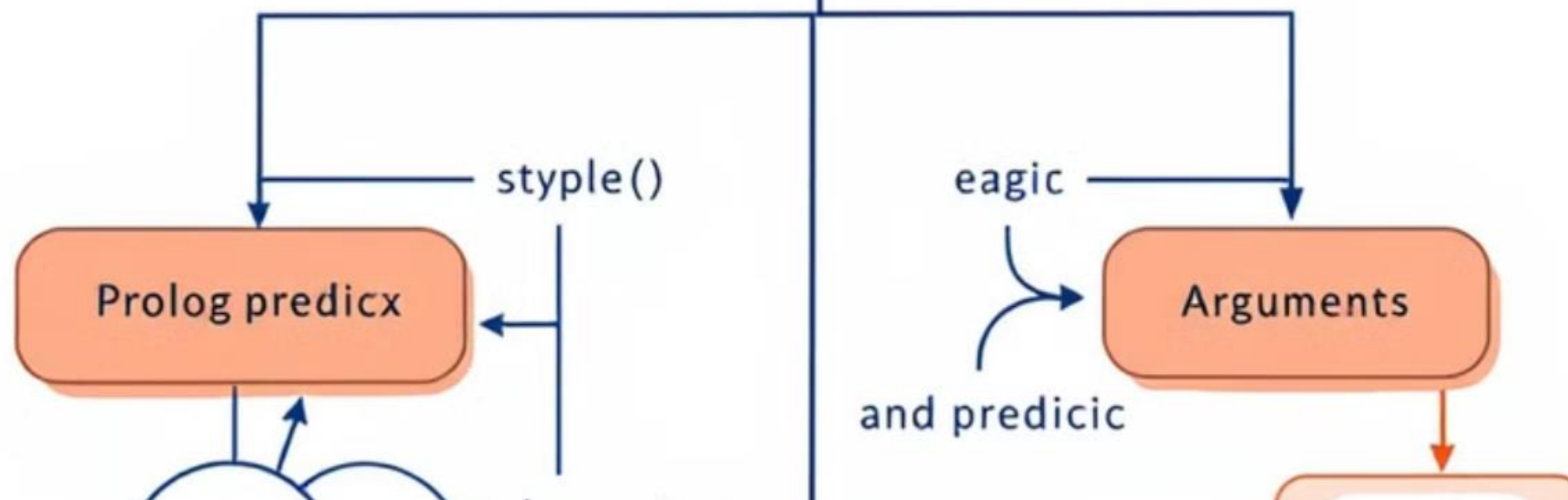


Вградени предикати в Пролог

Вградените предикати са основните функционални блокове в езика Пролог. Те осигуряват базовите операции за логическо програмиране.

В тази презентация ще разгледаме най-често използваните предикати и тяхното приложение в програмирането с Пролог.





Структура на вградените предикати

1 Синтаксис

Предикатите следват схемата
име(аргумент1, аргумент2, ...).
Името винаги започва с малка
буква.

2 Аргументи

Аргументите могат да бъдат
променливи, константи или
сложни термове.
Променливите започват с
главна буква.

3 Връщани стойности

Предикатите не връщат, а
успяват или се провалят.
Променливите се обвързват
при успех.

