LAB: Praca z listami w Prologu

# 1. Notacja list

Lista to uporządkowany zbiór elementów. Elementem może być dowolna struktura w Prologu (czyli term). Listę zapisujemy:

[a,b,c]  
[2,4,6,ala,ma,kota]  
[]

Każda lista składa się z głowy (ang. head) oraz ogona (ang. tail).

Głowę od ogona rozdzielamy operatorem | (pionowa kreska), np.:

?- [X|Y]=[a,b,c,d].  
X = a  
Y = [b, c, d] ;

Dekompozycja i strukturalizacja list jest realizowana przez mechanizm unifikacji. Na przykład:

?- [X,Y|Z]=[a,b,c,d].  
X = a  
Y = b  
Z = [c, d] ;  
  
?- [X,Y,a]=[Z,b,Z].  
X = a  
Y = b  
Z = a ;

Ćwiczenie:  
Proszę sprawdzić poniższe unifikacje:  
  
?- X=[a,b,c].  
?- [X|Y]=[a,b,c].  
?- [[a,b],c]=[X,Y].  
?- [a(b),c(X)]=[Z,c(a)].  
?- [X|Y]=[a].

Wybieranie elementu:

trzeci([A,B,C|Reszta],C).

Ćwiczenie:  
Zdefiniować predykat porównujący 2 wybrane elementy listy, np. 3. i 4.  
  
Przykład użycia:  
?- porownaj([a,b,c,d]).  
false  
?- porownaj([a,b,c,c]).  
true  
  
Zdefiniować predykat zamieniający 2 wybrane elementy listy, np. 3. i 4.  
  
?- zamien([a,b,c,d],X).  
X=[a,b,d,c]  
true

# 2. Przynależność do listy

nalezy(X,[X|\_]).  
nalezy(X,[\_|Yogon]) :-  
 nalezy(X,Yogon).

Ćwiczenie:  
Dopisać predykat nalezy do pliku listy-1.pl  
  
Sprawdzić i przemyśleć działanie poniższych:  
  
?- nalezy(c,[a,b,c,d]).  
?- nalezy(x,[a,b,c,d]).  
?- nalezy(X,[a,b,c,d]).  
?- nalezy(x,a).  
?- nalezy(X,a).

# 3. Liczenie elementów

dlugosc([],0).  
dlugosc([\_|Ogon],Dlug) :-  
 dlugosc(Ogon,X),  
 Dlug is X+1.

Ćwiczenie:  
Dopisać predykat dlugosc do pliku listy-1.pl  
  
Sprawdzić i przemyśleć działanie poniższych:  
  
?- dlugosc([a,b,c],X).

# 4. Rekurencyjna analiza list

a2b([],[]).

a2b([a|Ta],[b|Tb]) :-

a2b(Ta,Tb).

Ćwiczenie

Dopisać predykat a2b do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- a2b([a,a,a],[b,b,b]).

?- a2b([a,a,a,a],[b,b,b]).

?- a2b([a,s,d],[b,s,d]).

?- a2b([a,a,a,a],X).

?- a2b(X,[b,b]).

?- a2b(X,Y).

Uwaga: ten predykat robi „coś ciekawego” tylko na listach zaczynających się od a i b!

# 5. Sklejanie list

sklej([],X,X).

sklej([X|L1],L2,[X|L3]) :-

sklej(L1,L2,L3).

Ćwiczenie

Dopisać predykat sklej do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- sklej([a,b],[c,d],X).

?- sklej([a,b],X,[c,d]).

?- sklej([a,b],X,[a,b,c,d]).

?- sklej(A,B,[a,b,c,d,e]).

?- sklej([1,2,3],[a,b,c],X).

?- sklej([1,2,3],[a(e),b(f),c(d,g)],X).

?- sklej(Przed,[5|Po],[1,2,3,4,5,6,7,8,9]).

?- sklej(\_,[A,5,B|\_],[1,2,3,4,5,6,7,8,9]).

?- sklej(A,[x,x,x|\_],[a,b,x,x,c,x,x,x,d,e]).

