Derleyici ilk ders

Kaynak kodu

↓

lexical Analiz ⇒ metnimizi karakter karakter dayup ve bizim kaynak kodumuzu (derleyici tarafından anlamlı) en kucuk atomik birimlere ayırır. Derleyici tarafından tanımlı bir karakteri ona bakar yanlışlarla ilgilenmez. if) x<5) bunun gibi lexical hata olusması icin tanımlanmayan tokenların olması lazımdır. Öğ Dolar gibi Javada

↓

Syntax Analiz ⇒ söz dizimsel analiz. lexical aşamasında bulduğumuz tokenler syntax aşamasında doğru dizilmişmi diye değerlendirilir

↓

Orta seviye kod üretimi ⇒ Yüksek seviyeli kodun daha düsük daha sade haline donuşturulmesidir. Amac makina kodlarına daha iyi yaklaşıp anlamak

↙ ↘

kod optimizasyonu

↓

kod üretimi

Hatabulma ve düzeltme ⇒ Kullanıcı arayüzleri icerisinde kişilere uygun mesajları veren bileşen

↓

Sembol tabloları ⇒ degiskenlerle ilgili bilgiler tutulur.

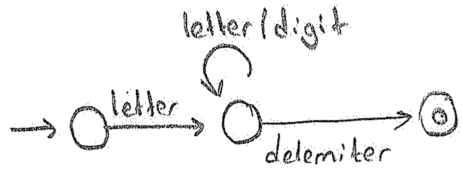
↘ ↙

Hedef kod ⇒ makina kodu

Öğ

```
State0 :  C=GETCHAR();
          if LETTER(c) then goto state1
          else FAIL();

State1 :  C= GETCHAR();
          if LETTER(c) or DIGIT(c) then goto state1
          else if DELIMETER(c) the goto state2
          else FAIL();

State2:   RETRACT();
          return (id, INSTALL());
```

⇒



4 Tip hata vardır.

⇒ lexcial error

⇒ Syntax error

⇒ semantic error

⇒ Runtime error

Derleyici: Yazdığımız kodları makine kodlarına çevirmeyi sağlar.

1. LEXICAL ANALİZ :

* Yazdığımız kodları en anlamlı küçük parçalara ayırır. (Token) Derleyici tarafından tanınmayan bir karakter girilirse lexical hatadır. Mesela Türkçe bir karakter (ü, ö) Java'da tanınmaz.

Örn: if (a<5)
b=a+c;

if
(
a
<
5
)
b
=
a
+
c
;

Bunlar Java'da tanımlı karakterler olduğu için lexical hata üretmez.

2. SYNTAX ANALİZ (SÖZ DİZİMSEL ANALİZ)

* Derleyici kurallarına göre dizilip dizilmediğine bakar.

Örnek: if) a<5)

Lexical hata vermez fakat if'ten sonra "(" gelmesi gerektiği için syntax hata verir.

if
)
a
<
5
)

syntax error

3. ORTA SEVİYE KOD ÜRETİMİ :

* Yüksek seviyeli dille oluşturulmuş kodu if ve go to ile orta seviye koda dönüştürür.

for
while
⋮
⇓
(if ve go to)

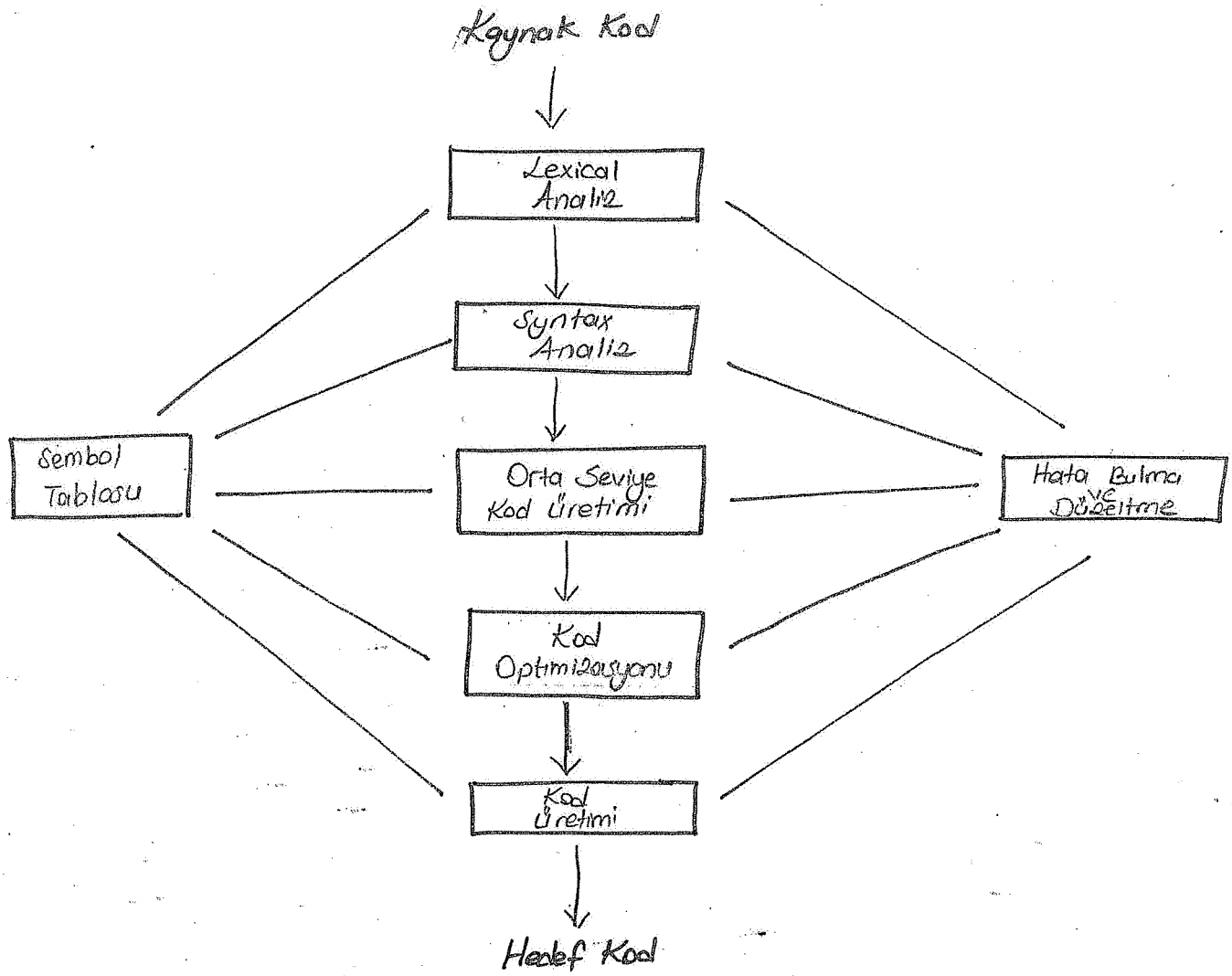
if ve go to ile orta seviye koda dönüştürür.

