

# **Изкуствен интелект / интелигентни системи**

*гл. асистент д-р Венета Табакова-Комсалова  
ФМИ, ПУ „П. Хилендарски“, Пловдив*

# Упражнение 3

- ❖ Механизъм за възврат
- ❖ Отрицанието като неуспех
- ❖ Аритметични изрази и оператори

# Механизъм за възврат

**Механизмът за връщане назад (backtracking)**, с цел търсене на нови решения, е начинът, по който Пролог търси решение в програмата. Този процес дава възможност да търси решение през всички известни факти и правила за решение.

Съществуват няколко основни принципа на този механизъм:

- Подцелите се удовлетворяват в реда на посочването им.
- Клаузите на предикатите се проверяват отново по реда на срещането си в програмата.
- Когато подцелта съвпадне с глава на правило, тогава започва удовлетворяването на тялото. Тялото на правилото определя нова последователност от подцели за удовлетворяване.
- Целта се удовлетворява като цяло, когато се намерят всички факти за всички проверени подцели в дървото на целите.
- Обръщение към предикат, към който са свързани няколко решения е неопределено (**non-deterministic**), докато обръщение което се свързва само с едно решение е определено (**deterministic**).

Пролог предлага три средства за контролиране на последователността за намиране на логическия извод при търсене на решение:

- ∅ Предикатът **fail** винаги не успява. По този начин стартира незабавно търсене на ново решение с механизма на възврат.
- ∅ Предикатът **not** успява, когато свързаната с него подцел е неистина.
- ∅ Прекъсването посредством **cut (!)** предотвратява търсенето на нови решения.

## Пример:

харесва(иван,Х):- храна(Х), вкусове(Х,добър).  
вкусове(пица,добър).  
вкусове(брюкселско\_зеле,лош).  
храна(брюкселско\_зеле).  
храна(пица).

Тази малка програма има две двойки факти и едно правило. Правилото описва връзката **харесва**, която има смисъл, че **иван** харесва вкусната храна.

За да се поясни как работи механизмът на възврат при намиране на решение се подава целта:

**?- харесва(иван, Какво).**

***Когато Пролог започва да удовлетворява целта, търсенето на решения започва от началото на програмата.***

Когато се намира съвпадение с първата клауза в програмата, променливата ***Какво*** се унифицира с променливата ***X***. Съвпадението на целта с главата на правилото определя началото на удовлетворяването на правилото. Удовлетворяването на цялото правило се разбива на удовлетворяване на всички подцели в това правило. И първото удовлетворяване е на ***храна(X)***.

**Когато трябва да се извърши ново удовлетворение, търсенето на съвпадение отново започва от началото на програмата.**

Заради това променливата ***X*** се свързва с първия намерен факт ***брюкселско\_зеле***. Тъй като има повече от една възможност за отговор на ***храна(X)***, Пролог създава точка на възврат до факта ***храна(брюкселско\_зеле)***. Тази точка ще укаже мястото от където Пролог ще започне ново търсене на решение за ***храна(X)***.

**Когато се намери успешно съвпадение, започва да се търси удовлетворение на следващата подцел.**

С променливата ***X*** се обвързва ***брюкселско\_зеле*** и следващата цел е ***вкусове(брюкселско\_зеле, добър)***.



Пролог започва да търси удовлетворяване на тази подцел отново от началото на програмата. Тъй като няма клауза, която да съвпада с поставената цел Пролог задейства механизма на възврат. По този начин търсенето на съвпадения се връща в точката на възврат.

***Когато една променлива се обвърже в клауза, единственият начин да се „освободи” е чрез механизма за възврат.***

Връщането в точката на възврат освобождава всички променливи обвързани след тази точка и започва ново търсене с нова заявка. Променливата  $X$  вече е свободна и при търсене на следващ факт за ***храна( $X$ )*** се привързва със стойността ***пица***.

