Mantık Programlama

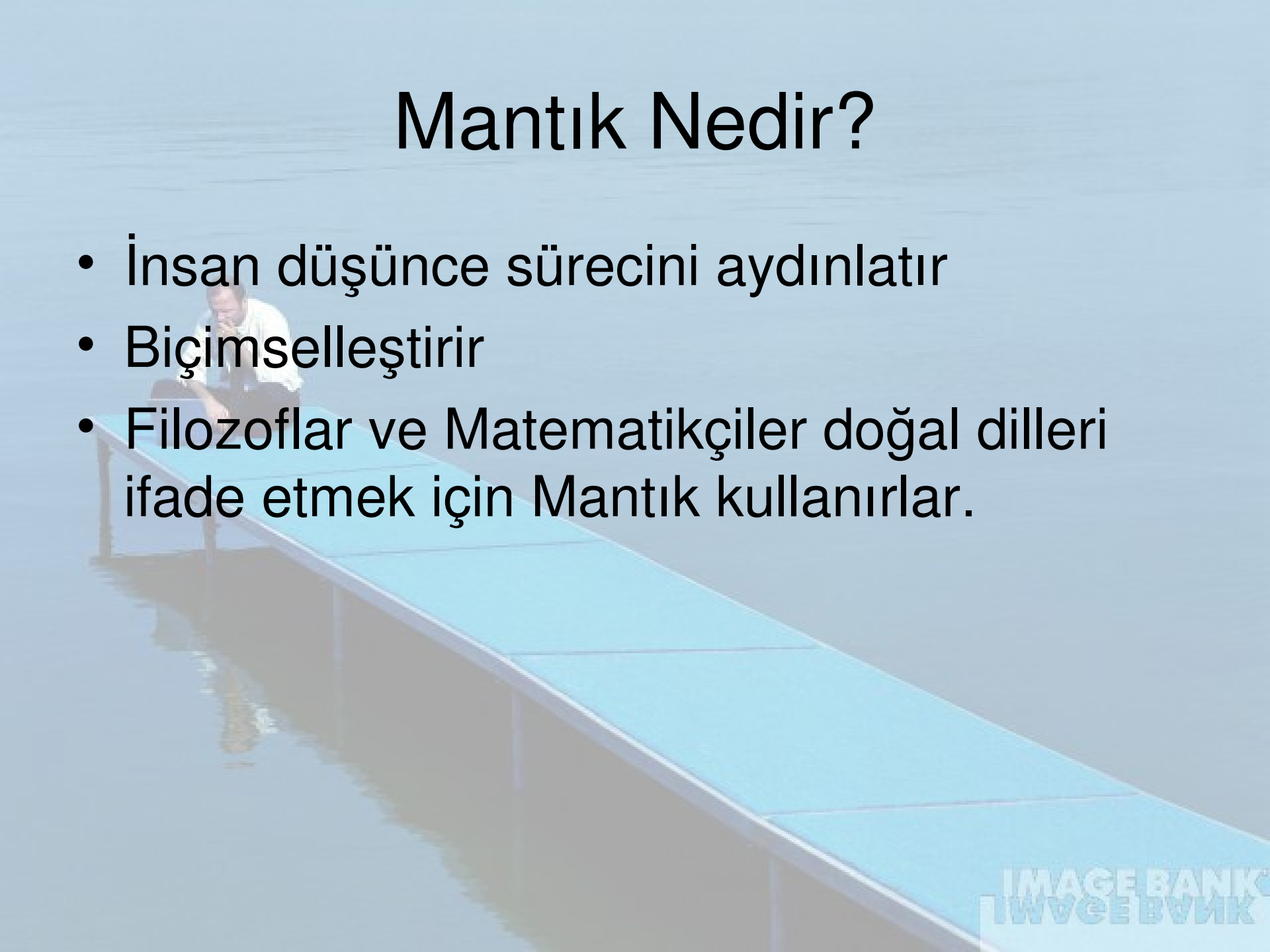
Hazırlayan:Tahir Emre KALAYCI
kalayci@bilmuh.ege.edu.tr

Mantık Programlama

- Mantık Nedir?
- Mantık Programlama
- Prolog
- Sonuçlar

Mantık Nedir?

- İnsan düşünce sürecini aydınlatır
- Biçimselleştirir
- Filozoflar ve Matematikçiler doğal dilleri ifade etmek için Mantık kullanırlar.



Mantık Nedir?

- Gerçek Yaşam:Aşağıdaki gibi tanımlayıcı cümleler vardır

-Nazım resmi çok sever.

-Abidin resim seven herkesin dostudur.

Bu tanıtımlardan herkes:

-Abidin, Nazım'ın dostudur.

Sonucunu üretir.(Klasik Mantık)

Mantık Nedir?

- Sembolik Mantık:

-A1:sever(nazım,resim)

-A2:HER X sever(resim)→dost(abidin,X)

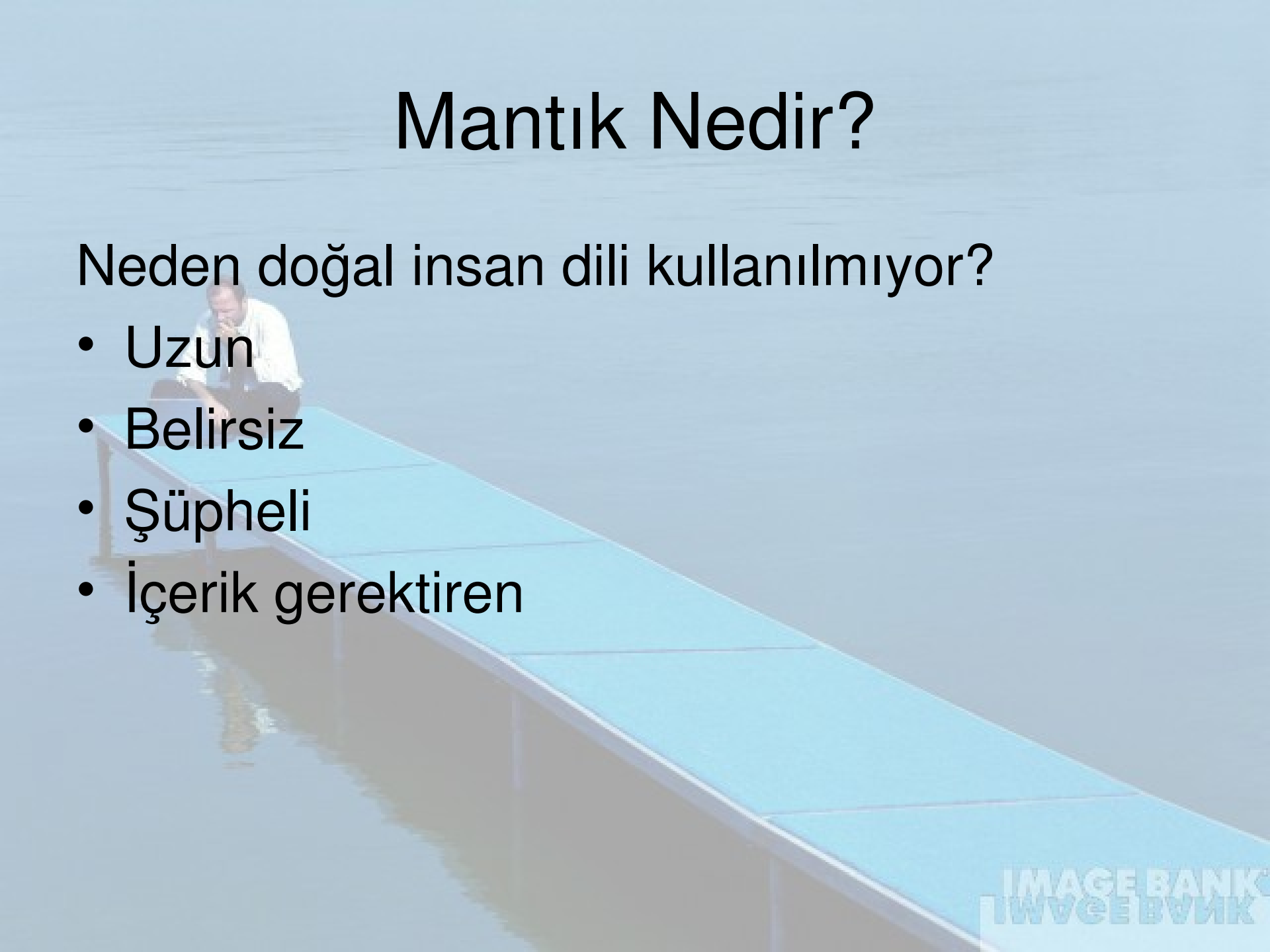
-T1:dost(abidin,nazım)

Bu şekilde insan dilinin sembolik bir ifadesidir.

Mantık Nedir?

Neden doğal insan dili kullanılmıyor?

- Uzun
- Belirsiz
- Şüpheli
- İçerik gerektiren



Mantık Nedir?

