Politechnika Wrocławska

Podstawy języka PROLOG

Dariusz Banasiak
Katedra Informatyki Technicznej
Wydział Elektroniki

PROLOG – akronim od **PRO**gramming in **LOG**ic

Prolog został stworzony w 1971 przez Alaina Colmeraurera i Philipa Roussela. Podstawy teoretyczne sformułował Robert Kowalski.

Prolog jest nazywany językiem programowania logicznego. Podstawą dla jego utworzenia była logika predykatów pierwszego rzędu oraz zasada rezolucji.

Podstawowe implementacje:

- Turbo Prolog (firma Borland)
- SWI-Prolog (Jan Wielemaker)
- Visual Prolog (Prolog Development Center)

Język Prolog zalicza się do języków deklaratywnych – program opisuje jedynie zadanie, które ma zostać rozwiązane (bez podania algorytmu). Program w języku Prolog składa się z następujących elementów:

- faktów (prawdziwe stwierdzenia)
- reguł (związki między faktami)
- celu (określa zadanie do wykonania).

Pisząc program w Prologu określamy **CO** należy rozwiązać, zamiast **JAK** należy problem rozwiązać.

Program napisany w języku Prologu różni się zasadniczo od programów w językach proceduralnych (Pascal, C, itd.), gdzie należy podać szczegółowy algorytm rozwiązania problemu.

Podstawowe zastosowania języka Prolog

- reprezentacja wiedzy
- przeszukiwanie przestrzeni stanów
- rozwiązywanie problemów logicznych
- dowodzenie twierdzeń
- przetwarzanie języka naturalnego
- realizacja systemów ekspertowych
- realizacja relacyjnych baz danych
- symboliczne rozwiązywanie równań
- różniczkowanie i całkowanie symboliczne

Reprezentacja faktów

W Prologu fakty (bezwarunkowo prawdziwe twierdzenia) przedstawia się za pomocą predykatów pierwszego rzędu, których składnia wygląda następująco:

nazwa_predykatu(ob_1, ob_2, ... , ob_n).

Powyższy predykat oznacza relację określoną przez nazwa_predykatu zachodzącą pomiędzy obiektami:

ob_1, ob_2, ... , ob_n, które są argumentami predykatu.

Predykat: lubi(marta, mandarynki).

przedstawia fakt, że Marta lubi mandarynki (obiektami są „marta” i „mandarynki”, zaś relacją „lubi”).

Przy tworzeniu predykatów obowiązują następujące reguły:

- nazwa relacji (nazwa_pedykatu) jest pisana małymi literami,
- argumenty predykatu znajdują się w nawiasach i są oddzielone przecinkami,
- liczba argumentów predykatu, ich rodzaj i kolejność jest ściśle określona,
- argumentami predykatu mogą być stałe lub zmienne,
- stałe pisane są małymi literami, a zmienne zaczynają się dużą literą (stałe można umieścić również w cudzysłowie i wówczas mogą zawierać dowolne znaki),
- predykat jest zakończony kropką.

Uwaga:

Dwa następujące predykaty:

```
lubi(piotr, marta).
lubi(marta, piotr).
```

reprezentują zupełnie różne fakty.

Reprezentacja reguł

Reguły są warunkowymi stwierdzeniami o istnieniu pewnych zależności między obiektami (faktami).

Język Prolog akceptuje reguły o następującej składni:

IF A_1 and A_2 and ... and A_n THEN B

W notacji języka Prolog powyższą regułę można zapisać w jednej z dwóch postaci:

B if A_1 and A_2 and ... and A_n .

B :- A_1 , A_2 , ... , A_n .

gdzie A_1, A_2, ... , A_n, B są predykatami.

Tworząc regułę należy pamiętać o następujących zasadach:

- nagłówek (konkluzja) reguły składa się tylko z jednego predykatu,
- część warunkową reguły stanowi koniunkcja (iloczyn) dowolnej liczby warunków,
- warunki oddzielone są słowem „and” (lub przecinkiem),
- reguła zakończona jest kropką.

Jako argumenty predykatów mogą wystąpić zmienne. Przedstawiają one nazwy obiektów, które nie są w danej chwili znane. W Prologu nazwy zmiennych zaczynają się od dużej litery lub znaku ”_”.

Regułę, że Marta lubi mężczyzn, którzy są przystojni i jeżdżą porsche można zapisać następująco:

```
lubi(marta,X) :- mezczyzna(X), przystojny(X), jeździ(X,porsche).
```

W regule tej X oznacza zmienną (nieznanego w danej chwili mężczyznę), który spełnia podane warunki.

