

Yapay Zekâ: Ders Notu 5

Yılmaz Kılıçaslan

13 Mart 2024

Anahtar Kavramlar: Mantık Programlama, Horn Cümlesi, Prolog, Olgu, Kural, Sorgu.

ÖNERMELER MANTIĞI İLE PROGRAMLAMA

1 Giriş

Mantık programlama, tümdengelim mantığının doğrudan bir programlama paradigması olarak kullanılmasıdır. Mantığın akıl yürütme bilimi ve tümdengelim mantıkta akıl yürütmenin de bir dizi öncülden bir sonuç çıkarma süreci olduğunu bir kez daha hatırlayalım. Bir mantık programı yazmak, bilgisayara öncül olarak kullanılacak bir dizi önerme sağlamaktır. Mantık programını kullanmak ise, bilgisayarı bir hedef önerme ile sorgulamaktır. Mantık programlama çerçevesinde gerçekleşecek bilgisayarlı akıl yürütmeye gelince; bu, öncül önermelere dayanarak hedef önermenin ispatlanması sürecinden başka bir şey değildir. Kısaca ifade edersek, bir mantık programı ve bu programa göre sorgulanacak bir hedef önerme birlikte bir çıkarım oluşturur. Bu çıkarımda yer alan önermelerin her biri, daha sonra açıklayacağımız bir **Horn cümlesi** olarak kodlanır:

(1) ÇIKARIM OLUŞTURMA FAALİYETİ OLARAK MANTIK PROGRAMLAMA:



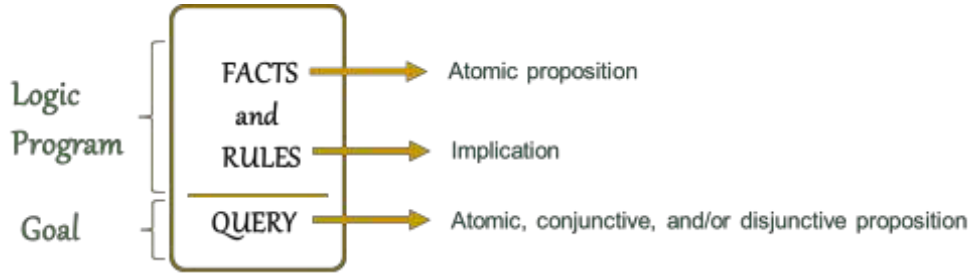
Daha açık ifadeyle, mantık programlamanın temel pratik işlevi, halihâzırda mevcut olan doğru önermelerden yeni doğru önermeler çıkarmaktır. Bu iş, minimal bir tümdengelimli mantık sisteminin iki bileşeninden birisi olan türetim bileşeninin görevidir. Sistemin diğer bileşeni ise, meşru önerme cümlelerini oluşturmaktan sorumlu sözdizimi bileşenidir. Bu derste; mantık programlamanın envanterindeki önerme türlerini ve önermeler mantığı ile sınırlı bir bakış açısıyla söz konusu önermelerin bir mantık programlama dili olan Prolog'taki sözdizimsel yapısını (bkz. Bölüm 2) ve mantık programlamanın türetim bileşeninin, program ifadelerinden bir sonuç ifadesi çıkarmak için kullandığı tümdengelim kurallarını (bkz. Bölüm 3) inceleyeceğiz.

2 MANTIK PROGRAMLAMADA ÖNERME TÜRLERİ

2.1 Minimal bir Önerme Tipi Envanteri

Mantık programlamada kullanılan önermeler; **olgular**, **kurallar** ve **sorgular** olmak üzere üç kategoriye ayrılır:

(2) MANTIK PROGRAMLAMADA ÖNERME TÜRLERİ:



Olgular atomik, yani önermeler mantığının herhangi bir operatörünü barındırmayan önermelerdir. Kurallar ise belirli türden implikasyonlardır. Bir mantık programı olgular ve/veya kurallar kümesidir. Sorgular, yani verilen bir mantık programından çıkarılacak sonuçlar, atomik, birleşimli ve/veya ayrık önermeler olabilir.

