Pontificia Universidad Javeriana


# Proyecto II - Introducción a la IA

Profesora:

Cecile Eugenie Gloria Gauthier Umaña

Mateo Henao
Gabriel Coronado

Brayan Fajardo

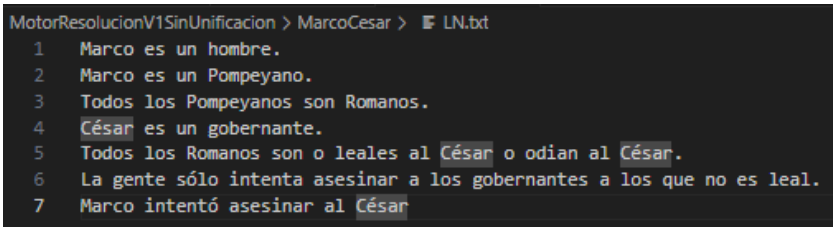
Bogotá D.C., Abril de 2024

**Proceso de Lectura y Procesamiento de Archivos**

El programa comienza leyendo los archivos LN.txt, Logico.txt y LogPrimOrden.txt para obtener la base de conocimientos en lenguaje natural, lógica de primer orden y las reglas lógicas, respectivamente. Luego, analiza cada archivo para extraer las constantes, predicados, la base de conocimientos, la pregunta y la negación.

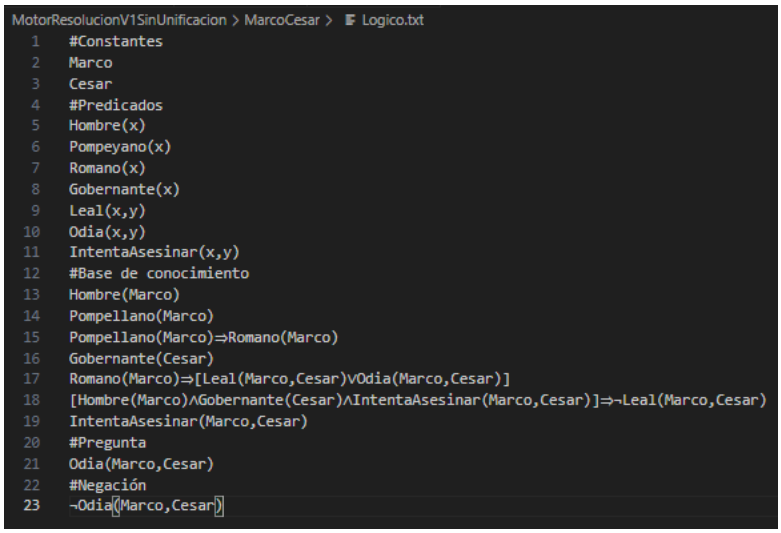
En resumen, el proceso de lectura de archivos comprende la extracción de información crucial del problema, tanto en lenguaje natural como en formato lógico, junto con las reglas y transformaciones necesarias para resolverlo. Esta información es fundamental para llevar a cabo la inferencia lógica y concluir la veracidad de la pregunta inicial.

**Archivo LN.txt (Lenguaje Natural):** Este archivo contiene la base de conocimientos en lenguaje natural. Cada línea presenta una afirmación sobre el problema, como "Marco es un hombre" o "La gente sólo intenta asesinar a los gobernantes a los que no es leal". Esta información proporciona el contexto inicial del problema en un formato comprensible para los humanos.




- Este archivo se utiliza para la representación de la información necesaria para este problema.

**Archivo Logico.txt (Lógica de Primer Orden):** Aquí se encuentra la base de conocimientos expresada en lógica de primer orden. Utiliza constantes y predicados para representar las afirmaciones y relaciones entre los elementos del problema. Por ejemplo, "Hombre(Marco)" indica que Marco es un hombre, mientras que "IntentaAsesinar(Marco, César)" expresa que Marco intentó asesinar a César.




- Cada línea se examina para identificar constantes, predicados y la base de conocimientos en lógica de primer orden.
- Las constantes representan elementos concretos del problema, como nombres de personas o entidades.
- Los predicados describen relaciones o propiedades entre estos elementos, como "IntentaAsesinar(x, y)".
- La base de conocimientos se compone de afirmaciones lógicas basadas en predicados y constantes.

**Archivo LogPrimOrden.txt (Reglas y Transformaciones Lógicas):** Este archivo contiene reglas y transformaciones lógicas que se aplicarán en el proceso de inferencia. Incluye reglas como el Modus Ponens y la eliminación de la implicación, necesarias para la manipulación y simplificación de las afirmaciones lógicas.