Предикати за вход/изход

Извеждане

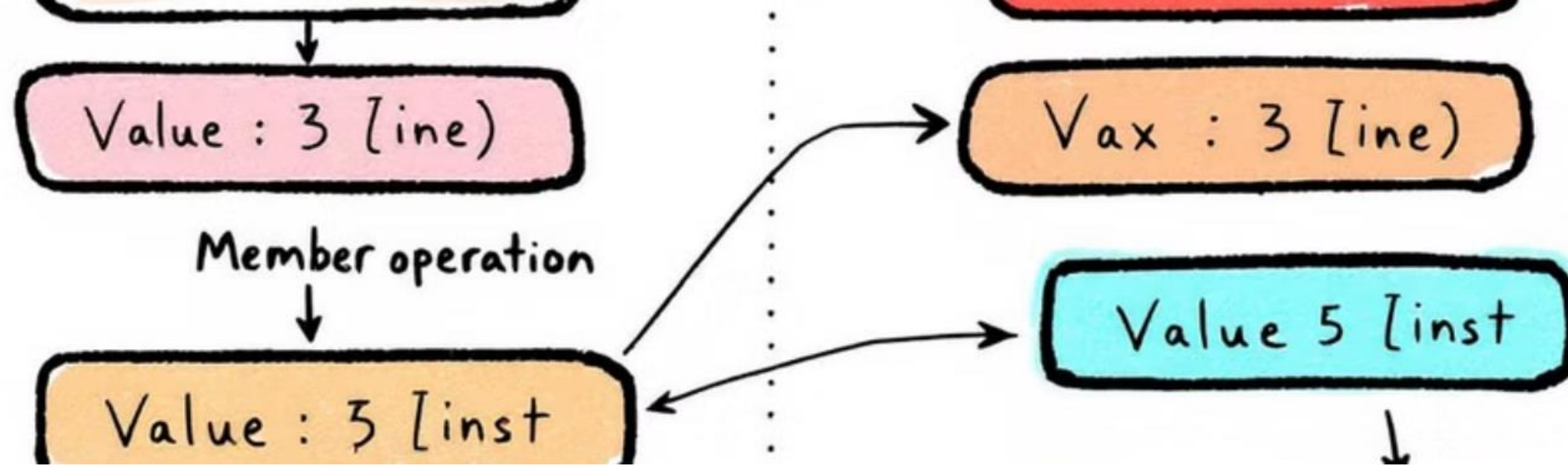
`write/1` извежда термове. `writeln/1` добавя нов ред след извеждането.

Пример: **`write('Здравей, свят!')`**

Въвеждане

`read/1` въвежда терм. `get/1` въвежда единичен символ.

Пример: **`read(X)`**



Предикати за работа със списъци

member/2

Проверява дали елемент принадлежи на списък. Може да генерира всички елементи на списък.

Пример: **member**(X, [1,2,3])

append/3

Съединява два списъка. Може да разделя списък на две части.

Пример: **append**([1,2], [3,4], L)

length/2

Определя дължината на списък. Може да генерира списък с определена дължина.

Пример: **length**([a,b,c,d], N)

Аритметични предикати



is/2

Изчислява аритметичен израз и обвързва резултата. Дясната страна трябва да е изчислима.

Пример: **X is 2+3*5**



Сравнения

:=/2 проверява за равенство, **>/2** за по-голямо от,

Пример: **5 := 2+3**



random/1

Генерира случайно число. Полезно за симулации и игри.

Пример: **random(X)**

```
prolog :, for subtraction ()
  iren } = {
    addition an+ subtraction ()
    addition subtraction,
  }
}
```

Предикати за управление

1

!/o (cut)

Отстранява точки на избор. Предотвратява връщане назад. Оптимизира изпълнението.

Пример: **max(A,B,A) :- A>=B, !**

2

fail/o

Винаги се проваля. Предизвиква връщане назад. Полезен с странични ефекти.

Пример: **list_all :- item(X), write(X), fail**

3

repeat/o

Винаги успява. Създава безкрайни точки на избор. Използва се за цикли.

Пример: **loop :- repeat, action, stop_condition, !**



Предикати за манипулация на термове

1

functor/3

Извлича или конструира функтор и арност на терм. Разглобява сложни термове.

Пример: **functor(date(15,8,2023), F, A)**

2

arg/3

Достъпва аргумент в сложен терм по индекс. Индексите започват от 1.

Пример: **arg(2, point(3,4,5), Y)**

3

=../2 (univ)

Преобразува терм в списък от функтор и аргументи. Мощен за динамично създаване на термове.

