Inteligencia Artificial
Práctica 6: Programación Lógica

Verónica Esther Arriola Ríos
Pedro Rodríguez Zarazúa
Luis Alfredo Lizárraga Santos

Fecha de entrega: Miércoles 6 de Marzo de 2016

1. Objetivo

Entender la manera declarativa de resolver problemas y crear código en el paradigma de la programación lógica.
Conocer los elementos básicos del lenguaje de programación Prolog.
Aprender a utilizar el intérprete SWIProlog para cargar un archivo, realizar consultas y ejecutar códigos.

2. Introducción

La programación lógica es un paradigma de programación basado en la lógica formal. Los programas son escritos como conjuntos de sentencias lógicas, expresando hechos y reglas sobre algún dominio del problema. Junto con un algoritmo de inferencia, forman un programa. El lenguaje de programación lógica más habitual es Prolog.

Las sentencias lógicas son comúnmente encontradas en la programación lógica como cláusulas de Horn, por ejemplo: $p(X, Y)\ if\ q(X)\ and\ r(Y)$

Las sentencias lógicas pueden entenderse de manera declarativa pero también pueden entenderse de manera procedural, por ejemplo: para resolver $p(X, Y)$, primero resuelve $q(X)$, después resuelve $r(Y)$.

3. Prolog

Es el lenguaje de programación lógica por excelencia. Tiene sus orígenes en la lógica de primer orden. Al ser declarativo, un programa lógico es expresado en términos de relaciones

representadas por hechos y reglas. Un cómputo es iniciado al correr una consulta sobre estas relaciones.

3.1. Tipos de datos

El único tipo de dato en Prolog es un *término*. Un término puede ser un átomo, números, variables o término compuesto.

- Un **átomo** es un nombre de propósito general sin significado particular. Por ejemplo: `x`, `azul`, `'Taco'`, `'algun atomo'`.
- **Números** que pueden ser flotantes o enteros.
- **Variables** formadas de letras, números y el caracter guión bajo (_) y que empiezan con letra mayúscula o guión bajo.
- Un término compuesto está formado de un átomo llamado "functor" y una cantidad de "argumentos" (que son nuevamente términos). Ejemplos de términos compuestos son: `ama(vincent,mia)` y `amigos(link, [tom, jim])`.

Un caso especial de término complejo son las listas.

- Una lista es una colección ordenada de términos. Es denotada por corchetes cuadrados con los términos separados por comas. La lista vacía es la única que no requiere usar comas y se denota por `[]`. Ejemplos de listas son: `[1,2,3]` y `[rojo, verde, azul]`.

3.2. Reglas y hechos

Los programas en Prolog describen relaciones por medio de cláusulas. Hay dos tipos de cláusulas: hechos y reglas. Una regla es de la forma:

```
Cabeza :- Cuerpo
```

y se lee como "Cabeza es verdadera si Cuerpo es verdadero". El cuerpo de la regla consiste de llamadas a predicados (metas de la reglas).

Cláusulas sin cuerpo son llamados hechos. Un ejemplo es:
`gato(tom).`
que es equivalente a la regla:
`gato(tom) :- true.`
ya que el predicado true siempre es verdadero.

Dado el hecho anterior, es posible preguntar:
¿Es tom un gato?
```
? - gato(tom).
true.
```

¿Qué cosas son gatos?
```
? - gato(X).
X = tom
```

Cláusulas con cuerpo son reglas. Un ejemplo de regla es:
```
animal(X) :- gato(X).
```

Si agregamos esta regla y ahora preguntamos:
¿Qué cosas son animales?
```
? - animal(X).
X = tom.
```

3.3. SWIProlog

SWIProlog es una implementación de Prolog muy utilizada y que servirá de intérprete para hacer consultas de un código en lenguaje Prolog. Prolog es multiplataforma y se puede instalar en cualquiera de los sistemas operativos Linux, Mac o Windows. Se puede obtener desde su página: http://www.swi-prolog.org/download/stable

Para quienes utilizan sistemas operativos basados en Debian (como Ubuntu), existe el repositorio swipl que puede instalarse mediante el comando de terminal:
```
sudo apt-get install swipl
```

Y se puede ejecutar el ámbiente de SWIProlog con la llamada:
```
swipl
```

La instalación en sistemas operativos Mac y Windows requieren la instalación del binario/ejecutable de la página.Los ejercicios de esta práctica requieren ser interpretados correctamente bajo el ambiente que ofrece SWIProlog.

3.4. Comandos útiles

3.4.1. Cargar un archivo

Hay varias maneras, una de ellas es desde la llamada a swipl con parámetros:
`$ swipl -s test.pl`
Donde `test.pl` es el nombre del archivo que contiene las cláusulas del programa y la carpeta desde donde se hace la llamada contiene dicho archivo.

Otras maneras de cargar un archivo dentro de SWIProlog son:
`?- consult('test').`
`?- ['test'].`

3.4.2. Salir de SWIProlog

La cláusula de paro de SWIProlog para volver a la terminal es:
`$ - halt.`

3.4.3. Base de conocimiento

Para consultar todas las cláusulas que se han cargado en SWIProlog como base de conocimiento para realizar consultas:
`?- listing.`

Si se tienen una colección muy grande de cláusulas se puede hacer una subconsulta de todas las apariciones de alguna en particular, al pasar el nombre de la cláusula como parámetro:
`?- listing(animal).`

Si se va a hacer cambios en archivos que ya fueron cargados, no es necesario volver a cargarlos si se emplea la cláusula:
`?- make.`
De esta manera se actualiza la base de conocimientos cargada en todos los archivos.

