YOZGAT BOZOK ÜNİVERSİTESİ

Programlama Dilleri Prensipleri

Dr. Öğr. Üyesi R. Sinan ARSLAN

Giriş

- İfade Notasyonları
- Soyut Sözdizim Ağaçları
- Metinsel Sözdizim
- Dilbilgisi
- Dilbilgisi Çeşitleri
- Sözdizim Grafikleri

Dillerin Tanımlanması ve Tasarımı

- Bir programlama dilinde yazılmış programın geçerli olup olmadığını belirleyen kurallar ile insanların iletişimi sırasındaki kurallar arasında benzerlikler bulunur.
- Bir insan konuşurken, kullandığı harflerin o dilin alfabesinde olup olmadığı kontrol edilir.
- Bu harfler o dilin kurallarına göre bir araya getirilerek mi kelimeler türetilmiştir.
- Kelimeler o dile özgü grammer kuralları kullanılarak mı cümleler oluşturulmuştur?
- Bu sıralama, bir program metninin oluştururken de işletilir.
- Dolayısı ile, bir programlama dilinin alfabesi olmalıdır. bu karakterler bir araya getirilerek kelime grupları oluşturulmalıdır.

Dillerin Tanımlanması ve Tasarımı

- örnek olarak, değişken tanımlamasında belli kurallar var ise bu kurallar dahilinde bir araya getirilmelidir.
- program metninde kullanılan kelimelerin belli kurallara uygun türetilmiş olduğunun kontrolü için araçlar tasarlanmalıdır. (regular exp, dfa, nfa vb.)
- Bu kelimelerin bir araya getirilerek oluşturulacak deyim listesinin BNF, CFG yapıları ile test edilmesini sağlayan araçları bulundurması gereklidir.
- Program metninin anlamlı olması gereklidir.

Dillerin Tanımlanması ve Tasarımı

- Yazılmış bir programın geçerli olup olmadığını anlamak için sözdizim analizi ve anlamsal analiz yapılmalıdır.
  - bir programın begin ile başlayıp end ile bitmesi veya her deyim sonuna noktalı virgül konulması sözdizim kurallarını ifade eder.
  - bir değişkenin kullanılmadan önce tanımlanması, tanımlanan değişkenin bir tip ile bağlanması, sadece o tipte veriler alabilmesi anlamsal kurallara örnektir.
- sözdizim kuralları dil içinde aynı olmak ile birlikte anlamsal farklılıklar ortaya çıkabilir.
  - 01/02/2020 tarihi Türkiyede 1 şubat 2020 iken ABD'de 2 ocak 2020'yi ifade eder.

Tanımlar

- Her dil sonlu sayıda simgeden oluşan belirli bir alfabe üzerinde tanımlanır.
- Alfabedeki simgeler art arda gelerek karakter dizilerini yani stringleri oluşturur.
- her biçimsel (formal) dil bu katarların bir kümesidir.
- her programlama dilinin yazılan bir programın syntax olarak doğru olup olmadığını belirleyen kuralları vardır.
- Programlama dillerinin sözdizimsel(syntactic) yapısı Contex-Free Grammer(CFG) / Backus-Naur Form(BNF) ile ifade edilir.
- Bu araçlar dilin hangi tümcelerden oluştuğunu gösteren bir kurallar bütünüdür.
- Grammerler programlama dilinin sözdizimsel yapısının kolayca anlaşılmasına ve yeni program yapılarının mevcut yapılara kolayca eklenmesine olanak sağlar.
- Grammerler, sözdizim kurallarını belirleme de yeterlidir ancak bu yapı anlam kurallarını belirlemede yeterli olmaz.

İfade Notasyonları

- Programlama dillerinde a + b * c ifadelerin yüksek düzeyli programlamayı gösterir.
- (-b + $\sqrt{b^2 - 4 * a * c}$) / (2*a) ifadesini yüksek seviyeli bir dil ile (-b + sqrt(b*b – 4.0*a*c))/(2.0*a) olarak ifade edebiliriz.
- Bu şekilde ifadeler sözdizim ağaçları sayesinde ayıklanarak anlamlandırılır.

Soyut Sözdizim Ağaçları

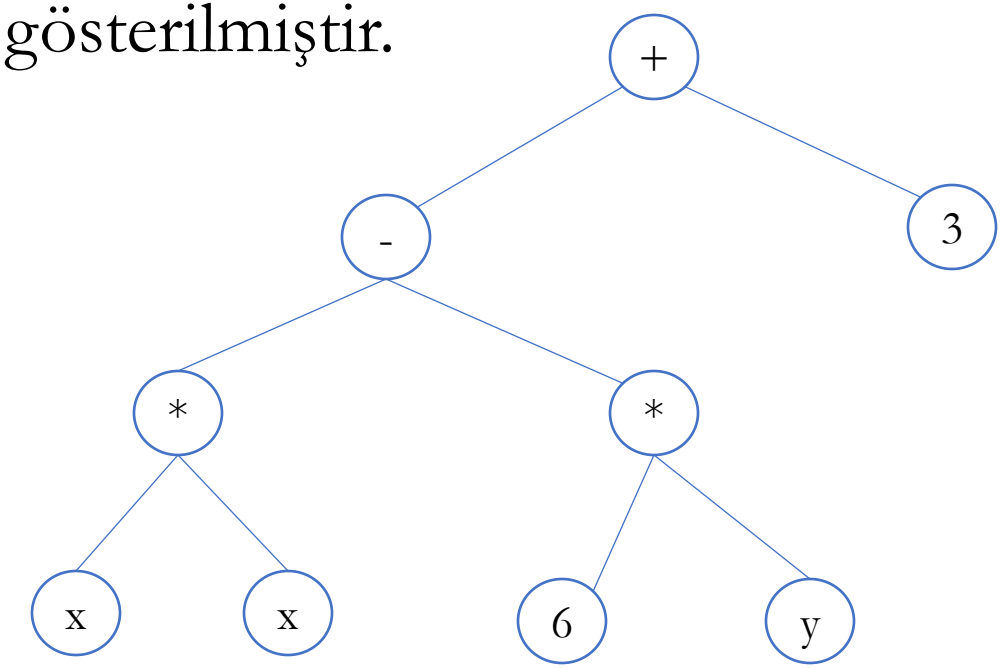
- Bir dilin soyut söz dizimi, dildeki her yapının anlamlı bir açıklamasıdır.
- Prefix, infix ve postfix notasyonları vardır.
  - Prefix : +xy
  - Infix : x+y
  - Postfix: xy+
- + işareti operatörü, x ve y değeri operand (ifadeleri) gösterir.

İfadelerin Sözdizim Ağaçları ile Gösterimi

- A-B gibi bir aritmetik ifade de A ve B'nın sırası çok önemlidir. Bu sebeple A+B gibi bir ifadeyi ağaç üzerine yerleştirirken sol çocuk ve sağ çocuk düğümün belirlenmesi önemlidir.
- İkili bir aritmetik ifade ağacında 0,1 ve 2 çocuk bulunabilir.
- Bilgisayar bilimlerinde ikili ağaç yapıları verileri organize etmek ve algoritmaları tanımlamak için kullanılır.

