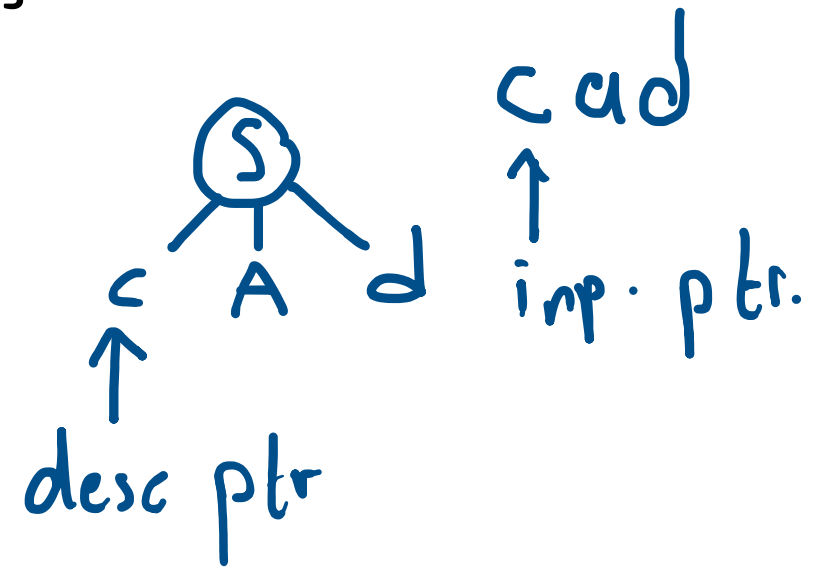
R = {

S -> cAd

A -> ab | a

}

w = cad



Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

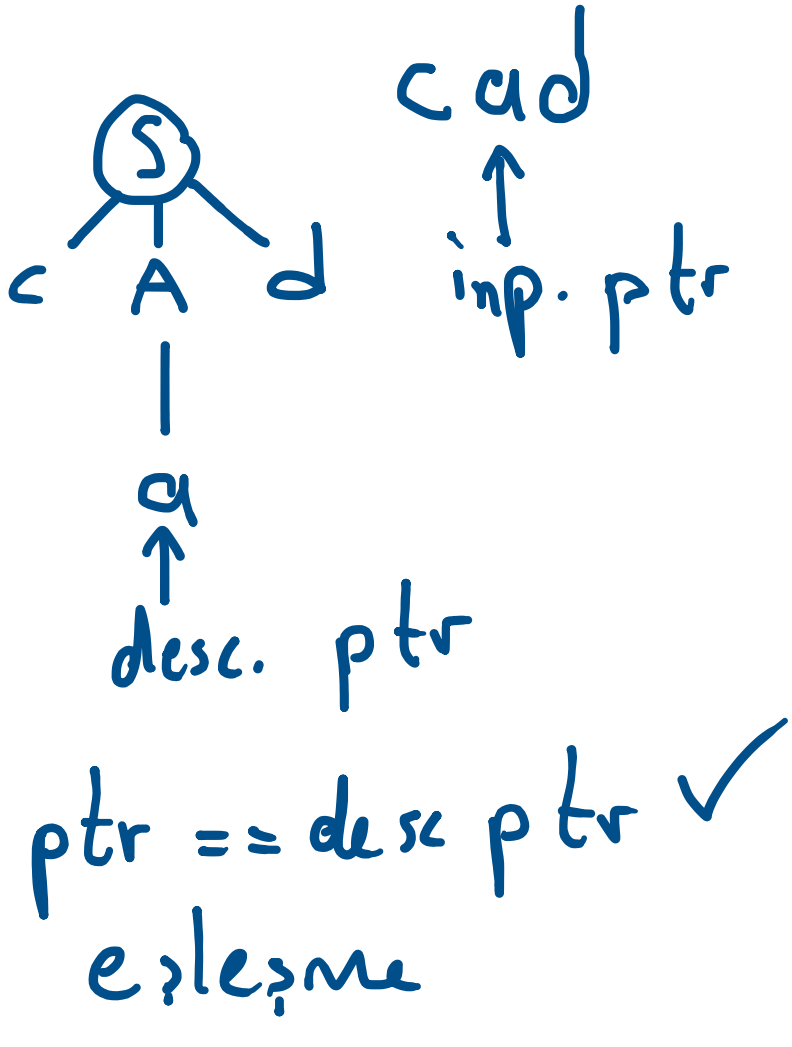
R = {

S -> cAd

A -> ab | a

}

w = cad



Tepeden Alta Ayrıştırma

G = ({S, A}, {a, b, c, d}, R, S)