Imperative Yaklaşım (Von Neumann)

- Makine özel mimarilerde “bottom-up” olarak çalışır.

```
        Sayi=0;
Dongu:  Kare=Sayi*Sayi;
        Yaz(Kare);
        Sayi=Sayi+1;
        Git(Dongu)
```

Mantık Nedir?

- Yukarıdaki algoritmayı daha yüksek seviyeli programlama paradigması ile yeniden tanımlarız.
- Ama bir soru ortaya çıkacaktır:
  - Mantık problemi temsil etmede mi kullanılacak?
  - Yoksa problemi çözmede mi?

Mantık Nedir?

- Mantık yardımıyla sayıların karesini üreten işlemin teknik ayrıntılarını yazalım:

  Sayilar:0→0,1→s(0),2→s(s(0)),3→s(s(0)),4→ s(s(s(0))),…..

  Dogal Sayilarin Tanimi:dogal(0) VE dogal(s(0)) VE dogal(s(s(0))) VE ….

  Daha İyi Bir Çözüm: dogal(0) VE HER X (dogal(X)→dogal(s(X)))

Mantık Nedir?

- Doğal Sayıların Toplanması:
  HER X (dogal(X))→topla(0,X,X) VE
  HER X HER Y HER Z (topla(X,Y,Z))→
  topla(s(X),Y,s(Z))
- Doğal Sayıların Çarpılması:
  HER X (dogal(X)→ carp(0,X,0)) VE
  HER X HER Y HER Z (carp(X,Y,W) VE
  topla(W,Y,Z) → carp(s(X),Y,Z))

Mantık Nedir?

- Doğal Sayıların Karesi:

  HER X HER Y (dogal(X) VE dogal(Y) VE carp(X,X,Y)→ dogal_kare(X,Y))

En son yazdığımız dogal_kare işlemi en üstte yazdığımız imperatif programın teknik ayrıntısı olarak görülebilir.

Mantık Nedir?

- Bu yaptığımız gösterim için iyidir
- Bu problemi çözmek için efektif tümdengelimi kullanabiliriz:
  - Tümdengelim programını bir kez olmak üzere bilgisayarda programlarız
  - Problem için uygun bir gösterim buluruz
  - Çözümleri elde etmek için sorular sorarız ve geri kalanı tümdengelime bırakırız.

Mantık Nedir?

- Yukarıdaki tanımlamamızı aşağıdaki şekilde işleyebiliriz:

Sorgu: doğal(s(0))?
Yanıt: (evet)
Sorgu: HERHANGİBİR X topla(s(0),s(s(0)),X)?
Yanıt: X=s(s(s(0)))
Sorgu: HERHANGİBİR X topla(s(0),X,s(s(s(0))))?
Yanıt: X=s(s(0))
Sorgu: HERHANGİBİR X doğal(X)?
Yanıt: X=0 VEYA X=s(0) VEYA X=s(s(0)) VEYA ….
Sorgu: HERHANGİBİR X HERHANGİBİR Y topla(X,Y,s(0))?
Yanıt: (X=0 VE Y=s(0)) VEYA (X=s(0) VE Y=0)
Sorgu: HERHANGİBİR X doğal_kare(s(s(0)),X)?
Yanıt: X=s(s(s(s(0))))
Sorgu: HERHANGİBİR X doğal_kare(X,s(s(s(s(0)))))?
Yanıt: X=s(s(0))
Sorgu: HERHANGİBİR X HERHANGİBİR Y doğal_kare(X,Y)?
Yanıt: (X=0 VE Y=0) VEYA (X=s(0) VE Y=s(0)) VEYA (X=s(s(0)) VE Y=s(s(s(s(0))))) VEYA …..

Mantık Nedir?

- Peki problemleri hangi mantık ve hangi yargılama (reasoning) yordamı ile çözeceğiz?
  - Efektif bir yargılama yordamımız var mı?
  - Anlatım gücümüzü yeterli mi?
  - Mantığımızın altında yatan özellikleri ve teorik olarak limitlerimizi biliyor muyuz?

Mantık Nedir?

- Seçim yapacağımız mantıklara örnekler:
  - Önermesel (Propositional) mantık
  - “First Order Predicate Calculus”
  - Yüksek dereceli mantıklar
  - “Modal” mantık
  - λ Calculus

Mantık Nedir?

- Yargılama Yordamları:
  - Doğal Tümdengelim,Klasik yöntemler
  - Kesin Çözüm (Resolution)
  - Prawitz/Bibel
  - “Bottom-up Fixpoint”
  - Yeniden yazma (Rewriting)
  - Daraltma (Narrowing)

Mantık Programlama

- Tarihçe
- Mantık Programlamanın Temelleri
  - Mantık Programlama Nasıl
  - Mantık Sistemlerinin Sözdizimi (Syntax)
  - Mantık Sistemlerinin Semantiği
  - “Predicate” Mantık
  - Resolution Yöntemi
- Mantık Programlamanın Alanları

Mantık Programlama-Tarihçe

- Mantığı programlama dili olarak kullanma fikri uzun zamandır var olan bir fikirdi.
- Erken zamanlarda teoremleri ispat eden programlar geliştirilmişti ama hız ve depolamaya bağlı kısıtlı kapasitede ve daha ciddi arama karmaşıklığından kaynaklanan problemler vardı.
- Alan Robinson'ın 1960'lı yıllarda geliştirdiği Resolution ilkesi mantıksal ispatlara dayanan programlama dillerinin gerçekleştirilmesine büyük katkı sağlamıştır.

Mantık Programlama-Tarihçe

- Robert Kowalski ve arkadaşları ve aynı dönemde Alain Colmerauer ve arkadaşları doğal dil çözümlemesi (parsing) için çalıştırılabilir (executable) gramerler üzerine çalışmaları sonucunda Prolog dili 1972 yılında gerçekleştirildi.

Mantık Programlama-Tarihçe

- Mantıksal Programlamanın Gelişiminde Önemli Adımlar:
  - 1934'te Alfred Tarski "Predicate Calculus" için anlamsal bir teori geliştirdi.
  - 1936'da Gerhard Gentzen "Bir cümle için bir Predicate Calculus ispatı varsa bu ispat konu dışı içeriklere girilmeden cümlenin içeriğinden de türetilir." fikrini ortaya atmış, bulduğu sonuçlar "Predicate Calculus" cümlelerinin ispatlarını bulan algoritma arayışını hızlandırmılştır.
  - 1960'larda "Predicate Logic"in daha basit bir şekli olan "Clausal Logic" ortaya çıkmıştır.
  - 1963'te Alan Robinson "Resolution" adını verdiği çok basit ve etkin bir çıkarım kuralını prensip alan bir "Clausal Logic" icat etmiştir. Resolution "Unification" işlemine dayanmaktadır(Jacques Herbrand).

Mantık Programlama-Tarihçe

- Mantıksal Programlamanın Gelişiminde Önemli Adımlar:
  - 1960'lı yılların sonunda yapay us komiteleri, yapay us amaçları için bilginin tanımlanarak mı yoksa prosedürel olarak mı daha iyi gösterileceğini tartışıyordu.
  - 1972'de Robert Boyer ve Strother Moore tarafından gerçekleştirim stratejilerinin geliştirilmesi için çalışmalar yapılmış ve bu çalışmalar uygulanabilir bir mantıksal programlama dili olan prologu ortaya çıkarmıştır.
  - 1974'te Robert Kowalski bilginin "Horn Clause" gösterimi ile prosedürel gösteriminin denkliğini ispatlamıştır. Bu aşamadan sonra mantık programlama sistemlerinin uygulanması ve geliştirilmesi için teknolojik bir ortam oluşmuştur

Mantık Programlama-Temelleri

- Bir mantık programı axiom kümeleri ve bir hedef cümlesinde oluşur. Hedef cümlesinin doğruluğu için axiomların yeterli olup olmadığının tespiti için çıkarım kuralları vardır.
- Mantık programının çalışması axiomlardan hedef cümlesinin ispatının kurulmasına kadardır.
- Diğer bir ad Kural Tabanlı Diller:Bu isim belirli durumların sağlanması ve sağlandığı zaman buna uygun bir tepkinin verilmesinden dolayı verilmiştir.

Mantık Programlama-Temelleri

- Mantık Programlama Sayesinde:
  - Yargılamalar,
  - Çıkarımlar yapmamız sağlanır
  - Bu iş otomatikleşir.
  - Otomatik dönüşümlere,
  - Paralelizayon,
  - Yarı otomatik debugging
  - Ve Program doğrulama için kapı açılır.
  - Geliştirilmiş programlama üreticiliği vardır.

Mantık Programlama-Temelleri

- Programcı temel mantıksal ilişkileri tanımlamaktan sorumludur ama çıkarım kurallarının uygulanmasının stilinden sorumlu değildir.

