Trabajo Práctico N°7

Inteligencia Artificial

Integrantes

Bicocchi, Damián — Legajo 21114/8

Ciamparella, Valentín — Legajo 21116/0

Hernández, Sebastián — Legajo 20996/9

Martínez Osti, María Josefina — Legajo 21583/5

Año 2025

1) Para el mundo del Wumpus descrito en Russell y Norvig, vimos algunas reglas que lo caracterizan, por ejemplo “En una casilla se siente una brisa si y sólo si hay un pozo en una casilla vecina”. Escriba en lenguaje natural y en lenguaje simbólico al menos dos reglas adicionales para caracterizar el mundo del Wumpus.

El mundo del Wumpus se compone de una grilla de casillas donde el agente debe encontrar el oro evitando caer en pozos o encontrarse con el Wumpus. El agente no percibe directamente la posición de los objetos, sino que recibe pistas del entorno (brisas, hedores, resplandores, choques y gritos), a partir de las cuales debe razonar utilizando reglas lógicas.

Ya conocemos la regla:
“En una casilla se siente una brisa si y sólo si hay un pozo en una casilla vecina.”

A continuación, se presentan dos reglas adicionales que caracterizan este mundo:

Regla 1 – El hedor del Wumpus

Lenguaje natural:

“En una casilla se percibe un hedor si y sólo si hay un Wumpus en alguna de las casillas vecinas.”

Lenguaje simbólico:

$$H_{(x,y)} \leftrightarrow (W_{(x-1,y)} \vee W_{(x+1,y)} \vee W_{(x,y-1)} \vee W_{(x,y+1)})$$

Donde:

$H_{(x,y)}$: hay hedor en la casilla (x,y)

$W_{(x,y)}$: hay un Wumpus en la casilla (x,y)

Regla 2 – El resplandor del oro

Lenguaje natural:
En una casilla se percibe un resplandor si y sólo si el oro se encuentra en esa misma casilla.

Lenguaje simbólico:

$$R_{(x,y)} \leftrightarrow G_{(x,y)}$$

Donde:

$R_{(x,y)}$: hay resplandor en la casilla (x,y)

$G_{(x,y)}$: hay oro en la casilla (x,y)

Regla 3 – El grito del Wumpus

Lenguaje natural:

Si se oye un grito, significa que el Wumpus ha sido eliminado del mundo.

Lenguaje simbólico:

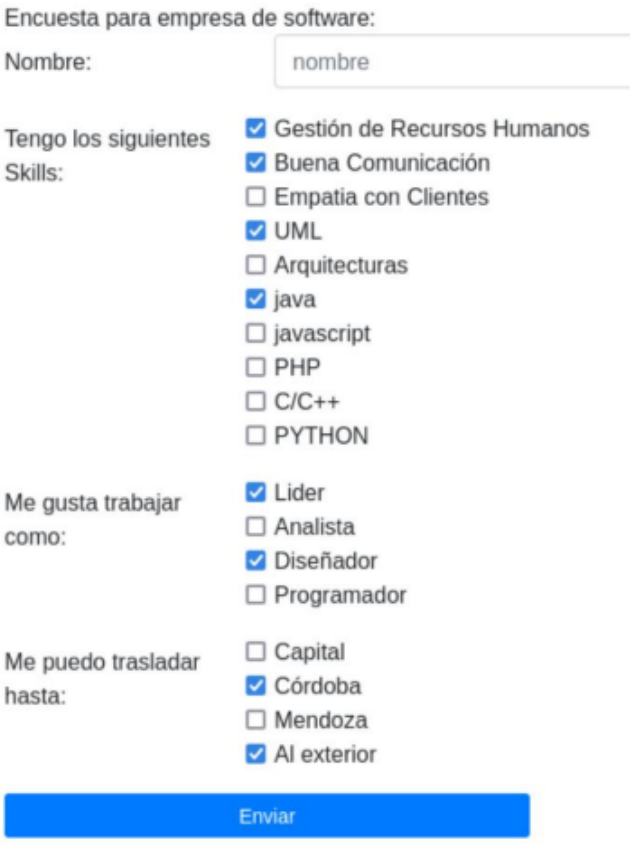
$S \rightarrow (\forall \mathbf{x})(\forall \mathbf{y})\ (\neg W_{(x,y)})$

Donde:

S: se oye un grito

$W_{(x,y)}$: Hay un Wumpus en la casilla (x,y)

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 postulantes:



En base a estos datos y considerando que un postulante podrá ocupar un rol si lo ha expresado como preferencia y si posee alguna habilidad relacionada con dicho rol, se desea consultar:%

- Las personas disponibles para ocupar un cierto rol en una determinada localidad.
- 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

Definición del REAS del agente para busqueda laboral:

- Rendimiento:
    - Encuentra postulantes aptos para los roles solicitados.
    - Devuelve resultados correctos, completos y coherentes con las habilidades y preferencias declaradas.
    - Permite realizar consultas de forma rápida y sin ambigüedades.
    - Maximiza la adecuación entre personas y roles.
    - Facilita la toma de decisiones para selección de personal.
- Entorno:
    - La base de datos de postulantes.
    - Las localidades disponibles.
    - Los roles posibles.
    - Las habilidades que posee cada persona.
    - Las preferencias laborales expresadas por cada postulante.
- Actuadores:
    - Producir listas de personas disponibles para un rol.
    - Filtrar por localidad.
    - Determinar si una persona es apta para un rol.
    - Listar todos los roles posibles para una persona.
    - Permitir realizar consultas desde el intérprete de Prolog.
- Sensores:
    - Lectura de la base de hechos declarada en Prolog.
    - Consultas realizadas por el usuario.
    - Entrada de nuevos datos.

Por otro lado, el programa realizado en Prolog, busca capturar la esencia de la captura adjunta, donde cada hecho representa la relación entre los objetos del dominio. En este caso decidimos relacionar a un sujeto llamado “Juan” con habilidades para poder ver el comportamiento del programa ante las reglas empleadas. Entre dichos hechos relacionados al sujeto tenemos los siguientes:

```
skill(juan, java).
prefiere(juan, programador).
puede_ir(juan, cordoba).
```

Estos hechos, reflejan el mundo real donde un sujeto posee ciertas habilidades para tener un hueco en el rubro laboral.

Por otro lado, para poder tener relaciones entre roles y habilidades se establecen hechos por cada habilidad y rol, se hace la asignación enfocándose en ejemplos del mundo real, por ejemplo:

```
requiere(programador, java).
requiere(programador, javascript).
requiere(analista, uml).
```

Luego por último, definimos las reglas, que representan el conocimiento derivado del sistema. Estas van a permitir al agente poder responder consultas, dándole el cómo razonar y qué acciones tomar. Por ejemplo, para saber si una persona está disponible para un rol en cierta localidad, definimos hacer la siguiente regla:

```
personas_para_rol_en_localidad(Persona, Rol, Localidad) :-
    prefiere(Persona, Rol),
    requiere(Rol, SkillReq),
    skill(Persona, SkillReq),
    puede_ir(Persona, Localidad).
```

Esto indica que, la persona debe preferir el rol y debe tener alguna habilidad necesaria para dicho rol y a su vez poseer esa habilidad dentro de sus posibilidades, y por último que se pueda trasladar hacia esa localidad.

Una consulta que se le puede hacer al agente sería, pedir las personas que puedan trabajar como “programador” en “La Plata”. El agente tiene esta query:

```
(personas_para_rol_en_localidad(P, programador, la_plata),
    writeln(P),
    fail ; true),
```

Esto imprime el nombre de la persona que pueda ocupar el puesto de programador y que esté disponible para trabajar en la localidad de La Plata.

En nuestra solución hay varias reglas que resultan iguales entre sí, mas con distinto nombre. Decidimos hacerlo así por dos razones

1. Nos parece mejor que el agente trabaje con una API más completa
2. Permite al agente poder evolucionar. Dese el caso de que en el futuro una persona es apta para un rol solamente si tiene TODAS las habilidades para ese rol en particular, solo debemos cambiar una regla en vez de buscar entre todos

Luego, para hacer las consultas casi todos los ítems requeridos en el ejercicio tienen una consulta específica para ellas.

Para ejecutar el script es necesario posicionarse en "../ejercicio2" (sin las comillas) y ejecutar el comando

```
swipl -q -f automatizacion_consulta.pl
```

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

- Escribir en Prolog los hechos que definen algunas relaciones familiares directas (ej, juan es padre de pedro, ana es madre de juana, etc).
- 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.

