MiniZinc

Technologie programistyczne, 2018-2019

http://www.minizinc.org

Co to jest MiniZinc?

- Język do opisów problemów decyzyjnych i optymalizacyjnych z ograniczeniami
- Zmienne decyzyjne liczbowe (`int`, `float`), logiczne (`bool`) oraz wyliczeniowe (`enum`)
- Możliwość współpracy z wieloma specjalizowanymi solverami, także implementacje referencyjne
- Rozwijany przez Monash University, Data61 Data Science i University of Melbourne (Australia)

MiniZinc i FlatZinc

```
int: nc = 3;

var 1..nc: wa;
var 1..nc: nt;
var 1..nc: sa;
var 1..nc: q;
var 1..nc: nsw;
var 1..nc: v;
var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;

solve satisfy;
```

MiniZinc: *model*

```
array [1..2] of int: X_INTRODUCED_0 = [1,-1];
var 1..3: wa:: output_var;
var 1..3: nt:: output_var;
var 1..3: sa:: output_var;
var 1..3: q:: output_var;
var 1..3: nsw:: output_var;
var 1..3: v:: output_var;
var 1..3: t:: output_var;
constraint int_lin_ne(X_INTRODUCED_0,[wa,nt],0);
constraint int_lin_ne(X_INTRODUCED_0,[wa,sa],0);
constraint int_lin_ne(X_INTRODUCED_0,[nt,sa],0);
constraint int_lin_ne(X_INTRODUCED_0,[nt,q],0);
constraint int_lin_ne(X_INTRODUCED_0,[sa,q],0);
constraint int_lin_ne(X_INTRODUCED_0,[sa,nsw],0);
constraint int_lin_ne(X_INTRODUCED_0,[sa,v],0);
constraint int_lin_ne(X_INTRODUCED_0,[q,nsw],0);
constraint int_lin_ne(X_INTRODUCED_0,[nsw,v],0);
solve  satisfy;
```

FlatZinc: *model*

Solver

MiniZinc: *dane*

Problem 1: kolorowanie mapy


```
int: nc = 3;

var 1..nc: wa;
var 1..nc: nt;
var 1..nc: sa;
var 1..nc: q;
var 1..nc: nsw;
var 1..nc: v;
var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;

solve satisfy;

output ["wa=\(wa) nt=\(nt) sa=\(sa) q=\(q) nsw=\(nsw) v=\(v) t=\(t)"];
```

parametr – jednokrotne przypisanie wartości
typy parametrów: `int`, `float`, `bool`, `string`, także tablice i zbiory

zmienne decyzyjne – ustalane przez solver (mogą mieć przypisaną wartość początkową), dziedziny takie same, jak dla parametrów, możliwość definiowania przedziałów

ograniczenia, jakie muszą spełniać zmienne decyzyjne (rozwiązanie), dostępne typowe operatory relacyjne oraz bardziej złożone predykaty

wyświetlenie listy z wynikami (tylko jedno polecenie w programie), możliwość prostego formatowania (funkcje `show`, `show_int` oraz `show_float`), sklejanie łańcuchów przez `++`

Problem 1: kolorowanie mapy


```
Compiling aust.mzn
Running aust.mzn
wa=3 nt=2 sa=1 q=3 nsw=2 v=3 t=1
----------
wa=3 nt=2 sa=1 q=3 nsw=2 v=3 t=2
----------
wa=3 nt=2 sa=1 q=3 nsw=2 v=3 t=3
----------
wa=2 nt=3 sa=1 q=2 nsw=3 v=2 t=1
----------
wa=2 nt=3 sa=1 q=2 nsw=3 v=2 t=2
----------
wa=2 nt=3 sa=1 q=2 nsw=3 v=2 t=3
----------
wa=3 nt=1 sa=2 q=3 nsw=1 v=3 t=1
----------
wa=3 nt=1 sa=2 q=3 nsw=1 v=3 t=2
----------
wa=3 nt=1 sa=2 q=3 nsw=1 v=3 t=3
----------
wa=1 nt=3 sa=2 q=1 nsw=3 v=1 t=1
----------
wa=1 nt=3 sa=2 q=1 nsw=3 v=1 t=2
----------
wa=1 nt=3 sa=2 q=1 nsw=3 v=1 t=3
----------
wa=2 nt=1 sa=3 q=2 nsw=1 v=2 t=1
----------
wa=2 nt=1 sa=3 q=2 nsw=1 v=2 t=2
----------
wa=2 nt=1 sa=3 q=2 nsw=1 v=2 t=3
----------
wa=1 nt=2 sa=3 q=1 nsw=2 v=1 t=1
----------
wa=1 nt=2 sa=3 q=1 nsw=2 v=1 t=2
----------
wa=1 nt=2 sa=3 q=1 nsw=2 v=1 t=3
----------
==========
Finished in 155msec
```

wykorzystanie solvera *finite domain* do znalezienia rozwiązań

Problem 2: planowanie wypieków

ograniczenia na zasoby:
mąka, banany, cukier, masło, kakao

```
1 var 0..100: b;
2 var 0..100: c;
3
4 constraint 250*b + 200*c <= 4000;
5 constraint 2*b <= 6;
6 constraint 75*b + 150*c <= 2000;
7 constraint 100*b + 150*c <= 500;
8 constraint 75*c <= 500;
9
10 solve maximize 400*b + 450*c;
11
12 output ["# of banana cakes=\(b) # of chocolate cakes=\(c)"];
```

definicja funkcji celu: typowe operatory arytmetyczne, `div` i `mod`, typowe funkcje matematyczne

```
Compiling cakes.mzn
Running cakes.mzn
# of banana cakes=2 # of chocolate cakes=2
----------
==========
Finished in 70msec
```