Пример: **T =.. [sum, 1, 2, 3]**



Предикати за манипулация на списъци

1

sort/2

Сортира и премахва дубликати от списък.

2

reverse/2

Обръща реда на елементите в списък.

3

select/3

Избира елемент от списък, връща остатъка.

4

nth0/3, nth1/3

Достъпва елемент по индекс, започвайки от 0 или 1.

Предикатите за списъци са фундаментални в Пролог. Списъците са основната структура за съхранение на данни.

Голяма част от задачите в логическото програмиране се свеждат до работа със списъци.

Съществуват три “стандартни” предиката за генериране на всички решения

- `findall/3` събира всички екземпляри в реда, в който те са намерени, но не обработва свободни променливи вдясно
- `setof/3` връща множество от екземпляри (списък в стандартния ред без повторения) и обработва свободни променливи вдясно
- `bagof/3` хибрид между `findall/3` и `setof/3`

```
likes(иван, вино).
likes(стоян, бира).
likes(гого, бира).
likes(део, вино).
likes(тони, бира).
likes(тони, вино).
```

```
% ?- setof(X, likes(X, Y), S).
% ?- setof((Y,S), setof(X,likes(X, Y), S),SS).
```

 `setof(X, likes(X,Y), S).`

```
S = [гого, стоян, тони],
Y = бира
S = [део, иван, тони],
Y = вино
```

?- `setof(X, likes(X,Y), S).`

 `setof((Y,S), setof(X, likes(X,Y), S), SS).`

```
SS = [(бира,[гого, стоян, тони]), (вино,[део, иван, тони])]
```

?- `setof((Y,S), setof(X, likes(X,Y), S), SS).`


```
% ?- findall(X, likes(X, Y), S).
```

 `findall(X,likes(X,Y),S).`

```
S = [иван, стоян, гого, део, тони, тони]
```

?- `findall(X,likes(X,Y),S).`

```
% ?- bagof(X, likes(X, Y), S).
```

 `bagof(X,likes(X,Y),S).`


```
S = [стоян, гого, тони],
Y = бира
S = [иван, део, тони],
Y = вино
```

?- `bagof(X,likes(X,Y),S).`

Променливите в Goal могат да се считат за свободни, освен ако те изрично не са свързани с Goal посредством квантора за съществуване:

$$Y^{\exists} Q$$

означава че “съществува Y такава че Q е *true*”, където Y е някаква променлива в Пролог.

 `setof(X, Y^(likes(X,Y)), S).`

S = [гого, део, иван, стоян, тони]

?- `setof(X, Y^(likes(X,Y)), S).`

 `bagof(X, Y^likes(X,Y), S).`

S = [иван, стоян, гого, део, тони, тони]

?- `bagof(X, Y^likes(X,Y), S).`

 `findall(X, likes(X,Y), S).`

S = [иван, стоян, гого, део, тони, тони]

?- `findall(X, likes(X,Y), S).`

Стандартни предикати

`findall (Template, Enumerator, List)`

Намира всички екземпляри на *Template*, за които *Enumerator* е удовлетворим, и ги натрупва в списък по реда, в който те са намерени (т.е. първият намерен екземпляр се поставя като първи елемент на списъка, а последният - като последен елемент на списъка) и свързва *List* с този списък. Обикновено *List* е променлива, а *Template* е променлива или терм, чийто аргументи са променливи, но могат да бъдат и произволни

Стандартни предикати

`bagof/3` и `setof/3` имат същия формат като `findall/3`:

```
bagof(Template, Enumerator, InstanceList)  
setof(Template, Enumerator, InstanceSet)
```

`bagof/3` прави разлика между променливи в `Enumerator`, които са несвързани и не се съдържат в `Template`. Това означава, че той може да изброява/пресмята свързванията за тези променливи

