Temario

- Programación Lógica . Agentes inteligentes basados en conocimiento. Programación lógica. Prolog

Bibliografıa

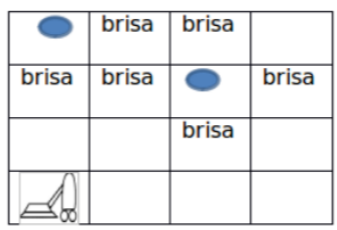
- Inteligencia artificial. Un enfoque moderno. Russell y Norvig

Ejercicios

De fran : <https://docs.google.com/document/d/1XXQhbDEQGRhEjoc0lHvbOohaNmLaBTUaHMw-9PihVDo/edit>

1. Diseñe un agente reactivo con estado, que sea capaz de limpiar de manera segura (sin caer en un pozo) un edificio formado por 16 habitaciones. En algunas habitaciones hay pozos y deben ser evitadas por el agente. En las habitaciones contiguas (arriba, abajo, a la derecha y a la izquierda, no en diagonal) a la que tiene un pozo se percibe una brisa.

Por ejemplo:



Defina el Rendimiento, Entorno, Actuadores y Sensores del agente. El conocimiento inicial del agente contiene las reglas del entorno, sabe que está en la casilla (1,1) y que allí nunca hay un pozo. Su conocimiento debe evolucionar a medida que recibe nuevas percepciones y va ejecutando acciones.

a- Considere que el agente sólo puede percibir el nivel de suciedad de la habitación donde se encuentra.

b- Considere que el agente puede percibir el nivel de suciedad de todas las habitaciones del entorno.

Teniendo en cuenta las indicaciones del enunciado y los vídeos de teoría que nos dieron voy a intentar armar el REAS del agente:

R: Limpiar habitación +1000,

Caer en pozo -1000,

Moverse -1

E: Matriz 4x4 donde puede haber hoyos.

A: girarse 90º a la derecha, girarse 90º a la izquierda, moverse a la casilla de adelante.

S: brisa(b), suciedad(s), muro(m)

Conocimiento inicial:

* **R1:** ¬H(1,1)
* Para x∈{1,2,3,4} e y∈{1,2,3,4}:

**R2:** B(x,y) ↔ H(x-1,y) ∨ H(x+1,y) ∨ H(x,y-1) ∨ H(x,y+1)

Esta regla viene a representar que si en una casilla se siente una brisa es porque hay un hoyo en una casilla adyacente, si estamos en un borde descartamos los lados que no existan de la regla.

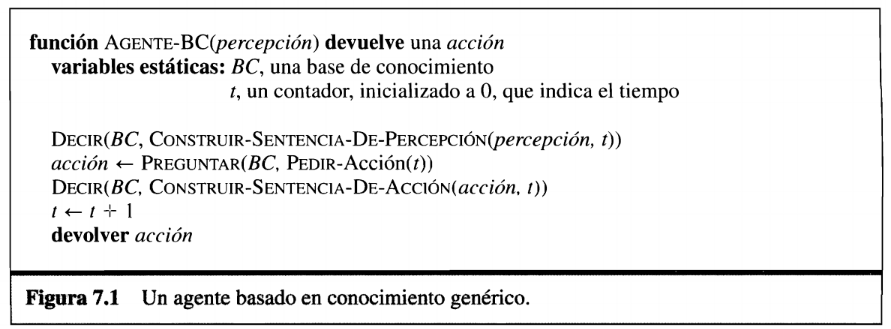
* Para x∈{1,2,3,4} e y∈{1,2,3,4}:

**R3:** S(x,y)

Esta regla viene a representar que se sabe desde el inicio donde está la suciedad en el mapa, me pareció la mejor forma de representarlo.

Con esto tendríamos el conocimiento inicial del agente. Me gustaría destacar que es perfectamente posible que la configuración inicial del mundo provoque que sea imposible para el agente cumplir su misión, por ejemplo un mundo con suciedad en [2,2] y hoyos en [2,1] y en [1,2], el agente nunca podrá llegar a la suciedad.

A continuación muestro el codigo de un agente basado en conocimiento genérico, partiremos de este para diseñar al nuestro:



Los métodos Decir son necesarios, ya que el primero actualiza la BC con las últimas percepciones y el último actualiza la BC con la acción que planea realizar, la variable t representa el tiempo, no es de importancia por ahora, y el método Preguntar retorna una acción a tomar en base a la información de la BC.

