Giriş

Giriş

arkauçta makine bağımlıdır (alt kısım)
ara kod üretecinden itibaren böyledir

.net framework ve jre sistem bağımlı çalışır hibrittir makine kodu üretilmez

yakında bu sürece yapay zeka etkisi eklenebilir

Lexical Analiz

23 Şubat 2026 Pazartesi 11:23

Lexical Analiz

- lexeme(token) en küçük anlamlı parçaya böler. symbol tablosuna bakarak ona göre örn: for kelimesini ayırır.

- parçalama ve nitelendirme (for nedir abc gibi değişkenleri bir aldı = operatörünü atama olarak ayırır. 10.2 yi alır ve ondalıklı sayı olarak sayar)

Token Listesi oluşumu:

Sonuc := veriler / 25

Sonuc	degisken,1	tanımlayıcı
:=	atama,nil	atama işlemcisi
veriler	değişken,2	tanımlayıcı
/	bölme,nil	bölme işlemcisi
25	tamsayı,3	tamsayı

degiskende bosluk olması lexical analize aykırıdır.

derleyici tasarımında lexical analizin nasıl olacağı değişebilir.
lexical analizin çıktısı tokenlardır. syntax analiz burdan tokenları alır

Syntax Analiz

23 Şubat 2026 Pazartesi 11:23

Syntax Analiz

- programlama dilinin gramer kurallarına uygun sıralamda olup olmadıklarını belirler
- double a = 3.2; int x = a; gibi bir hatayı syntax analiz yakalayamaz çünkü sözdizimsel olarak doğrudur.
- BNF ve GFG gibi gramer oluşturma araçlarını kullanır.
- hata yoksa gramer katarı üretilir
- sürekli tekrarlanmadığından performans verimliliği analizi genelde yapılmaz ihmal edilir.

Grameri oluşturan yapılar:

1. Başlangıç nonterminali
2. Nonterminaller
3. Terminaller
4. Kurallar.

çıktısı bir sürü parse ağacıdır. yapraklarında nonterminal kalırsa o zaman derlenmez hata verir

syntaxla kontrol edilemeyecekler:

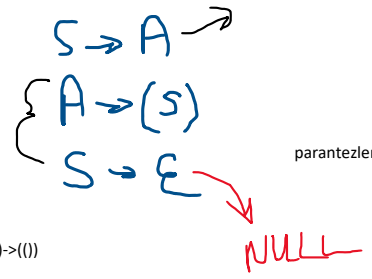
1. tür atamaları
2. null nesne erişimi
3. değer atamadan okuma

bunlar semantik anaiiz devreye girer.

CFG

başlangıç nonterminali

nonterminaller



parantezler terminaller

S->A->(S)->(A)->((S))->(())

bu ifadeler de kurallardır

BNF

<reel sayı>	::=	<tam sayı kısım> . <kesir>
<tamsayı - kısım>	::=	<sayı> | <tam sayı - kısım> <sayı>
<kesir>	::=	<sayı> | <sayı> <kesir>
<sayı>	::=	<rakam> |<rakam> <sayı>
<rakam>	::=	0 | 1 | 2 | 3| 4| 5 | 6 | 7| 8 | 9
<kesir>	::=	<rakam> |<rakam> <kesir>

nonterminal ile duramazsın terminalle durmalısın.
hata verir üretemez ayrıştırma ağacının yapraklarında nonterminal olmaz
parse ağacı (ayrıştırma ağacı çıktısı üretir)

Önceki mesajdaki 4 temel bileşenin BNF notasyonunda nasıl yazıldığına bakalım:

1. Nonterminaller (Değişkenler)

BNF'de bir yapının nonterminal olduğunu belirtmek için o kelime her zaman **küçüktür-büyüktür işaretleri** `< >` arasına yazılır. Bu, onun daha alt parçalara ayrılabileceğini gösterir.

- **BNF Örneği:** `<atama_islemi>`, `<matematiksel_ifade>`, `<sayi>`

2. Terminaller (Tokenlar)

Terminaller en küçük yapı taşları olduğu için `< >` işaretleri kullanılmaz. Karmaşıklığı önlemek için genellikle **kalın harflerle**, normal metin olarak veya **tırnak işaretleri** `" "` arasına yazılarak belirtilirler.

- **BNF Örneği:** `"if"`, `"+"`, `"="`, `"while"`, `0`, `1`

3. Kurallar (Production Rules)

BNF'nin en belirgin özelliğidir. Matematiksel CFG'de kullanılan ok işareti (→) yerine, BNF'de `::=` sembolü kullanılır. Bu sembol, *"olarak tanımlanır"* veya *"şundan oluşabilir"* anlamına gelir. Ayrıca, bir nonterminalin dönüşebileceği birden fazla alternatif varsa, bunları ayırmak için `|` **(VEYA)** işareti kullanılır.

