CENG 306 Biçimsel Diller ve Otomatlar
Formal Languages and Automata

DETERMINISM and PARSING

Konular

- Determinism and Parsing
- Top-Down Parsing
- Bottom-Up Parsing

Determinism and Parsing

- Context-free diller programlama dillerinin syntax analizinde yoğun bir şekilde kullanılmaktadır.
- Programlama dili için geliştirilmiş bir compiler, bir parser oluşturmak zorundadır.
- Parser girilen bir string'in ilgili dile ait olup olmadığını belirler. Dile aitse o string icin bir parse tree oluşturur.
- Compiler daha sonra parse tree'yi assembly dili gibi temel bir dildeki programa dönüştürür.
- Compiler icin parser oluşturmada en başarılı sonuçlar pushdown automata ile alınmaktadır.
- Ancak PDA'lar yapısı gereği nondeterministic'tir ve **deterministic olarak çalıştırılması zorunludur.**

Determinism and Parsing

- *Bir pushdown automaton* ***deterministic****'tir (DPDA)* ***eğer herbir configuration için kendisini izleyen sadece bir configuration varsa.***
- ***Eğer biri diğerinin prefix'i (öneki) ise iki katar consistent (tutarlı) dır,***
- *iki transition relation* $((p, a, \beta), (q, y))$ *ve* $((p, a', \beta'), (q', y'))$ ***compatible****'dır (uyumludur) eğer* <u>*a ve a' ile ve aynı zamanda*</u> β ve β' ile **consistent** ise yani her iki geçişin de aynı anda *kabul edildiği bir durum varsa.*
- *!!! Eğer* ***M PDA'sı deterministic ise iki farklı compatible geçiş olamaz.***

Determinism and Parsing

Örnek: *L = {wcw^R : w ∈ {a, b}*} dilini kabul eden aşağıdaki PDA deterministic'tir*

M = (K, ∑, Γ, Δ, s, F), K = {s, f}, ∑ = {a, b, c}, Γ = {a, b}, F = {f}

Δ toplam 5 adet geçiş ilişkisine sahip olsun;

1. *((**s**, a, e), (**s**, a))*
2. *((**s**, b, e), (**s**, b))*
3. *((**s**, c, e), (**f**, e))*
4. *((**f**, a, a), (**f**, e))*
5. *((**f**, b, b), (**f**, e))*

- *iki transition relation ((p, a, β), (q, y)) ve ((p, a', β'), (q', y')) **compatible**'dır (uyumludur) eğer a ve a' ile aynı zamanda β ve β' consistent ise yani her iki geçişin de aynı anda kabul edildiği bir durum varsa.*

Her bir durum ve giriş sembolü için sadece bir geçiş vardır.

Determinism and Parsing

Örnek: $L = \{ww^R : w \in \{a, b\}^*\}$ dilini kabul eden aşağıdaki PDA nondeterministic'tir

$M = (K, \sum, \Gamma, \Delta, s, F)$, $K = \{s, f\}$, $\sum = \{a, b\}$, $\Gamma = \{a, b\}$, $F = \{f\}$

Δ toplam 5 adet geçiş ilişkisine sahip olsun;

1. $((\mathbf{s}, a, e), (\mathbf{s}, a))$
2. $((\mathbf{s}, b, e), (\mathbf{s}, b))$
3. $((\mathbf{s}, e, e), (\mathbf{f}, e))$
4. $((\mathbf{f}, a, a), (\mathbf{f}, e))$
5. $((\mathbf{f}, b, b), (\mathbf{f}, e))$

- *iki transition relation $((p, a, \beta), (q, y))$ ve $((p, a', \beta'), (q', y'))$* ***compatible****'dır (uyumludur) eğer a ve a' ile aynı zamanda β ve β' consistent ise yani her iki geçişin de aynı anda kabul edildiği bir durum varsa.*