Uwaga: dopisać i przetestować predykat:

nalezy1(X,L) :-

sklej(\_,[X|\_],L).

Zdefiniować predykat:

ostatni(Element,Lista).

Z użyciem i bez użycia sklej.

# 6. Dodawanie elementów

dodaj(X,L,[X|L]).

W praktyce nie byłby tu potrzebny dodatkowy predykat.

Ćwiczenie

Dopisać predykat dodaj do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- dodaj(a,[c,d],X).

?- dodaj([a,b],[c,d],X).

7. Usuwanie elementów

usun(X,[X|Reszta],Reszta).

usun(X,[Y|Ogon],[Y|Reszta]) :-

usun(X,Ogon,Reszta).

Ćwiczenie

Dopisać predykat usun do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- usun(a,[a,b,a,c,a,a],X).

?- usun(a,[a,b,c,d],X).

?- usun(c,[a,b,c,d],X).

?- usun(c,X,[a,b,c,d]).

?- usun(1,X,[a,b,c,d]).

Proszę znajdywać wszystkie rozwiązania (;).

Uwaga: dopisać i przetestować predykat:

wstaw(X,L,Duza) :-

usun(X,Duza,L).

Uwaga: dopisać i przetestować predykat:

nalezy2(X,L) :-

usun(X,L,\_).

# 8. Zawieranie list

zawiera(S,L) :-

sklej(\_,L2,L),

sklej(S,\_,L2).

Ćwiczenie

Dopisać predykat zawiera do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- zawiera(a,[a,b,c]).

?- zawiera([a],[a,b,c]).

?- zawiera(X,[a,b,c]).

?- zawiera([X],[a,b,c]).

?- zawiera([X,Y],[a,b,c]).

?- zawiera([X,Y,Z],[a,b,c]).

# 9. Permutacje list

permutacja([],[]).

permutacja([X|L],P) :-

permutacja(L,L1),

wstaw(X,L1,P).

permutacja2([],[]).

permutacja2(L,[X|P]) :-

usun(X,L,L1),

permutacja2(L1,P).

Ćwiczenie

Dopisać predykat permutacja do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- permutacja([a,b,c],X).

?- permutacja2([a,b,c],X).

# 10. Odwracanie list

odwroc([],[]).

odwroc([H|T],L) :-

odwroc(T,R),

sklej(R,[H],L).

Ćwiczenie

Dopisać predykat odwroc do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- odwroc([a,b,c,d],X).

?- odwroc([a,b,c,d],[d,c,b,a]).

# 11. Listy a napisy

Napis w Prologu może być reprezentowany przez:

atom - jest trudny w obróbce,

listę kodów ASCII znaków,

listę jednoliterowych atomów.

Reprezentacja przy pomocy listy zwiększa możliwości przetwarzania.

Przydatny predykat:

wypisz([H|T]) :-

put(H), wypisz(T).

wypisz([]).

Inna sytuacja: Przykład wykorzystania wbudowanych predykatów name i append do przekształcania napisów. Predykat plural(Noun, Pl) - przekształca rzeczownik regularny języka angielskiego z liczby pojedynczej na liczbę mnogą.

plural(Noun, PL) :-

name(Noun, L),

name(s,T),

append(L,T, NL),

name(PL, NL).

Ćwiczenie

Sprawdzić i przemyśleć działanie poniższych:

?- write('ala').

?- write('ala ma kota').

?- write("ala").

?- write("ala ma kota").

?- X = 'a', put(X).

?- X = 'ala', put(X).

?- X = "a", put(X).

?- put(65),put(66),put(67).

Dopisać predykaty wypisz i plural do pliku listy-1.pl

Sprawdzić i przemyśleć działanie poniższych:

?- wypisz("ala ma kota").

?- permutacja("abc",X),wypisz(X),nl,fail.

?- wstaw(" ", "abcdefgh",Z),wypisz(Z),nl,fail.

?- plural(day,PL).