Regułę, że osoba S jest siostrą osoby X możemy zapisać następująco:

```
siostra(S,X) :- kobieta(S), rodzice(Matka,Ojciec,S),
                rodzice(Matka,Ojciec,X).
```

Reguła ta przedstawia prawidłowość: S jest siostrą X, jeżeli S jest kobietą oraz X i S mają tych samych rodziców.

Reguły i fakty w Prologu nazywa się klauzulami.

Klauzule zawarte w programie tworzą tzw. bazę danych programu.

W bazie może znajdować się większa liczba predykatów o takiej samej nazwie. Wówczas taki zbiór predykatów będziemy nazywać procedurą.

Cel

Po wprowadzeniu do bazy faktów i reguł można zadawać pytania (stawiać cele do realizacji). Celem może być:

- pytanie o prawdziwość faktu,
- polecenia znalezienia nazw obiektów będących w relacji z innymi obiektami.

Cel można określić przez podanie pojedynczego predykatu lub koniunkcji predykatów (tzw. cel złożony). W drugim przypadku predykaty składowe oddzielone są przecinkami (nazywamy je wówczas podcelami).

Jeżeli w predykacie (predykatach) celu występują tylko symbole (stałe), to wówczas jest to pytanie o prawdziwość faktów. Odpowiedzią jest wówczas PRAWDA lub FAŁSZ.

Jeżeli w predykatach występują zmienne, to celem jest znalezienie wszystkich obiektów, które znajdują się w określonej relacji z innymi obiektami.

Inaczej mówiąc mogą one zastąpić zmienną w predykacie celu nie prowadząc do sprzeczności z faktami zawartymi w bazie danych programu.

Przykład

Baza wiedzy zawiera następujące fakty:

```
lubi(jan, wino).
lubi(jan, tenis).
lubi(marta, jabłka).
lubi(marta, wino).
```

Zadane pytania i uzyskane odpowiedzi:

lubi(marta, wino)	odpowiedź: YES
lubi(marta, tenis)	odpowiedź: NO
lubi(X, wino)	odpowiedź: X=jan, X=marta
lubi(jan, X), lubi(marta, X)	odpowiedź: X=wino
lubi(X, wino), lubi(jan, X)	odpowiedź: NO

Mechanizm wnioskujący w Prologu

Podstawą wnioskowania w Prologu jest uzgadnianie (dopasowanie) klauzul zawartych w bazie wiedzy z danym celem (lub podcelami).

Klauzule próbuje się uzgadniać wtedy, gdy ich predykaty są takie same. Procesowi porównywania poddawane są argumenty predykatów.

Ponieważ argumentem predykatu może być symbol (stała) lub zmienna mogą zachodzić dwa następujące przypadki:

- na tej samej pozycji występuje w jednym predykacie stała, a w drugim zmienna (wówczas zmienna przyjmuje wartość symbolu) – operacja ukonkretniania,

- na tej samej pozycji występują dwie zmienne (wówczas następuje powiązanie zmiennych tzn. jeżeli któraś z nich w pewnym momencie zostanie ukonkretniona, to druga zmienna automatycznie przyjmie tą samą wartość) – operacja powiązania.

Operacje ukonkretniania i powiązania są bardzo istotne w procesie wnioskowania realizowanym przez mechanizm wnioskowania języka Prolog.

Przykłady uzgadniania klauzul:

Fakt	**Pytanie**	**Uzgadnianie**	**Rodzaj**
lubi(jan, wino)	lubi(X, wino)	X = jan	ukonkretnienie
lubi(marta, tenis)	lubi(marta, Y)	Y = tenis	ukonkretnienie
lubi(piotr, maria)	lubi(V, Z)	V = piotr Z = maria	ukonkretnienie
lubi(jan, Cos)	lubi(jan, X)	Cos = X	powiązanie

Przykład

Baza wiedzy zawiera następujące fakty (tekst zawarty między symbolami /* ... */ oznacza komentarz):

```
matka(alicja, maria).     /* k1 - „alicja” jest matką „maria” */
matka(ewa, piotr).        /* k2 */
matka(teresa, maria).     /* k3 */
matka(olga, jan).         /* k4 */
```

Mamy podany cel:

```
matka(X, maria).
```

Cel ten oznacza zadanie znalezienia tych matek, których córki mają na imię Maria.

Proces wnioskowania polega na kolejnym uzgadnianiu celu z kolejnymi klauzulami znajdującymi się w bazie wiedzy.