Şunu da belirtmek gerekir ki; Prolog'un kökleri, önermeler mantığına değil, birinci dereceden yüklem mantığına (İng. First-Order Predicate Logic - FOPL) dayanır. Tam gücünü ancak bu ikinci mantık sisteminin niceleyici operatörlerini kullandığında kazanır. Bununla birlikte, nicelenecek değişkenlerle ilgili ayrıntılara boğulmadan, Prolog ile mantık programlamanın nasıl gerçekleştirildiği hususunda biraz fikir edinmek için, önermeler mantığının sınırları içerisinde kalarak bazı çok küçük Prolog programlarını yazmak ve yorumlamak genel sözdizimsel ve anlamsal çerçeveyi kavramak açısından faydalı olacaktır.

2.2 Olgular

Tek bir olgu, Prolog ile minimal olarak bir noktanın (.) takip ettiği bir önerme sembolü ile kodlanabilir. Aşağıda böyle bir önerme ile oluşturulmuş, mümkün olabilecek en küçük Prolog programlarından biri yer almaktadır:

(3) MÜMKÜN OLABİLECEK EN KÜÇÜK PROLOG PROGRAMI:

```
p.
```

Bu program yalnızca şunu söylemektedir: p önermesi doğrudur. Önermeyi temsil etmek için küçük p harfinin kullanıldığına dikkat edin. Bu bir zorunluluktan kaynaklanmaktadır. Zira, daha sonra göreceğimiz gibi, Prolog'un sözdizimsel kuralları, önermeleri temsil eden ifadelerin büyük harfle başlamasına izin vermez.

Birleşik bir önermenin Prolog'taki karşılığı, her bir atomik birleşenin ayrı bir olgu olarak görüldüğü bir programdır. Örneğin, aşağıda Prolog programı, $p \wedge q$ ifadesinin karşılığıdır:

(4) BİRLEŞİMLİ BİR ÖNERME İÇİN MÜMKÜN OLABİLECEK EN KÜÇÜK PROLOG PROGRAMI:

```
p.  
q.
```

Bir Prolog programında bu şekilde birleştirilecek önermelerin sayısında herhangi bir kısıtlama yoktur:

(5) DÖRT ÖNERMENİN BİRLEŞMESİ İLE OLUŞAN BİR PROLOG PROGRAMI:

```
p.  
q.  
r.  
p1.
```

Fakat, şunu da belirtmek gerekir. Yukarıdaki gibi programlarda açık olarak kullanılan bir birleşme (ve) operatörü bulunmamaktadır. Birleşme yorumu, her zaman yaptığımız gibi, cümleleri ardışık olarak sıralarken kendiliğinden bu ardışıklığa kattığımız bir yorumdur. Aslında, Prolog'un bir birleştirme operatörü mevcuttur. Virgül (,) Prolog'un birleştirme operatörünü sembolize eder. Ancak, virgül, olguları ya da kuralları birleştirmek için, yani bir ana operator olarak kullanılamaz. Dolayısıyla, aşağıdaki kullanım sözdizimsel olarak hatalı olacaktır:

(6) ',' OPERATÖRÜNÜN HATALI KULLANIMI:

```
p,  
q.
```

Prolog'un envanterinde bir ayırma (veya) operatörü de bulunmaktadır. Noktalı virgül (;), bu amaçla kullanılmaktadır. Fakat, tıpkı virgül gibi, bu operatör de ana operatör olarak kullanılamaz. Örneğin, aşağıdaki program sözdizimsel açıdan doğru bir Prolog program olarak Kabul edilemeyecektir:

(7) ';' OPERATÖRÜNÜN HATALI KULLANIMI:

```
p;  
q.
```

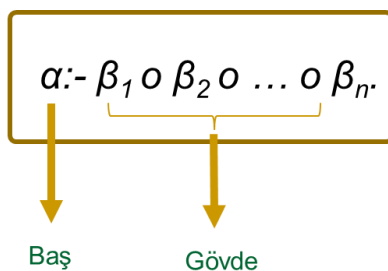
Aşağıda göreceğimiz gibi; virgül ve noktalı virgül, kuralların gövdesinde ve sorgularda, sırasıyla birleştirme ve ayırma operatörleri olarak kullanılabilirler.

