Elementy języka **Prolog**

- Cechy języka
- Podstawy
- Zamrażanie celów (korutyny)
- Programowanie ograniczeń

Cechy języka

- Deklaratywne programowanie w logice
- Formuły rachunku predykatów
- Rezolucja liniowa
- Atrybuty zmiennych

Podstawy

Termy

- zmienne: X, _, _N
- stałe: janek, 'Janek', []
- termy złożone: f(t1, t2, …, tn)

Stałe interpretujemy jako obiekty (indywidua, rzeczy, dane, itp.).

Funktory n-argumentowe interpretujemy jako n-argumentowe funkcje.

janek — chłopiec o imieniu Janek

ojciec/1 — jednoargumentowy funktor interpretowany jako funkcja przyporządkowująca osobie jej ojca

matka/1 — jednoargumentowy funktor interpretowany jako funkcja przyporządkowująca osobie jej matkę

term	interpretacja
janek	chłopiec Janek
matka(janek)	matka Janka
ojciec(janek)	ojciec Janka
matka(matka(janek))	babcia Janka ze strony matki
matka(ojciec(janek))	babcia Janka ze strony ojca
ojciec(matka(janek))	dziadek Janka ze strony matki
ojciec(ojciec(janek))	dziadek Janka ze strony ojca

Listy

Funktor ./2 łączy głowę H z ogonem listy T w listę .(H, T)

[] stała reprezentująca listę pustą

Zapis uproszczony listy: [e1, e2, …, en] [e1, e2 | T]

```
.(1, .(2, .(3, .(4, [])))) = [1,2,3,4]
                           = [1,2,3,4 | []]
                           = [1,2,3 | [4]]
                           = [1,2 | [3,4]]
                           = [1 | [2,3,4]]
```

Unifikacja

Podstawienie $\sigma = \{X_1/t_1, X_2/t_2, \ldots, X_m/t_m\}$.

Zastosowanie t σ podstawienia σ do termu t:

$$t\sigma = \begin{cases} X & \text{gdy } t = X \text{ i nie jest podstawiane w } \sigma \\ t_i & \text{gdy } t = X_i \\ c & \text{gdy } t = c \\ f(s_1\sigma, \ldots, s_n\sigma) & \text{gdy } t = f(s_1, \ldots, s_n) \end{cases}$$

Przykład:

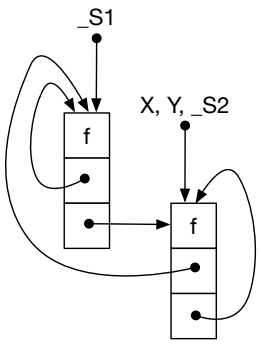
$$f(X, g(Y, Z))\{X/a, Y/b\} = f(a, g(b, Z))$$

Unifikatorem dwóch termów *t* i *s* jest takie podstawienie σ, że $t\sigma$ i $s\sigma$ są identyczne.

Algorytm Martelli-Montanari (złożoność liniowa, średnia stała):

```
S←{s=t}  // układ równań
while S zmienia się do
  if f(s1,…,sn)=f(t1,…,tn)∈S then
    zastąp f(s1,…,sn)=f(t1,…,tn)
    przez s1=t1, …, s1=tn
  if f(s1,…,sm)=g(t1,…,tn)∈S then
    zakończ z niepowodzeniem
  if X=X∈S then usuń X=X z S
  if t=X∈S i t nie jest zmienną then
    zastąp t=X przez X=t
  if X=t∈S i X występuje w t then    // occurs check
    zakończ z niepowodzeniem
  if X=t∈S i X występuje gdzieś w S then
    zastosuj do innych równań {X/t}
end while
```

```
?- f(X) = f(g(a)).
X = g(a)
?- f(X, Y, Z) = f(a, g(X, X), g(Y, Y)).
X = a
Y = g(a, a)
Z = g(g(a, a), g(a, a))
?- X = f(X).
X = f(X) % term cykliczny gdyż Prolog nie wykonuje occurs check
?- X = f(X, Y), Y = f(X, Y).
X = Y, Y = _S2, % where
    _S1 = f(_S1, _S2),
    _S2 = f(_S1, _S2).
```




```
?- X = f(X, Y), Y = f(X, Y), Z = Z, Z).
X = Y, Y = Z, Z = f(Z, Z).
```

