İçindekiler

SCHEME (LISP)

SORU-1: Polynomials (Çokterimliler) hakkında bilgi veriniz.

Pekçok temel bilim ve mühendislik için oldukça önemli olan polinomlar (çok terimliler) basitçe bir değerin tek bilinmyenli farklı üssel terimlerle ifade edilmesidir. Bir polinomda toplama, çıkarma ve çarpma işlemleri bulunabilir. Bir polinomun herhangi bir bilinmeyen değerinin bölme işleminde bölen olması (rasyonel sayının paydası olması) durumunda bu ifade polinom olmaktan çıkar.

Örneğin

$3x^2 + x/2 + 4$ ifadesi bir polinomdur

$3x^2 + 2/x + 4$ ifadesi ise bir polinom değildir çünkü bilinmeyen değer paydada gösterilmiştir.

Diğer bir ifadeyle polinomlardaki bilinmeyenlerin üstleri tam sayı olmalıdır.

Bir polinomun derecesi, polinomdaki en yüksek üstteki bilinmeyenin derecesi kadardır.

Örneğin

$3x^7 + 2x^3 + 2x + 5$

polinomunundaki en yüksek dereceli terim $3x^7$'dir ve dolayısıyla polinomun derecesi 7′dir.

Polinom fonksiyonları

Bir polinomun fonksiyonel olarak gösterilmesi, bu polinomun belirli bir değer için çözümünün bulunması demektir.

Örneğin

$f(x) = 3x^2 + 5x + 7$

olarak tanımlanmış bir fonksiyon, anlaşılacağı üzere bir polinom fonksiyonudur çünkü fonksiyonun tanımı bir polinom ile ifade edilmiştir. Fonksiyon çözümü ise değişken yerleştirerek yapılır.

Örneğin yukarıdaki fonksiyon için f(2) çözümü:

$f(2) = 32^2 + 52 + 7$

$f(2) = 12 + 10 + 7$

$f(2) = 29$

olarak bulunur.

Polinom gösterimleri

Polinomlar cebirsel olarak işlemlere tabi tutularak farklı gösterimlere sokulabilir.

Örneğin $(x + 1)^2$ ifadesi bir polinomdur ve açıldığında

$2x^2 + 2x + 1$ polinomu bulunur. Görüldüğü üzere iki farklı gösterim de aynı değeri ifade etmektedir.

Bu gösterimin yanında bilgisayar programlamasında kolaylık sağlaması açısından polinomlar sadece çarpanların tutulduğu diziler olarak da gösterilebilir.

Örneğin

$2x^2 + 2x + 1$ polinomunu bir dizide (array) tutarken 2,2,1 olarak tutmak işlem kolaylığı sağlamaktadır.

Polinomlar yüksek dereceli terimden başlanarak tutulabileceği gibi düşük dereceli terimden başlanarak da tutulabilir.

Örneğin

$2x^2 + 2x + 1$ polinomu bir önceki örneğe benzer şekilde 1,2,2 şeklinde en düşük dereceli terimden başlanarak da tutulabilir.

Yukarıdaki bu dizide tutma işlemlerinin bir dezavantajı eksik terimlerin yer kaplamasıdır.

Örneğin

$2\times7 + 2x +1$ şeklindeki bir polinomu dizide tutarken 2,0,0,0,0,2,1 şeklinde tutmak gerekir. Burada aslında çarpanı 0 olan 6.,5.,4. ve 3. üs dereceleri de gösterilmek zorunda kalmıştır ve bu bilgi yer kaplar.

Bu boş bilginin tutulmasını engellemek için polinomların üst değerleri ile bilrlikte tutulması mümkündür.

Örneğin

$2x^{22} + 2x^{11} + 2x + 2$ polinomunu şu şekilde düşünebiliriz.

((2,22) , (2,11) , (2,1) , (2,0))

Yani değişkenin 22. kuvvetinin çarpanı 2′dir veya 0. kuvvetinin çarpanı 2′dir şeklinde gösterilebilir.

Burada dikkat edilirse bir önceki sıralı gösterime göre daha az yer kaplanmıştır (Sıralı gösterimde 22 terim tutulması gerekirken yukarıda sadece 8 terim tutulmuştur).

