Máster Universitario en
Lógica, Computación e Inteligencia Artificial
Aprendizaje Automático

Apellidos: .....

Nombre : .....

Vamos a realizar ejercicios de Programación Lógica Inductiva con Aleph. Aleph puede descargarse de la página de Aleph. El programa original fue escrito por Ashwin Srinivasan para un compilador de Yap Prolog (Yet Another Prolog) que puede obtenerse en la página de YAP Prolog, aunque también podía ejecutarse en SWI Prolog (descargable desde aquí). Con el tiempo, YAP Prolog quedó obsoleto y las nuevas versiones de SWI Prolog habían cambiado algunos de los predicados, por lo que el programa Aleph no podía ser utilizado sin modificar el código. A la vista de estas diicultades, el profesor Fabrizio Riguzzi de la Università degli studi di Ferrara ha reescrito el código original de Aleph para las últimas versiones de SWI-Prolog. El manual de Aleph puede encontrarse en su página original, aunque también puede ser útil el documento The Aleph system made easy escrito por João Paulo Duarte Conceição. Para realizar esta práctica

- Descarga e instala SWI-Prolog de su página oficial.
- Descarga de versión de Aleph de Fabrizio Riguzzi de aquí. Junto a este documento pdf que estás leyendo, también has descargado esta versión de Aleph.

En la versión original, para realizar aprendizaje necesitábamos suministrar a Aleph tres ficheros diferentes, uno con extensión `.b`, donde se definía el conjunto de hipótesis y el conocimiento base, otro con extensión `.f` donde se proporcionan los ejemplos positivos y un tercero con extensión `.n` con los ejemplos negativos. En la versión de Fabrizio Riguzzi se sumninistra un único fichero con extensión `.pl` con la siguiente estructura:

- Cabecera: Que incluye la carga del módulo, la inicialización del programa Aleph y la definicón del espacio de hiótesis. Esta definición de hace mediante el uso de parámetros (usando `set` si se quieren cambiar el valor fijado por defecto, las definiciones de modo y las determinaciones.
- El conocimiento base, que debe ir entre las directivas `:- begin_bg` y `:- end_bg`.
- El conjunto de ejemplos positivos, que debe ir entre las directivas `:- begin_in_pos` y `:- end_in_pos`.
- El conjunto de ejemplos negativos, que debe ir entre las directivas `:- begin_in_neg` y `:- end_in_neg`.
- La directiva `:-aleph_read_all` que cierra el fichero.

Puedes ver las distintas secciones en el fichero `trenes.pl`, donde se incluye la información de *Los trenes de Michalski*. Vamos a usar este fichero en nuestra primera prueba. Copia el fichero `aleph.pl` en el mismo directorio de los ficheros y llama a Prolog.

```
$ prolog
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

A continuación leemos el fichero `trenes.pl`.

```
?- [trenes].

A L E P H
Version 5
Last modified: Sun Mar 11 03:25:37 UTC 2007

Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph/index.html

true.

?-
```

Aleph ya está preparado para buscar una solución al problema de aprendizaje. Esto se hace con el predicado `induce(_)`, pero antes vamos a ver cómo es una `bottom clause`. Podemos pedir a Aleph que sature un ejemplo concreto y nos devuelva el resultado de la saturación, esto es, la cláusula más específica que cubre el ejemplo dado según nuestro espacio de hipótesis. Esto es hace con el predicado `sat(+N)` donde `N` es un número natural. Si tecleamos `sat(N)`, Aleph saturará el `N`-ésimo ejemplo. Probemos con el cuarto.