  Böylece:

  Mantık+Kontrol=Algoritmalar olmaktadır.
- Mantık Programlama değişken kümelerine dayanmaktadır. Predicateler değişken kümelerinin veri tipinin soyutlanması ve genelleştirilmesidir.

Mantık Programlama-Temelleri

- Bir değişken kümesini

  S0xS1xS2x…..xSn

  şeklinde ifade edebiliriz.
- Örneğin doğal sayıların kare alma fonksiyonu aşağıdaki şekilde değişken kümeleriyle ifade edilebilir:

  {(0,0),(1,1),(2,4),..}
- Bu şekildeki bir kümeye ilişki denir ve aşağıdaki ifadede kare alma ilişkisinin tanımı görülmektedir:

  Doğal_kare={(0,0),(1,1),(2,4),..}

Mantık Programlama-Temelleri

- Her çiftler için genelleştirme ve isim için bir soyutlama yaparak aşağıdaki ifadeyi elde ederiz:

  Doğal_kare=(x,x^2)
- İsmi parametreleştiririz ve şu ifadeyi elde ederiz:

  Doğal_kare(X,Y)<-- Y is } X*X

Mantık Programlama-Temelleri

- Mantık sisteminin sözdizimi, sembollerden üretilen iyi biçimledirilmiş formül (well formed formulas-legal logical expressions)leri tanımlar. Tipik sözdizimi aşağıdaki elemanları içerir:
- Sabitler (Constants)
- Fonksiyonlar (Functions)
- Yüklemler (Predicates)
- Değişkenler (Variables)
- Bağlaçlar (Connectives)
- Tanımlayıcılar (Quantifiers)
- Noktalama işaretleri

Mantık Programlama-Temelleri

- Kullanacağımız dilin anlambilimini (semantic) bilmeden o dilin ifade ettiklerini anlamayız. Bu yüzden mantıkta kullanılan formülleri anlamak için önce anlambilimini öğrenmeliyiz
- W bir formüller kümesi olsun(İçerdiği tüm formüllerin birleşimi) ve A bir formül olsun,

  W | = A

  İfadesi her W modeli için A doğrudur demektedir.

  Bu yüzden aşağıdaki şekilde cümleler ile bunu açıklarız:

  W A yı gerektirir (W entails A)

  W A yı içerir (W implies A)

  A W den sonra gelir (A follows from W)

  A W nin mantıksal sonucudur (A is a logical consequense of W)

  A W nin bir teoremidir (A is a theorem of W)

Mantık Programlama-Temelleri

- Örnek: Bloklar dünyamız:
  üstünde(A,B) :A B nin üstündedir
  yukarısında(A,B): A B nin yukarısındadır
- W kümesindeki formüller: (a,b,c farklı bloklar)
  üstünde(a,b)
  üstünde(a,b)
  HER X,Y için yukarısında(X,Y) ← üstünde(X,Y)
  HER X,Y için yukarısında(X,Z) ← üstünde(X,Y)
  &yukarısında(Y,Z)
- Bu verilere göre
  - yukarısında(a,c) (doğru)
  - yukarısında(c,b) (yanlış)
- Veri olmadan bunu söyleyemezdik. Bu yüzden yargılara varabilmek için mantık verileri gerektirmektedir.

Mantık Programlama-Temelleri

- "Propositional" (Önermesel) Mantık:
  - Önerme ve bağlaçlardan oluşur:
  - Bağlaçlar:
    - Λ (&,VE),V (VEYA), ¬ (DEĞİL),← (→olarakta yazılabilen "implication")
  - Bağlaçların herbiri için bir doğruluk tablosu kullanılarak sonuçlar üretilir:

A	B	A Λ B	A V B	¬A	A ← B
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	1

Mantık Programlama-Temelleri

- Önermesel mantık için, verilen sonlu sayıda eleman içeren bir küme için sonlu sayıda atama olabileceği için bu doğruluk tablosundan her zaman yararlanılır. Çok fazla sayıda önerme içeren ifadeler için bu tablo çok büyük olacaktır.
- Örnek: W=(a← (a V b))

  D= a V b

  W | = D ?

  Aşağıdaki tabloyu kullanarak bu soru için HAYIR cevabını verebiliriz:

a	b	a ← (a V b)	a V b
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	1

Mantık Programlama-Temelleri

- Bağlaçların durumuna göre bir formül ya Doğru yada Yanlış olacaktır.
  - Formülü Doğru durumunda değerlendiren yorumlamaya formülün modeli denilir.
    - Bir formül en az bir model içeriyorsa tatmin edici (*satisfiable*) ,
    - Her yorumlama için doğru ise totoloji
    - Her yorumlama için yanlış ise “contradiction” adını alır.

Mantık Programlama-Temelleri

- W | = a kullanarak iyi biçimlendirilmiş formüller arasındaki ilişkiler hakkında konuşabiliriz. Predicate mantık için doüruluk tablosu yerine çıkarım sistemi kullanırız.
  - Çıkarım sistemi yeni formüller türetmemize yarayan çıkarım kurallarından oluşan bir sistemdir.
  - Örneğin “modus-ponus” yaygın kullanılan bir çıkarım kuralıdır:

    Ne zaman bir x ve x→y e sahipsek y yi türetebiliriz

    x, x → y

    ---

    y

Mantık Programlama-Temelleri

- İspat, iyi biçimlendirilmiş formüllerin bir serisidir:

  W | = D

  W kümesinden D formülünü türetebiliyoruz.

- Türetmeler yalnızca çıkarım kurallarıyla belirlenir.
  - Çıkarım kuralları W den D ye bir ispat üretemiyorsa, W | = D doğru değilse “sound” olurlar.
  - W | = D doğru ise ve W den D ye bir ispat üretebiliyorsa “complete” çıkarım sistemimiz vardır.

Mantık Programlama-Temelleri

- “Implication”($\rightarrow$) ve eşitlik(equivalence) operatörleri yardımıyla yazılan “Predicate Calculus” cümleleri $\wedge$ (conjunction), $\vee$ (disjunction) ve $\neg$ (negation) operatörleri kullanılarakta yapılabilirler.
- Bütün “Predicate Calculus” cümleleri “clausal form” adı verilen özel bir biçimde ifade edilebilir.
- “Clausal form” “Predicate Calculus” cümleleri Prolog cümlelerine benzetilebilir. Bu nedenle “clausal form” Prolog ve mantık arasındaki ilişkiyi anlamak açısından önemlidir.

Mantık Programlama-Temelleri

Bir “Predicate Calculus” cümlesinin normal forma çevrilmesi :

1. Adım: “Implication”ların yokedilmesi : İlk adımla $\rightarrow$ ve $\leftrightarrow$ operatörleri ortadan kaldırılır.

   $a \rightarrow B \qquad \neg a \vee B$

   $a \leftrightarrow B \qquad (a \wedge B) \vee (\neg a \wedge \neg B)$

   $a \leftrightarrow B \qquad (a \rightarrow B) \wedge (B \rightarrow a)$

   Bu durumda $HER(X, adam(X) \rightarrow insan(X))$ cümlesi $HER(X, \neg adam(X) \vee insan(X))$ olarak çevrilebilir.

Mantık Programlama-Temelleri

2. Adım : “Negation” operatörünü içe doğru hareket ettirmek

¬(insan(sezar) ∧ yaşıyor(sezar)) cümlesi

¬(insan(sezar) ∨ ¬ yaşıyor (sezar))

olarak çevrilebilir.

Başka Bir Örnek:

¬HER(Y,şahıs(Y)) cümlesi VAR(Y,¬şahıs(Y)) olarak çevrilebilir.

Kural:

¬(a∧B)	¬a∨¬B
¬VAR(v,P)	HER(v,¬P)
¬HER(v,P)	VAR(v,¬P)

Mantık Programlama-Temelleri

3. Adım : Skolem normal form

   Diğer bir adım varoluş tanımlayıcılarını(existential quantifier) ortadan kaldırmaktır.

   Örnek:

   VAR(X,kadın(X) $\wedge$ annesi(X,havva))

   Skolem Normal Formu:

   kadın(g197) $\wedge$ annesi(g197,havva)

   g197 =yeni bir sabit

   g197 =annesi havva olan bir kadın

   VAR(X, kadın(X) $\wedge$ annesi(X,havva))

   (HAVVA'nın kızı olan insanın varlığını bildirir)

   – Evrensel tanımlayıcı(universal quantifier) varsa:
     HER(X,insan(X)$\rightarrow$VAR(Y,annesi(X,Y)))

     "her insanın annesi vardır" cümlesi

     HER(X,insan(X)$\rightarrow$annesi(X,g2)) (Bütün insanların annedi g2 sabitinin gösterdiği aynı kişi mi?)

