TUTORIAL DE PLMAN Y PROLOG

PRÁCTICAS DE MATEMÁTICAS-1

OBJETIVO: Implementar procedimientos en lenguaje Prolog para que resolver mapas con Plman.

Página principal de Plman **http://logica.i3a.ua.es**

- Usuario: identificador (aaa) que hay delante del correo de UA (**aaa**@alu.ua.es).
- Password : DNI o NIE con la letra en mayúscula.

Contenido de la web

- Inicio: página principal con toda la información de la asignatura.
- Entrega de prácticas: muestra la nota acumulada hasta el momento y las distintas fases.
- Datos de usuario: nombre y email.
- Descargas: material para prácticas.
- Rankings: media de puntos de participantes por grupos.

En el link **Descargas** se encuentra la carpeta **plman** con todo el material software para hacer la práctica:

- Carpeta **docs**/ que contiene manual de plman.
- Carpeta **maps**/ con 12 mapas de ejemplos.
- Carpeta **plman-game**/ que contiene el código fuente del juego.
- Script **plman** que interactúa con el sistema operativo y con el usuario para compilar y ejecutar el código que resuelve los mapas que debe recorrer Plman .

EVALUACIÓN- enero [40p]

Ejercicios del juego Plman **M** : [36p]

Control para validar M **C** : [4p]

Nota de prácticas P = M + C. Si C < 2 → M = 0.

EVALUACIÓN- julio [40p]

Nota de prácticas P = M + C'.

M: Nota obtenida en los ejercicios de Plman durante el curso : [20p].

C': Examen de recuperación de Plman : [20p].

OjO: M no recuperable en julio, se mantiene nota de enero hasta máximo de 20p.
Si C' < 10p → M = 0.

1 INTRODUCCIÓN

Sólo a los lógicos puros –que son muy pocos– les interesa la lógica por sí misma. La mayoría de los que estudian lógica se interesan por sus aplicaciones pues saber aplicar la lógica consiste, sobre todo, en saber probar si una sentencia dada es deducible de otras, concepto imprescindible en programación.

En las clases de teoría aprenderéis a representar conocimiento y a obtener, de manera intuitiva y formal, nuevo conocimiento a partir de otro conocido. Con las clases de prácticas tendréis la posibilidad de favorecer este aprendizaje de manera automática usando material software.

Con esto, las prácticas de -Matemáticas 1- están enfocadas a la programación de procedimientos para jugar con el videojuego Plman (abreviatura de Prolog-Man) usando el lenguaje de programación lógica, Prolog.

2 PRESENTACIÓN DEL JUEGO PLMAN

El juego lógico Plman, similar al archiconocido juego del "comecocos", consiste en que un comecocos llamado plman, representado por el símbolo @, se mueve por mapas que contienen cocos comiéndoselos y evitando que lo maten. El principal objetivo de esta práctica es el aprendizaje de la deducción automática relacionada con la inferencia lógica de primer orden usando el lenguaje Prolog.

- **¿Qué hay que hacer exactamente?**

Hay que resolver una serie de mapas de dificultad creciente hasta obtener la nota deseada. Un mapa está resuelto si plman ha conseguido comerse todos los cocos.

3 SOBRE LOS MAPAS QUE HAY QUE RESOLVER

Los mapas se clasifican en **5 fases** de dificultad creciente, son rectangulares, limitados por paredes (#), y contienen caracteres como cocos (.), objetos (-, +, …) y enemigos (E).

Número de mapas que se deben resolver de cada fase y nota máxima.		
Fase 0 (Tutorial)	5 mapas	(Nota máxima 0,5)
Fases 1 y 2	6 mapas	(Nota máxima 4)
Fase 3	2 mapas	(Nota máxima 4,55)
Fase 4	1 mapa	(Nota máxima 2,93)
Total	Hasta **14 Mapas diferentes**	

Dentro de cada fase (menos fase 0) los mapas están catalogados en niveles de **dificultad del 1 al 5**. Hay que resolver varios mapas de cada fase, pero no es obligatorio hacer un número determinado de fases de forma que cada estudiante puede llegar hasta la nota que quiera.

FASE	DIFICULTAD				
	D1	D2	D3	D4	D5
0	0,100				
1	0,350	0,450	0,500	0,550	0,650
2	0,525	0,675	0,750	0,825	0,975
3	1,225	1,575	1,750	1,925	2,228
4		2,025	2,250	2,475	

4 DESCRIPCIÓN DE LAS FASES

Fase 0: mapas para iniciarse a la resolución de mapas y al funcionamiento de Plman. El jugador se familiariza con el uso de la regla **do/1** y con las posibles acciones.

- Compilar y ejecutar programas en Prolog. Poner "en marcha" Plman en el mapa. Usar write.

Fase 1: mapas de tipo determinista que son mapas en los que el comportamiento de los objetos y enemigos siempre es el mismo en cada ejecución, es decir, si en el mapa hubiera un enemigo que se mueve éste siempre hará el mismo movimiento cada vez que Plman comience a jugar. Se aprende a usar la regla **see/3** que permite a Plman recorrer el mapa mirando y condicionar el movimiento. Los mapas de esta fase se deben resolver fácilmente usando sólo los predicados see/3 y do/1.

Fase 2: dificultad media en mapas deterministas. En cada ejecución los objetos siempre comienzan en la misma posición, aunque se pueden mover.

- Conocer las variables, havingObject, appearence.

