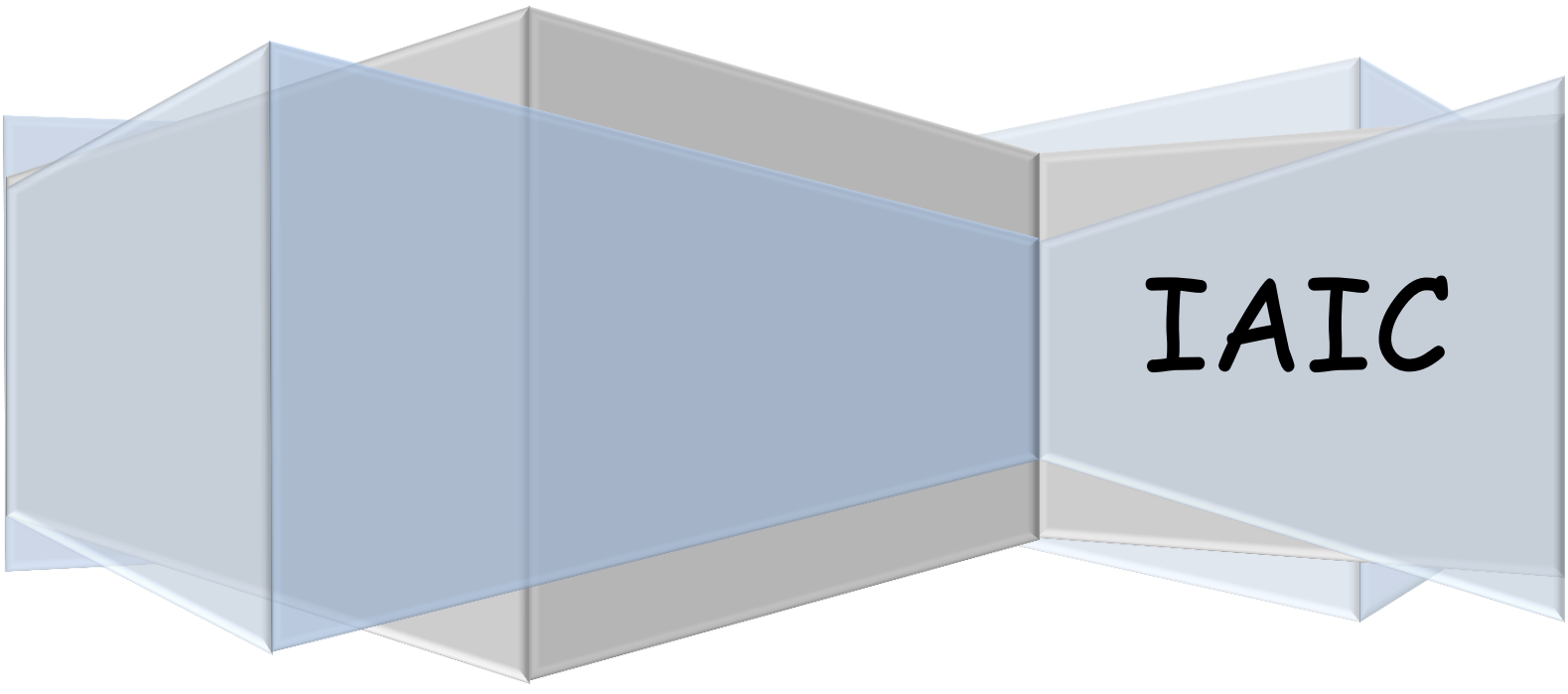
Práctica 2 IA

Satisfacción de Restricciones

Carlos Jesús Fernández Basso



Índice

Problema de las N reinas

A. Implementación básica en Prolog

En esta implementación vamos proponiendo sitios donde poner reinas y después comprobando.

