

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

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Context-Free (bağlamdan bağımsız, içerikten bağımsız) Grammars (CFGs) and Languages (CFLs)
- Parse Trees and Derivations



Context-Free Grammars and Languages

- Dil tanıyıcı cihaz bir dile ait geçerli string'leri kabul eder.
- Dil üretici cihaz bir dile ait string'leri oluşturur.
- Dil üretici cihazlar bir başlangıç işaretiyle bir string oluşturmaya başlarlar ve belirlenmiş kuralları kullanarak string oluştururlar.
- Regular expression bir dil üretici olarak kabul edilir.

Örnek:

$a(a^* \cup b^*)b$ regular expression ile önce bir a üretilir. Ardından iki durumdan birisine göre devam edilir. İstenen sayıda a üretilir veya b üretilir. Son olarakta bir b üretilir.



Context-Free Grammars and Languages

- Context-free grammar'ler çok daha karmaşık dilleri üretebilir.

Örnek:

- $a(a^* \cup b^*)b$ regular expression ön kısım, orta kısım ve son kısım olarak ayrıştırılabilir.
- S dilde bir string ve M ise orta kısım olmak üzere
 $S \rightarrow aMb$ şeklinde bir kural yazılabilir.
Burada M , a 'lardan veya b 'lerden oluşturulabilir.
 $M \rightarrow A$ ve $M \rightarrow B$ yeni iki kural, A ve B dile ait stringlerdir.
 $A \rightarrow e, A \rightarrow aA$ ve $B \rightarrow e, B \rightarrow bB$
- $a(a^* \cup b^*)b$ regular expression tarafından oluşturulan dil yukarıdaki kurallarla oluşturulabilir.

$aaab$ string'i aşağıdaki gibi oluşturulur;

$S \rightarrow aMb, S \rightarrow aAb (M \rightarrow A), S \rightarrow aaAb (A \rightarrow aA), S \rightarrow aaab$



Context-Free Grammars and Languages

- Kurallarla değiştirme işlemi, sadece değişecek sembol üzerinde yapılır ve önündeki ve ardındaki kısımlara bakılmaz.
- Bu yüzden **context-free** olarak adlandırılır.
- S, A, B, M nonterminal, a, b, e terminal olarak adlandırılır.
- Tüm string'ler sadece terminallerden oluşabilir.
- Kuralların uygulanması ve yeni bir string elde edilmesi regular expression'larda yeni bir string elde edilmesi gibidir.



Context-Free Grammars and Languages

Tanım:

- Bir context-free grammar $G = (V, \Sigma, R, S)$ şeklinde tanımlanır.

V alfabe (terminal ve nonterminaller)

Σ terminaller ($\Sigma \subseteq V$)

R kurallar ($V - \Sigma \times V^*$)

S başlangıç sembolü ($S \in (V - \Sigma)$)

- $A \in V - \Sigma$ ve $u \in V^*$ için $(A, u) \in R$ ise $A \rightarrow_G u$ yazabiliriz.
- $u, v \in V^*$ için $u \Rightarrow_G v$ yazabiliriz sadece ve sadece $x, y \in V^*$ ve $A \in V - \Sigma$ ve $u = xAy$ ve $v = xv'y$ ve $A \rightarrow_G v'$ ise



Context-Free Grammars and Languages

- $\Rightarrow_G^*$, $\Rightarrow_G$ ilişkisinin *reflexive, transitive, closure*'udur.
- G grammar'ı tarafından oluşturulan dil $L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$ şeklindedir.
- Kurallar uygulanarak dile ait tüm string'ler elde edilebilir.
- Oluşturulan $L(G)$ dili **context-free language** olarak adlandırılır.
- $A \rightarrow_G u$ ve $u \Rightarrow_G v$ yerine $A \rightarrow u$ ve $u \Rightarrow v$ yazılabilir.
- $w_0 \Rightarrow_G w_1 \Rightarrow_G \dots \Rightarrow_G w_n$ **derivation** (türetme) olarak adlandırılır. w_n string'i w_0 'dan türetilmiştir.
- $w_0, \dots, w_n \in V^*$ olabilir. n **derivation length** (türetme uzunluğu) olarak adlandırılır.



Context-Free Grammars and Languages

Örnek:

$G = (V, \Sigma, R, S)$ grammar'ı için

$V = \{S, a, b\}$, $\Sigma = \{a, b\}$ ve R kümesi $S \rightarrow aSb$ ve $S \rightarrow e$ olmak üzere iki tane kurala sahip olsun. Örnek bir derivation aşağıdaki gibi olabilir:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

İlk iki adımda $S \rightarrow aSb$ ve son olarak $S \rightarrow e$ uygulanmıştır.

