MODELIZACIÓN DE TIMETABLING MEDIANTE EL LENGUAJE TOY CON RESTRICCIONES SOBRE DOMINIOS FINITOS

Por
RAMÓN GONZÁLEZ DEL CAMPO RODRÍGUEZ BARBERO
Profesor Asociado en la Universidad Complutense de Madrid

SUMARIO: 1. INTRODUCCIÓN.- 2. PROBLEMA. 2.1. Planteamiento.- 2.2. Nivel 1: Marco Laboral Básico. 2.3. Nivel 2: Especificación Detallada.- 3. IMPLEMENTACIÓN.

1. INTRODUCCIÓN

Timetabling, confección automática de calendarios, es un tipo de problemas que podemos implementar eficazmente mediante la programación declarativa, en particular, el lenguaje TOY con restricciones sobre dominios finitos, desarrollado por la Universidad Complutense de Madrid, ofrece una solución elegante y eficaz

2. PROBLEMA

En un centro de trabajo se presta un servicio continuado que se cubre con personal que se organiza en turnos. Se trata de asignar el personal en tramos horarios para cubrir todo el horario de servicio según las especificaciones que se indican en el siguiente apartado.

2.1. PLANTEAMIENTO

Los trabajadores se organizan en turnos de trabajo; cada turno de trabajo está compuesto por tres trabajadores. En cada turno de trabajo hay dos trabajadores con nivel de cualificación 1 (N1) y un trabajador con nivel de cualificación 2 (N2). Los trabajadores de nivel 1 tienen experiencia probada en asumir todas las responsabilidades del turno, mientras que los de nivel 2 no tienen la suficiente experiencia o están en proceso de aprendizaje.

El planteamiento del problema se va a realizar con diferentes niveles de detalle.

2.2. NIVEL 1: MARCO LABORAL BÁSICO

Planteamiento del marco laboral básico:

- Un día de trabajo se corresponde con un intervalo de 24 horas que comienza a las 8 de la mañana y termina a la misma hora del día siguiente.
- El descanso mínimo de un trabajador después de un turno de trabajo nocturno es de dos días.
- Cada trabajador debe cumplir el número de horas anuales fijadas por el calendario laboral que proporciona la empresa. Dado que no se admite el pago de horas extras, se contempla la posibilidad de un

exceso o defecto a lo sumo de 24 horas trabajadas a lo largo del año.

- Cada trabajador debe cumplir las horas mensuales del calendario laboral con un exceso o defecto máximo de 21 horas.
- Las horas por exceso o por defecto acumuladas a lo largo de los meses no puede sobrepasar el límite de 50 horas.
- Hay 36 días naturales de vacaciones que se pueden dividir hasta en 19 periodos diferentes, siendo uno de ellos de 18 días consecutivos. El resto se puede elegir a conveniencia del trabajador y conforme a las restricciones siguientes:
    - De cada grupo de trabajo sólo un trabajador puede estar de vacaciones.
    - Solo se pueden conceder simultáneamente vacaciones a un máximo de 4 trabajadores.
    - Sólo se pueden conceder simultáneamente vacaciones a un máximo de 3 trabajadores de nivel 1.

2.3. NIVEL 2: ESPECIFICACIÓN DETALLADA

- 13 trabajadores organizados en cuatro turnos de trabajo de 3 personas fijas en cada turno y un trabajador comodín para resolver las bajas eventuales.
- Tramos horarios:
    - Tramo 1 (T1) : 8 a 8 (24 horas).
    - Tramo 2 (21) : 8 a 22 (14 horas).
    - Tramo 3 (T3) : 18 a 8 (14 horas).
    - Tramo 4 (T4) : 15 a 8 (17 horas).
    - Tramo 5 (T5) : 8 a 21 (13 horas).
    - Tramo 6 (T6) : 8 a 15 (7 horas).