Dla Zainteresowanych

# Minecraft

Zastanówmy się nad próbą implementacji prostej gry typu Minecraft. Kluczowym elementem gry jest silnik wokselowy, który odpowiada za renderowania ogromnego świata przy użyciu prostych klocków zwanych wokselami. Zaczniemy od rysowania przykładowego woksela. Proszę uruchomić program cuboid.pl (wymagane XPCE) i przeanalizować jego działanie.

?- cuboid(2,2,2).

Następnie proszę zapoznać się z kodem odpowiadającym za animację animation.pl:

?- sm.

Zadania:

Przerobić predykat cuboid tak, żeby można było sprecyzować jego pozycję w przestrzeni

Napisać predykat cuboids, który przyjmuje listę współrzędnych i rysuje w nich sześciany o zadanej długości boku

Przy pomocy predykatu cuboids należy zamodelować złożony obiekt ze świata Minecraft

Bazując na kodzie z animation.pl należy wprawić krowę w ruch sinusoidalny, imitujący kota z filmu

[Dla odważnych] dodać do animacji dźwięk podobny do tego z filmu

Pytania:

Czy rozsądne jest rysowanie wszystkich klocków? Jak wykryć, które klocki są widoczne z perspektywy gracza? Pomocny może okazać się link

Czy możliwe jest zrobienie w podobny sposób imitacji poniższego filmu?

Efektywność odwracania list

Predykat podany w sekcji odwracanie list, tzw. naiwny, używa predykatu sklej, co powoduje jego nieefektywność. Innym rozwiązaniem jest użycie tzw. akumulatora, w którym przechowujemy budowaną na bieżąco listę. Opis procedury: tutaj

W kolejnych krokach predykat ten buduje listę Accumulator:

List: [a,b,c,d] Accumulator: []

List: [b,c,d] Accumulator: [a]

List: [c,d] Accumulator: [b,a]

List: [d] Accumulator: [c,b,a]

List: [] Accumulator: [d,c,b,a]

Klauzule wyglądają następująco:

odwroc2(L,R) :-

odwr2(L,[],R).

odwr2([H|T],A,R) :-

odwr2(T,[H|A],R).

odwr2([],A,A).

# 12. Listy różnicowe

Dużą bolączką operacji na listach w Prologu (jak też w innych językach programowania, które wymuszają przechodzenie po elementach listy od lewej do prawej, np. Haskell) jest nieefektywność operacji działających na końcu listy. Sztandarowym przykładem takiej operacji jest łączenie dwóch list, vel. predykat sklej/3 implementowany na tych laboratoriach — rezolucja musi w nim przejść kolejno po wszystkich elementach pierwszej listy; złożoność czasowa jest zatem liniowa i przy częstym sklejaniu list znacząco spowalnia działanie programu.

Standardowym rozwiązaniem problemu są listy różnicowe (ang. difference lists), które reprezentują jedną listę jako różnicę między dwiema listami. Różnicę rozumiemy jako odjęcie elementów z drugiej listy od końca pierwszej listy, np. lista

[a,b,c].

może być reprezentowana równoważnie przez wszystkie poniższe pary list:

[a,b,c,d,e],[d,e].

[a,b,c,d,e,f],[d,e,f].

[a,b,c],[].

[a,b,c|[d,e,f,g]],[d,e,f,g].

[a,b,c|[]],[].

[a,b,c|End],End.

Szczególnie interesująca jest ostatnia linijka ze względu na swoją ogólność. Każdą listę [L] można przedstawić jako parę [L|End],End. Kiedy End jest zmienną bez przypisanej wartości, listę o takiej postaci nazywamy otwartą (ang. open list). Z proceduralnego punktu widzenia zmienna End jest „wskaźnikiem” na koniec listy L; unifikując z End inną listę, wstawiamy jej elementy na koniec listy L. Poniżej przedstawiony jest predykat sklej\_roznicowo/3, który wykonuje tę operację (uwaga: operator '-' służy tutaj jedynie grupowaniu argumentów; parę [L|End], End zapiszemy jako [L|End] - End).

sklej\_roznicowo(L - End, End, L).