Oluşturulan dil $L(G) = \{a^n b^n : n \geq 0\}$ olmuştur.

Bazı context-free diller regular değildir, ancak tüm regular diller context-free dildir.

Context-Free Grammars and Languages

Örnek:

$G = (W, \Sigma, R, S)$ grammar'i için

$W = \{ S, A, N, V, P \} \cup \Sigma$,

$\Sigma = \{ Jim, big, green, cheese, ate \}$,

$R = \{ P \rightarrow N,$

$P \rightarrow AP,$

$S \rightarrow PVP,$

$A \rightarrow big,$

$A \rightarrow green,$

$N \rightarrow cheese,$

$N \rightarrow Jim,$

$V \rightarrow ate \}$

$S = sentence$

$A = adjective$

$V = verb$

$N = noun$

$P = phrase$

$L(G)$ deki grammar olarak doğru bazı string'ler aşağıdadır;

Jim ate cheese

big Jim ate green cheese

big cheese ate Jim

Bazı string'ler anlamsız olabilir;

big cheese ate green green big green big cheese

green Jim ate green big Jim

Context-Free Grammars and Languages

- Herhangi bir bilgisayar programlama diliyle yazılmış bilgisayar programının, yazımının doğru olabilmesi için katı kurallara uyması gerekir.
- Konuşma dillerindeki tersine birçok programlama dilinin yazımının doğruluğunu kontrol etmek için context-free grammar'ler kullanılabilir.
- Özellikle programların syntax analizinde parse edilmesi aşamasında çok faydalıdır.
- Bir programlama dilinde bazı kısımlar regular expression'larla ifade edilebilir, program yapısı veya blok yapıları context-free grammar'ler tarafından ifade edilebilir.



Context-Free Grammars and Languages

Örnek: Bütün programlama dillerindeki ortak bir kısmı oluşturan bir dil tanımlayalım. Bu dil doğru yazılmış aritmetik ifadeleri gösterebilir.
 $id * (id * id + id)$ yazımı doğru ancak $* id + ($ ve $+ * id$ yanlıştır.
 (id değişken adlarıdır)

$$G = (V, \Sigma, R, E)$$

$$V = \{+, *, (,), id, T, F, E\},$$

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

$$R = \{ E \rightarrow E + T, \quad (R1)$$

$$E \rightarrow T, \quad (R2)$$

$$T \rightarrow T * F, \quad (R3)$$

$$T \rightarrow F, \quad (R4)$$

$$F \rightarrow (E), \quad (R5)$$

$$F \rightarrow id \quad (R6)$$

$E = \text{expression}$

$T = \text{term}$

$F = \text{factor}$



Context-Free Grammars and Languages

Örnek: (devam)

G grammar'ini $(id * id + id) * (id + id)$ string'ini aşağıdaki gibi oluşturur;

$$E \Rightarrow T \quad (R2)$$

$$\Rightarrow T * F \quad (R3)$$

$$\Rightarrow T * (E) \quad (R5)$$

$$\Rightarrow T * (E + T) \quad (R1)$$

$$\Rightarrow T * (T + T) \quad (R2)$$

$$\Rightarrow T * (F + T) \quad (R4)$$

$$\Rightarrow T * (id + T) \quad (R6)$$

$$\Rightarrow T * (id + F) \quad (R4)$$

$$\Rightarrow T * (id + id) \quad (R6)$$

$$\Rightarrow F * (id + id) \quad (R4)$$

$$\Rightarrow (E) * (id + id) \quad (R5)$$

$$\Rightarrow (E + T) * (id + id) \quad (R1)$$

$$\Rightarrow (E + F) * (id + id) \quad (R4)$$

$$\Rightarrow (E + id) * (id + id) \quad (R6)$$

$$\Rightarrow (T + id) * (id + id) \quad (R2)$$

$$\Rightarrow (T * F + id) * (id + id) \quad (R3)$$

$$\Rightarrow (F * F + id) * (id + id) \quad (R4)$$

$$\Rightarrow (F * id + id) * (id + id) \quad (R6)$$

$$\Rightarrow (id * id + id) * (id + id) \quad (R6)$$



Context-Free Grammars and Languages

Örnek: Düzgün dağılımlı sağ ve sol parantezleri üreten bir grammar oluşturalım.

$$G = (V, \Sigma, R, S)$$

$$V = \{ S, (,) \},$$

$$\Sigma = \{ (,) \},$$

$$R = \{ S \rightarrow e,$$

$$S \rightarrow SS$$

$$S \rightarrow (S) \}$$

Aşağıdaki türetmeleri G grammar' i oluşturur;

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S((S)) \Rightarrow S(()) \Rightarrow ()(())$$

