**Dilin tarihçesi**

Prolog ismi Philippe Roussel tarafından seçilmiştir. Fransızcası “programmation en logique” nın kısaltması. Robert Kowalski'nin “Horn clauses” üzerine yaptığı yordamsal yorum üzerine 1972 civarı Alain Colmerauer Philippe Roussel ile birlikte yapmıştır. Mantığı bilgi belirtim yöntemi olarak kullanma isteği üzerine yapılmıştır. Robert Kowalski ye göre prolog dili 1972 de Colmerauer and Phillipe Roussel tarafından geliştirilmiştir. İlk prolog uygulaması tercümansal yazı olmak üzere Fortran by Gerard Battani and Henri Meloni. David H. D. Warren bu yazıyı Edinburghi’a vermiştir ve Edinburghi bu yazıyla “Edinburgh Prolog” olarak adlandırılan çalışmaya başlamıştır. “Edinburgh Prolog” syntax ı modern kullanımda yer alır. Warren aynı zamanda ilk prolog derleyicisini yapmıştır, Fernando Pereira ile işbirliği yaparak etkili DEC-10 Prolog yaratmıştır. Warren daha sonra fikri DEC-10 Prolog ardındaki fikirleri genelleştirip Warren Abstract Machine olarak isimlendirilen yapıyı yapmıştır.

Avrupalı araştırmacılar Yapay zekayı alanında Prolog tercih ederken Amerikalılar Lisp dilini tercih etmiştir. Denilenlere bakılarak alanda dil geçerliliği üzerine bir çok uluslararası rekabete sebebiyet vermiştir. Bir çok modern Prolog gelişimi Fifth Generation Computer Systems (FGCS) projesinden gelmiştir. Bu proje Prolog varyantı olan Kernel dilini geliştirilmiştir. Kernel bu alanda ilk işletim sistemi için kullanılır.

Saf Prolog Horn clauses ile theorem-prover kanıtlayıcı ile kullanılan kısıtlı formu bu şekildedir:

H :- B1, ..., Bn.

Theorem-prover uygulaması bu tür cümlecikleri süreç olarak değerlendirir:

to show/solve H, show/solve B1 and ... and Bn.

Saf(Pure) Prolog yakın zamanda genişledi, ama olumsuzlukları hata olarak değerlendirerek.

Bu olumsuz durumu formu not(Bi) pozitif duruma karşılık(corresponding) Bi durumlarının çözüm için deneme yanılma(trying and failing) olarak gösteriliyor.

Sonraki Prolog uzantıları uygulamaları, uygulamalarda mantıksal dil kısıtlama özellikleri tanıtan orjinal takımdandır.

**Dilin tasarım amaçları(design goals)**

Mantığı bilgi belirtim yöntemi kullanımı isteği üzerine geliştirilir Prolog. Bilgi işlemde mantıksal veri ve bilgi kullanımı ihtiyacı gelişmiştir yöntemler ve uygulamalar geliştikçe.

Bilgi aktarımından sonra uygulamalarda kullanılmak üzere geliştirilen Prolog Avrupa AI araştırma ve geliştirmede yaygınca kullanılmıştır.

**Programlama dilinin hedef kitlesi**

Soru cevaplama bilgisayar sistemi olan Watson’da kullanılmıştır. Watson IBM’nin DeepQA yazılımını ve Apache UIMA(Unstructured Information Management Architecture) yapısını kullanmaktadır. Sistem Java, C++ ve Prolog olmak üzere birçok dil kullanılmıştır ve dağıtılmış işleme sağlayan Apache Hadoop kullanmak için SUSE Linux Enterprise Server 11 işletim sistemi kullanılmıştır. Prolog normal dillerin çözümleme ağaçların üzerinde desen eşleştirme amacıyla kullanılmıştır. Geliştiriciler belirtir: “Biz çözümleme ağaçlarında rahatla desen eşleştirmesi yapan ve bu kuralları verimli bir şekilde uygulayabilen dile gerek duyduk” ve “Anlatılabilirliği ve basitliği sebebiyle Prolog’u kullanılan dil olarak uygun bir seçim olarak gördük.”

Bunun üzerine hedef kitle daha çok mantıksal işlem yapan sistemler kullananlar olmuştur. AI araştırma ve geliştirmede yapay zekaya durumları izah edebilme amacıyla birçok projede Prolog dili kullanılmıştır. Sonuç olarak hedef kitle günlük dil ile bilgisayara bilgi aktarımı yapmayı planlayan her türlü sistem veya proje geliştiricisi dir.

**Programlama dilinin kullanım alanları**

* **ISO Prolog**

Standard ISO Prolog 2 kısımdan oluşur.

ISO/IEC 13211-1 1995 da yayınlandı, Prolog’un temel elementlerinin uygulamalarını standartlaştırması hedeflenmiştir, önden tanımsız ve mobil uygulama hedefli. Cor.1:2007, Cor.2:2012, ve Cor.3:2017 olmak üzere üç düzeltmesi vardır.

