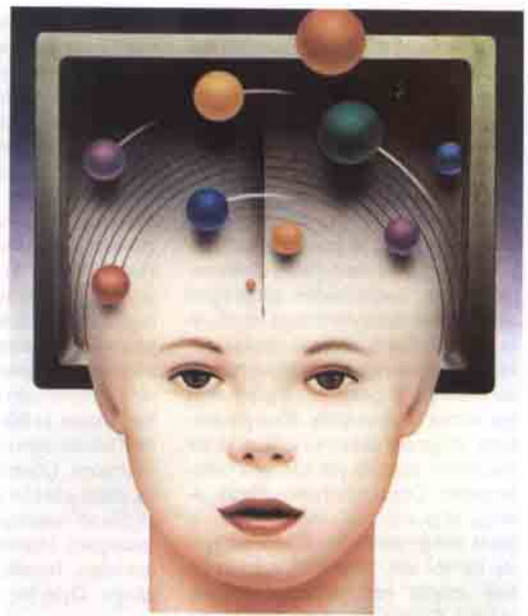
Bir Mantık Programlama Dili

PROLOG

(Bölüm VI)

Selçuk TARAL*



Temmuz ayında başladığımız bu diziyi bu sayıda noktalıyoruz. Bu bölümde Prolog programlarının yordamsal çalışmasını etkileyen bir sistem yüklemini tanıtacağız; Prolog'un içine yerleştirilmiş ve **kes** olarak çevirebileceğimiz **cut** yüklemini kullanarak bir programın yordamsal akışının yönünü değiştirebiliriz. Bir ünlem işareti (' ! ') ile gösterilen bu sistem yükleminin işlevi, nasıl yaptığını biraz sonra anlatacağımız, *geriye doğru iz sürme algoritmasının normal akışını kesmektir.*

Prolog ile programlama sanatını öğrenmek veya ilerletmek isteyen okuyuculara, stillerini geliştirirken Prolog dilinin mantıksal yönünü iyi kavramaya çalışmalarını sağlık veriri. Şimdiye kadar verdiğimiz örnek programlar da bu amaca yönelik oldu. Biraz sonra vereceğimiz örneklerden de anlaşılacağı gibi, *kes* yüklemini yanlış kullanmak programın hiç istenmeyen sonuçlar vermesine neden olabilir. Bu nedenle, Prolog programcısı olarak biraz ustalaşmadan, *kes* yüklemini sınırlı olarak kullanmamızı öneririz.

Anımsayacaksınız, önceki bölümlerde Prolog'un hesaplama yöntemini bir labirentten çıkış yollarını aramaya benzetmiştik. Geriye doğru iz sürülen bir yol sonuna kadar denendikten sonra *(depth-first search)* çözüm vermezse, Prolog son sapmanın yapıldığı noktaya dönüp (varsa) yeni bir yolu dener. Bu noktadan çıkan tüm yollar daha önce denenmiş ise, bir önceki sapma noktasına gidip aynı işleme devam eder.

Bazı sapmaların bizi bir çözüme veya yeni çözümlere götürmeyeceğini önceden bildiğimiz durumlarda, *kes* yüklemini kullanarak Prolog'un gereksiz yere çözüm aramasını önleyebiliriz. Şimdi *kes*'in etkisini genel bir Prolog tümcesi üzerinde açıklamaya çalışalım: $C = A \leftarrow B_1,\ldots,B_k, ! ,B_{k+2},\ldots,B_n$. Burada C ile gösterilen Prolog tümcesinin başı (A) programın akışı sırasında o anda sağlanmaya çalışılan bir G hedefi ile birleşir ve $B_1,\ldots,B_k$ hedefleri sırayla başarılı olursa, *kes* G hedefinin eksiltilmesi işlemini artık yalnız C tümcesine bağlar (committed); yani A yüklemi için C tümcesinin altında başka alternatif tümceler olsa bile, Prolog bunları dikkate almaz.

Şimdi, ! içeren birkaç basit programa bakalım. Şekil 9'da verilen birinci ilişki, **min(X,Y,Z)**, üç sayı arasındadır: Z değişkeni X ve Y'nin küçük olanının değerine bağlanır. Birinci kural X sayısı Y'den küçük veya eşit ise Z'nin X'e eşit olacağını deklare eder. Sonda gelen ! ise, ikinci tümcenin yeni bir çözüm için gereksiz yere denenmesine engel olur; X sayısı Y'den küçük veya eşit, ve aynı zamanda daha büyük olamaz. Burada *kes*'in kullanılışı programın mantıksal yapısını değiştirmez; sadece yeni bir çözüm getirmesi olanaksız olan geriye sapma noktalarına dönülmesine engel olur ve programın performansını artırır.