Fase 3: aparece indeterminismo. Los enemigos y objetos móviles pueden tener comportamientos diferentes y aparecer en lugares aleatorios en cada ejecución. Se introducen los predicados dinámicos y el uso de subreglas. En todos los mapas hay algún componente indeterminista y algún objeto para coger/usar.

Fase 4: dificultad avanzada. Mapas con alta componente de incertidumbre. Puede requerirse la utilización del sensor de lista en el predicado see/3. Todas las características anteriores están presentes en los objetos, entidades móviles y enemigos, pero con indeterminismo. Se desarrollan conceptos avanzados y creatividad.

5 ENTORNO DE TRABAJO PARA HACER LA PRÁCTICA

- **Sistema Operativo: Linux-Ubuntu**.

  Si necesitas crear una máquina virtual en la asignatura Programación 1 os indicarán la forma de hacerlo.

- **Intérprete SWI_Prolog**

  Como los procedimientos de Plman se implementan con el lenguaje Prolog se debe instalar su interfaz, SWI-Prolog. En Ubuntu está la última versión de este intérprete que se descarga desde la consola:

  $ sudo apt-get install swi-prolog*

  Si no te permite hacer la instalación completa actualiza la versión de SWI-Prolog: **$ sudo apt-get update** y luego repite la descarga.

Comprueba que la instalación se ha efectuado correctamente abriendo un terminal y tecleando **$ swipl.** Debe aparecer el **shell "?"** en una nueva ventana. (Ver: "Instrucciones_instalar_SWIProlog_Ubuntu.pdf" y para ordenadores MAC ver: "Instrucciones_instalar-SWIProlog_Mac.pdf").

- **Sistema online para la entrega y evaluación de prácticas:** http://logica.i3a.ua.es
  - En la fecha indicada descargarás desde el sistema uno de los mapas que tendrás que resolver.
  - Desde la Fase 1 puedes elegir un mapa con una determinada dificultad.
  - Descargado el mapa **NO SE PUEDE CAMBIAR** por otro. Para descargar otro mapa debes resolver el anterior descargado como mínimo hasta el 75%.
  - Programada la solución la entregas al sistema para su corrección.
  - Antes de entregar la solución corrígelo de forma "local" en tu ordenador con el mismo corrector que el sistema on-line.
  - Cuando entregas la solución de un mapa al sistema éste lo corrige y te comunica la nota obtenida de forma automática, en tiempo real, incluyendo un informe con las estadísticas de ejecución de la corrección, mostrando el porcentaje de cocos consumidos, número de movimientos realizados y las acciones erróneas e incorrectas efectuadas.
  - Puedes entregar la solución de un mapa tantas veces como quieras hasta que obtengas el 75% de su resolución, en ese momento sólo puedes entregar 3 soluciones. La nota obtenida será la mayor de todas las soluciones.
  - La resolución de un mapa se mostrará en un entorno gráfico según las instrucciones programadas.

• **Carpeta plman**

Contiene el código completo de Plman junto con un script, plman, con el cual se realiza la compilación y ejecución de la práctica.

6 PASOS PARA RESOLVER UN MAPA

- Comprueba la instalación del intérprete SWI-Prolog en Linux-Ubuntu:
  - Para abrir la consola del intérprete, teclea en un terminal: **$ swipl**.
  - Para cerrar la consola, teclea: **? halt.**
- Descarga de la web de prácticas la **carpeta plman**. Comprueba que está el **script** plman.
- Descarga del sistema el **mapa a resolver**, por ejemplo, **mapa1.pl**, en la carpeta plman.
- Abre un terminal y accede a la carpeta plman.
- Crea un fichero, p.ej., **solucion.pl** (siempre con extensión .pl) para escribir la solución al mapa. Usa el editor de texto "**gedit**":

  …Escritorio/plman **$ gedit solucion.pl &**

>> en lo siguiente el mapa descargado (mapa1.pl) y el fichero solución (solucion.pl) se encuentran en la carpeta de plman donde se encuentra el script, plman.

- Para **ejecutar** la solución a un mapa escribe en un terminal el comando:

$ ./plman mapa1.pl solución.pl

- La **primera línea** del archivo solución (solución.pl) siempre debe tener la siguiente regla sin cabeza que permite cargar el módulo que contiene el juego:

```
:- use_module('pl-man-game/main').
```

7 USO DEL SCRIPT PLMAN

El script plman compila el archivo del mapa que se debe resolver y lanza su ejecución con trazado paso a paso (keypress) y con visualización activada.

El prototipo del script **plman** es:

```
$ ./plman mapa.pl fichero_solucion.pl [PARAMETROS]
```

mapa.pl: nombre o ruta del fichero que contiene el mapa a resolver.

fichero_solucion.pl: nombre o ruta del fichero que contiene el código en Prolog que resuelve el mapa.

El comportamiento del script se puede cambiar con los PARAMETROS opcionales siguientes:

-d S --delay S

Cambia el modo de ejecución paso a paso por un modo de ejecución continua con S segundos de espera entre movimientos.

-h --help

Muestra esta pantalla de ayuda.

-l LF --log-file LF

Indica a Plman que cree un log de la ejecución y lo guarde en el archivo LF para poder reproducirlo en un futuro si es necesario.

-m MV --max-movements MV

Ejecuta Plman restringiendo la cantidad de movimientos disponibles para resolver el mapa a MV movimientos como máximo.

-n --no-draw

Ejecuta Plman sin visualización. Imprime por pantalla los mensajes de compilación y las estadísticas finales al terminar.

-o OF --output OF

Por defecto, una vez terminada la compilación de forma exitosa se ejecuta el archivo binario y después se elimina. El parámetro -o --output compila y genera un archivo binario de salida OF que no es ejecutado ni eliminado.

