Tarea Programada II

Elizabeth Córdoba
Eduardo Álvarez
Andrés Abarca
Josué Fallas
Marianela Cordero

15 de octubre del 2013

Tabla de Contenidos

Descripción del problema

La segunda tarea programada tiene como objetivo la implementación de un lenguaje de programación como lo es prolog, (básicamente un lenguaje funcional) en otro lenguaje de programación, en nuestro caso es java, un lenguaje orientado a objetos y en el sistema operativo Ubuntu de Linux.

El proyecto se divide en dos partes específicamente:

- **Administración de conocimiento:** En esta modalidad, permite agregar hechos o reglas a la base del conocimiento, siempre utilizando la forma estándar del lenguaje de programación prolog, esta es la siguiente:

  Para reglas: predicado (arg1, arg2,..., argn):-ant-1,..., ant-n.
  Para hechos: hecho (arg1, arg2).

  El usuario podrá agregar la cantidad de hechos o reglas deseadas y nuestro programa será capaz de guardar estas, en una base de conocimiento temporal, para luego poder ser evaluada por las consultas.

  En esta parte se acordó realizar una serie de predicados especiales propios de prolog (Built-in-predicates) como lo son los siguientes: write (arg), nl, fail, los cuales serán los únicos tomados en cuenta en nuestro sistema.

- **Modo consulta:** En esta parte se logra la interacción con la base del conocimiento, la cual evaluara y revisara los hechos y predicados digitados por el usuario, para así tener de resultado dos únicas respuestas: YES o NO.

  Para llevar a cabo esta parte se deberá investigar sobre el método backtracking que sirve para resolución de problemas, así como investigar librerías o métodos para realizar análisis léxico y semántico en el sistema operativo Ubuntu.

  Además se acordó realizar el desarrollo del programa a modo consola, es decir sin implementar algún tipo de interfaz gráfica.

Diseño del programa

El programa debe cumplir con 2 funcionalidades: agregar hechos o reglas y efectuar consultas sobre la base de conocimientos agregada anteriormente.

Se decidió realizar un menú para que el usuario decida qué actividad desea realizar, el menú ofrece: 1 para agregar reglas o hechos, 2 para realizar consultas y 3 para salir del programa.

En agregar consultas, el usuario debe ingresar un string que corresponde a una regla o un hecho según lo desee. Este string va a ser analizado primeramente por el analizador léxico, para garantizar que el usuario no haya ingresado un token incorrecto. Si el análisis léxico resulta exitoso se procede a realizar un análisis sintáctico, el cual busca verificar que la estructura de la regla o hecho sea la correcta.

Para el análisis léxico se utilizó la librería regex, la cual va a devolver un arreglo con los tokens identificados, este arreglo se examina para así saber si es una regla o un hecho, dependiendo del que sea se llama al análisis sintáctico correspondiente. El análisis sintáctico recibe este arreglo y lo recorre revisando ciertas condiciones. Si el análisis sintáctico es exitoso, entonces se continúa a almacenar en una lista doble el hecho o regla según corresponda.

A partir de estas listas se conforma la base de conocimiento.

La estructura de los nodos de estas listas se diseñó con el propósito de facilitar el manejo de la información para efectuar las consultas.

Librerías Utilizadas

- ***StringTokenizer:*** *La* clase *StringTokenizer* nos ayuda a dividir un string en substrings o tokens, en base a otro string (normalmente un carácter) separador entre ellos denominado delimitador.

- **Regex.Matcher:** Matcher se crea a partir de una compilación mediante la invocación de método matcher del patrón. Una vez creado, un matcher se puede utilizar para llevar a cabo tres tipos de operaciones:
  El método matches intenta hacer coincidir toda la secuencia de entrada contra el patrón.

  - El método lookingat intenta hacer coincidir la secuencia de entrada, empezando por el principio, contra el patrón.

  - El método find analiza la secuencia de entrada en busca de la próxima subsecuencia que coincida con el patrón.

  - Cada uno de estos métodos devuelve un booleano que indica el éxito o el fracaso. Más información sobre un partido de éxito se puede obtener mediante la consulta del estado del emparejador.

- **Regex.Pattern**: Es una representación compilada de una expresión regular, se especifica como una cadena, primero debe ser compilado en una instancia de esta clase. El patrón resultante entonces se puede utilizar para crear un objeto Matcher que puede coincidir con las secuencias de caracteres arbitrarios en contra de la expresión regular. Todo el estado implicado en la realización de un partido reside en el emparejador, tantos comparadores pueden compartir el mismo patrón.

Manual de Usuario

El programa consta de 2 funciones: Agregar un hecho o una regla y realizar consultas.