Пролог преминава към следващата подцел в правилото спрямо новото обвързване на променливата. Следващата стъпка е в ход **вкусове(пица, добър)** и отново от началото на програмата започва търсене на съвпадение. Този път съвпадение се открива и целта се удовлетворява успешно. Тъй като променливата **Какво** в целта се унифицира с променливата **X** в правилото **харесва**, а променливата **X** получи стойност **пица** Пролог връща следния резултат:

Какво=пица

1 Solution

# Предотвратяване на механизма на възврат (Прекъсване - Cut)

## Оператор „!“

Една възможност за ограничаване на пространството се предоставя посредством използването на стандартния предикат „!“ (cut, отрязване). Отрязването е специален стандартен предикат, който инструктира интерпретатора на Пролог да не прави възврат зад точката на програмата, в която той се появява. Операторът няма явна декларативна семантика (винаги успява), така че неговото използване става за сметка на яснотата на програмата. Операторът трябва да се използва внимателно, понеже неговото присъствие може да наруши изпълнението на една програма по непредвидим начин. Ще разгледаме типични случаи за използване на оператора „!“.

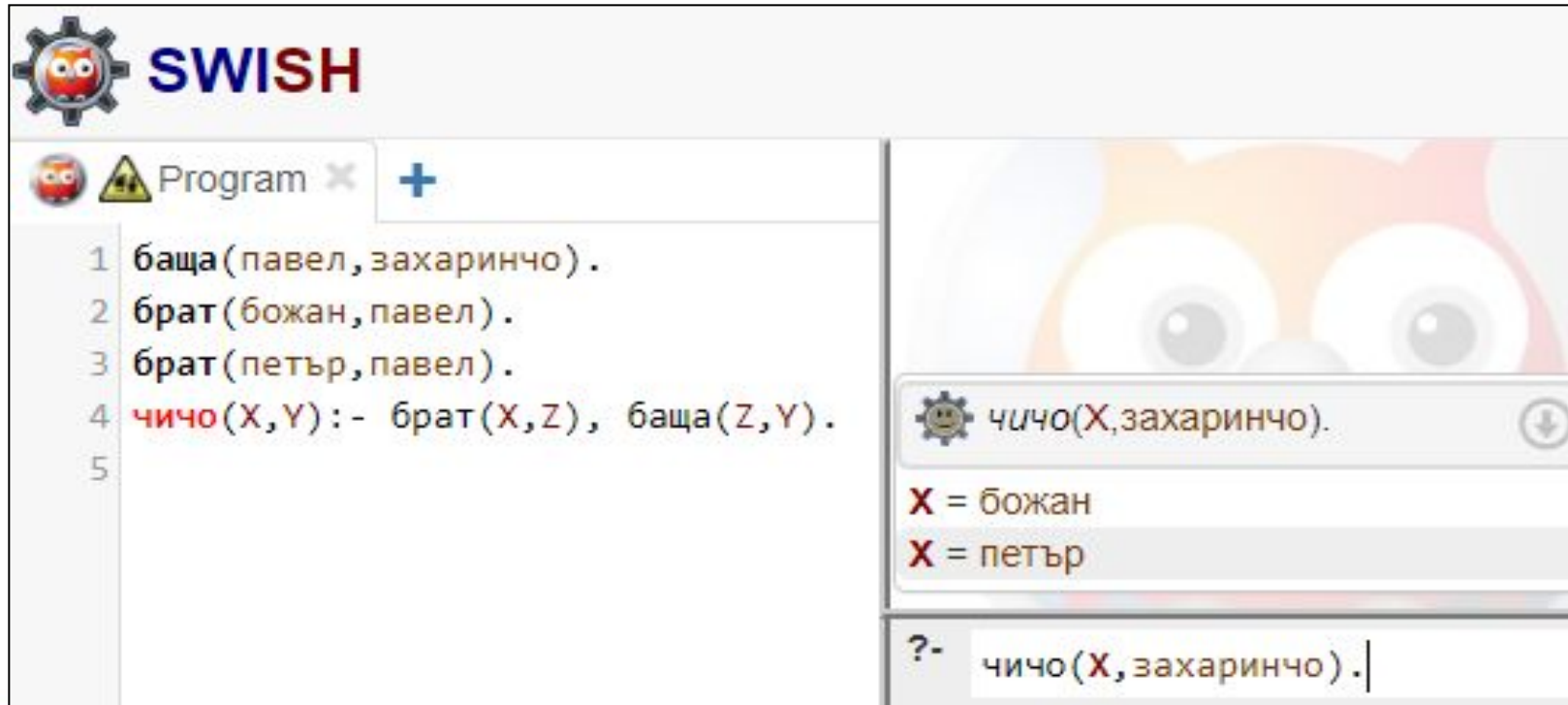
**Като цел в съставни въпроси.** Когато искаме да получим само един отговор можем да използваме „!“ в един съставен въпрос. Нека разгледаме следната примерна БД:

баща(павел,захаринчо).