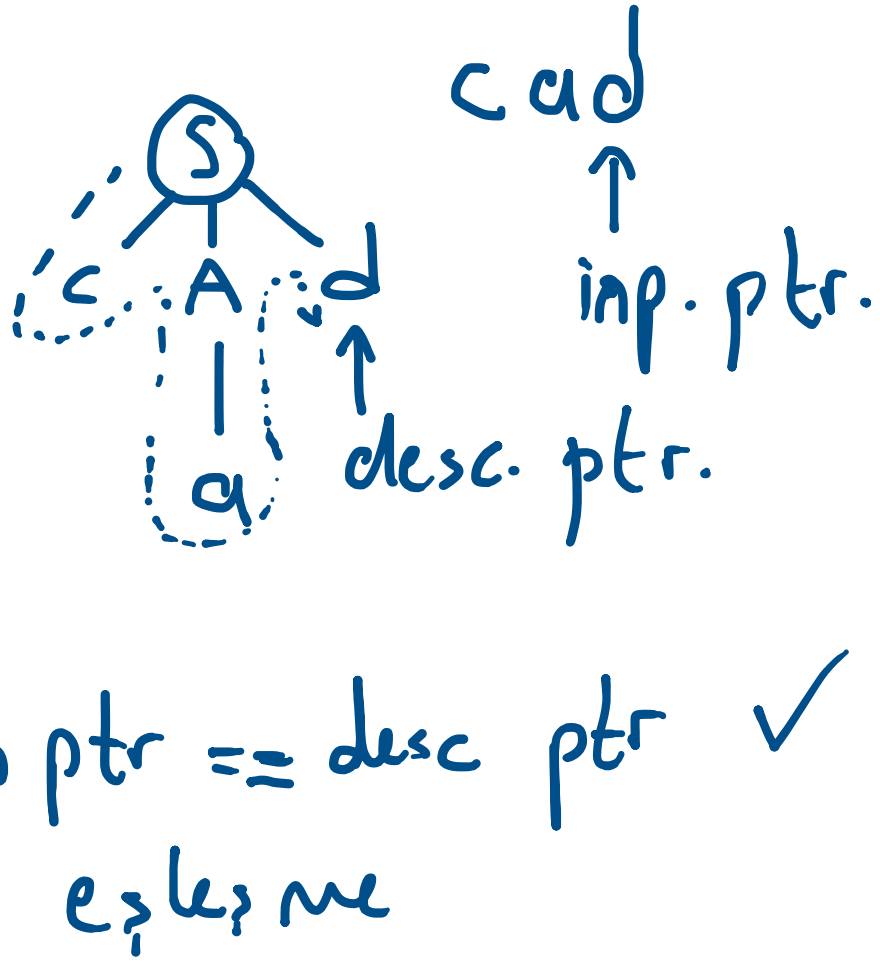
R = {

S -> cAd

A -> ab | a

}

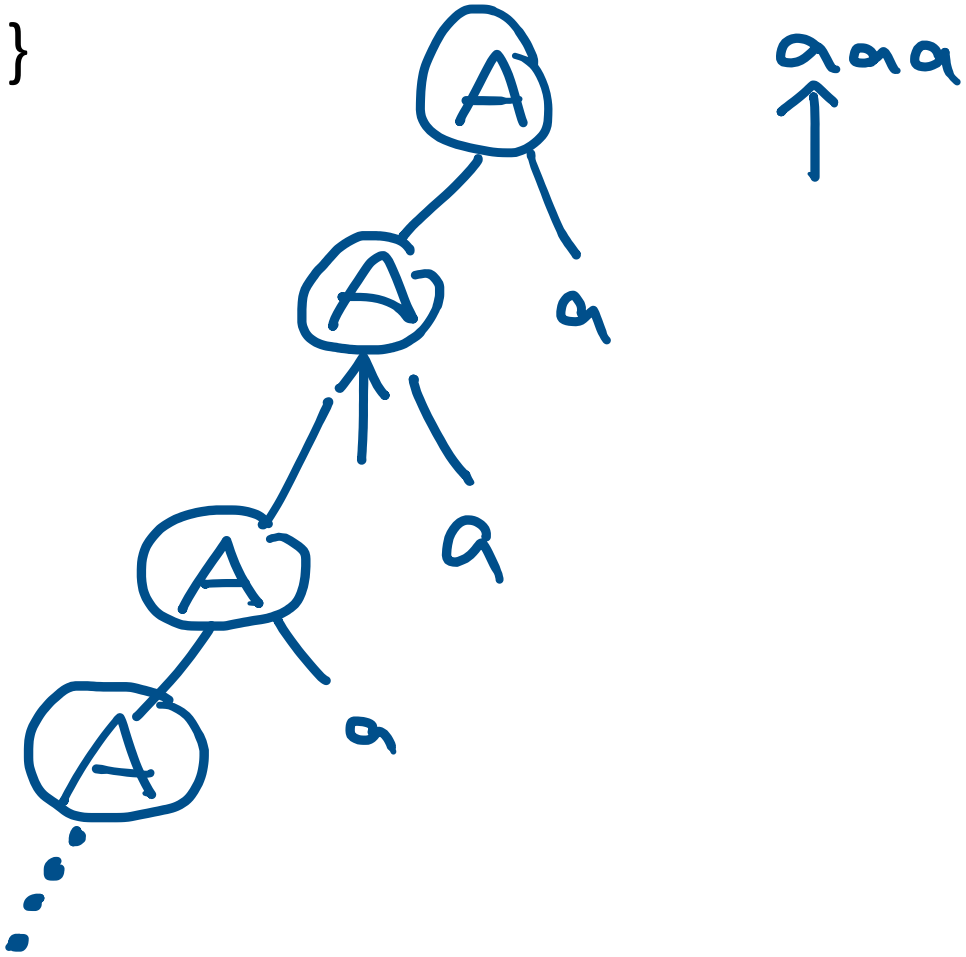
w = cad



Recursive Descent Parser (Öz yineli alçalmalı ayrıştırma)

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

- R = { A -> Aa | ε }
- Hedef aaa



Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