Zbiory danych i asercje

- Możliwość zewnętrznego ustawienia wartości parametrów oraz zmiennych decyzyjnych
- Zewnętrzny plik danych (`.dzn`) albo linia poleceń (opcja `-D`)
- Asercje służące do sprawdzania poprawności danych wejściowych (także do ustalania wartości)

Problem 2: planowanie wypieków (wydzielone dane)

```
flour = 4000;
banana = 6;
sugar = 2000;
butter = 500;
cocoa = 500;
```

polecenie `output` można opuścić → wyjście w formacie zgodnym z formatem pliku danych

```
flour = 8000;
banana = 11;
sugar = 3000;
butter = 1500;
cocoa = 800;
```

```
int: flour;
int: banana;
int: sugar;
int: butter;
int: cocoa;

constraint assert(flour >= 0,   "Amount of flour should be non-negative");
constraint assert(banana >= 0,  "Amount of banana should be non-negative");
constraint assert(sugar >= 0,   "Amount of sugar should be non-negative");
constraint assert(butter >= 0,  "Amount of butter should be non-negative");
constraint assert(cocoa >= 0,   "Amount of cocoa should be non-negative");

var 0..100: b;
var 0..100: c;

constraint 250*b + 200*c <= flour;
constraint 2*b <= banana;
constraint 75*b + 150*c <= sugar;
constraint 100*b + 150*c <= butter;
constraint 75*c <= cocoa;

solve maximize 400*b + 450*c;

output ["# of banana cakes=\(b)\n# of chocolate cakes=\(c)"];
```

```
Compiling cakes2.mzn, with additional data pantry1.dzn
Running cakes2.mzn
# of banana cakes=2
# of chocolate cakes=2
----------
==========
Finished in 104msec
```

```
Compiling cakes2.mzn, with additional data pantry2.dzn
Running cakes2.mzn
# of banana cakes=3
# of chocolate cakes=8
----------
==========
Finished in 103msec
```

Problem 3: spłata pożyczki


```
1  var float: R;
2  var float: P;
3  var 0.0..5.0: I;
4
5  var float: B1;
6  var float: B2;
7  var float: B3;
8  var float: B4;
9
10 constraint B1 = P*(1.0 + I/4) - R;
11 constraint B2 = B1*(1.0 + I/4) - R;
12 constraint B3 = B2*(1.0 + I/4) - R;
13 constraint B4 = B3*(1.0 + I/4) - R;
14
15 solve satisfy;
16
17 output [
18   "Borrowing \(0, 2, P) at \(I*100.0)% interest\n",
19   "and repaying \(0, 2, R) per quarter leaves \(0, 2, B4) owing"
20 ];
```

parametry i zmienne decyzyjne typu `float` – możliwość ograniczenia wartości do przedziału

możliwość formatowania zmiennych decyzyjnych osadzonych w łańcuchu znakowym (takie samo, jak przy użyciu funkcji `show_...`)

Problem 3: spłata pożyczki

```
I = 0.16;
P = 1000.0;
R = 260.0;
```

```
I = 0.16;
P = 1000.0;
B4 = 0.0;
```

```
I = 0.16;
R = 250.0;
B4 = 0.0;
```

wykorzystanie solvera *mixed integer programming* do znalezienia rozwiązania

```
solve satisfy;

output [
  "Borrowing \(0, 2, P) at \(I*100.0)% interest\n",
  "and repaying \(0, 2, R) per quarter leaves \(0, 2, B4) owing"
];
```

```
Compiling loan.mzn with data loan1.dzn
Running loan.mzn
Borrowing 1000.00 at 16.0% interest
and repaying 260.00 per quarter leaves 65.78 owing
----------
Finished in 168msec
```

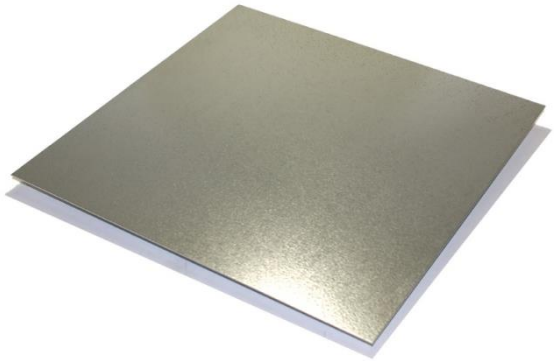
```
Compiling loan.mzn with data loan2.dzn
Running loan.mzn
Borrowing 1000.00 at 16.0% interest
and repaying 275.49 per quarter leaves 0.00 owing
----------
Finished in 153msec
```

```
Compiling loan.mzn with data loan3.dzn
Running loan.mzn
Borrowing 907.47 at 16.0% interest
and repaying 250.00 per quarter leaves 0.00 owing
----------
Finished in 153msec
```

Tablice i zbiory

- Tablice (wielowymiarowe) → złożone struktury z parametrami oraz zmiennymi decyzyjnymi
- Zbiory → definiowanie nowych typów danych, np. służących do indeksowania tablic
- Podstawowe operacje na zbiorach, np.: `in`, `subset`, `superset`, `union`, `intersect`, `diff`, `symdiff`, `card`