B. Implementación en ECL^{i}PS^{e} basada en la biblioteca *suspend*

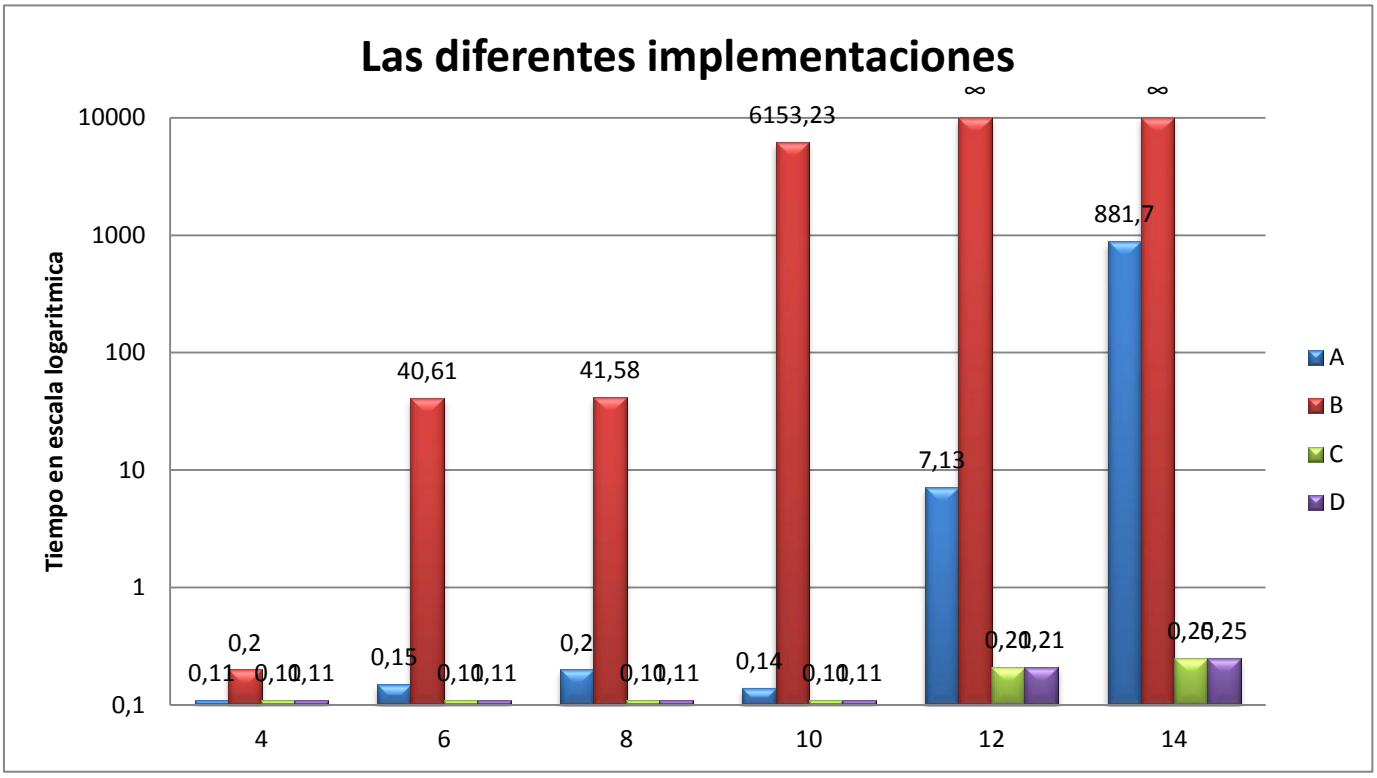
Esta implementación es parecida a la anterior pero con la biblioteca *suspend*. Lo que hacemos es imponer las restricciones antes de los valores, con ello la eficiencia empeora puesto que se hace lo mismo que antes (se explora todo el espacio sin ir propagando las restricciones).