Kurtulacağımız kural:

A -> Aα | β (β, A ile başlamayan herhangi bir terim.)

Kurtaran kurallar:

A -> β A'

A' -> α A' | ε

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

A -> Aα | β

A -> β A'

A' -> α A' | ε

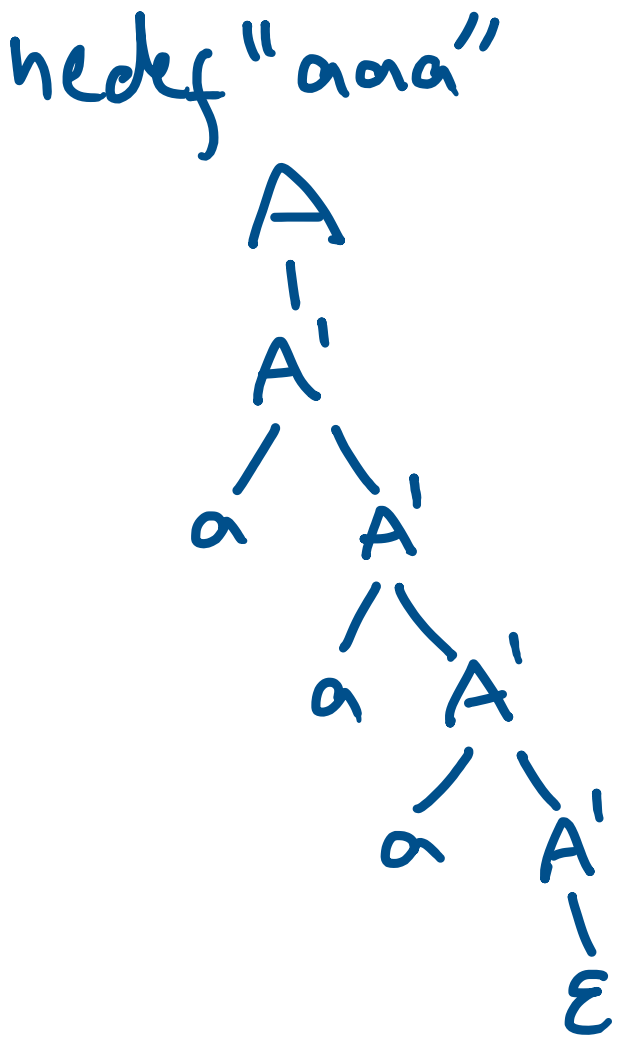
$A \rightarrow A\alpha \mid \varepsilon$