Programy

Program składa się z faktów i reguł.

```
lubi(przemko, programowanie).
```
Przemko lubi programowanie.

```
lubi(przemko, X) :-
    lubi(X, programowanie).
```
Przemko lubi tych, którzy lubią programowanie.

```
?- lubi(przemko, X).
```
Co lubi Przemko?

```
X = programowanie ;
X = przemko ;
false.
?- lubi(X, przemko).
```
Kto lubi Przemka?

```
X = przemko ;
false.
```

```
member(X, [X | _]).
member(X, [_ | Y]) :-
    member(X, Y).

?- member(2, [1, 2, 3]).
true.
?- member(X, [1, 2, 3]).
X = 1 ;
X = 2 ;
X = 3 ;
false.
```

Konkatenacja w Haskellu:

```
append :: [a] -> [a] -> [a]
append [] xs = xs
append (x:xs) ys = x:append xs ys
```

Konkatenacja w Prologu:

```
append([], Xs, Xs).
append([X | Xs], Ys, [X | Zs]) :-
    append(Xs, Ys, Zs).

?- append([1, 2, 3], [4, 5, 6], X).
X = [1, 2, 3, 4, 5, 6].
?- append(X, Y, [1, 2, 3]).
X = [], Y = [1, 2, 3] ;
X = [1], Y = [2, 3] ;
X = [1, 2], Y = [3] ;
X = [1, 2, 3], Y = [] ;
false.
```

```
select(X, [X | Xs], Xs).
select(X, [Y | Ys], [Y | Zs]) :-
    select(X, Ys, Zs).

?- select(X, [1, 2, 3], L).
X = 1, L = [2, 3] ;
X = 2, L = [1, 3] ;
X = 3, L = [1, 2] ;
false.
?- select(0, X, [1, 2, 3]).
X = [0, 1, 2, 3] ;
X = [1, 0, 2, 3] ;
X = [1, 2, 0, 3] ;
X = [1, 2, 3, 0] ;
false.
```

Film Woody Allena „Seks nocy letniej”.

Maszynka do wyciągania ości z ryby.

Ta sama maszynka może również wkładać ości w rybę.

```
permutacja([], []).
permutacja(Xs, [X | Zs]) :-
    select(X, Xs, Ys),
    permutacja(Ys, Zs).
```

Wyjmowanie głowy permutacji z danej listy.

```
?- permutacja([1, 2, 3], X).
X = [1, 2, 3] ;
X = [1, 3, 2] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [3, 1, 2] ;
X = [3, 2, 1] ;
false.
```

Kolejność leksykograficzna.

```
permutacja([], []).
permutacja([X | Xs], Zs) :-
    permutacja(Xs, Ys),
    select(X, Zs, Ys).
```

Wkładanie głowy danej listy do permutacji.

```
?- permutacja([1, 2, 3], X).
X = [1, 2, 3] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [1, 3, 2] ;
X = [3, 1, 2] ;
X = [3, 2, 1] ;
false.
```

Inna kolejność.

Negacja

\+ Warunek

Uwaga: koniunkcja warunków zawierających negację nie jest przemienna.

```
?- member(X, [1, 2]), \+ member(X, [2, 3]).
X = 1 ;
false.
?- \+ member(X, [2, 3]), member(X, [1, 2]).
false.
```

Zagadka z książki Smullyana pod tytułem „Jaki jest tytuł tej książki?”:
Na wyspie żyją rycerze, którzy zawsze mówią prawdę, i łotrzy, którzy zawsze kłamią. Spotykamy trzech jej mieszkańców **A**, **B** i **C**. Pytamy się **A** kim on jest. Ten odpowiada ale nie zrozumieliśmy odpowiedzi. Pytamy się więc pozostałych co powiedział **A**. **B** odpowiada, że **A** powiedział, że jest łotrem. Na co **C** mówi by nie wierzyć **B**, bo **B** jest łotrem.
Kim są **B** i **C**?

```
rycerz(rycerz).
łotr(łotr).
powiedział(rycerz, X) :- X.
powiedział(łotr, X) :- \+ X.

?- powiedział(B, powiedział(A, łotr(A))),
   powiedział(C, łotr(B)).
B = łotr, C = rycerz
```