ISO/IEC 13211-2 2000 yılında yayınlandı. Modüller için yardım lar standard olarak eklendi. Standard ISO/IEC JTC1/SC22/WG17 çalışma grubu tarafından ilgilenildi. ANSI X3J17 ise Standard için US Teknik Danışman Grubudur.

* **Compilation**

Verimlilik için Prolog soyut makine koduna derlenir. Tür türetme ve derleme süresi ifade etme bazı uygulamalar soyutsal yorumlama kullanır veya yüksek performans için gerçek makine kodu derleme. Mantıksal programlama topluluğu için Prolog kodu için verimli uygulama tasarlama metotları bir aktiş araştırma alanıdır ve değişik uygulama metodları bazı uygulamalarda faydalanılır. “clause binarization” ve “stack-based virtual machines” vb.

* **Tail recursion**

Prolog sistemleri sıklıkla çok bilinen Tail call optimization (TCO) optimizasyon metodu ile uygulanır. Rasgele olmayan tahminler kuyruk özyinelemesi sergiliyor mu yoksa genel çağrımı. Kuyruk çağırımı: Bir clause yığın çerçevesi çağrı olmadan değersizleştirilir. Daha sonra belirleyici kuyruk çapırım belirtimleri sabit yığın alanı ile icra edilir. Diğer dillerdeki looplar gibi.

* **Term indexing**

Birçok Prolog sadece ilk endekslemeye bakar, her terim için endeksleme masraflıdır. Ama fıeld-encoding kelime teknolojisi temel alınarak veya birleştirilmiş kod kelimeleri hızlı endeksleme sağlar tam sorgu he kafada.

* **Hashing**

WIN-PROLOG ve SWI-Prolog gibi bazı Prolog sistemleri büyük veri kümelerini etkili şekilde ele almak için hashing kullanılır. WordNet gibi büyük işlem özelliği olan alanlarda bu çok büyük performans kazanımına yol açar.

* **Tabling**

B-Prolog, XSB, SWI-Prolog, YAP, and Ciao gibi bazı Prolog sistemlerinde method ezberleme metodu olan tabling de kullanılır. Bu kullanıcıyı manuel basit sonuç depolamadan kurtarır.

* **Implementation in hardware(donanım uygulamaları)**

Fifth Generation Computer Systems projes süresinde hızlı işleme amacıyla atanmış mimariyle Prolog donanıma eklendi. Ama hızlı işlem genel amaçla donanıma özelleşmiş mimari ele aldı.

**Desteklediği paradigmalar(imperative, declarative, functional, object oriented etc.)**

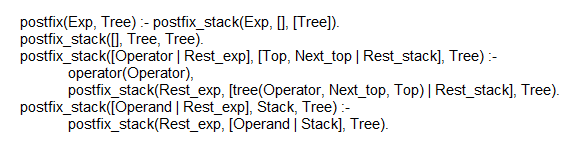
Prolog saf olmayan ifadelere izin verir. Bazı belirli ifadelerde doğru değeri kontrol etmek bazen yan etkilere sebebiyet verir. Ekrana değer yazdırmak gibi. Bu yüzden, Mantıksal paradigmalar uygunsuz olduğunda program “**imperative** programlama” kullanımı için sınırlandırılmıştır.

Saf mantık altyapısı “pure Prolog” olarak adlandırılır ve bazı mantık üzeri içerik özellikleriyle.

**Aritmetik işlem notasyonu (inﬁx/preﬁx/postﬁx)**

* “read/1” gibi built-in mekanizmaları okuma ve denetlemede “:-/2” gibi infix operatörleri daha iyi okur.
  + - 25.5 / 4, 9 - 7
      * | ?- X is 6+3.
      * X = 9
      * Yes
* legal(X) :- \+ illegal(X)
  + “ illegal(X)” kanıtlanmaya çalışılır.
  + Amacın kanıtı bulunursa “\+ illegal(X)” başarısız olur ve orijinal hedef başarılı olur.
  + Amaç kanıtlanamaz ise “?- \+ Goal” başarılı olur. Bu nedenle “\+/1 prefix” kanıtlanamaz ifadedir.
    - -5, \+
      * | ?- X is +(6,3).
      * X = 9
      * Yes
* Fonksiyonel matematikteki “5!” gibi ifadeler Postfix(suffix) operatörlerdir. GNU Prolog’da önceden tanımlı postfix operantlarını yoktur.

Programları “postﬁx(Exp, Tree) :- postﬁx\_stack(Exp, [], [Tree])”... şeklinde postfix ifadeye çevirme mümkündür.



**Bellek yönetimi(programcı denetimli/çöp toplayıcı)**

Çöp toplayıcı ve haricinde haricinde bazı bellek yönetim metodları:

1. garbage\_collect\_atoms
2. garbage\_collect\_clauses
3. set\_prolog\_gc\_thread
4. trim\_stacks

loop :-

generator,

trim\_stacks,

potentially\_expensive\_operation,

stop\_condition, !.

