Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
1er cuatrimestre 2019

Fecha de entrega: 27 de junio

Introducción

En este trabajo vamos a modelar, utilizando Prolog, un juego de adivinanzas al estilo de los famosos *Adivina quién* y *Akinator*. En esta versión jugará una persona contra una computadora.

La mecánica es la siguiente. Al inicio del juego, la persona piensa en un personaje de algún entorno prefijado (por ejemplo, *personajes de Los Simpsons*, o *famosos argentinos*). El objetivo de la computadora es adivinar dicho personaje. Para ello, podrá hacer preguntas del estilo *¿Tiene el personaje {cierto atributo}?*, las cuales deberán ser respondidas por el jugador con *sí* o *no*. El juego termina cuando la computadora logra adivinar el personaje.

Preliminares

En esta oportunidad trabajaremos con predicados *dinámicos*. Estos son predicados que pueden modificarse durante la ejecución del programa.

Prolog cuenta con una *base de datos*, cuyo objetivo principal es preservar información ante el backtracking y poder modificarse en tiempo de ejecución. La primera regla de esta herramienta es la siguiente:

Si puede evitar usarla, considere fuertemente no usarla.

La base de datos es una extensión no-lógica de Prolog. Sus cambios no son revertidos al hacer backtracking, destruyendo un poco la belleza de los fundamentos de la programación lógica.

Un predicado dinámico se "define" utilizando `dynamic/1`. Por ejemplo:

```
:- dynamic predicadoDinamico/1.
```

Esta cláusula le indica al compilador que el predicado podría no tener cláusulas aún y se podrían agregar o quitar cláusulas en tiempo de ejecución.

Para restaurar la base de datos sobre un predicado se utiliza `retractall/1`. Por ejemplo: `retractall(predicadoDinamico(_))`. El metapredicado `call_cleanup/2` llama al segundo argumento sin importar cómo terminó el primero.

Los metapredicados `asserta` y `assertz` permiten guardar hechos y reglas en la base de datos. La única diferencia entre ambos predicados es en qué lugar de la base guardan las reglas: `asserta` lo hará en la primera cláusula y `assertz` en la última.

Utilizaremos dos predicados dinámicos `si/1` y `no/1` para que el programa guarde las respuestas dadas a las preguntas realizadas.

Ejercicios

En los siguientes ejercicios podrán usar ! (cut).

Ejercicio 1. Elegir al menos 10 personajes de una serie de televisión o internet. Dar un predicado `atributos(Personaje, Atributos)` que relacione cada personaje con su lista de atributos. Notar que, para que el juego tenga sentido, no puede haber dos personajes con el mismo conjunto de características. Para hacerlo más interesante, debe haber varios pares de personajes cuya intersección de atributos no sea vacía.

Ejercicio 2. El término `write(A)` siempre unifica y muestra el valor de `A` en pantalla. El término `read(R)` pide al usuario que ingrese un valor, el cual unificará con `R`. Con esto en cuenta, implementar los predicados `mostrarPregunta(+A)` y `leerRespuesta(-R)`, que se comportan como sus nombres lo indican.

Ejercicio 3. Dado el siguiente predicado:

```
pregunta(A) :- mostrarPregunta(A), leerRespuesta(R), guardarRespuesta(A,R).
guardarRespuesta(A, R) :- R == 'si', !, assertz(si(A)).
guardarRespuesta(A, R) :- R == 'no', !, assertz(no(A)).
guardarRespuesta(A, R) :- write('Respuesta inválida. Se pregunta nuevamente.\n'),
                          pregunta(A).
```

Implementar el predicado `satisface(+A)` que tiene éxito si se respondió "sí" a la pregunta sobre el atributo `A`, falle si se respondió "no" a la pregunta sobre el atributo `A`, o pregunte sobre el atributo `A` en caso de no haber realizado la pregunta aún. Notar que un goal `satisface(`a`)` debe unificar sólo con una de estas opciones.

Ejercicio 4. Dar el predicado `satisfaceAtributos(+AS)` que tiene éxito si cada atributo de la lista `AS` se satisface. En caso de no haber realizado la pregunta sobre un atributo de la lista, deberá hacerla.

Ejercicio 5. Dar el predicado `borraRespuestas/0` que permita borrar de la base de datos las respuestas dadas a las preguntas. Debe borrar todos los datos de los predicados `si/1` y `no/1`.

Ejercicio 6. Dar el predicado `adivinarPersonaje/0` que juegue a adivinar el personaje. El mismo deberá mostrar por pantalla el nombre de un solo personaje que cumpla con las características que indicó el jugador, y al final deberá borrar todas las respuestas guardadas para permitir jugar nuevamente.

Ejercicio 7. Jugar y probar varias combinaciones de atributos. ¿Qué ocurre cuando el juego no conoce el personaje? ¿Cómo influye el orden de los atributos en el predicado `atributos(Personaje, Atributos)`? Escribir un comentario con las respuestas.

Ejercicio 8. Dar un predicado `agregarPersonaje(+Personaje, +Atributos)` que sea capaz de agregar un personaje a la base de datos. Para esto es necesario que no exista otro personaje cuyo conjunto de atributos sea el mismo (sin importar el orden). Notar que ahora `atributos/2` tiene que ser dinámico. ¿Es lo mismo utilizar `asserta` y `assertz` en este caso? ¿Por qué?

Ejercicio 9. Modificar `adivinarPersonaje/0` para que sea capaz de agregar un personaje a la base de datos en caso de no adivinarlo. Se deberá pedir al usuario que ingrese el nombre del nuevo personaje y también una lista de atributos. Con esta información, sumada a la obtenida por las respuestas dadas a las preguntas que se hicieron, se guardará el nuevo personaje en la base de datos.

Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado.

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp3.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía → Programación Lógica*).

Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Cosas útiles* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.