2.3 Kurallar

Genel olarak, kurallar aşağıdaki formda yapılandırılmış ifadelerdir:

(8) KURAL FORMATI:

```

$$\alpha :- \beta_1 \circ \beta_2 \circ \dots \circ \beta_n.$$


Baş      Gövde


```

Burada $n \geq 0$ 'dır ve \circ ',' veya ';' olabilir.

Kuralın *başı* (α) atomik tek bir önerme olmak zorunda iken *gövdesi* (β_i 'ler) bileşik bir önerme olabilir. Gövde ve baş birlikte, birincisinin ikincisini gerektirdiği ettiği bir implikasyon oluşturur:

(9) PROLOG KURALININ MANTIKSAL EŞDEĞERİ:

$$\alpha \leftarrow (\beta_1 \circ \beta_2 \circ \dots \circ \beta_n)$$

VEYA

$$(\beta_1 \circ \beta_2 \circ \dots \circ \beta_n) \rightarrow \alpha$$

Yani, kurallar ana operatörü implikasyon olan karmaşık formüllere karşılık gelir. Gövdedeki önermeleri ayıran virgüller (,) ayırma (ve) operatörü olarak yorumlanır. Örneğin, yukarıdaki kuralın mantıksal yorumu şöyle olabilir:

(10) (8) İÇİN OLASI MANTIKSAL YORUM:

$$\alpha \leftarrow (\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n)$$

ya da daha bilindik bir gösterimle, şu da olabilir:

(11) (8) İÇİN OLASI MANTIKSAL YORUM:

$$(\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n) \rightarrow \alpha$$

Aşağıdaki $(p \wedge q) \rightarrow r$ için meşru bir Prolog programıdır:

(12) BİR KURAL İÇİN PROLOG PROGRAMI:

$$r :- p, q.$$

Ayrıklık operatörü de bir kuralın gövdesinde kullanılabilir. İşte $(p \vee q) \rightarrow r$ için bir Prolog programı:

(13) BİR KURAL İÇİN PROLOG PROGRAMI:

$$r :- p; q.$$

Kurallar da ayrı ayrı yazıldıklarında birleşik olarak yorumlanırlar:

(14) BİR DEN FAZLA KURAL İÇİN PROLOG PROGRAMI:

```
r:- p, q.  
r1:- p1, q.  
p2:- q2.
```

Ancak, bir kez daha tekrarlırsak, kurallar ',' ve ';' sembolleri ile birleştirilemez veya ayrılamazlar. Aşağıdakiler kabul edilebilir Prolog programları değildir:

(15) SÖZDİZİMSSEL HATALI PROLOG PROGRAMLARI:

a. ',' OPERATÖRÜNÜN HATALI KULLANIMI:

```
r :- p, q,  
r1:- p1, q.
```

b. ';' OPERATÖRÜNÜN HATALI KULLANIMI:

```
r :- p, q;  
r1:- p1, q.
```

2.4 Sorgular

Sorgular hedef cümleleridir. Programdan bilgi alma araçlarıdır. Programın parçası değildir. Tıpkı program oluşturan önermeler gibi sonlarına bir nokta konur:

(16) ATOMİK BİR SORU:

```
?- p.
```

Bir programa ilişkin bir sorgulama, sorunun programın mantıksal bir sonucu olup olmadığını belirlemektir. Bu nedenle, yukarıdaki sorgu mantıksal olarak yorumlandığında, "*p* önermesi verilen programdan *zorunlu* olarak çıkar mı?" veya eşdeğer bir ifadeyle, "Bunu programa dayanarak kanıtlayabilir miyiz?" gibi bir soruya karşılık gelir. Ancak, daha doğal bir yorumlama şu olacaktır: *p doğru* mu?

Sorgular birleşik veya ayırık önermeler içerebilir:

(17) a. BİRLEŞİK BİR SORU:

```
?- p, q.
```

b. AYRIK BİR SORU:

```
?- p; q.
```

Hem birleşme (ve) hem ayırma (veya) operatörlerini içeren sorgular oluşturmak da

mümkündür:

(18) KARMAŞIK BİR SORU

?- ((p, q); (p1, q1)).