Örnek: for i=0; i<n; i++;
a=a+i;

if ve go to ile yap.

```
    i=0;
x:  if (i<n)
      a=a+i;
      i=i+1;
      go to x
    end
```

DERLEYİCİ TASARIMI



1. LEXICAL ANALİZ

* Lexical analiz kullanılan kelime ve sembollerin kendisinde tanımlı olup olmadığını kontrol eder.

if (a < 5) } if (a < 5)

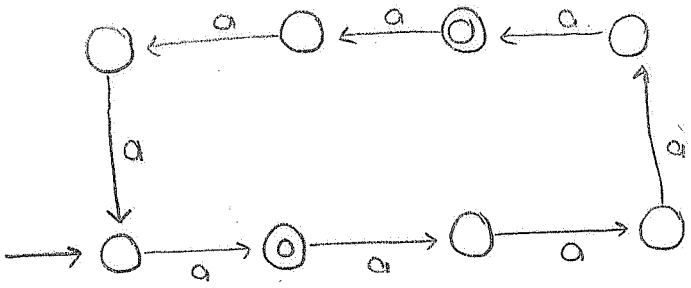
2. SYNTAX ANALİZ

* Syntax analizde kullanılan kelime ve sembollerin amacı dışında yanlış yerde kullanılıp kullanılmadığının kontrolüdür.

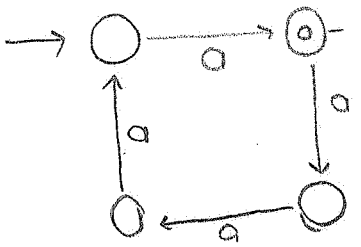
if) a < 5) } Lexical analizde hata vermez. Fakat syntax analizde ")" yanlış yerde kullanıldığı için syntax hatası verir.

Ö/ `int count=0;` token sayısı = 30
`double sum = 0.0;`
`while (count < maxVect) {`
`sum = sum + vect[count];`
`count++;`
`}`

Ö/



minumum DfA?
regüler ifade?
Dil? ...



context free grammer

$a.(aaaa)^*$

$S \rightarrow aA$
$A \rightarrow aaaaA \mid \lambda$

aA
aaaaaA
aaaaA

aaaaaaaaa

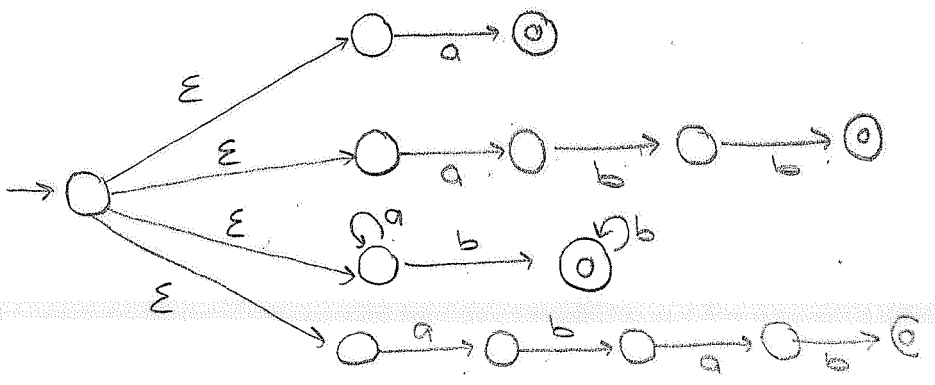
Ö/ aa veya aaaa tokenlerini içeren regüler ifade?

$aa.(aa+aaa)^* + aaa(aa+aaa)^*$

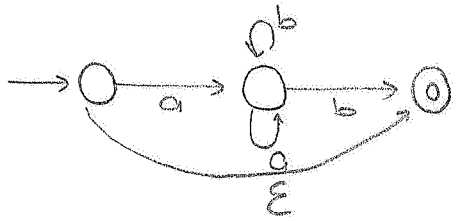
Hocanın yaptığı
$aa(aa)^* + aaa(aaa)^*$

Ö/ a
abb
a^*b^+
abab

tokenlerini tanıyan nfa?



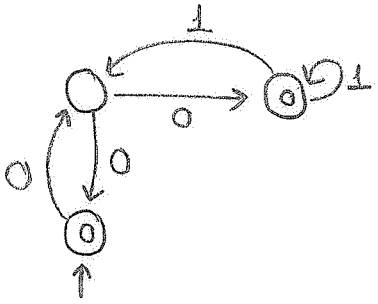
Ö2



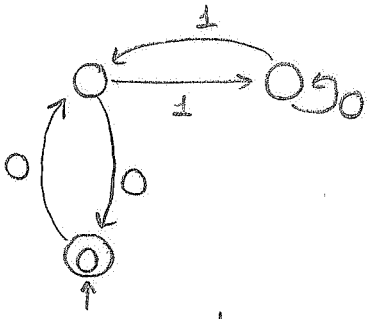
regüler ifade?

a.(a+b)*.b + λ

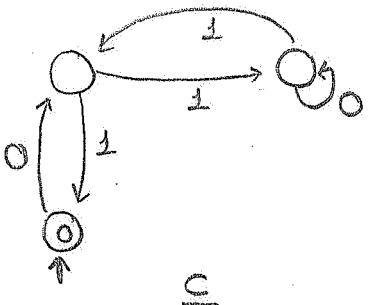
Ö3



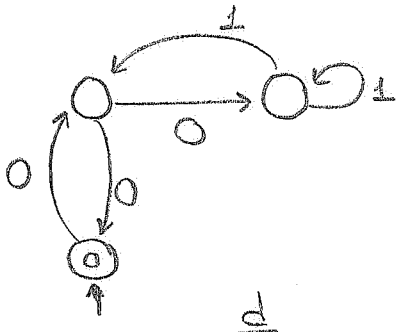
a



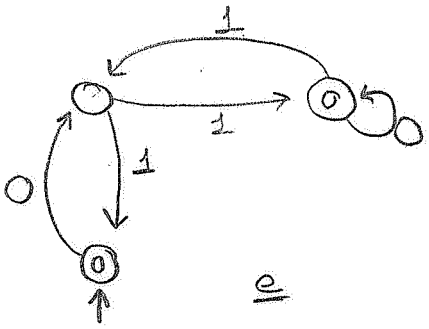
b



c



d



e

1) $\varepsilon + 0(01^*1 + 00)^*01^*$ a

2) $\varepsilon + 0(10^*1 + 10)^*10^*$ e

3) $\varepsilon + 0(10^*1 + 00)^*0$ b

4) $\varepsilon + 0(01^*1 + 00)^*0$ d

5) $\varepsilon + 0(10^*1 + 10)^*1$ c

LR Parsing

E' → E
E → E+T | T
T → T*F | F
F → (E) | id

closure fonksiyonu ⇒ bir kez uygulanır.

Goto foksiyonu ⇒ birden fazla kullanılabilir.

I_0 ⇒ Closure foksiyonu uygulanırsa bütün durumların başına nokta koyulur.

E' → .E
E → .E + T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

Noktanın E'den hemen önce olduğu kuralları uyguluyoruz.

I_1 (I_0, E)
E' → E.
E → E. + T

I_2 goto (I_0, T)
E → T.
T → T. * F

I_3 goto (I_0, F)
T → F.

I_4 goto (I_0, ( )
F → (.E)
E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

(yeniden ekliyoruz) (I_3, I_4)

I_5 goto (I_0, id)
F → id.