Transition 3, Transition 1 ve 2 ile compatible'dır. Ayrıca string'in orta noktası ***tahmin edilmektedir (belirgin değil)****.*

Determinism and Parsing

- *Deterministic context-free diller DPDA tarafından kabul edilir.*
- *Deterministic context-free diller giriş string'inin sonunu gösterebilmelidirler.*
- *$L \subseteq \sum^*$ deterministic context-free dildir eğer DPDA M için $L\$ = L(M)$ ise.*
- *Burada $\$$ işareti string'in sonunu göstermektedir ve $\$ \notin \sum$* olur.
- *$\$$ işareti tüm string'lere otomatik olarak eklenmiştir.*

Top-Down Parsing

Örnek : $L = \{a^n b^n : n >= 0\}$ *context-free dildir ve* $G = (\{a, b, S\}, \{a, b\}, R, S)$ *grammar'i tarafından üretilir.* $R = (S \rightarrow e, S \rightarrow aSb)$ *kurallarına sahiptir. Önce bir PDA oluşturalım.*

$M_1 = (\{p, q\}, \{a, b\}, \{a, b, S\}, \Delta_1, p, \{q\})$.

$L_1 = \{((p, e, e), (q, S)),$

$((q, e, S), (q, aSb)),$

$((q, e, S), (q, e)),$

$((q, a, a), (q, e)),$

$((q, b, b), (q, e))\}$

M_1 *otomatı deterministic hale dönüştürülebilir ve L$ dilini kabul eder.*

$M_2 = (\{p, q, q_a, q_b, q_\$\}, \{a, b\}, \{a, b, S\}, \Delta_2, p, \{q_\$\})$

Top-Down Parsing

Örnek : $L = \{a^n b^n : n >= 0\}$ *context-free dildir ve* $G = (\{a, b, S\}, \{a, b\}, R, S)$ *grammar'i tarafından üretilir.* $R = (S \rightarrow e, S \rightarrow aSb)$ *kurallarına sahiptir. Önce bir PDA oluşturalım.*

$M_1 = (\{p, q\}, \{a, b\}, \{a, b, S\}, \Delta_1, p, \{q\})$.

$L_1 = \{((p, e, e), (q, S)), ((q, e, S), (q, aSb)), ((q, e, S), (q, e)), ((q, a, a), (q, e)), ((q, b, b), (q, e))\}$

M_1 *otomatı deterministic hale dönüştürülebilir ve L\$ dilini kabul eder.*

$M_2 = (\{p, q, q_a, q_b, q_\$\}, \{a, b\}, \{a, b, S\}, \Delta_2, p, \{q_\$\})$

$\Delta_2 = \{((p, e, e), (q, S))$, (1)
$((q, a, e), (q_a, e))$, (2)
$((q_a, e, a), (q, e))$, (3)
$((q, b, e), (q_b, e))$, (4)
$((q_b, e, b), (q, e))$, (5)
$((q, \$, e), (q_\$, e))$, (6)
$((q_a, e, S), (q_a, aSb))$, (7)
$((q_b, e, S), (q_b, e))\}$ (8)

M_2 *q durumundayken stack'ta işlem yapmadan girişten bir sembol okur ve* q_a, q_b *veya* $q_\$$ *durumlarından birisine geçer. Böylece compatible iki geçiş olan*

$((q, e, S), (q, aSb))$ *ve* $((q, e, S), (q, e))$ *geçişlerini ayırır.*

Top-Down Parsing

Δ_2 ={((p, e, e), (q, S)), (1)
((q, a, e), (q_a, e)), (2)
((q_a, e, a), (q, e)), (3)
((q, b, e), (q_b, e)), (4)
((q_b, e, b), (q, e)), (5)
((q, $, e), ($q_\$$, e)), (6)
((q_a, e, S), (q_a, aSb)), (7)
((q_b, e, S), (q_b, e)) } (8)