брат(божан,павел).

брат(петър,павел).

чичо(X,Y):- брат(X,Z), баща(Z,Y).



**SWISH**

Program

```
1 баща(павел, захаринчо).  
2 брат(божан, павел).  
3 брат(петър, павел).  
4 чичо(X,Y):- брат(X,Z), баща(Z,Y).  
5
```

чичо(X,захаринчо).

X = божан  
X = петър

?- чичо(X, захаринчо).|

Ако искаме да разберем кои са чичовците на Захаринчо задаваме следния въпрос:

?- чичо(X, захаринчо).

X = божан ;

X = петър.

Ако искаме, обаче да разберем дали захаринчо изобщо има чичо, можем да формулираме следния съставен въпрос към Пролог:

?- чичо(X, захаринчо), ! .

X = божан.

Друга възможност, при която обаче трябва да знаем името на един от чичовците на захаринчо е следната:

?- чичо(божан, захаринчо).

true.

## Премахване на повторения.

Нека си припомним предиката братя/2 от БД „Железният светилник“, представен по-рано. Версията, която разгледахме страдаше от един малък недостатък – имаше излишно повторение в резултата. Това повторение можем да избегнем посредством използване на „!“, както следва:

```
братя1(X,Y) :- родител(Z,X),родител(Z,Y),мъж(X),  
мъж(Y),X\==Y,!.
```

Искаме да разберем кои са братя в сценария:

```
?- братя(X,Y).
```

```
X = кочо, Y = лазар.
```

## Предотвратяване на механизма на възврат за предишни подцели в правило

С правилото  $r1 :- a, b, !, c.$  се указва на Пролог, че ако решенията за подцели  $a$  и  $b$  са удовлетворени успешно, и въпреки че Пролог би могъл да намери множество решения за  $c$ , с използване на възврат и търсене на алтернативи за  $a$  и  $b$  това няма да се извърши. Не е възможно също използването за механизма на възврат и за друга клауза, която определя предиката  $r1$ .

Нека разгледаме следния пример:

купува\_кола(Модел,Цвят):- кола(Модел,Цвят,  
Цена),

цвят(Цвят,топъл),

Цена < 25000.

кола(мазерати,зелен,25000).

кола(корвет,черен,24000).

кола(порше,червен,23000).

кола(корвет,червен,22000).

цвят(червен,топъл).

цвят(черен,среден).

цвят(син,елегантен).

?- купува\_кола(X, Y).



С този пример отговаря на въпроса какъв модел автомобил да бъде купен, ако желаният цвят е „топъл” , при заложено ценово ограничение.

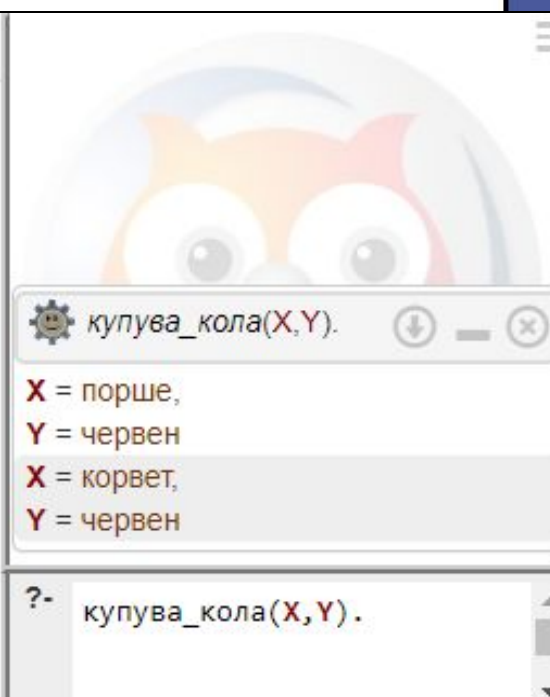
След стартиране на програмата и разглеждане на всички възможни решения с помощта на механизма на възврат се получава и следното решение:

X=порше, Y=червен

X=корвет, Y=червен

2 Solutions

```
Program x +
1 /*PREDICATES
2 nondeterm купува_кола(symbol,symbol)
3 nondeterm кола(symbol,symbol,integer)
4 цвят(symbol,symbol)
5 CLAUSES*/
6 купува_кола(Модел,Цвят) :- кола(Модел,Цвят,Цена),
7   цвят(Цвят,топъл),Цена < 25000.
8 кола(мазерати,зелен,25000).
9 кола(корвет,черен,24000).
10 кола(порше,червен,23000).
11 кола(корвет,червен,22000).
12 цвят(червен,топъл).
13 цвят(черен,неутрален).
14 цвят(син,студен).
```



Ако добавим прекъсване в търсенето на решения, а именно:

купува\_кола(Модел,Цвят):- кола(Модел,Цвят,Цена),  
цвят(Цвят,топъл),!,  
Цена < 25000.

кола(мазерати,зелен,25000).

кола(корвет,черен,24000).

кола(порше,червен,23000).

кола(корвет,червен,22000).

цвят(червен,топъл).

цвят(черен,среден).

цвят(син,елегантен).

?- купува\_кола(X, Y).

Резултата би бил следния:

X=порше, Y=червен

1 Solution

В този случай първият автомобил, който удовлетворява и трите подцели на правилото е крайното решение, въпреки че съществува и друго. Все пак остава въпроса какво би трябвало да се промени, за да се получи резултата:

X=корвет, Y=червен

1 Solution

# Предотвратяване на механизма на възврат за други клаузи на правило:

Прекъсване може да се използва и когато искаме да укажем на Пролог, че е подбрана коректната клауза от множество зададени клаузи. Например:

`r(1):- ! , a , b , c.`

`r(2):- ! , d.`

`r(3):- ! , c.`

`r(_):- write('Това е универсална клауза.').`

Използването на прекъсване тук води до това, че предиката *r* става еднозначно определен. В конкретния случай Пролог използва предиката *r* с единствен целочислен аргумент. Ако се предположи, че това е 1, т.е. за *r(1)* Пролог започва да търси съвпадение и го намира още с първата клауза.

Въпреки, че съществуват повече от една възможности Пролог поставя точка на възврат до тази клауза. По този начин няма да се търси друго удовлетворяване на клаузата *r*. Ако се разгледа и от друг ъгъл този пример, може да се каже, че по този начин се реализира множествен избор като в другите езици за програмиране (switch в C и case в Pascal).

# Предикатът not. Отрицанието като неуспех

Представянето и работата с отрицанието не е тривиален проблем. Интуитивно, на декларативно ниво (декларативна семантика) имаме сравнително ясна представа за отричане на нещо. На процедурно ниво (процедурна семантика) нещата стават значително по-сложни. Защо? Какво значи да отричаме нещо. Как да покажем или изброим всичко останало? По тази причина съществуват различни подходи за практическото реализиране на отрицанието. Един от тях, който се използва в Пролог, се нарича **отрицание като неуспех**.  
Една от най-полезните функции на Пролог е простият начин, по който ни позволява да формулираме обобщения. За да кажем, че Иван обича зеленчуци, просто пишем:

обича(иван, X) :- зеленчуци(X).

Но в реалния живот правилата имат изключения. Да предположим, че Иван не харесва зеленчуци, приготвени на скара. Така, коректното правило е всъщност: Иван обича зеленчуци, които не са приготвени на скара. Как да заявим това в Пролог?

обича(иван, X) :- зеленчуци\_на\_скара(X), !, fail.

обича(иван, X) :- зеленчуци(X).

зеленчуци(X) :- зеленчуци\_на\_пара(X).

зеленчуци(X) :- зеленчуци\_на\_скара(X).

зеленчуци(X) :- зеленчуци\_на\_фурна(X).

зеленчуци\_на\_пара(броколи).

зеленчуци\_на\_скара(домати).

зеленчуци\_на\_пара(моркови).

зеленчуци\_на\_фурна(чушки).

