DERLEYİCİ TASARIMI

Bu Haftaki Konu Başlıkları

- **Syntax Analysis**
- **Gramer**
- **İçerik-Bağımsız Gramerler (CFG)**
- **Bir gramerin tanımlanması**
- **CFG'ler Nasıl Yazılır?**
- **Türetim (Derivation)**
- **Syntax Ağaçları**

Syntax Analysis

- Lexical Analizin amacı karakter akışını tokenlere ayırmak iken, **syntax analizin (parsing)** amacı ise bu tokenleri tekrar bir araya getirmektir. Ancak bu dönüşümü bir karakterler listesi olarak değil metnin yapısını yansıtacak şekilde gerçekleştirir. Bu genelde, bir metnin **syntax ağacı** olarak adlandırılan bir veri yapısıdır. Bu yapı isminden de anlaşıldığı gibi bir ağaç veri yapısıdır. Bu ağacın yaprakları lexical analiz tarafından bulunan token'lardır ve eğer yapraklar soldan sağa okunursa, giriş metninin okunması ile aynı sırada olur. Bu nedenle, syntax ağacında önemli nokta, bu yaprakların ağaç üzerinde nasıl bir araya getirildikleri ve ağacın iç düğümlerinin nasıl etiketlendikleridir.
- Giriş metninin yapısını bulmanın yanında syntax analiz, syntax hataları üreterek geçersiz metinleri de kabul etmemelidir.

Syntax Analysis

- Syntax analiz lexical analize göre daha da gelişmiş yöntemler gerektirir. Ancak benzer strateji kullanılır. İnsanların anlayabileceği uygun bir gösterim etkin şekilde çalıştırılabilecek makineye yakın düşük seviyeli bir gösterime çevrilir. Bu süreç **parser generation** olarak adlandırılır.

- İnsanların anlayabileceği uygun gösterim yöntemi **içerik-bağımsız gramerler** (context-free grammars)'dir. Bu yöntem string'in kümelerini tanımlayan yineleyici (recursive) bir gösterimdir. Böyle bir gösterim bazı durumlarda doğrudan rekürsif programlara dönüştürülebilir ancak **yığınlı otomata (stack automata)** çevirmek daha uygundur. Bu otomatlar lexical analizde kullanılan otomatlara benzerler ancak ek olarak bir yığın kullanırlar. Bu yığın otomatlar tarafından okunan sembollerin sayılmasını sağlar. Bu otomatlar iki şekilde oluşturulacaktır.
  - İlk olarak kıyasla basit olan LL(1) yöntemi ile gerçekleştirilecektir ancak bu yöntem sadece kısıtlı sayıda gramer sınıfı için uygundur.
  - İkincisi SLR yapısıdır. Bu yöntem daha karmaşıktır ancak daha geniş bir gramer sınıfı için uygundur. Yine de her ikisi de tüm içerik-bağımsız gramer sınıfları için yeterli değillerdir.

Gramer

- **Gramerler,** bir programlama dilinde programların yapısını tanımlamak için gerekli şekillendirmedir. Uygulamada bir dilin grameri yalnızca söz dizimi yapısını tanımlamasına rağmen bir de anlamların tanımlamasında kullanılır.

- Uygulamada farklı gramer tipleri de vardır. Derleyici tasarımında başlıca CF gramerler kullanılır. Gramerin diğer tipleri derleyici tasarımında önemli bir rol oynamazlar.

- Bir gramer sembol katar gruplarının yapılanmasını sağlayan bir planlamadır. Programlama dilleri için uygulandığında dildeki tokenler sembolleri, sembol stringleri, program metnini ve string grupları programlama dilini ifade eder.

Gramer

- Bir gramer bir dizi üretim kuralından ve bir başlama sembolünden oluşur. Her bir üretim kuralı isimlendirilmiş bir sözdizimi yapısı tanımlar. Bir üretim kuralı iki parçadan oluşur;
  - sol taraf (left-hand side) ve
  - sağ taraf (right-hand side).

  Bu iki parça soldan sağa bir ok ile ayrılmıştır. Sol taraf söz dizimi yapısının adıdır, sağ taraf söz dizimi yapısının mümkün olan şekillerini gösterir.

```
expression → '(' expression operator expression ')'
```

İçerik-Bağımsız Gramerler (CFG)

- Kurallı ifadeler gibi, CFG'ler de string kümelerini tanımlar. Ek olarak CFG'ler aynı zamanda dilin içinde tanımlanan stringlerin yapısını da tanımlarlar. Alfabedeki semboller **terminaller (uç-semboller)** olarak adlandırılır. Terminal sembol üretim sürecinin son noktasıdır ve gramer tarafından üretilen kataın parçası olabilir.