W danym przypadku proces ten przebiega następująco:

- w wyniku dopasowywania celu z klauzulą 1 zmienna X zostanie ukonkretniona stałą „alicja” (cel został osiągnięty i następuje wyświetlenie wartości zmiennej X = alicja). Następuje też ustawianie tzw. znacznika powrotu na klauzulę 1,
- program próbuje dopasować cel z pierwszą klauzulą znajdującą za znacznikiem - jest to klauzula nr 2 (klauzuli tej nie da się uzgodnić z celem),

- w wyniku uzgodnienia klauzuli 3 z celem otrzymamy nową wartość zmiennej X (zostanie ona ukonkretniona stałą „teresa”) - program wyświetli, a znacznik pozycji zostanie przesunięty na klauzulę nr 3,
- następną klauzulą jest klauzula 4 (ponieważ nie można uzgodnić celu i jest to ostatnia klauzula w bazie wiedzy przeszukiwanie zostaje zakończone).

Zostały więc znalezione dwa rozwiązania:

X = alicja

X = teresa

Kolejny przykład ilustruje rolę znaczników w procesie wnioskowania.

Przykład

Baza wiedzy zawiera następujące fakty i reguły:

```
/* 1 */  ojciec(karol,edward).          /* ojciec(o,d) - o jest ojcem d */
/* 2 */  ojciec(karol,august).
/* 3 */  dziadek(X,Y) :- ojciec(X,Z),dziecko(Y,Z).
                                        /* dziadek(d,w) - d jest dziadkiem w */
/* 4 */  dziadek(karol,maurycy).
/* 5 */  dziecko(henryk,edward).        /* dziecko(d,r) - d jest dzieckiem r */
/* 6 */  dziecko(teofil,august).
```

Celem niech będzie znalezienie wszystkich wnuków Karola, co zapisujemy w sposób formalny:

goal dziadek(karol,W).

Proces wnioskowania w języku Prolog polega na próbie uzgodnienia celu po kolei z klauzulami występującymi w bazie wiedzy:

➢uzgodnienie celu głównego z klauzulą 3

dziadek(karol,W). $\overset{u}{\longleftrightarrow}$ dziadek(X,Y) :- ojciec(X,Z), dziecko(Y,Z).

X = karol (ukonkretnienie), W = Y (powiązanie)

podcel 1: ojciec(karol,Z)

zn_1 = klauzula 3 (znacznik nawracania)

- uzgodnienie podcelu 1 z klauzulą 1

  ojciec(karol,Z) $\overset{u}{\longleftrightarrow}$ ojciec(karol,edward).

  Z = edward (ukonkretnienie)

  podcel 2: dziecko(W,edward) - drugi podcel klauzuli 3

  zn_2 = klauzula 1 (znacznik nawracania)

- uzgodnienie podcelu 2 z klauzulą 5

  dziecko(W,edward) $\overset{u}{\longleftrightarrow}$ dziecko(henryk,edward).

  W = henryk (ukonkretnienie)

  rozwiązanie: W = henryk – udowodniono oba podcele klauzuli 3

  zn_3 = klauzula 5 (znacznik nawracania)

➢szukanie kolejnych klauzul dla podcelu 2

Podcel 2 próbujemy uzgodnić z kolejnymi klauzulami w bazie (od miejsca wskazywanego przez znacznik zn_3). Znajdujemy klauzulę 6:

dziecko(W,edward) $\overset{u}{\longleftrightarrow}$ dziecko(teofil,august).

Ponieważ próba uzgodnienia kończy się niepowodzeniem i klauzula 6 jest ostatnią klauzulą w bazie należy: usunąć znacznik zn_3 i wrócić do znacznika zn_2 (podcel 1: ojciec(karol,Z), zn_2 = klauzula 1)

➢uzgodnienie podcelu 1 z klauzulą 2

ojciec(karol,Z) $\overset{u}{\longleftrightarrow}$ ojciec(karol,august).

Z = august (ukonkretnienie)

podcel 2: dziecko(W,august) - drugi podcel klauzuli 3

zn_2 = klauzula 2 (znacznik nawracania)

- uzgodnienie podcelu 2 z klauzulą 6

  dziecko(W,august) $\overset{u}{\longleftrightarrow}$ dziecko(teofil,august).

  W = teofil (ukonkretnienie)

  rozwiązanie: W = teofil – udowodniono oba podcele klauzuli 3

  zn_3 = klauzula 6 (znacznik nawracania)

- szukanie kolejnych klauzul dla podcelu 2

  Podcel 2 próbujemy uzgodnić z kolejnymi klauzulami w bazie. Ponieważ klauzula 6 jest ostatnią klauzulą w bazie należy: usunąć znacznik zn_3 i wrócić do znacznika zn_2 (podcel 1: ojciec(karol,Z), zn_2 = klauzula 2)

- szukanie kolejnych klauzul dla podcelu 1

  Podcel 1 próbujemy uzgodnić z kolejnymi klauzulami (od znacznika zn_2). Ponieważ próba uzgodnienia kończy się niepowodzeniem należy: usunąć znacznik zn_2 i wrócić do znacznika zn_1 (cel główny: dziadek(karol,W), zn_1 = klauzula 3).