İfadelerin Ağaç ile Gösterimi-Örnek

- x*x -6*y+3 ifadesini ifade ağacı halinde gösterimi aşağıda gösterilmiştir.



Soru: Farklı matematiksel işlemlerin ifade agaçlarını çizmek ile ilgili örnekler çözünüz.

a*b-c/d+f ifadesinin söz dizim ağacını çiziniz.

Bir ifadenin operatör ve operand yapısını gösteren ağaçlara soyut sözdizim ağaçları denir.

Metinsel(Lexical) Analiz

- Programlama dillerinden karakter dizilerine deyim denir.
- Bir programlama dilinin sözdizim kuralları ile program metnindeki deyimlerin o dilde bulunup bulunmadığı belirlenir.
- Sözdizim kuralları tüm programlama dilleri için basit yapılardan oluşur.
- Lexical analiz, kaynak programın atomik birimleri(token) belirler ve sınıflandırır.

Token'lar ve Heceler

- Bir programlama dilinin sözdizimi token yada terminal adı verilen birimler ile belirlenir.
- Metinsel söz dizim, yazılan program metni ile grammerdeki tokenların diziliminin örtüşmesini açıkça ifade eder.
- dil içerisindeki birim olarak kullanılan alfabetik karakter serilerine anahtar kelimeler denir. if, for, while gibi.
- anahtar kelimeler reserved kelimelerdir ve değişken ismi olarak kullanılamazlar.
- <=, > gibi semboller reserved kelimeler gibi değerlendirilirler.

Token'lar ve Heceler

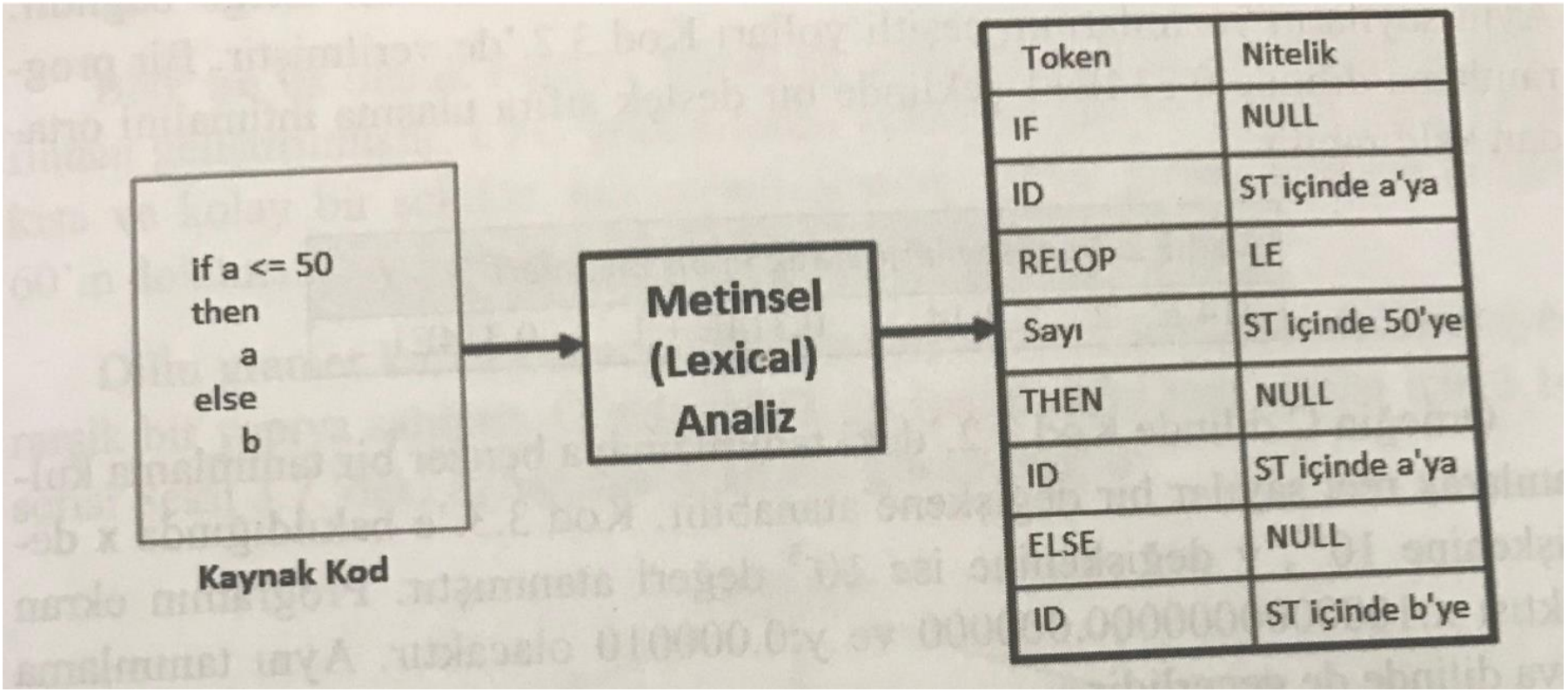
Genel matematiksel işlemler için farklı dillerin karşılaştırılması

İkili İşlem	Sembol	Pascal	C/C++ Java	Lisp
Küçük	<	<	<	<
Küçük-eşit	≤	<=	<=	<=
Eşit	=	=	==	==
Eşit değil	≠	<>	!=	/=
Büyük	>	>	>	>
Büyük-eşit	≥	>=	>=	>=
Toplama	+	+	+	+
Çıkarma	-	-	-	-
Çarpma	x	*	*	*
Bölme reel sayı	/	/	/	/
Bölme tam sayı	/	div	/	/
Modüler	MOD	mod	%	mod

Token'lar ve Heceler

- Bir programlama dilinde en düşük düzeyli sözdizimsel birimlere lexem denir.
- programlar karakter yerine lexem'ler dizi olarak ifade edilir.
- lexem'ler gruplanarak o dile ait tokenlar tanımlanır.
- programlama dilinin metinsel sözdizimi token'lar ile ifade edilir.
- tokenlar bazı durumlarda tek bir olası lexem'e karşılık gelirken, bazıları programın anlamını değiştirmeye bilir(tab, space vb.)
- örnek bir token dizisi:
  - b*b-4*c için isimb*isimb-sayi4*isima*isimc

Token'lar ve Heceler



Parçalanan tokenlar daha küçük birimlere ayrılmazlar. if token'ı bir nitelik içermez bu sebeple NULL olarak tanımlanmıştır. Değişkenler token ID olarak alınır ve Sembol Tablosuna(ST)'a bakılır. <= operatörü relational operator(RELOP) olarak ifade edilir. Küçük eşittir yani less eques(LE) olarak gösterilir.

