Universidad Nacional Autónoma de México
Facultad de Ciencias
Logica Computacional | Grupo: 7068

Proyecto Final Logica Computacional

Jacome Delgado Alejandro
Jimenez Sanchez Emma Alicia
Labonne Arizmendí Raúl Emiliano

9 de junio de 2024

1. Resumen:

En el mundo de la programación, existen diferentes paradigmas que nos ayudan a modelar y resolver problemas de una manera sencilla, cada uno con sus respectivas ventajas y desventajas, siendo los imperativos y declarativos; los imperativos incluyen uno de los modelos de programacion más famoso el cual es la Programación Orientada a Objetos que lenguajes como `C++`, `Java` o `Rust` se basan en este paradigma. Sin embargo también mencionamos el paradigma declarativo, el cual abarca la programación funcional, que lenguajes como `Haskell` y `LISP` se basan en ello, y la programación lógica que lenguajes como `Prolog` se basan.

En este proyecto nos adentraremos en la programación Lógica y en la utilidad de estos, empezando con definir la programación Lógica y desde como es que funciona, ¿Para que usarlo?, ¿En que cambia?, ¿Como se modela un problema en base a esta programación?, son preguntas que a lo largo de este proyecto iremos abarcando para terminar con un problema modelado desde el paradigma declarativo con el lenguaje de programación `Prolog`, y con esto poder apreciar cuales son las ventajas que conlleva resolver problemas con un paradigma declarativo en lugar de un paradigma imperativo.

2. Preliminares:

La programación lógica como explican Pascual Julián Iranzo y María Alpuente Frasnedo, se basa en la lógica de primer orden, es decir la lógica de predicados, la cual a traves de símbolos y cuantificadores abstrae relaciones del mundo real. Con el fin de poder abstraer problemas del mundo real que implica objetos y relaciones entre objetos. Su base es la forma de Clausulas de Horn, las cuales se emplean debido a su facil modelado y al tener una semántica operacional clara se puede realizar una implementación precisa y eficiente en los programas, además de tener como principio operacional la regla de inferencia por resolución SLD.

Se emplea una semántica declarativa basada en teoría de modelos tomando como dominio la interpretación de universo de discurso como un conjunto puramente sintáctico, es decir, es una forma de interpretar fórmulas lógicas sin necesidad de información o recursos externos.

Por otro lado, la resolución SLD fue dado por Maarten van Emden y Robert Kowalski, es un metódo fundamental de la programación lógica que a base de pruebas por refutación, en otras palabras, por contradicción; el cual busca las proposiciones que sean verdaderas a traves de una contradición. Además de ser un método correcto; garantizando que todas las respuestas sean válidas, y completo; si se puede encontrar todas las respuestas posibles, empleando el algoritmo de unificación para obtener, a traves de diferentes combinaciones, las respuestas deseadas de la consulta lógica. (Pascual Julián Iranzo, 2007).

La programación lógicas, en particular Prolog, es útil en la inteligencia artificial, sistemas expertos, procesamiento de lenguaje natural y construcción de interprete y compiladores, como es un lenguaje que se especializa en inferir datos a partir de una base de conocimientos, se puede implementar en general en

problemas donde se requiera una base de datos e inferir información a partir de ella. Como es el caso de **RFuzzy**, es una herramienta capaz de representar y trabajar lógica difusa, permitiendo manejar conceptos subjetivos, se ha utilizado en potenciar la inteligencia de los robots, un ejemplo sería su implementación en robots de la liga mundial de fútbol de robots (RoboCupSoccer). (Angel MEa, 2016)

Otra caso de su uso es la técnica **Cloze**, es una técnica a base de prolog para la inteligencia artificial y aprendizaje de lenguajes, consiste en la presentación de un texto con palabras sustituidas y según un criterio establecido previamente, se tiene que adivinar o proponer la palabra omitida a partir de las claves sintácticas o semánticas. En 2012 se libero un complemento para moodle, permitiendo que sea posible generar preguntas tipo *Cloze* en esta plataforma mediante una interfaz gráfica.(Angel MEa, 2016)

Por último, podemos ver el desarrollo de **Sistemas Multi-Agentes** elaborado mediante *JavaLog*, es decir, utilizando Java y el intérprete de ProLog. Es un sistema en el cual diferentes agentes inteligentes interactúan entre sí en lugar de un solo agente, logrando así poder resolver problemas difíciles como el modelador de estructuras sociales, respuesta a desastres y parte del comercio en línea. (Angel MEa, 2016) Por ejemplo *Javalog Analyzer* para Windows, el cual personaliza los registros de archivos en tiempo real con filtros de texto y colores. (ONWORKS, s.f.)

