# NOPT042 Constraint programming: Tutorial 7 – Rostering, table constraint

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.5#5
```

## The constraint `regular`

```
regular(L, Q, S, M, Q0, F)
```

> Given a finite automaton (DFA or NFA) of $Q$ states numbered $1, 2, \ldots, Q$ with input from $\{1, \ldots, S\}$, transition matrix $M$, initial state $Q_0$ ($1 \leq Q_0 \leq Q$), and a list of accepting states $F$, this constraint is true if the list $L$ is accepted by the automaton. The transition matrix $M$ represents a mapping from $\{1, \ldots, Q\} \times \{1, \ldots, S\}$ to $\{0, \ldots, Q\}$, where $0$ denotes the error state. For a DFA, every entry in $M$ is an integer, and for an NFA, entries can be a list of integers.

---from the guide

## Example: Global contiguity

Given a 0-1 sequence, express that if there are 1's, they must form a single, contiguous subsequence, e.g. accept `0000` and `0001111100` but not `00111010`. (Problem from the book.)

In [2]:
```
!picat global-contiguity/global_contiguity.pi 0011100
!picat global-contiguity/global_contiguity.pi 0110111
```

```
ok
*** error(failed,main/1)
```

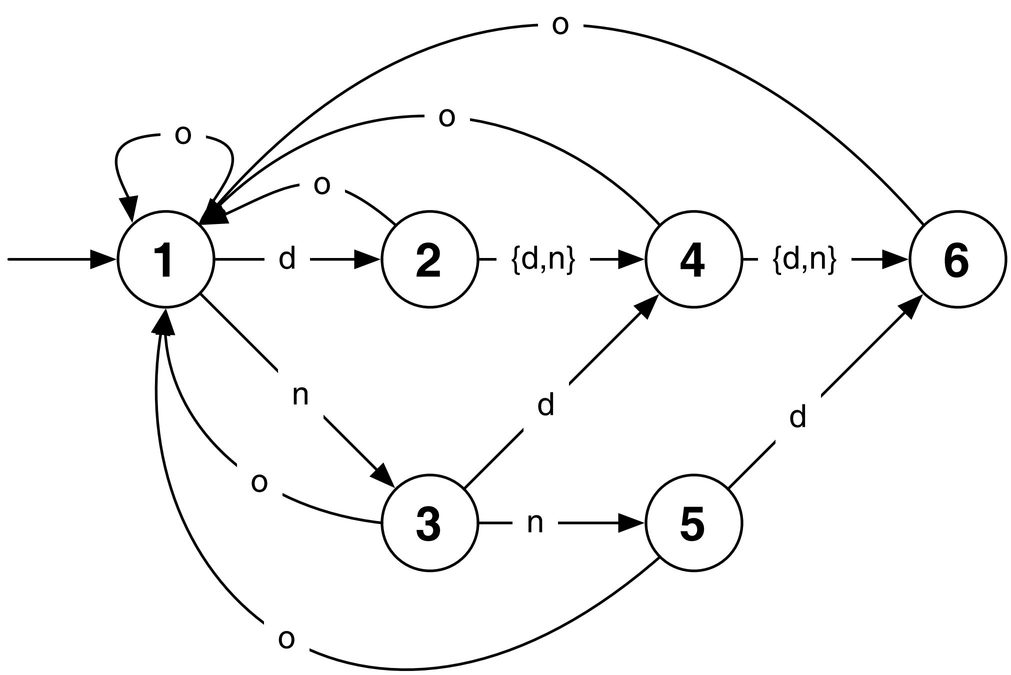
In [3]:
```
!cat global-contiguity/global_contiguity.pi
```

```
/*********************************************************
  Adapted from
  global_contiguity.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
*********************************************************/
import cp.

main([Xstr]) =>
  X = map(to_int,Xstr),
  global_contiguity(X),
  solve(X),
  println("ok").



global_contiguity(X) =>
  N = X.length,

  % This uses the regular expression "0*1*0*" to
  % require that all 1's (if any) in an array
  % appear contiguously.
  Transition = [
               [1,2], % state 1: 0*
               [3,2], % state 2: 1*
               [3,0]  % state 3: 0*
               ],
   NStates = 3,
   InputMax = 2,
   InitialState = 1,
   FinalStates = [1,2,3],

