

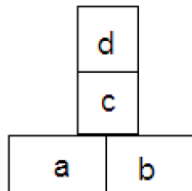
PODSTAWY PROGRAMOWANIA DEKLARATYWNEGO

PROLOG

Ćwiczenia 9

Zadanie 1.

Dany jest świat klocków:



- Utworzyć w Prologu bazę wiedzy opisującą ten świat za pomocą relacji `na/2`, takiej że `na(X,Y)` zachodzi, jeżeli klocek `X` leży na klocku `Y`. Zapisać bazę w pliku `klocki_1.pl`.
- Przeprowadzić sesję prologową w celu uzyskania odpowiedzi na następujące pytania:
 - Czy klocek `d` leży na klocku `c`?
 - Czy klocek `c` leży na klocku `a`?
 - Czy klocek `b` leży na klocku `c`?
 - Jaki klocek leży na klocku `c`?
 - Na jakim klocku leży klocek `c`?
 - Para klocków `(X,Y)`, taka że `X` leży na `Y`?
 - Czy jakiś klocek leży na `c`?
 - Czy jakiś klocek leży pod klockiem `c`?
 - Między jakimi klockami leży klocek `c`?
 - Jaki klocek leży między dwoma innymi klockami?
- Uzupełnić bazę wiedzy `klocki_1` o nowe predykaty : `pod/2` i `między/3`. W definicjach tych predykatów zastosować odpowiednie reguły. Zapisać uzupełnioną bazę wiedzy w pliku `klocki_2.pl` i zadając odpowiednie zapytania sprawdzić poprawność dodanych definicji.

Zadanie 2.

Plik **dane.pl** zawiera informacje o studentach. Zadać pytania (cele) do bazy danych:

- Kto studiuje informatykę?
- Kto nie studiuje w Poznaniu?
- Kto urodził się po roku 1990?
- Kto studiuje w Poznaniu na PP lub UAM?
- Gdzie można studiować informatykę?
- Czy w Łodzi można studiować matematykę?
- Czy Kowalski Jan studiuje matematykę?

Zdefiniować relacje:

- znajdującą wszystkich studentów studiujących w Poznaniu.
- znajdującą studentów urodzonych przed 1994 rokiem, studiujących w Poznaniu informatykę.

- c) znajdującą studentów studiujących informatykę na politechnice.
- d) znajdującą uczelnie, na których studenci studiują matematykę.

Zadanie 3.

Plik **dane.pl** zawiera dane o członkach pewnej rodziny. Zdefiniować podstawowe relacje pokrewieństwa takie, jak **ojciec**, **matka**, **rodzeństwo** **brat**, **siostra**, **dziadek**, **babcia**, **dziadkowie**, **wuj**, **kuzyn**, **przodek**.

Zadanie 4. (3p.)

Zdefiniuj procedurę **nwd(X,Y,Z)**, prawdziwej dla liczb naturalnych, gdy Z jest największym wspólnym dzielnikiem liczb X i Y.

?-nwd(12,14,2).

true.

?-nwd(12,8,X).

X=4.

Zadanie 5. (3p.)

Znajdź przyjaciół (ludzi, którzy się nawzajem lubią) w grupie osób o następujących upodobaniach:

- a) Ola lubi jaroszy, którzy dodatkowo uprawiają sport;
- b) Ewa lubi spędzać czas wśród niepijących kawy jaroszy;
- c) Iza preferuje miłośników książek lub aktywnych sportowo wrogów kawy;
- d) Janek najlepiej czuje się wśród uprawiających sport;
- e) Piotr może się zaprzyjaźnić z uprawiającymi sport jaroszami albo z zapalonymi czytelnikami książek;
- f) Paweł wymaga od przyjaciela, by był jaroszem, uprawiał sport i lubił czytać książki;

jeśli Ola, Ewa, Jan i Paweł są jaroszami, Iza, Piotr i Paweł piją kawę, Ola, Iza i Paweł czytają książki, a Iza, Ola, Piotr i Paweł uprawiają sport.

Uwaga: najpierw należy zapisać fakty w Prologu, a następnie odpowiednie reguły.

W końcu regułę: $\text{przyjaciele}(X,Y):-\text{lubi}(X,Y),\text{lubi}(Y,X),X\neq Y$.

Zadanie 6.

Sprawdź, czy poniższe cele zostaną spełnione i (ewentualnie) które zmienne zostaną jak ukonkretnione:

$\text{rok}(1998)=\text{rok}(1999-1)$.

$\text{lata}(1999,2000,Z)=\text{lata}(\text{l}(X),D,2000)$.

'student'=student.

'Student'=student.

'Student'=Student.

$f(X,X)=f(a,b)$.

$f(X,a(b,c))=f(Z,a(Z,c))$.
 $odcinek(punkt(1, 2), punkt(A)) = odcinek(B, punkt(1, 2))$.
 $odcinek(punkt(1, 2), punkt(A,B)) = odcinek(B, punkt(1, C))$.
 $a(X,p,1)=a(p,Y,1)$.
 $Punkt=punkt(1,2)$.
 $r(a(1),b(X))=r(a(Y),Z)$.
 $1+2=3$.
 $1+2=1+2$.
 $1+2:=3$.
 $1+2=\backslash=3$.
 $X=X$.
 $X=Y$.
 $X==X$.
 $X==Y$.
 $X=1,Y=2,X==Y$.
 $[1,2,3,4]=[A|B]$.
 $[A,B]=[A|B]$.
 $[1,[A],2]=[1,0,2]$.
 $[1,2,3]=[1|2,3]$.
 $[1,2,3]=[1,2|[3]]$.
 $[[A],B,C]=[[a,b,c],[d,e,f],1]$.
 $[W,Z]=[1,2]$.
 $[W,Z]=[1,2|[]]$.
 $[W,Z]=[1,[2]]$.
 $[W,Z]=[1|[2]]$.
 $[A,B,C|D]=[1,2,[a],5]$.
 $[A|[S|W]]=[1,[2],[a,b]]$.
 $[A,b|C]=[D|R]$.
 $[a,b,c,[1,2,3]]=[a,Q|N]$.
 $[a|M]=[a,b,r,z|[1,2,3]]$.
 $1+2+3=W+3$.
 $1+2-3=1+K$.
 $[1+2,3+4]=[X|Y]$.