Problem 4: temperatura arkusza metalu



```
int: w = 4;
int: h = 4;

set of int: HEIGHT = 0..h;
set of int: IHEIGHT = 1..h-1;
set of int: WIDTH = 0..w;
set of int: IWIDTH = 1..w-1;

array[HEIGHT, WIDTH] of var float: t;
```

tablica zmiennych decyzyjnych

```
var float: left;
var float: right;
var float: top;
var float: bottom;

constraint forall(i in IHEIGHT, j in IWIDTH)
  (4.0*t[i, j] = t[i-1, j] + t[i, j-1] + t[i+1, j] + t[i, j+1]);

constraint forall(i in IHEIGHT)(t[i, 0] = left);
constraint forall(i in IHEIGHT)(t[i, w] = right);
constraint forall(j in IWIDTH)(t[0, j] = top);
constraint forall(j in IWIDTH)(t[h, j] = bottom);

constraint t[0,0] = 0.0;
constraint t[0,w] = 0.0;
constraint t[h,0] = 0.0;
constraint t[h,w] = 0.0;

left = 0.0;
right = 0.0;
top = 100.0;
bottom = 0.0;
```

metoda elementów skończonych, równanie Laplace'a

Problem 4: temperatura arkusza metalu

```
solve satisfy;

output [show_float(6, 2, t[i, j]) ++
  if j == w then "\n" else " " endif |
  i in HEIGHT, j in WIDTH
];
```

wykorzystanie solvera *mixed integer programming* do znalezienia rozwiązania

```
Compiling laplace.mzn
Running laplace.mzn
 -0.00 100.00 100.00 100.00  -0.00
 -0.00  42.86  52.68  42.86  -0.00
 -0.00  18.75  25.00  18.75  -0.00
 -0.00   7.14   9.82   7.14  -0.00
 -0.00  -0.00  -0.00  -0.00  -0.00
----------
Finished in 419msec
```

Problem 5: planowanie produkcji

- Rozszerzenie problemu wypieku ciasta na dowolną listę produktów oraz zasobów (dane wejściowe)
- Rozdzielenie deklaracji i definicji typu wyliczeniowego
- Wykorzystanie typów wyliczeniowych do indeksowania tablic

Problem 5: planowanie produkcji

```
enum Products;
array[Products] of int: profit;

enum Resources;
array[Resources] of int: capacity;

array[Products, Resources] of int: consumption;
constraint assert(forall (p in Products, r in Resources)
                  (consumption[p, r] >= 0), "Error: negative consumption");
```

tabele z parametrami, sprawdzenie poprawności danych wejściowych

```
int: mproducts = max (p in Products)
                     (min (r in Resources where consumption[p, r] > 0)
                          (capacity[r] div consumption[p, r]));
```

oszacowanie maksymalnej liczby wyprodukowanych produktów

```
array[Products] of var 0..mproducts: produced;
array[Resources] of var 0..max(capacity): used;

constraint forall (r in Resources) (
  used[r] = sum (p in Products)(consumption[p, r]*produced[p])
  /\ used[r] <= capacity[r]
);
```

tabele ze zmiennymi decyzyjnymi i ograniczenia narzucone na zmienne

```
solve maximize sum (p in Products) (profit[p]*produced[p]);

output
  ["\(p) = \(produced[p]);\n" | p in Products] ++
  ["\(r) = \(used[r]);\n" | r in Resources];
```

funkcja celu

Problem 5: planowanie produkcji

```
Products = {BananaCake, ChocolateCake};
profit = [400, 450];

Resources = {Flour, Banana, Sugar, Butter, Cocoa};
capacity = [4000, 6, 2000, 500, 500];
```

```
consumption = [| 250, 2, 75, 100, 0,
               | 200, 0, 150, 150, 75 |];
```

definicje typów wyliczeniowych oraz wartości parametrów

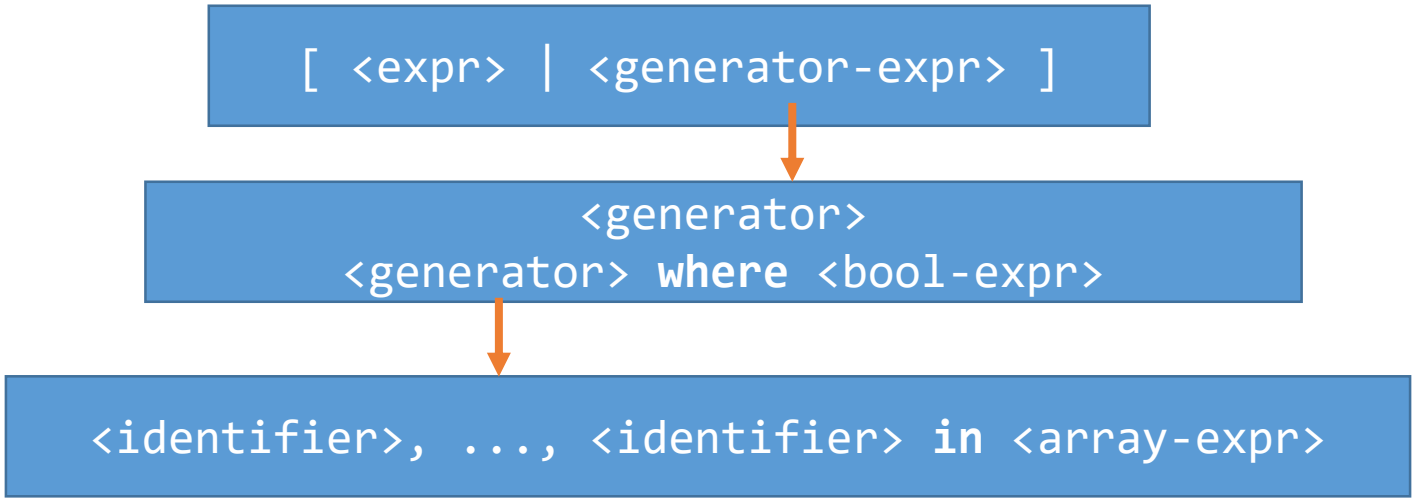
definicja tabeli dwuwymiarowej, także funkcje `array`*`nd`*`(indeksy, lista)`

```
solve maximize sum (p in Products) (profit[p]*produced[p]);

output
  ["\(p) = \(produced[p]);\n" | p in Products] ++
  ["\(r) = \(used[r]);\n" | r in Resources];
```