Пример 2

```
calendar (tom, algebra, monday) .  
calendar (tom, cooking, tuesday) .  
calendar (tom, english, wednesday) .  
calendar (sue, algebra, tuesday) .  
calendar (sue, history, wednesday) .  
calendar (sue, biology, thursday) .
```

```
bagof (Subject, calendar (Person, Subject, _),  
      Subjects)
```

***Subject* - споменава се в Template**

***Person* - искаме да свържем**

***'_'* - искаме да игнорираме**

Пример 2а

?- bagof(Subject, Day^calendar(Person, Subject, Day),
Subjects).

 bagof(Subject,calendar(Person,Subject,_),Subjects)

Person = sue,
Subjects = [biology]


Person = sue,
Subjects = [algebra]

Person = sue,
Subjects = [history]

Person = tom,
Subjects = [algebra]

Person = tom,
Subjects = [cooking]

Person = tom,
Subjects = [english]

 bagof(Subject,Day^calendar(Person,Subject,Day),Subjects).

Person = sue,
Subjects = [algebra, history, biology]
Person = tom,
Subjects = [algebra, cooking, english]

?- bagof(Subject,Day^calendar(Person,Subject,Day),Subjects).

?- bagof(Subject,calendar(Person,Subject,_),Subjects)

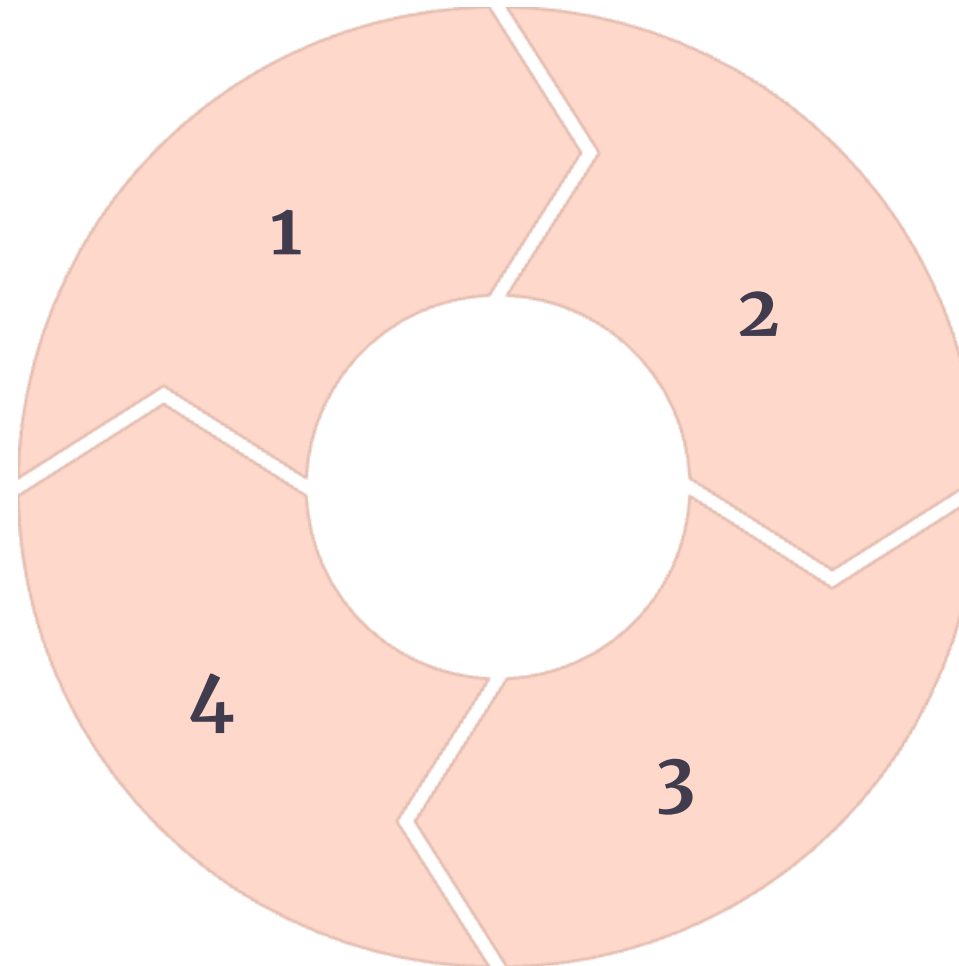
Предикати за мета-програмиране

call/1

Изпълнява предикат динамично.
Основен за мета-интерпретатори.