Yukarıdaki iç içe liste yapısı list (scheme) gibi programlama dillerinde kolaylıkla gösterilebilirken C,C++, C# veya JAVA gibi dillerde gösterim için dizi kullanma meburiyeti sonucunda örneğin bir dizinin tek elemanları ve çift elemanları yukarıdaki bilgiyi tutacak şekilde tasarlanabilir. Bu durumda yukarıdaki örnek polinomu aşağıdaki şekilde bir dizide (array) tutabiliriz:

2,22,2,11,2,1,2,0

Yukarıdaki dizide 0,2,4,6 gibi çift sayılı elemanlar çarpanları tutarken, 1,3,5,7 numaralı elemanlar üstleri tutmaktadır.

SORU-2: Anlambilimsel Tertip (Semantic Composition) hakkında bilgi veriniz.

Doğal dil işleme çalışmaları sırasında bir metinden (derlem (corpus) , paragraf , cümle veya kelimeden) çıkarılan anlamın bilgisayar tarafından bir şekilde modellenmesi gerekmektedir.

Bu modelleme sırasında kullanılan gösterim çeşitlerine anlambilimsel tertip ismi verilir. Anlambilimsel dilbilgisi (semantic grammer) kadar kesin kuralları olmayan bu gösterim şekillerinde çoğu zaman doğru veya yanlış kaygısı güdülmeden sadece bir bilginin gösterimi hedeflenir.

Anlambilimsel tertipleri cümle gösterimlerinin (Syntactic representation) ötesinde fonksiyonel ve çalışabilir tertipler olarak görmekte yarar vardır.

Örneğin en çok kullanılan ve en etkili gösterimlerden birisi birinci derece haber mantığıdır (first order predicate logic). Bu mantığa göre örneğin aşağıdaki şekilde bir gösterim yapılabilir.

"Ali uyudu" şeklindeki bir cümleyi ele alalım. Bu cümleden aşağıdaki şekilde birinci derece haber mantığı gösterimi elde edilebilir:

```
uyudu(Ali).
```

Basitçe uyudu isminde bir fonksiyonumuz varsa bu fonksiyonun parametresi de Ali olmaktadır. Bu durum özellikle birinci derece haber mantığını kullanan diller için çok önemlidir. Örneğin PROLOG yukarıdaki bu satırı doğrudan kod olarak kabul edip çalıştırabilir.

Farklı bir örnek olarak lambda matematiğinde (lambda calculus) modelleme yapmaya çalışalım.

"Ali peynir yedi" şeklindeki bir cümleyi lambda matematiği ile modellersek:

```
(λa) (λb) yedi (a,b)
```

şeklinde gösterebiliriz. Yukarıdaki bu gösterimde modelimizde bulunan değişkenler (variables) birer lambda değişkeni olarak tanımlanmıştır. Yani yukarıdaki gösterime göre a ve b isminde iki değişken yedi fonksiyonunun parametresidir. Benzer bir durum için:

```
(λa) yedi (a,peynir)
(λb) yedi (ali,b)
```

şeklinde bir gösterim de yapılabilir. Yukarıdaki ilk satırda "peynir yemek" eylemi modellenirken ikinci satırda Ali tarafından yenilen şeyler modellenmiştir.

Lamda matematiğindeki bu gösterim de LISP (veya Scheme) gibi bu gösterimi doğrudan kullanan diller için oldukça önemlidir.

Yukarıdaki örneklere ilave olarak modelin daha detaylı hale getirilmesi de mümkündür. Örneğin yukarıdaki cümlelerde fiil zaman, şahıs, kip gibi bilgilerinden arındırılmadan olduğu gibi fonksiyon haline çevrilmiştir. Oysaki bu bilgilerin de modelde birer gösterimi bulunabilir.

```
uyumak (Ali, geçmiş).
```

Yukarıdaki birinci derece haber mantığında gösterilen satıra göre Ali uyuma eylemini geçmişte yapmıştır. Elbette bu gösterim daha önceki gösterime göre daha detaylıdır ve doğal dil işlemesi yapacak bilgisayar için daha kıymetlidir çünkü ilk gösterimde bilgisayarın olay zamanı hakkında fikri yokken artık olayın zamanını parametre olarak almaktadır.

Ancak burada bir problem olayın modellenmesi sırasnda kullanılan yöntemin ne kadar bilgi içerebileceğidir. Örneğin yukarıdaki gösterime göre uyuma eyleminin geçmişte olduğunu tutuyoruz ancak bilindiği üzere "uyudu" fiili aslında söyleyen kişinin uyuma eylemi yapılırken orada olduğunu göstermektedir (örneğin uyumuş fiili de geçmiş zamanlıdır). Dolayısıyla bu ilave bilginin de tutulması gerekir.