- uzgodnienie celu głównego z klauzulą 4

  dziadek(karol,W). $\overset{u}{\longleftrightarrow}$ dziadek(karol,maurycy).

  W = maurycy (ukonkretnienie)

  rozwiązanie: W = maurycy

  zn_1 = klauzula 4 (znacznik nawracania)

- szukanie kolejnych klauzul dla celu głównego

  Cel główny próbujemy uzgodnić z kolejnymi klauzulami (od znacznika zn_1). Ponieważ próba uzgodnienia kończy się niepowodzeniem należy usunąć znacznik zn_1. Ponieważ nie ma już żadnych znaczników powrotu proces przeszukiwania zostaje zakończony.

W wyniku procesu wnioskowania znaleziono następujące rozwiązania:

W = henryk

W = teofil

W = maurycy

Modyfikatory procesu wnioskowania

Operator odcięcia („cut”) jest to bezargumentowy predykat, który zawsze jest prawdziwy i służy do ograniczania nawrotów. Występuje przeważnie jako jeden z podcelów w części warunkowej reguły i uniemożliwia nawrót do któregokolwiek z poprzedzających go podcelów przy próbie znajdowania rozwiązań alternatywnych:

```
dziadek(X,Y) :- ojciec(X,Z), dziecko(Y,Z), ! .  /* znajduje tylko pierwsze
                                                   rozwiązanie W = henryk */

dziadek(X,Y) :- !, ojciec(X,Z), dziecko(Y,Z).   /* zapobiega nawrotom do
                                                   kolejnych predykatów tego
                                                   samego typu – W = henryk,
                                                   W = teofil */
```

Operator odcięcia wykorzystywany jest również przy definicjach rekurencyjnych.

Przykład

Procedura obliczająca silnię dowolnej liczby:

```
silnia(0,1) :- !.
silnia(X,Y) :- N is X-1,      /* operator podstawienia N = X-1 */
               silnia(N,B),
               Y is X*B.
```

Drugim operatorem modyfikującym proces wnioskowania jest predykat „fail”. Predykat ten jest zawsze fałszywy. Najczęściej jest używany do wymuszenia nawrotów.

Listy

Lista jest podstawową strukturą danych w Prologu. Jest ona uporządkowanym zbiorem dowolnej liczby elementów, którymi mogą być: stałe, zmienne lub struktury (np. inne listy). Przykłady list:

[ ] - lista pusta

[1,2,5,4,7]

[[1,2],[5,4],7] - lista zawiera inne listy

Listę można podzielić na dwa elementy: głowę (head) oraz ogon (tail). Głową jest pierwszy element listy, a ogonem pozostała część listy.

Wybrane operacje na listach

[X | Y] – podział listy na głowę (X) i ogon (Y)

member(X,L) – sprawdzenie czy element X należy do listy L
np. member(2,[1,2,3,4,5,6,])

append(L1,L2,L3) – utworzenie listy L3 przez dołączenie listy L2 na koniec listy L1 np. append([1,2],[5,6],X)

Operacje arytmetyczne w Prologu

Prolog jest językiem służącym przede wszystkim do wykonywania operacji na symbolach.

Posiada on jednak również możliwości wykonywania operacji arytmetycznych na liczbach całkowitych. Służą do tego wbudowane operatory arytmetyczne:

* (mnożenie), / (dzielenie) , - (odejmowanie), + (dodawanie) i mod (reszta z dzielenia).

Dostępne są również operatory relacyjne:

= (równy), \ (różny), < (mniejszy), > (większy), <= (mniejszy lub równy) i >= (większy lub równy).

Przykład

W bazie danych mamy następujące fakty:

```
urodzony(jan,1974).
urodzony(ludwik,1979).
urodzony(andrzej,1980).
urodzony(marcin,1970).
```

Aby porównać wiek osób (kto jest starszy od kogo) można zdefiniować następujący predykat:

```
starszy(X,Y):- urodzony(X,R1), urodzony(Y,R2), R1>R2.
```

Operacje wejścia i wyjścia

Operacje wejścia i wyjścia umożliwiają komunikację z użytkownikiem podczas wykonywania programu.

Wbudowane predykaty wejścia-wyjścia:

read(X) – odczytuje term z bieżącego strumienia wejściowego
write(X) – zapisuje term X do bieżącego strumienia wyjściowego
nl – przejście do nowego wiersza

Przykład:

```
pytanie1(Dziecko) :- read(X), rodzic(X, Dziecko).
```