Token'lar ve Heceler- Örnek

toplam = ogrSayisi*20 + 15; deyimi için lexem ve token listesi

Lexeme	Token
toplam	Tanımlayıcı (identifier)
ogrSayisi	
20	Tam sayı sabit
15	
=	Atama
*	Çarpma operatörü
+	Toplama operatörü
;	Noktalı virgül

Token'lar ve Heceler

- Boşluklar ve yeni satır karakterleri programın anlamını değiştirmeyen gösterimlerdir. Bu sebeple gösterimlerde dikkate alınmazlar.
- Benzer şekilde yorum satırları da /* ... */ gösterimlerde dikkate alınmaz.
- Boşluk, yorumlar, gösterimler ve onların yazımları arasındaki uyumluluk için geçerli olmayan tanımlamalar yeterlidir.
- Reel sayılar alfabetik sözdizim içerisinde en karışık kurallara sahiptir. çünkü sözdizimin bölümleri isteğe bağlı olarak tanımlanır.
  - 314.E-2, 3.14, 0.314E+1 0.314E1

Token'lar ve Heceler

```
int main(int argc, char *argv[]){
        double x = 1E12;
        double y = 1E-5
        printf('x:%1f\n', x);
        printf('y:%1f\n',y);
        return 0;
}
```

Örnek C kodu.
çıktı x için 10 üzeri 12 olacak iken
y için 10 üzere -5 olacaktır.
dolayısı ile benzer sözdizimine sahip kodlar için çıktılar oldukça farklı olabilir.
Benzer durum Java dili içinde geçerlidir.

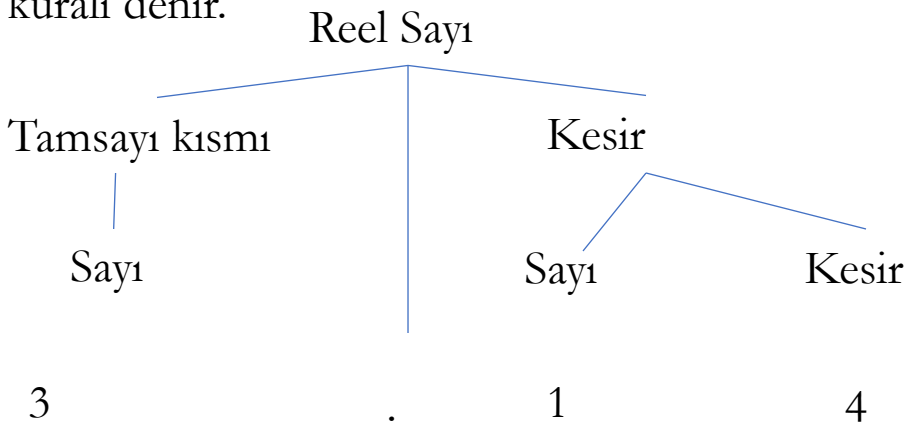
Dilbilgisi(Gramerler)

- Dilbilgisi, bir programlama dilinin metinsel sözdizimini açıklamak için kullanılan bir gösterimdir.
- Gramerler, anahtar kelimelerin ve noktalama işaretlerinin yerleri gibi metinsel ayrıntılarda dahil olmak üzere, bir dizi kuraldan oluşur.
- Dilin somut söz dizimi, anahtar kelimelerin yeri ve noktalama işaretleri gibi söz dizimsel detayları içeren yazılım ifadeler dilin gramerini oluşturur.
- BNF(Backus-Naur Formu) ve CFG(Context-Free Grammer) söz dizimini açıklamak için geliştirilmiş gösterim araçlarıdır.
- CFG içerikten bağımsız gösterim şeklidir. Belirsizlik içeren durumlar gösterilemez. Anlamlı ve belli olan kurallar gösterilebilir.

Dilbilgisi(Gramerler)

BNF ile Fortran dilinin de geliştiricisi olan John Backus tarafından geliştirilmiştir. CFG gramerlerinin matematiksel ve formal yolla daha kısa ve kolay bir şekilde açıklamayı amaçlar.

- Dilin grammer kuralları parse ağacı olarak adlandırılan hiyerarşik bir yapıya sahiptir.
- ör. 3.14 reel sayısı aşağıdaki gibi ayrıştırılır. Reel sayı bir tamsayı, bir nokta ve bir kesir ile ifade edilir. Reel sayı demek tamsayı kısmı nokta ve kesir kısımdan meydana gelir demektir. Buna gramer türetim kuralı denir.



Dilbilgisi(Gramerler)

- Parse ağacının tepesindeki yapraklar(3,.,1,3) terminaller olarak adlandırılır.
- Terminallerin alt düğümleri olmaz, uç simgelerdir ve tek başlarına simgelenirler.
- Terminaller dilin alfabesindeki simgelerden oluşurlar.
- Parse ağacının diğer düğümleri ise(reel sayı, sayı, tamsayı, kesir, reel sayılar) non-terminal olarak adlandırılır.
- nonterminaller dilin sözdizim değişkenleridir ve dilin yapısını dil yapısını simgelerler.
- Her non-terminal ifade bir terminale doğru gitmelidir ve böylece parse ağacındaki her düğüm bir sonuca dayanır.

Dilbilgisi(Gramerler)

- Programlama dilini tasarlamak için kullanılan grammer
  - başlangıç terminali
  - non terminaller
  - terminaller
  - kurallar

  dan oluşur.
- Terminaller dilin alfabesinin simgeleridir. yeni türetimler yapılmaz.
- nonterminaller, bir programlama dliindeki yapıları gösteren söz dizim değişkenleridir.
- kurallar kümesi, bir cümlesel yapının bileşenlerinin saptanması için kullanılır.
- her bir terminal bir nonterminalden belirli kurallar ile oluşturulur.
- Nonterminaller ile terminal arasına -> veya ::= gibi işaretler konur ve olabilir olarak okunur.
- başlangıç terminali ise, türetime başlamak için kullanılır.
- BNF 4 gramer yapısını kullanarak dil tasarımında kullanılan bir yapıdır.
- BNF'te CFG'den farklı olarak nonterminaller <> sembolleri arasında alınır. ve boş dizi için lamda sembolü yerine <boş> şeklinde yazılır. Örnek: <tamsayı>, <kesir>, <sayı> (non terminaller), 0, 1,2,3(terminaller)

Türetimler

- ::= olabilir, | veya gibi değerlendirilir.
- Reel Sayılar için BNF yazımı aşağıdaki gibidir.

<reel sayı> ::= <tamsayı-kısım>.<kesir>

<tamsayı-kısım> ::= <sayı>|<tamsayı-kısım><sayı>

<kesir> ::= <sayı>|<sayı><kesir>

<sayı> ::= <rakam>|<rakam>

<rakam> ::= 0|1|2|3|4|5|6|7|8|9

<kesir> ::= <rakam>|<rakam><kesir>, <rakam>(alternatif)

Türetimler

- Örnek: 0.45 sayısının BNF grammeri ile sözdizim ifadesini oluşturunuz. .45'değeri de kabul edildiğini düşünelim.