Implikacja

```
(G1 -> G2; G3)
```

Jeśli cel G1 jest spełniony, to sprawdź cel G2. W przeciwnym przypadku sprawdź cel G3.

```
max(X, Y, Z) :-
    (   X > Y
    ->  Z = X
    ;   Z = Y).
```

Arytmetyka

```
?- X = 2 + 2.
X = 2 + 2.    % !?!
?- X is 2 + 2.
X = 4.
?- 2 +2 =:= 2 * 2.
true.
```

operator	znaczenie
+	suma
-	różnica
*	iloczyn
/	iloraz
mod	modulo
div	dzielenie całkowite
rem	reszta z dzielenia

relacja	znaczenie
=:=	równe
=\=	różne
<	mniejsze
>	większe
=<	mniejsze lub równe
>=	większe lub równe

Aksjomatyka Peano

```
nat(0).
nat(X) :- nat(Y), X is Y+1.
```

Zero jest liczbą naturalną.

Następnik liczby naturalnej jest liczbą naturalną.

```
?- nat(X).
X = 0 ;
X = 1 ;
X = 2 ;
X = 3 ;
… % nieskończenie wiele odpowiedzi
```

Ciąg Fibonacciego

```
fib(0, 0).
fib(1, 1).
fib(N, F) :-
    N > 1, N1 is N-1, N2 is N-2,
    fib(N1, F1), fib(N2, F2),
    F is F1+F2.
```

```
?- time(fib(29, X)).
% 3,328,157 inferences, 71.688 CPU in 76.628 seconds (94% CPU, 46425 Lips)
X = 514229
```

Tablicowanie wyników

```
:- dynamic fib/2.

fib(0, 0).
fib(1, 1).
fib(N, F) :-
    N > 1, N1 is N-1, N2 is N-2,
    fib(N1, F1), fib(N2, F2),
    F is F1+F2,
    asserta(fib(N, F)).
```

Dopisanie faktu na początku definicji.

```
?- time(fib(29, X)).
% 141 inferences, 0.000 CPU in 0.000 seconds (83% CPU, 2350000 Lips)
X = 514229 .

?- time(fib(29, X)).
% 0 inferences, 0.000 CPU in 0.000 seconds (55% CPU, 0 Lips)
X = 514229 .
```

Generowanie wyrazów ciągu Fibonacciego

```
fib(0).
fib(1).
fib(X) :-
    fib(0, 1, X).

fib(A, B, X) :-
    S is A + B,
    (   X = S
    ;   fib(B, S, X)).

?- fib(X).
X = 0 ;
X = 1 ;
X = 1 ;
X = 2 ;
X = 3 ; % nieskończenie wiele odpowiedzi
```

Zamrażanie celów (korutyny)

```
?- X is 2, X > 1.
X = 2.
?- X > 1, X is 2.
ERROR…
```

Koniunkcja warunków zawierających operacje lub relacje arytmetyczne nie jest przemienna.

Należy odroczyć (zamrozić) sprawdzenie warunku do momentu kiedy zmienna będzie miała ustaloną wartość.

```
?- freeze(X, X > 1), X is 2.
X = 2.
```

freeze(Zmienna, Cel) odracza sprawdzenie celu aż zmienna przyjmie wartość.

Zamrożenie ze względu na wszystkie zmienne X_1, …, X_n:

```
freeze(X1, freeze(X2, freeze(X3, …)))
```

Aby zamrozić ze względu na co najmniej jedną ze zmiennych X_1, …, X_n trzeba skorzystać z predykatu **when**:

```
when((nonvar(X1); nonvar(X2); …; nonvar(Xn)), Cel)
```

when(Warunek, Cel) zamraża sprawdzenie celu aż warunek zostanie spełniony.

Strumień danych organizujemy, podobnie jak w języku Oz, w postaci list otwartych:

_ pusty strumień, w którym nie pojawiła się jeszcze żadna wartość;

[1 | _] strumień, w którym pojawiła się liczba 1 ale nie ma na razie kolejnych danych;