C. Implementación en ECL^{i}PS^{e} basada en el predicado *labeling* de la biblioteca *ic*

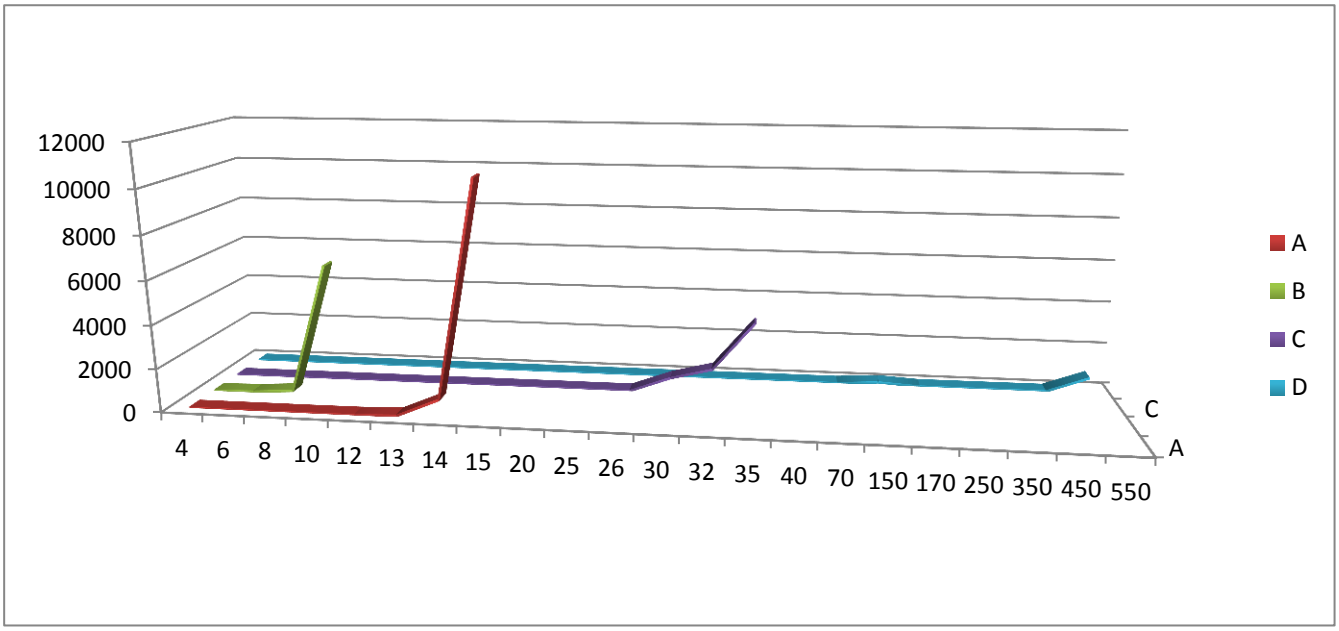
Esta implementación es como la de *suspend* pero cuando proponemos una reina propagamos las restricciones. Por ello, se van podando resultados y es más eficiente.

D. Implementación en ECL^{i}PS^{e} basada en el predicado *search* de la biblioteca *ic*

Esta es igual que la anterior pero con la heurística de coger el elemento que menor número de posibilidades tiene. Es mucho más eficiente, pero en función de N la eficiencia puede ser diferente.