<reel sayı> ::= <tamsayı kısmı>.<kesir kısmı>

<tamsayı kısmı> ::=<boş>|<rakam serisi>

<kesir kısmı> ::= <rakam serisi>

Parse Agaçlarının Somut Sentaks Tanımlanması

- Grammerdeki türetim sonuçları token katarlarının bir araya getirilme kurallarıdır.
- Grammere uyan bir parse ağacı aşağıdaki kuralları sağlamalıdır.
  - her bir yaprak boş katarı gösteren bir terminal veya <boş> ile etiketlenmiştir.
  - her bir yaprak olmayan düğüm bir non-terminal durum ile etiketlenmiştir.
  - bir yaprak olmayan düğümün etiketi bir ürünün sol tarafıdır ve düğümün çocuklarının etiketleri solda sağa doğru o ürünün sağ tarafının biçimlendirir.
  - kök düğüm bağlangıç non-terminali ile etiketlenmiştir.
  - bir parse agacı soldan sağa terminaller okuyarak biçimlendirilen katarı meydana getirir. bir dildeki katar için bir parse agacı oluşturulabiliyor ise bu katar o dile aittir denir.
  - Bir parse agacının oluşturulması işlemine parsing denir. (ayrıştırma=

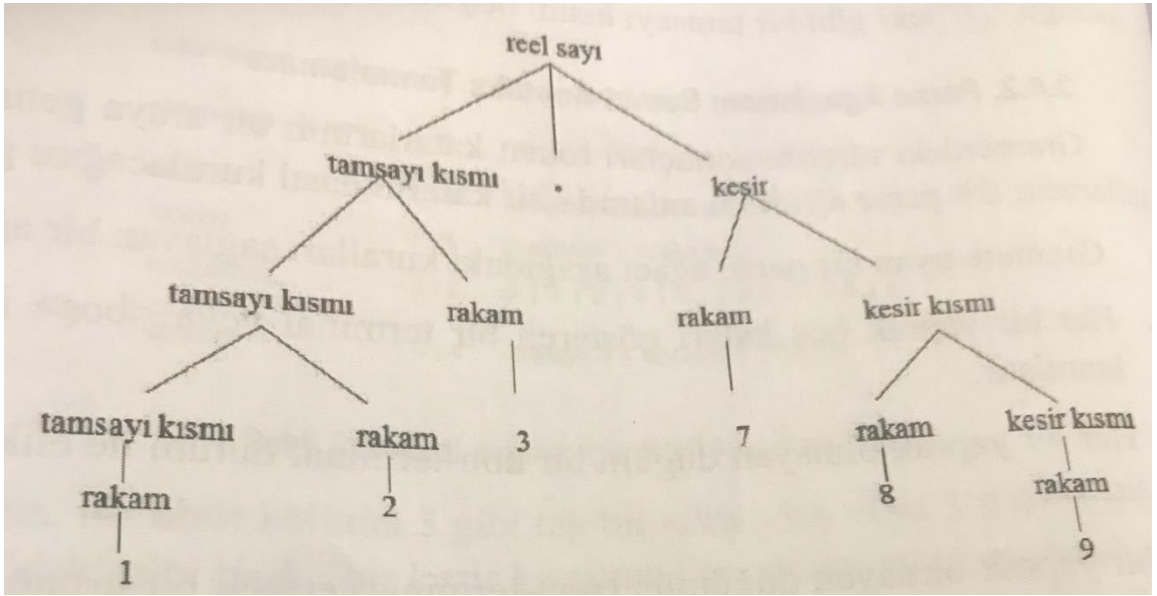
Parse Agaçlarının Somut Sentaks Tanımlanması

<reel sayı> ::= <tamsayı kısmı>.<kesir kısmı>

<tamsayı kısmı> ::=<boş>|<rakam serisi>

<kesir kısmı> ::= <rakam serisi>

örnek: 123.789 reel sayısının parse ağacının verilmiş olan BNF kuralına göre çizimi gösterilmiştir.
123 değeri sola doğru büyürken, 789 kısmı sağa büyümektedir.



Belirsizlik

- Bir dilin grameri eğer bazı katarlar için birden çok parse ağacı oluşturulabiliyor ise bu gramer belirsiz olarak adlandırılır. ve Programlama dilleri belirsiz olmayan grammer ile tanımlanmalıdır. Eğer belirsizlik bulunuyorsa, her bir katar için bir parse ağacını dışında hepsini yöneten kabul edilmiş düzenin saptanması gereklidir.
- Belirsizliği ortadan kaldırmak için yeni non-terminal simgeler eklenmelidir ve sadece sol recursive veya sağ recursive yapıya izin verilmelidir.

Belirsizlik

- Sol recursive yapı örneği

  S -> Ra | a

  R -> ab | Rb

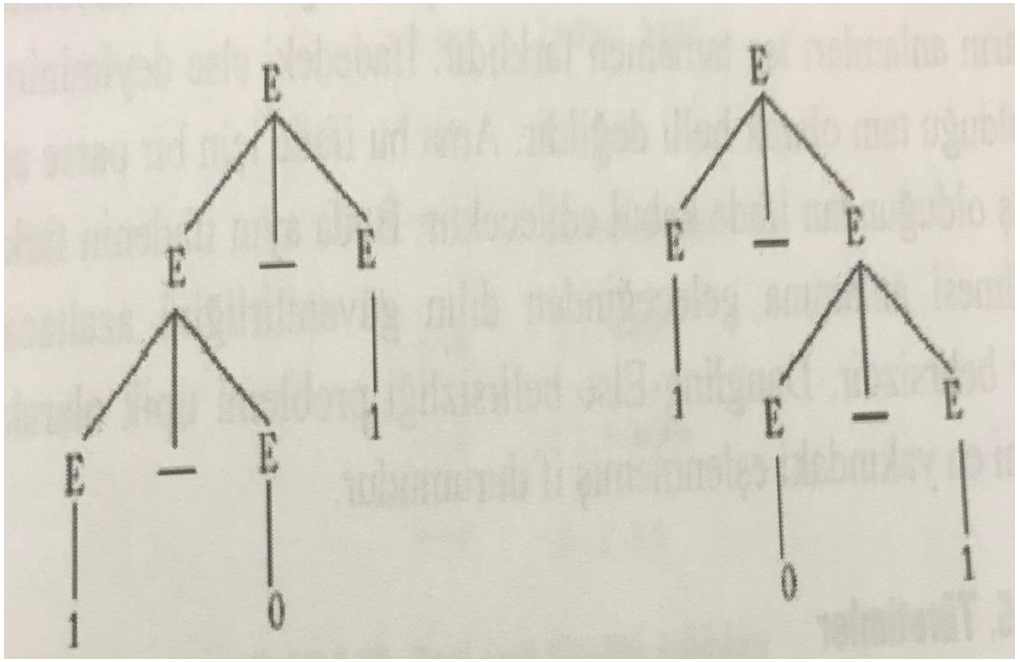
- Sağ recursive yapı örneği

  S -> aR | a

  R -> ab | bR

Belirsizlik-Örnek

E ::= E-E | 0 | 1 BNF grameri ile 1-0-1 katarının parse ağacını aşağıdaki gibi 2 şekilde çizebiliriz.



2 farklı gramer çizebildiğimiz için belirsizdir. bu sebeple soldan veya sağdan parantez değişimi ile gramer yeniden düzenlenmelidir. (1-0)-1 veya 1-(0-1) şeklinde düzenlenmelidir.

Gramer if-else belirsizliği

Bu şekilde 2 farklı parse ağacı çizilmesine imkan tanındığında dilde kararsızlık olacaktır. Bu nedenle dilin kabul edeceği ağaçlarda bu kararsızlık ortadan kaldırılmalıdır.