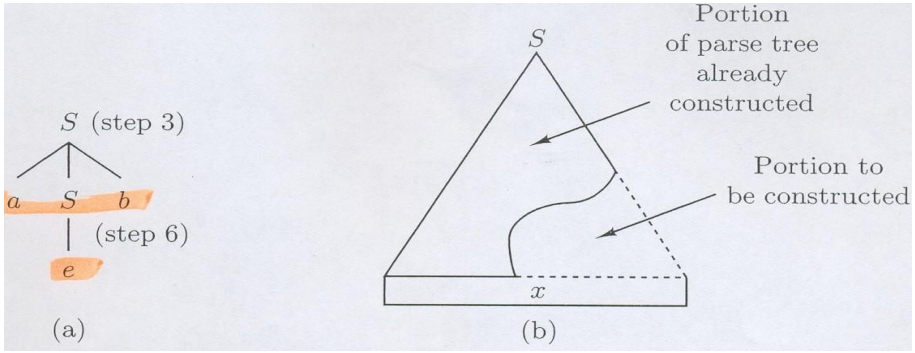
Örnek : (devam) DPDA M_2'nin ***ab$*** *için geçişleri aşağıda verilmiştir.*

Step	State	Unread Input	Stack	Transition Used	Rule of G
0	p	$ab\$$	e	-	
1	q	$ab\$$	S	1	
2	q_a	$b\$$	S	2	
3	q_a	$b\$$	aSb	7	$S \to aSb$
4	q	$b\$$	Sb	3	
5	q_b	$\$$	Sb	4	
6	q_b	$\$$	b	8	$S \to e$
7	q	$\$$	e	5	
8	$q_\$$	e	e	6	

Top-Down Parsing

Örnek : (devam)

- *M_2, $L = \{a^n b^n\}$ diline ait string'leri tanımak için deterministic olarak çalışır.*
- *M_2 giriş string'ini leftmost derivation ile üretir.*
- *Örnekteki 3. ve 6. adımlar parse tree'nin oluşturulduğu adımlardır.*



- *M_2 string'in dile ait olup olmadığını bulur, aynı anda parse tree oluşturur.*
- *Parse tree compiler'ların kullandığı parser'larda assembly dilinde program oluşturmak için kullanılmaktadır.*
- *M_2 **top-down parser**'dır, parse tree **top-down** ve **left-to-right** yaklaşımıyla oluşur.*

Top-Down Parsing

Örnek: Daha önce doğru yazılmış aritmetik ifadeler için oluşturulan grammar'e $F \rightarrow (E)$, şeklinde bir kural ekleyelim. Bu yeni kural fonksiyon çağırmalarını saglar.

*(Örn.: sqrt(x * x + 1)) Bu grammar için bir top-down parser oluşturalım.*

$M_3 = (\{p, q\}, \sum, \Gamma, \Delta, p, \{q\})$,

$\sum = \{(,), +, *, id\}$,

T = L ∪ {E, T, F}

L=*0. ((p, e, e), (q, E))*

1. ((q, e, E), (q, E+T))
2. ((q, e, E), (q, T))
3. ((q, e, T), (q, T*F))
4. ((q, e, T), (q, F))
5. ((q, e, F), (q, (E)))
6. ((q, e, F), (q, id))
7. ((q, e, F), (q, id(E)))

8-12: *ve son olarak tüm $x \in \sum$ için $((q, x, x), (q, e)) \in \Delta$ olsun.*

Top-Down Parsing

Örnek: (devam)

- *Bu otomatta nondeterminism 1-2, 3-4 ve 5-6-7 kurallarından kaynaklanmaktadır.*

Transition 6 ve 7: *M_3 otomatının (q, id, F) konfigürasyonunda olduğunu düşünelim. M_3 bu durumda 5, 6 veya 7 geçişlerinden birisini seçebilir. Input string'teki bir sonraki sembole (**id**) bakarak 5 elenir. Transition 5'te (**(**)bir sonraki semboldür. Ancak bir sonraki sembol 6 ve 7 için aynıdır (**id**).*