   – Yukarıdaki cümlenin doğru çevrimi:

     HER(X,insan(X)$\rightarrow$annesi(X,g2(X)))

     g2 bir fonksiyon

Mantık Programlama-Temelleri

4. Adım : Bütün evrensel tanımlayıcılar cümlenin dışına doğru hareket ettirilir.

   Örnek:

   $HER(X,kadın(X) \wedge HER(Y,çiçek(Y) \rightarrow sever(X,Y)))$

   cümlesi

   $HER(X,HER(Y,kadın(X) \wedge (çiçek(Y) \rightarrow sever(X,Y)))$

   şekline çevrilir.

   En son olarak evrensel tanımlayıcılar cümleden çıkartılabilirler:

   $kadın(X) \wedge çiçek(Y) \rightarrow sever(X,Y)$

Mantık Programlama-Temelleri

5. Adım : $\wedge$ bağlacını $\vee$ bağlaçları üzerine dağıtmak.

   Bu adımda cümle yeni bir forma (“conjunctive normal form”) sokulacaktır ve artık cümlede “conjunction”lar “disjunction”lar içinde yer almayacaklardır.

   Kurallar:

   $(A \wedge B) \vee C$ işlemi $(A \vee C) \wedge (B \vee C)$

   $(A \vee B) \wedge C$ işlemi $(A \wedge C) \vee (B \wedge C)$

Mantık Programlama-Temelleri

6. Adım : “clause” forma sokmak

En son elde edilen formül : $(A \wedge B) \wedge (C \wedge (D \wedge E))$

Parantezler gereksiz:

$(A \wedge B) \wedge (C \wedge (D \wedge E)) = (A \wedge ((B \wedge C) \wedge (D \wedge E)) = (A \wedge B) \wedge ((C \wedge D) \wedge E)$

$= A \wedge B \wedge C \wedge D \wedge E$

$= \{A,B,C,D,E\}$ (Arasında $\wedge$ olan literaller kümesi)

Böylece “Predicate Calculus” ifadesini “clause”lar kümesi haline getirmiş olduk.

Mantık Programlama-Temelleri

Tüm Adımları Gösteren Bir Örnek:

HER(X,HER(Y,şahıs(Y) $\rightarrow$ saygı_duyar(Y,X)) $\rightarrow$kral(X))

Bu cümle “Eğer herkes bir kişiye saygı duyuyorsa bu kişi kraldır” anlamına gelmektedir.

1.Adım sonunda:

HER(X, $\neg$(HER(Y, $\neg$şahıs(Y) $\vee$ saygı_duyar(Y,X))) $\vee$kral(X))

2.Adım sonunda:

HER(X, VAR(Y,şahıs(Y) $\wedge$ $\neg$(saygı_duyar(Y,X)) $\vee$ kral (X))

3.Adım sonunda:

HER(X,şahıs(f1(X)) $\wedge$ $\neg$saygı_duyar(f1(X),X)) $\vee$ kral (X))

Not: f1 bir skolem fonksiyondur.

Mantık Programlama-Temelleri

4.Adım sonunda:

(şahıs(f1(X)) $\wedge\neg$saygı_duyar(f1(X),X)) $\vee$ kral(X)

5.Adım sonunda (conjunctive normal form):

(şahıs(f1(X)) $\vee$ kral(X)) $\wedge$ ($\neg$saygı_duyar(f1(X),X) $\vee$ kral(X))

6.Adım:Burada iki “clause” vardır. Birinci “clause” iki literalden oluşmuştur

şahıs(f1(X)) kral(X)

İkinci “clause”daki literaller ise

$\neg$saygı_duyar(f1(X),X) kral(X)

Mantık Programlama-Temelleri

Prolog dilindeki “,” “$\wedge$” sembolüne, “;” “$\vee$” sembolüne, “:-“ “$\rightarrow$” sembolüne karşılık gelir.

Bu durumda

HER(X,HER(Y,şahıs(Y) $\rightarrow$ saygı_duyar(Y,X)) $\rightarrow$ kral (X)) $\equiv$ (şahıs(f1(X)) $\vee$ kral(X)) $\wedge$ ($\neg$saygı_duyar(f1(X),X)

cümlesi Prolog’da aşağıdaki gibi yazılabilir :

```
sahis(f1(X));kral(X):-.
kral(X):-saygı_duyar(f1(X),X).
```

Mantık Programlama-Temelleri

- Resolution Yöntemi

  1960'lı yıllarda insanları makineleri teoremleri otomatik olarak ispatlayacak şekilde programlamak istediler.
- J.Alan Robinson, "resolution principle" tekniğini ortaya attı.
  - Resolution principle mekanik teorem ispatıdır. Resolution "clausal form"daki cümlelerle çalışır. Birbiriyle ilişkili iki "clause"dan, bu "clause"ların mantıksal sonucu olan yeni bir "clause" yaratır.

Mantık Programlama-Temelleri

- Resolution tekniğindeki temel prensip şudur:

  Aynı literal bir “clause”un sol tarafında ve diğer “clause”un sağ tarafında bulunursa, iki “clause” biraraya getirilip tekrar eden literal atılarak yeni bir “clause” oluşturulur.Örnek olarak;

  üzgün(turgay);kızgın(turgay):-çalışma_günü(bugün),yağmurlu(bugün).

  tatsız(turgay) :- kızgın(turgay), yorgun(turgay).

  Resolution tekniği ile

  üzgün(turgay); tatsız(turgay):-
  çalışma_günü(bugün),yağmurlu(bugün),yorgun(turgay).

  cümlesi çıkartılır.

Mantık Programlama-Temelleri

- Yukarıdaki örnekte iki şey atlanmıştır:
  - "Clause"lar değişken içerdiğinde durum daha da zorlaşır.
  - Tekrar eden literaller tamamıyla aynı olmak zorunda değildir.

- İkinci örnekte bu iki durumun nasıl çözüme kavuşturulduğu anlatılmıştır:

  1. sahis (f1(X));kral(X):-.
  2. kral(Y):-saygi_duyar(f1(Y),Y).
  3. saygi_duyar(Z,arthur):-sahis(Z).

  (2) ve (3) "clause"larından çıkarılan sonuç:

  4. kral(arthur):-sahis(f1(arthur)).

  (1) ve (4) "clause"larından çıkarılan sonuç:

  5. kral(arthur);kral(arthur):-. (Arthur bir kraldır)

- Resolution tekniğinde eşleme(matching) olarak adlandırılan işlem "unification" adını alır.

Mantık Programlama-Temelleri

- Resolution tekniği belirli bir şeyi ispatlamak için nasıl kullanılır? Yukarıdaki örnekte “kral(arthur)” “clause”unu mantıksal bir sonuç olsa da verilen “clause”lardan çıkarmak mümkün değildir.

  Resolution tekniği güçlü bir teknik değil mi?

  – Resolution tekniği eğer “clause” mantıksal bir sonuçsa problemi çözmeyi garanti eder.

  Resolution tekniğinin bu önemli özelliği “refutation complete” olması olarak bilinir.

  Bu teknikle ispatlamaya çalışılan “clause”un terslenmiş hali (“goal”) hipotezlere eklenir ve resolution tekniği uygulanır. Sonuçta “empty clause” oluşursa ispatlanmaya çalışılan teorem doğrudur.

  Bu prensip yukarıdaki örneğe uygulanırsa;

  6.:-kral(arthur). (hedefin değillenmesi (negation of goal))

  5.kral(arthur);kral(arthur):-.

  7.kral(arthur):-.

  (6) ve (7) numaralı “clause”lardan aşağıdaki sonuç elde edilir:

  :- (boş clause)

  Bu sonuç “Arthur bir kraldır” “clause”unun (1),(2) ve (3) numaralı “clause”ların mantıksal sonucu olduğunu göstermiştir.

Mantık Programlama-Alanları

- Mantık programlamanın özellikle kullanıldığı problem alanları aşağıdaki gibidir:
  - Teorem kanıtlama
  - Yorumlarıcı ve derleyiciler
  - Çizge algoritmaları
  - Protokol doğrulama
  - Bilgi temsili
  - Uzman sistemler, karar destek sistemleri
  - Planlama ve programlama (scheduling)
  - Doğal dil işleme
  - Veritabanı modelleme