İkinci ilişki ise bildiğiniz **üye** ilişkisine çok benzemekle beraber, aralarındaki tek fark olan birinci tümcenin sağındaki ! programın anlamını önemli ölçüde değiştirir. Programa **'?-üye_kontrol (1,[2,1,3,1]).'** sorusunu yöneltirsek, birinci yüklem listenin ikinci elemanı olarak 1 sayısını bulunca, ! hesaplamayı durdurur. Amacımız sadece elemanın verilen listede olup olmadığını kontrol etmek olduğundan, ! gereksiz yere aramaya devam edilmesini önlemiş olur. Ancak artık, üye yükleminin tersine, verilen listenin elemanlarını birer birer **'?-üye_kontrol (X,[2,1,3,1]).'** sorusuyla bulmak olanaksızlaşır. Programa bu hedef verildiğinde Prolog yanıt olarak listenin sadece birinci elemanını verir: **X = 2.** Diğer elemanların da verilmesine ! engel olur.

```
min(X,Y,X) :- X <= Y, !.
min(X,Y,Y) :- X > Y, !.

üye_kontrol(X,[X|Xs]) :- !.
üye_kontrol(X,[Y|Ys]) :- üye(X,Ys).
```

Programın akışını ! ile değiştirmenin nasıl istenmeyen sonuçlar doğurabileceğini yapay bir örnekle göstermeye çalışalım. Şekil 10 da verilen programa **'?-adam(X).'** hedefini verirseniz, alacağınız yanıt **X = george** olur. Aynı programda birinci ve ikinci gerçeklerin yerlerini değiştirirsek, ! nedeniyle aynı sorunun yanıtı, *çözüm bulunamadı* anlamındaki

```
insan(george).
insan(mary).
erkek(george).

adam(X) :- insan(X), !, erkek(X).
```

* *TÜBİTAK, Bilgi İşlem Daire Başkanı.*

bir 'no' şeklindedir; *mary* erkek olmadığı için ve ! geriye dönüp yeniden aramaya izin vermediğinden sonuçta *george'u* bulmak olanaksızlaşır.

Şimdi yine program akışını etkileyen ama çok daha basit bir başka sistem yüklemini tanıtalım. Sistem yüklemi **fail** hiçbir argüman almaz ve *daima başarısızlıkla sonuçlanır !* Daha önce gösterdiğimiz gibi, bazı hedeflere programın dışından ';' işaretini kullanarak yeniden çözüm aratmak yerine, *fail* yüklemiyle geriye doğru iz sürme işini otomatiğe alabiliriz. Örneğin, birinci bölümde verdiğimiz programa, süleyman'ın tüm çocuklarını ekrana bir defada yazdırmak amacıyla aşağıdaki bileşik (composite) hedefi verebiliriz:

?- baba(süleyman,X), write(X), nl, fail.

Burada bir sistem yüklemi olan **write** **X**'in değerini ekrana yazar. Ondan sonra gelen **nl** yüklemi ise imlece satır atlatır. Bu sorunun yanıtı olarak, süleyman'ın çocukları *(ibrahim* ve *ayşe)* alt alta yazıldıktan sonra, programın bilgi tabanında süleyman'ın başka çocuğu verilmediği için, son olarak ekrana 'no' gelecektir. Burada *fail* yüklemi birinci çözümün ekrana gelmesinden sonra başarısız olacağından, geriye doğru iz sürme mekanizmasını çalıştırıp, Prolog'u yeni bir çözüm aramaya zorlar. Kendisinden önce gelen yüklemler (nl ve write) Prolog sistemine ait olup yeni bir çözüm üretmeleri zaten söz konusu değildir. Geriye doğru iz sürme **baba** yüklemine kadar gelir ve X süleyman'ın bir sonraki çocuğuna bağlanıp *write* ile ekrana yazılır. Bu işlem bilgi tabanında süleyman'ın başka çocuğu kalmayana kadar devam eder.

Şimdi ! ve **fail** yüklemlerinin birlikte kullanılmasına bir örnek verelim. Şekil 11'de verilen program *seçilebilir* insanları tanımlar; *30 yaşından büyük her vatandaş, o arada ağır suçlu değilse, seçilme hakkına sahiptir.* Programın ilk tümcesi virgüller arasında verilen istisnai durum içindir. X ağır suçlu bir kişiyse, başka yere bakmaya gerek kalmaz; ! ile yeniden aramayı önleyip, *fail* ile de bu kişinin seçilemez olduğunu *deklare* edebiliriz.

```
seçilebilir(X) :-
    ağır_suçlu(X), !, fail.
seçilebilir(X) :-
    vatandaş(X),
    yaş(X,Y),
    Y >= 30.

vatandaş(ahmet).     yaş(ahmet,32).
vatandaş(ayşe).      yaş(ayşe,25).
vatandaş(hasan).     yaş(hasan,45).

ağır_suçlu(hasan).
```

