NOPT042 Constraint programming: Tutorial 5 – Advanced constraint modelling

What was in the lecture? Nothing, the lecture was canceled.

The element constraint

Picat doesn't support indexation in constraints, e.g. `X #= L[I]`. Instead, it implements the `element` constraint:

```
element(I, L, X)
```

("X is on the Ith position in the list L"). But this constraint can also be used for "reverse indexing": when we have X and want to find its position in L. The constraint doesn't care about the direction; this is called *bidirectionality*.

For matrices (2D arrays), `Matrix[I,J]#=X` is expressed using the following constraint:

```
matrix_element(Matrix,I,J,X)
```

In [1]: `%load_ext ipicat`

```
Picat version 3.7
```

In [2]:
```
%%picat
import cp.

main =>
    L = new_array(10),
    L :: 0..9,
    all_different(L),
    % X #= L[I], % this does not work, use element instead
    element(I, L, X),
    X #= 5,
    solve($[max(I)],L),
    println(L).
```

```
{0,1,2,3,4,6,7,8,9,5}
```

Exercise: Langford's number problem

Consider the following problem (see the book):

> Consider two sets of the numbers from 1 to N. The problem is to arrange the $2N$ numbers in the two sets into a single sequence in which the two 1's appear one number apart, the two 2's appear two numbers apart, the two 3's appear three numbers apart, etc.

Try to formulate a model for this problem.

In [3]: `!ls exercises/langford/.solution/`

```
langford-channeling.pi   langford-primal.pi
langford-dual.pi         langford-primal-search-strategy.pi
```

In [4]: `!picat exercises/langford/.solution/langford-primal 8`

```
Default search strategy:
CPU time 1.924 seconds. Backtracks: 250746

{1,3,1,6,7,3,8,5,2,4,6,2,7,5,4,8}
```

In [5]: `!picat exercises/langford/.solution/langford-primal-search-strategy 8`

```
Good search strategy:
CPU time 0.106 seconds. Backtracks: 9160

{2,4,7,2,8,6,4,1,5,1,7,3,6,8,5,3}
```

In [6]: `!picat exercises/langford/.solution/langford-dual 8`

```
CPU time 0.0 seconds. Backtracks: 59

Positions: {1,4,8,11,9,6,2,5,3,7,12,16,15,13,10,14}
```

In [7]: `!picat exercises/langford/.solution/langford-channeling 8`

```
CPU time 2.844 seconds. Backtracks: 250746

{1,3,1,6,7,3,8,5,2,4,6,2,7,5,4,8}
Positions: {1,9,2,10,8,4,5,7,3,12,6,15,14,11,13,16}
```

In [8]: `!picat exercises/langford/.solution/langford-dual 20`

```
CPU time 9.171 seconds. Backtracks: 4440408

Positions: {1,2,4,6,7,9,19,20,18,21,24,25,26,22,23,14,17,15,10,12,3,5,8,11,13,16,27,
29,28,32,36,38,40,37,39,31,35,34,30,33}
```

The assignment problem

From Wikipedia:

The assignment problem is a fundamental combinatorial optimization problem. In its most general form, the problem is as follows:

> The problem instance has a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform as many tasks as possible by assigning at most one agent to each task and at most one task to each agent, in such a way that the total cost of the assignment is minimized.

Alternatively, describing the problem using graph theory:

> The assignment problem consists of finding, in a weighted bipartite graph, a matching of a given size, in which the sum of weights of the edges is minimum.

If the numbers of agents and tasks are equal, then the problem is called balanced assignment.

Exercise: Swimmers

(From: W. Winston, Operations Research: Applications & Algorithms.)

In medley swimming relay, a team of four swimmers must swim 4x100m, each swimmer using a different style: breaststroke, backstroke, butterfly, or freestyle. The table below gives their average times for 100m in each style. Which swimmer should swim which stroke to minimize total time?