Mantık Programlama-Alanları

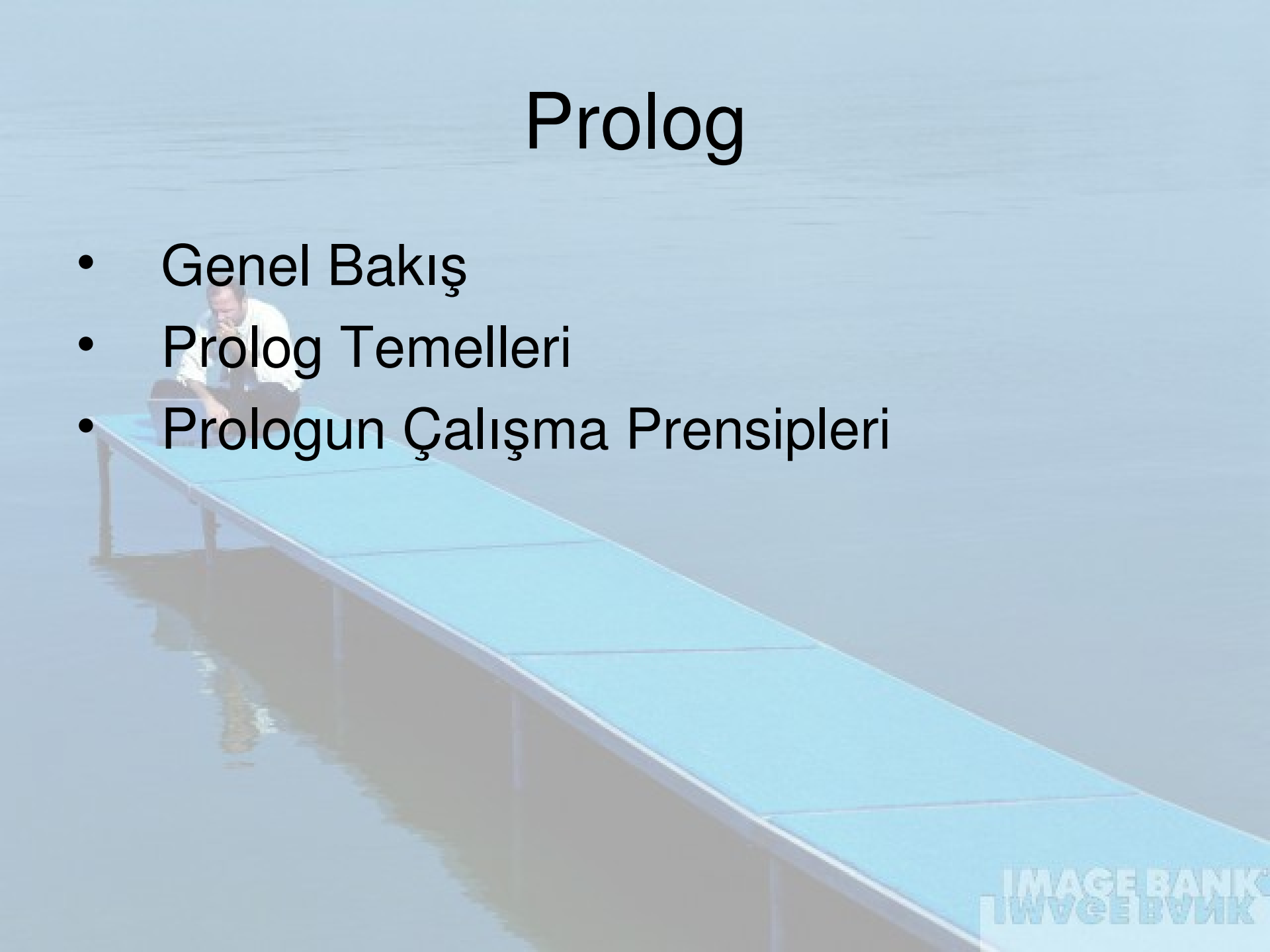
- Yukarıdaki alanlarda aşağıdaki uygulamalar geliştirilmektedir:
  - İlişkisel veritabanları
  - Doğal dil arayüzleri
  - Uzman sistemler
  - Sembolik denklem çözümleri
  - Planlama
  - Prototip üretimi
  - Simülasyon
  - Programlama dil gerçekleştirimleri

Mantık Programlama-Alanları

- Alternatif programlama şemaları ise şu şekildedir:
  - coroutining
  - concurrency
  - Denklemlere dayalı mantık programlama
    - Fonksiyonel yetenekler
  - Sınırlamalı mantık programlama (constraint)
    - Mantıksal aritmetik,kümeler,…
  - Tümdengelimli veritabanları

Prolog

- Genel Bakış
- Prolog Temelleri
- Prologun Çalışma Prensipleri



Prolog-Genel Bakış

- Prolog en geniş şekilde kullanılan mantık programlama dilidir. Bazı özellikleri:
  - Mantıksal değişkenleri kullanır.Bunlar diğer dillerdeki değişkenlere benzemezler. Programcı bunları veri yapıları içerisindeki hesaplama sürdükçe doldurulabilen “delikler” olarak kullanabilir.
  - Birleştirme (Unification) parametreleri geçiren, sonuçları döndüren, veri yapılarını seçen ve yaratan kurulu manipulasyondur.
  - Temel kontrol akış modeli backtracingtir.
  - Program cümleleri ve veri aynı forma sahiptir.
  - Yordamların ilişkisel formu ‘tersinir’(reversible) yordamlar tanımlanabilmesine olanak sağlar.
  - Cümleler durum analizi ve nondeterminizmi ifade etmek için elverişli bir yol sunar
  - Bazı durumlarda mantığın içermediği kontrol özellikleri kullanılabilir.
  - Bir prolog programı kuralları içerdiği gibi olgularıda içeren bir ilişkisel veritabanı olarak gözükebilir.

Prolog-Genel Bakış

- İfadeler bir problem hakkında çözümün nasıl yapılacağı yerine neyin doğru olduğuna dair bilgiler içerir. Prolog sistemi ifadeleri mümkün çözümler uzayı içerisinden bir çözüm arama yolu ile çözüm gerçekleştiren ifadeler içerir. Bütün problemler saf deklaratif tanımlamalar içermediğinden bazen “extralogical” cümleler gerekebilir.

Prolog-Genel Bakış

- Prolog için gramer örneği:

Program ::= Clause... Query | Query
Clause ::= Predicate . | Predicate :- PredicateList .
PredicateList ::= Predicate | PredicateList , Predicate
Predicate ::= Atom | Atom(TermList)
TermList ::= Term | TermList , Term
Term ::= Numeral | Atom | Variable | Structure
Structure ::= Atom (TermList)
Query ::= ?- PredicateList .
Numeral ::= Tam yada reel sayı
Atom ::= küçük harfle başlayan yada tırnak içinde yazılmış karakter katarları
Variable ::= Büyük harfle yada alt çizgi (underscore) ile başlayan karakter katarları
Terminals = {Numeral, Atom, Variable, :-, ?-, virgül, nokta, sol ve sağ parantezler }

Prolog-Genel Bakış

- Bir Prolog programı örneği:

```
elephant(george).
elephant(mary).
panda(chi_chi).
panda(ming_ming).
dangerous(X) :- big_teeth(X).
dangerous(X) :- venomous(X).
guess(X, tiger) :- stripey(X), big_teeth(X), isaCat(X).
guess(X, koala) :- arboreal(X), sleepy(X).
guess(X, zebra) :- stripey(X), isaHorse(X).
```

Prolog-Temelleri

- **Cümleler: Gerçekler ve Kurallar**

  Bir Prolog programcısı nesneleri, bağıntıları (relation) ve bu bağıntıların doğru olduğu kuralları tanımlar.

  Örnek:

  Bill likes dogs.

  (“Bill” ve “dog” nesneleri arasındaki bağıntıyı gösteren cümle)

  Bu cümleye bağıntının ne zaman doğru olduğunu gösteren bir kural ekleyebiliriz:

  Bill likes dogs if the dogs are nice.

Prolog-Temelleri

- **Gerçekler: Bilinenler**
  Prolog'da nesneler arası bağıntılar "predicate" olarak adlandırılır.
  Konuşma dilinde "likes" bağıntısı:
  Bill likes Cindy.
  Cindy likes Bill.
  Bill likes dogs.
  Prolog dilinde yazılmış gerçekler:
  likes(bill, cindy).
  likes(cindy, bill).
  likes(bill, dogs).
- Gerçekler nesnelerin özelliklerini belirtmek amacıyla da kullanılırlar:
  "Kermit is green." green(kermit)
  "Caitlin is a girl" girl(caitlin)

Prolog-Temelleri

- **Kurallar: Verilen Gerçeklerle Hangi Sonuçlara Ulaşılabilir**

  Kurallar verilen gerçeklerden yeni gerçekler türetmeye yararlar.

  “likes” bağıntısı için kurallar:

  Cindy likes everything that Bill likes.

  Caitlin likes everything that is green.

  Verilen kurallar sonucunda ulaştığımız yargılar:

  Cindy likes Cindy.

  Caitlin likes Kermit.

  Aynı kuralların Prolog dilinde gösterimi:

  likes(cindy,Something):-likes(bill,Something).
  likes(caitlin,Something):-green(Something).

Prolog-Temelleri

- **Sorgular (“GOALS”)**

  Verilen gerçekler ve kurallar doğrultusunda sorular sorulup, cevap ararız:

  Does Bill likes Cindy?

  Prolog dilinde bu soru aşağıdaki gibi sorulur:

  likes(bill, cindy).

  Bu sorguya karşılık Prolog “yes” cevabını döndürür.

Prolog-Temelleri

- Bir diğer soru:

  “What does Bill likes?” sorusudur.

  Prolog:

  likes(bill,What).