3. Implementación:

A continuación, se nos pide encontar una solucion e implementación del siguiente problema:

Considérese una caja cuadriculada rectangular de dimensiones NxM. Las cuadrículas de la caja pueden estar vacías o tener un pedazo de queso el cual puede tener veneno o ron. Se coloca un ratón en alguna cuadrícula (X, Y) en una dirección dada D. El ratón se mueve en la caja hasta encontrar la cuadrícula de salida, pero al encontrar un pedazo de queso sucede lo siguiente:

- Si el queso no tiene veneno ni ron, se lo come y se mueve en la misma dirección.
- Si el queso tiene ron se lo come y se emborracha, avanzando 7 pasos en dirección aleatoria.
- Si el queso está envenenado, se lo come sólo si está borracho y muere. Si el ratón está sobrio, no come el queso y avanza en la misma dirección.

Si el ratón llega a la pared de la caja y no puede moverse más en la dirección actual, entonces:

- Si está sobrio, gira a la izquierda y sigue caminando.
- Si está borracho, choca con la pared hasta que se le pase la borrachera.

lo primero que hacemos para resolver el problema es plantear los requerimientos. Lo primero que notamos es que se requiere de una cuadricula de dimensiones NxM, esto lo podremos modelar como una lista de listas. Ademas, en cada casilla puede haber un queso o la salida, por lo que la cuadricula debera tener marcado que contiene, por tanto, si pensamos las cuadriculas como una matriz, por ejemplo:

$$\begin{bmatrix} nada & queso(normal) & nada \\ queso(normal) & nada & nada \\ queso(ron) & nada & salida \end{bmatrix} \tag{1}$$

escribirlas en Prolog seria de la siguiente forma:

$[[nada, queso(normal), nada], [queso(normal), nada, nada], [queso(ron), nada, salida]]$ para que sea mas fácil la implementación, lo que haremos es tratar a la casilla que no tiene 'nada' y a la casilla que es la salida como tipos de queso, de esta forma abstraemos el tablero y ya no tenemos que preguntar si hay un queso en la casilla o no, así pues, en Prolog se modelan las casillas del tablero:

```
%% el queso puede terner distintos tipos de estados,
%% como consideraremos que el tablero va a tener quesos en todas la casillas (para facilitar la implementacion) se añadira el queso vacio.
%% Lo definimos a continuacion:
% - nor = Normal
% - ron = Ron
% - ven = Veneno
% - vac = Vacio
% - fin = salida

queso(nor).
queso(ron).
queso(ven).
queso(vac).
queso(fin).
```

Además del tablero, el ratón posee la propiedad de estar viendo a una "dirección", lo que va a determinar hacia adonde camina, las únicas direcciones a las que puede ver son: "Norte, Sur, Este y Oeste", asi pues al implementarlas en Prolog nos queda de la siguiente forma:

```
%% el raton puede estar viendo hacia distintos lados, que son:
% - no = Norte
% - su = Sur
% - es = Este
% - oe = Oeste

pos(no).
pos(su).
pos(es).
pos(oe).
```

ya con las direcciones definidas, se puede definir el movimiento del ratón, el cual depende en gran medida de como se ha abstraído el tablero. Recordemos que; lo hemos modelado como una matriz, en la cual las coordenadas se dan de la siguiente forma:

$$\begin{bmatrix} 1,1 & 1,2 & \cdots & 1,N \\ 2,1 & 2,2 & \vdots & 2,N \\ \vdots & \vdots & \ddots & \vdots \\ M,1 & M,2 & \cdots & M,N \end{bmatrix} \tag{2}$$

ya que entendemos como se manejan las coordenadas en una matriz, lo que haremos sera modelar dar un paso hacia la dirección a la que mira el ratón, tomando que $N = X$ y $M = Y$, de esta forma, el movimiento queda de la siguiente forma:
suponiendo te encuentras en la posición (X, Y):