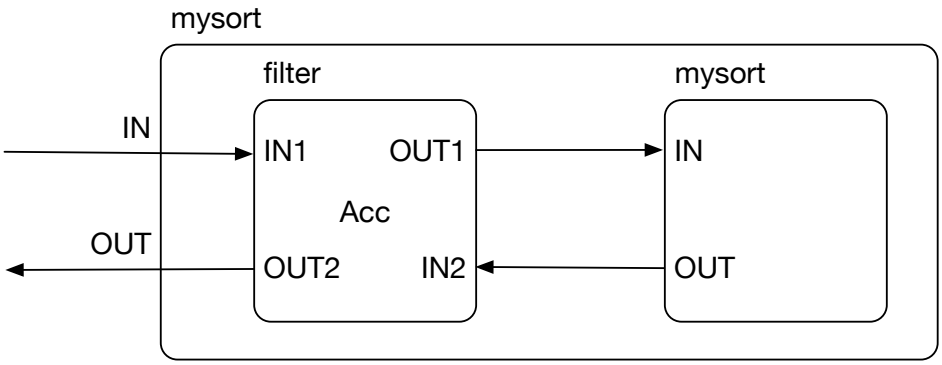
[] strumień zamknięty, w którym nie pojawi się już żadna dana;

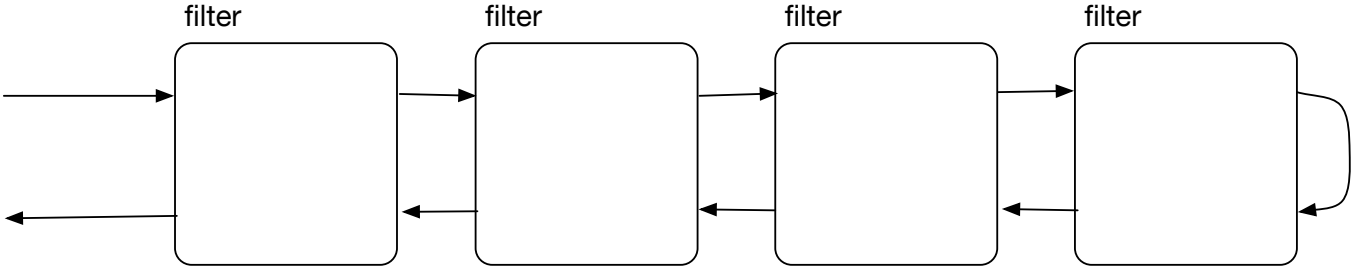
[1, 2, 3] strumień zamknięty, w którym pojawiły się trzy liczby.

```
% strumienie.pl

main(N) :-
    drukowanie(S1),
    podwajanie(S2, S1),
    numlist(1, N, S2). % generowanie liczb od 1 do N

podwajanie(IN, OUT) :-
    freeze(IN, % czekaj na liczbę w strumieniu IN
           (   IN = [H | IN_] % jeśli nowa liczba H
           ->  H2 is 2 * H,
               OUT = [H2 | OUT_], % wyślij podwojoną
               podwajanie(IN_, OUT_)
           ;   OUT = [])).

drukowanie(IN) :-
    freeze(IN,
           (   IN = [H | IN_]
           ->  writeln(H),
               drukowanie(IN_)
           ;   true)).
```

```
?- main(10).
2
4
6
8
10
12
14
16
18
20
?-
```

ZERO-TIME SORTING NETWORK





```
% mysort.pl

mysort(IN, OUT) :-
    freeze(IN,
           (   IN = [H | IN_]
           ->  filter(H, IN_, OUT1, IN2, OUT),
               mysort(OUT1, IN2)
           ;   OUT = [])).
```

```
filter(Acc, IN1, OUT1, IN2, OUT2) :-
    freeze(IN1,
           (   IN1 = [H | IN1_]
           ->  (   H >= Acc
               ->  OUT1 = [H | OUT1_],
                   filter(Acc, IN1_, OUT1_, IN2, OUT2)
               ;   OUT1 = [Acc | OUT1_],
                   filter(H, IN1_, OUT1_, IN2, OUT2))
           ;   OUT1 = [],
               OUT2 = [Acc | IN2])).
```

```
?- mysort(X, Y).
freeze(X, …).

?- mysort(X, Y), X = [2 | A].
X = [2|A],
freeze(A,…),
freeze(_1796, …).

?- mysort(X, Y), X = [2 | A], A = [4 | B].
X = [2, 4|B],
A = [4|B],
freeze(B, …),
freeze(_542, …),
freeze(_668, …).



?- mysort(X, Y), X = [2 | A], A = [4 | B],
   B = [1 | C].
X = [2, 4, 1|C],
A = [4, 1|C],
B = [1|C],
freeze(C,),
freeze(_1292, …),
freeze(_1436, …),
freeze(_1580, …).

?- mysort(X, Y), X = [2 | A], A = [4 | B],
   B = [1 | C], C = [].
X = [2, 4, 1],
Y = [1, 2, 4],
A = [4, 1],
B = [1],
C = [].
```