- Bir CFG yinelemeli olarak stringlerin birkaç kümesini tanımlar. Her bir küme **non-terminal** olarak adlandırılan bir isimle simgelenir. Non-terminallerin kümesi terminaller kümesinden elde edilir. Non-terminallerden birisi gramer tarafından tanımlanan dili simgelemek için seçilir. Bu sembol başlangıç sembolü olarak adlandırılır. Her bir üretim bir non-terminal tarafından simgelenen kümedeki olası stringlerin bazılarını tanımlar.

İçerik-Bağımsız Gramerler (CFG)

Bir (V,T,S,P) gramerinde tüm türetim kuralları (P) , A ϵ V ve x ϵ (V U T)* olmak üzere A → x seklinde ise baglamdan(içerik) bagımsızdır. Yani baglamdan bagımsız her gramer kuralının sol tarafında tek bir degisken vardır.

Bir üretim aşağıdaki yapıya sahiptir:

$$N \rightarrow X_1 \ldots X_n$$

Burada, N bir non-terminaldir ve $X_1 \ldots X_n$ her biri ya bir terminal ya da non-terminal olan sembollerdir. Bu kümede bir terminal tek bir elemana sahip bir kümeyi simgeler. Örneğin;

$$A \rightarrow \mathrm{a}$$

ifadesi A non-terminal simgesi tarafından simgelenen küme bir karakter stringi **a**'yı içerir. **a** simgesi burada terminaldir.

İçerik-Bağımsız Gramerler (CFG)

$$A \rightarrow \mathtt{a}$$

$$A \rightarrow \mathtt{a}A$$

İfadesi A tarafından simgelenen küme A ile simgelenen kümeden alınan bir stringin önüne **a** konulmasıyla biçimlendirilen tüm stringleri içerir. Bu iki ifadede A boş olmayan **a**'ların sırasını içerir ve bu nedenle kurallı ifade **a**$^{+}$'ya eşittir.

Kurallı ifade **a***'a eşit bir gramer ise aşağıdaki iki üretim tarafından tanımlanabilir.

$$B \rightarrow$$

$$B \rightarrow \mathtt{a}B$$

Burada ilk ifade de B kümesinin bir parçasının boş olabileceği görülmektedir. Sağ tarafları boş olan üretimler **boş-üretimler (empty productions)** olarak adlandırılırlar. Bu tür üretimlerde sağ taraf boş bırakılmak yerine ε da kullanılabilir.

İçerik-Bağımsız Gramerler (CFG)

Örneğin $\{a^n b^n | n \geq 0\}$ kurallı bir dil değildir. Bu dil kolaylıkla gramer yoluyla aşağıdaki gibi ifade edilebilir.

$$S \rightarrow$$
$$S \rightarrow \mathrm{a}S\mathrm{b}$$

İkinci üretim **a**'ların ve **b**'lerin ortadaki string'in çevresinde simetrik olarak üretilmelerini sağlar. Burada **a** ve **b** sayısı eşit olarak oluşur.

Şimdiye kadarki örneklerde her gramer için bir non-terminal kullanıldı. Birkaç non-terminal kullanıldığı zaman hangisinin başlangıç sembolü olduğu açık hale getirilmelidir. Bir başkası belirtilmediyse, ilk üretimin sol tarafındaki non-terminal sembol başlangıç sembolü olarak seçilir.

İçerik-Bağımsız Gramerler (CFG)

Örnek olarak aşağıdaki gramerin

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathtt{a}T\mathtt{a} \\
R &\rightarrow \mathtt{b} \\
R &\rightarrow \mathtt{b}R
\end{aligned}
$$

başlangıç sembolü T 'dir ve bir veya daha fazla **b'**lerden oluşan string veya başta ve sonda eşit sayıda ve en az bir a, ortada ise en az bir b bulunduran string kümesini simgeler. Bazen kısa bir gösterim bir non-terminalin tüm üretimlerinin tek bir kural altında birleştirilmesinde kullanılabilir. Bunun için **|** sembolü kullanılır.