Yukarıdaki basit örnekten anlaşılacağı üzere bir anlambilimsel model geliştirildiğinde ve kullanılırken doğal dilde çok basit ve kısa ifadelerle anlatılabilen bilgilerin analiz edilip çeşitli alternatifler ile bir ortamda tutulması gerekir.

SORU-3: Factorial (Faktöriyel) hakkında bilgi veriniz.

Bilgisayar bilimlerinde sıkça kullanılan bir örnek olan faktöriyel fonksiyonu yapısı itibari ile özyineli (recursive) bir fonksiyondur.

Yani fonksiyonun çözümünde yine kendisi cinsinden yazılma şansı vardır. Faktoriyel fonksiyonunun tanımını şu şekilde yapabiliriz:

n! = n (n-1) (n-2) … 1

Yanibasitçe faktöriyel fonksiyonu herhangi bir sayı için 1′den o sayıya kadar olan ardışık tam sayıların çarpımına eşittir.

Örneğin 5 için :

5!= 5 4 3 2 1

yani bu 5 sayının çarpımı olan 120 olarak bulunabilir.

Yukarıda iteratif olarak tanımlana faktöriyel fonksiyonunu rekürsif (özyineli) olarak tanımlamak da mümkündür.

n! = n (n-1)!

Yani bir sayının faktöriyeli o sayının bir eksiğinin faktöriyeli ile o sayının çarpımıdır. İşte bu tanım recursive (özyineli) olduğunu gösterir. Teorik olarak biliyoruz ki bütün özyineli (recursive) fonksiyonlar döngü (loop) ve bütün döngüler de özyineli fonksiyon olarak yazılabilir. Bu durumda faktöriyel fonksiyonunun çeşitli dillerdeki yazılımı aşağıda verilmiştir.

C dilinde döngü ile faktöriyel:

```
int sonuc=1;
   for(int i = 1 ;i<=n;i++){
      sonuc = sonuc * i;
}
```

Yukarıdaki kodda çarpmaya göre etkisiz eleman olan 1 ile başlatılan değişkenlerde sayısal sonuç döngü değişkeni olan ve 1′den n'e kadar değerler alan i değerinin çarpılarak biriktirilmesi (accumulate) sonucunda elde edilmiştir.

C dilinde özyineli olarak faktöriyel:

```
int fact (int n){
   if(n==1)
      return 1;
   return n*fact(n-1);
}
```

Yukarıdaki kodda özyineli bir fonksiyon olan fact fonksiyonu yazılmıştır. Fonksiyonun bitiş değeri girilen sayının 1 olmasına bağlıdır. Sayı 1 olunca sonuç da 1 olmaktadır. 1′den büyük sayılar için ise sayının faktöriyel değeri sayının kendisi ile bir eksiğinin faktöriyel değerinin çarpımına eşittir.

Scheme dilinde faktöriyel

Programlama mantığı olarak özyineli olan ve dolayısıyla döngülerin tamamının yerine özyineli fonksiyon yazıldığı Lisp dilinde ve bu dilin bir türevi olan Scheme dilinde ise aşağıdaki şekilde kodlanabilir:

(define factorial

```
   (lambda (n)
      (if (= n 1) 1
         (* n (factorial (- n 1))))))
```

Yukarıdaki kodda factorial isminde bir fonksiyon tanımlanmış ve bu fonksiyonun tek parametresi olduğu (lambda) ve isminin n olduğu belirtildikten sonra C dilindeki fonksiyonumuza benzer şekilde n değerinin 1 olması durumunda 1 diğer durumlarda ise n'in bir eksiği ile n'in çarpımı döndürülmüştür.

Prolog dilinde faktöriyel

Yapı olarak kaziye mantığı (predicate calculus) kullanan Prolog dilinde faktöriyel aşağıdaki şekilde kodlanabilir:

```
factorial(1,1).
factorial(N,F) :-
   N>0,
   N1 is N-1,
   factorial(N1,F1),
   F is N * F1.
```

Yukarıdaki kodda faktöriyel hesaplayan bir fonksiyon için N giriş değeri F ise sonuç değerini tutan değişkenlerdir. Bu fonksiyonda iki kural tanımlanmıştır.

Birinci kuralımız (ilk satır) 1 için sonucun 1olduğudur. Yani N=1 değeri sorgulandığında sonuç 1 bulunur. ikinci kuralımız ise (alttaki fonksiyon) N için F sonucunu verir.

F sonucu ise N'in 0′dan büyük olduğu değerler için N1 ismindeki ikinci değişkenin değeridir. Bu ikinci değişken ise N1′in yani N-1′in değerinin faktöriyel değeridir. Yani şimdiye kadar yazdığımız n* (n-1) mantığıdır.