Sklejanie tabel jednowymiarowych (list) za pomocą operatora ++

```
Compiling simple-prod-planning.mzn, with additional data simple-prod-planning-data.dzn
Running simple-prod-planning.mzn
BananaCake = 2;
ChocolateCake = 2;
Flour = 900;
Banana = 4;
Sugar = 450;
Butter = 500;
Cocoa = 150;
----------
==========
Finished in 311msec
```

Definiowanie list i zbiorów (→ *List and set comprehension*)




```
[i + j | i,j in 1..3 where i < j]
```

```
[1+2, 1+3, 2+3] → [3, 4, 5]
```

Wywoływanie generatorów (→ *Generator call expressions*)

```
<agg-func> (<generator-expr>) (<expr>)
```

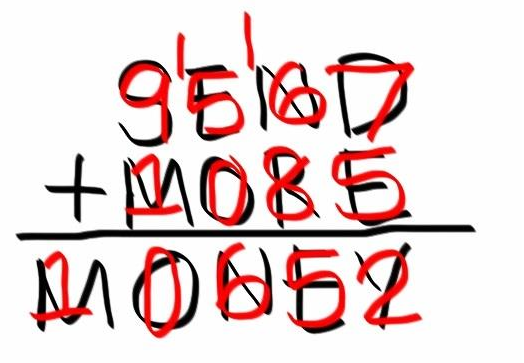
```
<agg-func> ([<expr> | <generator-expr>])
```

Funkcje agregujące

- tablice liczbowe: `sum`, `product`, `min`, `max`
- tablice logiczne: `forall`, `exists`, `xorall`, `iffall` (nieparzysta i parzysta liczba spełnionych warunków)

```
forall([a[i] != a[j] | i,j in 1..3 where i < j])
```

Problem 6: send more money...




```
include "alldifferent.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint          1000*S + 100*E + 10*N + D
                  + 1000*M + 100*O + 10*R + E
         =10000*M + 1000*O + 100*N + 10*E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   \(S)\(E)\(N)\(D)\n",
        "+  \(M)\(O)\(R)\(E)\n",
        "= \(M)\(O)\(N)\(E)\(Y)\n"];
```

predefiniowany predykat sprawdzający unikalność poszczególnych wartości

```
Compiling send-more-money.mzn
Running send-more-money.mzn
   9567
+  1085
= 10652
----------
Finished in 350msec
```

Wyrażenia warunkowe

```
if <bool-expr> then <expr-1> else <expr-2> endif
```

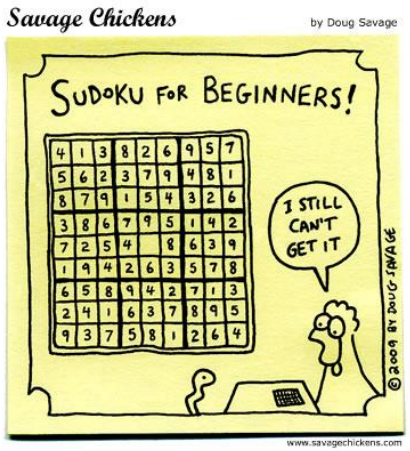
```
int: r = if y != 0 then x div y else 0 endif;
```

```
constraint forall(i,j in PuzzleRange) (
       if start[i,j] > 0
              then puzzle[i,j] = start[i,j]
              else true endif );
```

```
if j mod S == 0 then " " else "" endif
```

Problem 7: sudoku

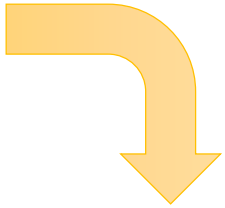
```
include "alldifferent.mzn";

int: S;
int: N = S*S;
int: digs = ceil(log(10.0, int2float(N)));

set of int: PuzzleRange = 1..N;
set of int: SubSquareRange = 1..S;

array[PuzzleRange, PuzzleRange] of 0..N: start;
array[PuzzleRange, PuzzleRange] of var PuzzleRange: puzzle;

constraint forall(i, j in PuzzleRange) (
  if start[i, j] > 0 then puzzle[i,j] = start[i, j] else true endif);

constraint forall (i in PuzzleRange)
                  (alldifferent( [puzzle[i, j] | j in PuzzleRange]));
constraint forall (j in PuzzleRange)
                  (alldifferent( [puzzle[i, j] | i in PuzzleRange]));

constraint forall (bi, bj in SubSquareRange)
                  (alldifferent([puzzle[(bi - 1)*S + i, (bj - 1)*S + j] | i, j in SubSquareRange]));

solve satisfy;

output [
  show_int(digs, puzzle[i, j]) ++ " " ++
  if j mod S == 0 then " " else "" endif ++
  if j == N /\ i != N then
    if i mod S == 0 then "\n\n" else "\n" endif
  else "" endif
  | i, j in PuzzleRange] ++ ["\n"];
```



Problem 7: sudoku

```
S=3;
start=[|
0, 0, 0, 0, 0, 0, 0, 0, 0|
0, 6, 8, 4, 0, 1, 0, 7, 0|
0, 0, 0, 0, 8, 5, 0, 3, 0|
0, 2, 6, 8, 0, 9, 0, 4, 0|
0, 0, 7, 0, 0, 0, 9, 0, 0|
0, 5, 0, 1, 0, 6, 3, 2, 0|
0, 4, 0, 6, 1, 0, 0, 0, 0|
0, 3, 0, 2, 0, 7, 6, 9, 0|
0, 0, 0, 0, 0, 0, 0, 0, 0|];
```

```

solve satisfy;

