ARA KOD ÜRETİMİ (2)

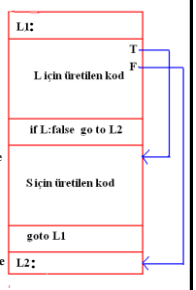
Lojik İfadelerin Çevirisi

- Lojik ifadeler iki amaçla kullanılır:
  - Lojik değerler hesaplamak için
  - Program akışını denetleyen deyimlerde (if, while..) koşul olarak
- Lojik ifade şu şekilde oluşturulabilir
  - {and, or, not} lojik operatörlerini lojik değişkenlere uygulayarak
  - {==, !=, >=, <=, <, >} karşılaştırma operatörlerini kullanarak

Lojik İfadelerin Nitelikleri

- Her E lojik ifadesi için iki nitelik oluşturulur
  - E.true: kontrol akışının, ifade **doğru** olarak değerlendirildiği zaman geçerli olacak olan hedefi
  - E.false: kontrol akışının, ifade **yanlış** olarak değerlendirildiği zaman geçerli olacak olan hedefi

Örnek: "while L do S" (L:lojik ifade)
L.true: L ifadesinin hesabı true ile sonuçlanırsa akışı S için üretilen koda yönlendirir
L.false: L ifadesinin hesabı false ile sonuçlanırsa akışı while deyimini izleyen deyim için üretilen koda yönlendirir



Lojik İfadelerin Hesabı

$E \rightarrow E_1$ or E_2

- E_1 doğru ise ifadenin kalanını incelemeden E'nin doğru olduğuna karar ver → E.true = E_1.true
- E_1 yanlış ise E_2 değerlendirilmelidir → E_1.False, kontrol akışını E_2 için üretilen kodun ilk satırına yönlendirmelidir. Bu durumda E ifadesinin true ve False nitelikleri E_2 tarafından belirlenir→

  E.true = E_2 .true ve E.False= E_2 .False

Lojik İfadelerin Hesabı (2)

$E \rightarrow E_1$ and E_2

- E_1 yanlış ise ifadenin kalanını incelemeden E'nin yanlış olduğuna karar ver →

  E.false = E_1.false
- E_1 doğru ise E_2 değerlendirilmelidir → E_1.true, kontrol akışını E_2 için üretilen kodun ilk satırına yönlendirmelidir. Bu durumda E ifadesinin true ve false nitelikleri E_2 tarafından belirlenir→

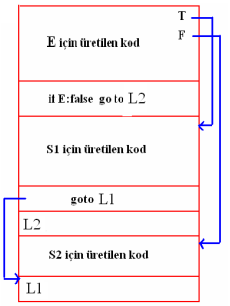
  E.true = E_2 .true ve E.false= E_2 .false

5

Lojik İfade İçinde Kontrol Akışı

- "E" lojik ifadesi için oluşturulan kod ifadeyi değerlendiren işlem komutları ve koşullu / koşulsuz dallanma komutları içerir.
- Dallanma komutlarının hedefi ifadenin E.true ve E.false çıkışları olacaktır.

L1: Örnek:
"if E then S1 else S2"



6

Yamama (Backpatching)

- Lojik ifadeyi değerlendiren kodun içerdiği koşullu / koşulsuz dallanma komutlarının hedefleri kod dörtlükleridir
  - Dallanma komutunda, hedef adres olarak, kontrol akışının yönleneceği üç adresli komut kodunu taşıyan dörtlük indisi kullanılır
- **Problem:** Dallanma komutu üretilirken, sözkonusu hedef dörtlüklerin bazıları zaten üretilmiş iken, bazıları ise henüz üretilmemiş olacaktır.

7

Yamama (2)

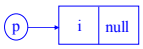
- **Çözüm:** Üretilen dallanma komutunun hedef dörtlük adresi o anda bilinmiyor ise, hedef alanı geçici olarak boş bırakılmış bir dallanma komutu üretilir. İlerde, söz konusu hedef adres oluştuğunda, komutun hedef alanı belirlenen adresle doldurulur.
- **Nasıl:** Bu tür hedef alanları boş kalan dörtlük indisleri liste yapıları içinde saklanırlar ve hedef adres belli olduğu zaman, listeler taranarak dörtlüklere ulaşılır.
- Yamama: Üretilen koda geri dönüp eksik bilgileri giderme (backpatching)
- **Her E ifadesi için E.true ve E.false bu listelere işaretçidir**