```
?- sat(4).
[sat] [4]
[eastbound(east4)]

[bottom clause]
eastbound(A) :-
   has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
   short(E), short(D), short(C), short(B),
   closed(D), open_car(E), open_car(C), open_car(B),
   double(C), load(E,rectangle,1), load(D,rectangle,1), load(C,triangle,1),
   load(B,triangle,1), wheels(E,2), wheels(D,2), wheels(C,2),
   wheels(B,2).
[literals] [22]
[saturation time] [0.0013299110000000017]
true.
```

Como advertimos, la `bottom clause` de un ejemplo consta de muchos literales. En el proceso de aprendizaje Aleph tiene que decidir qué subconjunto de los literales

del cuerpo de la primera bottom clause toma la mejor puntuación e incorpora la cláusula obtenida al programa de salida. Aleph empieza eligiendo aleatoriamente un ejemplo, por eso a veces obtenemos programas diferentes a partir de los mismos ficheros de entrada.

Si has entendido el proceso de saturación, podemos continuar. En esta primera aproximación sólo falta pedirle a Aleph que realice el proceso de aprendizaje. Esto lo hacemos con el comando `induce(_)`. Nos da mucha información, sobre el proceso pero al final nos da la teoría encontrada, que en este caso consta de una única regla que cubre a los 5 ejemplos positivos y ninguno de los neghativos.

```
[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]
eastbound(A) :-
   has_car(A,B), short(B), closed(B).

[time taken] [0.006899509999999998]
```

A continuación proponemos los siguientes ejercicios.

Ejercicio 1. Para este ejercicio suministramos el fichero `leer.pl` donde aparece la tabla correspondiente a un conjunto de entrenamiento. Se trata de determinar si un artículo que recibimos lo leemos o nos lo saltamos en función de cuatro atributos (Ejemplo tomado del libro *Computational Intelligence A Logical Approach* de Poole, D., Mackworth, A. y Goebel, R. Oxford University Press, New York, 1998.). En el fichero puedes encontrar los ejemplos positivos, negativos y el conocimiento base que describe la tabla con los predicados binarios `autor/2`, `tema/2`, `longitud/2`, `sitio/2` con los argumentos ¡ejemplo,valor¿. Además suministramos los tipos: `ejemplo, autor, longitud, tema` y `sitio`.

Se pide fijar los parámetros, declaraciones de modo y determinaciones necesarias para realizar el proceso de aprendizaje. Se espera que la teoría aprendida sea similar a la siguiente:

```
[theory]
[Rule 1] [Pos cover = 7 Neg cover = 0]
leer(A) :-
   tema(A,nuevo), longitud(A,corto).

[Rule 2] [Pos cover = 6 Neg cover = 0]
leer(A) :-
   autor(A,conocido), longitud(A,corto).

[time taken] [0.003770340000000011]
[total clauses constructed] [15]
true.
```

Ejercicio 2. En este ejercicio suministramos los resultados de un experimento sobre 20 individuos (numerados del 1 al 20). En algunos hemos obtenido resultado positivo y otros no. Los ejemplos están en el fichero `ej_2.pl`. Vamos a pedir a

Aleph que realice síntesis del conocimiento e intente dar una descripción más corta de los datos obtenidos. Para ello debes suministrarle como conocimiento base las definiciones de los predicados `mayor_o_igual_que` y `menor_o_igual_que` y vamos a ver si Aleph es capaz de acotar los intervalos correspondientes a las instancias positivas. El ejercicio consiste en completar el fichero `ej_2.pl`. Se espera obtener como resultado del aprendizaje una teoría equivalente a:

```
[theory]

[Rule 1] [Pos cover = 4 Neg cover = 0]
inter(A) :-
   mayor_o_igual_que(A,3), menor_o_igual_que(A,6).

[Rule 2] [Pos cover = 3 Neg cover = 0]
inter(A) :-
   mayor_o_igual_que(A,11), menor_o_igual_que(A,13).

[Rule 3] [Pos cover = 1 Neg cover = 0]
inter(17).
```

Ejercicio 3. En este ejemplo vamos a suministrar a Aleph como ejemplos positivos y negativos los diez primeros números pares y los diez primeros números impares con la notación del sucesor. Además declaramos que los 20 primeros números naturales son de tipo "n". Se pide completar el fichero `ej_3.pl` para poder aprender la definición de número par. ¿Qué conocimiento base será necesario en este caso?