-r RF --replay RF

Activa el modo de reproducción de un log de ejecución anterior. El reproductor de logs no requiere los parámetros de fase, mapa ni regla; solo se limita a leer el log de ejecución anterior y reproducirlo.

-t TF --temporal-file TF

Especifica el nombre del archivo temporal donde se almacenarán los resultados de la compilación, sólo en el caso de que se produzcan errores o warnings.

8 EJEMPLOS DE EJECUCIÓN SEGÚN LOS PARÁMETROS USADOS EN EL SCRIPT

Una vez creado el archivo solución (solucion.pl) para un mapa (mapaej.pl) para probarlo debemos tener en cuenta la ubicación de ambos archivos. Si no están en la carpeta plman se pueden indicar al script las rutas relativas y absolutas de dichos ficheros aunque lo mejor es situarlos dentro de la carpeta plman/ al mismo nivel que el script. El comando ./plman -h ó ./plman --help muestra el prototipo de uso del script plman.

Se dan ejemplos del uso del script plman teniendo en cuenta sus parámetros y la ubicación de los ficheros mapa.pl y solución.pl.

Comando de ejecución: $ **./plman mapa.pl solucion.pl**

A. Para imprimir por pantalla sólo los mensajes de compilación y las estadísticas finales al terminar.

Comando de ejecución: $ **./plman mapa.pl solucion.pl -n**

B. En la misma carpeta donde estamos situados (plman/), generamos un archivo ejecutable (ej. sol_ej) que no se ejecuta ni elimina

Comando de ejecución: $ **./plman mapa.pl solucion.pl -o sol_ej**

C. El archivo solucion.pl tiene **errores y/o warnings** y queremos generar un archivo temporal (ej. tmp.txt) con la lista de errores para poder corregirlos.

Comando de ejecución: $ **./plman mapa.pl solucion.pl -t tmp.txt**

D. Generamos un archivo de log, **log.txt** para la solución anterior (el archivo se creará en la subcarpeta **logs/**). Si esta carpeta no existe hay que crearla primero.

Comando de ejecución: $ **./plman mapa.pl solucion.pl -l logs/log.txt**

E. Para reproducir el archivo **log.txt**, generado con el comando anterior, ejecutamos el comando:

Comando de ejecución: $ **./plman -r logs/log.txt**

F. Probamos con un **retraso** en la ejecución de 0.5 segundos entre movimientos.

Comando de ejecución: $ **./plman mapa.pl ssolucion.pl -d 0.5**

G. Queremos un máximo de 100 movimientos para resolver el mapa.

Comando de ejecución: $ **./plman mapa.pl solucion.pl -d 0.5 -m 100**

H. El siguiente comando considera que tanto el mapa como el fichero solución están en distintas carpetas p.ej., **/home/chus/Documentos/soluciones**, y mapa.pl, en **/home/chus/Documentos/maps**, con un **retraso** en la ejecución de 0.3 segundos entre movimientos.

Comando de ejecución:

$ **./plman/home/chus/Documentos/maps/mapa.pl**

/home/chus/Documentos/soluciones/solucion.pl -d 0.3

9 PLMAN EN FUNCIONAMIENTO

Lanzada una ejecución con visualización (sin el parámetro -n), nos aparece una ventana similar a ésta:

La ventana de ejecución de Plman presenta dos partes: en la parte izquierda (rectángulo violeta) se muestra el mapa y en la derecha (rectángulo gris) una ventana de mensajes al usuario según las acciones de Plman.

En los mapas se pueden distinguir: cocos (.), paredes (#), enemigos (E u otra forma) y Plman (@); las paredes, los cocos y Plman siempre se mostrarán en color verde y los enemigos y objetos se presentarán en otros colores; los cocos, las paredes y los objetos serán elementos inmóviles; Plman y los enemigos serán móviles; las paredes no se podrán atravesar; los enemigos son mortales de tal forma que si Plman entra en contacto con alguno de ellos muere y termina la ejecución.

Algunos ejemplos de mensajes que se mostrarán en la ventana inferior son:

- Ninguna de las cláusulas de tu regla puede evaluarse con éxito (todas las cláusulas de tu regla fracasan), por lo que Plman se queda sin capacidad de decisión:
  ADVERTENCIA: La regla de control de tu personaje ha fracasado

- Tu regla intenta que Plman se mueva hacia una posición que está ocupada por un objeto sólido (ej. pared):
  ADVERTENCIA: Tu personaje ha intentado moverse hacia una posición que está ocupada por un objeto sólido

- Tu regla realiza una acción que provoca que Plman entre en contacto con un enemigo u objeto mortal (ej. una explosión) y por tanto muera:
  1 : Oh no! Tu personaje no ha podido terminar este nivel!
  2 :Pulsa 'X' para continuar.

➔ Para **volver al terminal** :

- Si tu regla logra que Plman se coma todos los cocos del mapa: **pulsa la tecla 'X'**.
- Si quieres parar la ejecución en cualquier momento: pulsa la tecla **Escape 'Esc' y luego 'X'**.

Después de pulsar 'X' se mostrará una pantalla en el terminal con las **estadísticas de la última ejecución**. Se puede observar el estado final en que ha quedado Plman, los cocos que se ha comido, los movimientos realizados por Plman para comerse los cocos, las inferencias realizadas por Prolog, el tiempo de CPU consumido en la ejecución, las colisiones producidas con elementos sólidos (como paredes), los intentos de acciones fallidas (intentar coger un objeto que no se puede coger), acciones erróneas (mensaje 2) y fallos de la regla.