Swimmer	Free	Breast	Fly	Back
A	54	54	51	53
B	51	57	52	52
C	50	53	54	56
D	56	54	55	53

Write a general model, generate larger instances, and try to make your model as efficient as possible.

```
In [9]:  !picat exercises/swimmers/instances
```

Sample instance:
{A,B,C,D}
{Free,Breast,Fly,Back}
{{54,54,51,53},{51,57,52,52},{50,53,54,56},{56,54,55,53}}

Random instance:
{Swimmer1,Swimmer2,Swimmer3,Swimmer4,Swimmer5,Swimmer6,Swimmer7,Swimmer8}
{Style1,Style2,Style3,Style4,Style5,Style6,Style7,Style8}
{{46,49,54,64,53,55,55,54},{60,55,47,64,65,49,65,52},{48,60,61,61,62,59,57,54},{47,5
0,50,58,46,64,50,45},{48,57,62,54,46,52,61,61},{60,63,57,59,65,55,65,47},{47,60,62,6
4,52,53,53,54},{56,56,46,55,54,51,56,57}}

In [10]:
```
!picat exercises/swimmers/.solution/swimmers
```

```
CPU time 0.001 seconds. Backtracks: 2


Primal model:
Swimmer A is swims Fly
Swimmer B is swims Back
Swimmer C is swims Free
Swimmer D is swims Breast

CPU time 0.0 seconds. Backtracks: 5


Primal model:
Swimmer A is swims Fly
Swimmer B is swims Free
Swimmer C is swims Breast
Swimmer D is swims Back

CPU time 0.0 seconds. Backtracks: 9


Primal model:
Swimmer A is swims Breast
Swimmer B is swims Fly
Swimmer C is swims Free
Swimmer D is swims Back

CPU time 0.0 seconds. Backtracks: 12


Primal model:
Swimmer A is swims Back
Swimmer B is swims Fly
Swimmer C is swims Free
Swimmer D is swims Breast

CPU time 0.001 seconds. Backtracks: 18


Primal model:
Swimmer A is swims Breast
Swimmer B is swims Back
Swimmer C is swims Free
Swimmer D is swims Fly

CPU time 0.0 seconds. Backtracks: 20


Primal model:
Swimmer A is swims Fly
Swimmer B is swims Breast
Swimmer C is swims Free
Swimmer D is swims Back

CPU time 0.0 seconds. Backtracks: 24
```

```
Primal model:
Swimmer A is swims Free
Swimmer B is swims Fly
Swimmer C is swims Breast
Swimmer D is swims Back

CPU time 0.001 seconds. Backtracks: 0


Dual model:
Style Free is swum  by A
Style Breast is swum  by C
Style Fly is swum  by B
Style Back is swum  by D

CPU time 0.0 seconds. Backtracks: 0


Channeling model:
Swimmer A is swims Free
Swimmer B is swims Fly
Swimmer C is swims Breast
Swimmer D is swims Back
or in the dual view
Style Free is swum  by A
Style Breast is swum  by C
Style Fly is swum  by B
Style Back is swum  by D
CPU time 54.654 seconds. Backtracks: 417964
```