- *Bu problem sağ tarafı aynı olmasa da ilk sembolü aynı olan $F \rightarrow id$ ve $F \rightarrow id(E)$ kurallarından kaynaklanmaktadır.*
- *$F \rightarrow id$ ve $F \rightarrow id(E)$ kurallarının yerine $F \rightarrow idA$, $A \rightarrow e$ ve $A \rightarrow (E)$ kuralları konularak giderilebilir. Burada A yeni bir nonterminaldir.*
- *Transition 6 ve 7 yerine aşagıdaki kurallar konur;*
- *6'. ((q, e, F), (q, idA)) 7'. ((q, e, A), (q, e)) 8'. ((q, e, A), (q, (E)))*

Geçişler $(q, id(id), F) \vdash_M (q, id(id), idA) \vdash_M (q, (id), A) \vdash_M (q, (id), (E)) \vdash_M \ldots$ *olur.*

Top-Down Parsing

Nondeterminismi ortadan kaldırmak için kullanılan bu teknik ***left factoring*** *olarak adlandırılır. Aşagıdaki kural ile özetlenebilir;*

Heuristic Rule 1: *Eğer*

$$A \rightarrow \alpha\,\beta_1,\ A \rightarrow \alpha\,\beta_2, \ldots, A \rightarrow \alpha\,\beta_n\ ,$$

şeklinde kurallar varsa ve

$$\alpha \neq e \text{ ve } n \geq 2 \text{ ise,}$$

bu kurallar

$$A \rightarrow \alpha\,A',\quad A' \rightarrow \beta_i$$

kurallarıyla degiştirilir. A' yeni nonterminaldir.

Top-Down Parsing- Left Recursion

Transition 1 ve 2: *Eğer M_3 otomatı bir sonraki input sembol için **id** görürse, ve stack'taki **E** ise birkaç farklı işlem yapılabilir. Transition 2 yapılarak E yerine T yazılır. Girişin sadece **id** olması durumunda bu geçerlidir.*

*Transition 1 kullanılarak E yerine E + T yazılır. Girişin **id + id** olması durumunda geçerlidir. Transition 1 iki defa ve Transition 1 bir defa kullanılabilir. Girişin **id + id + id** olması durumunda geçerlidir. Burada sağ taraftaki işlemin kaç defa tekrarlanacağının sınırı belli degildir.*

- *Bu olay **left recursion** olarak adlandırılır.*
- *Bu problem E→ E + T kuralından kaynaklanmaktadır. Soldaki nonterminal sağdaki ilk semboldür.*
- *E → E + T ve E → T kuralları yerine E → TE', E' → +TE' ve E' → e kuralları konularak giderilebilir. Burada E' yeni bir nonterminaldir.*
- *Aynı işlem T → T * F, T → F içinde yapılır: T → FT ', T ' → *FT ' ve T ' → e*

Top-Down Parsing

Örnekteki grammar'in son şekli aşağıdaki gibi olur.

$G' = (V', \sum, R', E)$,

$V' = \sum \cup \{E, E', T, T', F, A\}$,

R = 1. $E \rightarrow TE'$

2. $E' \rightarrow +TE'$

3. $E' \rightarrow e$

4. $T \rightarrow FT'$

5. $T' \rightarrow *FT'$

6. $T' \rightarrow e$

7. $F \rightarrow (E)$

8. $F \rightarrow idA$

9. $A \rightarrow e$

10. $A \rightarrow (E)$

Top-Down Parsing

Nondeterminismi ortadan kaldırmak için kullanılan bu left recursion teknigi aşagıdaki kural ile özetlenebilir;

Heuristic Rule 2: *Eğer* $A \rightarrow A\alpha_1, \ldots, A \rightarrow A\ \alpha_n$ *ve* $A \rightarrow \beta_1, \ldots, A \rightarrow \beta_m$ *şeklinde kurallar varsa ve* β_i*ler A ile başlamıyorsa ve n > 0 ise, bu kurallar*

$A \rightarrow \beta_1 A', \ldots, A \rightarrow \beta_m A'$ *ve* $A' \rightarrow \alpha_1 A', \ldots, A' \rightarrow \alpha_n A'$ *ve* $A' \rightarrow e$

kurallarıyla degiştirilir. A' yeni nonterminaldir.