Penalizaciones:

- Colisiones: 0,25 puntos.
- Intentos de acción: 0,5 puntos.
- Acciones erróneas: 0,5 puntos.
- Fallos de regla: 0,5 puntos.

10 PONIENDO A PLMAN EN MOVIMIENTO

- **Acciones** que puede realizar Plman: **moverse**, **coger, dejar y usar objetos.**
  Predicado que permite realizar las acciones: **do/1** con un único argumento que es la acción a realizar en una dirección: up, down, left, right.

Acción	Comando	Ejemplo	Descripción
Moverse	move/1	do(move(up))	Plman se mueve hacia arriba 1 casilla en el siguiente ciclo de ejecución
Coger objeto	get/1	do(get(down))	Plman coge el objeto que se encuentra en la casilla inferior a él en el siguiente ciclo de ejecución. *Nota*: Sólo puede llevar un objeto
Dejar objeto	drop/1	do(drop(left))	Plman deja el objeto que lleva en la casilla de la izquierda a la que se encuentra en el siguiente ciclo de ejecución
Usar objeto	use/1	do(use(right))	Plman usa el objeto que lleva en la casilla sobre lo que haya a su derecha en el siguiente ciclo de ejecución

- **Sensor de visión**: Plman puede condicionar su movimiento dependiendo de lo que está viendo.

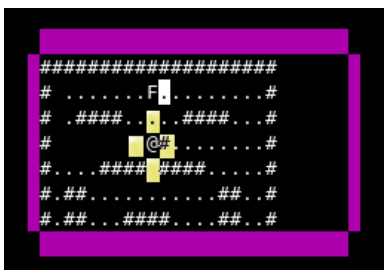
  Predicado de visión, que funciona como un sensor: **see/3**

  Sintaxis: **see(Sensor, DIR, OBJ).**

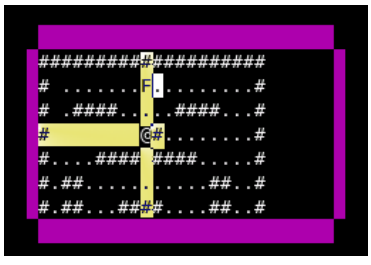
  Sensor: tipo de sensor que queremos utilizar (normal o list).

  - DIR: up, down, left, right, here, up-left, left-up, up-right, right-up, down-left, left-down.
  - OBJ: el resultado del sensor.

Ejemplo: en la siguiente Figura tenemos 2 mapas.



Sensor normal (see(normal, _, _))



Sensor de lista (see(list, _,_))

El sensor **normal** permite ver el contenido de las 8 celdas que Plman tiene a su alrededor y el de la celda donde se encuentra. El sensor de lista proporciona una lista con todo lo que se puede ver hasta el primer objeto

sólido. De esta forma, en el ejemplo de la Figura anterior, lo que se ve hacia arriba está en la lista ['.', 'F', '#'] y lo que se ve a la derecha es la lista ['#']. Más ejemplos del predicado see/3 para cada mapa:

```
?- see(normal, up, X).
X = '.'
?- see(normal, right, X).
X = '#'
?- see(normal, left, '.').
No
?- see(normal, down, ' ').
Yes
```

```
?- see(list, up, X).
X = ['.', 'F', '#']
?- see(list, right, ['#']).
Yes
?- see(list, left, [' ']).
No
?- see(list, down, X).
X=[' ', '.', '#']
```

11 ES MUY IMPORTATE QUE TENGAS EN CUENTA ESTAS NORMAS

1. Por el hecho de estar registrado y acceder a la web de prácticas de Plman te comprometes a realizar un **uso honrado** de la misma. Cualquier **plagio** que sea detectado por los profesores supondrá el **suspenso automático de la asignatura** durante el curso actual.
2. **No puedes elegir** dos mapas de la misma dificultad en una misma fase.
3. Cuanto **mayor sea el nivel** de dificultad de un mapa mayor es la puntuación que obtienes por resolverlo.
4. Elegido un mapa ya **no se puede cambiar**, debes resolverlo al menos, el 75% antes de pasar al siguiente.
5. Puedes entregar soluciones para un mismo mapa tantas veces como quieras hasta llegar al **75%** pero cuando sobrepases dicho porcentaje sólo dispondrás de **3 entregas más** de ese mapa para intentar llegar al 100%. Cada vez que entregues una solución a un mapa debes esperar **10 minutos** para entregar de nuevo.
6. Entregar nuevas soluciones **nunca** baja nota. La nota será la de la **mejor** solución que hayas entregado.
7. Aseguraos de que vuestras soluciones compilan bien (sin warnings ni errores), que la regla no fracasa, ni hace acciones imposibles o se choca con alguna pared, ya que el auto-corrector os **restará** nota.
8. Aseguraos que escribís bien el nombre de vuestra regla al auto-corrector, si no, no compilará.
9. **DEADLINE**: cada fase tiene fijada una **fecha de inicio y una fecha de fin** (deadline). No se podrá elegir dificultad para el primer mapa de una fase hasta que comience su fecha de inicio. La última solución a un mapa debe entregarse antes de cumplirse su fecha de fin. Las fechas de las fases se avisarán con la suficiente antelación por email, Campus Virtual y web del sistema.
10. **¡CUIDADO!** con la **fecha-hora máxima de entrega** de soluciones de mapas. Por ejemplo, si la fecha deadline para la Fase 1 es el 10 de noviembre entonces podrás entregar soluciones a mapas de dicha fase hasta el 9 de noviembre a las 23:59:59 h.
11. **¡PENALIZACIÓN!:** en el caso de **no entregar** las soluciones a los mapas de una fase en el plazo fijado para la misma se aplicará una **penalización del 25% en la nota** para todas las soluciones a los mapas que haya entregado de dicha fase, además de no disponer de la posibilidad de entregar más soluciones para dicha fase.
12. **¡IMPORTANTE**! Los **mapas de las fases son limitados**. Cada grupo de prácticas dispone de un número determinado de mapas de diferente dificultad en cada fase para que cada alumno resuelva un mapa diferente. Los alumnos elegirán los mapas disponibles en cada fase de la dificultad disponible, es decir, si un estudiante va a escoger el nivel de dificultad del 2º mapa de la Fase 1 y se acaban los mapas de nivel de dificultad 3, éste ya no tendrá la opción de escoger el nivel 3 para el segundo y sucesivos mapas de la Fase 1.

