UNIVERSIDAD NACIONAL DE COLOMBIA

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

3010476

Sistemas expertos basados en reglas

PROFESOR: JAIME ALBERTO GUZMÁN LUNA

2025-1

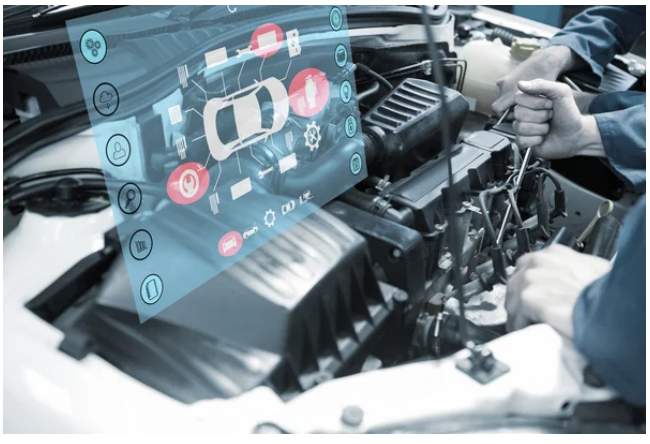
Índice

1. Sistema experto para diagnóstico de fallas en un vehículo **2**

2. Árboles Genealógicos **4**

3. Experto de Diagnóstico Médico **7**
3.0.1. Funcionamiento de las reglas 7
3.0.2. Reglas a completar 7
3.0.3. Consejos para la implementación 7
3.0.4. Implementar las reglas de recomendación médica 7
3.0.5. Funcionamiento de las reglas 8
3.0.6. Reglas a completar 8
3.0.7. Consejos para la implementación 8

1. Sistema experto para diagnóstico de fallas en un vehículo



Un mecánico recibe vehículos con problemas y necesita un asistente experto que sugiera diagnósticos a partir de los síntomas observados y que también pueda registrar el progreso de las reparaciones eliminando síntomas que ya han sido resueltos.

Definición de Hechos

- **Symptom(tipo=str)**: Representa un síntoma en el vehículo.
- **CarState(estado=str)**: Representa el estado general del vehículo.
- **Diagnosis(resultado=str)**: Representa la conclusión del sistema.
- **RepairAction(tipo=str)**: Nuevo hecho que representa una acción de reparación específica.

Reglas de Diagnóstico

Cuando se diagnostica un problema, el sistema genera automáticamente una acción de reparación correspondiente mediante `RepairAction` (mirar sección de reglas de reparación). Para cada tipo de reparación, existe una regla asociada que elimina el síntoma relacionado utilizando `retract`. Este proceso simula la resolución de fallas en el vehículo al eliminar los síntomas detectados.

- Si se cumple la falla grave `Symptom(tipo='humo_blanco')` y `Symptom(tipo='luz_aceite')`, el sistema debe declarar:
  `Diagnosis(resultado='Posible junta de cabeza mala, motor en riesgo')`
- Si se cumple la falla moderada `Symptom(tipo='ruido_metalico')`, el sistema debe declarar:
  `Diagnosis(resultado='Revisar sistema de frenos')`
- Si se cumple la falla moderada `Symptom(tipo='fuga_liquido')` y `CarState(estado='motor_caliente')`, declarar:
  `Diagnosis(resultado='Pérdida de refrigerante, posible sobrecalentamiento')`
- Si no hay síntomas críticos, declarar:
  `Diagnosis(resultado='Revisión general recomendada')`

Reglas de Reparación

- Si se ha declarado `RepairAction(tipo='reparar_frenos')` y existe el síntoma `Symptom(tipo='ruido_metalico')`, se elimina dicho síntoma y se cambia el estado del vehículo a `VehicleStatus(estado='verificar_reparacion')`.
- Si se ha declarado `RepairAction(tipo='reparar_motor')` y existe el síntoma `Symptom(tipo='humo_blanco')`, se elimina el síntoma, se cambia el estado del coche a `CarState(estado='en_reparacion')` y se declara `VehicleStatus(estado='verificar_reparacion')`.
- Si se ha declarado `RepairAction(tipo='reparar_motor')` y existe el síntoma `Symptom(tipo='luz_aceite')`, se elimina dicho síntoma y se declara `VehicleStatus(estado='verificar_reparacion')`.