8

Yardımcı Fonksiyonlar

Kullanılan fonksiyonlar:

- makelist(i) - i değerini taşıyan bir tek düğüm içeren liste oluşturur ve geriye bu listeye bir işaretçi (p) getirir (i: dörtlük dizisine bir indis)



- merge(p1,p2) - p1 ve p2 üzerinden erişilen listeleri birleştirip oluşturulan yeni listeye bir işaretçi geri getirir
- backpatch(p,i) - p listesinde yer alan her dörtlük satırının hedef alanına "i" değerini yerleştirir

9

Problem: dallanma hedefi(1)

Dallanma hedefinin belirlenmesi

Örnek: $E \rightarrow E_1$ and E_2

1. Eğer E_1 yanlış ise, E de yanlıştır. O halde, E_1.false listesinde yer alan dörtlükler E yanlış ise yürütülecek olan dörtlüğün indisi ile doldurulmalıdır. Sonuç: anlamsal işlemler, E_1.false listesini E.false listesine eklemelidir
2. Eğer E_1 doğru ise, E_2 sınanmalıdır. O halde, E_1.true listesinde yer alan dörtlüklerin hedef alanları E_2 için üretilen kodun ilk dörtlüğü olmalıdır. Bu adresi öğrenmek için, "E_1 and E_2"nin E'e indirgenmesini beklersek gecikmiş oluruz, çünkü elimize E_2 ait kodun son dörtlük indisi geçer, ilki değil.

   Çözüm: Gramere bir eklenti yaparak istenen dörtlük adresini sakla

10

Problem: dallanma hedefi(2)

- Gramere eklenti:
  Türetim kuralının saklanmak istenen dörtlük adresinin elde edilebileceği noktasına işaretçi nonterminali olan M nonterminalini ekle

  $E \rightarrow E_1$ or M E_2

  $E \rightarrow E_1$ and M E_2

Türetim Kuralı	Anlamsal İşlemler
M → ε	{ *M.quad = nextquad* }

M.quad : dörtlük adresini tutacak olan nitelik

nextquad: birsonraki boş dörtlüğün adresi (indis değeri)

11

Lojik İfadelerin Çevirisi

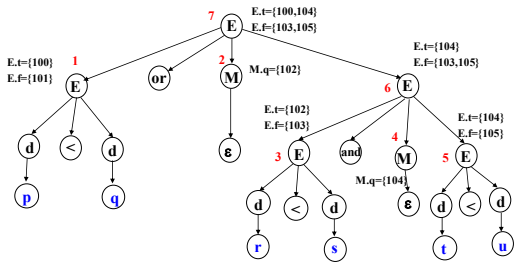
Türetim Kuralı	Anlamsal İşlemler
$E \rightarrow d$	*{ E.true = Makelist(nextquad);* *E.false= Makelist(nextquad+1);* *gen (if* d *.place "goto"___);* *gen ("goto"___)}*
$E \rightarrow d_1$ relop d_2	*{ E.true = Makelist(nextquad);* *E.false= Makelist(nextquad+1);* *gen (if* d_1*.place "relop"* d_2*place "goto"___);* *gen ("goto"___)}*
M → ε	{ *M.quad = nextquad* }

12

Lojik İfadelerin Çevirisi(2)

Türetim Kuralı	Anlamsal İşlemler
$E \rightarrow E_1$ or $M\ E_2$	*{ Backpatch (*E_1*.false, M.Quad);* *E.true = Merge(*E_1*.true,* E_2*.true);* *E.false =* E_2*.false }*
$E \rightarrow E_1$ and $M\ E_2$	*{ Backpatch (*E_1*.true, M.Quad);* *E.true =* E_2*.true;* *E.false = Merge(*E_1*. false,* E_2*. false)}*
$E \rightarrow$ not E_1	{ *E.true =* E_1*.false;* *E.false =* E_1*.true }*
$E \rightarrow (E_1)$	{ *E.true =* E_1*.true ;* *E.false =* E_1*.false }*

Örnek: "p<q or r<s and t<u"