1. set\_prolog\_stack
   1. min\_free(+Cells)
   2. low(+Cells)
   3. factor(+Number)
   4. spare(+Cells)

* Çöp Toplayıcı

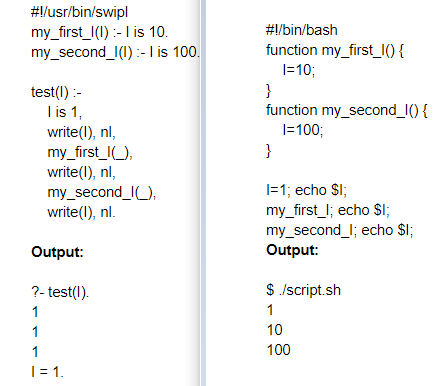
Global ve iz süren çöp toplayıcıları çağırır. Normalde gerek görüldüğü halde otomatik çöp toplayıcısı çağrılır. Kodun zaman kritik segmentlerinde açık çöp toplayıcısı çağrısı faydalı olabilir.

**Değişken kapsamları(lexical scope(static)/dynamic scope)**

* Değişkenler

Değişkenlerde kelimelere bağlı ifadeler: string ile ifade, numaralar, altı çizili karakterler ve başlangıç büyük harf veya başlangıç altı çizili. Keyfi terimler için alan tutucu olarak değişkenler normal mantık daki değişkenleri anımsatır.

Prolog tanımlı değişkenleri takip ettiği için **dynamic scope** kullanır. Dinamik scope kullanır ve döngülerdeki değişimler değişkenleri etkiler.



**Tip sistemi (güçlü(strong)/zayıf(weak)) ve tip kontrolü (static type checking/dynamic type checking)**

Prolog tipsiz bir sistemdir. 1980’lerde ve 2008 de tip yöntemleri tanıştırılmaya çalışıldı, günümüzde hala Prolog’u tip sistemiyle geliştirmeye çalışıyorlar. Tip bilgisi sadece tip güvenliği değil Prolog programlama gerekçelendirme içinde kullanışlıdır.

* Veri tipleri
  + Genel kullanım için atom. ‘x’, ‘red’, 'Taco' ve 'some atom' gibi.
  + Numaralar
  + Değişkenler
  + "functor" atomu için bir "compound term"

**Örnekler:**

**Helloworld!**

program :- write('Hello, world!').

:- program.

**1 - Asal sayı hesaplama**

divisible(X,Y) :- 0 is X mod Y, !.

divisible(X,Y) :- X > Y+1, divisible(X, Y+1).

isPrime(2) :- true,!.

isPrime(X) :- X < 2,!,false.

isPrime(X) :- not(divisible(X, 2)).

program :-

isPrime(11) -> write('true') ; write('false').

:- program.

**Output:**

true

**2 - Fibonacci sayısı hesaplama**

fib(0, 1) :- !.

fib(1, 1) :- !.

fib(N, Result) :- N1 is N - 1, N2 is N - 2, fib(N1, Result1), fib(N2, Result2), Result is Result1 + Result2.

program :-

%2+3=5

fib(4, 5) -> write('true') ; write('false').

:- program.

**Output:**

true

**3 - Mükemmel sayı hesaplama**

factors(\_,N,N,\_).

factors(X,N,NT,Sum):-

N < NT,

( 0 is X mod N ->

Sum1 = Sum + N

; Sum1 = Sum),

N1 is N + 1,

factors(X,N1,NT,Sum1).

perfect(X):-

XT is X/2 + 1,

factors(X,1,XT,0).

program :-

perfect(28) -> write('true') ; write('false').

:- program.

**Output:**

true

**4 - Sihirli kareler**

solve3x3([A1, A2, A3, B1, B2, B3, C1, C2, C3]):-

/\* 1-9 arasında doğru permütasyon kontrolü\*/

permutation([1,2,3,4,5,6,7,8,9], [A1, A2, A3, B1, B2, B3, C1, C2, C3]),

/\* Satır, sütun ve diyagramların toplamlarının 15 olduğunun kontrolü\*/

A1 + A2 + A3 =:= 15, /\*Checks the sum of Row 1\*/

B1 + B2 + B3 =:= 15, /\*Checks the sum of Row 2\*/

C1 + C2 + C3 =:= 15, /\*Checks the sum of Row 3\*/

A1 + B1 + C1 =:= 15, /\*Checks the sum of Col 1\*/

A2 + B2 + C2 =:= 15, /\*Checks the sum of Col 2\*/

A1 + B2 + C3 =:= 15, /\*Checks the sum of Col 3\*/

A1 + B2 + C3 =:= 15, /\*Checks the sum of Diag 1\*/

A3 + B2 + C1 =:= 15. /\*Checks the sum of Diag 2\*/

program :-

solve3x3([8, 3, 4, 1, 5, 9, 6, 7, 2]) -> write('true') ; write('false').

:- program.

**Output:**

True

[KAYNAK - 1](https://rextester.com/l/prolog_online_compiler)

[KAYNAK - 2](http://www.swi-prolog.org/)

[KAYNAK - 3](http://wikizero.biz/)