  Burada dikkat edilmesi gereken nokta birinci nesne (“bill”) küçük harfle başlar. Bu “bill”in sabit bir nesne olduğunu gösterir. “What” ise büyük harfle başlar ve değişken bir nesnedir.

Prolog-Temelleri

- Prolog bir sorguya cevap verebilmek için daima en tepeden başlayarak, taranacak gerçek kalmayana kadar gerçekleri tarar.
- likes(bill,What). Sorgusunun sonucu:

  What=cindy

  What=dogs

  2 solutions

  Sorguyu değiştirip “What does Cindy likes?” şekline getirirsek:

  likes(cindy, What).

  Sonuçlar:

  What=bill

  What=cindy

  What=dogs

  3 solutions

Prolog-Temelleri

- **Gerçekler, Kurallar ve Sorguların Birleştirilmesi**

  Aşağıdaki gibi gerçeklerin ve kuralların verildiği varsayılırsa:

  A fast car is fun.

  A big car is nice.

  A little car is practical.

  Bill likes a car if the car is fun.

- Bu gerçekler incelendiğinde Bill'in hızlı arabalardan hoşlandığı sonucu çıkartılabilir. Prolog'da aynı sonucu bulur. Hızlı arabaların eğlenceli olduğu bilgisi verilmese bile bir insan hızlı bir arabanın eğlenceli olacağı sonucunu tahmin edebilirdi. Fakat Prolog'un tahmin edebilme yeteneği yoktur; sadece verilen gerçekleri bilir.

Prolog-Temelleri

- Aşağıda Prolog'un sorguları cevaplamak için gerçekleri nasıl kullandığını gösteren bir örnek sunulmuştur. Program-I'in aşağıda verilen parçasındaki gerçekler ve kurallar incelenirse :

```
likes(ellen, tennis).
likes(john, football).
likes(tom, baseball).
likes(eric, swimming).
likes(mark, tennis).
likes(bill, Activity):-likes(tom, Activity).
```

- Programın son satırı bir kuraldır:

```
likes(bill, Activity):-likes(tom, Activity).
```

Bu kural konuşma dilinde aşağıdaki cümleye karşılık gelir:

Bill likes an activity if Tom likes that activity.

- Bu örnekte Bill'in "baseball"dan hoşlandığı hakkında bir bilgi yoktur. Prolog'un cevabı bulması için aşağıdaki sorgu verilebilir:

```
likes(bill,baseball).
```

- Bu sorgunun cevabını bulabilmek için Prolog aşağıdaki kuralı kullanır:

```
likes(bill, Activity):-likes(tom, Activity).
```

Prolog-Temelleri

- Program bütün olarak yazılırsa:
  - PREDICATES
    nondeterm likes(symbol, symbol)
  - CLAUSES
    likes(ellen, tennis).
    likes(john, football).
    likes(tom, baseball).
    likes(eric, swimming).
    likes(mark, tennis).
    likes(bill, Activity):-likes(tom, Activity).
  - GOAL
    likes(bill,baseball).
- Sistem diyalog penceresine aşağıdaki sonucu döndürür:
  Yes

Prolog-Temelleri

- Sistem likes(tom,baseball) gerçeği ile likes(bill, Activity):-likes(tom, Activity) kuralını birleştirip likes(bill,baseball) gerçeğine ulaşmıştır:

  Aşağıdaki sorguya cevap bulmak istenirse:

  likes(bill,tennis).

  Sistem “no” cevabını gönderir çünkü program parçasında Bill’in tenisten hoşlandığı gerçeği verilmediği gibi verilen gerçeklerden ve kurallardan da bu sonuç çıkarılamamaktadır.

Prolog-Temelleri

- **Değişkenler: Genel Cümleler**

  Prolog'da değişkenler kullanılarak genel gerçekler ve kurallar yazılabildiği gibi genel sorularda sorulabilmektedir. Konuşma dilinde değişkenler çok sık kullanılır. Örnek olarak aşağıdaki İngilizce cümle ele alınırsa:

  Bill likes the same thing as Kim.

  cümle Prolog'da

  likes(bill,Thing):-likes(kim,Thing).

  şeklinde yazılır. Yukarıda görüldüğü gibi Prolog'da değişkenler büyük harfle veya "_" karakteri ile başlar.

Prolog- Çalışma Prensipleri

- **Unification**

  Prolog bir goal'ü bir subgoal (call) ile uyuşturmaya çalışırken Unification adı verilen bir arama işlemi yapılır. Unification, subgoal'de bulunan veri yapıları ile verilen clause'daki bilgileri uyuşturma çabasıdır. Unification, diğer geleneksel dillerde bulunan parametre iletme, durum seçimi (case selection), yapı yaratma (structure building), yapı erişimi (structure access) ve atama gibi işlemleri implement eder.

Prolog- Çalışma Prensipleri

- DOMAINS
  title, author = symbol
  pages = unsigned
- PREDICATES
  book(title, pages)
  nondeterm written_by(author, title)
  nondeterm long_novel(title)
- CLAUSES
  written_by(fleming, “DR NO”).
  written_by(melville, “MOBY DICK”).
  book(“MOBY DICK”, 250).
  book(“DR NO”, 310).
  long_novel(Title):- written_by(_ , Title),
  book(Title, Length),
  Length > 300.
- Verilen örnek program için goal:
  written_by(X,Y)’dir.

Prolog- Çalışma Prensipleri

- Bu goal'ü sağlayacak veriyi bulmak için Prolog, programa verilmiş tüm "written_by" clause'larını test eder. X ve Y değişkenlerini "written_by" clause'undaki veri yapıları ile eşleştirmek için program baştan sona doğru taranır. Bu goal ile uyuşan bir clause bulunduğunda clause'un değerleri X ve Y değişkenlerine bağlanır, böylece goal ile clause identical (özdeş) hale gelirler. Yani goal ile clause unify edilmiş (birleştirilmiş) olur. Bu, Unification işlemidir.

  Bu programda goal, ilk "written_by" clause'u ile özdeşleşebilir.

  written_by(X , Y).

  written_by(fleming,"DR NO").

  X değişkeni *fleming* verisine, Y değişkeni ise "DR NO" verisine bağlanır. Program çalıştırıldığında sonuç:

  X=fleming, Y=DR NO

  olarak görüntülenir.

Prolog- Çalışma Prensipleri

Prolog tüm çözümleri aradığından programın goal'ü varsa diğer "written_by" clause'ları ile de özdeşleşir:

written_by(melville,"MOBY DICK").

Ve Prolog ikinci çözümü de görüntüler:

X=fleming, Y=DR NO

X=melville, Y=MOBY DICK

2 solutions.

- Eğer programa "written_by(X, "MOBY DICK")" goal'ü verilirse Prolog yine tüm "written_by" clause'ları ile goal'ü uyuşturmaya çalışacaktır. Karşılaşacağı ilk clause'da "MOBY DICK" ve "DR NO" verileri uyuşmadığından program ikinci gerçek (fact) olan

  written_by(melville,"MOBY DICK")

  clause'unu deneyecektir. Bu clause, goal ile özdeşleşeceği için X değişkeni *melville* verisi ile bağlanır.

Prolog- Çalışma Prensipleri

- Örnek program için verilen bir diğer goal

  long_novel(X)'tir.

  Prolog bu goal'ü gerçekleştiren X değişkenine değer atarken clause'lar arasındaki "long_novel(Title)" kuralını (rule) bulur. Title da X gibi bir değişkendir ve ikisi de clause'a bağlanmış durumda değildir. Ama goal ile Clauses bölümünde verilmiş olan

  ```
  long_novel(Title):- written_by( _ , Title),
                      book(Title, Length),
                      Length > 300.
  ```

  kuralı uyuşmakta olduğu için unification yapılır.

Prolog- Çalışma Prensipleri

- Prolog sırasıyla kuralın alt goal'lerini sağlamaya çalışır. Önce kuralın içindeki ilk subgoal olan

  written_by(*, Title) ele alınır. Burada kitabın yazarının kim olduğu bilgisi önemli olmadığından Author değişkeni yerine "_"* sembolü konmuştur. Prolog önce ele aldığı ilk subgoal için bir çözüm arar. Yani yazarın adı önemsenmeksizin "written_by" clause'una uygun Title verileri aranır.

  written_by(_ , Title).

  written_by(fleming,"DR NO").

  İlk veri için Title değişkeninin değeri "DR NO" olur. Daha sonra bir sonraki subgoal olan

  book(Title, Length)

  ele alınır. Bu clause'a uygun verinin aranması sırasında daha önceden Title değişkenine bağlanmış olan "DR NO" bilisi için uygun olan Length bilgisi aranmaktadır. Yani asıl çözümü aranan clause artık

  book("DR NO", Length)

  olmuştur.