Definición del REAS del agente para el árbol genealógico:

- Rendimiento:
    - Devuelve correctamente las relaciones familiares que se le soliciten.
    - Es capaz de inferir parentescos indirectos y determinar antepasados de cualquier nivel
    - Permite realizar consultas de forma rápida y sin ambigüedades.
- Entorno:
    - La base de los hechos familiares, que incluye género, relaciones de paternidad y maternidad.
    - Las relaciones derivadas que pueden obtenerse mediante las reglas lógicas
    - La estructura lógica del árbol genealógico representado en Prolog
- Actuadores:
    - Responder consultas sobre relaciones familiares directas e indirectas.

    - Determinar si se cumple una relación dada.
    - Listar personas que cumplen una relación determinada.
    - Permitir realizar consultas desde el intérprete de Prolog.
- Sensores:
    - Lectura de la base de hechos declarada en Prolog.
    - La interpretación de las reglas definidas
    - Entrada de nuevos datos.
    - Las consultas del usuario introducidas en el intérprete.

La resolución de este ejercicio se encuentra en el archivo “ejercicio3/ejercicio_3.pl”.
En primer lugar, armamos la base del conocimiento, que en este caso va a describir:

- El género de cada persona, mediante los predicados hombre y mujer
- Las relaciones directas de parentesco, mediante padre y madre.

Por ejemplo:

```
hombre(pedro).
hombre(juan).
mujer(ana).

padre(juan, pedro).
madre(ana, pedro).
```

Con estas líneas definimos tres personas: Ana, Pedro y Juan. Luego, decimos que Juan y Ana son los padres de Pedro.
Luego, para evitar tener que repetir la lógica con padre y madre, definimos la relación progenitor de la siguiente forma:

```
progenitor(X, Y) :- padre(X, Y).
progenitor(X, Y) :- madre(X, Y).
```

Con esta relación, tenemos una forma de saber quién es padre de quién sin distinguir el género.
Luego, para definir la relación antepasado, tenemos que tener en cuenta que un antepasado puede referirse a:

- Un progenitor directo
- Un progenitor de un antepasado.

Para representar estos casos, usamos:

```
antepasado(X, Y) :- progenitor(X, Y).
```

```
antepasado(X, Y) :- progenitor(X, Z), antepasado(Z, Y).
```

Con esto permitimos deducir cualquier nivel de ascendencia (abuelos, bisabuelos, tatarabuelos, etc.) sin definirlos uno por uno explícitamente.

Luego, para representar otro tipo de relaciones podemos ir utilizando la información que ya definimos. Por ejemplo, para definir la relación de hermanos usamos:

```
hermano(X, Y) :-
    hombre(X),
    progenitor(Z, X),
    progenitor(Z, Y),
    X \= Y.

hermana(X, Y) :-
    mujer(X),
    progenitor(Z, X),
    progenitor(Z, Y),
    X \= Y.
```

En este caso, la primera condición sirve para distinguir si se trata de un hermano o de una hermana(género). Las siguientes condiciones se utilizan para indicar que existe al menos una persona Z que sea progenitora tanto de X como de Y. La última condición la utilizamos para que no se pueda dar que X e Y sean la misma persona, ya que con las reglas anteriores no nos aseguramos de que no sea así. Cabe aclarar que en este caso estamos considerando que dos personas son hermanas si comparten al menos un progenitor. Si quisiéramos considerar la relación de hermanos como que comparten ambos progenitores, habría que cambiar

```
  progenitor(Z, X),
  progenitor(Z, Y),
```

por

```
  padre(P, X), padre(P, Y),
  madre(M, X), madre(M, Y),
```

Las otras relaciones que definimos, abuelo-abuela y tío-tía, siguen esta misma lógica descrita para especificar las relaciones.