İçerik-Bağımsız Gramerler (CFG)

Bu gösterime göre yukarıdaki grameri tekrar yazacak olursak;

$$
\begin{aligned}
T &\rightarrow R \mid \mathtt{a}T\mathtt{a} \\
R &\rightarrow \mathtt{b} \mid \mathtt{b}R
\end{aligned}
$$

grameri elde edilir. Gramerden elde edilen dizgilere örnek verelim:

b, bb, bbb, bbb…, aba, abba, abbba, aabaa, aabbaa, aabbbaa…

İçerik-Bağımsız Gramerler (CFG)

Bir G CFG tarafından oluşturulan dil L(G) ile gösterilir. CFL'de (Bağlamdan Bağımsız Dil) başlangıç sembolü ile başlanır. Herhangi bir türetim kuralı birçok kez kullanılabilir. Sonuçta elde edilen dizgi sadece sembollerden oluşur.

Örnek:

Bir G=({S},{a,b}, S, P) grameri

S→ aSa

S→ bSb

S→ λ

türetim kuralları ile bağlamdan bağımsızdır.

Bu gramere ait tipik bir türetim:

S=>aSa=>aaSaa=>aabSbaa=>aabbaa

$L(G)=\{ww^R : w \in \{a,b\}^*\}$ (Palindrome)

İçerik-Bağımsız Gramerler (CFG)

N_i için üretimler s_i'nin şekline bağlı olarak aşağıdaki tablo üzerinde gösterilmiştir.

Form of s_i	Productions for N_i
ε	$N_i \rightarrow$
a	$N_i \rightarrow$ a
$s_j s_k$	$N_i \rightarrow N_j N_k$
$s_j \mid s_k$	$N_i \rightarrow N_j$ $N_i \rightarrow N_k$
s_j*	$N_i \rightarrow N_j N_i$ $N_i \rightarrow$
s_j+	$N_i \rightarrow N_j N_i$ $N_i \rightarrow N_j$
$s_j?$	$N_i \rightarrow N_j$ $N_i \rightarrow$

İçerik-Bağımsız Gramerler (CFG)

Aşağıda bir gramer yapısı verilmiştir. Burada terminal olan semboller tırnak işareti ile ayrılmıştır. Terminal olmayanlar ise tanımlayıcılardır. Başlangıç sembolü expression 'dır.

```
[1]    expression → '(' expression operator expression ')'
[2]    expression → '1'
[3]    operator → '+'
[4]    operator → '*'
```

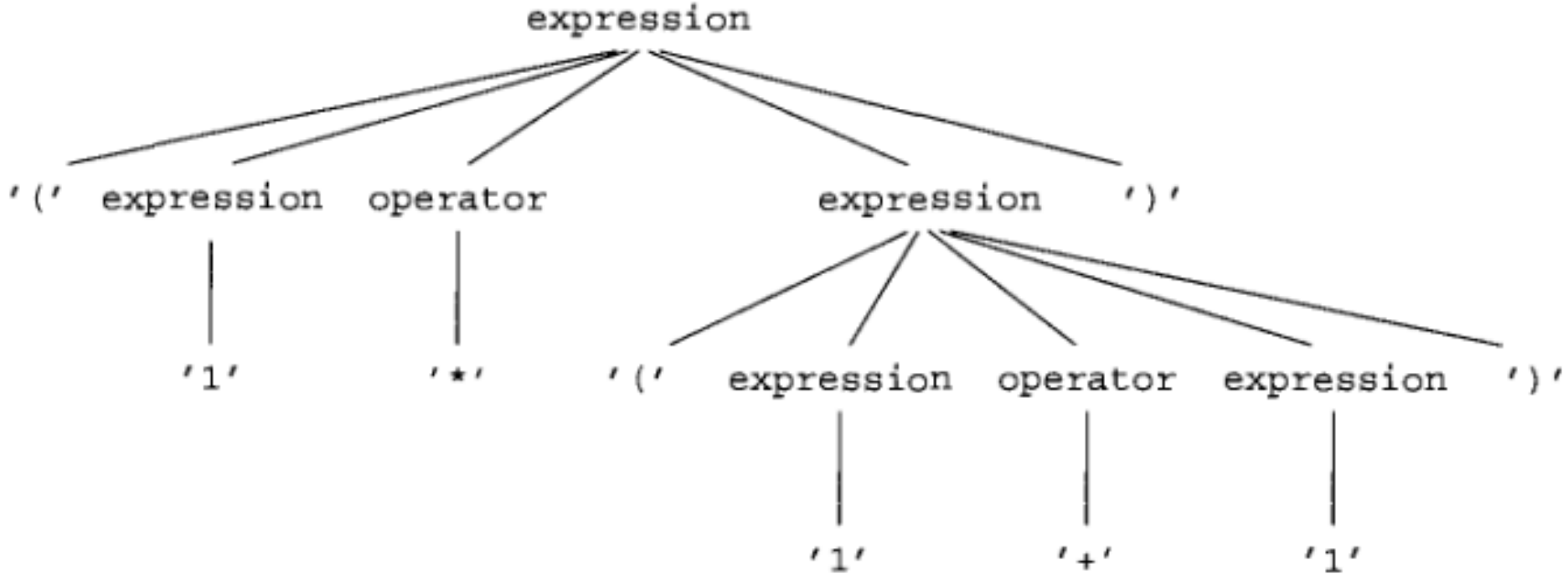
İçerik-Bağımsız Gramerler (CFG)