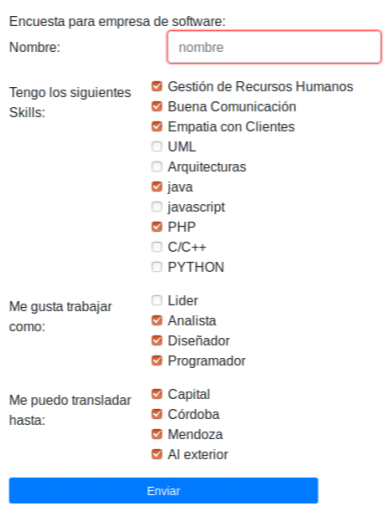
El método Decir permite que el conocimiento del agente evolucione a medida que recibe nuevas percepciones y va ejecutando acciones como pide el enunciado.

El método Preguntar permite que nos abstraigamos de un mecanismo concreto de inferencia ocultando los detalles, en los que asumimos que se utiliza un mecanismo adecuado para elegir acciones acorde al propósito y el conocimiento del agente.

Los detalles finos de los métodos lo dejaremos para el nivel de implementación, parafraseando el libro de “RUSSELL Y NORVIG” en la página 220 “Pero gracias a las definiciones de Decir y Preguntar, el agente basado en conocimiento no obtiene acciones mediante un proceso arbitrario. Es compatible con una descripción al nivel de conocimiento, en el que solo necesitamos especificar lo que el agente sabe y los objetivos que tiene para establecer su comportamiento”.

Con esto tenemos un primer diseño de nuestro agente, como solo se pidió diseño y no implementación voy a dejar el ejercicio en este punto.

2. Diseñar e implementar en Prolog un agente que asista en las búsquedas laborales. Especifique el REAS del agente. Se ha realizado la siguiente encuesta para determinar las habilidades y preferencias de los candidatos:



En base a estos datos se desea consultar:

* Las personas disponibles para ocupar un cierto rol en una determinada localidad. Considerando que una persona podría ocupar un rol si lo prefiere y si posee alguna habilidad relacionada con dicho rol.
* Las personas disponibles para un cierto rol.
* Listar los roles que puede ocupar una persona dada.
* Si una persona en particular es apta para un cierto rol.

Primero el REAS:

* R: asistir en consulta +100.
* E: entorno puramente digital, va a ser un agente con el que nos comunicamos via consola de Prolog.
* A: resolver solicitudes con hechos y reglas de la BC.
* S: algo que le permita saber que ingresa el usuario por consola.

No me convence pero tampoco sé qué mas poner como REAS.

En la clase 10b se usa de ejemplo un programa similar, lamentablemente no se muestra el código de Prolog correspondiente ni cómo crearlo, así que miraré algunos tutoriales e intentaré enfrentar este punto.

Luego de ver algunos tutoriales que muestran cómo programar en Prolog llegue a la conclusión que lo que necesito definir es una base de conocimiento, una vez con esta a través de “swi\_prolog” el cual acabo de instalar se podrá consultar la información que requerimos.

Primero debemos construir la base de conocimiento, lo haré creando un archivo “BC.pl” el cual es un block de notas que contiene lo siguiente:

Voy a agregar los datos de una empleada imaginaria llamada Gabriela y de otro empleado llamado Tomas para construir los ejemplos.