PROLOG

La programación de los procedimientos para resolver mapas con Plman se hará con el lenguaje de programación lógica Prolog bajo el entorno **SWI-Prolog.** SWI-Prolog es un intérprete Prolog de dominio público para ordenadores PC desarrollado en el Dept. of Social Science Informatics (SWI) de la Universidad de Amsterdam. Tanto el software, el manual de referencia (PostScript), como distinta información sobre el intérprete la podemos encontrar en: http://www.swi.psy.uva.nl/projects/SWI-Prolog . Es potente y flexible (permite integración con C) y se ejecuta sobre un entorno gráfico.

En Ubuntu se encuentra la última versión del intérprete que se descarga desde la consola:

$ sudo apt-get install swi-prolog*

Si no os deja hacer la instalación completa, actualiza con : **$ sudo apt-get update**
y luego repite la descarga. Comprueba que ya está instalado abriendo un terminal y tecleando **$ swipl**
Aparecerá el Shell "?" en una nueva ventana.

- **PROGRAMACIÓN EN PROLOG**

Debido a que la lógica de primer orden siempre ha sido uno de los sistemas más utilizados para representar y trabajar con problemas de razonamiento y con demostraciones de teoremas, en los años 60, Robinson demuestra la validez de una regla de inferencia, regla de resolución, con la idea de realizar demostraciones lógicas en sistemas automáticos. Nace así la programación lógica.

Un sistema automático de programación lógica, por excelencia, es el sistema Prolog. Prolog ("PROgrammation en LOGique") es un lenguaje de programación para ordenadores que se basa en el lenguaje de la Lógica de Primer Orden (restringido a cláusulas de Horn) para representar datos y conocimiento en los que entran en juego objetos y relaciones entre ellos. Tiene un papel importante en Inteligencia Artificial y su principal objetivo es el procesamiento automático del lenguaje natural.

En los años 70 fue creado por Alain Colmerauer (U. Marseille-Aix) y por Robert Kowalski (U. Edimburgo). Después en los 80 se extiende por Japón como base de los ordenadores de 5ª generación y en 1995 al ser normalizado con el estándar ISO se convierte en una herramienta de desarrollo de software de gran aceptación.

Bibliografía

- W.F. Clocksin & C.S.Mellish. **Programming in Prolog**. Springer-Verlag, 5ª ed., 2003.
- I. Bratko. **Prolog Programming for Artificial Intelligence**. Addison-Wesley, 4ª ed., 2011.
- R. O'Keefe. **The Craft of Prolog**. The MIT Press, Cambridge, MA, 1990.

Cursos de Prolog en web

http://cs.union.edu/~striegnk/courses/esslli04prolog/
http://www.cs.bham.ac.uk/~pjh/prolog_course/se207.html
http://homepages.inf.ed.ac.uk/pbrna/prologbook/

- **INTRODUCCIÓN AL LENGUAJE**

Existen problemas de razonamiento con n-hipótesis que se pueden escribir mediante un programa en Prolog que nos permita deducir de forma automática las proposiciones que son conclusiones deducibles de ellas. Dicho programa describe una Bases de Conocimiento del problema.

Un problema de razonamiento implementado en Prolog se muestra en el siguiente ejemplo.

Ejemplo-1: razonamiento implementado en lenguaje Prolog

"Del ayuntamiento de Corruptópolis conocemos que Mario es el alcalde; Juan su asesor; Ana y Luis son los concejales de urbanismo; Benito, Clara y Sonia los de festejos. El sueldo de todos los concejales es de 1000€ pero los de urbanismo cobran además un plus de 500€; el alcalde cobra 1000€ más que su asesor que cobra 1200€. En este ayuntamiento resulta que cualquier sujeto envidia a todos los que ganan más que él y todos los que envidian a alguien se han hecho corruptos".