Uso de Prioridades (salience)

- Reglas de fallas graves: `salience = 100`.
- Reglas de diagnóstico moderado: `salience = 50`.
- Reglas de reparación: `salience = 150`, para que se ejecuten antes que otras acciones menos prioritarias.
- Revisión general: `salience = 10`, para activarse solo si ninguna otra regla se disparó.
- Verificación final del estado del vehículo: `salience = 5`, como paso posterior a las reparaciones.

A continuación pueden encontrar el enlace a la actividad desde google Colab:
Enlace de la actividad

Una vez completado el código, responda las siguientes preguntas:

- ¿Qué resultado se obtiene si se agrega un nuevo síntoma como `ruido_metalico` en la entrada? Explique su respuesta

```
engine.declare(Symptom(tipo='humo_blanco'))
engine.declare(Symptom(tipo='luz_aceite'))
engine.declare(Symptom(tipo='ruido_metalico'))
```

- Utilizando las declaraciones anteriores, ¿Qué sucede si se cambia la saliencia de `motor_grave_dano` de 100 a 40?
- ¿Qué sucede si eliminamos salience de todas las reglas?
- ¿Qué ocurre si se activan múltiples reglas dentro del sistema experto, y cada una declara un hecho del mismo tipo `Diagnosis`, pero con diferentes valores en su atributo `resultado`, como `Diagnosis(resultado=...)`? ¿Se almacenan todos los diagnósticos generados, se sobrescribe alguno, o el motor de inferencia solo considera uno de ellos?
- ¿Cómo cambia el comportamiento del sistema cuando se ejecuta nuevamente después de eliminar algunos síntomas mediante retract?
- ¿Qué ventajas presenta el uso de un hecho intermedio (RepairAction) para gestionar las reparaciones?

2. Árboles Genealógicos

--

El estudio de los lazos familiares es fundamental en diversas áreas como la genealogía, el derecho de herencias, la biología y la sociología. En muchos casos, contar con un modelo estructurado que permita inferir relaciones familiares a partir de una base de datos limitada es un reto, ya que la estructura de parentesco involucra reglas complejas y múltiples niveles de relaciones. Imagina que trabajas en un departamento de genealogía, donde los clientes proporcionan información sobre algunos de sus familiares más cercanos (padres, madres, hermanos). A partir de esta información, un sistema experto debería ser capaz de deducir automáticamente otras relaciones, como abuelos, tíos, primos y hermanos, permitiendo construir un árbol genealógico de manera automatizada. El problema radica en que no siempre se tiene acceso a toda la información desde el inicio. Muchas veces, se conocen solo los padres y los hijos, pero no los abuelos, hermanos o tíos. Es aquí donde un sistema experto basado en reglas puede ayudar a deducir relaciones a partir de los datos existentes.

Se te proporciona:

- ✓ Una base de conocimiento con información sobre relaciones familiares básicas (padres, madres, hombres y mujeres).
- ✓ Consultas manuales para mostrar los resultados inferidos.

Tu tarea es implementar las reglas de inferencia necesarias para deducir automáticamente nuevas relaciones familiares.

Requerimientos de la implementación

Se deben definir los hechos que representarán las relaciones familiares en el sistema experto.
Hechos básicos que serán proporcionados en la base de hechos:

- Hombre(nombre=str): Representa a un hombre en la familia.
- Mujer(nombre=str): Representa a una mujer en la familia.
- Padre(padre=str, hijo=str): Representa la relación de paternidad.
- Madre(madre=str, hijo=str): Representa la relación de maternidad.