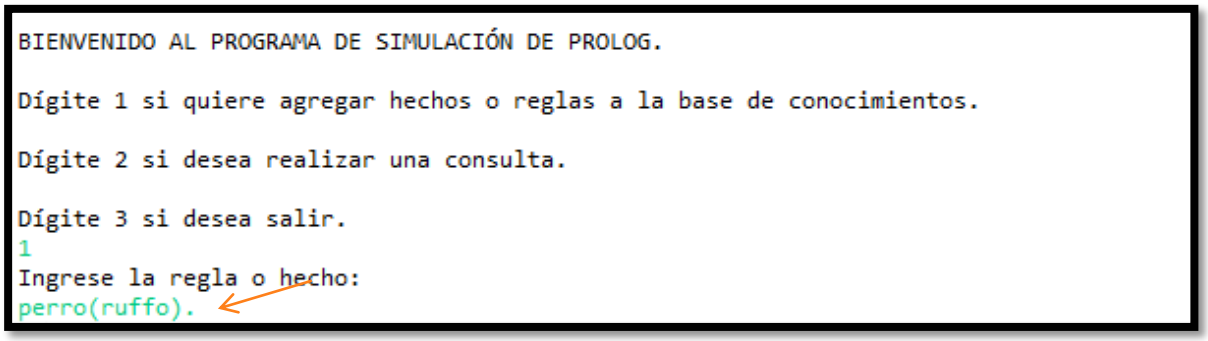
Para poder ingresar a cualquiera de estas funciones, el usuario debe de seleccionar en el menú la acción que quiere seguir:

```
BIENVENIDO AL PROGRAMA DE SIMULACIÓN DE PROLOG.

Dígite 1 si quiere agregar hechos o reglas a la base de conocimientos.

Dígite 2 si desea realizar una consulta.

Dígite 3 si desea salir.
```

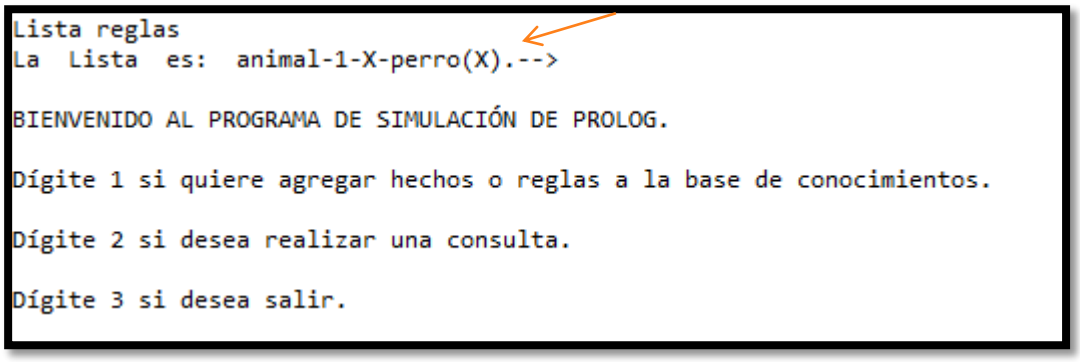
1. Al introducir un 1, el usuario puede elegir si quiere ingresar un hecho o una regla en la base de conocimiento. Si desea introducir una regla lo hace de la siguiente manera:

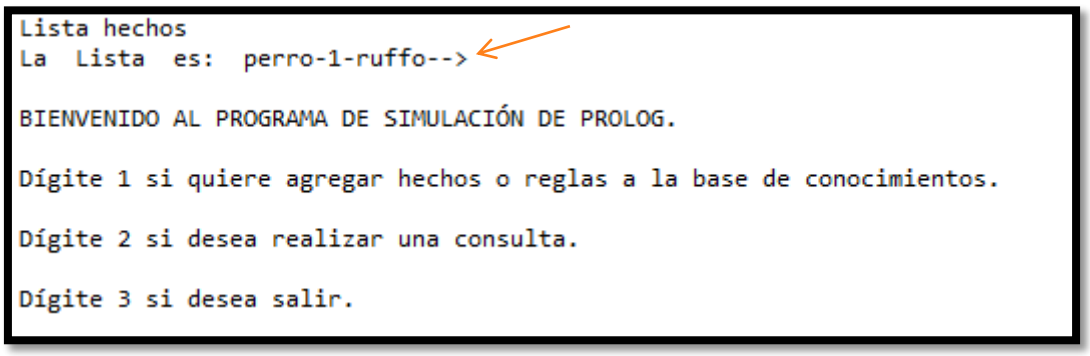
```
BIENVENIDO AL PROGRAMA DE SIMULACIÓN DE PROLOG.

Dígite 1 si quiere agregar hechos o reglas a la base de conocimientos.

Dígite 2 si desea realizar una consulta.

Dígite 3 si desea salir.
1
Ingrese la regla o hecho:
animal(X):-perro(X).
```

2. Pero si lo que desea ingresar es un hecho, lo requiere hacer de la siguiente forma:



Al ingresar la regla o el hecho el programa le va mostrando al usuario la base de conocimiento, donde se encuentran dichas rechas o hechos.