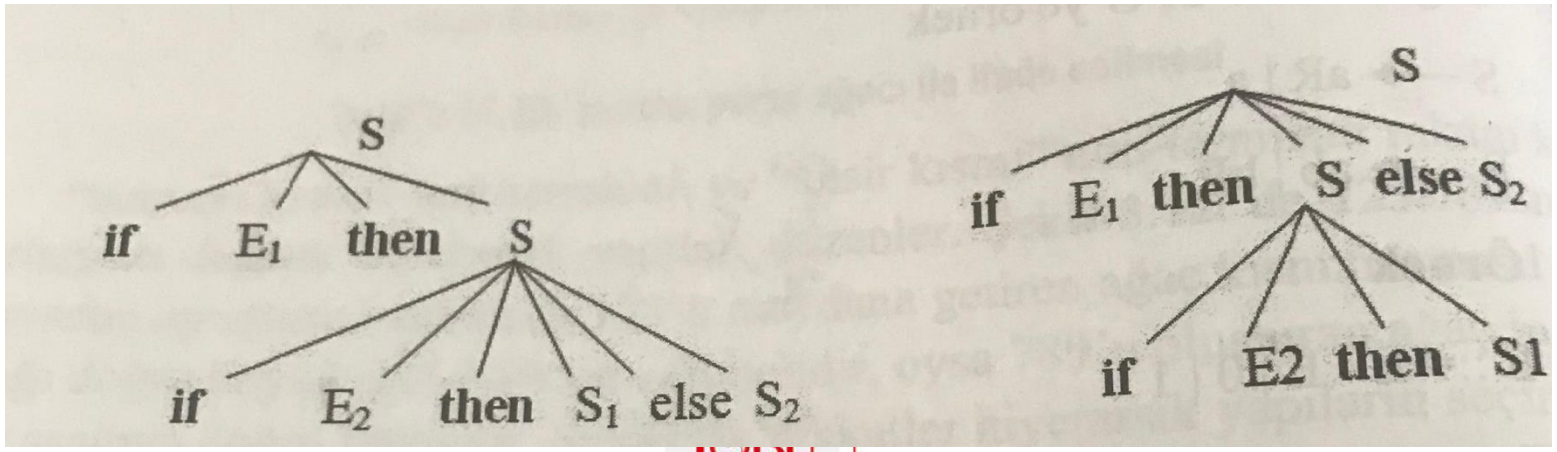
Kural

S:: if E then S

S:: if E then S else S

ise

if E1 then if E2 then S1 else S2 ifadesinin kabul edilip edilmeyeceğini inceleyiniz.

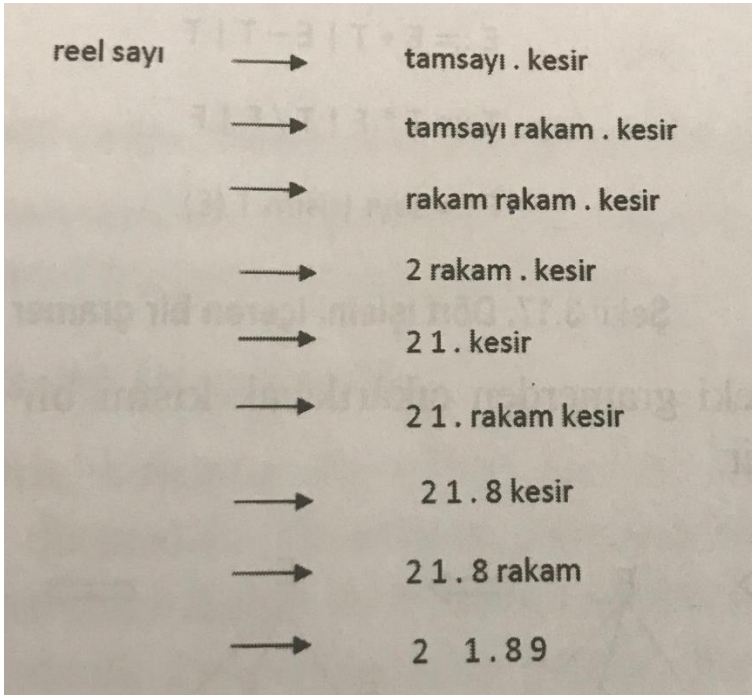


Türetimler

- Top-down parse ağacı: ağacın kökünden alt tarafa doğru işler. non-terminallerden başlar terminallere doğru ilerler.
- Bottom-up ise parse ağacının altından köke doğru işler. terminallerden başlayıp non-terminallerde doğru gider.
- Türetimler karakter katarlarından oluşmalıdır.

Türetimler

- örnek olarak 21.89 reel sayısının türetim ağacı aşağıda gösterilmiştir.



Aritmetik İfadeler için Grammerler

- Aritmetik ifadelerinde gramerleri olmalıdır.
- İşlem önceliği, birleşme özelliği gibi kavramların grammer ile ifadesi gereklidir.
- İyi bir programlama dilinde tüm ifadelerinde sorunsuz olarak operatörlerin sınıflandırılması ve işlem süreçlerine dahil edilmesi beklenir.

İşlem önceliği

- a+b+c+d için programlama dili + işareti ile ayrılmış elamanlar listesi olarak değerlendirilir.
- a*b+c*d-e için de a*b, c*d, e terimlerinin listesi olarak değerlendirilir.
- Aynı şekilde bir matematiksel ifadeyi terimler listesi haline getirme süreci, her bir terimin elemanları içinde uygulanır. Yani a*b 'de * işaretine göre ayrılmış a ve b terimlerinden oluşur denilir.

