BM 305 Biçimsel Diller ve Otomatlar (Formal Languages and Automata)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Context-Free and Non-Context-Free Languages
- Determinism and Parsing
- Top-Down Parsing
- Bottom-Up Parsing

Context-Free and Non-Context Free Languages

- Context-free dillerin üretilmesi için context-free grammar'ler kullanılmaktadır.
- Context-free dillerin tanınması için pushdown automata kullanılmaktadır.
- Context-free grammar'in ürettiği dili tanıyan bir pushdown automata oluşturulabilir.
- Bir dilin context-free veya non-context-free olduğunu belirlemek için yöntemler vardır.
- Closure properties ve pumping lemma iki farklı yöntem olarak regular dillerde olduğu gibi kullanılabilir.

Context-Free and Non-Context Free Languages

Theorem: *Context-free diller union, concatenation ve Kleene star işlemleri için kapalıdır.*

Proof: $G_1 = (V_1, \Sigma_1, R_1, S_1)$ *ve* $G_2 = (V_2, \Sigma_2, R_2, S_2)$ *iki farklı grammar olsun. Bu iki grammar için nonterminal kümeleri disjoint olsun.* $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \phi$

Union

S yeni bir sembol ve $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R, S)$ *ve* $R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$ *olsun. Amaç* $L(G) = L(G_1) \cup L(G_2)$ *olduğunu göstermektir. Herhangi bir w string'i için (* $S \rightarrow S_1, S \rightarrow S_2$ *olduğundan)* $S \Rightarrow^*_G w$ *olur eğer sadece ve sadece* $S_1 \Rightarrow^*_G w$ *veya* $S_2 \Rightarrow^*_G w$ *ise.*

Nonterminaller kümeleri disjoint olduğu için ilk kuralla S_1 ***veya*** S_2***'ye geçildikten sonra diğerine tekrar dönülmez.***

Context-Free and Non-Context Free Languages

Proof: (devam)

Concatenation

$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}, S)$ *şeklinde tanımlanan bir grammar' le* $L(G_1)L(G_2)$ *dili oluşturulabilir.*

Birinci grammar'deki nonterminaller (S_1 içindeki) terminallere dönüştürüldükten sonra ikinci grammar'deki nonterminaller (S_2 içindeki) terminallere dönüştürülür.

Context-Free and Non-Context Free Languages

Proof: (devam)

Kleene star

$G = (V_1 \cup \{S\}, \Sigma_1, R_1 \cup \{S \rightarrow e, S \rightarrow SS_1\}, S)$ *şeklinde tanımlanan bir grammar' le* $L(G_1)^*$ *dili oluşturulabilir.*

$S \rightarrow SS_1$ kuralının tekrarı ile dildeki kuralın ($S \rightarrow S_1$) tekrarı istenen sayıda yapılabilir.

Context-Free and Non-Context Free Languages

Tanımlar:

*$G = (V, \sum, R, S)$ bir context-free grammar olsun. G'nin **fanout** değeri $\phi(G)$ olarak gösterilir ve R kurallar kümesinde sağ kısmı en uzun olan kuralın sağ kısmındaki sembol sayısıdır.*

*Bir parse tree üzerinde **path** root node ile yaprak node arasında farklı node'lardan geçilerek elde edilen sıradır.*

*Yolun **length** değeri üzerindeki çizgi sayısıdır.*

*Bir parse tree için **height** en uzun yolun length değeridir.*

Context-Free and Non-Context Free Languages

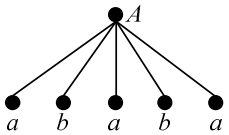
*Lemma: **G** grammar'ine ait **h** height değerine sahip bir parse tree'nin ürettiği string'in uzunluğu en çok $\phi(G)^h$ olabilir.*

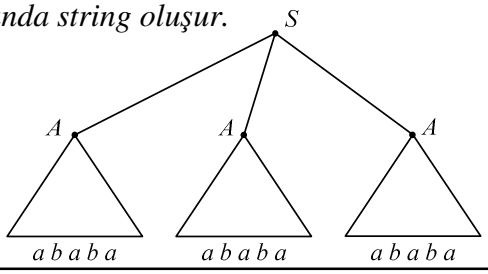
Proof: h = 1 için parse tree grammar içinde bir kuraldır (2.durum). Ençok $\phi(G)^h=\phi(G)$ uzunluğunda string üretilir. ($S\rightarrow abc$, $S\rightarrow abcabcabc$)