$\alpha \quad \beta$

$A \rightarrow \varepsilon A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)



$$A \rightarrow A\alpha \mid \varepsilon$$

α β

$$A \rightarrow \varepsilon A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

$A \rightarrow A\alpha \mid \beta$ ✗

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$ ✓

Örn:

$S \rightarrow ABC$ ✓

$A \rightarrow Aa \mid Ad \mid b$ ✗

$B \rightarrow Bb \mid e$ ✗

$C \rightarrow Cc \mid g$ ✗

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

$A \rightarrow A\alpha \mid \beta$ ✗

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$ ✓

Örn:

? $A \rightarrow Aa \mid Ad \mid b$

$A \rightarrow Aa \mid b$

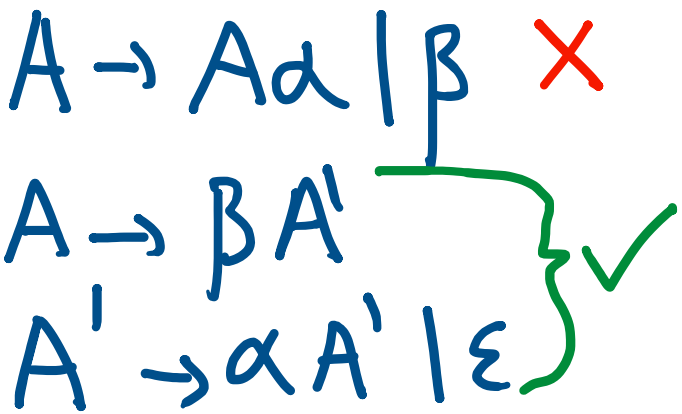
α β

1 {

$A \rightarrow bA'$

$A' \rightarrow aA' \mid \varepsilon$

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)



Örn:

? A → Aa | Ad | b

A → Ad | b

α β

2 { A → bA'
  { A' → dA' | ε

Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)