Se debe averiguar (1) *"Quién es corrupto".*

Programa escrito en Prolog que constituye la Base de Conocimiento que define el problema

```
% alcalde(X):  el sujeto X tiene la propiedad de ser alcalde.
alcalde(mario).

% asesor(X):  el sujeto X tiene la propiedad de ser asesor.
asesor(juan).

% concejal_urba(X):  el sujeto X es concejal de urbanismo.
concejal_urba(ana).
concejal_urba(luis).

% concejal_fest(X):  el sujeto X es concejal de festejos.
concejal_fest(benito).
concejal_fest(sonia).
concejal_fest(clara).

%sueldo(X,Su):  El sujeto X tiene un sueldo de Su euros.
sueldo(X,Su) :- concejal_fest(X), Su=1000.
sueldo(X,Su) :- concejal_urba(X), Su=1500.
sueldo(X,Su) :- alcalde(X), asesor(Y), sueldo(Y,Su2), Su is Su2 + 1000.
sueldo(X,Su) :- asesor(X), Su=1200.

% envidia(X,Y): X envidia a Y si el sueldo Su1 de X es menor que Su2 de Y.
envidia(X,Y):- sueldo(X,Su1), sueldo(Y,Su2), X\=Y, Su1<Su2.

% corrupto:   El sujeto X es corrupto si X envidia a Y.
corrupto:- envidia(X,Y), X\=Y, write(X), write('envidia a'), writeln(Y),  write('el corrupto es'), write(X).
```

Para obtener la respuesta a la pregunta (1) se escribe en la ventana del sistema SWI_Prolog: **? corrupto.**

El sistema contestará con los nombres de los sujetos que verifican las condiciones anteriores.

Por ejemplo, X = benito.

- **RUTINA DE TRABAJO PARA ESCRIBIR Y EJECUTAR PROGRAMAS EN PROLOG EN EL ENTORNO SWI-PROLOG**

Para escribir y ejecutar programas Prolog usaremos:

PASOS PARA TRABAJAR CON PROGRAMAS PROLOG (no relacionados con PLMAN)

- Abrir un terminal e iniciar el sistema SWI-Prolog: **$ swipl**
- Abrir un fichero con extensión .pl **?- emacs('fichero.pl').**
- Compilar desde su barra de menú: **Compile Buffer**.
- Compilar desde el terminal usando el comando: **$ swipl –c fichero.pl**.
- Lanzar y compilar desde el terminal usando el comando: **$ swipl –f fichero.pl.**
- Para compilar un fichero Prolog y generar un ejecutable: **$ swipl -c fichero.pl -o ejecutable**
- Ejecutar el programa o preguntar por un objetivo: **?- objetivo.**

PASOS PARA TRABAJAR CON PROGRAMAS PROLOG (PLMAN)

- Situarse en la carpeta plman.
- Abrir el fichero donde escribiremos la solución del mapa: **$ gedit solucion.pl &**
- Descargar en la carpeta plman el mapa (mapa.pl) que se debe resolver.
- Escribir en la primera línea del archivo solucion.pl :- **use_module('plman-game/main').**
- Escribir los procedimientos que resuelvan el mapa.
- Para ejecutar la solución al mapa, escribir el siguiente comando en un terminal:

  $ ./plman mapa.pl solucion.pl regla

  donde regla: nombre del procedimiento principal que ejecuta la solución al mapa y que se encuentra en el fichero solución.pl

¡! SWI-Prolog no tiene interfaz, es todo en consola, aunque existe una interfaz gráfica: J Prolog-Editor, que se puede descargar desde el siguiente enlace:

http://www.trix.homepage.t-online.de/JPrologEditor/JPrologEditor.jar

*¡! Suponemos que **solucion.pl** está en el mismo directorio en el que estamos trabajando en el terminal, si no es así, debemos navegar hasta el directorio con **cd**, o añadir la ruta completa delante de **solucion.pl.***

- **ESTRUCTURA DE UN PROGRAMA EN PROLOG**

Un programa Prolog es un conjunto de hipótesis escritas mediante estructuras lógicas predicativas llamadas **hechos y reglas**, que definen una base de conocimiento (BC) para un determinado problema. Los hechos o átomos son sentencias que formalizan a las proposiciones atómicas y las reglas a las condicionales.

Las sentencias tienen la forma: **A :- B1,…,Bn,** con n≥0 y se conocen como **cláusulas de Horn.**

- Si **n=0** la sentencia es de la forma: **A**, y se denomina **HECHO**.
- Si **n>0** la sentencia es de la forma: **A:- B1,…Bn**, y se denomina **REGLA**.
- Las **preguntas** u objetivos tienen la forma: **A1, A2,…Am**, con m>0 (conjunción de hechos).

En un programa Prolog pueden aparecer líneas de comentarios que será texto escrito en líneas precedidas por **%**, o párrafo entre **/*** …. ***/.**

- **ELEMENTOS DEL LENGUAJE**

PREDICADOS

Toda sentencia Prolog está formada por predicados que son estructuras que determinan las propiedades de los objetos que intervienen en el problema y las relaciones entre ellos.

Un predicado se declara mediante: **nombre_predicado(arg1,arg2,...,argn).**

nombre-predicado: identificador elegido para denotar la propiedad o relación.

argi: objetos que están "afectados" por el predicado. Son los argumentos del predicado.

Ej. P1: Ana es hermosa.

Nombre del predicado: ser_hermosa

Sujeto afectado por el predicado: Ana.

Con la sintaxis de Prolog la proposición P1 se escribe: ser_hermosa(ana).

TIPOS DE PREDICADOS

- **de propiedad:** expresa la propiedad, característica o cualidad de un objeto.
  Sólo tienen un argumento (aridad 1).
- **de relación:** expresa la relación entre varios objetos.
  Tiene más de un argumento (aridad n>1).

SINTAXIS

- Primera letra del nombre del predicado en minúscula.
- Se pueden usar caracteres especiales: _, -.

ARGUMENTOS DEL PREDICADO: son los objetos que conforman el dominio del problema. Se clasifican en: constantes, variables y términos compuestos.

CONSTANTES: átomos y números.

Átomos: Representan a sujetos concretos del dominio. Son cadenas:
- Formadas por letras, dígitos y el símbolo _. Empiezan siempre por una letra minúscula.