Prolog- Çalışma Prensipleri

- Programın ilk bulacağı veri

  book(“MOBY DICK”, 250)

  olacağı için burada bir uyuşma sağlanamayacaktır. Dolayısıyla program bir diğer “book” clause’una bakacaktır. Diğer clause’da “DR NO” verisi uyuştuğu için Length değişkeninin değeri 310 olarak belirlenecek, 310 verisi ile Length değişkeni birbirine bağlanacaktır. Bu noktadan sonra, asıl goal olan “long_novel” kuralının üçüncü goal’ü olan

  Length > 300

  subgoal olur ve Length değişkenine bağlanan 310 verisi için subgoal sağlanmış olur. Çünkü Length değişkenine bağlanmış sayının 300’den büyük olması uzun roman olmak için aranan üçüncü koşulu da sağlamaktadır. Bu durumda “long_novel” kuralının üç koşulunu da sağlayan veri “DR NO” olur ve program çıktısı aşağıdaki gibi elde edilir.

  X = DR NO

Prolog- Çalışma Prensipleri

- **Backtracking**

  Bir problemin çözümünü ararken mantıksal bir yol izlemek gereklidir. Ama bazen izlenen mantıksal yol istenen sonuca ulaşamayabilir. Böyle bir durumla karşılaşıldığında başka alternatif çözüm yolları denemek gerekir. Örneğin bir labirentin içinde çıkış yolu bulmak üzere dolaşılırken izlenebilecek ve kesin çözüme götürecek yöntemlerden biri de yolun her ikiye ya da üçe ayrıldığı noktada hep soldaki yolu seçmektir. Bu şekilde ilerlenirken çıkmaz bir yere gelinirse en son sola dönülen yere kadar geriye dönülerek bu sefer sağdaki yol seçilip yine bir sonraki yol ayrımlarından hep sola dönme yöntemi izlenebilir. Bu şekilde olası tüm yollar denenmiş olur ve sonuca mutlaka ulaşılır.

Prolog- Çalışma Prensipleri

Prolog da bu yöntemin aynısını kullanmaktadır. Bu yönteme “Backtracking”, yani “Geriye dön tekrar dene” yöntemi denmektedir. Prolog bir probleme çözüm ararken, iki olası durum arasında karar vermek zorunda kalabilir. Her seçim yaptığı noktaya bir “Backtracking point”), yani bir belirteç koyarak subgoal’lerden ilkini sağlama işini gerçekleştirmeye çalışır. Eğer seçtiği subgoal çözüme ulaşmazsa Prolog geriye dönecek, belirteç koyduğu noktada diğer bir alternatifi seçerek çözümü aramaya devam edecektir.

Prolog- Çalışma Prensipleri

- Backtracking kavramı örnek bir program üzerinde açıklanabilir.

```
PREDICATES
    nondeterm likes(symbol,symbol)
    nondeterm tastes(symbol,symbol)
    nondeterm food(symbol)
CLAUSES
    likes(bill, X): - food(X), tastes(X,good).
    tastes(pizza,good).
    tastes(spinach,bad).
    food(spinach).
    food(pizza).
GOAL
    likes(bill,What).
```

Bu örnekteki kuraldan Bill'in tadı güzel olan yiyeceklerden hoşlandığı anlaşılmaktadır. Program Bill'in neden hoşlandığını aramak üzere

```
    likes(bill,What)
```

goal'ünü gerçekleştirecektir.

Prolog- Çalışma Prensipleri

- Bu iş için program baştan sona doğru taranmaya başlanır. Bulunan ilk clause'da What değişkeni X değişkeni ile *unify* edilir çünkü aranan ve bulunan clause'lar birbiriyle uyuşmuştur (match).

  likes(bill, What)

  likes(bill, X)

  Unification işleminden sonra Prolog, ele aldığı kuralın goal'ü sağlayıp sağlamadığını araştırır. Bunun için kuralın ilk bölümü olan

  food(X)

  subgoal'ü ele alınır. Bu ilk subgoal'ü sağlamak için Prolog tekrar programın en başına döner. İlk clause'da X değişkeni *spinach* bilgisi ile sağlanır. Elde edilen çözüm dışında bir başka alternatif daha olduğu için Prolog ilk çözümü bulduğu noktaya bir Backtracking point koyar. Bu nokta Prolog'un *spinach* verisi için çözüm gerçekleşmezse geriye döneceği ve diğer alternatifi deneyeceği noktadır.

Prolog- Çalışma Prensipleri

- Bu veri ile Prolog

  tastes(spinach, good)

  subgoal'ünü sağlamaya çalışır. Bu subgoal kuralın ikinci maddesidir. Clause'lar arasında

  tastes(spinach, good)

  durumu sağlanmadığından Prolog Backtracking point koyduğu noktaya

  food(spinach)

  durumuna geri döner. Bu noktada Prolog başka bir değişkeni deneyebilmek için bu noktadan sonra bağladığı tüm değişkenleri bırakır. Programın bu noktada "food(X)" clause'u için deneyebileceği diğer alternatif veri *pizza*'dır. X değişkeni *pizza* verisine bağlanır. Bu noktadan sonra program sağlaması gereken ikinci subgoal olan

  tastes(pizza,good)

  clause'unu ele alır.

Prolog- Çalışma Prensipleri

- Bu clause'a uygun bir durum olup olmadığı programın yine en başına dönülerek program sonuna kadar aranır. Aranan yeni subgoal program içinde bulunduğundan, çözümü aranan goal için başarıya ulaşılmış olur. Kontrol edilen

  likes(bill,What)

  goal'ü için "pizza" çözümü bulunur çünkü What değişkeni "likes" kuralındaki X değişkeni ile unify edilmiş ve X değişkeni de *pizza* verisine bağlanmıştır. Sonuç aşağıdaki gibi rapor edilir:

  What = pizza

  1 solution.

Prolog- Çalışma Prensipleri

- Prolog problemin birden fazla çözümü olduğu durumlarda sadece ilk çözümünü bulmakla kalmaz, tüm olası çözümleri de bulur. Dikkat edilmesi gereken bir nokta Prolog'un gereksiz sonuçlar da bulabileceğidir.
- Backtracking'in Dört Temel İlkesi
  1. Subgoal'ler yukarıdan aşağıya doğru sırayla sağlanmalıdır.
  2. Clause'lar programın içinde bulundukları sıra ile test edilirler.
  3. Bir subgoal bir kuralın sol taraf değeri ile uyuştuğu zaman, kuralın sağ tarafının da doğruluğu sağlanmalıdır. Kuralın sağ tarafı doğruluğu sağlanacak bir alt goal'ler kümesidir.
  4. Bir goal, bulunması gereken tüm gerçekler bulunduğunda sağlanmış olur.

Prolog- Çalışma Prensipleri

- **Çözümler İçin Arama Kontrolü**

  Prolog'un Backtracking kurma mekanizması gereksiz aramalara neden olabilir ve bu durum etkinsizliği arttırabilir. Örneğin verilen problemin tek çözümlerinin bulunması gereken zamanlar olabilir. Diğer durumlarda özel bir amaç tanımlanmış olsa bile ek çözümler için Prolog'u zorlamak gerekli olabilir. Buna benzer durumlarda Backtracking süreci kontrol edilmelidir.

  Prolog, Backtracking mekanizmasını kontrol edecek iki araca izin verecek şekilde geliştirilmiştir.bunlardan ilki *fail* predicate, ikincisi ise *cut*'tır. *Fail* mekanizması programı Backtracking yapmaya zorlarken *Cut* Backtracking'i engellemek amacıyla kullanılır.

Prolog- Çalışma Prensipleri

- **Fail Predicate Kullanımı**

  Prolog fail, yani başarısızlık durumunda backtracking'e başlar. Prolog *fail* adı verilen özel bir predicate yardımıyla programda başarısızlığa ulaşmayı ve programın backtracking yapmasını sağlar. *Fail* etkisi 2=3 gibi doğru olmayan karşılaştırmalar ve mümkün olmayan amaçları gerçekleştirmeyi sağlar.

Prolog- Çalışma Prensipleri

- Bu komutun kullanımıyla ilgili bir örnek program aşağıda verilmiştir.

```
DOMAINS
    Name=symbol
PREDICATES
    nondeterm father(name,name)
    everybody
CLAUSES
    father(leonard,katherine).
    father(carl,jason).
    father(carl,marilyn).
    everybody :- father(X,Y),
                 write(X,"is",Y,"'s father. \n"),
                 fail.
    everybody.
```

Everybody predicate'ı bir kere başarılı bir biçimde sonlandığında Prolog programına backtracking yapmasını söyleyen herhangi bir koşul yoktur.bu yüzden father clause'u sadece bir çözümle programdan dönecektir. Ama *everybody* predicate'ı tüm çözümlerin bulunması için backtracking yapmak üzere *fail* predicate'ını içermektedir. Dolayısıyla program geri dönüp tekrar çözüm aramak suretiyle tüm olası sonuçları bulacaktır. Everybody predicate'ının amacı programın çalışmasından daha net bir cevap elde etmektir.