```



```
Compiling sudoku.mzn, with additional data sudoku.dzn
Running sudoku.mzn
5 9 3  7 6 2  8 1 4
2 6 8  4 3 1  5 7 9
7 1 4  9 8 5  2 3 6

3 2 6  8 5 9  1 4 7
1 8 7  3 2 4  9 6 5
4 5 9  1 7 6  3 2 8

9 4 2  6 1 8  7 5 3
8 3 5  2 4 7  6 9 1
6 7 1  5 9 3  4 8 2
----------
==========
Finished in 261msec
```

Typy wyliczeniowe

```
enum <enum-name> = { … } / anon_enum
    [var] <enum-name> : <var-name>
      [var] <l>..<u> : <var-name>
```

- Specjalizowane funkcje do obsługi `enum`
  - `enum_next(X, x)` – poprzednia wartość przed `x`
  - `enum_next(X, x)` – następna wartość po `x`
  - `to_enum(X, i)` – konwersja liczby `i` do wartości z `X`
- Standardowe funkcje stosowalne do `enum`
  - `card(X)` – liczba elementów w typie `X`
  - `min(X)` – pierwszy element w typie `X`
  - `max(X)` – ostatni element w typie `X`

Złożone ograniczenia

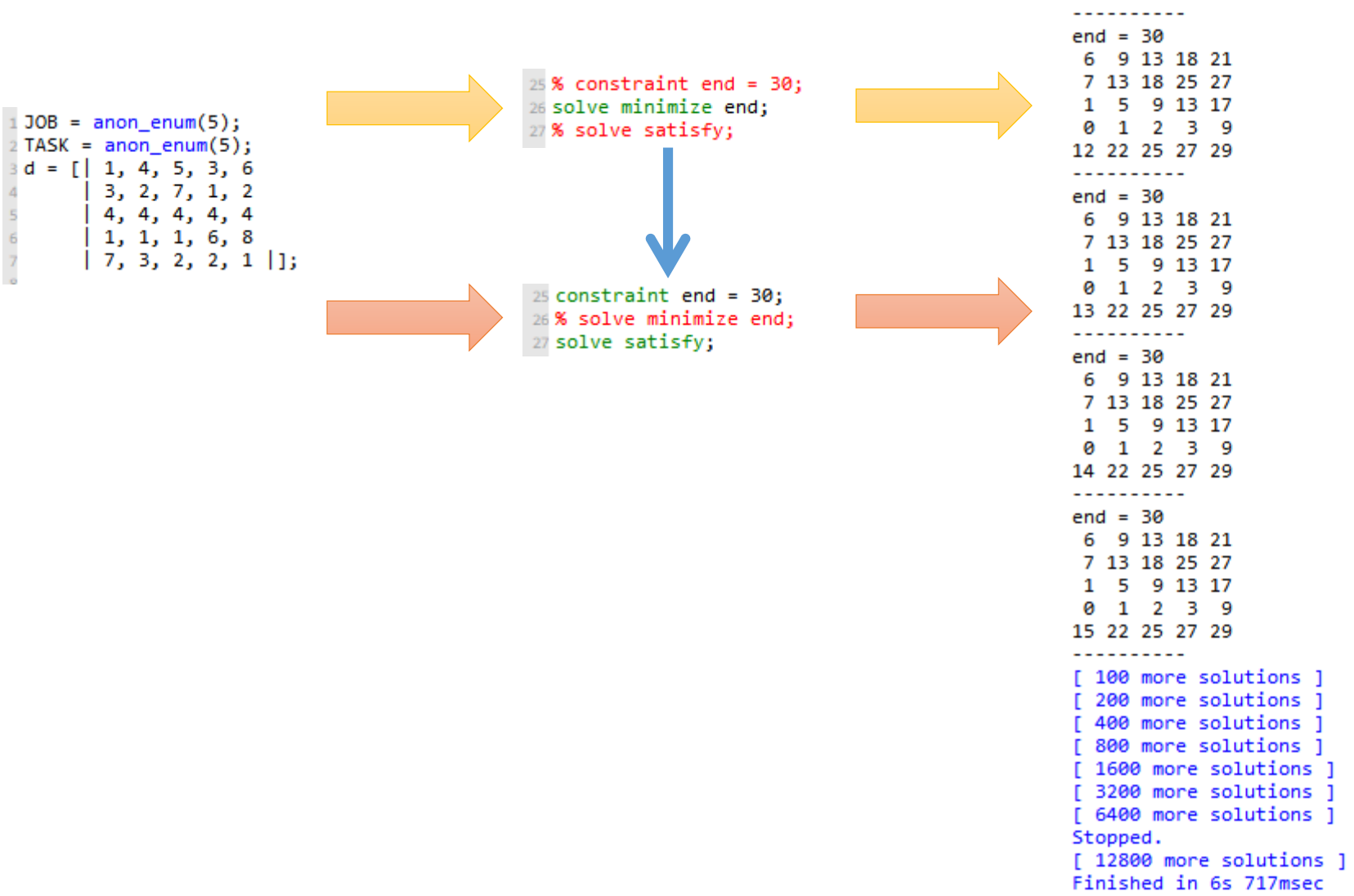
- Złożone wyrażenia logiczne w ograniczeniach
- Zastosowanie składni „matematycznej”
  - `true`, `false`
  - `not`
  - koniunkcja (`/\`) i dysjunkcja (`\/`)
  - *only-if* (`<-`), *implies* (`->`), if-and-only-if (`<->`)

Problem 8: warsztat (*jobshop*)