- Si te mueves al Norte, terminaras en $(X, Y - 1)$
- Si te mueves al Sur, terminaras en $(X, Y + 1)$
- Si te mueves al este, terminaras en $(X + 1, Y)$
- Si te mueves al oeste, terminaras en $(X - 1, Y)$

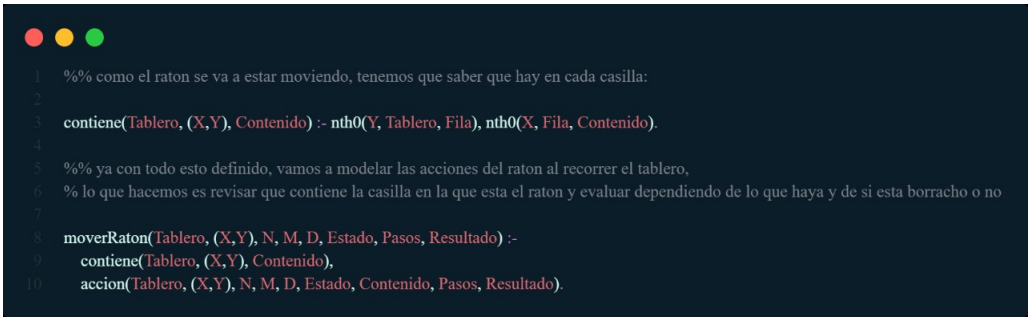
ya que hemos modelado como se movería el ratón en el tablero, lo implementaremos en Prolog de la siguiente manera:

```
%% tenemos que el raton se puede mover en linea recta adentro de la cuadricula, lo que se modela de la siguiente forma:

mover((X,Y), no, (X,Y1)) :- Y1 is Y - 1.
mover((X,Y), su, (X,Y1)) :- Y1 is Y + 1.
mover((X,Y), es, (X1,Y)) :- X1 is X + 1.
mover((X,Y), oe, (X1,Y)) :- X1 is X - 1.
```

ya que hemos implementado la capacidad de moverse por el tablero, puede pasar que al desplazarnos terminemos fuera de los limites de la cuadricula, para evitar esto, notemos que; dependiendo cual sea la dirección a la que mira el ratón, este se saldrá por esa frontera de la cuadricula, por ejemplo, si esta viendo hacia el norte, este se saldrá por la frontera norte (la frontera de los pares ordenados de la forma (X, Y) con $X = 0$), y así con todas las direcciones, por tanto, lo que haremos sera verificar que nuestro ratón este en la cuadricula y además en caso de que se salga, lo resolveremos regresandolo a los limites de la cuadricula; también implementaremos el girar a la izquierda para el caso cuando el ratón esta sobrio lo que al implementarlo nos queda de la siguiente forma:

```
%% como nos podemos salir de la cuadricula si no tenemos quidado, verificaremos que siempre estemos en la cuadricula:

verificar((X,Y),N,M,Z) :- (N<X -> Z is 2;(X<0 -> Z is 4 ; (M < Y -> Z is 3;(Y < 0 -> Z is 1; Z is 0)))).

%% calculamos el index, el cual nos va a decir si nos hemos salido y en que direccion fue
% 0 - seguimos adentro de la cuadricula
% 1 - nos salimos por el norte
% 2 - nos salimos por el este
% 3 - nos salimos por el sur
% 4 - nos salimos por el oeste

rectificar((X,Y), _, _, D, 0, (C,V), NewD) :- C is X, V is Y, igual(D, NewD).
rectificar((X,_), _, _, D, 1, (C,V), NewD) :- C is X, V is 0, girar(D, NewD).
rectificar((_,Y), N, _, D, 2, (C,V), NewD) :- C is N, V is Y, girar(D, NewD).
rectificar((X,_), _, M, D, 3, (C,V), NewD) :- C is X, V is M, girar(D, NewD).
rectificar((_,Y), _, _, D, 4, (C,V), NewD) :- C is 0, V is Y, girar(D, NewD).

igual(no,no).
igual(es,es).
igual(su,su).
igual(oe,oe).


%% cuando el raton se topa con el limite de la cudaricula, tiene que girar a la izquierda, por lo que lo modelaremos:

girar(no,oe).
girar(oe,su).
girar(su,es).
girar(es,no).
```