MotorResolucionV1SinUnificacion > LogPrimOrden.txt

```
#Modus Ponens
a⇒b,a
b
#Eliminación de Conjunción (NO)
a∧b
a
a∧b
b
#Eliminación de doble negación
¬(¬a)
a
#Eliminación por implicación
a⇒b
¬a∨b
#Eliminación de bicondicional
a⇔b
(a⇒b)∧(b⇒a)
#Leyes de Morgan-Conjunción
¬(a∧b)
¬a∨¬b
#Leyes de Morgan-Disyunción
¬(a∨b)
¬a∧¬b
#Propiedad Distributiva
(a∧(b∨y))
(a∧b)∨(a∧y)
(a∨(b∧y))
(a∨b)∧(a∨y)
#Resolución unitaria
(a∨¬b),b
a
(a∨b),¬b
a
```

- Se lee y analiza este archivo para extraer reglas lógicas y transformaciones que se aplicarán durante la inferencia.
- Las reglas, como el Modus Ponens o la resolución unitaria, se utilizan para manipular y simplificar las afirmaciones lógicas.
- Estas transformaciones son esenciales para convertir la base de conocimientos en una forma adecuada para la inferencia lógica.

**Simplificación de base de conocimiento y aplicación de reglas:**

Una vez leído el archivo, se procede a realizar la simplificación de las frases lógicas ingresadas. Partiendo de las reglas para la conversión a forma normal conjuntiva, recordemos que:

1. Si a => b y a, entonces b.
2. Se pueden separar elementos si estos se encuentran conectados por el conector lógico AND
3. ¬(¬a) es a (Doble negación)
4. a => b implica ¬a v b
5. a <=> b implica (a =>b) ^ (b=>a)
6. Por las leyes de Morgan, ¬(a^b) = ¬a v ¬b y ¬(a v b) = ¬a ^ ¬b
7. Por las leyes distributivas:
    a. (a ^ (b v c) = (a ^ b) v (a ^ c)

b. (a v (b ^ c) = (a v b) ^ (a v c)

A partir de las anteriores reglas, se creó un archivo LogPrimOrden.txt que funciona como una "referencia" para el programa de las normas a seguir. A partir de la lectura de este archivo, el programa genera un diccionario separado por el nombre de las diferentes reglas (Expresados en el archivo como #nombre), donde cada entrada es una dupla que contiene la forma no normalizada (Identificada cuando cont = 1) y su forma simple (Identificada cuando cont = 2).

```
def reglas_simp():
    diccionario = {}
    contador = 0

    with open("LogPrimOrden.txt", "r", encoding="utf-8") as archivo:
        lineas = archivo.readlines()
        for linea in lineas:
            linea = linea.strip()
            if(linea.startswith("#")):
                nombre = ""
                reglas = []
                resultados = []
                partes = linea.split("#")
                nombre = partes[1]
                contador = 1
                continue
            elif(contador == 1):
                reglas.append(linea)
                contador = 2
                continue
            elif(contador == 2):
                resultados.append(linea)
                contador = 1
            diccionario[nombre] = [reglas, resultados]
    archivo.close()
    return diccionario
```

Una vez generado el diccionario, el siguiente paso es eliminar los simbolos ∀ y ∃ de las frases lógicas, para permitir correctamente el proceso de simplificación. Este proceso se hace Asumiendo que estos ya se encuentran exclusivamente al inicio de la frase, y se encuentran escritos de la forma ∀x y ∃x. Con esto en cuenta, se realiza un ciclo en el cual se eliminan los dos primeros caracteres de la frase hasta que ya no se encuentre ninguna de las dos expresiones:

```python
def quitarAyE(base):
    limpio = []
    for i in range(0,len(base)):
        if "∀" in base[i] or "∃" in base[i] :
            nuevoi=base[i][2:]
            if "∀" in nuevoi or "∃" in nuevoi :
                nuevoi=nuevoi[2:]
                base[i]=nuevoi
            else:
                base[i]=nuevoi
    return base
```