Ej. Cadenas válidas: luis, luis1, luisLopez.

Ej. Cadenas no válidas: 1luis, Luis, _luis.

- o encerradas entre comillas simples.

Ej. 'Luis López', '28003 Madrid'.

Números: enteros y reales.
- En notación decimal es, por ejemplo, 0, 1.0, -320, 120000, etc.
- En notación exponencial, por ejemplo: 12.3E6, -0.123e+3, 123.0e-2. Es necesario que siempre haya por lo menos un dígito a cada lado del punto.

VARIABLES: representan a un objeto indeterminado del dominio. Se formalizan con cadenas formadas por letras, dígitos y el símbolo del subrayado _. Sólo pueden empezar por letra en mayúscula o "_". Las que tienen sólo este símbolo son variables anónimas y son útiles cuando su valor no es relevante en el programa.

Ej. Variables: **X, Sumando, Ana, _hola, _**

Se dice que una variable está:

- Instanciada: si existe un objeto concreto representado por ella.
- No instanciada: si la variable no tiene ningún valor constante.

TÉRMINOS COMPUESTOS: son estructuras que se construyen mediante un símbolo de función, llamado *funtor*.

Ej. libro(autor(pepito), titulo('Amanecer')).

- **SENTENCIAS DE UN PROGRAMA PROLOG: HECHOS Y REGLAS**

HECHOS: formalizan a las proposiciones atómicas.

- El nombre del predicado debe comenzar con una letra en **minúscula**. Al final se pone un punto.
- Entre paréntesis los argumentos estarán separados por comas.
- Si m-sujetos tienen la misma propiedad se deben escribir m-hechos.

Ej. En el Ejemplo-1 que hemos visto anteriormente algunos hechos son:

```
concejal_urba(ana).
concejal_urba(luis).
```

REGLAS: son hechos que dependen de otros hechos. Formalizan las implicaciones lógicas que tienen un sólo consecuente pero cualquier número de proposiciones en el antecedente. Son cláusulas de Horn que tienen como máximo un hecho atómico no negado:

¿Cómo se obtiene una regla ?

Dado un condicional $P_1 \wedge P_2 \wedge \ldots \wedge P_N \rightarrow Q \Leftrightarrow \neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_N \vee Q$ (por la regla: $A \rightarrow B \Leftrightarrow \neg A \vee B$)

La regla en Prolog es **Q:- P_1, P_2,….P_N**, donde Q y Pi son hechos (átomos).

- El símbolo **:-** es el "si" condicional.
- **Q** es la cabeza de la regla que se corresponde con el consecuente del implicador.
- **P_1, P_2,….P_N**. es el cuerpo de la regla que se corresponde con el antecedente del implicador. El antecedente puede ser una conjunción de condiciones que se denomina secuencia de objetivos. Cada objetivo se separa con una coma (conjunción lógica en Prolog).
- Pi pueden ser predicados predefinidos de Prolog.
- Las reglas pueden no tener cabeza (sólo cuerpo), reglas decapitadas que se ejecutan siempre.
- Los hechos de una regla pueden tener predicados con argumentos constantes y/o variables.
- Al final de una regla debe ir un punto (".").

Semántica: la regla **Q:- P_1, P_2,….P_N** indica que el objetivo Q es cierto **si** (:-) lo es cada uno de los objetivos P_i.

Ej. En el Ejemplo-1 que hemos visto anteriormente, reglas son:

```
sueldo(X,Su) :- concejal_fest(X), Su=1000.
sueldo(X,Su) :- concejal_urba(X), Su=1500.
```

Ejemplos de cómo se obtiene una regla Prolog a partir de una fórmula lógica predicativa

Ej. **S1: "Si Juan es profesor y empresario entonces es amigo de Luis".**

La formalización de S1 con el lenguaje de predicados es, por ejemplo:

Fbf-S1: Pr(juan) $\wedge$ Em(juan) $\rightarrow$ Am(juan,luis)

Aplicando equivalencias lógicas (A $\rightarrow$ B = ¬A v B), se obtiene la siguiente fbf:

Fbf-S1: ¬Pr(juan) $\vee$ ¬Em(juan) $\vee$ Am(juan,luis).

Esta fbf es una cláusula de Horn, luego se puede escribir en Prolog. Elegimos predicados.

```
% profesor(X): X es profesor;
% amigo(X,Y): X es amigo de Y;
% empres(X): X es empresario;
amigo(juan,luis) :- profesor(juan), empres(juan).
```

La regla que formaliza la sentencia S1 en Prolog contiene hechos con argumentos constantes (juan y luis).

Ej. **S2: "Cualquier profesor es amigo de Luis".**

La formalización de S2 con el lenguaje de predicados es, por ejemplo:

Fbf-S2: $\forall$x [Pr(X) $\rightarrow$ Am(X,luis)]

Aplicando equivalencias lógicas la regla Prolog es:

amigo(X,luis) :- profesor(X).

La cabeza de la regla está formada por un predicado de relación con un argumento constante (luis) y otro variable (X); el cuerpo está formado por un predicado de propiedad que tiene un argumento variable X. Para que sea cierto el hecho de la cabeza deben ser ciertos todos los hechos del cuerpo (esto significa, en este caso, que si no existiera un sujeto que fuera profesor en la BC sería "falso" que hubiera un sujeto que fuera amigo de Luis).

PREGUNTAS: Son hechos o conjunción de hechos que permiten deducir nuevo conocimiento de la BC. Se escriben en la ventana del intérprete con la sintaxis de los hechos de Prolog.