2.5 Olumsuzlama

Mantık programlama, olumsuzlamayı *başarısızlık-olarak-olumsuzlama* stratejisinden türetir: α 'nın elde edilmesindeki başarısızlıktan *değil*(α) türetilir. “Kabaca bu, yorumlayıcının α 'yı doğrulamada başarısız olması durumunda $\neg\alpha$ 'nın doğrulanmış olarak kabul edileceği anlamına gelir” (Kunen 1987).

Yalnızca bir hedef, yani bir kuralın gövdesindeki bir önerme veya bir sorunun bileşeni olan bir önerme olumsuzlanabilir. ‘not’ Prolog’un kullandığı olumsuzlama operatörlerinden birisidir:

(19) PROLOG’DA OLUMSUZLAMA OPERATÖRÜNÜN KULLANIMI:

a. KURALIN GÖVDESİNDE:

$r :- \text{not}(p), q.$

b. SORGUDA:

?- $\text{not}(p).$

Olumsuzlama operatörünü bir kuralı, bir kuralın başını veya bir olguyu olumsuzlamak amacıyla kullanmayız:

(20) OLUMSUZLAMA OPERATÖRÜNÜN HATALI KULLANIMI:

a. KURAL OLUMSUZLAMA:

$\text{not}((r) :- p, q)).$

b. KURAL BAŞI OLUMSUZLAMA:

$\text{not}(r) :- p, q.$

c. OLGU OLUMSUZLAMA:

$\text{not}(p).$

2.6 Horn Cümleleri

Prolog Horn cümleleri üzerine inşa edilmiştir. Bu nedenle, bu cümleler hakkında birkaç söze ihtiyaç vardır. Bir Horn cümlesi, bir (pozitif) atomik önerme veya bir atomik önermenin olumsuzlaması olan, eğer birden fazla iseler ayırma operatörü ile ayrılmış olan ve en fazla bir tanesinin pozitif olabileceği işlenenlerden oluşan bir cümledir. Bu tanımdan üç tür Horn cümlesi olduğu sonucu çıkar. Aşağıda Horn cümlelerinin türleri ve sahip oldukları bazı çeşitli biçimler verilmiştir:

(21) HORN CÜMLE TİPLERİ:

Tip Adı:	Ayrık Form:	İmplikasyon Formu:	Prolog Formu:
Belirli	$\alpha \vee \neg\beta_1 \vee \neg\beta_2 \vee \dots \vee \neg\beta_n$	$\alpha \leftarrow \beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n$	$\alpha :- \beta_1, \beta_2, \dots, \beta_n.$
Olgu	α	$\alpha \leftarrow \text{doğru}$	$\alpha.$
Hedef	$\neg\beta_1 \vee \neg\beta_2 \vee \dots \vee \neg\beta_n$	$\text{yanlış} \leftarrow \beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n$	$?- \beta_1, \beta_2, \dots, \beta_n.$

burada α ve β_i 'nin her biri atomik bir önermedir.)

Tam olarak bir pozitif atomik önerme içeren bir Horn cümlesine belirli cümle denir. Bu tek pozitif önerme Prolog'da kuralın başı olarak görünür. Olgular boş bir gövdeye sahip kurallar olarak düşünülebilir. Hiçbir pozitif atomik önermesi olmayan bir Horn cümlesine, özellikle mantık programlamada, hedef cümlesi denir.

3 ÇIKARIM KURALLARI

Bir programa ilişkin bir sorguyu yanıtlamak, sorgunun programın mantıksal bir sonucu olup olmadığını belirlemektir. Sorguları yanıtlarken, bir mantık programı dört tümdengelim kuralı kullanabilir:

(22)



Bunlardan ikisi, *genelleme* ve *örnekleme* değişkenlerle ilgilidir. Bu nedenle, incelenmelerini sonraya, birinci dereceden yüklem mantığı tabanlı Prolog çalışmaları yapacağımız derse bırakacağız. Önergeler mantığı tabanlı Prolog'da etkili olan iki tümdengelim kuralı *özdeşlik* ve *modus ponens*'tir. İlki en basit tümdengelim kuralıdır:

(23) ÖZDEŞLİK KURALI:

$\alpha.$

$?- \alpha.$

Şunu söyler: α 'dan α türetilir. İkinci ise, mantığın en bilindik çıkarım kuralı olan *modus ponens*dir: $\alpha \leftarrow \beta$ ve β 'dan α türetilir:

(24) MODUS PONENS:

$\beta :- \alpha.$
$\alpha.$

$?- \beta.$

4 SONUÇ

We have made the following points in this lecture.