In [11]: `!cat exercises/swimmers/.solution/swimmers.pi`

```
import cp.

main =>
    cl(sample_instance),
    sample_instance(SwimmerNames, StyleNames, Times),
    go(SwimmerNames, StyleNames, Times),

    M = 1000,
    time2(test(M)).

test(M) =>
    cl(random_instance),
    MinTime = 45,
    MaxTime = 65,
    N = 12,
    K = 12,
    foreach(_ in 1..M)
        random_instance(N, K, MinTime, MaxTime, SwimmerNames, StyleNames, Times),
        primal_model(StyleOfSwimmer, Times, TotalTime),
        solve([$min(TotalTime), StyleOfSwimmer]),
        % dual_model(SwimmerOfStyle, Times, TotalTime),
        % solve([$min(TotalTime), SwimmerOfStyle]),
        % channeling_model(StyleOfSwimmer, SwimmerOfStyle, Times, TotalTime),
        % solve([$min(TotalTime), StyleOfSwimmer ++ SwimmerOfStyle])
    end.


go(SwimmerNames, StyleNames, Times) =>
    primal_model(StyleOfSwimmer, Times, TotalTime),
    time2(solve([$min(TotalTime), StyleOfSwimmer])),
    println("\nPrimal model:"),
    foreach(I in 1..SwimmerNames.length)
        printf("Swimmer %w is swims %w\n", SwimmerNames[I], StyleNames[StyleOfSwimme
r[I]])
    end,

    dual_model(SwimmerOfStyle, Times, TotalTime),
    time2(solve([$min(TotalTime), SwimmerOfStyle])),
    println("\nDual model:"),
    foreach(J in 1..StyleNames.length)
        printf("Style %w is swum  by %w\n", StyleNames[J], SwimmerNames[SwimmerOfSty
le[J]])
    end,

    channeling(StyleOfSwimmer, SwimmerOfStyle, Times, TotalTime),
    time2(solve([$min(TotalTime), StyleOfSwimmer ++ SwimmerOfStyle])),
    println("\nChanneling model:"),
    foreach(I in 1..SwimmerNames.length)
        printf("Swimmer %w is swims %w\n", SwimmerNames[I], StyleNames[StyleOfSwimme
r[I]])
    end,
    println("or in the dual view"),
    foreach(J in 1..StyleNames.length)
        printf("Style %w is swum  by %w\n", StyleNames[J], SwimmerNames[SwimmerOfSty
le[J]])
    end.
```

```
primal_model(StyleOfSwimmer, Times, TotalTime) =>
    N = Times.length,
    %K = Times[1].length,
    K = N,
    StyleOfSwimmer = new_array(N),
    StyleOfSwimmer :: 1..K,
    all_different(StyleOfSwimmer),

    TimeOfSwimmer = new_array(N),
    foreach(I in 1..N)
        matrix_element(Times, I, StyleOfSwimmer[I], TimeOfSwimmer[I])
    end,
    TotalTime #= sum(TimeOfSwimmer).

dual_model(SwimmerOfStyle, Times, TotalTime) =>
    N = Times.length,
    K = N,
    SwimmerOfStyle = new_array(K),
    SwimmerOfStyle :: 1..N,
    all_different(SwimmerOfStyle),

    TimeOfStyle = new_array(K),
    foreach(J in 1..K)
        matrix_element(Times, J, SwimmerOfStyle[J], TimeOfStyle[J])
    end,
    TotalTime #= sum(TimeOfStyle).

channeling(StyleOfSwimmer, SwimmerOfStyle, Times, TotalTime) =>
    primal_model(StyleOfSwimmer, Times, TotalTime),
    dual_model(SwimmerOfStyle, Times, TotalTime),
    assignment(StyleOfSwimmer, SwimmerOfStyle).  %chanelling constraint
```

Modelling functions

In general, how to model a function (mapping) $f : A \to B$? Let's say $A = \{1, \ldots, n\}$ and $B = \{1, \ldots, k\}$.

- as an array:

  ```
  F = new_array(N},
  F :: 1..K.
  ```

- *injective*: `all_different(F)`
- *surjective*: a partition of A into classes labelled by B, to each element of B map a set of elements of A. In Picat we can model set as their characteristic vectors. More on modelling with sets later.
- *partial function*: a dummy value for undefined inputs

- *dual model*: switch the role of variables and values (not a function unless F injective, see above)
- *channelling*: combine the primal and dual models, if it is a bijection, then use `assignment(F, FInv)`

In [12]:
```
!picat functions.pi 4 4
```

{1,2,3,4}
{1,2,3,4}

In [13]:
```
!cat functions.pi
```

```
import cp.

main([N, K]) =>
    N := N.to_int,
    K := K.to_int,

    % function
    F = new_array(N),
    F :: 1..K,

    % injective
    all_different(F),

    % dual model if it is a bijection (K=N and injective)
    FInv = new_array(K),
    FInv :: 1..N,

    % channeling if it is a bijection (K=N and injective)
    assignment(F, FInv),



    % % dual model in general
    % FInv = new_array(K, N),
    % FInv :: 0..1,

    % % surjective in general
    % foreach(J in 1..N)
    %     sum([FInv[I, J]: I in 1..K]) #>= 1
    % end,

    % % channeling in general
    % foreach(I in 1..N)
    %     foreach(J in 1..N)
    %         (FInv[J, I] #= 1) #<=> (F[I] #= J)
    %     end
    % end,

    solve(F ++ FInv),
    println(F),
    println(FInv).
```