1,2,3... indirgeme adımları

Örnek için üretilen kod

(1) indirgeme:
E.t={100}, E.f={101}
100: if p<q goto ---
101: goto ---
(2) indirgeme:
M.q={102}
(3) indirgeme:
E.t={102}, E.f={103}
102: if p<q goto ---
103: goto ---
(4) indirgeme:
M.q={104}
(5) indirgeme:
E.t={104}, E.f={105}
104: if p<q goto ---
105: goto ---
(6) indirgeme:
E.t={104}, E.f={103,105}
backpatch({102},104)
(7) indirgeme:
E.t={100,104},E.f={103,105}
backpatch({101},102)

dörtlük dizisi

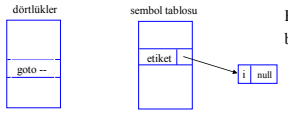
	op	*arg1*	*arg2*	*sonuç*
(100)	<	p	q	--------
(101)	goto			102
(102)	<	r	s	104
(103)	goto			--------
(104)	<	t	u	--------
(105)	goto			--------

Program Akışını Yönlendiren Deyimler

Koşulsuz dallanma: goto etiket

Durum1: Dallanma geriye doğrudur- etiket daha önce görülmüş ve sembol tablosuna eklenmiştir. Etikete sahip olan dörtlük adresi öğrenilir ve "goto adres" kodu üretilir

Durum2: Dallanma ileriye doğrudur- etiket henüz görülmemiştir ve hedef adres belirlenmemiştir. Hedefi boş bırakılmış "goto ----" kodu üretilir ve bu komutu taşıyan dörtlüğün adresi hedefi "etiket" olan dörtlükler listesine eklenir. "Etiket" sembol tablosuna eklenir ve sözkonusu listeye bir işaretçi atanır



Etiket adresi belirlendiğinde, liste taranır ve dörtlükler doldurulur

Yapısal Akış Yönlendirici Deyimler

If Deyimi: if E then S_1 else S_2

- S1 sona erince kontrolün nereye geçeceği ancak S2 için kod üretilince bilinebilir.
- S1 bir dallanma komutu ile sona ermiyorsa, bir"goto" komutu ile kontrolü S2'yi izleyen komuta geçirmek gerekir
- Türetim kuralına ek nonterminaller eklenir:
  - M: bir sonraki dörtlük adresini saklar
  - N: "goto ---" dörtlüğünü üretir

$S \rightarrow$ if E then M_1 S_1 N else M_2 S_2

17

If Deyiminin Çevirisi

- Her S için S.next nitelik bilgisi oluşturulur. "next" niteliği bir liste işaretçisidir. Listede, hedef adresi S deyimini izleyen ilk dörtlük olan koşullu / koşulsuz dallanma komutları içeren dörtlükler yer alır

Türetim Kuralı	Anlamsal İşlemler
$M \rightarrow \varepsilon$	*{ M.quad = nextquad }*
$N \rightarrow \varepsilon$	*{ N.next = makelist(nextquad); gen ("goto ----") }*
$S \rightarrow$ if E then M S_1	*{ Backpatch (E.true, M.Quad); S.next = Merge(E.false, S_1.next }*
$S \rightarrow$ if E then M_1 S_1 N else M_2 S_2	*{ Backpatch (E.true, M_1.Quad); Backpatch (E.false, M_2.Quad); S.next = Merge(S_1.next, N.next, S_2.next) }*

18

While Deyiminin Çevirisi

Türetim Kuralı	Anlamsal İşlemler
$S \rightarrow$ while M_1 E do M_2 S_1	*{ Backpatch (S_1.next, M_1.Quad); Backpatch (E.true, M_2.Quad); S.next = E.false; gen ("goto" M_1.Quad) }*
$S \rightarrow$ begin L end	*{ S.next = L.next }*
$S \rightarrow A$	*{ S.next = makelist () }*
$L \rightarrow L_1$; M S	*{ Backpatch (L_1.next, M.Quad); L.next = S.next }*
$L \rightarrow S$	*{ L.next = S.next }*

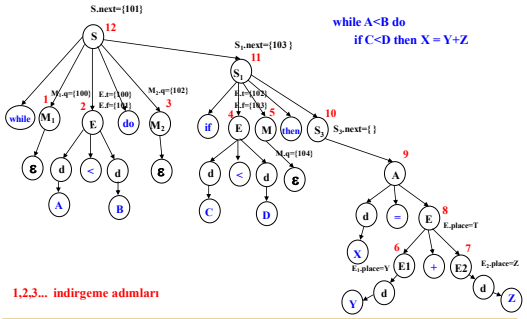
19

Atama Deyimlerinin Çevirisi (2)