| sirve\_para(gestion\_recursos\_humanos,lider).  sirve\_para(buena\_comunicacion,lider).  sirve\_para(empatia\_con\_clientes,lider).  sirve\_para(arquiteccturas,analista).  sirve\_para(uml,disenador).  sirve\_para(java,programador).  sirve\_para(javascript,programador).  sirve\_para(php,programador).  sirve\_para(c,programador).  sirve\_para(python,programador).  posee\_skill(gabriela,php).  posee\_skill(gabriela,buena\_comunicacion).  posee\_skill(tomas,gestion\_recursos\_humanos).  posee\_skill(tomas,buena\_comunicacion).  posee\_skill(tomas,empatia\_con\_clientes).  posee\_skill(tomas,java).  posee\_skill(tomas,php).  prefiere\_rol(gabriela,programador).  prefiere\_rol(gabriela,analista).  prefiere\_rol(gabriela,lider).  prefiere\_rol(tomas,analista).  prefiere\_rol(tomas,disenador).  prefiere\_rol(tomas,programador).  se\_puede\_transladar(gabriela,cordoba).  se\_puede\_transladar(gabriela,al\_exterior).  se\_puede\_transladar(tomas,capital).  se\_puede\_transladar(tomas,cordoba).  se\_puede\_transladar(tomas,mendoza).  se\_puede\_transladar(tomas,al\_exterior).  es\_apto(Persona,Rol) :- prefiere\_rol(Persona,Rol),posee\_skill(Persona,Skill),sirve\_para(Skill,Rol).  disponible\_para\_rol\_en(Persona,Rol,Localidad) :- se\_puede\_transladar(Persona,Localidad),es\_apto(Persona,Rol). |
| --- |

La BC consiste de Hechos y Reglas, una vez con esto armado y cargado, solo tendremos que realizar consultas a través de la cónsula, de la siguiente manera:

Mostraré una captura de cómo consulto cada cosa con la BC que armer:

* Las personas disponibles para ocupar un cierto rol en una determinada localidad:

| Para consultar esto se hace con la siguiente sintaxis  disponible\_para\_rol\_en(Persona,”rol\_concreto”,”localidad\_concreta”).  Esto nos listará todas las personas que pueden ocupar el rol indicado en la localidad indicada, por ejemplo si con la BC quisiera preguntar por programadores en Córdoba ocurriría lo siguiente:    En un inicio solo veremos la primera opción, pero tecleando “;” se nos mostrarán los demás empleados que cumplen con los requisitos.  También podemos preguntar por líderes en el exterior, esto sería así:    Esta vez nos dio una sola alternativa, ya que Gabriela es la única que cumple los requisitos. |
| --- |

* Las personas disponibles para un cierto rol:

| Esto se consulta con la siguiente sintaxis:  es\_apto(Persona,”rol\_concreto”).  Por ejemplo si queremos saber quienes pueden actuar como programadores:    En un inicio solo veremos la primera opción, pero tecleando “;” se nos mostrarán los demás empleados que cumplen con los requisitos.  Si quisiéramos saber quienes pueden actuar como líderes:    De esta forma podemos obtener lo que requiere el enunciado. |
| --- |

* Listar los roles que puede ocupar una persona dada:

| Esto se consulta con la siguiente sintaxis:  es\_apto(“persona\_concreta”,Rol).  Por ejemplo si queremos saber qué roles puede ocupar Gabriela:    En un inicio solo veremos la primera opción, pero tecleando “;” se nos mostrarán los demás empleados que cumplen con los requisitos.  Si queremos saber qué roles puede ocupar Tomas:    De esta forma podemos obtener lo que requiere el enunciado. |
| --- |

* Si una persona en particular es apta para un cierto rol:

| Esto se consulta con la siguiente sintaxis:  es\_apto(“persona\_concreta”,”rol\_concreto”).  Por ejemplo si queremos saber si Gabriela puede ser programadora:    El agente nos responde true porque efectivamente Gabriela es apta como programadora.  Si queremos saber si Gabriela puede ser analista:    El agente nos responde true porque efectivamente Gabriela no es apta como analista. |
| --- |

3. Diseñar e implementar en Prolog un agente que dé información sobre el árbol genealógico de las personas.

* Definir en Prolog los predicados que definen por extensión todas las relaciones familiares directas, padre(Padre, Hijo) y madre(Madre, Hijo).
* Definir la relación progenitor, utilizando las relaciones de padre y madre.
* Definir recursivamente la relación antepasado.
* Definir nuevas relaciones (como hermano, hermana, abuelo, abuela) añadiendo los predicados (por ejemplo mujer, hombre) y reglas necesarias.

