Constraint Systems

A New Global Constraint

The REGULAR Constraint

Let's see a new (strange) global constraint

$$\text{REGULAR}(X, T, s_0, F)$$

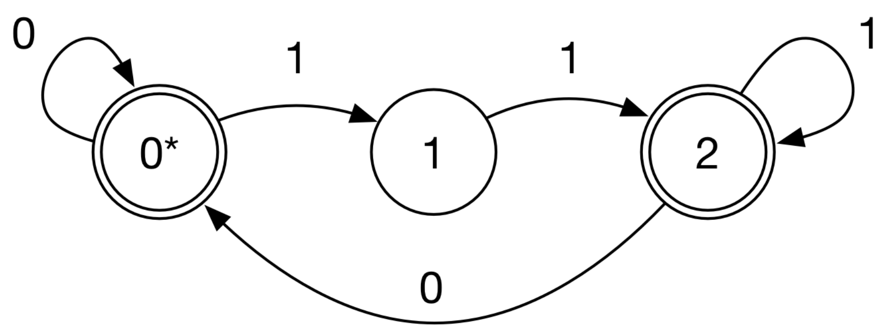
- The constraint ensures that the sequence of the X variables...
- ...Is compliant with a given Deterministic Finite Automaton (DFA)
  - T specifies the valid state transitions: (s_{cur}, v, s_{next})
  - s_0 is the initial state
  - F is the set of accepting states

In detail, the constrain is satisfied iff:

- All transitions are valid
- When the sequence ends, the DFA is in an accepting state

The REGULAR Constraint

Consider this example:



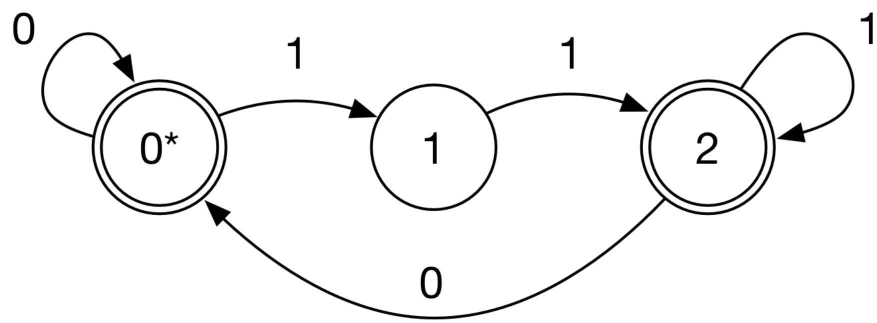
- The starred state is s_0 (initial state)
- The double-circled states are those in F (accepting states)

This is an example of a valid sequence (6 variables)

	X_0	X_1	X_2	X_3	X_4	X_5	
	0	0	1	1	1	0	
state:	0	0	0	1	2	2	0

The REGULAR Constraint

Consider this example:



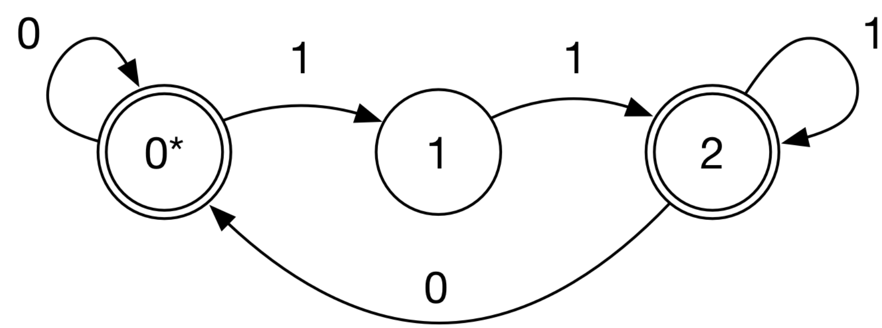
- The starred state is s_0 (initial state)
- The double-circled states are those in F (accepting states)

This an invalid sequence (forbidden transition)

	X_0	X_1	X_2	X_3	X_4	X_5	
	0	1	0	1	1	0	
state:	0	0	1	—	—	—	—

The REGULAR Constraint

Consider this example:



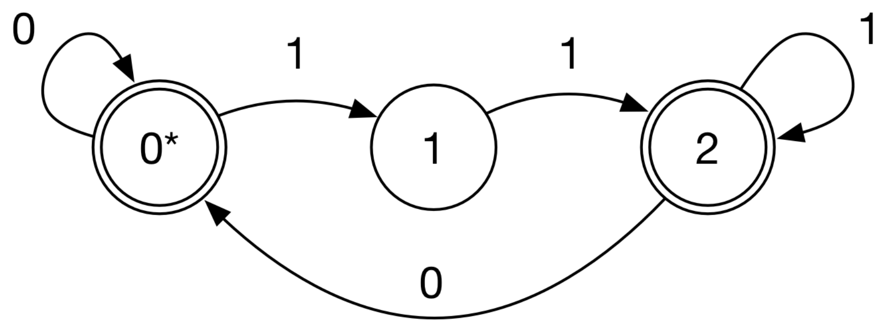
- The starred state is s_0 (initial state)
- The double-circled states are those in F (accepting states)

This an invalid sequence (the last state is not in F)

	X_0	X_1	X_2	X_3	X_4	X_5	
	0	1	1	1	0	1	
state:	0	0	1	2	2	0	1

The REGULAR Constraint

Consider this example:



- The starred state is s_0 (initial state)
- The double-circled states are those in F (accepting states)

As usual, the REGULAR constraint is capable of filtering

- In particular, it can enforce GAC on the X variables

The REGULAR Constraint

The constraint can sometimes be very useful:

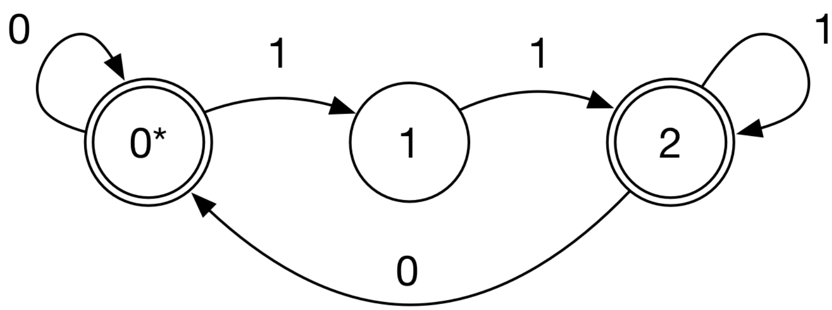
- Main example: complex regulations (laws) in work shift scheduling

Hover, the constraint is not always provided by solvers

The reason is that REGULAR can be decomposed

- We introduce a state variable Y_i for each X_i…
- …Plus an additional Y_n variable for the finale state
- We enforce valid transitions by posting multiple TABLE constraints
- Each TABLE constraint regulates a single transition

The REGULAR Constraint



extra index

index:	0	1	2	3	4	5	6
X	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	
Y	{0}	{0..2}	{0..2}	{0..2}	{0..2}	{0..2}	{0,2}

final state $\in F$

initial state = s_0

table constraint with $T =$

Y_i	X_i	Y_{i+1}
0	0	0
0	1	1
1	1	2
2	1	2
2	0	0

The REGULAR Constraint

Or-tools provides REGULAR via the API:

```
slv.TransitionConstraint(X, T, s0, F)
```

Where:

- `X` is a list with the X_i variables
- `T` is a "matrix" (list of lists) with the allowed transitions
- `s0` is the index of the initial state
- `F` is a list with the accepting states

The solver builds the decomposition automatically

Constraint Systems

A Shift-Scheduling Problem

A Shift-Scheduling Problem

Consider the following problem (by Tim Curtois):

We need to plan the working shifts for the employees of a company

- The planning horizon eoh is known (in days)

Each type of working shift k:

- Has a given duration l_k (in minutes)
- Cannot be followed by shifts in a given list I_k

Each worker i:

- Works a single shift type per day j (or takes a day off)
- Should work at most $M_{i,k}$ shifts of type k
- Should work at most D_i and at least d_i minutes overall

A Shift-Scheduling Problem

Again, each worker:

- Should work at most w_i week ends
  - For week-end days we have $j \mod 7 = 5 \text{ or } 6$
  - A week end is worked if there is a shift on Saturday or Sunday
- Should work at least c_i and at most C_i consecutive shifts
- Should take at least g_i consecutive days off
- Has a set of mandatory days off V_i (vacation)

There are positive preferences h over shifts:

- If worker i_h does not take shift k_h on day j_h, we pay a penalty w_h^p

There are negative preferences h over shifts:

- If worker i_h does take shift k_h on day j_h, we pay a penalty w_h^n

A Shift-Scheduling Problem

There are cover requirements h for shifts and days:

- Let y_h be the number of shifts k_h on day j_h
- If y_h is less than a given requirement r_h, we pay $w_h^u(r_h - y_h)$
- If y_h is greater than a given requirement r_h, we pay $w_h^o(y_h - r_h)$
- The penalty u_h is much greater than the penalty o_h

About the number of entities:

- Let n_e be the number of workers
- Let n_s be the number of shift types
- Let n_p be the number of positive preferences
- Let n_n be the number of negative preferences
- Let n_c be the number of cover requirements

A Shift-Scheduling Problem

This is a rather complex problem

- Writing a model takes time (more than we have in this session)...
- So we will see one possible approach together

The main variables are:

$$x_{i,j} \in \{0..n_s\} \qquad \forall i = 0..n_e - 1, j = 0..eoh - 1$$
$$w_{i,j} \in \{0, 1\} \qquad \forall i = 0..n_e - 1, j = 0..eoh - 1$$

- $x_{i,j}$ is the shift type for worker i on day j
- $x_{i,j} = n_s$ for a day off
- $w_{i,j} = 1$ if worker i does not take a day off on j

Chaining constraints:

$$w_{i,j} = (x_{i,j} \neq n_s) \qquad \forall i = 0..n_e - 1, j = 0..eoh - 1$$

A Shift-Scheduling Problem

Compatibility constraints between subsequent shifts:

$$\text{TABLE}([x_{i,j}, x_{i,j+1}], T) \qquad \forall i = 0..n_e - 1, j = 0..eoh - 2$$

- Where T contains all the valid transitions, i.e.

$$(k', k'') \in T \text{ iff } k' = n_s \vee k'' = n_s \vee k'' \notin I_{k'}$$

Maximum number of shift types per worker:

$$\text{GCC}(X_{i,:}, [0..n_s], [0..0], [M_{i,0}, M_{i,1}, \ldots \infty]) \qquad \forall i = 0..n_e - 1$$

Limits on the number of minutes per worker ($l_{n_s} = 0$):

$$\sum_{j=0..eoh-1} l_{x_{i,j}} \leq D_i \qquad \forall i = 0..n_e - 1$$

$$\sum_{j=0..eoh-1} l_{x_{i,j}} \geq d_i \qquad \forall i = 0..n_e - 1$$

A Shift-Scheduling Problem

Maximum number of weekends:

- First we introduce a set of additional variables

$$W_{i,h}^{we} \in \{0,1\} \qquad \forall i = 0..n_e - 1, h = 0..{}^{eoh}/_7$$

- $W_{i,h}^{we} = 1$ if worker i works on the h-th weekend
- Then we define the variable via the constraints:

$$W_{i,h}^{we} = \max(W_{i,7(h+1)-2}, W_{i,7(h+1)-1}) \qquad \forall i = 0..n_e - 1, h = 0..{}^{eoh}/_7$$

- Then we constraint the sum:

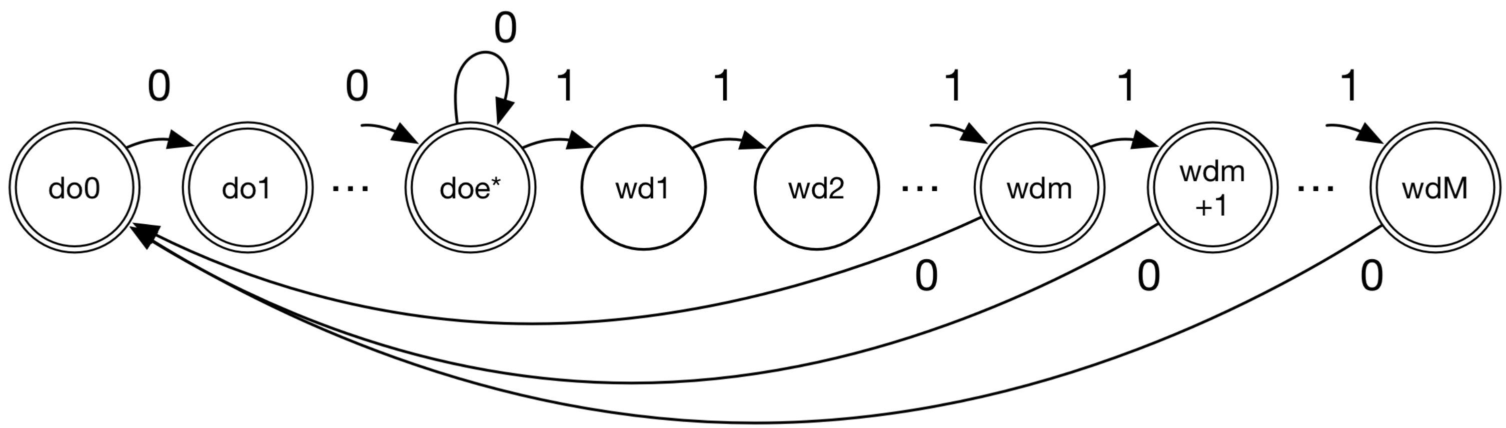
$$\sum_{h=0..{}^{eoh}/_7} W_{i,h}^{we} \leq w_i$$

Mandatory days off:

$$x_{i,j} = n_s \qquad \forall i = 0..n_e - 1, j \in V_i$$

A Shift-Scheduling Problem

Constraint on consecutive days:

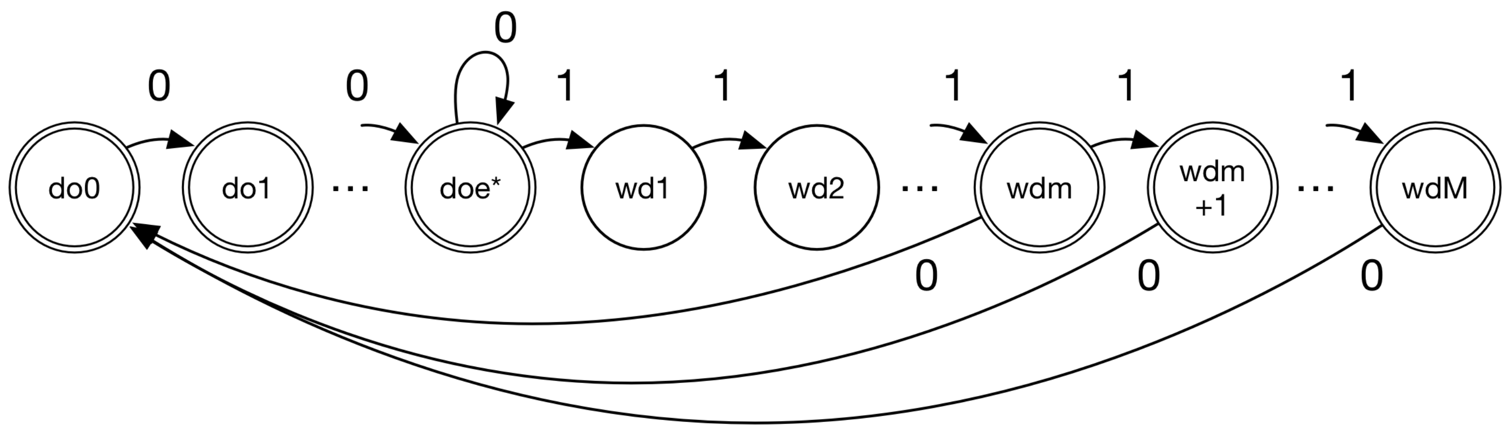


REGULAR on the $W_{i,j}$ variables for each worker i

- doX = X days off
- doe = enough days off ("doe" stands for a number)
- wdX = X working days
- wdm = min working days ("wdm" stands for a number)
- wdM = max working days ("wdM" stands for a number)