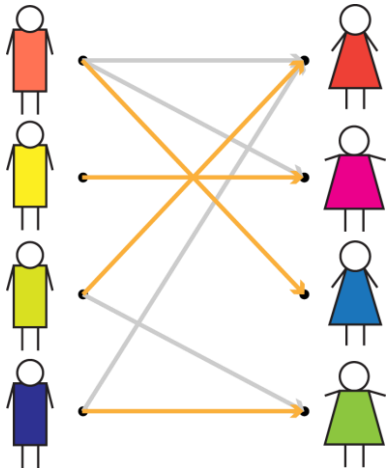
```
enum JOB;
enum TASK;
TASK: last = max(TASK);
array [JOB, TASK] of int: d;
int: total = sum(i in JOB, j in TASK)(d[i, j]);
int: digs = ceil(log(10.0, int2float(total)));
array [JOB, TASK] of var 0..total: s;
var 0..total: end;

constraint
  forall(i in JOB) (
    forall(j in TASK where j < last)
      (s[i,j] + d[i,j] <= s[i, enum_next(TASK, j)])
    /\ s[i,last] + d[i,last] <= end
  );

constraint
  forall(j in TASK) (
    forall(i, k in JOB where i < k) (
      s[i,j] + d[i,j] <= s[k,j] \/
      s[k,j] + d[k,j] <= s[i,j]
    )
  );

% constraint end = 30;
solve minimize end;
% solve satisfy;

output ["end = \(end)\n"] ++
  [show_int(digs, s[i,j]) ++ " " ++
   if j == last then "\n" else "" endif | i in JOB, j in TASK];
```

zastosowanie typów `enum` jako indeksów w tablicach (konwersja jednostronna)

Problem 8: warsztat (*jobshop*)



Problem 9: planowanie stabilnych małżeństw



```
int: n;

enum Men = anon_enum(n);
enum Women = anon_enum(n);

array[Women, Men] of int: rankWomen;
array[Men, Women] of int: rankMen;

array[Men] of var Women: wife;
array[Women] of var Men: husband;

constraint forall (m in Men) (husband[wife[m]]=m);
constraint forall (w in Women) (wife[husband[w]]=w);

constraint forall (m in Men, o in Women) (
  rankMen[m,o] < rankMen[m,wife[m]] -> rankWomen[o,husband[o]] < rankWomen[o,m]
);

constraint forall (w in Women, o in Men) (
  rankWomen[w, o] < rankWomen[w, husband[w]] -> rankMen[o, wife[o]] < rankMen[o, w]
);

solve satisfy;

output ["wives=\(wife)\nhusbands=\(husband)\n"];
```

wymuszenie „stabilności” z punktu widzenia rankingu (preferencji)

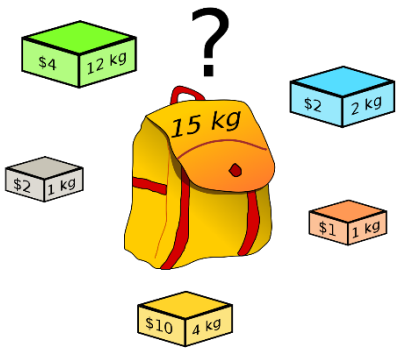
Problem 9: planowanie stabilnych małżeństw

```
n = 5;
rankWomen =
[| 1, 2, 4, 3, 5,
| 3, 5, 1, 2, 4,
| 5, 4, 2, 1, 3,
| 1, 3, 5, 4, 2,
| 4, 2, 3, 5, 1 |];
rankMen =
[| 5, 1, 2, 4, 3,
| 4, 1, 3, 2, 5,
| 5, 3, 2, 4, 1,
| 1, 5, 4, 3, 2,
| 4, 3, 2, 1, 5 |];
```

```
solve satisfy;

output ["wives=\(wife)\nhusbands=\(husband)\n"];
```

```
Compiling stable-marriage.mzn, with additional data stable-marriage.dzn
Running stable-marriage.mzn
wives=[Women_2, Women_1, Women_5, Women_3, Women_4]
husbands=[Men_2, Men_1, Men_4, Men_5, Men_3]
----------
wives=[Women_2, Women_3, Women_5, Women_1, Women_4]
husbands=[Men_4, Men_1, Men_2, Men_5, Men_3]
----------
wives=[Women_4, Women_1, Women_2, Women_3, Women_5]
husbands=[Men_2, Men_3, Men_4, Men_1, Men_5]
----------
==========
Finished in 212msec
```

Problem 10: problem plecakowy (*knapsack*)



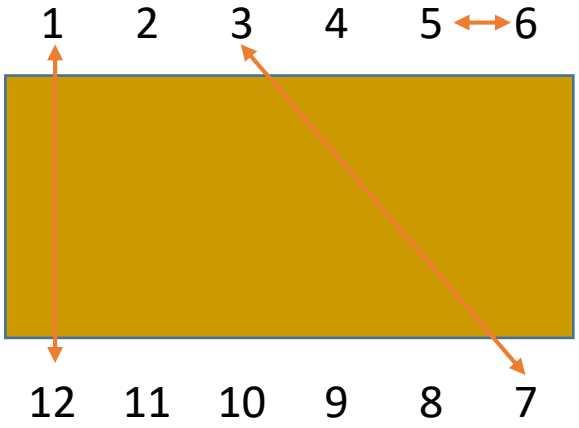

```
enum ITEM;
int: capacity;

array[ITEM] of int: profits;
array[ITEM] of int: weights;

var set of ITEM: knapsack;

constraint sum (i in knapsack) (weights[i]) <= capacity;

solve maximize sum (i in knapsack) (profits[i]);

output ["knapsack = \(knapsack)\n"];
```

zbiór jako złożona zmienna decyzyjna

```
capacity = 16;
ITEM = anon_enum(5);
profits = [5, 15, 2, 6, 1];
weights = [3, 10, 5, 3, 3];
```

```
> mzn-g12fd.bat knapsack.mzn knapsack.dzn
knapsack = {ITEM_1, ITEM_2, ITEM_4}
----------
==========
```

Problem 11: planowanie miejsc przy stole weselnym



- Stół prostokątny, 12-miejscowy
- Wymagania ogólne (np. państwo młodzi razem)
- Antypatie i fobie gości
- Maksymalizacja odległości pomiędzy „antypatiami”