Programowanie ograniczeń

W Prologu jest możliwość narzucania ograniczeń na zmienne nie związane jeszcze z wartością. W tym celu można użyć pakietu **clpfd** (ang. *Constraint Logic Programming over Finite Domains*).

```
?- use_module(library(clpfd)).
true.

?- X #> 1, X #= 2.
X = 2.

?- X #>2, X #< 6, X #\= 4.
X in 3\/5.
```

X in 1..10	zdefiniowanie dziedziny
[X1, X2, …, Xn] ins 1..10	zdefiniowanie dziedzin
#=, **#\=**, **#<**, **#>**, **#=<**, **#>=**	relacje między wartościami

indomain(X)
label([X1, X2, …, Xn])
labeling([Opcja, …], [X1 X2, …, Xn])
Możliwe opcje:

- wybór zmiennej
  leftmost (domyślna), **ff**, **ffc**, **min**, **max**
- wybór wartości
  up (domyślna), **down**
- strategie podziału
  step (domyślna), **enum**, **bisect**
- kolejność rozwiązań
  min(Wyrażenie), **max(Wyrażenie)**

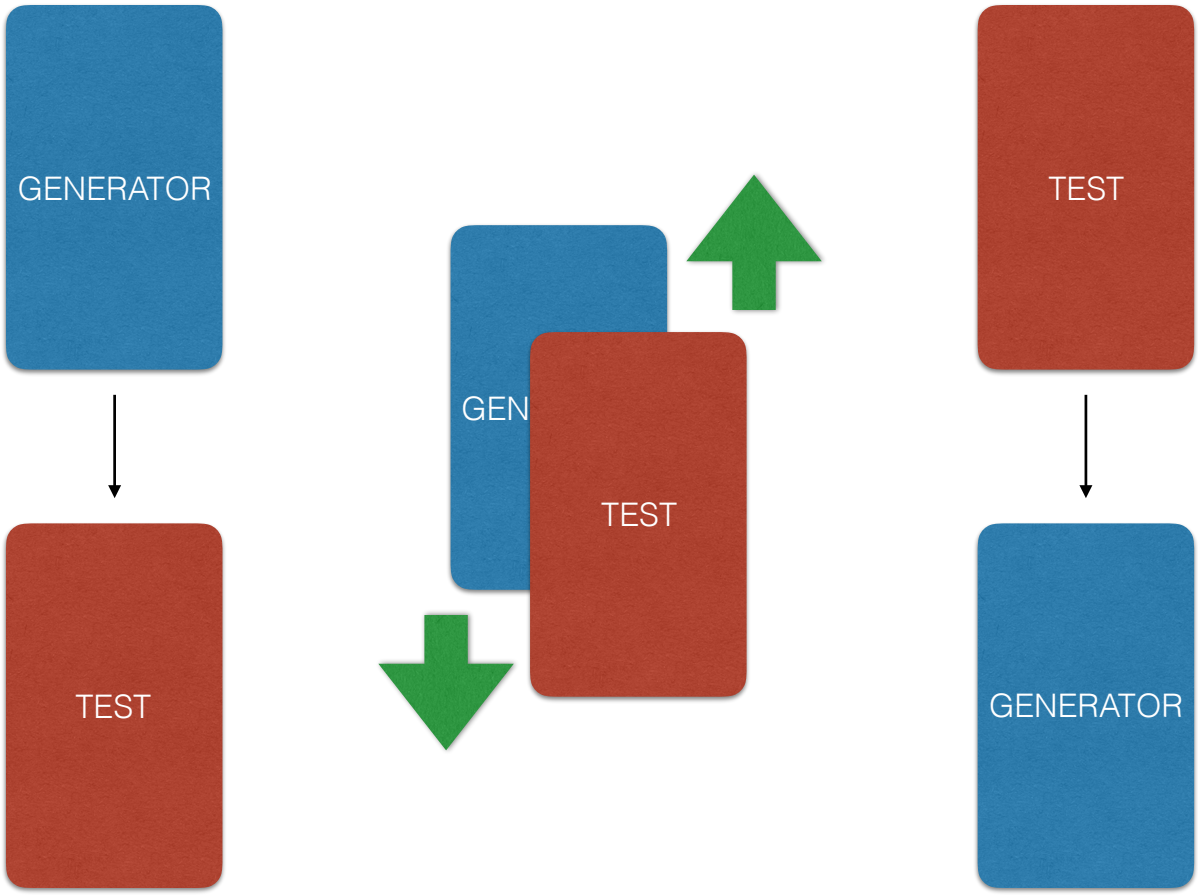
W Prologu podstawową metodą poszukiwania rozwiązania jest „generowanie i testowanie”.