- Tipos de jornadas:
    - Jornada 1 (J1): Un día laboral con tres trabajadores disponibles. Cada trabajador tiene asignado un tramo horario diferente (T1,T2 ó T3) que va rotando en los días de trabajo en los días sucesivos. Dos de ellos deben poseer nivel de cualificación N1 y el otro N2. Los dos posibles secuencias de un ciclo de días trabajados son :

Trabajador 1 → T1 T3 T2 T1
Trabajador 2 → T2 T1 T3 T2
Trabajador 3 → T3 T2 T1 T3

- Jornada 2 (J2): Un día laboral con dos trabajadores disponibles. Cada trabajador tiene asignado un tramo horario diferente (T1 ó T4) que va rotando en los días de trabajo sucesivos. Al menos uno de ellos debe tener cualificación N1. La única secuencia posible de un ciclo de días trabajados es:

Trabajador 1 → T1 T4 T1
Trabajador 2 → T4 T1 T4

- Jornada 3 (J3): Días especiales de trabajo sin servicio continuado: 24 y 31 de diciembre. Cada trabajador tiene asignado el mismo tramo horario (T5). Al menos uno de ellos debe tener cualificación N1.

Consideraciones adicionales:

- Bajas esporádicas.
- Bajas continuadas (enfermedad, embarazos, mili...)