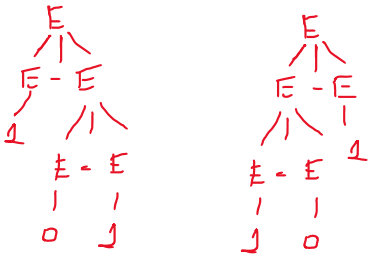
- **BNF Örneği:** * `<rakam> ::= "0" | "1" | "2" | "3" | "4"`
  (Okunuşu: Rakam; 0, 1, 2, 3 veya 4 olarak tanımlanır.)

4. Başlangıç Nonterminali

BNF kuralları yukarıdan aşağıya doğru yazılır ve dosyanın en üstünde tanımlanan, kapsayıcı olan ilk kural genellikle başlangıç nonterminali olarak kabul edilir.

*sadece ismini bil DFA otomat deterministik olmalıdır. belirsizlik olmamalı

Belirsizlik:
<E> ::= E - E | 0 | 1
1-0-1 üretilecek
aynı ifade aynı dilbilgisi 2 farklı apaç tarafından üretilememeli yoksa kodu yazılamaz.
örnek:



belirsizliği ortadan kaldırmak için belirsizliğin oluştuğu yere yeni non terminal simgeler eklenir ve sadece sol rekürsif veya sağ rekürsif yapıya izin verilir.

S-> Ra|a
R->ab|Rb
nt ler sola genişlediği için sol rekürsiftir.
S->Ra->Rba>Rbba>abbba

EBNF

BNF'nin geliştirilmiş halidir daha kısa ve öz yazım sağlar. gösterimi kolaylaştırır.
Yineleme (repetition)
Seçimlik (optionality)
Değiştirme (alternation)

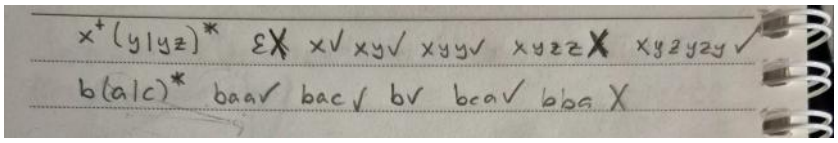
Örnekler:
YINELEME:
EBNF
<ifade_listesi> ::= {<ifade>;}

BNF
<ifade_listesi> ::= <boş> |<ifade>;<ifade_listesi>

süslü parantez tekrarlama sağlar {ab} ababab null da dahildir

SECİMLİK:
[]. kısım opsiyonel olur

DEGİSTİRME:
+'dan en az bir adet olucak
*0 veya daha fazla



Semantik Analiz

23 Şubat 2026 Pazartesi 11:25

Semantik Analiz

Tür atamaları
Null nesne erişimi
Değer atamadan okuma

gibi hataları yakalar.

Her dilde sıkı tutulmamıştır.
c'de c++da daha gevşektir.

```
int a;
print(a);
```

yukarıdaki kodda lexical veya syntax hata vermez.

buradan da kurtulan kod artık derlenir. Hata kontrolünün sonuna gelinmiştir.

Ara Kod Üreteci

23 Şubat 2026 Pazartesi 11:36

ön uç burda biter.

T3=T1*T2;

a=b+c/z;
buna çevirilir
T1=c/z;
T2=b+T1;

yazılan kodu daha rahat makine diline çevrilebilmesi için ara bir forma dönüştürür.
java->bytecode
c->assembly

Optimizasyon (ops)

23 Şubat 2026 Pazartesi 11:32

ara kod üzerinde yapılır.

```
x=5; y=8;
if(x>y) {
        ölü kod
}
```

bu kod arakoddan atılır çünkü çalışma ihtimali yoktur.

```
//optimize edilmemiş kod
do{
        deger = 10;
        toplam = toplam + deger;
} while(toplam<100);

//optimize kod
deger=10;
do{
        deger = 10;
        toplam = toplam + deger;
} while(toplam<100);
```

Kod Üreteci

23 Şubat 2026 Pazartesi 11:43

Makine bağımlıdır ara kodu makine diline çevirir.

linux ve windowsta farklı kod üretecleri kullanılır.
ismi aynı olabilir fakat o makine için üretilmiş hali olmalıdır.

Derleyici vs Yorumlayıcı

23 Şubat 2026 Pazartesi 11:45

analiz kısımları her ikisinde de vardır

yorumlayıcı satır satır işler. hataya gelene kadar program patlamaz. python javascript bu şekilde işler.

yorumlayıcıda kaynak kod her zaman gereklidir.