3.4.4. Unificación explícita

Para ver el proceso completo de unificación que realiza SWIProlog se utiliza:
`?- trace.`
El prompt de SWIProlog cambia y cada consulta se realize se mostrará cada uno de los pasos de unificación (útil para hacer debug o comprobaciones del código en tiempo de ejecución).

Por ejemplo:

```
?- trace.
true.

[trace] ?- woman(X).
    Call: (6) woman(_G386) ? creep
    Exit: (6) woman(mia) ? creep
X = mia;
    Redo: (6) woman(_G386) ? creep
    Exit: (6) woman(jody) ? creep
X = jody .
```

Para salir del entorno trace basta con escribir el comando:

```
[trace] ?- notrace.
true.
?-
```

4. Ejercicios

4.1. Ejercicio 1

Se tienen seis palabras italianas: astante , astoria , baratto , cobalto , pistola , statale. Se desean acomodar en un estilizado crucigrama como el siguiente:

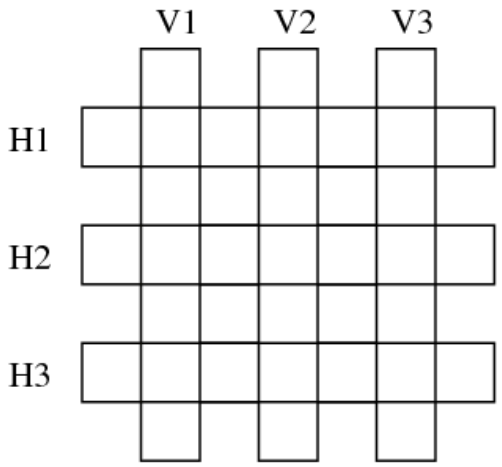

Figura 1

La siguiente base de conocimiento (cláusulas) representa a cada una de las palabras:

```
word(astante, a,s,t,a,n,t,e).
word(astoria, a,s,t,o,r,i,a).
word(baratto, b,a,r,a,t,t,o).
word(cobalto, c,o,b,a,l,t,o).
word(pistola, p,i,s,t,o,l,a).
word(statale, s,t,a,t,a,l,e).
```

Escribe un predicado `crossword/6` (con functor crossword de aridad 6) que nos diga cómo rellenar el crucigrama. Los primeros tres argumentos deben ser las palabras verticales de izquierda a derecha `(V1, V2, V3)`, y los últimos tres argumentos las palabras horizontales de arriba a abajo `(H1, H2, H3)`.

4.2. Ejercicio 2

El siguiente ejercicio consiste en saber si un cliente puede pagar de manera exacta cierta cantidad. Supongamos que un cliente tiene una bolsa con monedas de diferentes denominaciones representadas en una lista con repeticiones, por ejemplo `[1,1,5,5,5,10,20]`, el cliente puede pagar `16` pero no `18`.

Defina un predicado `cantex/2` que dada una lista de monedas y la cantidad a pagar, decide si se puede pagar de manera exacta.

Indique la especificación en español para la solución del problema mediante comentarios en el código fuente. Los comentarios se representan con el símbolo %.

4.3. Ejercicio 3

El teorema de los cuatro colores establece que cualquier mapa puede ser coloreado con cuatro colores diferentes, de tal forma que no queden regiones adyacentes con el mismo color. A partir de la siguiente figura, define un predicado `colorea(Reg1,Reg2,Reg3,Reg4,Reg5,Reg6)` el cual se cumple si `Reg1`, `Reg2`, `Reg3`, `Reg4`, `Reg5`, `Reg6` satisfacen el teorema de los cuatro colores.

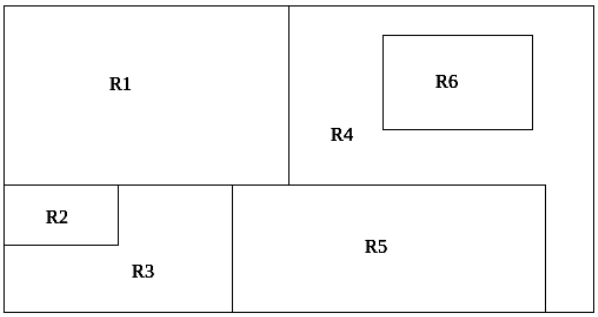

Figura 2

5. Notas adicionales.

La práctica es individual, anexen a su código un archivo readme.txt con su nombre completo, número de cuenta, número de la práctica y cualquier observación o notas adicionales (posibles errores, complicaciones, opiniones, críticas de la práctica o del laboratorio, cualquier comentario relativo a la práctica).
Pueden agregar cualquier biblioteca extra, sólo asegurense de que se encuentre bien comentada.

Compriman la práctica en un solo archivo (.zip, .rar, .tar.gz) con la siguiente estructura:

- ApellidopaternoNombreNúmerodepráctica.zip (por ejemplo: LizarragaLuis06.zip)
  - ApellidopaternoNombreNúmerodepráctica (por ejemplo: LizarragaLuis06)
    - src
      - plogica.pl
    - readme.txt

La práctica se entregará en la página del curso en la plataforma AVE Ciencias.
O por medio de correo electrónico a `luislizarraga@ciencias.unam.mx` con asunto `Práctica06[IA 2016-2]`

La fecha de entrega es hasta el día miércoles 6 de Marzo a las 23:59:59 hrs.