I_6 goto (I_1, +)
E → E+.T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_7 goto (I_2, *)
T → T*.F
F → .(E)
F → .id

I_8 goto (I_4, E)
F → (E.)
E → E.+T

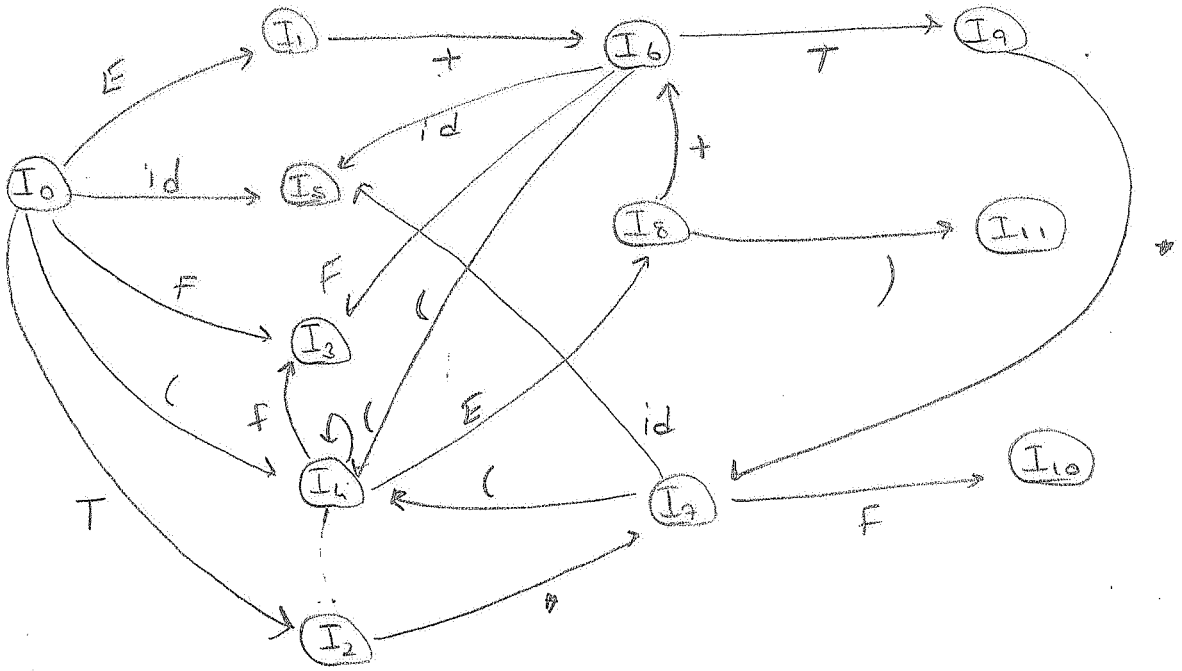
I_9 goto (I_6, T)
E → E+T.
~~E → T.~~ (buydonup)
T → T.*F

I_{10} goto (I_7, F)
T → T*F.

I_{11} goto (I_8,))
F → (E).

Gittiği kural içinde varmı.

02.03.2016
Sayfa 2



Ö/ $S \to A$

$A \to A+A \mid B++$

$B \to y$

I_0

$S \to .A$
$A \to .A+A$
$A \to .B++$
$B \to .y$

I_1 goto (I_0, A)

$S \to A.$
$A \to A.+A$

I_2 goto (I_0, B)

$A \to B.++$

I_3 goto (I_0, y)

$B \to y.$

I_4 goto $(I_1, +)$

$A \to A+.A$ ← A'nın old. durumlara bakılır.
$A \to .A+A$
$A \to .B++$
$B \to .y$

I_5 goto $(I_2, +)$

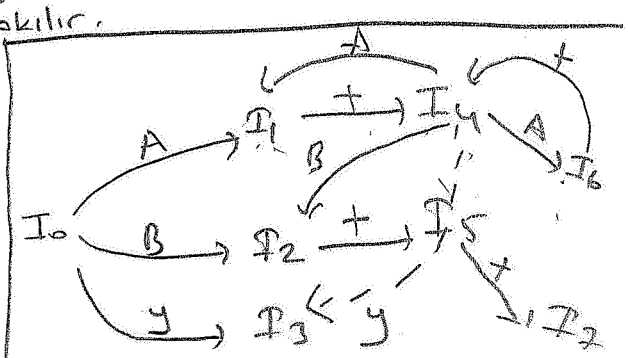
$A \to B+.+$

I_6 goto (I_4, A)

$A \to A+A.$
$A \to A.+A$

I_7 goto $(I_5, +)$

$A \to B++.$



Ö/ $S \to Aa$
$A \to aA \mid ba$ LR parsing diyagramını oluşturalım

I_0
$S \to .Aa$
$A \to .aA$
$A \to .ba$

I_1 goto (I_0, A)
$S \to A.a$

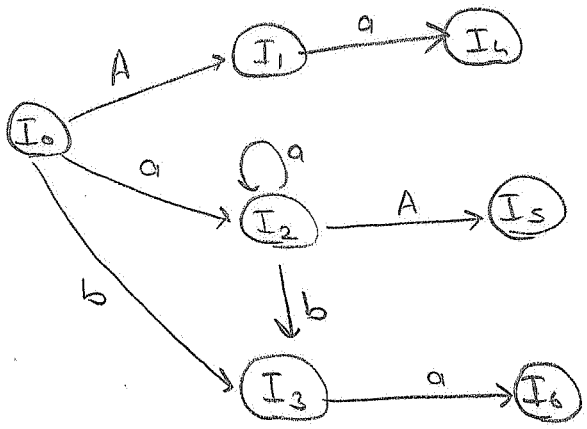
I_2 goto (I_0, a)
$A \to a.A$
$A \to .aA$
$A \to .ba$

I_3 goto (I_0, b)
$A \to b.a$

I_4 goto (I_1, a)
$S \to Aa.$

I_5 goto (I_2, A)
$A \to aA.$

I_6 goto (I_3, a)
$A \to ba.$

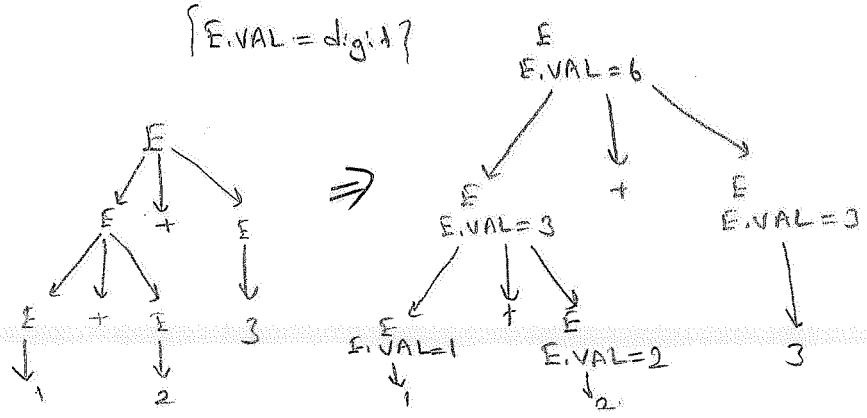


Orta seviyeli kod üretimi semantik hareketler

Semantik Hareketler

Kural	Semantik hareket
$E \to E^{(1)} + E^{(2)}$	$\{E.VAL = E^{(1)}.VAL + E^{(2)}.VAL\}$
$E \to digit$	$\{E.VAL = digit\}$
$digit \{0,1,\dots,9\}$	