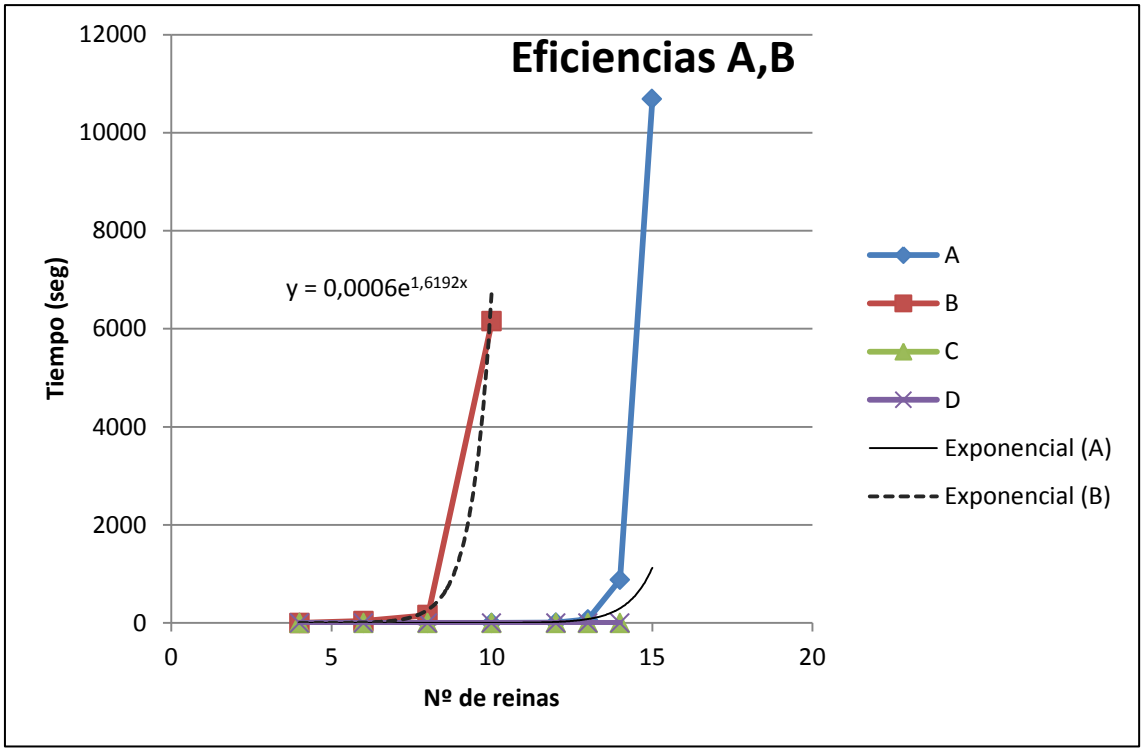


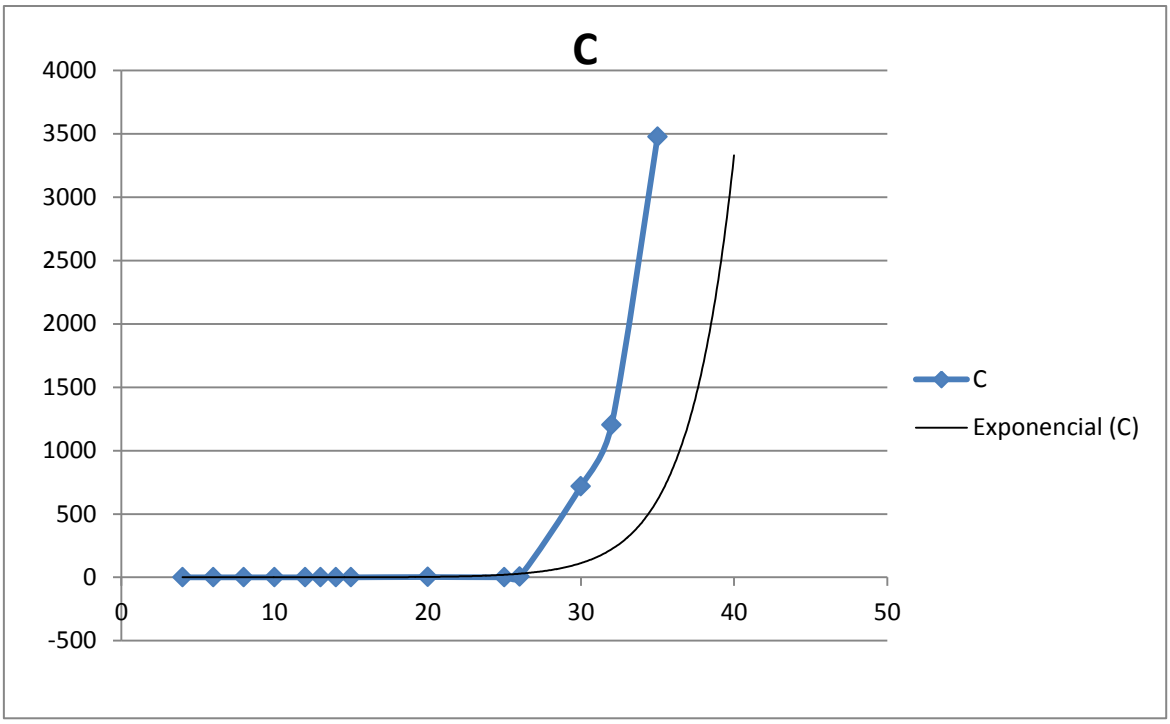
	A	B	C	D
4	0,11	0,2	0,11	0,11
6	0,15	40,61	0,11	0,11
8	0,2	41,58	0,11	0,11
10	0,14	6153,23	0,11	0,11
12	7,13	No ejecutado	0,21	0,21
14	881,7	No ejecutado	0,25	0,25