Ej. ? envidia(X,Y).

X = benito. Y=ana. Yes.

De una BC Prolog se puede deducir nuevo conocimiento que se conoce como la consecución de objetivos o la ejecución del programa. Si la pregunta no tiene éxito el sistema responderá con "no"; si tiene éxito devolverá un resultado que será una respuesta a la pregunta planteada.

CONECTIVAS LÓGICAS: Sólo aparecen en el cuerpo de las reglas y en las preguntas. Formalización:

- **Conjunción: ,** q , r.
- **Disyunción: ;** q ; r.
- **Negación: not** not(q).

- **PROCESO DE COMPUTACIÓN EN PROLOG AL HACER UNA PREGUNTA: RESOLUCIÓN Y REEVALUACIÓN**

  - Prolog utiliza encadenamiento hacia atrás con estrategia de control retroactiva sin información heurística (backtraking).
  - La obtención de resultados de un programa Prolog se hace a través de preguntas que son conjunción de objetivos que el sistema comprueba si son deducibles de la BC dando una respuesta.
  - Si la pregunta es deducible, Prolog informa del resultado y añade YES, en otro caso contesta NO.
  - La respuesta aparece en la línea siguiente a la pregunta.
  - El NO de la respuesta significa que no existe información en la BC.

  Para Prolog, contestar una pregunta es satisfacer la lista de objetivos que la define. Para ello, el sistema lleva a cabo una inferencia lógica sobre las sentencias del programa aplicando la regla de resolución, el mecanismo de unificación de términos y la estrategia de reducción al absurdo (refutación). El sistema acepta como ciertas las sentencias de la BC y falsa la pregunta que se le realiza que añade a dicha BC. Si el sistema encuentra una contradicción, dicha pregunta se deduce de la BC y Prolog responde con una respuesta afirmativa.

  En el Ejemplo 1 se hace la pregunta: **? sueldo(X,Y)** y usando "trace" se observa cómo Prolog busca la respuesta.

```
?- trace, sueldo(X,Y).

   Call: (9) sueldo(_G479, _G480) ? creep
   Call: (10) concejal_fest(_G479) ? creep
   Exit: (10) concejal_fest(benito) ? creep
   Call: (10) _G480=1000 ? creep
   Exit: (10) 1000=1000 ? creep
   Exit: (9) sueldo(benito, 1000) ? creep

X = benito
Y = 1000 ;
```

  Si queremos más respuestas, activamos la *Reevaluación*, pulsando la tecla "n". Prolog deja sin valor a las variables X, Y, y busca nuevos valores para ellas.

```
   Redo: (10) concejal_fest(_G479) ? creep
   Exit: (10) concejal_fest(ana) ? creep
   Call: (10) _G480=1000 ? creep
   Exit: (10) 1000=1000 ? creep
   Exit: (9) sueldo(ana, 1000) ? creep

X = ana
Y = 1000
```

- **PREDICADOS PREDEFINIDOS EN PROLOG**

Sobre directorios y ficheros: *¡! La notación: pred/n, indica que el predicado: pred, tiene n-argumentos.*

- **pwd/0.** Imprime el directorio de trabajo actual. Por ejemplo:

```
?- pwd.
:/prolog/marisol
```

- **ls/0. L**ista el contenido del directorio de trabajo actual. Por ejemplo, si el directorio actual contiene los ficheros 'patos.pl' y 'familia.pl' al ejecutar ls obtenemos:

  ?- ls.
  patos.pl familia.pl

- **cd/1.** Cambia el directorio actual. El nombre del nuevo directorio debe ser una ruta (absoluta o relativa) en notación Prolog, encerrada entre comillas simples. Por ejemplo:

  ?- cd('../pablo').
  establecerá c:\prolog\pablo como nuevo directorio de trabajo, mientras que:
  ?- cd('c:/prolog/marisol').
  Reestablece c:/prolog/marisol como directorio de trabajo.

De Entrada/Salida

- **write.**
  Sintaxis: write('Hello').
  Comillas simples para constantes; comillas dobles para listas; sin comillas y en mayúscula para variables.

- **nl.** Fuerza un retorno de carro en la salida. Por ejemplo:
  :- write('linea 1'), nl, write('linea 2'), tiene como resultado: linea 1 linea 2

- **read.** Lee un valor del teclado. La lectura del comando read no finaliza hasta que se introduce un punto ".".
  Sintaxis: read(X).
  Instancia la variable X con el valor leído del teclado. read(ejemplo). X toma el valor 'ejemplo'.

- **? listing.**
  Muestra en pantalla el contenido de la BC con todos los hechos y reglas con cabeza de todos los fichero cargados (las reglas sin cabeza de los ficheros anteriores no salen)

- **? listing(pred).** Muestra en pantalla el contenido del predicado: pred.

De depuración de programas

- **? trace.** Permite visualizar el recorrido del sistema Prolog cuando se le hace una pregunta.
- **? notrace**. Para salir de modo trace.

- **ERRORES COMUNES AL PROGRAMAR EN PROLOG**

Además de los errores normales de sintasis que Prolog anunciará indicando la línea del programa donde aparece, hay otros que son muy communes al empezar a programar, por ejemplo:

- Fichero.pl: Clauses of ----/nºarg not together of source file

Explicación: en un programa Prolog las cláusulas con el mismo nombre deben ir en orden correlativo, no se pueden intercalar cláusulas con otro nombre.

- Fichero.pl: Singleton variables [Y]

Explicación: la variable Y no toma un valor constante en la ejecución del programa.