Hechos derivados que deben ser inferidos mediante reglas:

- Progenitor(progenitor=str, hijo=str): Se deduce a partir de Padre y Madre.
- Abuelo(abuelo=str, nieto=str): Se deduce si una persona es progenitor del padre o madre de alguien.
- Abuela(abuela=str, nieto=str): Se deduce de manera similar a Abuelo, pero para mujeres.
- Hermano(hermano=str, hermano_de=str): Se deduce si dos personas tienen al menos un progenitor en común.
- Hermana(hermana=str, hermana_de=str): Se deduce de manera similar a Hermano, pero para mujeres.
- Tio(tio=str, sobrino=str): Se deduce si una persona es Hermano de un Progenitor.
- Tia(tia=str, sobrino=str): Se deduce de manera similar a Tio, pero para mujeres.

- Primo(primo=str, primo_de=str): Se deduce si dos personas tienen progenitores que son Hermanos o Hermanas.

Se deben escribir reglas en el sistema experto que permitan inferir las relaciones familiares de forma automática.

Reglas a implementar:

- Regla de progenitores
  - Si Padre(padre=X, hijo=Y), entonces Progenitor(progenitor=X, hijo=Y).
  - Si Madre(madre=X, hijo=Y), entonces Progenitor(progenitor=X, hijo=Y).
- Regla de abuelos y abuelas
  - Si Progenitor(progenitor=X, hijo=Y) y Progenitor(progenitor=Y, hijo=Z), entonces:
    - Abuelo(abuelo=X, nieto=Z), si X es un Hombre.
    - Abuela(abuela=X, nieto=Z), si X es una Mujer.
- Regla de hermanos y hermanas
  - Si X y Y tienen al menos un Progenitor en común, entonces:
    - Hermano(hermano=X, hermano_de=Y), si X es Hombre.
    - Hermana(hermana=X, hermana_de=Y), si X es Mujer.
- Regla de tíos y tías
  - Si X es Hermano de un Progenitor de Y, entonces Tio(tio=X, sobrino=Y).
  - Si X es Hermana de un Progenitor de Y, entonces Tia(tia=X, sobrino=Y).
- Regla de primos
  - Si X y Y son hijos de progenitores que son Hermanos o Hermanas, entonces Primo(primo=X, primo_de=Y).

Control de prioridades

Para garantizar que las reglas se ejecuten en el orden correcto, se deben asignar prioridades (salience) a cada una.

Orden de ejecución:

1. Progenitores: salience=100
2. Abuelos y abuelas: salience=80
3. Hermanos y hermanas: salience=60
4. Tíos y tías: salience=40
5. Primos: salience=20

```
# Base de conocimiento (hechos base)

# Hombres
motor.declare(Hombre(nombre='juan'))
motor.declare(Hombre(nombre='pedro'))
motor.declare(Hombre(nombre='carlos'))
motor.declare(Hombre(nombre='david'))
motor.declare(Hombre(nombre='luis'))

# Mujeres
motor.declare(Mujer(nombre='maria'))
motor.declare(Mujer(nombre='ana'))
motor.declare(Mujer(nombre='sofia'))
motor.declare(Mujer(nombre='laura'))
motor.declare(Mujer(nombre='carmen'))

# Relaciones padre
motor.declare(Padre(padre='juan', hijo='pedro'))
motor.declare(Padre(padre='juan', hijo='ana'))
motor.declare(Padre(padre='pedro', hijo='david'))
motor.declare(Padre(padre='pedro', hijo='sofia'))
motor.declare(Padre(padre='carlos', hijo='luis'))
motor.declare(Padre(padre='carlos', hijo='laura'))

# Relaciones madre
motor.declare(Madre(madre='maria', hijo='pedro'))
motor.declare(Madre(madre='maria', hijo='ana'))
motor.declare(Madre(madre='laura', hijo='david'))
motor.declare(Madre(madre='laura', hijo='sofia'))
motor.declare(Madre(madre='carmen', hijo='luis'))
motor.declare(Madre(madre='carmen', hijo='laura'))
```

Ejecutar el motor de inferencia

Una vez implementadas las reglas, se deben seguir los siguientes pasos:

1. Cargar la base de conocimiento proporcionada.
2. Ejecutar el motor de inferencia mediante la función run().
3. Verificar que las inferencias sean correctas revisando la salida del sistema.