- *$h \geq 1$ olan her h değeri için yeni bir root oluşur h-1 yüksekliğindeki parse tree'leri birbirine bağlar.*
- *h+1 için yüksekliği ençok h olan enfazla $\phi(G)$ adet parse tree birbirine bağlanır (3.durum). Her parse tree $\phi(G)^h$ uzunluğunda string oluşturur ve toplam en çok $\phi(G)^{h+1}$ uzunluğunda string oluşur.*

$R_1 = (A \rightarrow ababa, A \rightarrow aba, ...)$,
$R_2 = (S \rightarrow AAA, A \rightarrow ababa, ...)$

h= 1





Context-Free and Non-Context Free Languages

Pumping Theorem: *G = (V, ∑, R, S) bir CFG olsun. Uzunluğu $\phi(G)^{|V-\Sigma|}$ dan büyük her w∈ L(G) string'i* ***w = uvxyz*** *şeklinde yazılabilir.*
Tüm n ≥ 0 değerleri için ***v*** *veya* ***y*** *den birisi boş olmamak kaydıyla* ***$uv^nxy^nz \in L(G)$*** *olur. Bunu sağlamayan non-context-free dildir.*

Örnek: L = $\{a^nb^nc^n : n \geq 0\}$ dili non-context-free'dir. Bir CFG G = (V, ∑, R, S) için L = L(G) olduğunu düşünelim. w = $a^nb^nc^n$ dile ait olmalıdır ve w = uvxyz şeklinde gösterilebilmelidir. Burada v veya y' den en az birisi boş olamaz ve tüm n ≥ 0 için $uv^nxy^nz \in$ L(G) olmalıdır.

- *Eğer vy string'i a, b ve c'lerin üçünüde içerirse v ve y'den birisi en az ikisini (ab, bc) içerir. uv^2xy^2z string'i a,b,c 'lerin sırasını bozar. b'lerden sonra a veya c'lerden sonra b gelir.*
- *Eğer vy string'i a, b ve c'lerin bir kısmını içerirse uv^2xy^2z string'i eşit olmayan sayıda a, b ve c'ler üretir.*

Context-Free and Non-Context Free Languages

Theorem: Context-free diller complementation ve intersection için kapalı değildir.

Proof: $\{a^nb^nc^m : m,n \geq 0\}$ ile $\{a^mb^nc^n : m,n \geq 0\}$ dilleri context-free'dir.

Bu iki dilin kesişimi $\{a^nb^nc^n : n \geq 0\}$ olur. Bu dil non-context-free'dir.

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ *olduğu için eğer complementation için kapalı olsaydı kesişim içinde kapalı olurdu.*

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 için bir parse tree oluşturur.
- Compiler daha sonra parse tree'yi assembly dili gibi temel bir dildeki programa dönüştürür.
- Compiler için parser oluşturmada en başarılı sonuçlar pushdown automata ile alınmaktadır.
- Ancak PDA'lar 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.*
- *İki transition relation $((p, a, \beta), (q, \gamma))$ ve $((p, a', \beta'), (q', \gamma'))$ **compatible**'dır eğer ikisinide kabul eden bir durum varsa.*
- *Eğer M pushdown automata 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))*

Herbir durum ve giriş sembolü için sadece bir geçiş vardır.

Determinism and Parsing

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

M = (K, ∑, Γ, Δ, s, F), K = {s, f}, ∑ = {a, b}, 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, e, e), (f, e))*
4. *((f, a, a), (f, e))*
5. *((f, b, b), (f, e))*

Transition 3, 1 ve 2 ile compatible'dır. Ayrıca string'in orta noktası tahmin edilmektedir.

Determinism and Parsing

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

Top-Down Parsing

Örnek : $L = \{a^n b^n : n \geq 0\}$ 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.