Şekil 11'de verilen program, ismi verilen bir kişinin seçilebilir olup olmadığını verir. Örneğin, '**?- seçilebilir(hasan).**' sorusunun yanıtı 'no' olacaktır. Ancak programa, '**?- seçilebilir(X).**' sorusu yöneltirsek yanıt, belki beklediğiniz gibi seçilebilir kimselerin listesi değil, sadece bir 'no' olur. Programın bu davranışının nedenini açıklamayı okuyucuya bırakalım ve tüm seçilebilir kişileri alt alta dökecek bir hedef verelim:

?- vatandaş(X), seçilebilir(X), write(X), nl, fail.

Dizimizi ünlü *dört-renk problemi* ile kapatmak istiyoruz. Herhangi bir haritanın, *iki komşu bölgenin aynı renkte olmaması şartıyla,* boyanabilmesi için dört renk gerekli ve yeterlidir. En az dört rengin *gerekli olduğu* uzun zamandır bilinmesine karşın, dört sayısının *aynı zamanda yeterli olduğu* ancak 1976 yılında Amerika'da **Illinois Üniversitesi**'nde ispat edilmiştir. Problemi Prolog dilinde tanımlamak için, önce haritayı hangi veri yapısıyla göstereceğimize karar vermemiz gerekir.

Haritayı bir *bölgeler listesi* olarak gösterebiliriz; her *bölge* bir *isim,* bir *renk* ve *komşu bölgelerin renklerinin listesinden* oluşan karmaşık bir terimdir: **bölge(Ad,Renk,Komşular).** *Renk* bölgenin rengini, *Komşular* ise komşu bölgelerin renklerini içeren listeyi gösterir. Örneğin batı Avrupa'nın haritası şöyle gösterilebilir:

harita(b_avrupa,[bölge(portekiz,P,[E]),bölge(ispanya,E,[F,P]),bölge(fransa,F,[E,I,S,B,G,L]), bölge(belçika,B,[F,H,L,G]),bölge(hollanda,H,[B, G]),bölge(almanya,G,[F,A,S,H,B,L]),bölge(lüksemburg,L,[F,B,G]),bölge(italy,I,[F,A,S]),bölge isviçre,S,[F,I,A,G]),bölge(avusturya,A,[I,S,G])]).

Şekil 12'de verilen programda, haritayı boyayan yükleme **r_harita,** bölgeleri boyayana ise **r_bölge** adını verdik. Programa batı Avrupa haritasını boyatmak için '**?- renklendir(b_avrupa,Harita).**' hedefini veririz; tabiî programa yukarıda verilen batı Avrupa haritasını (bir gerçek olarak), girmeyi unutmamak şartıyla !

```
r_harita([Bölge¦Bölgeler],Renkler) :-
     r_bölge(Bölge,Renkler),
     r_harita(Bölgeler,Renkler).
r_harita([],Renkler).

r_bölge(bölge(Ad,Renk,Komşular),Renkler) :-
     seç(Renk,Renkler,Renkler1),
     üyeler(Komşular,Renkler1).
üyeler([X¦Xs],Ys) :-
     üye(X,Ys), üyeler(Xs,Ys).
üyeler([],Ys).

renklendir(Ad,Harita) :-
     harita(Ad,Harita),
     renkler(Ad,Renkler),
     r_harita(Harita,Renkler).

renkler(X,[kırmızı,sarı,mavi,beyaz]).
```

Programın kalbi, bölgeleri boyayan *r_bölge* yüklemidir. Burada *seç* ve *üyeler* yüklemleri iki işlev görürler; bölgelere renk vermek ve daha önce verilmiş renkleri test etmek. Dikkat ederseniz, *üyeler* yüklemi iki şekilde kullanılabilir; bir listenin elemanlarının ikinci bir listenin içinde olup olmadığını kontrol etmek ve eğer birinci listenin içinde değişkenler varsa, onlara ikinci listeden aldığı değerleri bağlamak. İki komşu bölgeye aynı renk verilmişse, *üyeler* yüklemi sağlanmaz ve *seç* ile yeni bir renk denemek için son boyama noktasına geri dönülür; yani program deneme-yanılmayla haritayı iki komşu aynı renkte olmayacak şekilde boyar.

Dizinin hazırlanışında kullanılan kaynaklar:

1) ***The Art of Prolog,*** *Leon Sterling, Ehud Shapiro.*

2) ***Knowledge Systems and Prolog,*** *Adrian Walker (editor).*

3) ***Prolog Programming,*** *Claudia Marcus.*