Top-Down Parsing

Örnek: *Önceki grammar'i tanıyan DPDA $M_4 = L(G')\$$ oluşturalım.*

$M_4 = (K, \sum \cup \{\$\}, V', \Delta, p, \{q_\$\})$,

$K = \{p, q, q_{id}, q_+, q_*, q_), q_(, q_\$\}$,

$\Delta = ((p, e, e), (q, E))$

$((q, a, e), (q_a, e))$	tüm $a \in \sum \cup \{ \$ \}$
$((q_a, e, a), (q, e))$	tüm $a \in \sum$
$((q_a, e, E), (q_a, TE'))$	tüm $a \in \sum \cup \{ \$ \}$
$((q_+, e, E'), (q_+, +TE'))$	
$((q_a, e, E'), (q_a, e))$	tüm $a \in \{), \$ \}$
$((q_a, e, T), (q_a, FT'))$	tüm $a \in \sum \cup \{ \$ \}$
$((q_*, e, T'), (q_*, *FT'))$	
$((q_a, e, T'), (q_a, e))$	tüm $a \in \{ +,), \$ \}$
$((q_(, e, F), (q_(, (E)))$	
$((q_{id}, e, F), (q_{id}, idA))$	
$((q_(, e, A), (q_(, (E)))$	
$((q_a, e, A), (q_a, e))$	tüm $a \in \{ +, \cup,), \$ \}$

M_4 deterministic pushdown automaton'u G' grammar'i için bir parser'dır.

Top-Down Parsing

*Örnek: (devam) id * (id)$ giriş string'i aşağıdaki tabloda görüldüğü gibi kabul edilir.*

Step	State	Unread Input	Stack	Rule of G'
0	p	id $*$ (id)$	e	
1	q	id $*$ (id)$	E	
2	q_{id}	$*$(id)$	E	
3	q_{id}	$*$(id)$	TE'	1
4	q_{id}	$*$(id)$	$FT'E'$	4
5	q_{id}	$*$(id)$	id$AT'E'$	8
6	q	$*$(id)$	$AT'E'$	
7	q_*	(id)$	$AT'E'$	
8	q_*	(id)$	$T'E'$	9
9	q_*	(id)$	$*FT'E'$	5
10	q	(id)$	$FT'E'$	
11	$q_($	id)$	$FT'E'$	
12	$q_($	id)$	$(E)T'E'$	7
13	q	id)$	$E)T'E'$	

Top-Down Parsing

Örnek: (devam)