3. IMPLEMENTACIÓN

Mediante el lenguaje de programación TOY sobre dominios finitos podemos representar fácilmente todas las restricciones y especificaciones planteadas en nuestro problema:

```
%%-------------------------------------------------------------
%% Planificación con dominios finitos: versión 0.3
%%
%%-------------------------------------------------------------

include «cflpfd.toy»
```

```
type nTramo = int
type t_dia = int
type t_grupo = [nTramo]
type t_diario = (t_grupo, t_dia)
type t_mensual = [t_diario]
type t_mes_turno = (t_mensual, int)
type t_planif = [t_mes_turno]

%Funcion principal:

planificar:: int -> int -> int -> [int] -> t_planif -> bool
%Obtenemos una planificacion correcta:

planificar Dias Turn Trab Fest P = true <== dimensionar P == formato
Dias Turn Trab,
                contenido P, gen_turnos_dia P Dias, cont_valido P,
                rotaciones_plan P F Dias,
                etiquetar P []
%gen_turnos_dia P Dias, cont_valido P,
%%====================
%% Establecemos las dimensiones de la planificación:
%%

dimensionar:: t_planif -> [([(int,int)],int)]
%Extraemos el formato de la planificacion:
dimensionar [] = []
dimensionar [(T,Tur)|Ts] = [((extraer_dias T),Tur)]++dimensionar Ts

extraer_dias:: t_mensual -> [(int,int)]
%Extraemos el numero de trabajadores y los dias del mes de trabajo de un
grupo :
extraer_dias [] = []
extraer_dias [(G,D)|Gs] = [(card G,D)]++extraer_dias Gs

formato:: int -> int -> int -> [([(int,int)],int)]
%Formato de la planificacion:
```

```
formato Dias Turno Trab = if Turno > 0 then
                    formato Dias (Turno-1) Trab ++ [(formato_turno_mes
Dias Trab,Turno)]
                  else
                    []

formato_turno_mes:: int -> int -> [(int,int)]
formato_turno_mes Dias Trab = if Dias > 0 then
                    formato_turno_mes (Dias-1) Trab ++ [(Trab,Dias)]
                  else
                    []

%%=======================
%% Contenido:
%%

contenido:: t_planif -> bool
%%contenido [] = true
contenido P = true <== X == rec_total P, domain X 0 3

%%=======================
%% Etiquetado:
%% Capturamos todas las variables y las etiquetamos
%%

etiquetar:: t_planif -> [labelingType] -> bool
etiquetar P Label = true <== Z == rec_total P, labeling Label Z

rec_total:: t_planif -> [nTramo]
%Recolectamos todas las variables:
rec_total [] = []
rec_total [(C,T)|Rs] = rec_turno C ++ rec_total Rs

rec_turno:: t_mensual -> [nTramo]
%Recolectamos las variables de cada turno
rec_turno [] = []
```

```
rec_turno [(T,D)|Rs] = T ++ rec_turno Rs
%%=====================================

cont_valido:: t_planif -> bool

cont_valido [] = true
cont_valido [(Mes,Tur)|Rs] = true <== cont_mes Mes, cont_valido Rs

cont_mes:: t_mensual -> bool

cont_mes [] = true
cont_mes [(Grupo,Dia)|Rs] = true <== valido Grupo, cont_mes Rs
%%=====================================
%% Unas pruebas:
%%
ccc:: t_grupo -> bool
ccc T = true <== domain T 0 3, valido T , labeling [] T
%%=====================================

valido:: t_grupo -> bool

valido T = true <== todosceros T // restovalidos T

todosceros:: t_grupo -> bool

todosceros [] = true
todosceros [Q|Qs] = true <== Q #= 0, todosceros Qs

restovalidos:: t_grupo -> bool

restovalidos T = true <== all_different T, ninguncero T

ninguncero:: t_grupo -> bool
```

```
ninguncero [] = true
ninguncero [Q|Qs] = true <== Q #\= 0, ninguncero Qs

%%==========================================
%%
%% Exigimos que solo trabaje un turno diario
%%

gen_turnos_dia:: t_planif -> int -> bool
% Genera el trabajo de los turnos por dias:

gen_turnos_dia L 0  = true
gen_turnos_dia L Nd = true <== restr (gen L Nd) 1, gen_turnos_dia L (Nd-1)

gen:: t_planif -> int -> [t_grupo]
%Genera el trabajo en un dia

gen [] Nd = []
gen [(L,T)|Ls] Nd = [obtener_turno L Nd] ++ gen Ls Nd

obtener_turno:: t_mensual -> int -> t_grupo
obtener_turno [(L,T)|Ls] Nd = if Nd == T then L
                              else obtener_turno Ls Nd

uno:: [t_grupo] -> bool
uno [R|Rs] = true <== nodescanso R, descanso Rs

descanso:: [t_grupo] -> bool
descanso [] = true
descanso [R|Rs] = true <== todosceros R, descanso Rs

nodescanso:: t_grupo -> bool
nodescanso [] = false

nodescanso [R|Rs] = true <== R #\= 0
nodescanso [R|Rs] = true <== nodescanso Rs