Las frases modificadas se agregan a una nueva base de conocimiento, y nuevamente se modifican ahora para ser simplificadas. Aunque la función conv_FNC está inicialmente pensada para aplicar todas las reglas de simplificación, para este caso en específico solo se aplica la regla 4 (Eliminación por implicación). Esto permite dejar la expresión separada por solo conectores lógicos AND y OR para su respectivo análisis:

```python
def conv_FNC(base,reglas):
    for i in range(0,len(base)):
        if "⇒" in base[i]:
            antecedente, consecuente = base[i].split("⇒")
            antecedente = antecedente.strip()
            consecuente = consecuente.strip()
            base[i]=f"¬{antecedente}∨{consecuente}"
    return base
```

Por último, se eliminan paréntesis innecesarios generados durante el proceso de simplificación. Por ejemplo, si un elemento queda de la forma ¬[a(b,c)], se eliminan los paréntesis para que quede de la forma ¬a(b,c). Del mismo modo, si quedan paréntesis innecesarios alrededor de un elemento, estos también se eliminan. De este modo, se puede pasar al proceso de unificación y la posterior comprobación por refutación:

```python
def quitarParentesis(base):

    for i in range(0,len(base)):
        #if i tiene parentesis:
        if "[" in base[i]:
            if "¬[" in base[i]:
                base[i]=base[i].replace('¬[','¬')
                base[i]=base[i].replace('∧','∨¬')
            else:
                base[i]=base[i].replace('[','')

            base[i]=base[i].replace(']','')

    return base
```

**Proceso de unificación (Para la versión 2 de la implementación):**

Una vez se cuenta con la base de conocimientos simplificada. Lo que se tiene que llevar a cabo es el proceso de unificación que consiste en modificar aquellas variables que hacen parte de las declaraciones y reemplazarlas por los valores fijos o constantes que se tienen (en este caso Marco y Cesar).

Para este procedimiento se empezó haciendo una lista de las constantes que se encuentran en la base de conocimiento; es decir, que líneas de la base de conocimiento no cuentan con ningún tipo de variable. Para este caso se encontraron 4 (Hombre(Marco), Pompeyano(Marco), Gobernante(Cesar) e IntentaAsesinar(Marco,Cesar)). *Este proceso ocurre en la función unificación entre los renglones 69 y 71*

Estas constantes son de suma importancia ya que son las que nos van a permitir realizar las modificaciones en el resto de la información guiándonos no por el contenido que este dentro de los paréntesis, sino que vamos a utilizar los predicados como valor para comparar. Ya que, una vez encontremos el mismo predicado, pero con una variable, lo que tenemos que hacer es reemplazar el valor que esta entre paréntesis por la constante que corresponda.

Este proceso se hace en el ciclo que ocurre entre las líneas 75 y 106. En donde utilizando expresiones regulares extrae el predicado y luego identifica si la información o las variables dentro del paréntesis de dicho predicado son una (x) o 2 (x,y).

Luego con esta información se ingresa al ciclo que altera sobre la base de conocimientos y trata de encontrar los respectivos valores de x o de x y de y según corresponda. Esto para hacer la modificación de las variables por sus respectivas constantes (en este caso específico del problema x corresponde a marco y y a César).

Luego, ingresa únicamente las líneas que no tienen más valores de x y de y adicionales. Es decir, información que solo cuente con constantes y predicados. Esta base de conocimientos final es la que podemos usar para la inferencia.

**Proceso de realización de inferencia lógica por refutación:**

Una vez con la base de conocimientos simplificada y sin ningún tipo de variable, es posible comenzar con el proceso de inferencia lógica por refutación. Este proceso va a ser un caso iterativo en donde a partir de un dato base buscamos "cancelarlo" con algún componente que haga parte de la base de conocimientos. Para esto, vamos a hacer un uso constante de la siguiente regla:

#Resolución unitaria

(a∨¬b),b

a

(a∨b),¬b

a

Todo esto con el fin de ir reduciendo la base de conocimientos hasta llegar a un punto en donde todo haya sido "cancelado" y por lo tanto se compruebe la veracidad de la pregunta inicial qué es Odia(Marco,Cesar). O por el contrario si está no logra ser comprobada generan a un resultado falso.