Ö/ 1+2+3

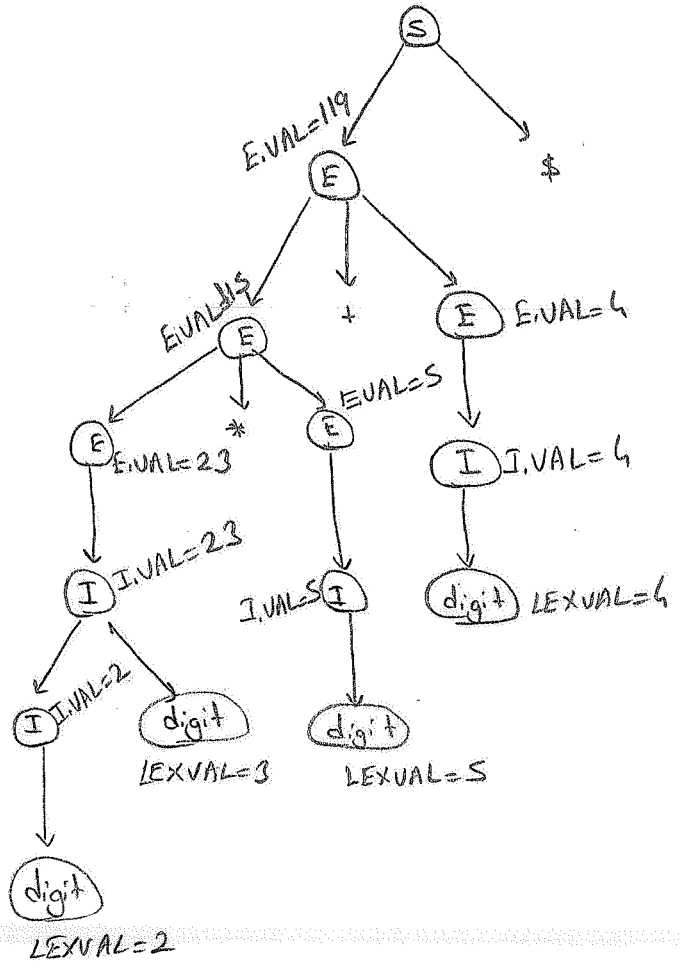


Ödev

+ ve * operatörlerinden oluşan
Stringlerin $ ile sonlandırıldığını
girişin sonuç değerini bulan bir hesap makinesi

Örneğin (23*5)+4$ ise sonucu 119 olu.

Context free gramer		semantik hareket
$S \to E\$$	$\longrightarrow$	print E.VAL
$E \to E+E$	$\longrightarrow$	$E.VAL = E^{(1)}.VAL + E^{(2)}.VAL$
$E \to E*E$	$\longrightarrow$	$E.VAL = E^{(1)}.VAL * E^{(2)}.VAL$
$E \to (E)$	$\longrightarrow$	$E.VAL = E^{(1)}.VAL$
$E \to I$	$\longrightarrow$	E.VAL = I.VAL
$I \to I\,digit$	$\longrightarrow$	I.VAL = 10*I.VAL + LEXVAL
$I \to digit$	$\longrightarrow$	I.VAL = LEXVAL



Postfix notasyonu

$ab+ \Rightarrow a+b$

$(a+b)*c \Rightarrow ab+c*$

$abc+* \Rightarrow a*(b+c)$

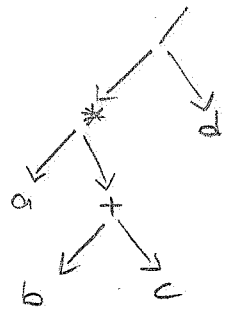
$abc+*$ 

operator gördüğümüz zaman işlemlerimizi yapıyoruz.

$(b+c)*a$

$(a+b)*(cd) \Rightarrow ab+cd*$

Syntax tree $\Rightarrow$ $a*(b+c)/d$ operatörler düğümdür.



Triples $\Rightarrow$ 3 değişken kullanıp bir operatör kullanmamız lazım.

$W = X+Y*Z$
$T_1 = Y*Z$
$T_2 = X+T_1$

3 adres kodlama

3 adres kodlamada sadece if'li ve goto'lardan oluşuyor. yüksek seviyeli yazdığımız kod buna dönüşüyor.

1 → A = B operator C
2 → A → op B
3 → goto L
4 → if A relop B goto L
5 → parametre A
≡
call P,N
6) A = B[I]
A[I] = B
7) A = addr B
A = *B
*A = B

09.03.2016
Sayfa 4

Ö/ if A<B then 1 else 0 bunun 3 adres kodunu yazınız.

1 → if A<B goto(4)
2 → T=0
3 → goto(5)
4 → T=1
5 → --- cik

Ö/ while (A<B) do
if (c<0) then X=Y*Z

3 adres kodlamasını yapınız?

100: if (A<B) goto 102
101: goto 107
102: if (c<0) goto 104
103: goto 100
104: X=Y*Z
105: X=T
106: goto 100
107: cik

Ö/ for I=1 step 1 until N do A[I]=0 for döngüsüne karşılık gelen 3 adres kodlo

1 → I=1
2 → N=10
3 ⇒ if (I ≤ N) goto 5
4 ⇒ goto 8
5 ⇒ A[I]=0
6 ⇒ I=I+1
7 ⇒ goto 3
8 ⇒ -- cik

Temel Bloklar

→ Blokları bulmak için 3 kriter gerekiyor. Bu kriterler sayesinde liderlerimizi belirlemiş oluyoruz.

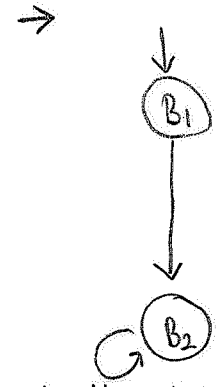
3 kriter :

1 Kriter ⇒ İlk durum liderdir.
2 Kriter ⇒ Gotonun hedefi liderdir.
3 Kriter ⇒ Şartlı gotodan sonraki satır liderdir.

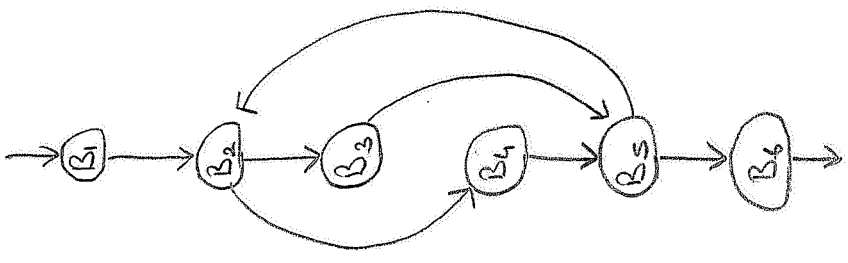
Liderler arasında kalan kısımlar bloklardır. Liderleri belirledikten sonra kontrol akış blokları düşürülür.

```
→ Pr = 0
  I = 1                                   } B1

→ T1 = 4 * I
  T2 = addr(A) - 4
  T3 = T2[T1]
  T4 = addr(B) - 4
  T5 = T4[T1]
  T6 = T3 * T5
  Pr = Pr + T6
  I = I + 1
  if I ≤ 20 goto (3)                      } B2
```

```
Ör
→ a = b = c = 1
  x[] = 0                                 } B1

→ L1 : if() goto L3                       } B2

→ A = a + 1
  B = b + 1                               } B3
  goto L4

→ L3 : a = a + 2
  b = b + 2                               } B4

→ L4 : c = c + (a * b)
  x[] = b + x[]
  if (i < 100) goto L1                    } B5

→ print (c, x[])                          } B6
```