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()(())$$



Context-Free Grammars and Languages

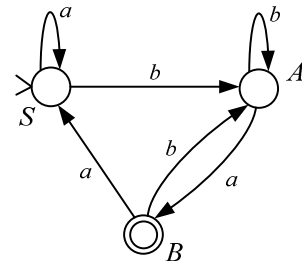
Örnek: Bir DFA $M = (K, \Sigma, \delta, s, F)$ tarafından tanınan regular dil $L(M)$, $G(M) = (V, \Sigma, R, S)$ grammar' i tarafından oluşturulabilir. Burada,

$$V = K \cup \Sigma,$$

$$S = s,$$

$$R = \{ q \rightarrow ap : \delta(q, a) = p \} \cup \{ q \rightarrow e : q \in F \}$$

Nonterminaller otomatın durumları ve a girişi için yapılan q dan p ye geçiş R içinde $q \rightarrow ap$ şeklinde bir kural olarak alınır.



Yandaki otomat için oluşturulan kurallar;

$$S \rightarrow aS, S \rightarrow bA, A \rightarrow bA, A \rightarrow aB, B \rightarrow aS, B \rightarrow bA, B \rightarrow e.$$



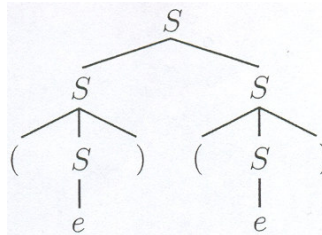
Context-Free Grammars and Languages

- Regular olmayan context-free diller vardır.
- Ancak bütün regular diller context-free dildir.
- Regular olan $\emptyset$ ve $\{a\}$ basit context-free dillerdir. Kuralı olmayan ve sadece $S \rightarrow a$ şeklinde kuralı olan dillerdir.
- Context-free diller union, concatenation ve Kleene star işlemleri için kapalıdır.
- Context-free diller pushdown otomatlar tarafından tanınır. Pushdown otomatlar finite otomatların genelleştirilmiş şeklidir.
- Her finite otomat basit yapıdaki pushdown otomat olarak düşünülür.



Parse Trees and Derivations

- G context-free grammar olsun. Bu grammar ile bir string'in oluşturulması farklı şekillerde olabilir.
- Dengeli parantez üreten bir context-free dil aşağıdaki farklı şekillerde aynı string'i üretir.
$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow () ()$$
$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow () (S) \Rightarrow () ()$$
- İki türetmede aşağıdaki şekilde gösterilebilir.



Bu şekil **parse tree** olarak adlandırılır.

- Her node V içinde bir nonterminal semboldür.
- Yapraklar (leaves) terminallerdir ve Σ içinde bir semboldür.
- Yapraklar soldan sağa concatenate edildiğinde oluşan string üretilir.



Parse Trees and Derivations

Bir context-free grammar $G = (V, \Sigma, R, S)$ için parse tree, roots, leaves aşağıdaki gibi tanımlanır.

1.

$\circ a$

Her $a \in \Sigma$ için bu bir parse tree'dir. Tek node hem root hemde yapraktır ve a oluşturur.

2.

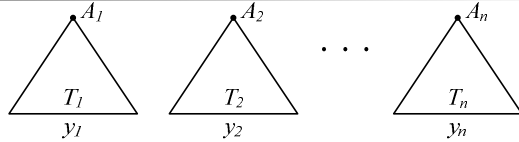


$A \rightarrow e$, R içinde bir kural ise bu bir parse tree'dir. A root ve e yapraktır. Sadece e üretir.

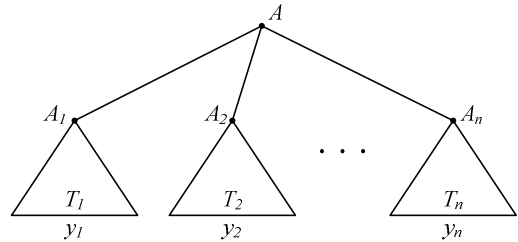


Parse Trees and Derivations

3.



Hepsi parse tree'dir. $n \geq 1$ için $A_1, \dots, A_n$ root ve $y_1, \dots, y_n$ üretilir. $A \rightarrow A_1, \dots, A_n$, R içinde kural ise aşağıdaki parse tree'dir. Üretilen string $y_1 \dots y_n$ olur.

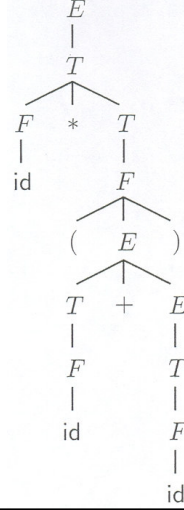


4. Bunların dışında hiçbir şey parse tree değildir.



Parse Trees and Derivations

Örnek: Aritmetik ifadeleri oluşturan grammar'ın $id * (id + id)$ için oluşturduğu parse tree aşağıdaki gibidir.