findall/3

Събира всички решения в списък.
Ключов за обработка на множество
резултати.



assert/1

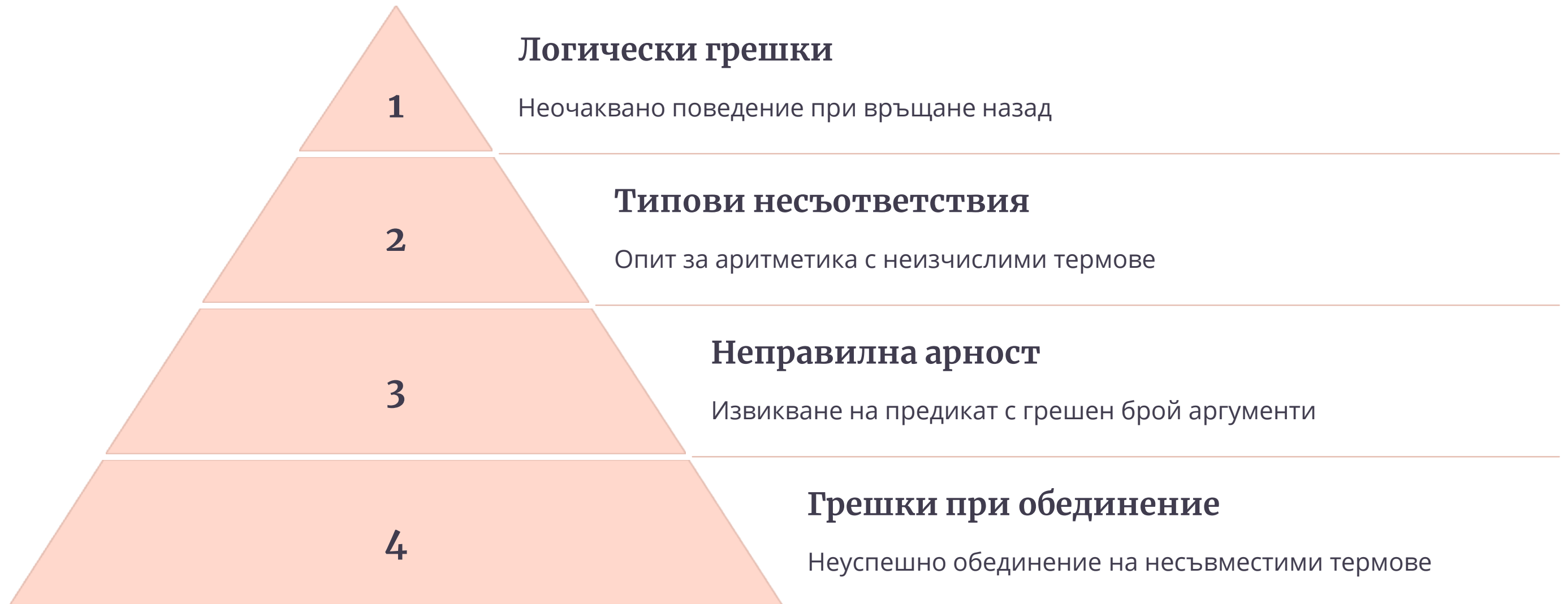
Добавя клауза към програмата.
Позволява динамично генериране
на код.

retract/1

Премахва клауза от програмата.
Използва се с assert за
модификация на програми.

Тези предикати правят Пролог мощен метаезик, способен да манипулира собствения си код.