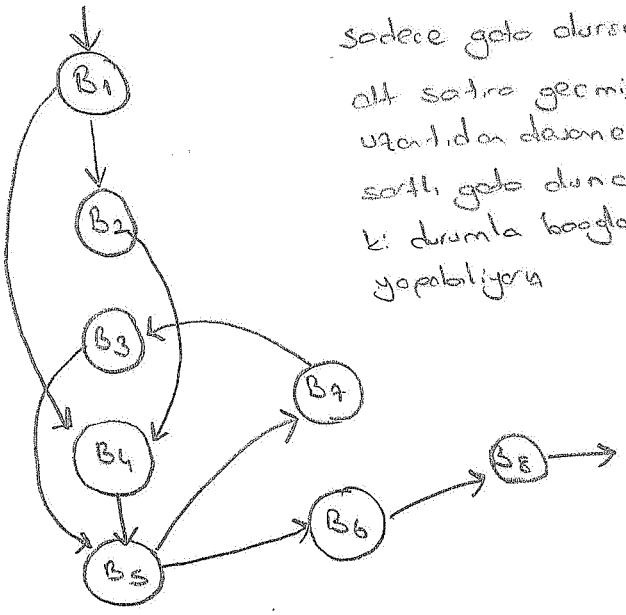


goto olduğu için kendi üzerinde döner.



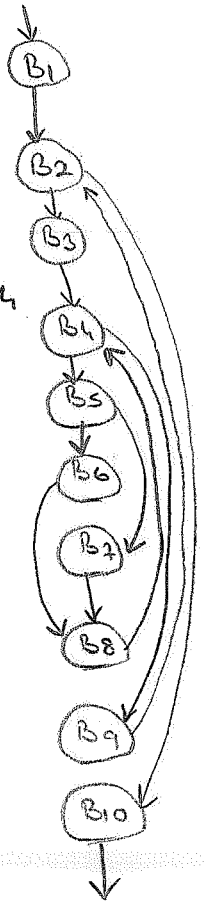
Kontrol akış diyagramını elde edin.

Ör:
→ b=i
b=2 } B_1
(if() goto B
→ e=b
goto B } B_2
→ A: goto D } B_3
→ B: c=3
b=4
a=6 } B_4
→ D: if() goto E } B_5
→ goto end } B_6
→ E: g=g+1
h=g
goto A } B_7
→ end h=9 } B_8

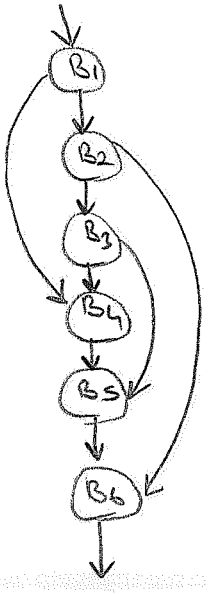


sadece goto olunca alt satıra geçmiyor. Yazdıdan devam ediyor. Sadece goto dunca olunca L: durumla bağlantı yapabiliyor.

Ör:
→ i=1 } B_1
→ L1: if() goto L6 } B_2
→ t1=0
J=1 } B_3
→ L2: if(J>n) goto L5 } B_4
→ tmp=te+t5
if() goto L3 } B_5
→ t1=t1+t5
goto L4 } B_6
→ L3: t1=t1+te } B_7
→ L4: J=J+1
goto L2 } B_8
→ L5: i=i+1
goto L1 } B_9
→ L6: t1=0 } B_{10}



Ör:
→ L1: if(i>0) goto L4 } B_1
→ L2: if(J>0) goto L6 } B_2
→ L3: if(k>0) goto L5 } B_3
→ L4: k=k-1 } B_4
→ L5: J=J-1 } B_5
→ L6: i=i-1 } B_6



Ö/ S ⇒ iSaT | a iaeiaab

T ⇒ eSE | ε bu gramer tanımıyormu tanımıyorsa çiziniz.

F ⇒ b

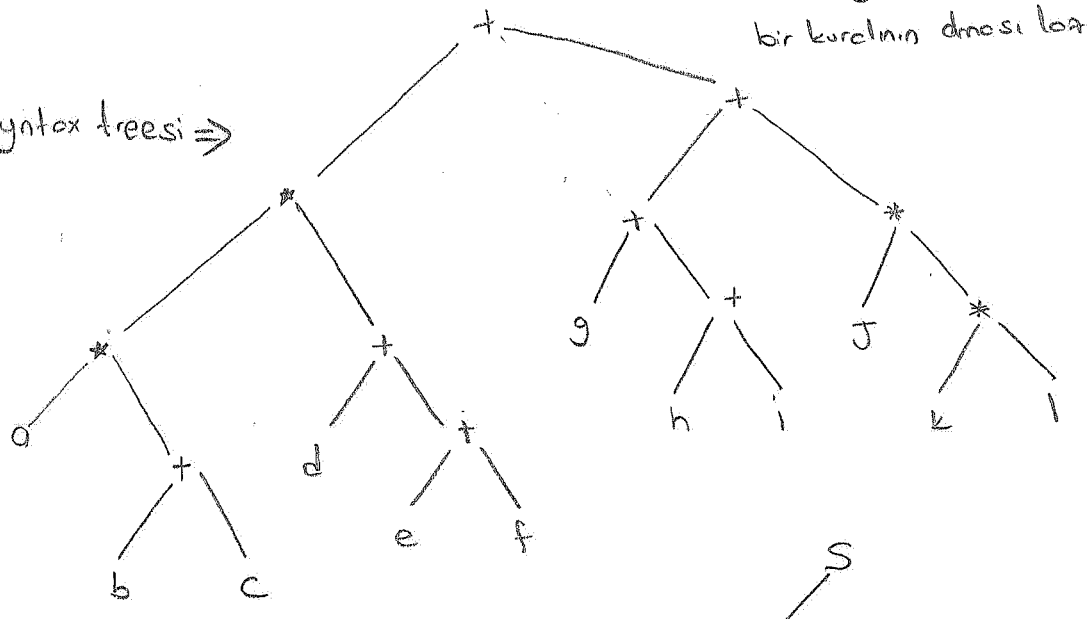
Syntax tree ⇒ düğümleri operatörden oluşan ağaçlara denilmektedir.

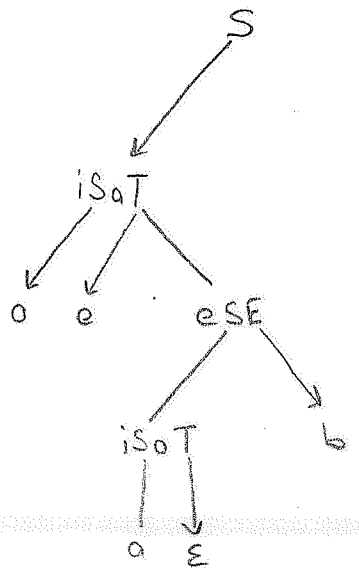
parse tree ⇒ Dil bilgisinde bulunan kuralların ağaç üzerine uygulaması işlemidir.

Ö/ ((a*(b+c))*(d+(e+f))) + ((g+(h+i)) + (J*(k*l))) ifadesinin syntax treesini ve parse treesini çiziniz.

parse tree çizilemez çünkü gramer unambiguous değildir. yukarıdaki örnekteki gibi belli bir kuralının olması lazımdır.

syntax treesi ⇒





Sınav Soruları

Ö/ $\Sigma = \{0,1\}$

a) En az 3 tane 1 içeren DFA

b) 0 ile başlayan tek uzunluklu
1 ile başlayan çift uzunluklu DFA çiziniz.