Cümlesel yapının sırası aşağıdaki şekilde gösterilmiştir. Yapı (1*(1+1)) 'den türetilmiştir. Bu bir soldan türetmedir. R@P'nin anlamı P durumda terminal olmayan sembolü tekrar yazmak için kullanılan gramer kuralı R'yi ifade eder.

```
          expression
1@1       '(' expression operator expression ')'
2@2       '(' '1' operator expression ')'
4@3       '(' '1' '*' expression ')'
1@4       '(' '1' '*' '(' expression operator expression ')' ')'
2@5       '(' '1' '*' '(' '1' operator expression ')' ')'
3@6       '(' '1' '*' '(' '1' '+' expression ')' ')'
2@7       '(' '1' '*' '(' '1' '+' '1' ')' ')'
```

İçerik-Bağımsız Gramerler (CFG)

Üretim süreci program metni ile birlikte üretim ağacı üretecek kadar çeşitlidir. Programın anlamlarını bulmada ağaca ihtiyaç duyduğumuz için bu işlemi gerçekleştiren '**ayrıştırıcı (parser)**' adında özel bir program kullanırız.



Bir gramerin tanımlanması

Bir CF gramer $\mathbf{G = (V_N, V_T , S, P)}$ biçiminde tanımlanır. Burada;

$\mathbf{V_N}$ ve $\mathbf{V_T}$ sembol grupları, **S** bir sembol ve **P** üretim kurallarıdır. $\mathbf{V_N}$'nin elemanları terminal olmayan semboller olarak adlandırılır. $\mathbf{V_T}$'ninkiler ise terminal semboller ve S başlangıç sembolüdür.

Bu yapı yalnızca CF gramer yapısını tanımlar. Bunu gerçekleştirmek için yapı üç içerik durumunu yerine getirmelidir.

$$V_N \cap V_T = \varnothing$$

$$S \in V_N$$

$$P \subseteq \{(N, \alpha) \mid N \in V_N, \alpha \in (V_N \cup V_T)^*\}$$

Terminal olmayanlar A, B, C, ve N gibi büyük harflerle, Terminaller ise x, y, z gibi küçük harflerle ifade edilir.

Türetim (Derivation)

Örnek: Bir CFG'nin tanımlanması aşağıda yapılmıştır :

S --> id = E

E --> T E'

E' --> + T E' | - T E' | ε

T --> F T'

T' --> * F T' | / F T' | ε

F --> id | num | - F | (E)

CFG 'nin belirtimleri :

Non-Terminaller{ S , E , E', T, T', F }

Terminaller { id , = , + , - , * , / , num , (,) ,ε }

Başlangıç sembolü: S

CFG'ler Nasıl Yazılır?

Bir kurallı ifade sistematik olarak kurallı ifadedeki her alt ifade için bir nonterminal kullanılarak ve her bir nonterminal için bir ya da iki üretimin kullanılmasıyla CFG olarak yeniden yazılmış oldu. Bu durumda, bir dil bir kurallı ifade olarak tanımlanabilirse onu gramere çevirme işlemi kolaylaşır. Ancak, gramerler kurallı olmayan dilleri tanımlamak için de kullanılabilir.

CFG'ler Nasıl Yazılır?

Programlama dillerindeki bir çok yapı CFG'ler yoluyla kolaylıkla tanımlanabilir. Gerçekte birçok modern dil şu şekilde tanımlanır:

Bir programlama dili için bir gramer yazıldığı zaman, farklı **anlamsal (syntactic) kategoriler** içerisinde dilin yapılarının ayrılması ile başlar. Bir anlamsal kategori özel bir anlamı oluşturan alt bir dildir. Programlama dillerindeki genel anlamsal kategorilerin örnekleri:

- **İfadeler (Expressions)** değerlerin hesaplanmasını ifade etmek için kullanılırlar.
- **Deyimler (Statements)** özel bir sırada oluşan işlevleri ifade ederler.
- **Bildirimler (Declaration)** programın diğer parçalarında kullanılan isimlerin özelliklerini ifade ederler.