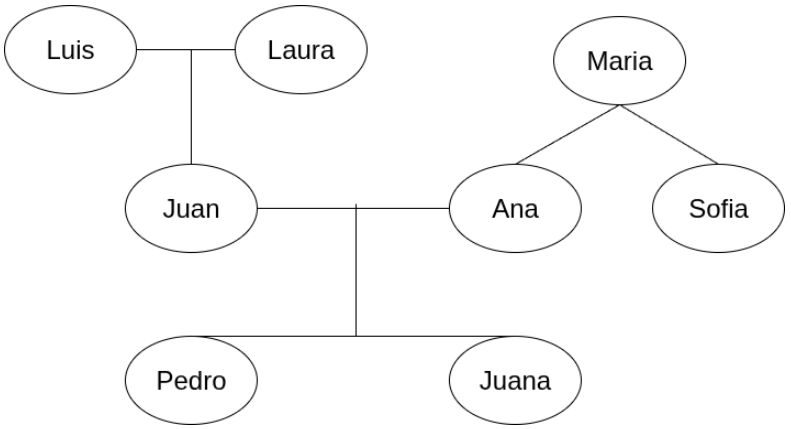
En particular en el caso de los abuelos tenemos que pensar la relación como ser “progenitor de un progenitor” y en el caso de los tíos tenemos que pensarla como “el hermano/a del progenitor”

Con esto, podemos usar el agente para que determine parentescos usando la inferencia lógica a partir de hechos básicos. Por ejemplo, podríamos hacer la consulta:

```
(tia(maria, pedro) -> writeln(true) ; writeln(false))
```

Para conocer si Maria es tía de Pedro.

Sabiendo que en la base de conocimiento se define este árbol genealógico



Para ejecutar el script es necesario posicionarse en “../ejercicio3” (sin las comillas) y ejecutar el comando

```
swipl -q -f automatizacion_consulta.pl
```

4) Implementar en Prolog el ejercicio de la práctica 4:

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 Sebastián es bueno.¿Y es posible deducir que es malo?

La base de conocimiento está implementada en el archivo “ejercicio4/ejercicio_4.pl”.

Tenemos por enunciado las siguientes reglas en lenguaje natural

- Una persona es buena si hace algo bueno
- Una persona es mala si hace algo malo

También, deducimos por el enunciado las siguientes declaraciones

- Mentir es malo
- Ayudar es bueno
- Sebastian ayuda (para este caso, es irrelevante a quién)

- Sebastian miente (para este caso, es irrelevante si lo hace siempre o no)

Planteamos esta base de conocimiento en el archivo previamente mencionado
Definimos dos reglas:
La primera define si una persona es buena a través de dos condiciones:

- Que esa persona haga algo
- Que ese algo que hace sea bueno

La segunda regla es muy similar a la anterior, donde definimos si una persona es mala a través de

- Que esa persona haga algo
- Que ese “algo” sea malo.

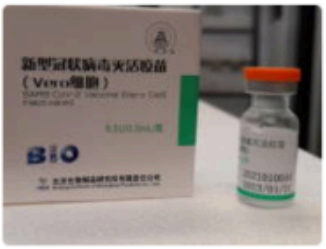
El conocimiento que queremos tener proviene de hacer dos consultas a esta base del conocimiento. Si “Sebastian” es bueno y si “Sebastian” es malo. Automatizamos el proceso con un script de prolog que ejecuta ambas queries y nos retorna el valor de verdad de las mismas.
Lo que se concluye a partir de todo lo dicho es que Sebastian es bueno y malo al mismo tiempo.

Para ejecutar el script es necesario posicionarse en “../ejercicio4” (sin las comillas) y ejecutar el comando

```
swipl -q -f automatizacion_consulta.pl
```

5) Implementar en Prolog un agente que asista en la distribución de la vacuna contra el COVID-19. Su base de conocimiento debe ser construida a partir de la especificación de cada una de las vacunas que se administran, disponible en ¿Cuáles vacunas aplicamos en el país? | Argentina.gob.ar Por ejemplo, en el caso de la siguiente vacuna:

Sinopharm



Nombre: SARS COV-2 (células vero) inactivada
Desarrollador: Beijing Institute of Biological Products - República Popular China
Autorizado edad (edad): ≥ 3 años
Plataforma: virus inactivados
Contraindicaciones: hipersensibilidad a cualquier componente; antecedente de reacciones alérgicas graves (con compromiso respiratorio que haya requerido asistencia médica); exacerbación de enfermedades crónicas, que impliquen compromiso del estado general.
Contraindicación para la segunda dosis: reacción anafiláctica con la primera dosis.