Problem 11: planowanie miejsc przy stole weselnym

```
enum Guests = { bride, groom, bestman, bridesmaid, bob, carol, ted, alice, ron, rona, ed, clara};
set of int: Seats = 1..12;
set of int: Hatreds = 1..5;
array[Hatreds] of Guests: h1 = [groom, carol, ed, bride, ted];
array[Hatreds] of Guests: h2 = [clara, bestman, ted, alice, ron];
set of Guests: Males = {groom, bestman, bob, ted, ron, ed};
set of Guests: Females = {bride, bridesmaid, carol, alice, rona, clara};

array[Guests] of var Seats: pos;
array[Hatreds] of var Seats: p1;
array[Hatreds] of var Seats: p2;
array[Hatreds] of var 0..1: sameside;
array[Hatreds] of var Seats: distance;

include "alldifferent.mzn";

constraint alldifferent(pos);
constraint forall (g in Males) (pos[g] mod 2 == 1);
constraint forall (g in Females) (pos[g] mod 2 == 0);

constraint not (pos[ed] in {1, 6, 7, 12});

constraint abs(pos[bride] - pos[groom]) <= 1
    /\ (pos[bride] <= 6 <-> pos[groom] <= 6);

constraint forall(h in Hatreds) (
    p1[h] = pos[h1[h]] /\
    p2[h] = pos[h2[h]] /\
    sameside[h] = bool2int(p1[h] <= 6 <-> p2[h] <= 6) /\
    distance[h] = sameside[h] * abs(p1[h] - p2[h])
        + (1 - sameside[h])*(abs(13 - p1[h] - p2[h]) + 1));

var int: total_distance;

constraint total_distance = sum(h in Hatreds)(distance[h]);

solve maximize total_distance;

output ["total distance = \(total_distance)\n"] ++ [show(g) ++ " " | s in Seats, g in Guests where fix(pos[g]) == s] ++ ["\n"];
```

antypatie wśród gości

panie siedzą na miejscach parzystych, panowie na nieparzystych

Ed ma fobie – nie może siedzieć na miejscu z brzegu stołu

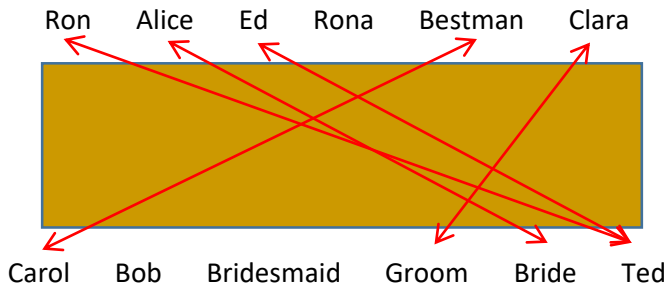
państwo młodzi muszą siedzieć obok siebie, po tej samej stronie stołu

maksymalizacja odległości pomiędzy gośćmi, którzy za sobą nie przepadają

Problem 11: planowanie miejsc przy stole weselnym

```
solve maximize total_distance;

output ["total distance = \(total_distance)\n"] ++
       [show(g) ++ " " | s in Seats, g in Guests where fix(pos[g]) == s] ++ ["\n"];
```

```
Compiling wedding.mzn
Running wedding.mzn
total distance = 18
ron clara ed alice bestman rona ted bride groom carol bob bridesmaid
----------
total distance = 20
ron alice ed rona bestman clara ted bride groom carol bob bridesmaid
----------
total distance = 22
ron alice ed rona bestman clara ted bride groom bridesmaid bob carol
----------
==========
Finished in 834msec
```



Definiowanie własnych predykatów

```
predicate <pred-name>(<arg-def>,…,<arg-def>) = <bool-expr>
```

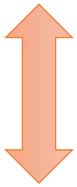
```
predicate lookup(array[int] of var int:x, int: i, var int: y) =
        assert(i in index_set(x),
                "index out of range in lookup"
                y = x[i]);
```

```
predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
        s1 + d1 <= s2 \/ s2 + d2 <= s1;
```

Definiowanie własnych predykatów

jobshop

```
16
17 constraint
18   forall(j in TASK) (
19     forall(i, k in JOB where i < k) (
20       s[i,j] + d[i,j] <= s[k,j] \/
21       s[k,j] + d[k,j] <= s[i,j]
22     )
23   );
24
```



```
17 predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
18   s1 + d1 <= s2 \/ s2 + d2 <= s1;
19
20 constraint
21   forall(j in TASK) (
22     forall(i, k in JOB where i < k) (
23       no_overlap(s[i,j], d[i,j], s[k, j], d[k,j])
24     )
25   );
```

Definiowanie własnych funkcji

```
function <ret-type> <func-name>(<arg-def>,…,<arg-def>) = <expr>
```

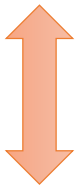
```
function var int: manhattan(
       var int: x1, var int: y1, var int: x2, var int: y2) =
              abs(x1 - x2) + abs(y1 - y2);
```

```
function int: posn(int: a, int: a1) = (a-1)*S + a1;
```

Definiowanie własnych funkcji

sudoku

```
20
21 constraint forall (bi, bj in SubSquareRange)
22                   (alldifferent([puzzle[(bi - 1)*S + i, (bj - 1)*S + j] | i, j in SubSquareRange]));
23
```



```
21 function int: posn(int: a, int: a1) = (a-1)*S + a1;
22
23 constraint forall (bi, bj in SubSquareRange)
24                   (alldifferent([puzzle[posn(bi, i), posn(bj, j)] | i, j in SubSquareRange]));
```

Zmienne lokalne

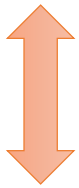
```
let {<var-def>, …, <var-def>} in <expr>
```

```
constraint let {var int: s = x1 + x2 + x3 + x4}
                in l <= s /\ s <= u;
```