Regular grammar

Sagdan ve Soldan Lineer Gramerler:

Bir G=(V, T, S, P) grameri A, B ϵ V ve x ϵ T* olmak üzere tüm türetim kuralları A → xB | x seklinde ise sagdan lineerdir.

G gramerinin soldan lineer olması için tüm türetim kuralları

A → Bx | x seklinde olmalıdır.

- Düzgün bir gramer ya sagdan ya da soldan lineerdir.

Linear grammar

- Bir gramer sağdan ya da soldan lineer olmadan da lineer olabilir.

- Example:

  $S \rightarrow aS \mid A$

  $A \rightarrow Ab \mid \lambda$

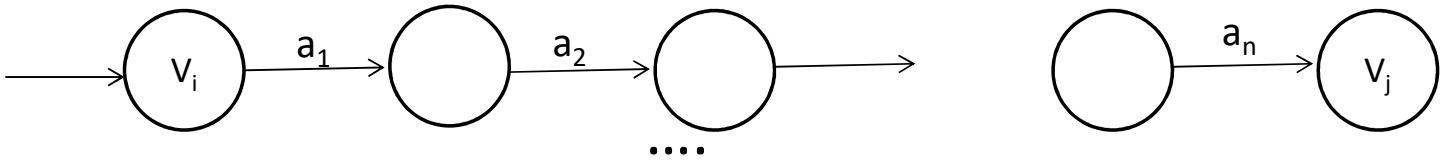
Bu gramer ‘lineer gramer’e örnektir. Lineer gramer türetim kuralının sağ tarafında en fazla bir değişken olan gramerdir. Bu değişkenin yeri önemli değildir. Bu durumda düzgün bir gramer her zaman lineerdir, ancak bütün lineer gramerler düzgün değildir.

Linear grammar

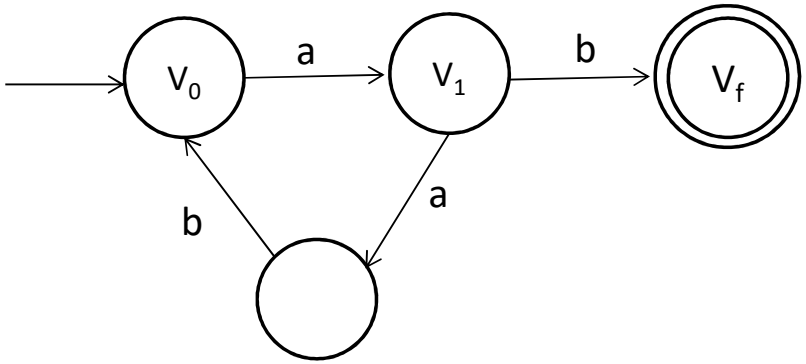
Sağdan lineer bir gramer tarafından türetilen dil her zaman düzgündür. Örneğin ab…cD => ab…cdE gibi bir türetimi D→dE türetim kuralı ile elde ettiğimizi varsayalım. Buna ait NFA'da gösterim D durumunda iken d sembolünü alarak E'ye gider. Bu durum aşağıdaki teoremin temelidir.

Teorem: G = (V, T, S, P) sağdan lineer bir gramer olsun. Bu durumda L(G) düzgün bir dildir. (Soldan lineer gramer için de geçerli)

$V_i \rightarrow a_1\ a_2 \ldots\ldots a_n V_j$



Linear grammar



- Örnek: $V_0 \rightarrow aV_1$

  $V_1 \rightarrow abV_0 \mid b$ grameri ile türetilen dili kabul eden sonlu otomatı gösterin.

Bu gramer tarafından türetilen ve otomat tarafından kabul edilen dil L ((aab)*ab)

Örnek: ∑ = {a, b} alfabesi üzerinde en fazla üç a içeren tüm dizgileri kapsayan dili oluşturan düzgün bir gramer bulun.

$S \rightarrow bS \mid aA \mid \lambda$

$A \rightarrow bA \mid aB \mid \lambda$

$B \rightarrow bB \mid aC \mid \lambda$

$C \rightarrow bC \mid \lambda$

Örnek: Çift uzunluktaki dizgileri içeren dili oluşturan düzgün grameri bulun.