Poniżej przykład wywołania predykatu:

?:- sklej\_roznicowo([a,b,c|End]-End,[d,e,f],Wynik).

End = [d, e, f],

Wynik = [a, b, c, d, e, f].

Predykat ten różni się od tego implementowanego na zajęciach pod względem złożoności obliczeniowej; sklej\_roznicowo nigdy nie wchodzi w rekurencję, wykonuje jedynie jedną unifikację, ma więc stałą złożoność obliczeniową.

Idąc dalej, jeżeli ustalimy, że argumentami sklej\_roznicowo mogą być jedynie listy różnicowe w postaci par L - End, możemy go przepisać do poniższej, interesującej postaci:

sklej\_roznicowo(L - End, End - End2, L - End2).

Poniżej przykład wywołania predykatu:

?:- sklej\_roznicowo([a,b,c|End]-End,[d,e,f|End2]-End2,Wynik).

End = [d, e, f|End2],

Wynik = [a, b, c, d, e, f|End2]-End2.

odczas wywołania Prolog unifikuje koniec pierwszej listy z drugą listą, zapamiętując przy tym koniec drugiej listy — dzięki temu lista wynikowa Wynik w parze z End2 stanowi kolejną listę różnicową, do której łatwo dołożyć kolejne elementy.

Do przećwiczenia:

przetestować predykat sklej\_roznicowo/3;

spróbować zastąpić predykat sklej/3 predykatem sklej\_roznicowo/3 w predykatach zaimplementowanych na zajęciach;

zastanowić się, jakie przewagi ma sklej/3 nad sklej\_roznicowo/3;

przepisać wybrany predykat z zajęć (poza sklej/3) na wersję korzystającą z list różnicowych.

Do poczytania:

<http://en.wikipedia.org/wiki/Difference_list>

<http://en.wikibooks.org/wiki/Prolog/Difference_Lists>

<http://stackoverflow.com/a/20441480>

<http://homepages.inf.ed.ac.uk/pbrna/prologbook/node180.html>

# 14. Problemy do samodzielnego rozwiązania

Proszę rozwiązać następujące problemy:

1. zdefiniować predykat, powodujący usunięcie 3 ostatnich elementów listy L, w wyniku powstaje lista L1, użyć sklej.
2. zdefiniować predykat, powodujący usunięcie 3 pierwszych elementów listy L, w wyniku powstaje lista L1, użyć sklej.
3. zdefiniować predykat, powodujący usunięcie 3 pierwszych i ostatnich elementów listy L, w wyniku powstaje lista L2, użyć sklej.
4. zdefiniować parę komplementarnych predykatów nieparzysta(L) oraz parzysta(L) sprawdzajacych czy argument jest listą o odpowiednio nie/parzystej długości.
   * czy Twój predykat potrafi również **utworzyć** listę o zadanej parzystości? (jako argument podajemy niewiadomą, a nie stałą)
5. zdefiniować predykat palindrom(L), L jest palindromem, jeżeli czyta się tak samo od przodu i tyłu, np. [a,l,a], [m,a,d,a,m]. (podpowiedź: można nie/użyć odwroc.)
6. zdefiniować predykat przesun(L1,L2), gdzie L2, jest przesuniętą rotacyjnie o jeden element L1, np.:

?- przesun([1,2,3,4,5,6,7,8],X),przesun(X,Y),przesun(Y,Z).

X = [2, 3, 4, 5, 6, 7, 8, 1]

Y = [3, 4, 5, 6, 7, 8, 1, 2]

Z = [4, 5, 6, 7, 8, 1, 2, 3]

1. zdefiniować predykat przeloz(L1,L2), który zamienia listę liczb (max. 0-9), na listę słów:

?- przeloz([1,4,7],X).

X = [jeden, cztery, siedem] ;

?- przeloz(A,[dwa,osiem,zero]).

A = [2, 8, 0] ;

posługując się faktami:

znaczy(0,zero). znaczy(1,jeden).

znaczy(2,dwa). znaczy(3,trzy).

znaczy(4,cztery). znaczy(5,piec).

znaczy(6,szesc). znaczy(7,siedem).

znaczy(8,osiem). znaczy(9,dziewiec).