```

```
rotacion:: [A] -> [A]
rotacion [A,B|Cs] = [B|Cs]++[A]

restr:: [t_grupo] -> int -> bool
restr L N = if N == card L then (uno L)
        else (uno L) // restr (rotacion L) (N+1)

%%=====================================
%%
%% Rotacion de turnos:
%%

rotaciones_plan:: t_planif -> [int] -> int -> bool
%Para cada turno las asignaciones deben rotar

rotaciones_plan [] F Max = true
% Versión antigua
% rotaciones_plan [(T,Num)|Rs] = true <== rot_turno (extraer
T),rotaciones_plan Rs
%
rotaciones_plan [(T,Num)|Rs] F Max = true <== rot_turno (festivos T F),
                                rot_turno (laborables T (generar_lab F Max)),
                                rotaciones_plan Rs

extraer:: t_mensual -> [t_grupo]
%Extraemos el trabajo efectivo de un mes

extraer [] = []
extraer [(T,D)|Rs] = [T]++extraer Rs

%%=====================================
num_dia:: t_mensual -> int -> t_grupo
%Accedemos al grupo de cierto dia
num_dia [(T,D)|Ts] N = if N==1 then T else num_dia Ts (N-1)
```

```
festivos:: t_mensual -> [int] -> [t_grupo]
%Extraer festivos
festivos L [] = []
festivos L [F|Fs] = [num_dia L F] ++ festivos L Fs

laborables:: t_mensual -> [int] -> t_grupo
laborables L [] = []
laborables L [F|Fs] = [num_dia L F] ++ laborables L Fs

generar_lab:: [int] -> int -> [int]
% Generamos la lista de los dias laborables

generar_lab L Max = quitar L (generar Max)

generar:: int -> [int]
generar 0 = []
generar Max = [Max]++generar (Max-1)

quitar:: [int] -> [int] -> [int]
quitar [] Todos = Todos
quitar [F|Fs] Todos = quitar Fs (eliminar F Todos)
%%==========================================

rot_turno::[t_grupo] -> bool
%El trabajo debe rotar para cada turno

rot_turno [] = true
rot_turno [T|Ts] = true <== rot_elem [T|Ts] 1, rot_turno Ts

rot_elem::[t_grupo] -> int -> bool
rot_elem [T|Ts] X = true <== todosceros T
rot_elem [T|Ts] X = true <== todos_descanso Ts
rot_elem [T|Ts] X = true <== rotacion T == elemento Ts X,todos_descanso
(nprimeros Ts (X-1))
rot_elem L X = if X + 1 < card L then rot_elem L (X+1) else false
```

```
todos_descanso::[t_grupo]-> bool
%Exigimos que todos los turnos sean de descanso

todos_descanso [] = true
todos_descanso [T|Ts] = true <== todosceros T, todos_descanso Ts

%%==========================================
%%
%% Descanso nocturno:
%%

descanso_plan:: t_planif -> bool
%Para cada turno las asignaciones tener descanso por nocturnidad

descanso_plan [] = true
descanso_plan [(T,Num)|Rs] = true <== des_turno (extraer
T),descanso_plan Rs

des_turno::[t_grupo]-> bool
des_turno [] = true
des_turno [T|Ts] = true <== des_elem [T|Ts], des_turno Ts

des_elem::[t_grupo]-> bool
des_elem [T|Ts] = true <== es_nocturno T == false
des_elem [T|Ts] = true <== es_nocturno T,todos_descanso(nprimeros Ts 2)

es_nocturno:: t_grupo -> bool
es_nocturno [] = false
es_nocturno [T|Ts] = true <== tramo_nocturno T // es_nocturno Ts

tramo_nocturno:: nTramo -> bool
%Un tramo es nocturno
tramo_nocturno 0 = false
tramo_nocturno 1 = true
tramo_nocturno 2 = false
tramo_nocturno 3 = true
```

```
%%-----------------------------
%% Funciones auxiliares:
%%

pertenece::int -> [int] -> bool
%Cierto si un entero pertenece a la lista de enteros
pertenece N [] = false
pertenece N [M|Ms] = if N==M then true else pertenece N Ms

nopertenece::int -> [int] -> bool
%Cierto si un entero no pertenece a la lista de enteros
nopertenece N M = true <== pertenece N M ==false

elemento::[A] -> int -> A
%Tomamos el N-esimo elemento de la lista

elemento [R|Rs] N = if N==1 then R else elemento Rs (N-1)

nprimeros::[A] -> int -> [A]
%Tomamos los N primeros elementos de la lista

nprimeros [] N = []
nprimeros [R|Rs] N = if N == 0 then [] else [R]++nprimeros Rs (N-1)

eliminar:: A -> [A] -> [A]
% Eliminamos un elemento de una lista

eliminar Elem [] = []
eliminar Elem [L|Ls] = if Elem==L then Ls else [L]++eliminar Elem Ls

card:: [A] -> int
card [] = 0
card [D|Ds] = 1 + card Ds
```

```
(++):: [A] -> [A] -> [A]
[] ++ Y = Y
[X|Xs] ++ Ys = [X|Xs++Ys]

rId:: (A -> bool) -> A -> A
rId F X = X <== F X

data nat = z | s nat
infixr 40 <<
(<<):: nat -> nat -> bool
z << X = true
(s X) << z = false
(s X) << (s Y) = true <== X << Y


infixr 40 //
(//):: A -> A -> A

X // Y = X
X // Y = Y

%Fin del pgm
```