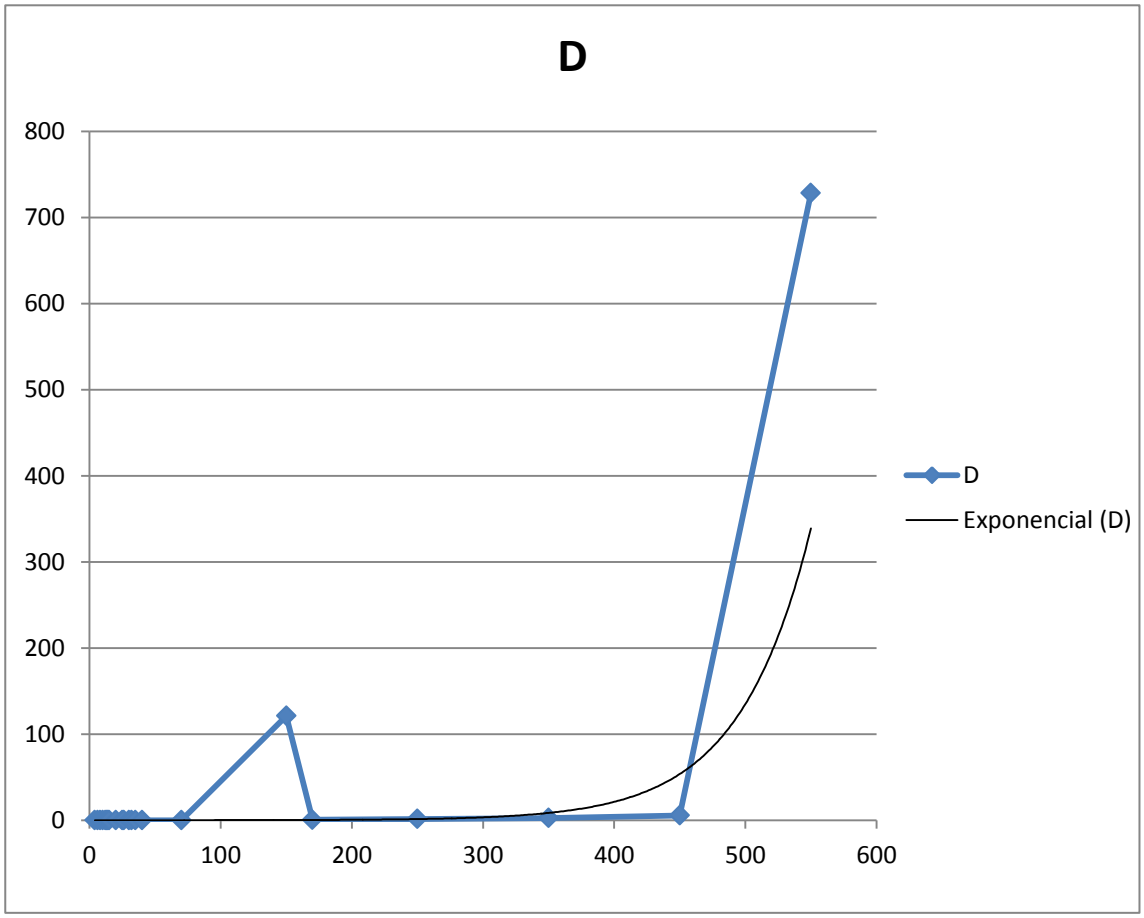


	A	B	C	D
4	0,01	0,2	0,11	0,01
6	0,01	40,61	0,11	0,01
8	0,01	161,58	0,11	0,01
10	0,11	6153,23	0,11	0,01
12	7,16		0,21	0,01
13	70,93		0,24	0,01
14	881,7		0,25	0,01
15	10687,1		0,15	0,01
20			2,5	0,02
25			0,53	0,02
26			4,62	0,02
30			717,43	0,02
32			1201,94	0,02
35			3476,25	0,02
40				0,03
70				0,06
150				121,21
170				0,53
250				1,51
350				2,5
450				5,57
550				728,49

Regresiones de las diferentes implementaciones







Como podemos ver en las gráficas, estas implementaciones se acercan a la gráficas exponenciales pero, como veremos en la última, al haber utilizado una heurística, dependiendo del número de N reinas podemos encontrarnos una solución más o menos eficiente, aunque es mucho más eficiente que el C.

Puzzle lógico

Este problema lo he resuelto solo con una implementación en Prolog sin utilizar ninguna otra librería, ayudándome de las siguientes funciones y estructuras de Prolog:

- Variable *Matriz:* lista de listas en la cual tenemos:
  - Tenemos 5 sublistas con 5 posiciones, como si fuera la calle con las cinco casas.
  - En esta pondremos una primera restricción (la número 5 en el guion). La codificaremos poniendo en cada lugar la diferente información suministrada(dueño,color,comida,cumpleaños).

  Nota: Las fechas las codificaremos con números del 0 al 4

- Para poner el resto de restricciones e ir rellenando los datos utilizaremos member (Elemento, Lista). La utilizaremos de la siguiente manera:
  - Member ([Dueño, Color, Comida, Cumpleaños], Matriz) pondremos valores dentro de la Primera lista y el resto lo dejaremos como variable libre (_). Esta función nos instanciará la lista con una de las posibles sublistas con la primera que satisfaga las restricciones, si no puede con toda la lista 'Matriz' dará false y volverá atrás (backtracking).

  - Para la codificación de las restricciones, las cuales implican las fechas y su relación de ordenación (mayor, menor), utilizaremos lo siguiente:
    - *Member ([Carac1, Carac2, Carac3, X],Matriz),*
      Member ([Carac4, Carac5, Carac6, Y],Matriz),
      X<Y (o X>Y),

      En las variables *CaracN* pondremos las características del individuo/comida/casa en las cuales la fecha de nacimiento del dueño está implicada en la restricción, el resto las dejaremos libres (_).
      X, Y instanciarán la fecha en cuestión que tiene y en la siguiente línea podremos la restricción de orden dicha (<, >).
  - Para el resto de restricciones utilizaremos listas con las características suministradas y el resto de variables libre (_) Ej.
    - *Member ([DueñoX, _, ComidaX, _],Matriz),* en esta restricción estamos determinando que el *DueñoX* tiene relacionada la *ComidaX*.

- En el caso de que la restricción fuera negada (*DueñoX* no tiene relación con *ComidaX*) utilizaremos la función *not* o negaremos la clausula (**member(Lista2, Matriz);…. member(ListaN, Matriz)**) donde pondremos en lista las posibilidades combinatorias menos la que nos han especificado (que no están dentro de las posibilidades) (*([DueñoX, _, ComidaX, _]*).