Asegúrese de ingresar correctamente los las reglas o los hechos, de lo contrario el sistema le mostrará algún mensaje de error:

1. Error al ingresar un hecho incorrectamente:

```
Ingrese la regla o hecho:
perro(ruffo.
Error, los argumentos deben terminar en )
Error sintactico, favor verifique que la estructura sea la adecuada.
BIENVENIDO AL PROGRAMA DE SIMULACIÓN DE PROLOG.

Dígite 1 si quiere agregar hechos o reglas a la base de conocimientos.

Dígite 2 si desea realizar una consulta.

Dígite 3 si desea salir.
```

2. Error al ingresar una regla incorrectamente:

```
Ingrese la regla o hecho:
mascota(X):-perro(X)
Correcto2
X
Error,falta . al final de la regla
Error sintactico, favor verifique que la estructura sea la adecuada.
BIENVENIDO AL PROGRAMA DE SIMULACIÓN DE PROLOG.

Dígite 1 si quiere agregar hechos o reglas a la base de conocimientos.

Dígite 2 si desea realizar una consulta.

Dígite 3 si desea salir.
```

Por ultimo si lo que desea es salir del programa, digite 3:

```
Dígite 1 si quiere agregar hechos o reglas a la base de conocimientos.

Dígite 2 si desea realizar una consulta.

Dígite 3 si desea salir.
3
FIN
```

Análisis de resultados

A continuación se especificaran los objetivos que se cumplieron en el proyecto programado II:

- Se lo logró implementar la sección de agregar hechos a la base del conocimiento, con sus respectivas validaciones. En este caso el usuario digita un hecho o una regla.
- Se obtuvo el analizador léxico y sintáctico correspondiente a los hechos o reglas.
- Se lograron las validaciones de reglas o hechos, por lo que si el usuario digita algo incorrecto el programa será capaz de identificar este error e informarlo.
- Se logró realizar el modo consulta solo con hechos, este será capaz de responder True o No, según sea el caso.

Objetivos que no se cumplieron:

- No es robusto, dado que cuando se presenta algún error en la evaluación del modo consulta.
- No se logró el modo consulta de las reglas, ya que nos hizo falta de tiempo para lograr implementarlo al 100%.
- El motor de inferencia no pudo concluirse, lo cual el sistema backtracking no se concluyó de manera efectiva, por lo que no se incorporó a la tarea.

Conclusión personal

En la Tarea Programada II, pusimos a prueba todos nuestros conocimientos sobre programación que hemos adquirido en otros cursos, además enfrentarnos al reto de implementar un lenguaje de programación funcional como lo es prolog, en otro orientado a objetos, en este caso java. También trabajamos con el sistema operativo Linux, específicamente con Ubuntu, el cual ha sido nuestra segunda experiencia.

Durante la creación o desarrollo de esta tarea implementamos algo distinto, que no solo se sale de algo estándar, sino que es un implemento e idea brillante, implementar un lenguaje escrito básicamente en otro lenguaje de programación, con su misma sintaxis e implementación lógica.

Esto nos permitió determinar conocimientos específicamente de prolog, ya que la idea es entender y poder implementar hechos y reglas de prolog en otro lenguaje, además del conocimiento y practica que nos brinda la implementación, sintaxis, análisis (léxico, sintáctico), que se tuvieron que entender y desarrollar.

En el desarrollo de esta tarea, reforzamos nuestra habilidad de trabajo en equipo ya que al trabajar en un grupo de 5 personas se dificulta un poco poder organizar las reuniones y poder adaptar los distintos puntos de vista de manera que se combinen de la mejor manera para obtener el resultado esperado. Pero al poder concluir exitosamente esta tarea, logramos superar la prueba y adquirimos experiencia tanto en Programación como en habilidades interpersonales, para saber trabajar en equipo.

Bibliografía

- García, A. F. (2000). *Curso del lenguaje Java.* Obtenido el 14 de octubre de 2013 de http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/colecciones/stringtokenizer.htm
- ORACLE. (1993). *Documentación Java: Class Matcher.* Obtenido el 15 de octubre de 2013 de Java™ Platform Standard Ed.7: http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html
- ORACLE. (1993). *Documentación Java: Class Pattern.* Obtenido el 15 de octubre de 2013de Java™ Platform Standard Ed.7: http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html