SORU-4: Devamsal Geçiş Tarzı (Continuation-passing style, CPS) hakkında bilgi veriniz.

Fonksiyonel programlamada kullanılan fonksiyon tarzlarından birisidir. Buna göre bu tarzda yazılmış olan bir fonksiyon doğrudan değer döndürmek yerine, ilave bir parametre ile fonksiyondaki hesaplamaları taşır.

Devamsal geçiş tarzında yazılan bir fonksiyon çağrıldığı zaman ilave olarak bir prosedür verilerek bu fonksiyonun dönüş değerine yazılır. Doğrudan çağırmada dahili olark yapılan bazı işlemler devamsal geçiş tarzında harici olarak yapılmaktadır. Örneğin fonksiyonların dönüşleri, ara değerleri, parametrelerin çalıştırılma dereceleri ve kuyruk özyinelemeler (tail recursion) gibi.

Doğrudan çağırmalı bir fonksiyonun devamsal geçiş tarzına çevrilmesi otomatik olarak yapılabilmektedir. Bu sayede derleyiciler (compilers) bu özellikten faydalanarak iyileştirme (optimisation) yapabilmektedirler. İşin aslında bu özelliğinden dolayı devamsal geçiş tarzı programcılardan çok derleyiciler tarafından kullanılan bir tarzdır.

Scheme dilinde bazı örnek çevrimler aşağıda verilmiştir:

Doğrudan Tarz

```
(define (pyth x y)
 (sqrt (+ (* x x) (* y y))))
```

Devamsal Geçiş Tarzı

```
(define (pyth x y k)
 (* x x (lambda (x2)
          (* y y (lambda (y2)
                   (+ x2 y2 (lambda (x2py2)
                              (sqrt x2py2 k))))))))
```

Doğrudan Tarz

```
(define (factorial n)
 (if (= n 0)
     1
     (* n (factorial (- n 1)))))
```

Devamsal Geçiş Tarzı

```
(define (factorial n k)
  (= n 0 (lambda (b)
           (if b
              (k 1)
              (- n 1 (lambda (nm1)
                       (factorial
                        nm1
```

```
                      (lambda (fnm1)
                       (* n fnm1 k))))))))
```

Doğrudan Tarz

```
(define (factorial n) (f-aux n 1))
(define (f-aux n a)
 (if (= n 0)
     a
     (f-aux (- n 1) (* n a))))
```

Devamsal Geçiş Tarzı

```
(define (factorial n k)
 (f-aux n 1 k))
(define (f-aux n a k)
 (= n 0 (lambda (b)
          (if b
              (k a)
              (- n 1 (lambda (nm1)
                       (* n a (lambda (nta)
                                (f-aux nm1 nta k)))))))))
```

Yukarıda verilen iki farklı faktöriyel hesabından ilki (yani yukarıdaki 3 fonksiyondan ikincisi) doğrudan çağırmanın çevrilmiş haliyken ikincisi (yani yukarıdaki 3 fonksiyondan sonuncusu) yardımcı bir fonksiyon (f-aux) kullanarak birikimsel tarzda (accumulation style) yazılmış bir fonksiyonun devamsal geçiş tarzına dönüştürülmesini göstermektedir.

Ayrıca tembel programlama (lazy programming) kullanılarak bir fonksiyonun körilenmesi (curry) gerekirse faktöriyel hesabı aşağıdaki şekilde yazılmalıdır:

```
(define (factorial n k)
  (if (= n 0)
      (k 1)
      (factorial (- n 1) (compose k (curry * n)))))

>(factorial 5 (lambda (x) x))
>120
```

Yukarıda verilen bu fonksiyonun temel farkı k parametresi ile biriktirilen (Accumulate) değerlerin birer fonksiyon olmasıdır. Yani basitçe her çağırmada gelen fonksiyon bir önceki fonksiyon ile birleştirilmiş (compose) ve faktöriyelin başlangıç durumu (initial state) olan n = 0 durumuna gelindiğinde (n = 0 olunca 0! = 1 olduğunu hatırlayalım) bu fonksiyon birleşimlerine 1 parametresi verilerek çağrılmıştır. Bu durum daha basit şekilde aşağıdaki matematiksel gösterimde yer almıştır:

```
factorial(factorial(factorial(factorial(factorial(1)))))
```

yani basitçe factorial fonksiyonları n kadar birleştirilmiştir. n = 0 olunca birleştirme işlemi durmuş bunun yerine bu birleştirilmiş fonksiyonlara 1 parametresi verilmiştir.