$S \rightarrow aaS \mid abS \mid baS \mid bbS \mid \lambda$

Example

Verilen bir gramerden bu gramerin hangi dili ürettiğini söyleyebilmeliyiz:

$S \rightarrow aaSB \mid \lambda$

$B \rightarrow bB \mid b$

It helps to list some of the strings that can be formed:

$S \Rightarrow aaSB \Rightarrow aaB \Rightarrow aab$

$S \Rightarrow aaSB \Rightarrow aaB \Rightarrow aabB \Rightarrow aabb$

$S \Rightarrow aaSB \Rightarrow aaB \Rightarrow aabB \Rightarrow aabbB \Rightarrow aabbb$

$S \Rightarrow aaSB \Rightarrow aaB \Rightarrow aabB \Rightarrow aabbB \Rightarrow aabbbB \Rightarrow aabbbb$

$S \Rightarrow aaSB \Rightarrow aaaaSBB \Rightarrow aaaaBB \Rightarrow aaaaBb \Rightarrow aaaabb$

$S \Rightarrow aaSB \Rightarrow aaaaSBB \Rightarrow aaaaBB \Rightarrow aaaaBbB \Rightarrow aaaaBbb \Rightarrow aaaabbb$

What is the pattern? Is it:

$L = \{\lambda + ((aa)^n b^n) b^*\}$, for $n >= 1$

- Verilen bir dilden, bu dili üreten gramerin bulunması:

Örnek: {a, b, c} üzerinde, a ile başlayan, sadece iki b içeren ve cc ile biten tüm dizgileri içeren dile ait düzgün (sağdan lineer) bir gramer bulun.

$S \rightarrow aA$

$A \rightarrow bB \mid aA \mid cA$

$B \rightarrow bC \mid aB \mid cB$

$C \rightarrow aC \mid cC \mid cD$

$D \rightarrow c$

Örnek:

L={$a^n b^m$: n≠m} dilinin bağlamdan bağımsız olduğunu göstermek için bu dil için bağlamdan bağımsız bir gramer (CFG) bulmalıyız.

İlk olarak n>m durumunu düşünelim. Eşit sayıda a ve b türetilir ve sol tarafa ekstradan a'lar eklenir.

$S \rightarrow AS_1$

$S_1 \rightarrow aS_1b \mid \lambda$

$A \rightarrow aA \mid a$

Aynı şey n<m için de düşünülür:

$S \rightarrow S_1B$

$S_1 \rightarrow aS_1b \mid \lambda$

$B \rightarrow bB \mid b$

Sonuç olarak:

$S \rightarrow AS_1 \mid S_1B$

$S_1 \rightarrow aS_1b \mid \lambda$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Bu gramer CFG'dir fakat lineer değildir.

Türetim (Derivation)

Gramer-3'e bakarsak aşağıda gösterilen türetim yoluyla aabbbcc stringini üretir. Burada sembollerin her bir sırasında nonterminallerin altı çizilmiştir.

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathtt{a}T\mathtt{c} \\
R &\rightarrow \\
R &\rightarrow R\mathtt{b}R
\end{aligned}
$$

Gramer-3

$$
\begin{aligned}
&\underline{T} \\
\Rightarrow\ & \mathtt{a}\underline{T}\mathtt{c} \\
\Rightarrow\ & \mathtt{aa}\underline{T}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}R\mathtt{b}\underline{R}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{bcc} \\
\Rightarrow\ & \mathtt{aa}R\mathtt{b}\underline{R}\mathtt{bcc} \\
\Rightarrow\ & \mathtt{aa}R\mathtt{b}\underline{R}\mathtt{b}R\mathtt{bcc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{bb}R\mathtt{bcc} \\
\Rightarrow\ & \mathtt{aabb}\underline{R}\mathtt{bcc} \\
\Rightarrow\ & \mathtt{aabbbcc}
\end{aligned}
$$

Gramer-3'ün türetilmesiyle aabbbcc stringinin elde edilmesi

Türetim (Derivation)

$$
\begin{aligned}
& \underline{T} \\
\Rightarrow\ & \mathtt{a}\underline{T}\mathtt{c} \\
\Rightarrow\ & \mathtt{aa}\underline{T}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{b}R\mathtt{cc} \\
\Rightarrow\ & \mathtt{aa}\underline{R}\mathtt{b}R\mathtt{b}R\mathtt{cc} \\
\Rightarrow\ & \mathtt{aab}\underline{R}\mathtt{b}R\mathtt{cc} \\
\Rightarrow\ & \mathtt{aab}\underline{R}\mathtt{b}R\mathtt{b}R\mathtt{cc} \\
\Rightarrow\ & \mathtt{aabb}\underline{R}\mathtt{b}R\mathtt{cc} \\
\Rightarrow\ & \mathtt{aabbb}\underline{R}\mathtt{cc} \\
\Rightarrow\ & \mathtt{aabbbcc}
\end{aligned}
$$

Gramer-3'ün Soldan türetilmesiyle **(Leftmost derivation)** aabbbcc stringinin elde edilmesi

Türetim (Derivation)

Bu türetimde, türetim en soldaki nonterminalin kullanılması ile gerçekleştirilmiştir. Aynı yöntem bazen en sağdaki nonterminalin **(Rightmost derivation)** kullanılması ile ya da aradan her hangi birisinin kullanılması ile de gerçekleştirilebilir.