İşlem önceliği-dört işlem

E::= E+T | E-T | T

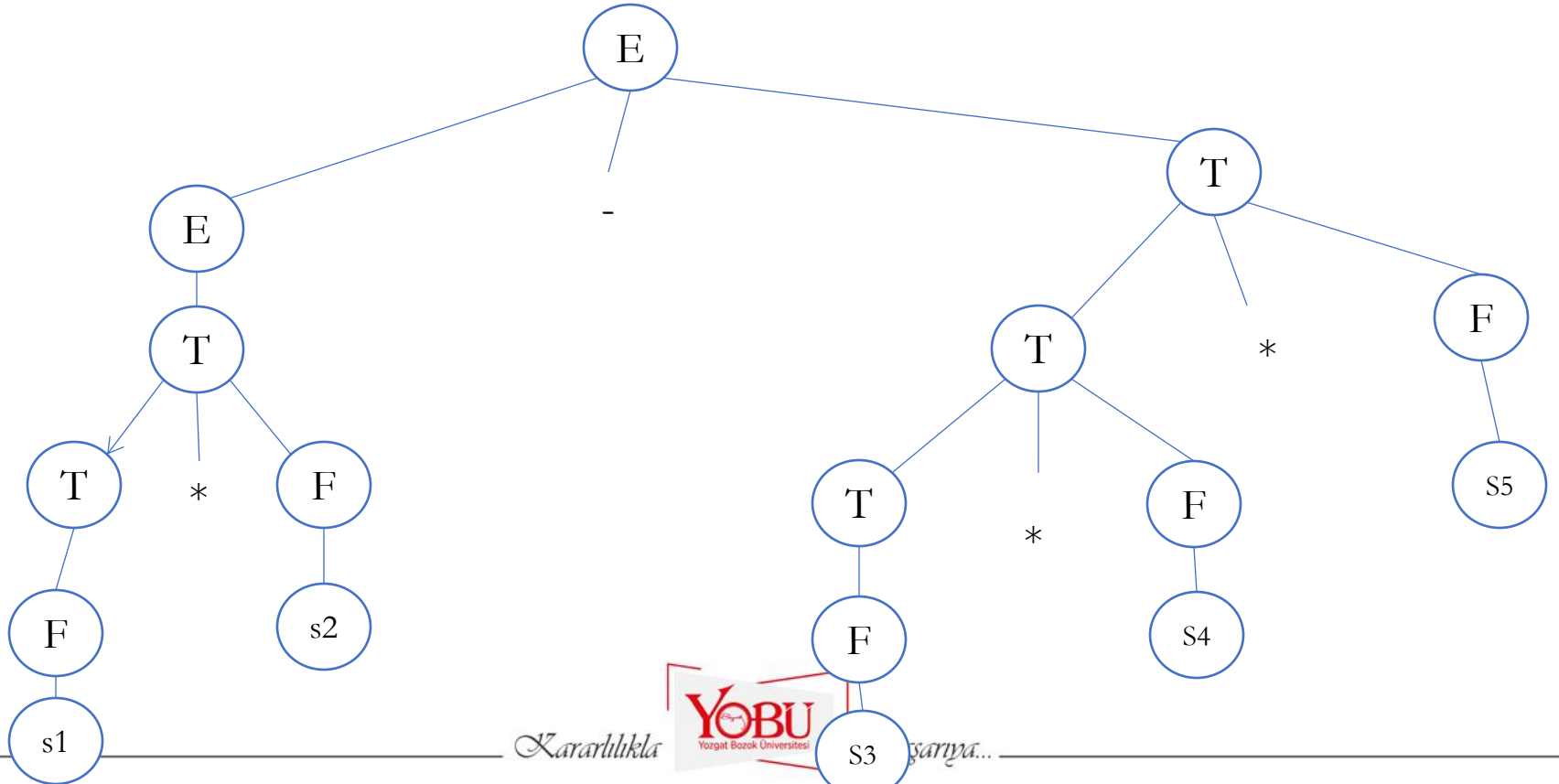
T::= T*F | T/F | F

F::= sayı | isim | (E)

Dört işlem içeren bir gramer yapısı örneği E, T ve F non-terminalleri sırasıyla ifadeleri, terimleri ve çarpanları göstermektedir.
a*b + c*d+ e ifadesinde önceliği de dikkate alacak şekilde bu gramer yapısı ile çözülebilir.

İşlem önceliği-dört işlem-örnek

sayi1*sayi2 – sayi3*sayi4*sayi5 katarının tam parse ağacını verilen gramere göre çiziniz.



Birleşme Özelliği

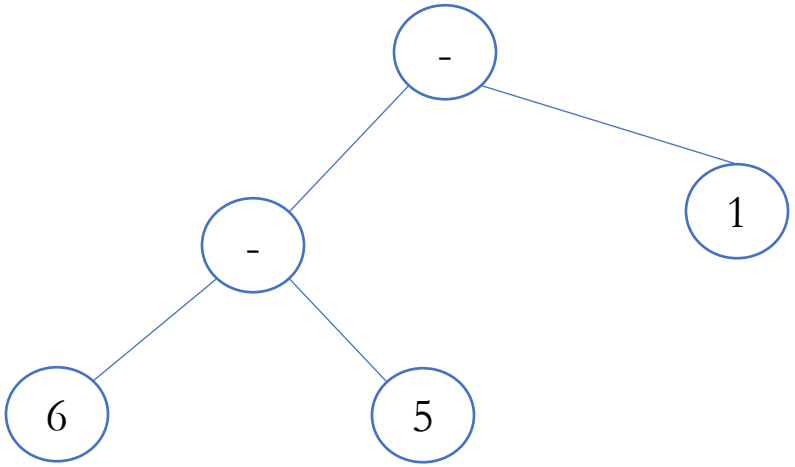
- Aritmetik ifadelerin birleşme özellikleri gramer tasarımında dikkate alınmalıdır.
- Tasarlanan gramerin belirsiz bir gramer olmaması için ilgili operatöre ilişkin birleşmenin soldan birleşim yada sağdan birleşim özellikli olarak kararlaştırılması gerekir.
- 6-5-1, 8+2-4-2 ifadelerinde soldan birleşme yapılmalıdır. Bu sebeple agaç yapısı sola doğru büyüyecektir.

Birleşme Özelliği

L :: = L + sayı | L-sayı | sayı

R:: = sayı + R | sayı-R | sayı

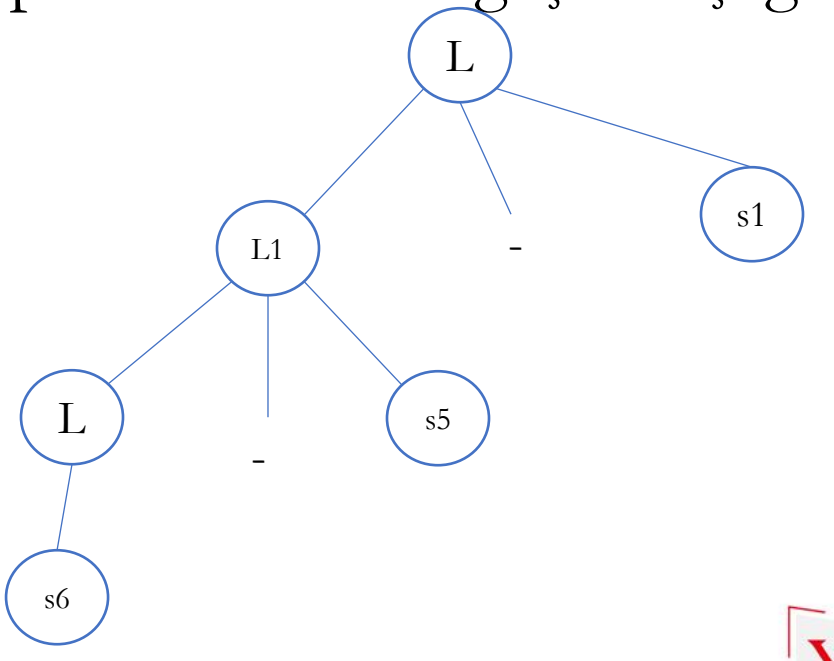
sol birleşmeli ve sağ birleşmeli gramer örneği