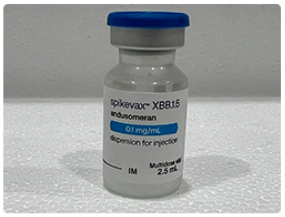
Se podrían definir los siguientes hechos:

```
edad_minima(sinopharm, 3).
contraindicacion(sinopharm, hipersensibilidad).
contraindicacion(sinopharm, enfermedades_cronicas).
contraindicacion(sinopharm, trombocitopenia).
contraindicacion(sinopharm, trastornos_coagulacion).
contraindicacion(sinopharm, epilepsia).
```

Crear una base de hechos con las edades y comorbilidades de un grupo de personas.
Escribir las reglas necesarias para que el agente pueda determinar cuál vacuna puede aplicarse a cada persona
Vacunas presentes en el momento de la resolución

Moderna

Nombre: Spikevax bivariante (Original/Ómicron BA.4/BA.5) (formulación 0.10 mg/ml)
Desarrollador: Moderna Switzerland GmbH
Autorizado edad: ≥ 6 meses
Plataforma: ARN mensajero
Contraindicaciones: anafilaxia o reacción alérgica grave inmediata a la administración de la primera dosis.



Nombre: Spikevax monovariante XBB.1.5 (formulación 0.10 mg/ml)
Desarrollador: Moderna Switzerland GmbH
Autorizado edad: ≥ 6 meses
Plataforma: ARN mensajero
Contraindicaciones: anafilaxia o reacción alérgica grave inmediata a la administración de la primera dosis.

Comirnaty

Nombre: Comirnaty bivariante (Original/Ómicron BA.4/BA.5)
Desarrollador: Pfizer-BioNTech
Autorizado edad: ≥ 12 años
Plataforma: ARN mensajero
Contraindicaciones: Anafilaxia o reacción alérgica grave inmediata a la administración de la primera dosis.

En primer lugar armamos la base de conocimiento para estas tres vacunas. Definimos dos hechos para las vacunas. Uno es “edad_minima” que nos indica la edad mínima que debe tener una persona para que se le suministre esta vacuna (Para ambas vacunas de Moderna, planteamos a la edad mínima como 0.5). El siguiente hecho es “contraindicacion”, que indica un padecimiento que debería evitar el suministro de una dosis. Con las vacunas

que se dan hoy en el país, las tres indican que ante “Anafilaxia” o “Reacción alérgica grave” se debería evitar

Luego creamos dos hechos (relevantes al ejercicio) a partir de 4 personas. Un hecho es la edad de la persona y el segundo sería si tiene una indicación.

Luego podemos crear las reglas para consultar si a una persona se le puede suministrar cierta vacuna.

Creamos una regla “puede_vacunarse” de aridad 2, cuyo objetivo es indicarnos si a una Persona se le puede asignar una Vacuna.

Esta regla se basa en 2 condiciones

- Que la persona tenga la edad recomendada para la vacuna
- Que la persona NO sea inapto para la vacuna debido a una contraindicación

Esta primera condición se puede definir como una nueva regla, donde solo se es verdadera si la Persona tiene una edad que sea mayor a la edad mínima de la vacuna

La segunda condición es solamente negar el resultado de si existe una contraindicación que la persona tiene.

Por último, para realizar las consultas armamos un script de Prolog que ejecuta de manera automática realiza las queries para ver la posibilidad de una persona de vacunarse con cada una de las vacunas del dominio.

Las personas que definimos son

- Claudia de 6 meses, sin indicación
- Hernan de 9 años, con indicación de anafilaxia
- Clara de 12 años, sin indicación
- Joaquín de 18 años, con indicación de reacción alérgica grave

Para ejecutar el script es necesario posicionarse en “../ejercicio5” (sin las comillas) y ejecutar el comando

```
swipl -q -f automatizacion_consulta_5.pl
```

Con los resultados del script, podemos armar la siguiente tabla

¿Puede esta persona vacunarse con esta vacuna?	Spikevax Bivariante	Spikevax Monovariante	Comirnaty
Claudia	SI	SI	NO
Hernan	NO	NO	NO
Clara	SI	SI	SI
Joaquín	NO	NO	NO

- Claudia está clasificada correctamente ya que no tiene la edad suficiente para la vacuna Comirnaty, pero si para ambas Spikevax, y no sufre de ninguna contraindicación.
- Hernan y Joaquin no pueden suministrarse ninguna ya que ambos presentan una contraindicacion de las que imposibilitan la aplicación
- Clara al no tener contraindicaciones y cumplir con las restricciones de edad puede vacunarse libremente con cualquiera de las tres opciones.