Очакваме, че ако искаме да разберем какви зеленчуци обича Иван, Пролог ще ни потвърди за броколи, моркови и чушки. И наистина, това се случва:

?- обича(иван, броколи).	true .
?- обича(иван, домати).	false.
?- обича(иван, моркови).	true .
?- обича(иван, чушки).	true.

Как работи това? Ключът е комбинацията от ! и fail/0 в първия ред (нарича се комбинация „cut-fail“). Когато задаваме въпроса обича(иван, домати), тогава се прилага първото правило и Пролог достига до отрязването. Това го обвързва с направения вече избор и по-специално блокира достъпа до второто правило. Но тогава се натъква на fail/0. Това предизвиква опит за възврат, но отрязването го блокира и така нашата заявка се проваля.

Това решение е интересно, но не е идеално. Обърнете внимание, че подреждането на правилата е от решаващо значение - ако разменим двете клаузи на предиката обича/2 не получаваме поведението, което искаме.

?- обича(иван, домати).  
true .

Тук отрязването е от решаващо значение. Ако го премахнем, програмата няма да се държи по същия начин (познато е като „червено отрязване“). Накратко, имаме две взаимно зависими клаузи, които използват присъщите на Пролог процедурните аспекти. Решаващото е, че първата клауза по същество е начин да се каже, че Иван не обича X, ако X е подготвен по определен начин. Комбинацията „cut-fail“ ни позволява да определим такава форма на отрицание, наречена „отрицание като неуспех“. Тази комбинация е нещо много полезно, но би било по-добре, ако можем да извлечем полезната част и да я „опаковаме“ по по-приемлив начин. Ето как:

```
neg(Goal) :- Goal, !, fail.
```

```
neg(Goal).
```



За всяка цел на Пролог `neg(Goal)` ще успее точно ако целта не успее. Използвайки новия предикат `neg/1` можем да опишем предпочитанията на Иван по много по-ясен начин:

```
обича(иван, X):-зеленчуци(X),  
neg(зеленчуци_на_скара(X)).
```

```
?- обича(иван, домати).
```

```
false.
```

```
?- обича(иван, чушки).
```

```
true.
```

Тоест Иван обича  $X$ , ако  $X$  не е приготвен на скара. Това е съвсем близко до оригиналното ни изявление.

В стандартния Пролог операторът `\+` означава отрицание като неуспех, така че бихме могли да определим предпочитанията на Иван, както следва:

```
обича(иван, X) :- зеленчуци(X), \+  
зеленчуци_на_скара(X).
```

## Още за декларативната и процедурна семантика

Въпреки, че намерихме приемлива форма за представяне на отрицанието, още един коментар не би бил излишен. Налага ни се да се върнем отново към проблема за декларативната и процедурната семантика. Не трябва да мислим, че „отрицанието като неуспех“ работи точно както „логическото (декларативното) отрицание“. Нека се върнем отново към БД за приготвяне на зеленчуци. Ако искаме да разберем които зеленчуци обича Иван в зависимост от тяхното приготвяне, ще зададем въпроса:

?- обича(иван, X).

X = броколи ;

X = моркови ;

X = чушки.

Получаваме коректната последователност на отговорите. Но сега да предположим, че пренаписваме предиката обича/2 като разменим двете клаузи както следва:

обича(иван, X) :- \+ зеленчуци\_на\_скара(X), зеленчуци(X).

Има ситуации, в които е по-добре да използваме !  
вместо отрицание като неуспех. Например, да  
предположим, че трябва да напишем програма, за да  
изразим следното условие: *p* важи, ако *a* и *b* са в  
сила или ако *a* и *c* не са в сила. Това може да бъде  
директно направено с помощта на отрицанието като  
неуспех:

*p* :- *a*, *b*.

*p* :- \+ *a*, *c*.

Да предположим, че *a* е много сложна цел, която  
отнема много време за изчисляване.

Програмирането по този начин означава, че може да  
се наложи да изчислим *a* два пъти, което би довело  
до неприемливо бавна производителност. Ако е така,  
би било по-добре да използвате следната програма:

*p* :- *a*, !, *b*.

*p* :- *c*.