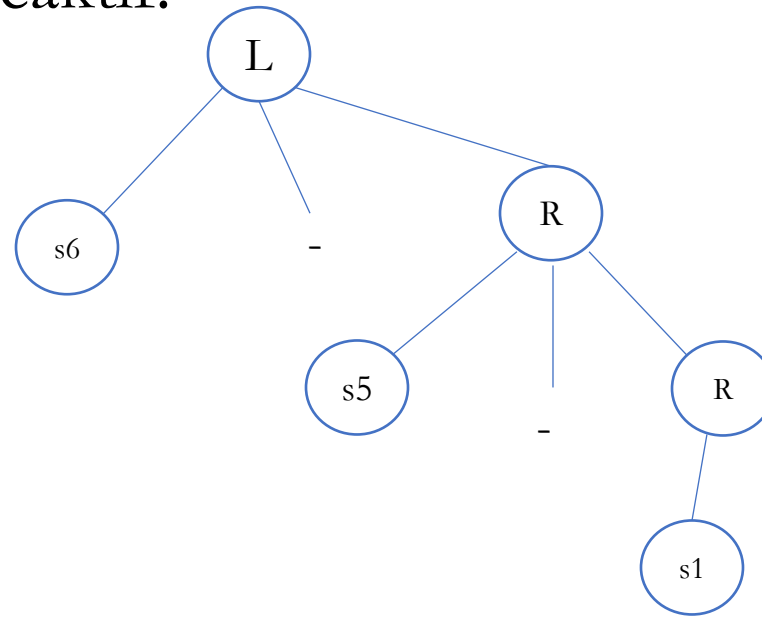


6-5-1 ifadesinin yukarıda verilen gramere uygun soldan birleşme örneği

Birleşme Özelliği-Sol ve Sağ Birleşme

6-5-1 ifadesinin sol ve sağ birleşme örnekleri aşağıda verilmiştir. Buna göre her iki çözümde mümkündür ancak genelde soldan birleşme gramer yapısı açısından daha pratiktir. Parse ağaçları aşağıdaki gibi olacaktır.





Birleşme Özelliği

- Bir dildeki deyimlerin sentaksı belirlenmiş bir tabloya göre birleşme ve öncelik operatörleriyle tanımlanır. C dilindeki ikili operatörler aşağıdaki gibidir.
- aynı sıradaki bütün operatörler

aynı birleşme ve önceliğe

sahiptir.

- öncelik üstten aşağı doğrudur. Yani

en düşük işlem önceliği atama operatörü

iken en yüksek çarpımsal operatörlerdir.

- atama operatörü sol birleşme,

diğer operatörler sağ birleşme özelliğine

sahiptir.

İşlem	Sembol
atama	=
mantıksal veya	||
mantıksal ve	&&
dahil veya	|
hariç veya	&
ve	^
eşitlik	== !=
ilişkisel	< <= >= >
değiştirme	<< >>
toplamsal	+ -
çarpımsal	* / %

- Aritmetik ifadeler için her bir öncelik seviyesi veya sırası için bir non-terminal seçilerek gramer tasarlanabilir.
- := sağ birleşmeli, (+ -) ve (/*) sol birleşmelidir.

Birleşme Özelliği Örnek

- :=, + - , * / için üç seviye non terminal (A,E,T)ve çarpanlar için 1 seviye(F) non-terminal kullanılarak gramer tasarımı yapalım.

:= sağ birleşmeli ve + - ile * / işlemleri soldan birleşmeli tasarlanmalıdır.

A ::= E := A | E

E ::= E + T | E - T | T

T ::= T * F | T / F | F

F ::= (E) | isim | sayı

Dilbilgisi çeşitleri

- EBNF, BNF gösteriminin geliştirilmiş halidir.
- Yineleme, seçimlik ve değiştirme özellikleri bulunur.
- okunabilirlik ve yazılabilirlik daha iyidir.
- gösterimi kolaylaştırır.
- Dilde yenilik yoktur. tüm gösterimleri hem BNF hem EBNF ile yapılabilir.

EBNF

Yineleme: EBNF'te yineleme {} parantezleri ile gösterilir.

<ifade listesi> ::= {<ifade>;} -> EBNF

<ifade listesi> ::= <boş> | <ifade>;<ifade listesi> ->BNF

Seçimlik: EBNF'te seçimlik [ ] ile gösterilir.

<gercel sayılar> ::= [tam kısım] . <kesirli ifade>

<gercel sayılar> ::= <tam kısım>. <kesirli ifade> | . <kesirli ifade>

EBNF

Değiştirme: * ve + sembolleri kullanılarak tekrarlı ifadelerin gösterimi yapılır.

<id> ::= <karakter> | <id><karakter>| <id><rakam> -> BNF gösterim

<id> ::= <karakter> (<karakter> | <rakam>)*

EBNF-BNF karşılaştırması

- EBNF'te BNF'e göre sunulan eklentiler
  - süslü parantezler, 0 yada daha fazla tekrar ifade eder.
  - köşeli parantezler, [] isteğe bağlı yapıyı ifade eder
  - dikey çizgi bir seçeneği
  - parantezler gruplamak için kullanılır.

- süslü parantezler gibi dilin tanımlanmasında özel bir yere sahip sembollere meta sembol denir. EBNF'te bu sembollerin sayısı daha fazladır. Bu semboller aynı zamanda dilin kendi sözdizimi yapısında da gösterilebilir. örnek olarak A[i] ifadesinde i isteğe bağlı değildir. bu yüzden belirtilmesi gereken ifadelerde meta sembolleri ayırt ederken dikkatli olmak gereklidir. Bu karmaşıklıktan ' işareti kullanarak kurtulabilmek mümkündür.

Aritmetik ifadeler BNF-EBNF örneği

BNF

<ifade> ::= <ifade> + <terim> | <ifade> - <terim> | <terim>

<terim> ::= <terim> * <faktör> | <terim> / <faktör> | <faktör>

<faktör> ::= sayı | isim | (<ifade>)