Propositions of logic programming, referred to as *statements*, fall only into three categories: *facts*, *rules*, and *queries*. A fact is an atomic proposition. A rule is an implication where the consequent is restricted to be an atomic proposition. A query is an atomic or compound proposition serving as a conclusion to be proved.

Negation is implemented in logic programming with the so-called *negation-as-failure* strategy: derive *not(α)* from failure to derive α .

Each Prolog statement is encoded as a Horn clause. A Horn is a clause where the operands are literals (i.e., atomic propositions or negations of atomic propositions); the operators, if any, are disjunctions; and at most one positive literal is used.

A logic program employs four deduction rules: *identity*, *modus ponens*, *generalization*, and *instantiation*.

Bu derste aşığıdaki tespitleri yaptık.

Mantık programlamada kullanılan önermeleri üç kategoriye ayırılır: *olgular*, *kurallar* ve *sorgular*. Bir olgu atomik bir önermedir. Bir kural, sonucun atomik bir önerme olarak sınırlandırıldığı bir çıkarımdır. Bir sorgu, kanıtlanacak bir sonuç olarak hizmet eden atomik veya bileşik bir önermedir.

Olumsuzlama, mantık programlamada *başarısızlık-olarak-olumsuzlama* stratejisi ile elde edilir: α 'nın elde edilmesindeki başarısızlıktan *değil(α)* türetilir.

Her Prolog cümlesi bir Horn cümlesi olarak kodlanır. Horn, işlenenlerin atomik önermeler veya atomik önermelerin olumsuzlamaları olduğu bir cümledir; işleçler, varsa, ayırma operatörleridir; ve en fazla bir pozitif atomik önerme kullanılır.

Bir mantık programı dört tümdengelim kuralı kullanır: *özdeşlik*, *modus ponens*, *genelleme* ve *örnekleme*.

5 ALIŖTIRMALAR

A. AŖağıdaki ifadelerin her birinin doğru veya yanlış olduğunu belirtiniz:

- 1) Her programlama dili mantıklı değildir.
- 2) Bir öğrenme algoritmasını bir veri kümesiyle eğitmek, 'mantık programlama' olarak adlandırdığımız şeydir.
- 3) Prolog tümevarımlı mantığa dayanır.
- 4) Prolog'da olgular ayrılamaz ('veya' ile bağlanamaz) ancak birleştirilebilir ('ve' ile bağlanmış gibi yorumlanabilir).
- 5) Prolog'da bir kuralın gövdesindeki hedefler birleştirilebilir ancak ayrılamaz.
- 6) Sorgular, bir çıkarımda türetilen sonuçlara karşılık gelir.
- 7) Prolog'da bir sorgudaki hedefler birleştirilebilir veya ayrılabilir.
- 8) Prolog'da Modus Tollens'i bir çıkarım kuralı gibi kullanabilir miyiz?
- 9) Bir Prolog kuralı, başında birleşik bir önerme çiftine sahip olabilir.
- 10) Prolog bir hedefin doğruluğunu veya yanlışlığını kanıtlayamazsa, doğru olduğunu beyan eder.

B. AŖağıdaki önermeleri Prolog programlarına çevirin:

- 1) $(p \rightarrow q)$
- 2) $(q \rightarrow p)$
- 3) $(p \wedge q \wedge r \wedge p_1 \wedge p_2 \wedge p_3)$
- 4) $(p \wedge q \wedge r \wedge ((p_1 \wedge p_2) \rightarrow p_3))$
- 5) $(\neg p \vee q)$
- 6) $(p \vee q)$

C. Prolog'un önerme ifadelerinin (değişkenleri belirtmek için kullanılan) büyük harflerle başlamasına neden izin vermediğini açıklayın.

KAYNAKLAR

KUNEN, K. (1987). “Negation in Logic Programming”. *The Journal of Logic Programming*.

STERLING, L. and SHAPIRO, E. (1986) *The Art of prolog*. Cambridge, Massachusetts: MIT Press.