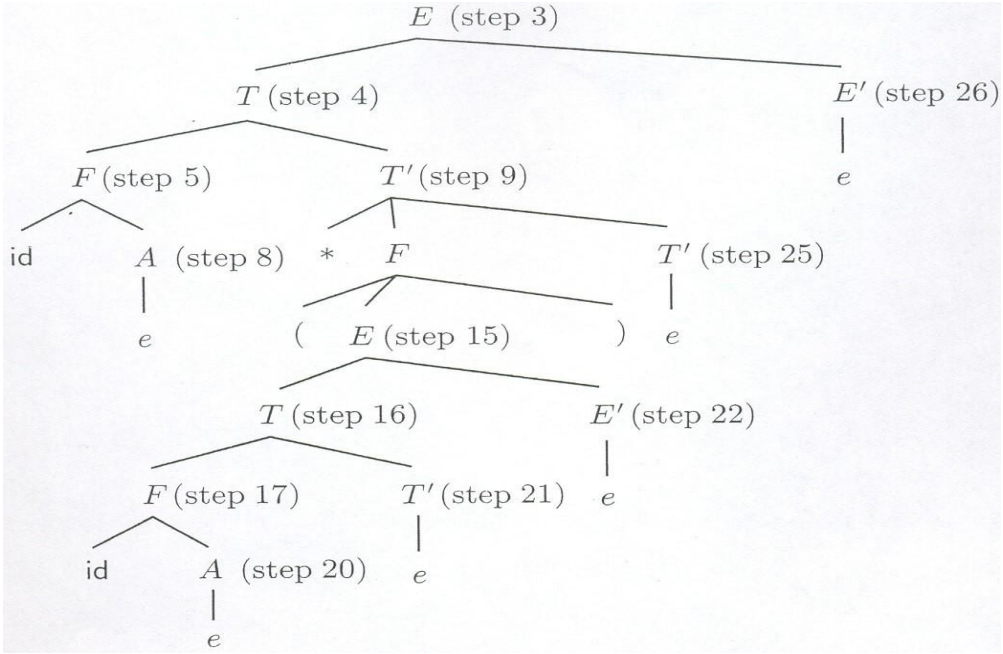
14	q_{id}	$)\$$	$E)T'E'$	
15	q_{id}	$)\$$	$TE')T'E'$	1
16	q_{id}	$)\$$	$FT'E')T'E'$	4
17	q_{id}	$)\$$	$\text{id}AT'E')T'E'$	8
18	q	$)\$$	$AT'E')T'E'$	
19	$q_{)}$	$\$$	$AT'E')T'E'$	
20	$q_{)}$	$\$$	$T'E')T'E'$	10
21	$q_{)}$	$\$$	$E')T'E'$	6
22	$q_{)}$	$\$$	$)T'E'$	3
23	q	$\$$	$T'E'$	6
24	$q_{\$}$	e	$T'E'$	
25	$q_{\$}$	e	E'	6
26	$q_{\$}$	e	e	3

Top-Down Parsing

Örnek: (devam) G' deki kurallar ile stack üzerinde nonterminal degi$tirilen adımlar tabloda son sütunda numaralandırılmı$tır. Sırayla bu kurallar uygulandıgında id(id)$ string'inin leftmost derivation'ı elde edilir.*

Oluşturulan parse yandadır.

$E \Rightarrow TE'$
$\Rightarrow FT'E'$
$\Rightarrow idT'E'$
$\Rightarrow id\ *FT'E'$
$\Rightarrow id\ *(E)T'E'$
$\Rightarrow id\ *(TE')T'E'$
$\Rightarrow id\ *(FT'E')T'E'$
$\Rightarrow id\ *(idT'E')T'E'$
$\Rightarrow id\ *(idE')T'E'$
$\Rightarrow id\ *(id)T'E'$
$\Rightarrow id\ *(id)E'$
$\Rightarrow id\ *(id)$



*Parse tree **top-down** ve **left-first** olarak oluşmuştur.*

Bottom-Up Parsing

- *Context-free dillerin parse edilmesinde en iyi yol yoktur. Farklı grammar'ler için farklı yöntemler vardır.*
- *Farklı bir yaklaşımda automaton ilk önce girişi okur ve derivation sonra yapılır.*
- *Sonuçta parse tree yapraklardan root node'a dogru gerçekleşir.*
- *Bu yöntemler **bottom-up** olarak adlandırılırlar.*

Bottom-Up Parsing

G = (V, L*, R, S) bir CFG için M = (K, ∑, Γ, Δ, p, F) bottom-up pushdown automaton'u oluşturalım. Burada K = {p, q}, Γ = V, F = {q} ve Δ aşağıdaki geçişlere sahip olsun.*

1. *((p, a, e), (p, a))* *tüm a ∈* L *için*
2. *((p, a, α^R), (p, A))* *tüm A → α ∈ R için*
3. *((p, e, S), (q, e))*

Her transition bir transition sınıfını göstermektedir. Transition 1 input sembolleri stack'a aktarır. Transition 2 stack'ta kuralların sag kısmının yerine sol kısmını degiştirir. Kuralların sag kısmı ters sırada bulunmalıdır. Transition 3 ise sonuç durumuna geçerek çalışmayı sonlandırmayı saglar.