EBNF

<ifade> ::= <terim>{(+|-)<terim>}

<terim> ::= <faktör>{(+|/)<faktör>}

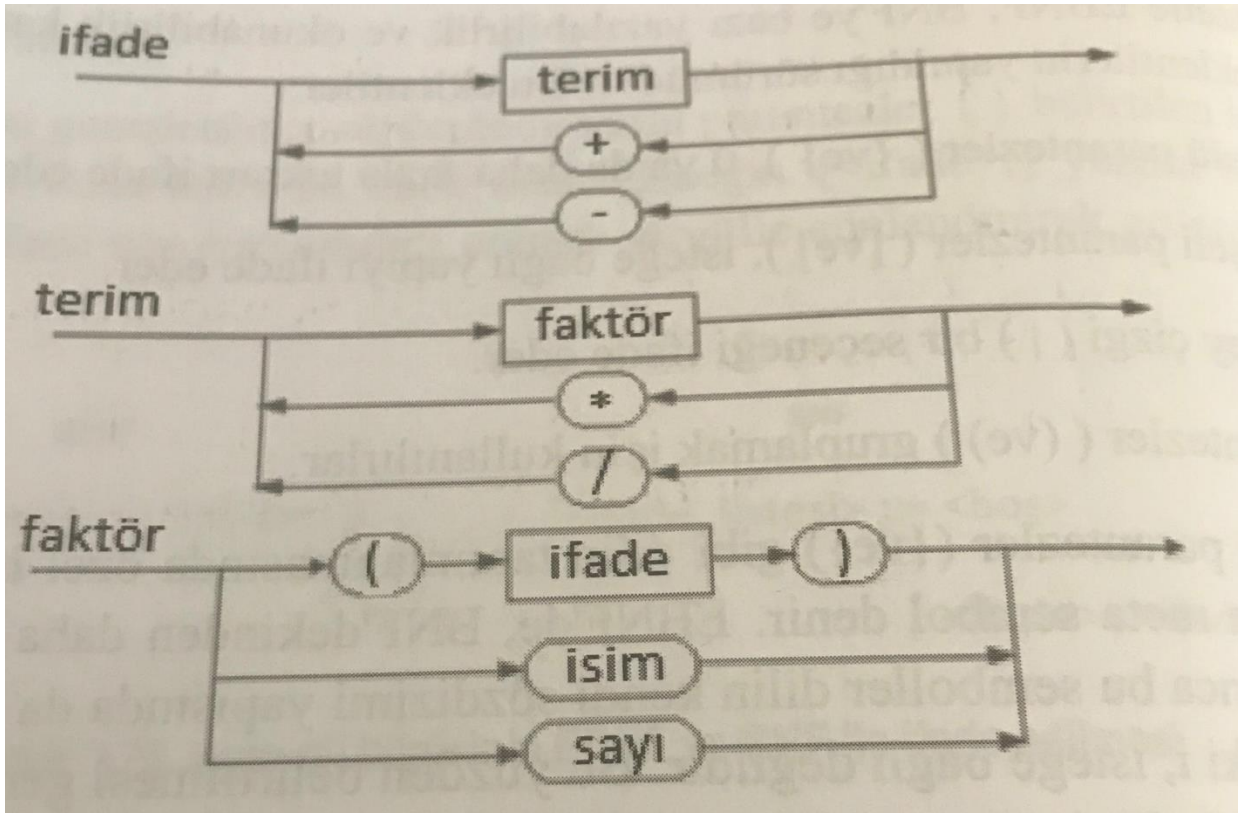
<faktör> ::= '('<ifade>')' | isim | sayı

Sözdizim Grafikleri

- Gramerleri göstermek için kullanılan grafiksel gösterimlerdir.
- Görsel ifade için kullanılır.
- terminaller daireler ile ve non-terminaller dikdörtgenler ile ifade edilir.

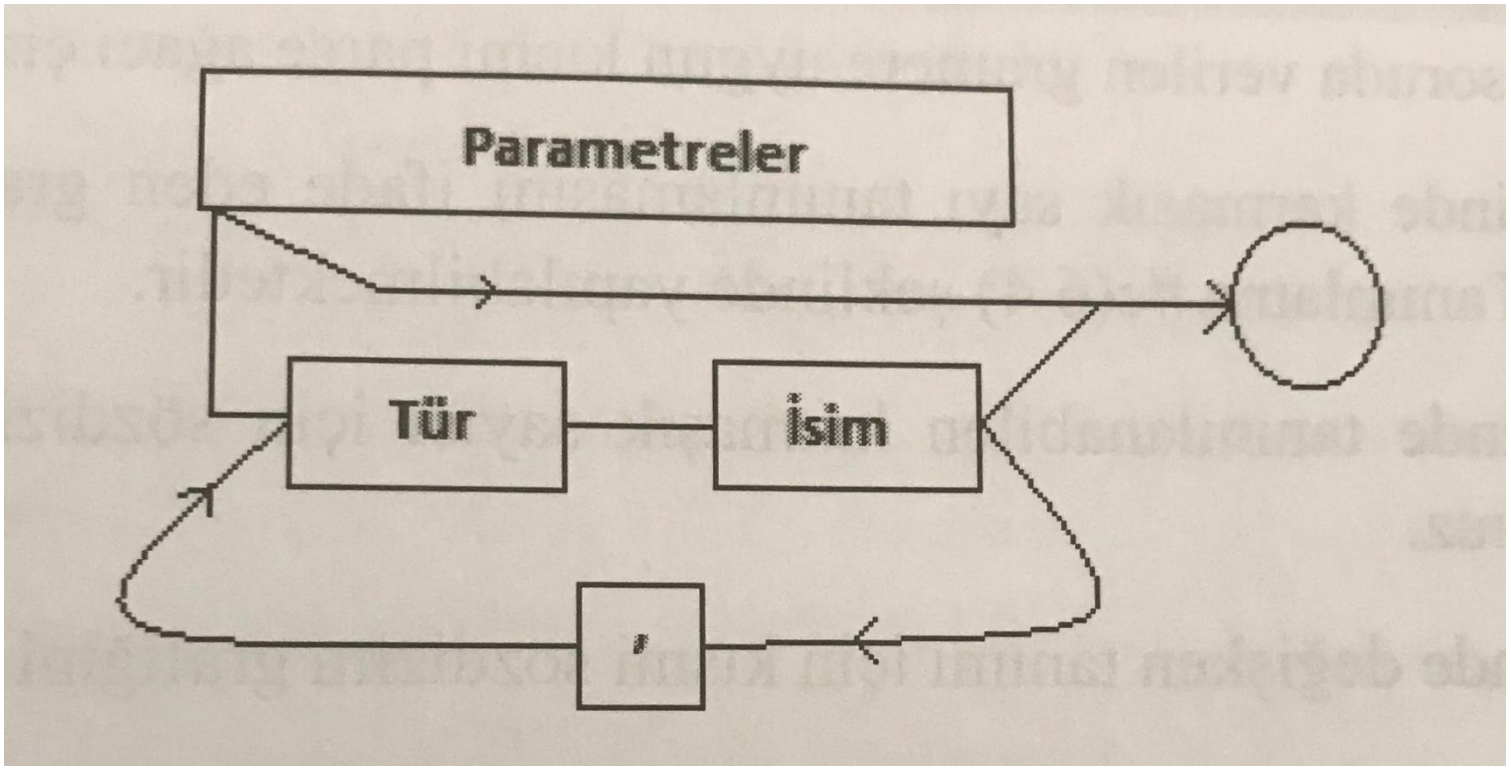
Sözdizim grafikleri

Aritmetik ifadeleri sözdizim grafiği örneği



ifade ve terimlerin alt çizelgelerin döngü olduğuna ancak ana faktörde döngü olmadığına dikkat edilmelidir.

c# parametre tanımlama sözdizim çizelgesi örneği



Örnekler

- Aşağıdaki örnekleri prefix, postfix ve infix formatında yazınız.
  - 3+2*3-4*8
  - 2-2*3-5*(7-3)
    - Prefix: - - 2 * 2 3 * 5 - 7 3
    - Postfix: 2 2 3 * - 5 7 3 - *
  - 5*(1+4*(2+3*(3+2*(4+1*5))))

BNF Örnekleri

Aşağıdaki grammerin oluşturduğu dili açıklayınız.

<S> → <A> <B> <C>

<A> → a <A> | a

<B> → b <B> | b

<C> → c <C> | c

BNF Örnekleri

İkili sayılar için BNF Grammer Yapısı (Belirsiz)

<ikili-metin> ::= 0 | 1 | <ikili-metin><ikili-metin>