Una vez que se ha definido como desplazarse, ya podemos plantearnos como es que el ratón va a interactuar con el tablero, para poder hacerlo, lo que nos falta es determinar como se va a implementar el estado de ratón borracho", como sabemos que este estado dura 7 turnos, lo que haremos es modelizarlo como un contador, si el contador esta en 0, el ratón se encuentra sobrio, si el contador es mayor a 0, el ratón se encuentra ebrio. así pues, ya que hemos modelado eso, lo que vamos a hacer es modelar el movimiento del ratón, para eso, cuando se encuentre en una casilla, se va a leer el contenido de la misma, y dependiendo de el contenido y del estado de la misma se va a determinar una acción:



```
%% como el raton se va a estar moviendo, tenemos que saber que hay en cada casilla:

contiene(Tablero, (X,Y), Contenido) :- nth0(Y, Tablero, Fila), nth0(X, Fila, Contenido).

%% ya con todo esto definido, vamos a modelar las acciones del raton al recorrer el tablero,
% lo que hacemos es revisar que contiene la casilla en la que esta el raton y evaluar dependiendo de lo que haya y de si esta borracho o no

moverRaton(Tablero, (X,Y), N, M, D, Estado, Pasos, Resultado) :-
    contiene(Tablero, (X,Y), Contenido),
    accion(Tablero, (X,Y), N, M, D, Estado, Contenido, Pasos, Resultado).
```

Como queremos saber los pasos que da el ratón y además queremos saber si logro salir del laberinto o murió, lo que hacemos es ir guardando en una lista los pasos que ha dado, incluidos los que solo son un cambio de dirección, pues esos nos ayudan a ver lo que hizo el ratón en todo momento.

Como la acción que realice el ratón depende del contenido que tenga en la casilla donde esta y si esta borracho o no, las acciones posibles son:

- si esta sobrio y no hay nada en la casilla y no esta en el borde, el ratón se moverá en la dirección en la que mira:
- si esta ebrio y no hay nada en la casilla y no esta en el borde, se moverá de forma aleatoria en una de las 4 direcciones
- si encuentra un queso normal avanzara una casilla a la dirección que mira, si esta en el borde, se quedara ahí si esta ebrio, girara a la izquierda si esta sobrio
- si encuentra un queso con ron, se lo comerá y se pondrá borracho, avanzando de forma aleatoria a un lado, si esta en un borde y trata de avanzar para salirse de la cuadricula, se quedara ahí sin moverse hasta que se le acabe el estado de ebriedad
- si encuentra un queso con veneno y esta sobrio, no lo come y avanza en la dirección que esta viendo
- si encuentra un queso con veneno y esta ebrio, el ratón se lo come y muere
- si el ratón llega a una casilla de salida, el ratón termina de moverse y sale del tablero

Como queremos guardar los pasos que de el ratón, plantearemos el movimiento del ratón como una acción recursiva, donde las acciones de salir del laberinto o morir serán los casos base, así pues, como varias acciones solo se diferencian sus condiciones en el estado de ebriedad del ratón, podemos resumir estas acciones a 5, que coinciden con el numero de cosas que puede contener una casilla. Entonces, la estructura de nuestra recursion sera:

- el ratón actúa según el contenido de la casilla
- el ratón se mueve según la acción que realice
- se llama de nuevo a la función que mueve al ratón

con esto en mente, y basándonos en las acciones que pueden pasar si come el queso con veneno, al implementarlo en Prolog quedaría así:

```
accion(Tablero, (X,Y), N, M, D, Estado, ven, Pasos, Resultado):-
    movSec((X,Y), D, N, M, Estado, (NX,NY), ND),
    comerQueso(Tablero, Y, X, vac, NewTablero),
    (Estado < 1 -> moverRaton( Tablero, (NX,NY), N, M, ND, Estado, [(X,Y)|Pasos], Resultado);
       accion(NewTablero,(X,Y),N,M,D,Estado,aux2,[(X,Y)|Pasos],Resultado)
    ).
```

una vez modeladas todas las acciones que tiene que hacer el ratón y como se mueve, lo único que basta es implementar como recibir la entrada, lo único que necesitamos es un tablero, la posición inicial del ratón, las dimensiones y la dirección a la que mira el ratón, un detalle a considerar, es que como las listas empiezan a contar de 0, tendremos que restarle 1 a las dimensiones, de forma que una matriz de 4x4 tendrá dimensiones 3x3 en el programa lo que nos quedara así: c

```
raton(Tablero, (X,Y), N, M, D, Resultado) :- moverRaton(Tablero, (X,Y), N-1, M-1, D, 0, [], Resultado).