**Пример:** Нека разгледаме ситуация, в която се търси студент с успех над 3.50, който не е на стаж:

търсен\_студент(Име):-студент(Име,Успех),  
Успех>=3.5, not(стаж(Име)).

студент("Иван Иванов", 3.5).

студент("Димитър Димитров", 2.0).

студент("Георги Георгиев", 3.7).

стаж("Иван Иванов").

стаж("Димитър Димитров").

?-търсен\_студент(X).

X = "Георги Георгиев"

## ЗАДАЧА

Дефинирайте предикат, който задава кой човек какво притежава. Нещата, които могат да се притежават са: банкова сметка с определена сума; апартамент или къща, които се характеризират с местоположение, площ и цена; кола от определена марка и с даден цвят; яхта.

Напишете:

1. Цел, извеждаща какво има Иван;
2. Цел, извеждаща кой има кола.
3. Цел, извеждаща кой каква марка кола има.
4. Цел, установяваща има ли двама с червени коли.
5. Цел, установяваща има ли двама с еднакъв цвят коли.
6. Цел, извеждаща кой има в банковата сметка толкова пари, колкото струва домът му.

7. Предикат, определящ кой какъв дом има (домът е цяла структура) и цел, извеждаща кой какъв дом има.

8. Предикат, определящ големите домове – с площ повече от 200 кв. м. и техните собственици и цел, извеждаща кой има голям дом.

9. Предикат, задаващ кой е богат – ако има къща или апартамент на цена повече от 30000, яхта, кола и банкова сметка над 200000 и цел, извеждаща кой е богат.

10. Предикат, определящ колко е най-голямата банкова сметка и цел, която я извежда.

/\* База факти с данни за това кой какво има \*/

has(ivan,yacht).

has(ivan, car(lada,red)).

has(tanya,bank\_account(200000)).

has(peter,house(plovdiv, space(160),price(48000))).

has(maria,flat(sofia,space(121),price(32000))).

has(kamen, car(renault,blue)).

has(kamen,bank\_account(1005000)).

has(ivan,flat(varna,space(110),price(60000))).

has(ivan,bank\_account(2200000)).

has(maria,bank\_account(600000)).

has(maria,yacht).

has(elena, car(ford,red)).

has(maria,house(pleven, space(300),price(600000))).

has(maria, car(volvo,black)).

has(peter,bank\_account(2000000)).

1. Какво има Иван?
2. Кой има кола?
3. Кой каква марка кола има?
4. Има ли двама с червени коли?
5. Има ли двама с еднакъв цвят коли?
6. Кой има в банковата си сметка толкова пари, колкото струва домът му?
7. а) Предикат, определящ кой какъв дом има (домът е цяла структура)  
б) Кой какъв дом има?
8. а) Предикат определят големите домове – с площ повече от 200 кв.м., и техните собственици  
б) Кой има голям дом?
9. а) Предикат, задаващ кой е богат – ако има къща или апартамент на цена повече от 30000, яхта, кола и банкова сметка над 2000000. (извежда по веднъж всеки богат-оне)  
б) Кой е богат?
10. а) Предикат, определящ колко е най-голямата банкова сметка.  
б) Колко е най-голямата сметка?



1. Какво има Иван?

?- has(ivan,X).

2. Кой има кола?

?-has(X,car(\_,\_)).

3. Кой каква марка кола има?

?-has(X,car(Y,\_)).

4. Има ли двама с червени коли?

?-has(X,car(\_,red)),  
has(Y,car(\_,red)), X\=Y, !.

5. Има ли двама с еднакъв цвят коли?

?-has(X,car(\_,Z)),  
has(Y,car(\_,Z)), X\=Y, !.

6. Кой има в банковата си сметка толкова пари, колкото струва домът му?

?-has(X,bank\_account(Y)),  
(has(X,flat(\_,\_,price(Y))),  
has(X,house(\_,\_,price(Y)))).