Zmienne lokalne

wedding

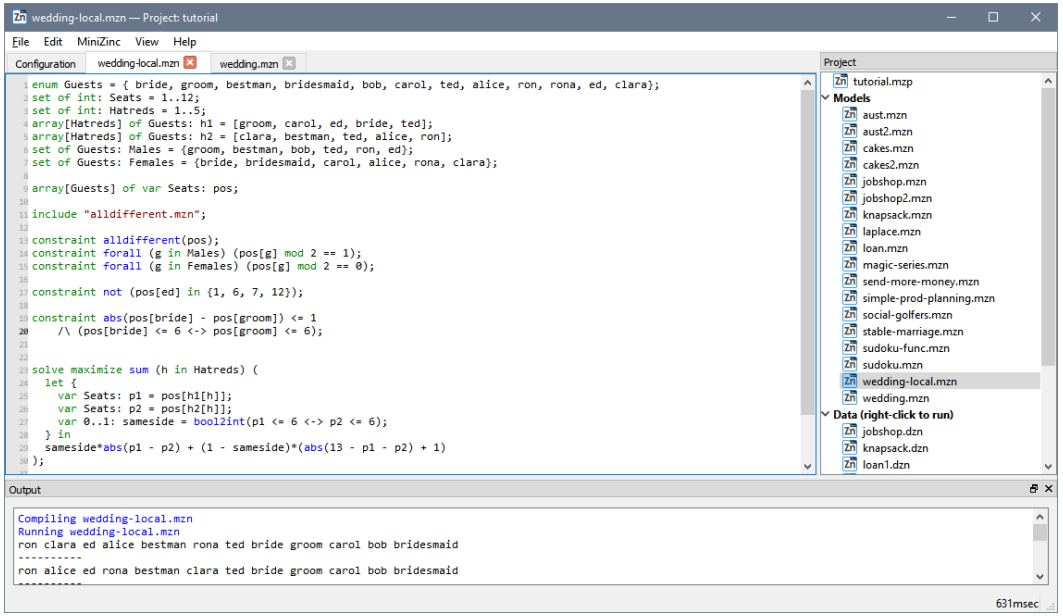
```
9  array[Guests] of var Seats: pos;
10 array[Hatreds] of var Seats: p1;
11 array[Hatreds] of var Seats: p2;
12 array[Hatreds] of var 0..1: sameside;
13 array[Hatreds] of var Seats: distance;

26 constraint forall(h in Hatreds) (
27     p1[h] = pos[h1[h]] /\
28     p2[h] = pos[h2[h]] /\
29     sameside[h] = bool2int(p1[h] <= 6 <-> p2[h] <= 6) /\
30     distance[h] = sameside[h] * abs(p1[h] - p2[h])
31         + (1 - sameside[h])*(abs(13 - p1[h] - p2[h]) + 1));
```



```
9  array[Guests] of var Seats: pos;

23 solve maximize sum (h in Hatreds) (
24   let {
25     var Seats: p1 = pos[h1[h]];
26     var Seats: p2 = pos[h2[h]];
27     var 0..1: sameside = bool2int(p1 <= 6 <-> p2 <= 6);
28   } in
29   sameside*abs(p1 - p2) + (1 - sameside)*(abs(13 - p1 - p2) + 1)
30 );
```

Wykorzystanie „samodzielne”

- Proste środowisko do edycji plików (`.mzn`, `.dzn`) i uruchamiania obliczeń
- Zestaw narzędzi do uruchamia z linii poleceń



Integracja z własnym oprogramowaniem

- Uruchomienie referencyjnych narzędzi z poziomu Python-a
- Możliwość przekazania danych z poziomu aplikacji oraz przejęcia wyników („surowe” i sformatowane)

Integracja z własnym oprogramowaniem

In [1]:
```
import pymzn
```

In [2]:
```
s = pymzn.minizinc('loan.mzn', 'loan1.dzn', solver=pymzn.g12mip)
print(s)
```

```
[{'B1': 780.0, 'B2': 551.2, 'B3': 313.248, 'B4': 65.77792000000005}]
```

In [3]:
```
for sol in s:
    print("B1 = {:.2f}, B2 = {:.2f}, B3 = {:.2f}, B4 = {:.2f}".format(sol['B1'], sol['B2'], sol['B3'], sol['B4']))
```

```
B1 = 780.00, B2 = 551.20, B3 = 313.25, B4 = 65.78
```

In [4]:
```
s = pymzn.minizinc('loan.mzn', 'loan1.dzn', solver=pymzn.g12mip, output_mode='item')
print(s)
```

```
['Borrowing 1000.00 at 16.0% interest\nand repaying 260.00 per quarter leaves 65.78 owing']
```

In [5]:
```
s = pymzn.minizinc('loan.mzn', data={'I':0.16, 'P':1000.0, 'B4': 0.0}, solver=pymzn.g12mip, output_mode='dict')
print(s)
```

```
[{'R': 275.4900453648024, 'B1': 764.5099546351977, 'B2': 519.6003074558032, 'B3': 264.894274389233}]
```

In [6]:
```
s = pymzn.minizinc('loan.mzn', data={'I':0.16, 'P':1000.0, 'B4': 0.0}, solver=pymzn.g12mip, output_mode='item')
print(s)
```

```
['Borrowing 1000.00 at 16.0% interest\nand repaying 275.49 per quarter leaves 0.00 owing']
```

Podsumowanie

- MiniZinc – język do (czytelnego) modelowania problemów decyzyjnych i optymalizacyjnych
- Dostępne solvery referencyjne, możliwa integracja z wieloma istniejącymi solverami
- Możliwość integracji z własnymi aplikacjami (Python, Java, …)