DM877

Discrete Optimization

Lecture 5

Introduction to MiniZinc: Examples

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume

- Examples
  - graph labelling with consecutive numbers
  - Cryptarithmetic (or verbal arithmetic or cryptarithm): Send + More = Money
  - Graph (map) coloring
  - Production planning
  - Investment planning
- Language elements:
  - parameters and variables, data types and enumerations
  - relational operators
  - arithmetics operators and functions (integer and float)
  - basic structure
  - arrays, sets, comprehensions
  - aggregate functions

Conditional Constraints

if-then-else-endif expression. An example of its use is int:

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

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

Conditional constraints are useful:

- to model alternative possibilities for variables, eg, initial board positions in the Sudoku
- as expressions in conditional assignments
- to format output.

Enumerations

```
enum Color;
```

```
$ minizinc --solver gecode -D"Color = { red, yellow, blue };" aust-enum.mzn
```

Enumerated types

An enumerated type parameter is declared as either:

```
<enum-name> : <var-name>
<l>..<u> : <var-name>
```

Example:

```
enum Color;
Color = { red, yellow, blue };
Color: cwa;
red..blue: cnt;
```

An enumerated type decision variable is declared as either:

```
var <enum-name> : <var-name>
var <l>..<u> : <var-name>
```

Example:

```
enum Color;
Color = { red, yellow, blue };
var Color: wa;
var red..blue: nt;
```

Built in operations on enumerated types

`enum_next(X,x)`	next value after x in the enum type X. False if last element.
`enum_prev(X,x)`	previous value before x in the enum type X. False if first element.
`to_enum(X,i)`	maps an integer expr i to an enum type value in type X. False if out of bounds.
`card(X)`	cardinality of an enumerated type X.
`min``(X)`	minimum element of an enumerated type X.
`max``(X)`	maximum element of an enumerated type X.

Complex Constraints

```
constraint s1 + d1 <= s2 \/ s2 + d2 <= s1;
```

Boolean literals are `true` and `false`

Boolean operators

conjunction (and)	`/\`
disjunction (or)	`\/`
only-if	`<-`
implies	`->`
if-and-only-if	`<->`
negation	`not`
casting to integers	`bool2int` or automatic

Example: Job Shop Scheduling

```
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 %% ensure the tasks occur in sequence
  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 %% ensure no overlap of tasks
  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]
    )
  );
```

```
solve minimize end;

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

```
JOB = anon_enum(5);
TASK = anon_enum(5);
d = [| 1, 4, 5, 3, 6
| 3, 2, 7, 1, 2
| 4, 4, 4, 4, 4
| 1, 1, 1, 6, 8
| 7, 3, 2, 2, 1 |];
```

Stable Marriage Problem

```
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;

% assignment
constraint forall (m in Men) (husband[wife[m]]=m);
constraint forall (w in Women) (wife[husband[w]]=w);
% ranking
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"];
```

```
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 |];
```

Note: array access `a[e]` implicitly adds the constraint

```
e in index_set(a)
```

Magic Series Problem

Magic series problem:
find a list of numbers $s = [s_0, \ldots, s_{n-1}]$ such that s_i is the number of occurrences of i in s. An example is $s = [1, 2, 1, 0]$.

```
int: n;
array[0..n-1] of var 0..n: s;

constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(bool2int(s[j]=i))));

solve satisfy;

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

Set Variables

Variables can also be sets containing integers
Hence, the set itself is the decision variable.
(This contrasts with an array in which the var keyword qualifies the elements in the array rather than the array itself since the basuc structure of the array is fixed)

```
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"];
```

It can be modelled also as an array of booleans. Which works best?
Set variables often help to remove symmetries.

Social Golfers

```
int: weeks;
set of int: WEEK = 1..weeks;
int: groups;
set of int: GROUP = 1..groups;
int: size;
set of int: SIZE = 1..size;
int: ngolfers = groups*size;
set of int: GOLFER = 1..ngolfers;

array[WEEK,GROUP] of var set of GOLFER: Sched;

constraint
  forall (i in WEEK, j in GROUP) (
         card(Sched[i,j]) = size
      /\ forall (k in j+1..groups) (
             Sched[i,j] intersect Sched[i,k] = {}
         )
  ) /\
  forall (i in WEEK) (
         partition_set([Sched[i,j] | j in GROUP], GOLFER)
  ) /\
  forall (i in 1..weeks-1, j in i+1..weeks) (
    forall (x,y in GROUP) (
       card(Sched[i,x] intersect Sched[j,y]) <= 1
    )
  );

solve satisfy;
```

Social Golfers

```
include "partition_set.mzn";

constraint % global constraint: redundant
  forall (i in WEEK) (
         partition_set([Sched[i,j] | j in GROUP], GOLFER)
         );


constraint
  % Fix the first week %
  forall (i in GROUP, j in SIZE) (
    ((i-1)*size + j) in Sched[1,i]
  ) /\
  % Fix first group of second week %
  forall (i in SIZE) (
    ((i-1)*size + 1) in Sched[2,1]
  ) /\
  % Fix first 'size' players
  forall (w in 2..weeks, p in SIZE) (
    p in Sched[w,p]
  );
```