Ö/ while(x<y) do
if(z<t) then a=b+c*d

3 adres kodlamasını yapınız?

Ö/
$S \rightarrow A$
$A \rightarrow A+A \mid B++$
$B \rightarrow y$

$y+++y++$

Parse treesini çiziniz.
LR Parsingini elde ediniz.

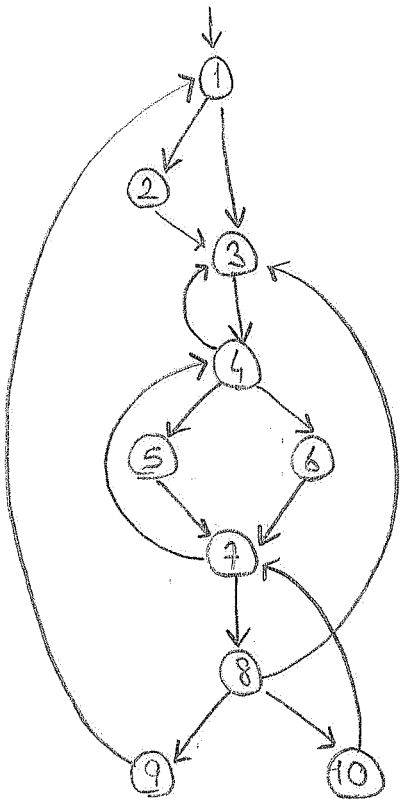
Ö/ Derleyicinin aşamalarını gösteren bir şekil çiziniz ve açıklayınız?

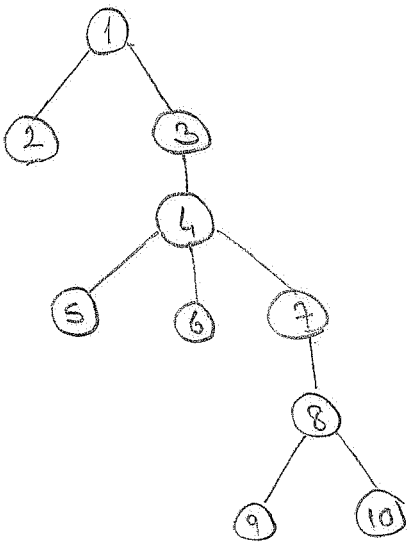
Dominator Agaci

Başlangıç düğümünden gecen her yol n düğümüne 'den gecerek gidiyorsa d n düğümünün dominantıdır.

Kök düğümü başlangıç düğümü olur.

Ör





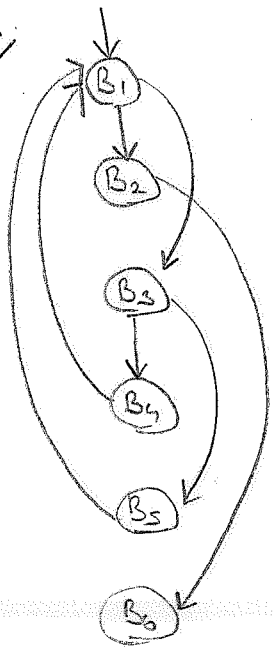
⇒ 2'nin dominantı 1'dir.

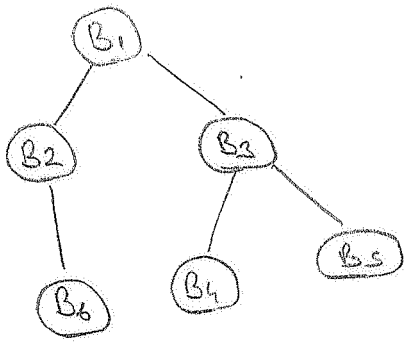
⇒ 3'ün dominatı 1'dir. Nede Çünkü 3'e gelmek için izlenecek yolları yazıyoruz.

1	1 ⇒ 2 yolda hangileri aynıysa
2	3
3	dominantıdır.

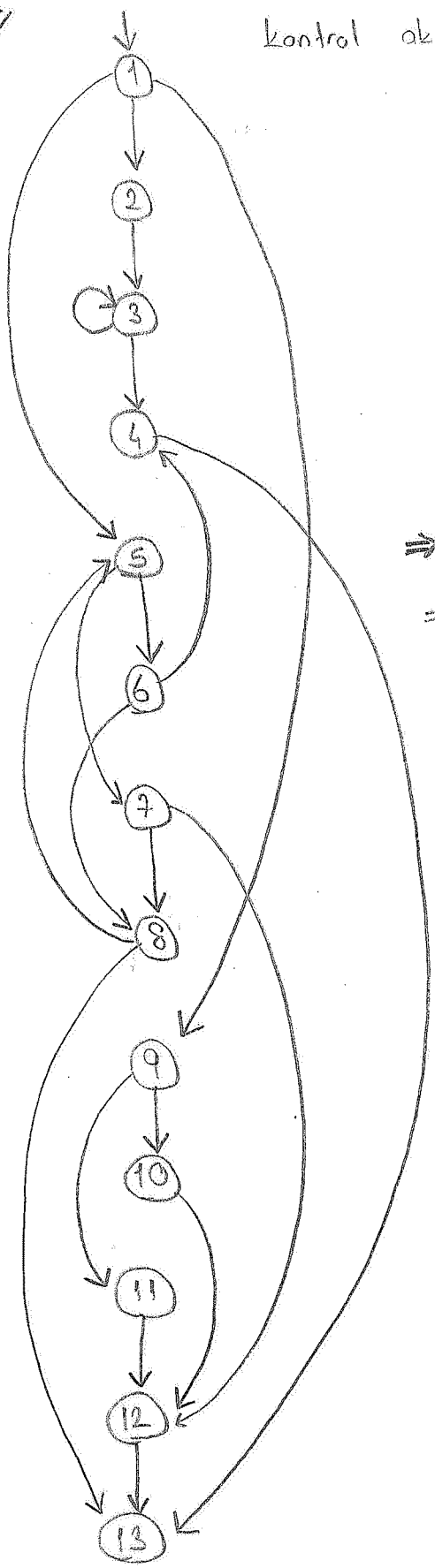
⇒ 7'nin dominantı 4'dür. Çünkü dörtten geçmek zorundayız.

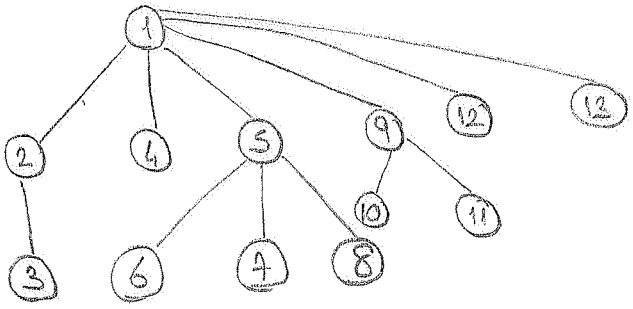
Ör





Ör Kontrol akış grafın dominator ağacını çıkartınız?





⇒ 1'den direk 2, 5, 9'a gidildiği için dominatörü 1'dir

⇒ 3'e sadece 2'den gidiliyor.

⇒ 4 düğümüne ulaşabileceğimiz durumlar

1	1
2	5
3	6
4	4

bu iki yol üzerinde ortak olan hangisi ise dominatörü odur 4'ün.