Parse Trees and Derivations

Bir context-free grammar $G = (V, \Sigma, R, S)$ için

$D = x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow x_n$ ve $D' = x'_1 \Rightarrow x'_2 \Rightarrow \dots \Rightarrow x'_n$ iki farklı türetmedir.

Burada $x_i, x'_i \in V^*$ ve $x_1, x'_1 \in V - \Sigma$ ve $x_n, x'_n \in \Sigma^*$

İki türetmede bir nonterminalden terminal string'leri türetilir.

D türetmesi D' türetmesinden öncedir ve $D \prec D'$ şeklinde gösterilir eğer;

$n > 2$ için $1 < k < n$ olacak şekilde bir k değeri varsa ve;

1. Tüm $i \neq k$ için $x_i = x'_i$
2. $x_{k-1} = x'_{k-1} = uAvBw$ burada $u, v, w \in V^*$ ve $A, B \in V - \Sigma$
3. $x_k = uyvBw$, burada $A \rightarrow y \in R$
4. $x'_k = uAvzw$, burada $B \rightarrow z \in R$
5. $x_{k+1} = x'_{k+1} = uyvzw$

En soldaki nonterminali değiştiren türetme diğerinden önce gelir.



Parse Trees and Derivations

Örnek: Herhangi bir grammar için aşağıdaki üç türetmenin önceliklerini çıkaralım.

$$D_1 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

$$D_2 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$$

$$D_3 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow ((S))() \Rightarrow (())()$$

Burada $D_1 \prec D_2$ ve $D_2 \prec D_3$ olur. $D_1 \prec D_3$ olamaz çünkü birden fazla ara string'te farklılık vardır. Bütün türetmeler aynı parse tree'ye sahiptir.



Parse Trees and Derivations

Örnek: devam

$$D_4 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$$

$$D_5 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow ((S))() \Rightarrow (())()$$

$$D_6 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

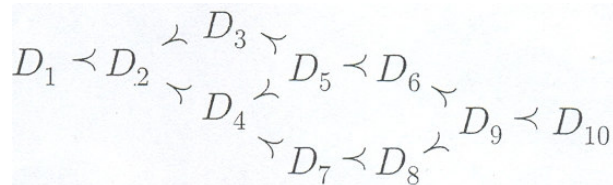
$$D_7 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$$

$$D_8 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow ((S))() \Rightarrow (())()$$

$$D_9 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

$$D_{10} = S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

Buradaki tüm öncelik ilişkileri aşağıdaki şekilde gösterilebilir.



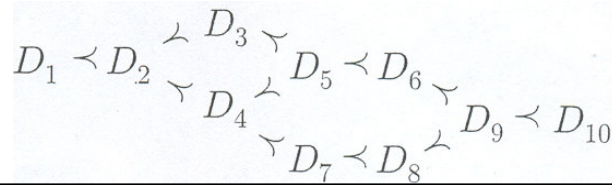


Parse Trees and Derivations

Bir parse tree üzerinde **leftmost derivation** ve **rightmost derivation** elde edilebilir.

Leftmost derivation için ağacın root node'undan başlanır ve sürekli en soldaki nonterminal değiştirilir.

Rightmost derivation için ağacın root node'undan başlanır ve sürekli en sağdaki nonterminal değiştirilir. Önceki örnekte D_1 leftmost D_{10} ise rightmost derivation yapar.



Parse Trees and Derivations

Leftmost derivation için $x \Rightarrow^L y$ ve rightmost derivation için $x \Rightarrow^R y$ kullanılır.

$x \Rightarrow^L y$ yazabiliriz eğer sadece ve sadece $x = wA\beta$, $y = w\alpha\beta$, ise ve burada $w \in \Sigma^*$ ve $\alpha, \beta \in V^*$ ve $A \in V - \Sigma$ ve $A \rightarrow \alpha$ kuralı grammar'de varsa.

$x_1 \Rightarrow^L x_2 \Rightarrow^L \dots \Rightarrow^L x_n$ tüm leftmost türetme sırasını ifade eder.



Ödev

- Problemleri çözünüz 3.1.1b, 3.1.1c, 3.1.1c (sayfa 120)
- Problemleri çözünüz 3.1.2, 3.1.3 (sayfa 120)
- Problemleri çözünüz 3.1.8 (sayfa 121)
- Problemleri çözünüz 3.2.4 (sayfa 129)