   RegInput = new_list(N),
   RegInput :: 1..InputMax,  % 1..2

   % Translate X's 0..1 to RegInput's 1..2
   foreach (I in 1..N)
      RegInput[I] #= X[I]+1
   end,

   regular(RegInput,NStates,InputMax,
           Transition,InitialState,FinalStates).
```

## Example: Nurse roster

Schedule the shifts of `NumNurses` nurses over `NumDays` days. Each nurse is scheduled for each day as either: (d) on day shift, (n) on night shift, or (o) off. In each four day period a nurse must have at least one day off, and no nurse can be scheduled for 3 night shifts in a row.

We require `ReqDay` nurses on day shift each day, and `ReqNight` nurses on night shift, and that each nurse takes at least `MinNight` night shifts. (Problem from the MiniZinc tutorial, similar [same?] problem is in the book.)




In [ ]:

## Constraint `sliding_sum` (not available in Picat)

```
sliding_sum(Low, Up, Seq, Variables) =>
   foreach(I in 1..Variables.length-Seq+1)
      Sum #= sum([Variables[J] : J in I..I+Seq-1]),
      Sum #>= Low,
      Sum #=< Up
   end.
```

-- from Hakank's Picat webpage, model sliding_sum.pi.

# The table constraint

> A *table constraint*, or an *extensional constraint*, over a tuple of variables specifies a set of tuples that are allowed (called positive) or disallowed (called negative) for the variables. A positive constraint takes the form

```
table_in(Vars,R)
```

> where `Vars` is either a tuple of variables or a list of tuples of variables, and `R` is a list of tuples in which each tuple takes the form $[a_1, \ldots, a_n]$, where $a_i$ is an integer or the don't-care symbol $*$. A negative constraint takes the form:

```
table_notin(Vars, R)
```

```
---from [the guide](http://picat-lang.org/download/picat_guide.pdf)
```

# Example: Graph homomorphism

Given a pair of graphs $G, H$, find all homomorphisms from $G$ to $H$. A *graph homomorphism* is a function $f : V(G) \to V(H)$ such that

$$\{u, v\} \in E(G) \implies \{f(u), f(v)\} \in E(H)$$

.

- Generalizes graph $k$-coloring ($c : G \to K_k$)
- Easier version: oriented graphs
- How would you model the Graph Isomorphism Problem?

# Example: Nurse roster using `table_in`

Model the above nurse roster problem using the constraint `table_in`.

# Homework: feast

A chef is planning a magnificent feast consisting of a sequence of dishes. Each dish can only be served once. Each dish tastes spicy, sour, salty, sweet, umami or bland.

- The chef would never serve two dishes of the same taste in a row, that would be boring.
- The first dish should be salty, and the last dish should be sweet.
- After a spicy dish the next dish must be bland or sweet.
- After a sour dish the next dish must be bland or umami.
- No spicy or umami dishes can go directly after a sweet dish.

The magnificence of the feast, which we want to maximize, is given by the sum of the value of the dishes.

Ue the constraint `regular` in your model. Running

```
picat feast.pi instance.pi
```

for the provided sample instance `instance.pi` should output the magnificence value of `33` and the sequence of dishes, one possibility is `[charsiubao,hotsoursoup,mapotofu,sesameprawn,kungpaochicken,coconutjelly]`.

(Adapted from Coursera course Basic modeling for discrete optimization.)

In [4]:
```
!cat feast/instance.pi
```

```
instance(Length, Dishes, Taste, Value) =>
    Length = 6,
    Dishes = [mapotofu, kungpaochicken, coconutjelly, laiwongbao, charsiubao, sesame
prawn, hotsoursoup, chilidumplings, glassnoodles, friedrice],
    Taste  = [umami, spicy, sweet, sweet, salty, salty, sour, spicy, bland, bland],
    Value  = [8, 4, 5, 3, 5, 7, 4, 3, 2, 1].
```