?- GENERATOR, TEST.

Warunek **GENERATOR** dostarcza kolejne rozwiązanie a **TEST** sprawdza czy jest ono dopuszczalne.

W przypadku niepowodzenia następuje nawrót i wycofanie się do generowania kolejnego rozwiązania.

```
hetmany_gt(N, P) :-
    numlist(1, N, X),
    permutation(X, P),   % GENERATOR
    \+ niebezpieczna(P). % TEST

niebezpieczna(P) :-
    append(_, [I | L1], P),
    append(L2, [J | _], L1),
    length(L2, K),
    abs(I - J) =:= K + 1.

?- time(hetmany_gt(8, X)).
% 93,593 inferences, 0.036 CPU in 0.046 seconds (78% CPU, 2592173 Lips)
X = [1, 5, 8, 6, 3, 7, 2, 4] .
?- time(hetmany_gt(10, X)).
% 1,366,508 inferences, 0.488 CPU in 0.948 seconds (51% CPU, 2799648 Lips)
X = [1, 3, 6, 8, 10, 5, 9, 2, 4|...] .
?- time(hetmany_gt(12, X)).
% 90,688,062 inferences, 26.720 CPU in 39.410 seconds (68% CPU, 3393964 Lips)
X = [1, 3, 5, 8, 10, 12, 6, 11, 2|...] .
?- time(hetmany_gt(14, X)).
^CAction (h for help) ? abort
% 284,900,718 inferences, 81.443 CPU in 131.441 seconds (62% CPU, 3498158 Lips)
% Execution Aborted
```




```
:- use_module(library(clpfd)). % dyrektywa dla kompilatora

hetmany(N, P) :-
    length(P, N),
    P ins 1..N,
    bezpieczna(P),        % TEST
    labeling([ffc], P).   % GENERATOR

bezpieczna([]).
bezpieczna([I | L]) :-
    bezpieczna(L, I, 1),
    bezpieczna(L).

bezpieczna([], _, _).
bezpieczna([J | L], I, K) :-
    I #\= J,
    abs(I - J) #\= K,
    K1 is K + 1,
    bezpieczna(L, I, K1).
```

```
?- time(hetmany(10, X)).
% 42,161 inferences, 0.006 CPU in 0.007 seconds (89% CPU, 6709262 Lips)
X = [1, 3, 6, 9, 7, 10, 4, 2, 5|...] .

?- time(hetmany(20, X)).
% 230,209 inferences, 0.033 CPU in 0.036 seconds (91% CPU, 6941324 Lips)
X = [1, 3, 5, 14, 17, 4, 16, 7, 12|...] .

?- time(hetmany(40, X)).
% 562,761 inferences, 0.080 CPU in 0.084 seconds (95% CPU, 7051171 Lips)
X = [1, 3, 5, 26, 33, 4, 28, 7, 34|...] .

?- time(hetmany(80, X)).
% 2,158,637 inferences, 0.318 CPU in 0.334 seconds (95% CPU, 6784838 Lips)
X = [1, 3, 5, 44, 42, 4, 50, 7, 68|...] .

?- time(hetmany(160, X)).
% 10,400,403 inferences, 1.619 CPU in 1.680 seconds (96% CPU, 6423880 Lips)
X = [1, 3, 5, 65, 68, 4, 74, 7, 85|...] .
```

all_different([X1, …, Xn])
all_distinct([X1, …, Xn])

sum([X1, X2, …, Xn], Rel, Expr)
scalar_product([C1, …, Cn], [X1, …, Xn], Rel, Expr)