Todo lo que piden se define a través de la BC (base de conocimiento), de la siguiente forma:

* **Definir en Prolog los predicados que definen por extensión todas las relaciones familiares directas, padre(Padre, Hijo) y madre(Madre, Hijo).**

Definire por extensión las relaciones familiares de personajes ficticios, ya que piden que defina por extensión y no dieron ningún ejemplo con el que trabajar, para esto debemos agregar las siguientes líneas en la BC:

| padre(jose,pablo).  padre(pablo,timmy).  padre(pablo,timon).  padre(pablo,melisa).  madre(telma,pablo).  madre(marta,timmy).  madre(marta,timon).  madre(marta,melisa). |
| --- |

Así tenemos definido en la BC las relaciones familiares directas.

* **Definir la relación progenitor, utilizando las relaciones de padre y madre.**

Para esto debemos agregar la siguiente línea en la BC:

| progenitor(P1,P2) :- padre(P1,P2);madre(P1,P2). |
| --- |

* **Definir recursivamente la relación antepasado.**

Para esto debemos agregar la siguiente línea en la BC:

| abuelo(P3,P1) :- progenitor(P3,P2),progenitor(P2,P1). |
| --- |

* **Definir nuevas relaciones (como hermano, hermana, abuelo, abuela) añadiendo los predicados (por ejemplo mujer, hombre) y reglas necesarias.**

Para esto debemos agregar las siguientes línea en la BC:

| mujer(marta).  mujer(melisa).  mujer(telma).  hombre(jose).  hombre(pablo).  hombre(timmy).  hombre(timon).  hermano(P1,P2) :- progenitor(P3,P1),progenitor(P3,P2),hombre(P1).  hermana(P1,P2) :- progenitor(P3,P1),progenitor(P3,P2),mujer(P1).  abuela(P3,P1) :- progenitor(P3,P2),progenitor(P2,P1),mujer(P3). |
| --- |

Y modificar la que creamos en el inciso anterior:

| abuelo(P3,P1) :- progenitor(P3,P2),progenitor(P2,P1),hombre(P3). |
| --- |

Para finalmente quedar la BC así:

| padre(jose,pablo).  padre(pablo,timmy).  padre(pablo,timon).  padre(pablo,melisa).  madre(telma,pablo).  madre(marta,timmy).  madre(marta,timon).  madre(marta,melisa).  mujer(marta).  mujer(melisa).  mujer(telma).  hombre(jose).  hombre(pablo).  hombre(timmy).  hombre(timon).  progenitor(P1,P2) :- padre(P1,P2);madre(P1,P2).  hermano(P1,P2) :- progenitor(P3,P1),progenitor(P3,P2),hombre(P1).  hermana(P1,P2) :- progenitor(P3,P1),progenitor(P3,P2),mujer(P1).  abuela(P3,P1) :- progenitor(P3,P2),progenitor(P2,P1),mujer(P3).  abuelo(P3,P1) :- progenitor(P3,P2),progenitor(P2,P1),hombre(P3). |
| --- |

Con esto ya tenemos armado lo que necesitamos para poder usar el agente solicitado en prolog.

Adjunto a la entrega enviamos un archivo llamado “BC.pl” que puede ser usado como base de conocimiento en swi\_prolog para poder utilizar el agente.

4. Implementar en Prolog el ejercicio de la práctica 6. Si alguien hace algo bueno, ese alguien es bueno. Del mismo modo, si alguien hace algo malo, es malo. Sebastián ayuda a su madre y también miente algunas veces. Mentir es malo y ayudar es bueno. Determinar si con el conocimiento disponible es posible deducir que Sebastian es bueno.¿Y es posible deducir que es malo?

Lo implementare con las herramientas que nos dieron:

Para implementar esto en prolog necesitamos definir hechos y reglas a través de un archivo de texto que llamaremos BC (base de conocimiento) para posteriormente cargarlo en prolog y utilizarlo como un agente, la BC quedo asi:

| %Hechos  hizo(sebastian,ayudar\_mama).  hizo(sebastián,mentir).  es\_un\_acto\_bueno(ayudar\_mama).  es\_un\_acto\_malo(mentir).  %Reglas  es\_alguien\_bueno(P1) :- hizo(P1,A1),es\_un\_acto\_bueno(A1).  es\_alguien\_malo() :- hizo(P1,A1),es\_un\_acto\_malo(A1). |
| --- |

Con respecto a lo que pregunta el enunciado es posible deducir ambas cosas, que sebastián es bueno y que es malo, por lo menos en la forma que resolví el ejercicio.