Bottom-Up Parsing

Örnek: *Aritmetik deyimleri üreten gramer için bir bottom-up pushdown automaton oluşturalım. Kurallar aşağıdaki gibi olsun.*

$E \rightarrow E + T$	(R1)	$E \rightarrow T$	(R2)
$T \rightarrow T * F$	(R3)	$T \rightarrow F$	(R4)
$F \rightarrow (E)$	(R5)	$F \rightarrow id$	(R6)

M pushdown automaton'u için aşagıdaki geçişler oluşturulur.

$(p, a, e), (p, a)$	tüm $a \in \sum$ için	($\Delta 0$)
$(p, e, T + E), (p, E)$		($\Delta 1$)
$(p, e, T), (p, E)$		($\Delta 2$)
$(p, e, F * T), (p, T)$		($\Delta 3$)
$(p, e, F), (p, T)$		($\Delta 4$)
$(p, e,)E(), (p, F)$		($\Delta 5$)
$(p, e, id), (p, F)$		($\Delta 6$)
$(p, e, E), (q, e)$		($\Delta 7$)

Bottom-Up Parsing

*Örnek: (devam) id * (id) aşagıdaki gibi kabul edilir.*

Step	State	Unread Input	Stack	Transition Used	Rule of G
0	p	id $*$ (id)	e		
1	p	$*$(id)	id	$\Delta 0$	
2	p	$*$(id)	F	$\Delta 6$	R6
3	p	$*$(id)	T	$\Delta 4$	R4
4	p	(id)	$*T$	$\Delta 0$	
5	p	id)	$(*T$	$\Delta 0$	
6	p	)	id$(*T$	$\Delta 0$	
7	p	)	$F(*T$	$\Delta 6$	R6
8	p	)	$T(*T$	$\Delta 4$	R4
9	p	)	$E(*T$	$\Delta 2$	R2
10	p	e	$)E(*T$	$\Delta 0$	
11	p	e	$F*T$	$\Delta 5$	R5
12	p	e	T	$\Delta 3$	R3
13	p	e	E	$\Delta 2$	R2
14	q	e	e	$\Delta 7$	

Bottom-Up Parsing

Örnek: (devam)

- *M otomatı deterministic degildir. Çünkü Δ0 diğer tüm geçişlerle (Δ 1 - Δ 8) compatible'dir.*
- *Herhangi bir anda M bir terminali stack'a aktarabilir (1, 4, 5, 6 ve 10 adımlar) veya stack'taki birkaç sembolü bir kuralın sag kısmı olarak elde edebilir.*
- *Kuralın sağ kısmı olarak görülen string sol kısımla değiştirilerek indirgenir (Δ1 –Δ6).*
- *indirgeme yapılan adımlar ters sırada alınırsa **rightmost derivation** yapılır.*

Bottom-Up Parsing

Örnek: (devam)

indirgeme yapılan adımların ters sırada alınmasıyla elde edilen rightmost derivation aşagıdaki gibidir.

$E \Rightarrow T$

$\Rightarrow T * F$

$\Rightarrow T * (E)$

$\Rightarrow T * (T)$

$\Rightarrow T * (F)$

$\Rightarrow T * (id)$

$\Rightarrow F * (id)$

$\Rightarrow id * (id)$

Ödev

- Problemleri çözünüz 3.7.1a, 3.7.1b (sayfa 173)