Podpowiedź: predykat ma być rekurencyjny.

1. zdefiniować predykat podzbior(L,Z), który sprawdza, czy Z zawiera się w L, oraz wypisuje wszystkie możliwe podzbiory L (jeżeli Z jest niewiadoma).

?- podzbior([a,b,c],[c]).

Yes

?- podzbior([a,b,c],[a,c]).

Yes

?- podzbior([a,b,c],X).

X = [a, b, c] ;

X = [a, b] ;

X = [a, c] ;

X = [a] ;

X = [b, c] ;

X = [b] ;

X = [c] ;

X = []

1. zdefiniować predykat podziel(L,L1,L2), który dzieli listę L, na dwa fragmenty L1 i L2, mniej więcej równej długości (z dokładnością do jednego el.), np.:

?- podziel([],X,Y).

X = []

Y = [] ;

?- podziel([1],X,Y).

X = [1]

Y = [] ;

?- podziel([1,2],X,Y).

X = [1]

Y = [2] ;

?- podziel([1,2,3],X,Y).

X = [1, 3]

Y = [2] ;

?- podziel([1,2,3,4],X,Y).

X = [1, 3]

Y = [2, 4] ;

?- podziel([1,2,3,4,5],X,Y).

X = [1, 3, 5]

Y = [2, 4] ;

?- podziel([1,2,3,4,5,6,7,8],X,Y).

X = [1, 3, 5, 7]

Y = [2, 4, 6, 8] ;

1. zdefiniować predykat splaszcz, który zamienia dowolnie zagnieżdżoną listę, w listę płaską (której el. nie są listami). (podstawowe rozwiązanie działa bez nawrotów - nie należy naciskać ;)

?- splaszcz([[a],b,c],X).

X = [a, b, c]

?- splaszcz([[a],[b,[d]],c],X).

X = [a, b, d, c]

?- splaszcz([a,b,c],X).

X = [a, b, c]

?- splaszcz(a,X).

X = [a]

1. Napisz program który obliczy na jakie monety można rozmienić zadaną sumę pieniędzy.
   * Zdefiniuj nominały monet: np. moneta(1) oznacza monetę jednozłotową,
   * Predykat rozmieniający powinien mieć dwa argumenty: rozmien/2, gdzie pierwszy to kwota, a drugi lista nominałów monet na jakie można rozmienić kwotę; uwaga: predykat będzie niedeterministyczny.

# 15. Przechwytywanie wyników

Z Prologiem dostarczonych jest kilka predykatów przydatnych przy obróbce wyników wyszukiwania.

Predykat *bagof/3*, użyty jako bagof(X,P,L) buduje listę L, złożoną z takich X, że spełnione jest P.

Podobnie działa *setof/3*, jednak powstała lista jest posortowana i nie zawiera ew. duplikatów.

Specjalny operator ^ pozwala na modyfikowanie zapytania i jest równoważny kwantyfikacji egzystencjalnej, np. zakładając istnienie bazy faktów zdefiniowanej za pomocą predykatu a/2:

* bagof(X,Y^a(X,Y),L) spowoduje znalezienie listy L na ktorej beda znajdować się wartości X niezależnie od tego jaką wartość przyjmuje Y (dokładnie jedno rozwiązanie).
* bagof(X,a(X,Y),L) spowoduje znalezienie listy L na ktorej beda znajdować się wartości X dla konkretnej (znalezionej) wartości Y (wiele rozwiązań, lista dla każdej wartości Y).

( składnia z ^ nie działa (w zainst. wersji SWI), jeżeli jest użyta jako cel w powłoce SWI - należy zdefiniować odpowiedni predykat jej używający w pliku. )

Predykat *findall/3* wymusza wyszukanie wszystkich możliwych wyników.

Ćwiczenie: Wczytać program rodzina1.pl

Sprawdzić działanie:

?- rodzic(X,robert).

?- bagof(X,rodzic(X,robert),L).