Türetim Kuralı	Anlamsal İşlemler
$A \rightarrow$ d := E	*{gen(d.place ':=' E.place) }*
$E \rightarrow E_1 + E_2$	*{E.place= newtemp ; gen(E.place':='E_1.place'+'E_2.place) }*
$E \rightarrow E_1 * E_2$	*{E.place= newtemp ; gen(E.place'='E_1.place'*'E_2.place) }*
$E \rightarrow - E_1$	*{E.place= newtemp ; gen(E.place ':=' 't-eksi' E_1.place) }*
$E \rightarrow (E_1)$	*{E.place= E_1.place ;}*
$E \rightarrow$ d	*{E.place = d.place ;}*

20

Örnek:



Örnek için üretilen kod

(1) indirgeme:
$M_1.q=\{100\}$
(2) indirgeme:
E.t={100}, E.f={101}
100: if A< B goto ---
101: goto ---
(3) indirgeme:
$M_2.q=\{102\}$
(4) indirgeme:
E.t={102}, E.f={103}
102: if C< D goto ---
103: goto ---
(5) indirgeme:
M.q={104}
(6) indirgeme:
$E_1.place=Y$

(7) indirgeme:
$E_2.place= Z$
(8) indirgeme:
104: T=Y+Z
(9) indirgeme:
105: X=T
(10) indirgeme:
$S_2.next=\{\ \}$
(11) indirgeme:
backpatch({102},104)
$S_1.next=\{103\ \}$
(12) indirgeme:
backpatch({103},100)
backpatch({100},102)
S.next={101}
106: goto 100

dörtlük dizisi

	op	*arg1*	*arg2*	*sonuç*
100	<	A	B	102
101	goto			----
102	<	C	D	104
103	goto			100
104	+	Y	Z	T
105	=	T		X
106	goto			100

Prosedür Çağrıları

- Prosedür çağrıları için gramer:

S→call id (elist)
elist→ elist , E
elist → E

- Çağrının gerçeklenme adımları:

1. Eğer basit değişkenler değil iseler, argümanları hesapla
2. Argümanları çağrılan prosedürün erişebileceği bir alana yerleştir
3. Dönüş adresini bilinen bir alana yerleştir
4. Alt program çağrısını gerçekleştir

Prosedür Çağrılarının Çevirisi

- Eğer argüman bir ifade ise, E'e ingendiği zaman ifadeyi değerlendiren arakod da üretilmiş olacaktır.
- Aktarım öncesi, parametreleri saklamak üzere bir kuyruk yapısından yararlanılır – her E'e indirgenen ifade için E.place değeri kuyruğa eklenir
- Kuyrukta yer alan her argüman için ayrı ayrı "param" komutu üretilir.
- Son olarak kontrolü altprograma aktaracak "call" arakodu üretilir. Bu komut, geri dönüş adresiyle ilgili ayrıntıları da halleder

Prosedür Çağrılarının Çevirisi(2)

Türetim Kuralı	Anlamsal İşlemler
S→ call id (elist)	*{ Kuyruktan çekilen her "p" parametre bilgisi için gen ("param" p); gen (call id.place) }*
elist→ elist , E	*{ E.place bilgisini kuyruğa ekle }*
elist→ E	*{ parametre kuyruğu oluştur (boş) ve E.place bilgisini kuyruğa ekle }*

25

Bildirim Deyimleri

- Bildirim deyimi değişkenlerin tip niteliklerini bildiren bir anahtar sözcük ve bu niteliğe sahip olan değişken listesinden oluşur. Anlamsal işlemler değişken adlarının ve niteliklerinin sembol tablosuna girilmesini sağlamalıdır
- Yardımcı fonksiyon: enter(p,a) - sembol tablosunun p girişine a niteliğini ekler

Türetim Kuralı	Anlamsal İşlemler
D→ D_1 , id	*{ enter (id.place, D_1.attr); D.attr = D_1.attr}*
D → integer id	*{ enter (id.place, "integer"); D.attr = "integer" }*
D → real id	*{ enter (id.place, "real"); D.attr = "real" }*

26