⇒ 12'ye gitmesi için;

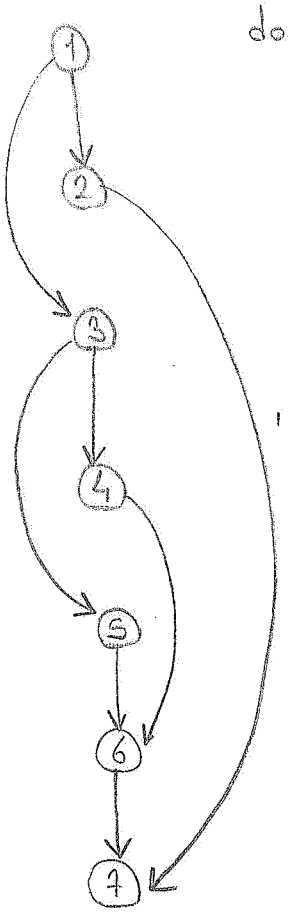
1	1	1
5	9	9
7	11	10
12	12	12

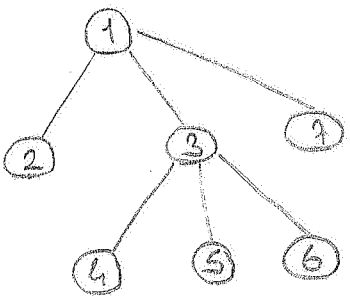
⇒ Ortak olan 1'dir Bu yüzden dominatörü 1'dir.

⇒ 8'e gitmesi için 5, 7 den geçmesi gerekir.
diğer bir yol 5, 6 dan geçmesidir.
5'ler ortak olduğu için 8'in dominatörü 5'dir.

Ör

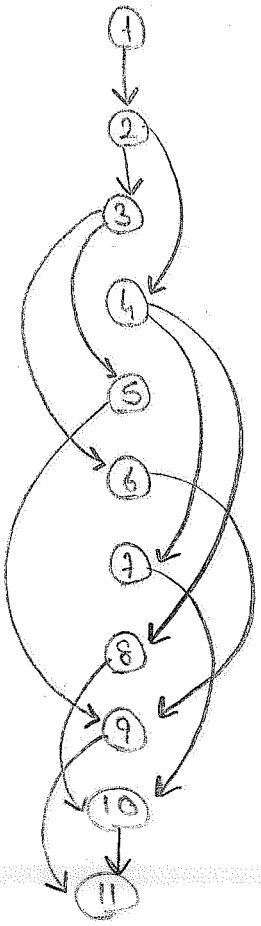
dominator agacını çiziniz?

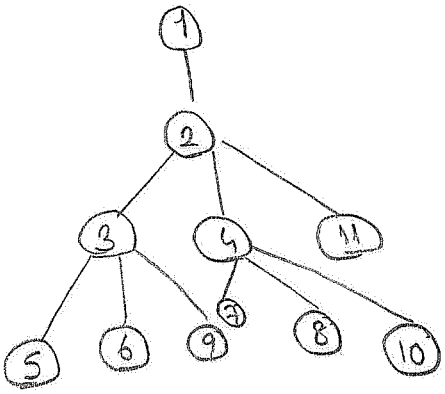




Ör

dominator agacını çiziniz?





Kod Optimizasyonu

Ö/
```
a=(b+c)*m
x=b+c
y=(b+c)*z
```
Optimize ediniz?
```
x=b+c
a=x*m
y=x*z
```

Ö/
```
for i=1 to 20
 for j=1 to 2
  write(x[i,j]);
```
Optimize ediniz?
```
for i=1 to 20
 write(x[i,1], x[i,2])
```

Ö/
```
x=1
y=2
if(x<y) jmp L1
jmp L2
L1:-----
```
Optimize ediniz?

Ö/
```
for (i=0; i<N; i++){
   if(i==0) x[i]=0
   else if (i==N) x[i]=N
     else x[i]=x[i]*c; }
```
Optimize ediniz?
```
x[0]=0;
for(i=1; i<N-1; i++)
     x[i]=x[i]*c;
     x[N]=N;
```

Ö/
```
int sc(int a, int b, int N){
 int i;
 int x,y;
 x=0;
 y=0;
for(i=0; i<=N; i++){
x=x+(4*a/b)*i+(i+1)*(i+1);
      x=x+b*y; }
      return x;
```
yandaki kod parçasını nasıl optimize etmeliyiz ki yürütme zamanı yarıya düşsün?
```
if(a>0){
 for(i=0; i<N; i++)
     x[i]=a
}
else {
 for(i=0; i<N; i++)
     x[i]=0;
}
```

Ö/
```
for(i=0; i<N; i++){
if(a>0) x[i]=a;
else x[i]=0;}
```
Kod optimizasyonunu yapınız?

Ö/
```
int a,b[256]
 a=0
for(i=0;i<256;i++)
   a+7 *b[i]      ⇒   a+=b[i]
                      a= 7*a
```
⇒ 255 kere carpmadan kurtermis oluyoruz.

Ö/
```
int a[256],b[256]
for(i=0;i<256;i++){
  if(i<128) a[i]=b[i]+2
    else  a[i]=b[i]+1;
  }
```
⇒
```
for(i=0; i<128)
    a[i]=b[i]+2
for(i=128; i<256)
    b[i]=b[i]+1
```

Kod optimizasyon teknikleri

- Degişmeyen ifadelerin döngü dışına aktarılması

```
for(i=0; i<n; i++){
    a=b+c;
    foo(a*a);
}
```

⇒

```
temp=b+c;
temp*=temp
for(i=0; i<n; i++)
    foo(temp);
```

- Koşul içeren dongülerin yeniden düzenlenmesi

```
for(i=0; i<n; i++){
    ifade 1;
    if(x)
    ifade 2;
}
```

```
if(x)
for(i=0; i<n; i++){
    ifade 1
    ifade 2;
}
else for(i=0; i<n; i++)
    ifade 1
```

- Ortak alt ifadelerin elimize edilmesi

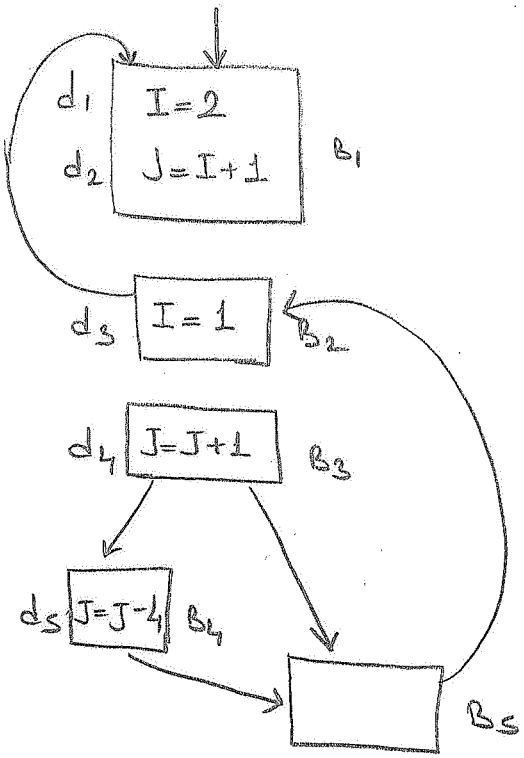
```
a=x*y+m
b=x*y+n
c=x*y+k
```

⇒

```
temp=x*y;
a=temp+m;
b=temp+n;
c=temp+k;
```

→Döngülerin ters çevrilmesi
→Döngü araştırması
→Döngü birleştirme
→Döngü yerleştirme
→Döngü daralması
→Ortak alt ifadelerin elimize edilmesi
→ifadelerin indirgenmesi
→Etkisiz kodların elimize edilmesi
→Degişmeyen ifadelerin dongü dışına aktarılması
→Koşul içeren döngülerin yeniden düzenlenmesi
→Ortak alt ifadelerin elimize edilmesi

Veri Akış Analizi



Her blogun 4 kolmanı vardır.