A Shift-Scheduling Problem

Constraint on consecutive days:

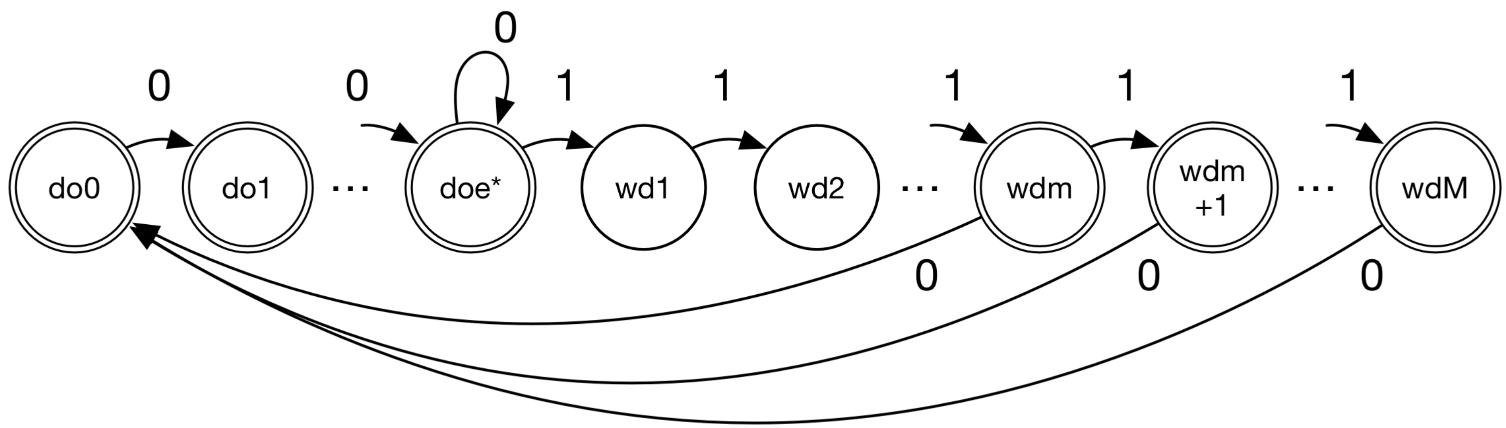


REGULAR on the $W_{i,j}$ variables for each worker I

- Once we reach doe, we stop counting the days off
- No working day allowed in doX before doe is reached
- No day off allowed in wdX before wdm is reached
- No working day allowed one wdM is reached

A Shift-Scheduling Problem

Constraint on consecutive days:



REGULAR on the $W_{i,j}$ variables for each worker I

- HP: infinite days off before and after the planning period
- Hence, doe is the initial state...
- ...And all doX and doe state are accepting...
- ...But only states between wdm and wdM are accepting

A Shift-Scheduling Problem

The penalty for not satisfying positive preferences:

$$z_p = \sum_{h=0..n_p-1} w_h^p (x_{i_h, j_h} \neq k_h)$$

The penalty for not satisfying negative preferences:

$$z_n = \sum_{h=0..n_n-1} w_h^n (x_{i_h, j_h} = k_h)$$

The penalty for not satisfying cover preferences:

$$\textsc{count}(X_{:,j}, k_h, y_h) \qquad \forall h = 0..n_c - 1$$

$$z_o = \sum_{h=0..n_c-1} w_h^o \max(y_h - r_h, 0)$$

$$z_u = \sum_{h=0..n_c-1} w_h^u \max(r_h - y_h, 0)$$

A Shift-Scheduling Problem

The overall cost function is:

$$\min z = z_p + z_n + z_o + z_u$$

The model and two instances are available on the start-kit

- Change the search strategy and solve `Instance1` to optimality
  - Pick the branching variables, select the var/value strategy...
  - ...Order the variables based on your ideas
- Then try to find the best possible solution for `Instance2`
  - The optimal solution is 828...
  - ...See how close you can get!

NOTE: both tasks are difficult!