- **Ensol Türetim (leftmost derivation):** Eğer her bir türetim adımında ensol nonterminal yer değiştirilirse buna ensol türetim denir

- **Ensağ Türetim (Rightmost derivation):** Eğer her bir türetim adımında ensağ nonterminal yer değiştirilirse buna ensağ türetim denir.

Örnek: S→ aAB

A→ bBb

B→ A | λ

türetim kurallarıyla verilmiş gramer için abbbb dizgisi;

Soldan türetme:

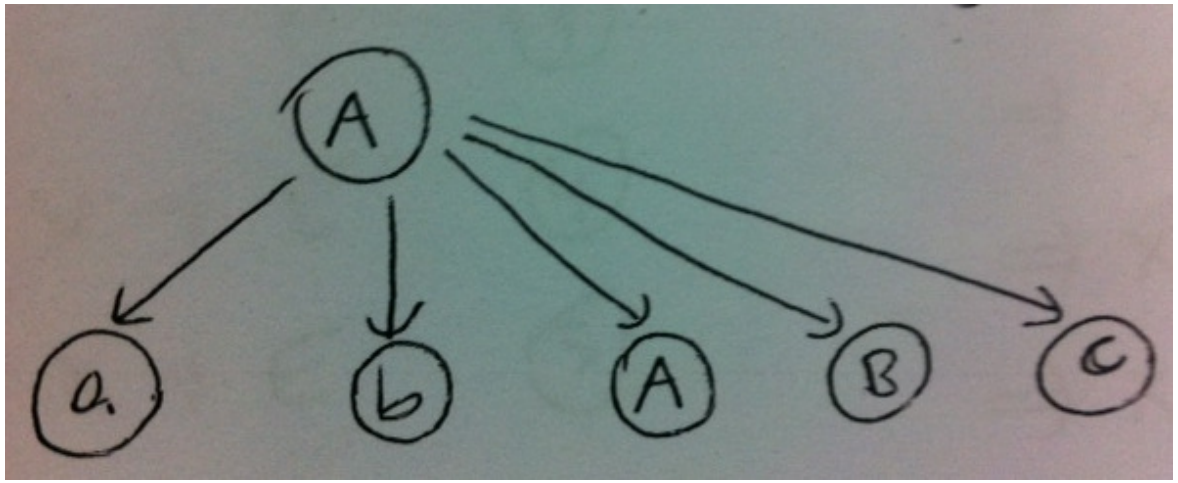
S=>aAB=>abBbB=>abAbB=>abbBbbB=>abbbbB=>abbbb

Sağdan türetme:

S=>aAB=>aA=>abBb=>abAb=>abbBbb=>abbbb

Türetim kurallarının kullanılma sırasından bağımsız bir şekilde türetimleri göstermenin diğer bir yolu da türetim ağaçları kullanmaktır.

Örneğin; A→ abABc kuralını gösteren türetim ağacı:



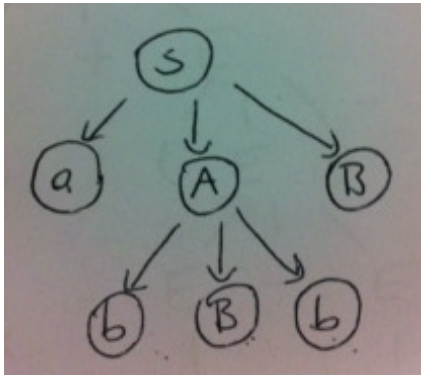
- Örneğin;