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

$\Delta_1 = \{((p, e, e), (q, S)), ((q, e, S), (q, aSb)), ((q, e, S), (q, e)),$
$((q, a, a), (q, e)), ((q, b, b), (q, e))\}$

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) $((q_b, e, b), (q, e))$, (5)
$((q, a, e), (q_a, e))$, (2) $((q, \$, e), (q_\$, e))$, (6)
$((q_a, e, a), (q, e))$, (3) $((q_a, e, S), (q_a, aSb))$, (7)
$((q, b, e), (q_b, e))$, (4) $((q_b, e, S), (q_b, e))\ \}$ (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

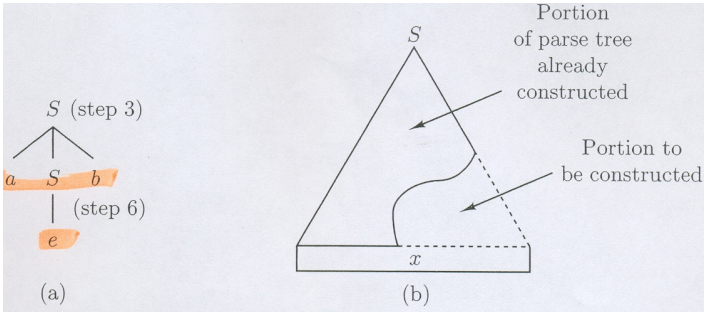
Örnek : (devam) DPDA M_2 'nin ab$ için geçişleri aşağıda verilmiştir.

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 oluşturulmuş grammar'e F → (E), şeklinde bir kural ekleyelim. Bu yeni kural fonksiyon çağırmalarını sağlar. (Örn.: sqrt(x * x + 1))*

Bu grammar için bir top-down parser oluşturalım.

$M_3 = (\{p, q\}, \Sigma, \Gamma, \Delta, p, \{q\})$,
$\Sigma = \{(,), +, *, id\}$,
$\Gamma = \Sigma \cup \{E, T, F\}$
Δ=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)))

ve son olarak tüm a ∈ Σ için ((q, a, a), (q, e)) ∈ Δ 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 olmasada ilk sembolü aynı olan F → id ve F → id(E) kurallarından kaynaklanmaktadır.*
- *F → id ve F → id(E) kurallarının yerine F → idA, A → e ve A → (E) kuralları konularak giderilebilir. Burada A yeni bir nonterminaldir.*
- *Transition 6 ve 7 yerine aşağıdaki 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. Aşağıdaki 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 $\alpha \neq e$ ve $n \geq 2$ ise, bu kurallar $A \rightarrow \alpha A'$, $A' \rightarrow \beta_i$ kurallarıyla değiştirilir. A' yeni nonterminaldir.*

Top-Down Parsing

*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 değildir.*

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

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 tekniği aşağıdaki 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 değiştirilir.* A' *yeni nonterminaldir.*

Top-Down Parsing

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

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

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

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

$((q, a, e), (q_a, e))$ tüm $a \in \Sigma \cup \{ \$ \}$

$((q_a, e, a), (q, e))$ tüm $a \in \Sigma$

$((q_a, e, E), (q_a, TE'))$ tüm $a \in \Sigma \cup \{ \$ \}$

$((q_+, e, E'), (q_+, +TE'))$

$((q_a, e, E'), (q_a, e))$ tüm $a \in \{), \$ \}$

$((q_a, e, T), (q_a, FT'))$ tüm $a \in \Sigma \cup \{ \$ \}$

$((q_*, e, T'), (q_*, *FT'))$

$((q_a, e, T'), (q_a, e))$ 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 $a \in \{ +, *,), \$ \}$