Zadanie 7.

Sprawdzić działanie procedur działających na listach:

is_list(L), **append**(L1,L2,L3), **member**(E,L), **memberchk**(E,L), **nextto**(X,Y,L),
delete(L1,E,L2), **select**((E,L,R), **nth0**(I,L,E), **nth1**(I,L,E), **last**(L,E), **reverse**(L1,L2),
permutation(L1,L2), **flatten**(L1,L2), **sumlist**(L,S), **numlist**(M,N,L), **length**(L,I)
sort(L1,L2), **msort**(L1,L2)

is_list(L) - sprawdza, czy L jest listą

Sprawdzić np.

`is_list([1,2,3,c,d]).`

`is_list(5).`

`is_list([5]).`

append(L1,L2,L3) – łączy listy L1 i L2 w listę L3

Sprawdzić np.

```
append([b,c,d],[e,f,g,h],X).  
append([a],[b],[a,b]).  
append(L1,L2,[b,c,d]).
```

member(E,L) – sprawdza, czy element E należy do listy L

Sprawdzić np.

```
member(a,[b,c,[s,a],a]).  
member(a,[b,c,[s,a]]).  
member([s,a],[b,c,[s,a]]).  
member(X,[a,b,c]).  
member(a, X).
```

memberchk(E,L) - równoważny predykatowi member, ale podaje tylko jedno rozwiązanie

Sprawdzić np.

```
member(Y,[1,2,3,4]).  
memberchk(Y,[1,2,3,4]).
```

nextto(X,Y,L) – predykat spełniony, gdy Y występuje bezpośrednio po X

Sprawdzić np.

```
nextto(X,Y,[a,c,d,r]).  
nextto(w,Y,[q,w,e,r]).  
nextto(X,4,[2,3,4,5]).
```

delete(L1,E,L2) – z listy L1 usuwa wszystkie wystąpienia elementu E, wynik uzgadnia z listą L2

Sprawdzić np.

```
delete([1,2,3,4],4,M).  
delete([2,1,2,1,2,1],1,K).
```

select((E,L,R) – z listy L wybiera element, który daje się uzgodnić z E. Lista R jest uzgadnianą z listą, która powstaje z L po usunięciu wybranego elementu

Sprawdzić np.

```
select(1,[2,1,2,1],K).  
select(X,[1,2,3],K).  
select(0,X,[1,2,3,4]).
```

nth0(I,L,E) – predykat spełniony, jeśli element listy L o numerze I daje się uzgodnić z elementem E

Sprawdzić np.

```
nth0(2,[a,b,c,d],X).  
nth0(X,[a,b,c,d],2).  
nth0(X,[a,b,c,d],c).
```

nth1(I,L,E) – predykat podobny do nth0. Sprawdzić różnicę!

last(L,E) – ostatni element listy L

Sprawdzić np.

```
last([1,2,3,4],L).  
last(X,2).
```

reverse(L1,L2) – odwraca porządek elementów listy L1 i unifikuje rezultat z listą L2
Sprawdzić np.

reverse([1,2,3,4],X).
reverse(Y,[a,b,c,d,e,f]).

permutation(L1,L2) – lista L1 jest permutacją listy L2
Sprawdzić np.

permutation([1,2,3],L).
permutation(M,[4,5,6,7]).

flatten(L1,L2) – przekształca listę L1 w listę L2, w której każda lista składowa zostaje zastąpiona przez swoje elementy
Sprawdzić np.

flatten([a,[b,[c,d],e,f]],X).
flatten([1,[5],[3],[8,[4]]],L).

sumlist(L,S) – suma listy liczbowej L
Sprawdzić np.

sumlist([1,2,3,4],X).
sumlist([1,2,3,4],10).

numlist(M,N,L) – jeśli M,N są liczbami całkowitymi takimi, że $M < N$, to L zostanie zuniifikowana z listą $[M, M+1, \dots, N]$
Sprawdzić np.

numlist(2,8,L).
numlist(-3,5,X).

length(L,I) – liczba elementów listy L
Sprawdzić np.

length([1,3,4,23,21,8],L).
length([a,e,[a],[x,y],l],T).

Analizując poniższe przykłady wyjaśnić różnicę między predykatami **sort** i **msort**.

sort([1,9,3,2,4,0],W).
sort([1,2,1,3,4,3,6,5,5,9,1],P).
msort([1,2,1,3,4,3,6,5,5,9,1],R).
msort([a,n,t,r,e,w],Q).

Zadanie 8. (2p.)

Czy można użyć **select** do dodawania elementu do listy? Jeśli tak, to w jaki sposób?

Uwaga:

Wykonane zadania 4,5,8 należy przekazać do **23.05.2018, 23:59** przez OLAT „Ćwiczenia 9” w jednym pliku .pl. Nazwa pliku ma zawierać nazwisko Studenta i numer ćwiczeń.