  S → aAB

  A → bBb

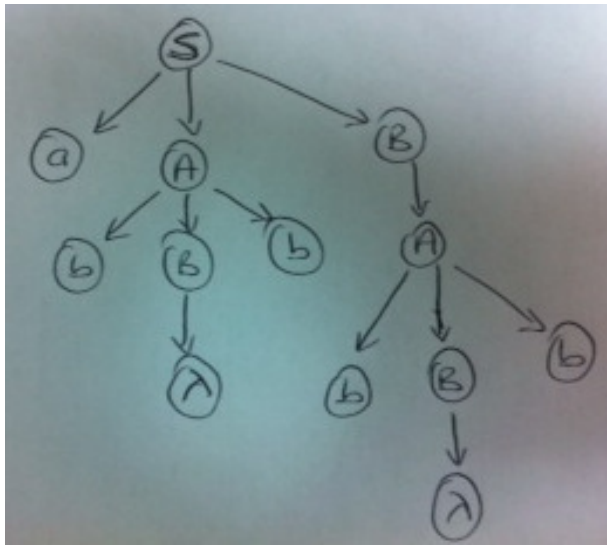
  B → A | λ

türetim kurallarıyla verilen G grameri için;



Kısmi türetim ağacı
abBbB

Türetim Ağacı:



abbbb

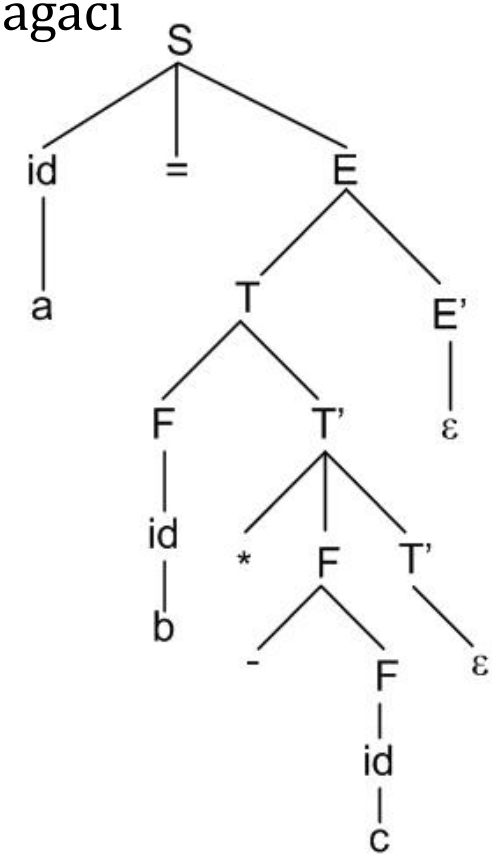
dizgisi için

Türetim (Derivation)

Türetimler : **a = b * - c** ifadesi için yapılacak

S --> id =E
E -->T E'
T --> F T'
F -->id
T' --> * F T'
F --> - F
F --> id
T' --> ε
E' --> ε

Parse Ağacı: Bu üretimler sonucu elde edilen parse ağacı



Syntax Ağaçları

Let G = (V, T, S, P) be a context-free grammar. An ordered tree is a *derivation tree* for G iff it has the following properties:

1. The root is labeled S
2. Every leaf has a label from $T \cup \{\lambda\}$
3. Every interior vertex (not a leaf) has a label from V.
4. If a vertex has label $A \in V$, and its children are labeled (from left to right) $a_1, a_2, \ldots, a_n$, then P must contain a production of the form $A \rightarrow a_1 a_2 \ldots a_n$
5. A leaf labeled λ has no siblings; that is, a vertex with a child labeled λ can have no other children

Syntax Ağaçları

Bir ağaç olarak bir türetim yapıldığında ağacın kökü gramerin başlangıç sembolüdür ve ne zaman bir nonterminal tekrar yazılsa kullanılan üretimin sağ tarafındaki sembolleri onun çocuğu olarak eklenir. Ağacın yaprakları okunan string'ten soldan sağa alınan terminallerdir. Eğer bir nonterminal bir boş üretimin kullanılması ile tekrar yazılırsa, bir ε onun çocuğu olarak gösterilir. Bu aynı zamanda yaprak bir düğümdür ancak ağacın yaprakları okunurken ihmal edilir.

Bu şekilde bir syntax ağacı yazıldığı zaman türetimin sırası ile ilgilenilmez. Soldan türetim, sağdan türetim ya da herhangi bir türetim için aynı ağaç elde edilebilir. Aşağıdaki örnek gramer 3 için aynı stringi elde eden iki farklı syntax ağacını göstermektedir.

Syntax Ağaçları

Gramer-3'ün kullanılmasıyla üretilen aabbbcc stringinin syntax ağaçları

Gramer-3

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathtt{a}T\mathtt{c} \\
R &\rightarrow \\
R &\rightarrow R\mathtt{b}R
\end{aligned}
$$

Bu ağaç okunan string'in yapısını yansıtır ve derleyici tasarımının sonraki evrelerinde kullanılır.