Често срещани грешки при използване



Разбирането на грешките е ключово за ефективно програмиране в Пролог. Грешките често имат много различна природа от тези в императивните езици.



Заклучение

300+

Вградени предикати

Пролог предлага богата библиотека от вградени функционалности.

5

Основни категории

Разделени по функционалност за по-лесно изучаване.

40+

Години развитие

Пролог продължава да еволюира като мощен логически език.

Вградените предикати са фундаментът на Пролог програмирането. Те превръщат теоретичната логика в практически инструмент за решаване на проблеми.

За допълнително изучаване: SWI-Prolog документация, "The Art of Prolog" и "Programming in Prolog".

Задача 1. Трите студентки

Задача. Весела, Мима и Пепа учат химия, биология и математика в градовете София, Пловдив и Варна. Знае се, че Весела не учи в София, а Мима не е в Пловдив. Тази, която е в София, не учи математика. Студентката в Пловдив изучава химия. Мима не харесва биологията, но вижда всеки ден залеза в морето. Определете всяка от тях какво учи и в кой град ?

Задача 2. Олимпиада в Камчия

На олимпиада по ИТ в комплекс Камчия участват пет момчета- от София, Пловдив, Плевен, Велико Търново и Русе. Те са Иван, Тодор, Александър, Николай и Виктор. На кръглата маса в края на състезанието софиянецът седи между русенеца и Виктор, пловдивчаненът - между Иван и Тодор, а срещу него стои Александър . Момчето от Велико Търново стои до Александър. Николай никога не е бил в Пловдив, а Иван никога не е бил в София и Русе. Освен това русенецът и софиянецът редовно си кореспондират с Александър. Определете в кой град живее всяко от момчетата.

Задача 3. Четири къщи

Има 4 къщи на една и съща улица. Всеки от тях е дом на един от 4-ма души: Симо, Николай, Ангел и Радо. Всеки от тях има професия: лекар, художник, ловец, треньор. Определете кой в каква къща живее и кой каква професия притежава.

Известно е, че: Художникът живее до треньора. Лекарят живее до Художника.

Ловецът е вляво от лекаря. Треньорът не е до ловеца. Художникът е отдясно на Симо.

Радо не е треньор. Симо е до Николай. Ангел не е до Радо.

Задача 4. Дефинирайте предикат, определящ N-я елемент в даден списък при известно N.

```
nmem(1,[X|_], X):-!.
```

```
nmem(N, [_|T], X):- N>1, N1 is N-1, nmem(N1,T,X).
```

Задача 5. Дефинирайте предикат, чрез който може да се вмъкне даден терм на N-тото място в даден СПИСЪК.

`ins(1, X, T, [X|T]).`

`ins(N, X, [H|T], [H|L]) :- N > 1, N1 is N-1, ins(N1, X, T, L).`

`?- ins(3, g, [1,2,4,3,5,6,3], Y).`

Задача 6. Дефинирайте предикат, който проверява дали един списък се получава от друг чрез замяна на всеки елемент на първия списък, равен на терм A, с терма B

`subst(_,_,[], []).`

`subst(A,B, [A|T], [B|L]) :- !, subst(A,B,T,L).`

`subst(A,B, [X|T], [X|L]) :- subst(A,B,T,L).`

`?-subst(3, g, [1,2,4,3,5,6,3], Y)`

Задача 7. Дефинирайте предикат, който от даден числов списък генерира два списъка: единият, съдържащ елементите на даден списък, които са по-малки от дадено число X , а другият, съдържащ всички останали елементи на първия списък.

`split(_, [], [], []).`

`split(X, [H|T], [H|T1], T2):- H<X, !, split(X,T,T1,T2).`

`split(X, [H|T], T1, [H|T2]) :- split(X,T,T1,T2).`

?- `split(3,[1,2,4,3,5,6,3],B, Y)`

Регистрация 19.05.2025

Регистрация

`https://tinyurl.com/
yosr8yau`



ПРОЕКТИ