7а) Предикат, определящ  
кой какъв дом има (домът е  
цяла структура)

```
home(X,Y):-has(X,Y),  
(Y=flat(_,_,_); Y=house(_,_,_)).
```

б) Кой какъв дом има?

```
?- home(X,U).
```

8а) Предикат определят  
големите домове – с площ  
повече от 200 кв.м., и  
техните собственици.

```
big_home(X,Y):-has(X,Y),  
(Y=flat(_,space(Z),_);  
Y=house(_,space(Z),_)), Z>200.
```

б) Кой има голям дом?

```
?- big_home(X,_).
```

9а) Предикат, задаващ кой е богат – ако има къща или апартамент на цена повече от 30000, яхта, кола и банкова сметка над 2000000. (извежда по веднъж всеки богат-one)

```
rich(X):-has(X,yacht), has(X,car(_,_)), has(X,bank_account(Z)),  
Z>2000000, (has(X,house(_,_price(Y)));  
has(X,flat(_,_price(Y)))), Y>30000.
```

б) Кой е богат?   ?- rich(X).

10а) Предикат, определящ колко е най-голямата банкова сметка.

```
biggest_account(Z):- has(_,bank_account(Z)),  
  \+ (has(_,bank_account(C)), C>Z), !.
```

б) Колко е най-голямата сметка?

```
?-biggest_account(X).
```

# Аритметични изрази и оператори

Те са специален вид терми, които могат да се изчисляват.

Операции: +, -, \*, /, //(целочислено деление), mod остатък

Стандартни функции: abs, int, sqrt, log, log10, sin, cos, ....

Сравняване на аритметични изрази: <, <=, >, >=, := (срвн. за равенство), != (за неравенство)

Пролог поддържа цели числа и целочислена аритметика, така, че във всеки Пролог по някакъв начин са реализирани:

- + събиране
- изваждане, когато искаме в SP да го използваме като унарен минус, трябва да заграждаме аргумента в скоби.

- \* умножение

- // целочислено деление

mod деление по модул на практика работи като остатък

rem остатък

В SP има допълнително:

- \*\* степенуване

**abs** абсолютна стойност естествено, тъй като това и следващото е унарен символ, то те имат префиксен запис и аргументът се загражда в скоби

sign знак.

В SP има и аритметика с плаваща точка. Ако Ви потрябва нещо, можете да погледнете в help-a и да го използвате.

В Пролог, както вече говорихме, има константи- числа, които се изписват по обичайния начин. За работа с числа има следните вградени предикати, които имат инфиксен запис: = , := , \= , =\= , < , > , =< , >= , is

## Примери:

?- a(1,X) = a(1,2).

X = 2

?- a(1,X) == a(1,2).

false

?- X = 2, a(1,X) == a(1,2).

X=2

?-2+3=1+4.

false

?- 2+3:=1+4.

true

?- X=1+4.

X=1+4

?- X is 1+4

X=5

?-X=5, X is  
1+4.

X=5

## Функции

abs(X) , sin(X), cos(X), sqrt(X), ...



Дефинирайте  
предикат за  
намиране  
стойността на  
функцията

$$f(x, y) = \begin{cases} \sqrt{|y^2 - 3x^2|}, & y > x \\ x \cdot \sin x, & x = y, \\ e^x + \ln(x - y), & x > y \end{cases}$$

```
f(X,Y,Result) :- Y>X, Result  
is sqrt(abs(Y*Y - 3*(X*X))).
```

```
f(X,Y,Result) :- X=Y, Result  
is X*sin(X).
```

```
f(X,Y,Result) :- X>Y, Result  
is e**X + log10(X-Y).
```

Дефинирайте  
предикат за  
намиране  
стойността на  
функцията

$$h(y) = \begin{cases} \frac{2 \cdot y^2}{y + 4}, & y < -4 \\ 0, & y \in [-4, 4] \\ \frac{\ln(y - 4) \cdot e^y}{y^3 + \sqrt{y}}, & y > 4 \end{cases}$$

$h(Y, \text{Res}) :- Y < -4, \text{Res is}$   
 $(2 * (Y * Y)) / (Y + 4).$

$h(Y, \text{Res}) :- Y =< 4, Y >= -4, \text{Res}$   
 $\text{is } 0.$

$h(Y, \text{Res}) :- Y > 4, \text{Res is}$   
 $(\log10(Y - 4) * (e^{**Y})) / (Y^{**3} + \text{sqrt}(Y)).$

$$f(x, y) = \frac{\log(3 + x^2 + y)}{2 + y + x^2}, \quad -1 < x < 1 \text{ и } y \neq 0$$