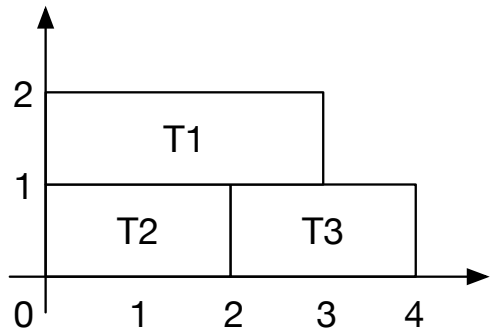
serialized([S1, …, Sn], [D1, …, Dn])

rozłączność przedziałów (Si, Si+Di)
Si + Di #=< Sj lub Sj + Dj #=< Si

cumulative([T1, …, Tn], [limit(L)])

zadania T1, …, Tn
Ti = task(Si, Di, Ei, Ci, IDi)
w żadnej chwili nie przekroczono L jednostek zasobu

```
tasks_starts(Tasks, [S1, S2, S3]) :-
    Tasks = [task(S1, 3, _, 1, _),
             task(S2, 2, _, 1, _),
             task(S3, 2, _, 1, _)].

?- tasks_starts(Tasks, Starts),
   Starts ins 0..10,
   cumulative(Tasks, [limit(2)]),
   label(Starts).
Tasks = [task(0, 3, 3, 1, _4380), task(0, 2,
2, 1, _4398), task(2, 2, 4, 1, _4416)],
Starts = [0, 0, 2] .
```