$A \to A\alpha \mid \beta$ ✗

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$ ✓

$A \to Aa \mid Ad \mid b$

1)
$A \to bA'$

$A' \to aA' \mid \varepsilon$

2)
$A \to bA'$

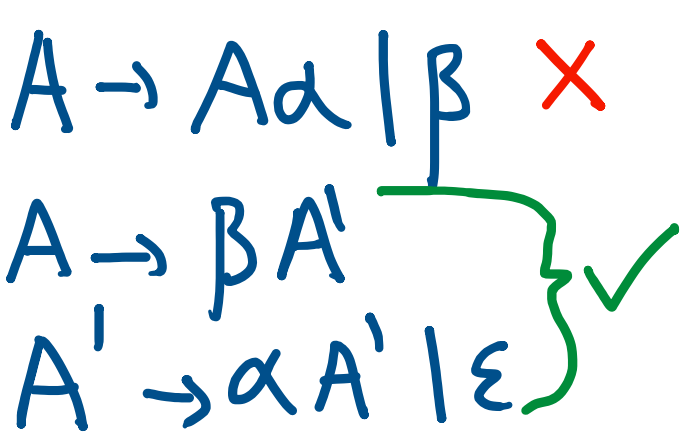
$A' \to dA' \mid \varepsilon$

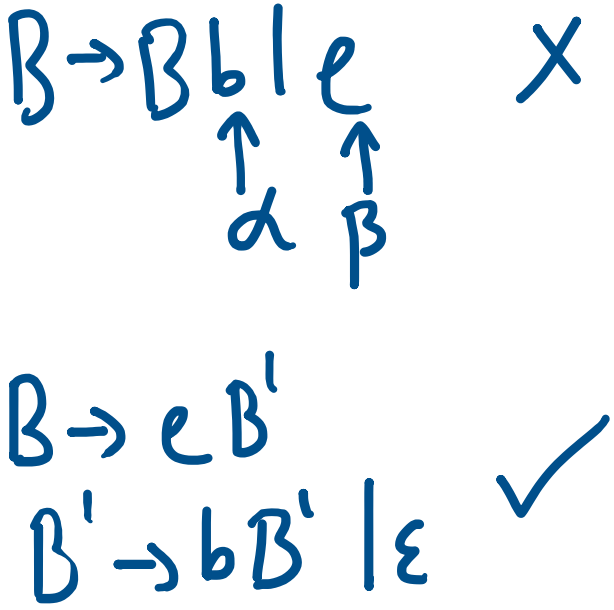
$1 \cup 2 =$

$A \to bA'$

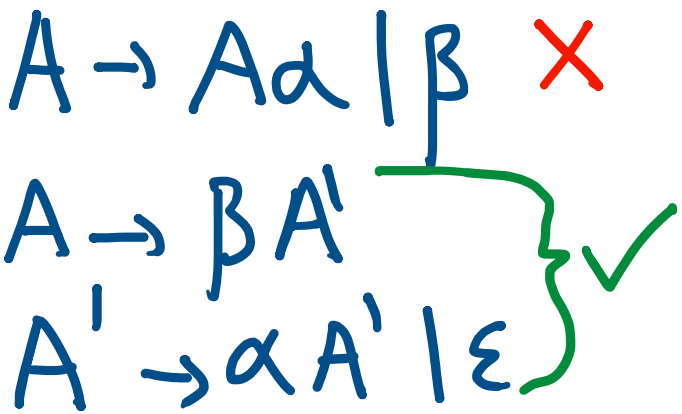
$A' \to aA' \mid dA' \mid \varepsilon$

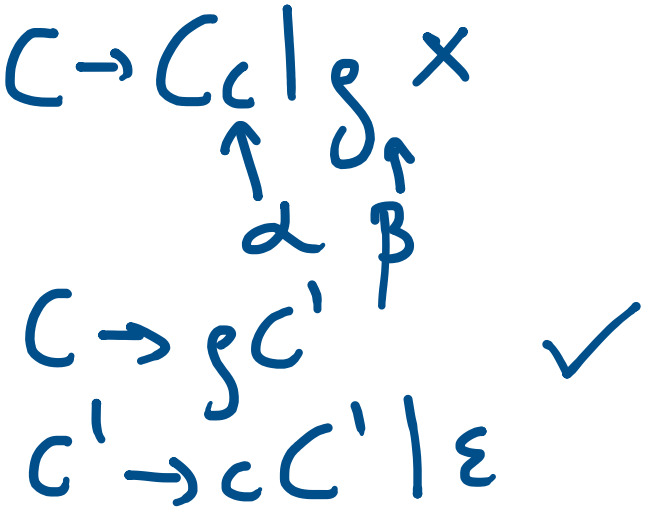
Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)





Sol Özyinelemeden Kurtulmak (Eliminating Left Recursion)





Bottom-up parsing (Alttan tepeye ayrıştırma)

- Shift-Reduce Parsing (Kaydırma-İndirgeme Ayrıştırması): Bir dizeyi (string) gramerin başlangıç sembolüne (S) dönüştürme işlemidir.
- Gramer için bir stack (yığın), ayrıştırılan string için bir girdi (input) kullanır.
- Amaç: string'i, başlangıç sembolüne indirgeme.
- 2 işlem uygulanabilir:
  - shift (kaydırma): girdi dizesindeki mevcut sembol bir stack'e push edilir.
  - reduce (indirgeme): stack'teki sembol bir non-terminal sembole dönüştürülür.

Shift Reduce Parser

- S -> S + S
- S -> S - S
- S -> (S)
- S -> a | b | c
- input: a - (b + c)

Stack	**Input**	**İşlem**
$	a - (b + c) $	shift a
$ a	- (b + c) $	reduce S -> a
$ S	- (b + c) $	shift -
$ S -	(b + c) $	shift (
$ S - (	b + c) $	shift b
$ S - (b	+ c) $	reduce S -> b
$ S - (S	+ c) $	shift +
$ S - (S +	c) $	chift c
$ S - (S + c	) $	reduce S -> c
$ S - (S + S	) $	shift)
$ S - (S + S)	$	reduce S -> S + S
$ S - (S)	$	reduce S -> (S)
$ S - S	$	reduce S -> S - S
$ S	$	accept

Shift Reduce Parser

- S -> S + S
- S -> S - S
- S -> (S)
- S -> a | b | c
- input: a * b

Stack	**Input**	**İşlem**
$	a * b $	shift a
$ a	- (b + c) $	reduce S -> a
$ S	- (b + c) $	shift -
$ S -	(b + c) $	shift (
$ S - (	b + c) $	shift b
$ S - (b	+ c) $	reduce S -> b
$ S - (S	+ c) $	shift +
$ S - (S +	c) $	chift c
$ S - (S + c	) $	reduce S -> c
$ S - (S + S	) $	shift)
$ S - (S + S)	$	reduce S -> S + S
$ S - (S)	$	reduce S -> (S)
$ S - S	$	reduce S -> S - S
$ S	$	accept