Prolog- Çalışma Prensipleri

- GOAL
  father(X,Y).
  X=leonard, Y=katherine
  X=carl, Y=jason
  X=carl, Y=marilyn

  ve

  GOAL
  everybody.
  leonard is katherine's father.
  carl is jason's father.
  carl is marilyn's father.

  goal ve çözümleri için aşağıdaki çıkarımlar yapılabilir:

Prolog- Çalışma Prensipleri

- *Everybody* predicate'ı father(X,Y) clause'u için daha fazla sonuç elde etmek üzere backtracking mekanizmasını kullanır. Program *everybody* kuralına doğru backtracking'e zorlanır:

  father(X,Y), write(X,"is",Y,"'s father. \n"), fail.

- *Fail* durumu hiçbir zaman sağlanamadığı için (hep başarısızlık olduğu için) Prolog backtracking'e zorlanır. Program birden fazla sonuç oluşturabilen son subgoal'e kadar backtracking yapar. Böyle bir call (çağırma) non-deterministic olarak nitelendirilebilir. Non-deterministic call, deterministic olan ve sadece bir çözüm üretebilen call ile çelişmektedir.
- Write cümlesi tekrar sağlanamadığı için (yeni bir çözüm önerilmediği için) Prolog bu sefer kuralın ilk subgoal'üne backtracking yapar.
- Bir kuralın içinde geçen *fail*'den sonraya bir subgoal yerleştirilmesi gereksizdir. *Fail* her zaman başarısızlığa uğrayacağı için *fail*'den sonraya yerleştirilmiş bir subgoal'e erişmek mümkün olmayacaktır.

Prolog- Çalışma Prensipleri

- **Cut – Backtracking'i Önlemek**

  Prolog, backtracking'i önlemek için kullanılan ve ünlem işareti (!) olarak yazılan bir *Cut* mekanizmasını içerir.

  Cut programın içerisine bir subgoal'ün bir kuralın içerisine yerleştirildiği gibi yerleştirilir. İşlem sırası *Cut*'a gelince *Cut* çağrısı (call) başarı ile sonuçlanır ve eğer varsa bir sonraki subgoal çağrılır. *Cut* bir kere geçildiğinde bir daha *Cut*'tan önce ele alınmış subgoal'lere ve *Cut*'ın içinde bulunduğu predicate'ı tanımlayan diğer predicate'lere backtracking yapmak da mümkün değildir. *Cut*'ın iki ana kullanımı vardır:

  1. Bazı olasılıkların asla anlamlı çözümlere ulaşmayacağı biliniyorsa alternatif çözüm aramak zaman kaybıdır. Eğer böyle bir durumda *Cut* kullanılıyorsa program daha hızlı çalışacak ve daha az bellek kullanacaktır. Bu durum Green *Cut* (Yeşil *Cut*) olarak adlandırılır.
  2. Program mantığı *Cut*'ı gerektiriyorsa alternatif subgoal'lerin dikkate alınmasını önlemek için kullanılır. Bu da Red *Cut* (Kırmızı *Cut*) olarak adlandırılır.

Prolog- Çalışma Prensipleri

- Cut Nasıl Kullanılır?
  - **Kuralın içindeki bir önceki subgoal'e backtracking'i önleme**
    - r1:- a,b,!,c.

      Bu, Prolog programına a ve b subgoal'leri için bir çözümün yeterli olduğunu söyleme yoludur. Ayrıca r1 kuralını tanımlayan diğer bir clause'a da backtracking yapılamaz.

Prolog- Çalışma Prensipleri

- Cut Nasıl Kullanılır?
  - **Bir sonraki clause'da backtracking'i önleme**
    - ***Cut*, Prolog'a belli bir predicate için doğru clause'un seçildiğini anlatmak için kullanılabilir.**

```
r(1):- !, a, b, c.
r(2):- !, d.
r(3):- !, c.
r(_):- write("This is a catchall clause.").
```

Bu örnek program kodunda *Cut*, r predicate'ını deterministik yapmaktadır. Burada Prolog r'yi tek bir integer argüman ile çağırır. Çağrının r(1) olduğu varsayılırsa:

7. **Prolog yapılan bir çağrıya uyan bir match yakalamak için programı araştırır ve r'yi tanımlayan ilk clause'u bulur. Çağrı için birden fazla olası çözüm olduğundan Prolog bu clause'a bir backtracking point yerleştirir.**
8. **Daha sonra program kuralın maddelerini işlemeye başlar. İlk önce *Cut*'ı geçer ve böylece başka bir r clause'una backtracking yapma olasılığını ortadan kaldırmış olur. Bu durum mevcut backtracking point'leri elimine eder ve çalışma zamanı verimliliği arttırılır. Ayrıca hata-kapanı (error-trapping) olan clause'un "başka hiçbir koşulda r'ye yapılan çağrıya match etmediğinde" çalıştırıldığını garanti eder.**

Prolog- Çalışma Prensipleri

- **Determinizm ve Cut**

  Prolog implementasyonlarında programcılar non-deterministic clause'lara çalışma zamanında yapılan bellek talepleri yüzünden özel bir dikkat göstermelidirler. Prolog non-deterministic clause'lar için içsel kontroller yapmakta ve programcının yükünü azaltmaktadır.

  Hata ayıklama ya da diğer amaçlar için kullanıcının programa müdahale etmesi gerekebilir. Bunun için programcıya derleyici tarafından *check_determ* komutu sağlanmıştır. Eğer check_determ komutu programın çok başında eklenirse, Prolog derleme sırasında bir non-deterministic clause ile karşılaştığında uyarı verecektir.

  Bir predicate'ı tanımlayan kuralın içerisine *Cut* eklemek sureti ile non-deterministic bir clause deterministic yapılabilir.

Prolog- Çalışma Prensipleri

- ***Not*** **yüklemi subgoal'ün doğruluğu kanıtlamadığı zaman başarıya ulaşır. Bu da, bağlanmamış değişkenlerin *Not*'ın içinde bağlanmasını önleyen bir durumla sonuçlanır. Boşta değişkenleri olan bir subgoal *Not* ile çağrılırsa Prolog bir hata mesajı döndürecektir:**

  "Free variables not allowed in 'not' or 'retractall'."

  Bu durum gerçekleşir çünkü Prolog için bir subgoal'de bulunan boştaki bir değişkenin bağlanabilmesi için subgoal başka bir clause ile unify ve subgoal başarıya ulaşmış olmalıdır. Bağlanmamış değişkenlerle *Not* subgoal'ünde başetmenin yolu anonymous (anonim) değişkenler kullanmaktır.

  Not kullanımı çok dikkatli bir biçimde yapılmalıdır.yanlış kullanımı bir hata mesajı alınması veya program mantığında bir hata ile sonuçlanabilir.

Sonuçlar

- **Mantıksal programlama kendisini pekçok kullanım alanında kanıtlamış bir yöntemdir. Bu kullanım alanları arasında diagnostic uzman sistemler, doğal dil işleme, agent tabanlı kontrol sistemleri yer almaktadır. Bunlardan doğal dil işleme mantıksal programlamanın kullanıldığı en zor uygulama alanları arasında gösterilmektedir. Doğal dil işleme mantıksal programlama ile doğal dillerin gramer tabanlı gösterimlerinin birleştirilmesiyle konuşma dilini anlayan araçların geliştirilmesi için çalışan bir uygulama alanıdır.**
- **Mantıksal programlamanın Web'e açılması da bir diğer kullanım alanı olarak yerini almış bulunmaktadır. Mantıksal programlamanın kullanımındaki amaç Web için geliştirilen uygulamalara bilgi tabanlı tekniklerin uygulanabilirliğini sağlamaktır. Böyle bir yaklaşımın avantajları kaynaklar hakkında bilgi seviyesinde yargılama yapma ve bilgi filtreleme tekniklerine mantık tabanlı teknikler ekleyebilme olacaktır.**
- **Prolog gibi mantıksal programlama için kullanılan bir programlama dilinin uygulama alanlarının genişletilmesini sağlayan bir etken de Prolog'un Java implementasyonlarının geliştirilmesi olmuştur. Böylece mantıksal programlamanın Java'nın etkisiyle Web üzerinden uygulamalarının da mümkün olabileceği düşünülmüştür.**
- **Bazı agent uygulamalarında mantıksal programlama dillerinin sağladığı yüksek formalizasyon seviyesinden de faydalanılabilmektedir. Şu anda kullanılan etmenlerin ilk mantık tabanlı modelleri ile etmenlerin diğer programlama dilleri ile gerçekleştirilmeleri arasında bir geçiş mevcut değildir. Bu geçiş deklaratif bir mantık tabanlı programlama dilinin yazılım etmenlerinin gerçekleştirilmesi için kullanılmasıyla gerçekleştirilebilir.**
- **Mantıksal programlamanın bir diğer kullanım alanı constraint programlamadır. Bu, programlama açısından çok esnek bir paradigma olup uygulamayı içeren constraint'i ve hesaplamaları yapan inference engine'i açıklar. Kontrol, scheduling ve bankacılık gibi pekçok kullanım alanı vardır.**