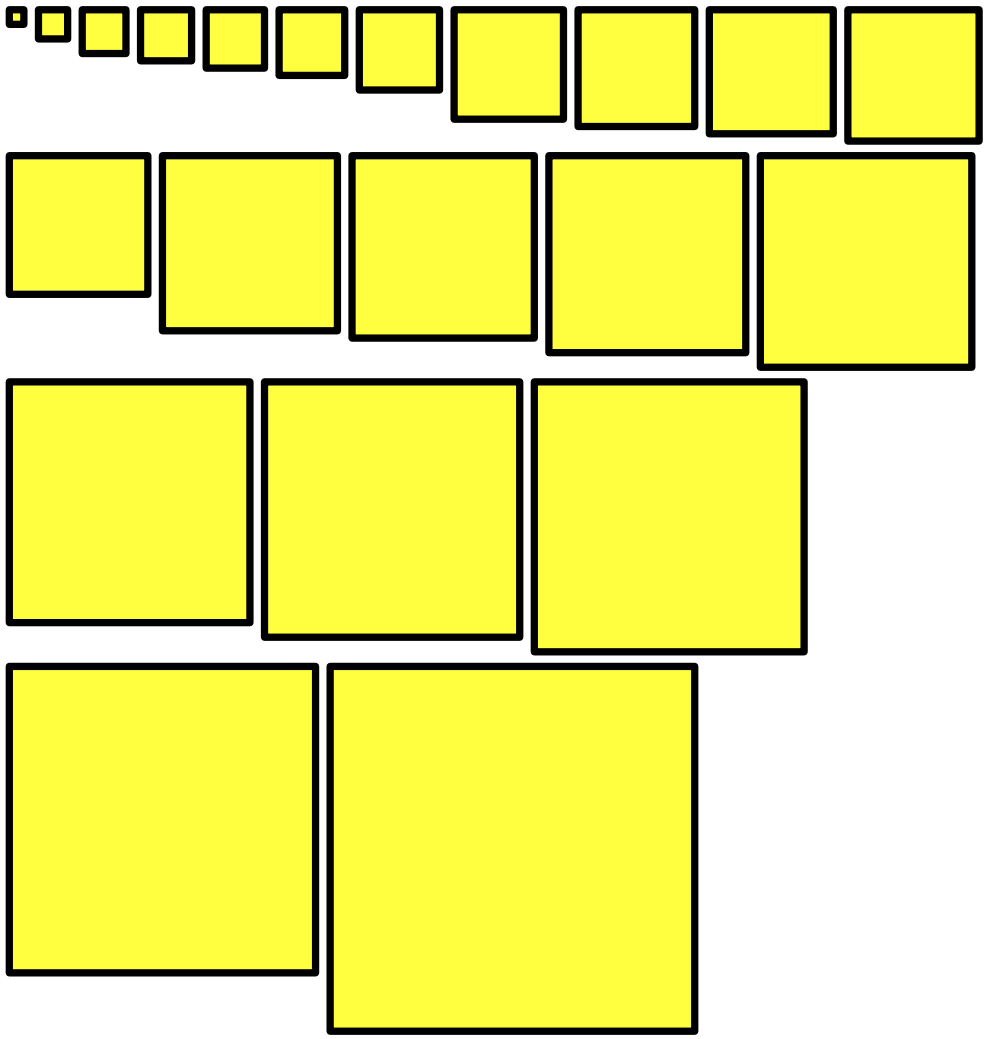
Podział kwadratu na kwadraty

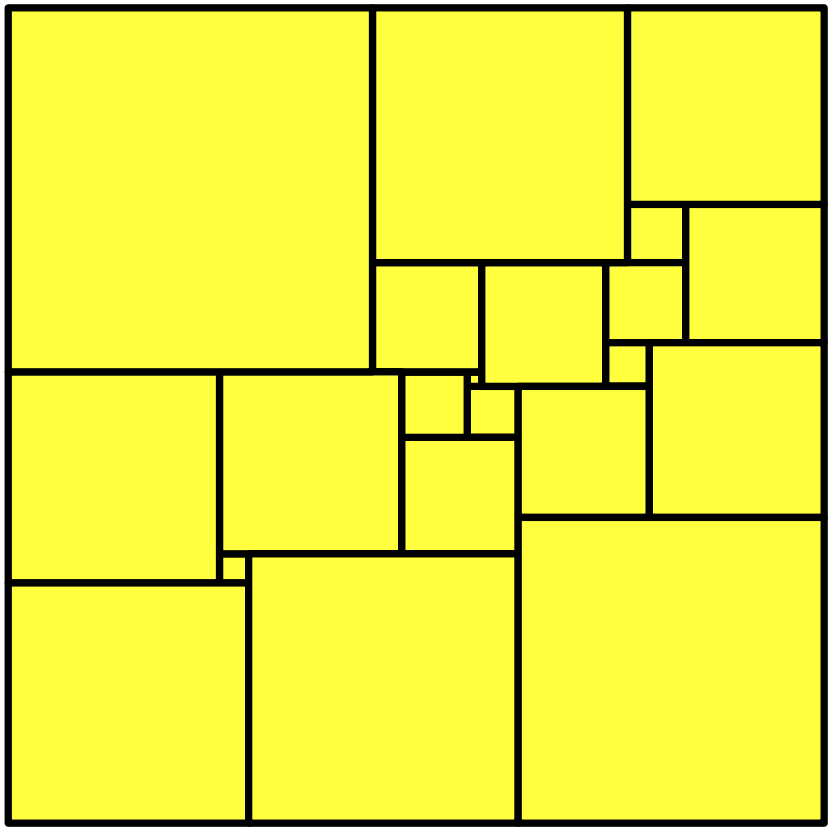
Zadanie: podzielić kwadrat na parami różne kwadraty.

Trywialne rozwiązanie:
jeden kwadrat

Najmniejsze nietrywialne rozwiązanie:
kwadrat 112x112 podzielony na 21 kwadratów o bokach 50, 42, 37, 35, 33, 29, 27, 25, 24, 19, 18, 17, 16, 15, 11, 9, 8, 7, 6, 4, 2

```
% kwadrat.pl

:- use_module(library(clpfd)).

main(Xs) :-
    kwadraty(112,
             [50,42,37,35,33,29,27,25,24,19,18,17,16,15,11,9,8,7,6,4,2],
             Xs).

kwadraty(D, Ds, Xs) :-
    length(Ds, N),
    D1 is D-1,
    length(Xs, N),
    Xs ins 0..D1,
    length(Ys, N),
    Ys ins 1..D,
    zadania(Xs, Ys, Ds, Zadania),
    cumulative(Zadania, [limit(D)]),
    % SICStus Prolog z opcją global(true) w cumulative/2 8500ms
    labeling([ffc], Xs).

zadania([], [], [], []).
zadania([X | L1], [Y | L2], [D | L3], [task(X, D, _, D, _) | L4]) :-
    Y #= X + D,
    zadania(L1, L2, L3, L4).
```





Rozwiązanie znalezione SICStus Prologiem.