Enlace de la actividad

En función a lo anterior, responda las siguientes preguntas

- ¿Cómo se podría modificar el código para inferir bisabuelos y bisabuelas?
- ¿Por qué la inferencia de Abuelo y Abuela requiere que haya dos niveles de Progenitor?

3. Experto de Diagnóstico Médico

En el ámbito de la salud, la detección temprana y precisa de enfermedades es crucial para garantizar un tratamiento adecuado y evitar complicaciones. Sin embargo, no siempre es posible contar con un médico en todo momento, especialmente en zonas de difícil acceso o en situaciones donde el acceso a un profesional de la salud es limitado.

Los sistemas expertos en medicina han surgido como una herramienta valiosa para asistir en el proceso de diagnóstico. Estos sistemas, basados en inteligencia artificial, utilizan reglas y conocimientos médicos para evaluar síntomas y proporcionar posibles diagnósticos junto con recomendaciones médicas.

Se te proporcionará un **notebook en Google Colab** con parte del código ya implementado. **Tu tarea es completar las secciones faltantes** para que el sistema pueda diagnosticar enfermedades y generar recomendaciones médicas correctamente.

- **Enlace de la actividad**

¿Qué debes hacer?
Implementar las reglas de diagnóstico
Debes completar las reglas que permitirán **diagnosticar enfermedades** en función de los síntomas ingresados por el usuario.

3.0.1. Funcionamiento de las reglas

- Cada enfermedad en la base de conocimiento tiene **una lista de síntomas**.
- Se comparan los **síntomas del paciente** con los síntomas de la enfermedad.
- Si **al menos la mitad** de los síntomas coinciden, el sistema **declara un diagnóstico** con la enfermedad detectada.

3.0.2. Reglas a completar

- `evaluar_resfriado`: Diagnosticar resfriado común.
- `evaluar_gripe`: Diagnosticar gripe.
- `evaluar_covid`: Diagnosticar COVID-19.
- `evaluar_neumonia`: Diagnosticar neumonía.

3.0.3. Consejos para la implementación

Para implementar estas reglas, debes asegurarte de que:

- Se comparen los síntomas correctamente.
- Se utilice una condición de coincidencia de síntomas: $\geq$ len(sintomas)/2.
- Se declare el diagnóstico con la instrucción: `self.declare(Diagnostico(enfermedad='nombre_enfermedad'))`.

3.0.4. Implementar las reglas de recomendación médica

Una vez que el sistema pueda diagnosticar enfermedades, debes completar las reglas que generan **recomendaciones médicas** en función de la **gravedad de la enfermedad diagnosticada**.

3.0.5. Funcionamiento de las reglas

- Cada enfermedad en la base de conocimiento tiene un **nivel de gravedad (1, 2 o 3)**.
- Cuando el sistema diagnostica una enfermedad, se busca su nivel de gravedad.
- Se genera una recomendación médica basada en la severidad.

3.0.6. Reglas a completar

- `recomendar_leve`: Si la enfermedad es de **nivel 1**, se recomienda **descanso en casa**.
- `recomendar_moderada`: Si la enfermedad es de **nivel 2**, se recomienda **consultar al médico**.
- `recomendar_grave`: Si la enfermedad es de **nivel 3**, se recomienda **buscar atención médica urgente**.

3.0.7. Consejos para la implementación

Para implementar estas reglas, debes asegurarte de que:

- Se verifique correctamente el nivel de gravedad.
- Se declare una recomendación con la instrucción: `self.declare(Recomendacion(enfermedad=enf, mensaje="mensaje adecuado"))`.