GEN[B]

KILL[B]

Toplam

$IN[B] = \cup\ OUT[]$

$OUT[B] = (IN[B] - KILL[B] \cup GEN[B])$

Veri akış analizini yapmak için bunlar önemli:

bloklarımız ⇒ d1, d2, d3, d4, d5

Blok içindeki tanımlamalar kendisi hariç hangi blokta varsa onu yazıyoruz.

Bloklar	GEN[B]	d1 d2 d3 d4 d5	KILL[B]	
B_1	{d1, d2}	11000	{d3, d4, d5}	00111
B_2	d3	00100	d1	10000
B_3	d4	00010	d2, d5	01001
B_4	d5	00001	d2, d4	01010
B_5	∅	00000	∅	00000

$IN[B1] = OUT[B2] = 00000$ → 1. seçenek bilmediğim için

2. seçenek ise GEN[B2] alabilirim 00100 olucak

$OUT[B1] = IN[B1] - KILL[B1] + GEN[B1]$

$GEN[B2] - 11 + 11$

B1'in içindeki tanımlamalara bakıyoruz.

I, J nerde var d3, d4, d5

B2'de sadece I tanımlaması var, sadece sol tarafta arıyoruz.

$= 00100 - 00111 + 11000 \Rightarrow 11000$

$IN[B2] = OUT[B1] + OUT[B2] = 11000 + 00000 = 11000$

$OUT[B2] = 11000 - 10000 + 00100 = 00100$

$IN[B2] - KILL[B2] + GEN[B2]$

IN[B3] = OUT[B2]

OUT[B3] = 01100 - 01001 + 00010 = 00110

IN[B4] = OUT[B3] = 00110

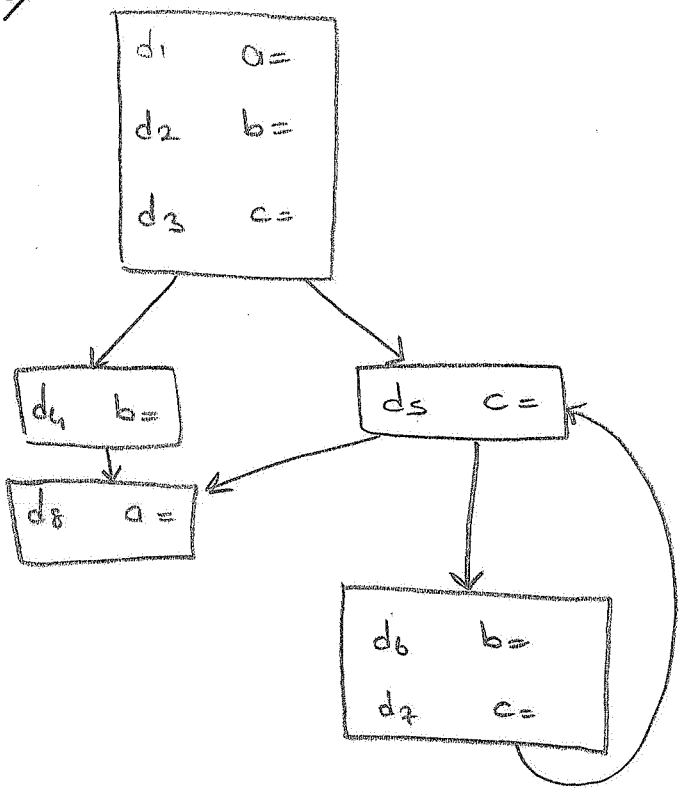
OUT[B4] = 00110 - 01010 + 00001 = 00101

IN[B5] = OUT[B3] + OUT[B4]

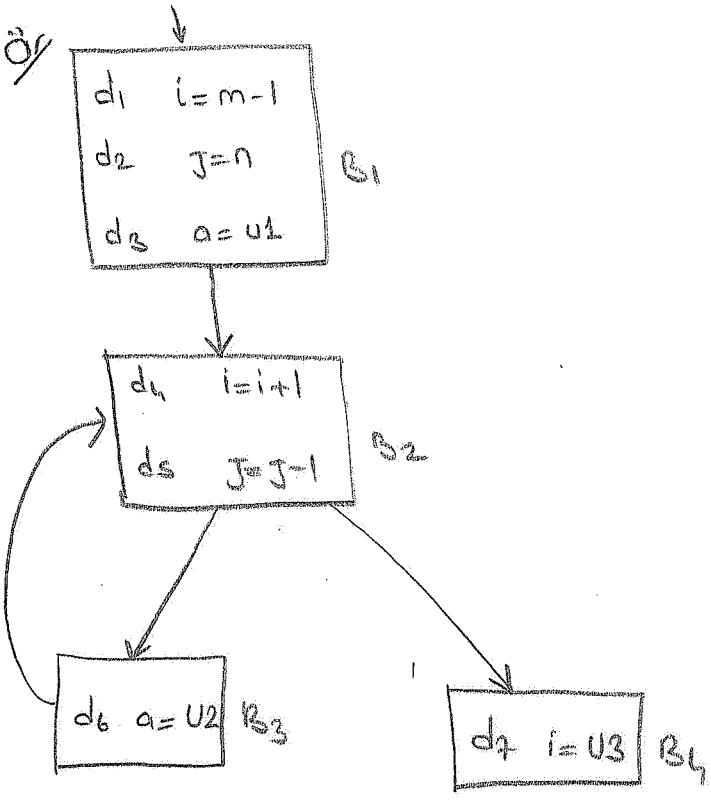
= 00110 + 00101 = 00111

OUT[B5] = 00111 - 00000 + 00000 = 00111

Ö/



Ö/



GEN, KILL ve OUT'ları hesapla Bunun iterasyonlarını bulunuz?

2. iterasyonunu yapınız?

04.05.2016

Sayfa 1

SU = Ina * 2 + 0 değeri bütün ağaç gezebilirsin

Bir koda ait sembol tablosu oluşturma

Otomata belirli tokenlar için DFA

LR Pars kodoptimum

seçecek veya in... satırın kodu kodlama

```
1 class Error
2 {
3 private static final int x=1..;
4 public static void main (String[] args)
5 }
6 int[] a= new int[10];
7 int i
8 boolean b;
9 String s=" abc
10        def";
11
12 x=1;
13 y=0;
14 i = 210
15 a[10] = 1;
16 b= false;
```

→ lexical hata
→ semantik hata

→ satır 3'de invalid tokens in number
→ aynı satırda incompatible type double

7 satır syntax hata

9 → lexical hata

11 → lexical hata

12 → semantik hata

13 → semantik hata

14 → lexical hata

ÖZ int count=0; double sum=0.0;
while(count < maxVect) { sum = sum+vect[count];
count++;}

compiler design data flow analysis pdf