```

Ya que hemos visto a grandes rasgos como se fue haciendo la implementación, hablaremos de un hecho curioso, y es el hecho de que se pueden formar tableros en los cuales; dependiendo de la posición inicial del ratón, sea imposible salir de ellos y a su vez sea imposible que el ratón muera, a continuación describiremos un ejemplo.

Sea x un tablero valido es decir un tablero que tiene salida, lo definimos como la siguiente matriz:

$$\begin{bmatrix} nada & queso(normal) & nada \\ queso(normal) & salida & nada \\ queso(normal) & nada & nada \end{bmatrix} \tag{3}$$

si elegimos que el ratón comience en la cuadricula $(2, 1)$ con dirección al sur, el ratón podrá salir de forma normal, sin embargo si lo colocamos en otra posición, por ejemplo en $(1, 1)$ viendo hacia cualquier dirección, no hay forma de que el ratón alcance la salida. Esto se debe a que en las condiciones del problema nunca se especifica algún sitio en especifico para colocar la salida, y además el movimiento del ratón estando sobrio es en linea recta, por tanto, a la hora de computar el recorrido del ratón y su destino final, cualquier programa se quedara en un estado de loop infinito, pues nunca se alcanzan las condiciones para finalizarlo (que en este caso es, salir del tablero o morir).

Habiendo notado esto, vemos que; no podemos generar un algoritmo que nos diga si dado un tablero y un ratón en una posición, este llegara a un estado final o se seguirá moviendo de forma indefinida, es decir esto es una versión en especifico del problema del paro, planteado por Alan Turing, lo único que podemos hacer es plantear ciertas condiciones que nos puedan decir que tableros con una forma muy concreta y que tengan al ratón en una posición de inicio concreta terminan, pero no es posible decirlo de forma general.

4. Conclusiones

La programación logica tiene la gran ventaja de funcionar con una base de conocimentos establecida por el propio usuario, lo que le da al propio usuario definir que es verdad y que no es posible, y así poder inferir a partir de una base de conocimientos establecidas, nuevo conocimiento.

Lo que vuelve a la programación lógica vital para el desarrollo de IA's que hoy en día ha aumentado su popularidad, y por ende, su demanda en grandes empresas que quieren meterse en este mundo en

desarrollo. Ademas de que su uso se extiende a base de datos, pues al tener por lo mismo, una base de conocimientos, facilita mucho el acceso y peticiones a la misma, sabiendo que la base de conocimientos siempre es verdadera desde el punto de vista de la programación lógica.

La versatilidad de este tipo de paradigma ayuda a resolver problemas que justo esten relacionados entre si como nuestro problema planteado.

Analizando nuestra solución, al ser de carácter recursivo, esta tiende a tener un uso mas exhaustivo de la memoria, puesto que cada llamada recursiva se guarda en la pila, ya que no podemos hacer que sea una recursion de cola, pues por como funciona Prolog, para encontrar el valor del resultado esperado, las llamadas se van acumulando. Sin embargo, si contar eso, las peticiones mas complejas y lentas (es comerQueso()) solo se usan en los momentos necesarios, además por como funciona el algoritmo, su tiempo de ejecución solo depende que cuantos pasos da el ratón, si por ejemplo esta al lado de la salida, el programa solo tardara un ciclo para terminar la ejecución, en cambio, si esta del otro lado del tablero, tampoco le tomara muchos pasos. por tanto, vemos que el programa a nivel de tiempo de ejecución, es bastante eficiente, pues solo hace las consultas necesarias.

5. Referencias

Referencias

Angel MEa, J. M. D. S. M. A. M. A. E. G., Violeta Leon. (2016, julio). *Prolog- bases de datos deductivas-lógica modal y temporal.* `https://www.slideshare.net/slideshow/prolog-64494802/64494802`.

ONWORKS. (s.f.). *Analizador javalog.* `https://www.onworks.net/es/software/app-javalog-analyser`.

Pascual Julián Iranzo, M. A. F. (2007). *Programación lógica: Teoría y práctica.* Pearson Educación S.A. `https://gc.scalahed.com/recursos/files/r161r/w24546w/programacion-logica.pdf`.