Planificación de tareas I (ensamblado de un coche)

En este problema haremos uso de la biblioteca ‘ic’ y ‘branch_and_bound’ para la minimización. El esquema para este problema será una lista de tareas en la cual cada una de ellas tendrá una variable. Por ejemplo colocar ejes sería dos variables TejesD,TejesT, uno para el delantero y otro para el trasero. Además, tendríamos dos variables más: Tin (tiempo de inicio) y TF tiempo final (sería cuando se realiza la última inspección).

Para las restricciones colocamos la suma de los tiempos dependiendo de estas. Por ejemplo, si tenemos que las ruedas traseras se colocan en orden después del eje trasero lo que haremos será TiempoRueda1>=TiempoEjesTrasero+1(Tiempo que tarda una rueda). Para todas las restricciones de orden lo haremos igual: tarea X es igual o mayor a la anterior más el tiempo de realización de la tarea X.

Para la restricción en la que tenemos un recurso (manómetro) necesario para una tarea, lo que haremos será ver qué tareas se necesitan hacer primero y cuales después. Es decir, si tenemos que revisar una rueda y al no poderse revisar varias simultáneamente (solo tenemos un manómetro), lo que haremos será poner en orden las ruedas, ocupando el primer lugar aquella que acabe antes (una de las dos del eje trasero el cual se ha puesto primero). En el caso de que tengamos dos que acaben a la vez, se empieza con una de ellas y se continúa inmediatamente con la otra.

Planificación de tareas II (lectura de periódicos)

En este problema la dificultad añadida al anterior es que tenemos los periódicos como recursos que solo pueden utilizar una misma persona. Para ello he utilizado funciones encontradas en los ejemplos de la pág. web de eclipse-clp que nos suministraron en la primera clase de prácticas:

- Disjunctive(?Starts, +Durations, ?Flags) esta función la utilizamos para que las tareas (que son de un mismo periódico) no se superpongan, es decir, que no se esté leyendo un mismo periódico a la vez por dos personas.

- Cumulative(+StartTimes, +Durations, +Resources, ++ResourceLimit) con esta función conseguimos que cada lector lea un único periódico en el momento.

- Por último, tenemos funciones para exponer los datos en pantalla, meterlos en una lista y devolverlos por la variable *Vars*.

Calendario de exámenes

En este ejemplo utilizamos el mismo esquema que en el primer ejercicio pero con una forma diferente. Tenemos una lista de listas, de esta manera:

Dias = [[Dia1],[Dia2]]

Dia1= [_,_],[_,_](donde cada una de estas listas es un turno: mañana o tarde)

En la primera posición de esta lista de día estarán los alumnos y en la segunda el número de asignatura. A los alumnos los utilizaremos para imponer las restricciones. Emplearemos los siguientes predicados para resolver el problema:

- Pre ([ListaAlumnos,NumeroExamen],Dias) este predicado instanciará cada examen a una posición (día y turno) en la lista días.

- Bien (Dias), esta función comprueba las restricciones mediante el predicado *comprueba*, que nos verifica si en un par de turnos contiguos un alumno tiene dos exámenes. Esto lo realizamos mediante la función *probar*, que mira si los alumnos de un examen no están en el examen del turno siguiente. Para esto utilizaremos *notmember* que nos dice si un elemento no es miembro de una lista

- Utilizo las ‘,’ que son un *and* en prolog para hacer que las asignaciones iniciales (pre) estén bien hechas, puesto que bien() es true.

Bibliografía

- http://www.eclipseclp.org/
- http://www.eclipseclp.org/examples/index.html
- http://www.cs.us.es/~jalonso/cursos/d-pl-04/temas/tema-8.pdf
- Apuntes del acceso identificado (decsai)
- Apuntes de Prolog (asignatura Programación declarativa)