Sprawdzić działanie:

?- bagof(X,ojciec(tomek,X),L).

?- setof(X,ojciec(tomek,X),L).

Następnie:

?- bagof(X,Y^ojciec(X,Y),L).

?- setof(X,Y^ojciec(X,Y),L).

Oraz:

?- bagof(X,ojciec(X,Y),L).

?- findall(X,ojciec(X,Y),L).

# 16. Labirynt

Dany jest program poszukujacy drogi w labiryncie: maze.pl Proces poszukiwania uruchamiany jest za pomocą predykatu solve\_maze/.

Głównym predykatem definiujacym algortym poszukiwania jest path/2.

Labirynt znajduje się na ponumerowanych polach. Połączenia pomiędzy polami, którymi można przejść zdefiniowane są za pomocą predykatu connect/2. Wejście do labiryntu oznaczone jest start, wyjście: finish. Poniższy przykład definoiuje bardzo prosty labirynt.

connect(start,1).

connect(1,2).

connect(2,3).

connect(2,4).

connect(3,finish).

Ćwiczenie:

1. Przeanalizuj program maze.pl. Do czego służy predykat connected\_to/2?

2. Napisz odpowiednie klauzule predykatu connect/2 dla labiryntu danego na rysunku poniżej oraz uruchom program poszukujący drogi. [Obraz zawierający tekst, diagram, Prostokąt

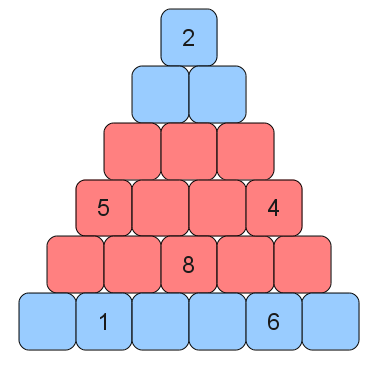
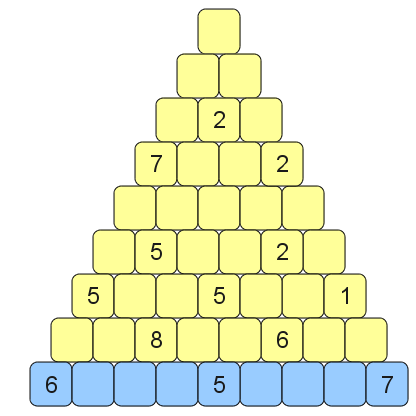
Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.](https://web.archive.org/web/20200808095211/https:/ai.ia.agh.edu.pl/_detail/pl:prolog:prolog_lab:maze.jpg?id=pl%3Aprolog%3Aprolog_lab%3Alisty2)

3. Zmodyfikuj labirynt, tak aby do wyjścia prowadziła więcej niż jedna droga. Czy solve\_maze/0 znajdzie więcej niż jedną drogę? Czy path/2 znajdzie więcej niż jedną drogę?

# 17. Zagadka

Na blokach piramidek (patrz rysunki) należy umieścić cyfry od 1 do 9. W niebieskich rzędach cyfry muszą być różne, w żółtych - mogą się powtarzać, a w różowych - przynajmniej jedna powtórka jest obowiązkowa. Każda cyfra (poza umieszczonymi w podstawie) musi być sumą **lub** różnicą dwu cyfr znajdujących się bezpośrednio pod nią. Część liczb jest już na swoich miejscach. Napisz program w Prologu, który poda rozwiązania piramidek.

Wskazówka: Przeczytaj *„helpa”* do predykatu *list\_to\_set/2*. Może się okazać przydatny…

[](https://web.archive.org/web/20200808095211/https:/ai.ia.agh.edu.pl/_detail/pl:prolog:prolog_lab:zagadka-1.png?id=pl%3Aprolog%3Aprolog_lab%3Alisty2)[](https://web.archive.org/web/20200808095211/https:/ai.ia.agh.edu.pl/_detail/pl:prolog:prolog_lab:zagadka-2.png?id=pl%3Aprolog%3Aprolog_lab%3Alisty2)