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

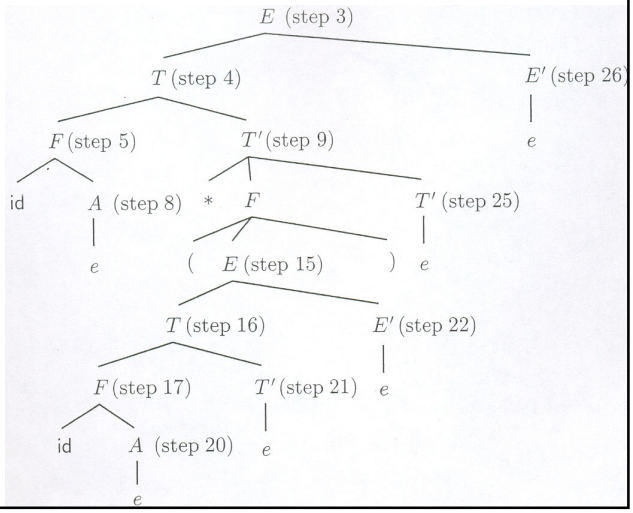
Top-Down Parsing

Örnek: (devam) ***G′*** *deki kurallar ile stack üzerinde nonterminal değiştirilen adımlar tabloda son sütunda numaralandırılmıştır. Sırayla bu kurallar uygulandığında 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 doğru gerçekleşir.*
- *Bu yöntemler **bottom-up** olarak adlandırılırlar.*

Bottom-Up Parsing

$G = (V, \Sigma, R, S)$ bir CFG için $M = (K, \Sigma, \Gamma, \Delta, p, F)$ bottom-up pushdown automaton'u oluşturalım. Burada $K = \{p, q\}$, $\Gamma = V$, $F = \{q\}$ ve Δ aşağıdaki geçişlere sahip olsun.

1. $((p, a, e), (p, a))$ tüm $a \in \Sigma$ için
2. $((p, a, \alpha^R), (p, A))$ tüm $A \rightarrow \alpha \in R$ 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 sağ kısmının yerine sol kısmını değiştirir. Kuralların sağ kısmı ters sırada bulunmalıdır. Transition 3 ise sonuç durumuna geçerek çalışmayı sonlandırmayı sağlar.

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 aşağıdaki geçişler oluşturulur.

(p, a, e), (p, a)	*tüm* $a \in \Sigma$ *için*	*(Δ0)*
(p, e, T + E), (p, E)		*(Δ1)*
(p, e, T), (p, E)		*(Δ2)*
*(p, e, F * T), (p, T)*		*(Δ3)*
(p, e, F), (p, T)		*(Δ4)*
(p, e,)E(), (p, F)		*(Δ5)*
(p, e, id), (p, F)		*(Δ6)*
(p, e, E), (q, e)		*(Δ7)*

Bottom-Up Parsing

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

Step	State	Unread Input	Stack	Transition Used	Rule of G
0	p	id $*$ (id)	e		
1	p	$*$(id)	id	Δ0	
2	p	$*$(id)	F	Δ6	R6
3	p	$*$(id)	T	Δ4	R4
4	p	(id)	$*T$	Δ0	
5	p	id)	$(*T$	Δ0	
6	p	)	id$(*T$	Δ0	
7	p	)	$F(*T$	Δ6	R6
8	p	)	$T(*T$	Δ4	R4
9	p	)	$E(*T$	Δ2	R2
10	p	e	$)E(*T$	Δ0	
11	p	e	$F*T$	Δ5	R5
12	p	e	T	Δ3	R3
13	p	e	E	Δ2	R2
14	q	e	e	Δ7	

Bottom-Up Parsing

Örnek: (devam)

- *M otomatı deterministic değildir. Çünkü Δ0 diğer tüm geçişlerle (Δ1 - Δ8) 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 sağ kısmı olarak elde edebilir.*
- *Kuralın sağ kısmı olarak görülen string sol kısımla değiştirilerek indirgenir(Δ1 – Δ6).*
- *İndirgeme yapılan adımlar ters sırada alınırsa **rightmost derivation** yapılır.*

Bottom-Up Parsing

Örnek: (devam)

İndirgeme yapılan adımların ters sırada alınmasıyla elde edilen rightmost derivation aşağıdaki gibidir.

$$
\begin{aligned}
E &\Rightarrow T \\
&\Rightarrow T * F \\
&\Rightarrow T * (E) \\
&\Rightarrow T * (T) \\
&\Rightarrow T * (F) \\
&\Rightarrow T * (id) \\
&\Rightarrow F * (id) \\
&\Rightarrow id * (id)
\end{aligned}
$$

Ödev

- Problemleri çözünüz 3.5.2c (sayfa 148)
- Problemleri çözünüz 3.5.5a (sayfa 148)
- Problemleri çözünüz 3.5.14a, 3.5.14c (sayfa 149)
- Problemleri çözünüz 3.7.1a, 3.7.1b (sayfa 173)