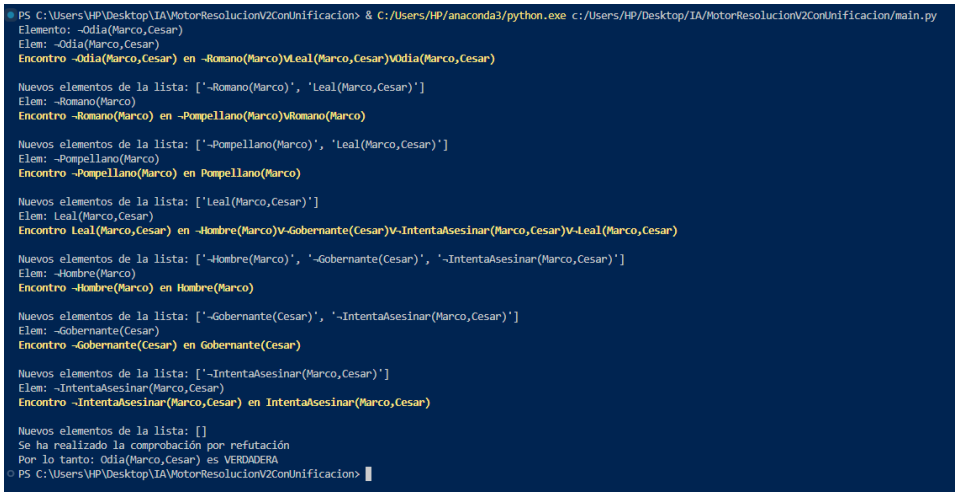
Por lo tanto, desde la función operar que empieza en la línea 189 hoy va a irse comprobando si esta inferencia ya pudo ser verificada o no. Está en compañía de la función actualizar que es la que se va a encargar del proceso de búsqueda de los predicados que sea posible eliminar. Y esta función que empieza en las líneas 157 lo que hace es buscar un elemento que sea "contrario" al que me pasan como parámetro. Ya que con base en las reglas resolución unitaria es la forma en la que podemos simplificar una expresión. Entonces en actualizar una vez se encuentra el elemento contrario. Podemos limpiarlo de la base de conocimiento, considerando o teniendo en cuenta el resto de los predicados que acompañaban al que acabé de cancelar. Para después en un proceso iterativo buscar el respectivo dato que me permita cancelar es información residual y así sucesivamente. Estos datos residuales van almacenarse en el parámetro lista. Y la base del conocimiento va a ir eliminando aquellas líneas ya utilizadas para qué no se caiga en un posible ciclo infinito.

El proceso va a terminar cuando ya no haya más elementos en la lista que sean necesarios cancelar por medio de la resolución unitaria que sería el caso en donde la pregunta inicial sería verdadera. O por el contrario cuando sigue habiendo elementos en la lista pero ya no hay más información en la base de conocimiento que se pueda utilizar lo que daría que la pregunta inicial sería falsa.

**Ejecución y Resultados finales:**

Para ejecutar el proyecto no es necesario ingresar ningún tipo de parámetro. La ubicación relativa del archivo utilizado esta quemada en la implementación

- **Proceso de inferencia por refutación (Proceso que se utiliza en la implementación con unificación y la implementación sin unificación):**

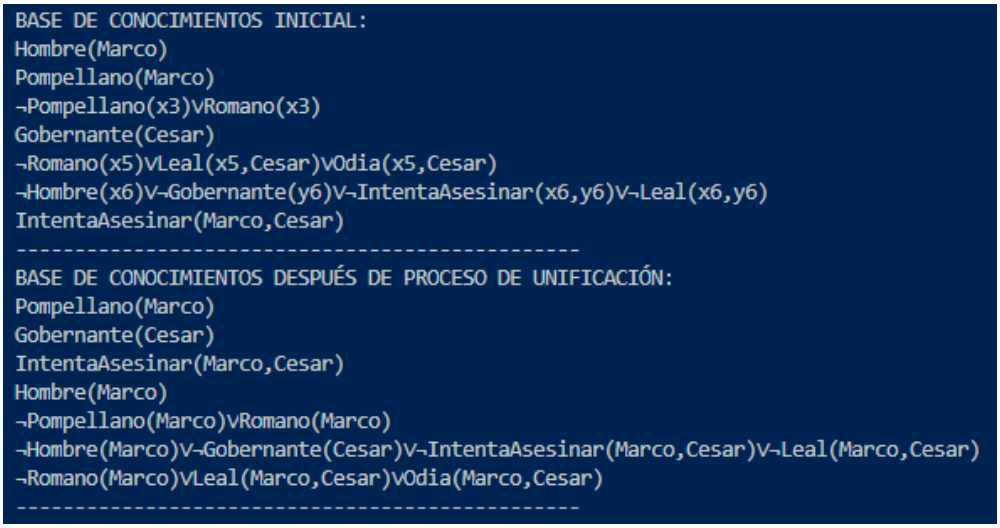



En esta información se encuentra para la primera iteración la negación, y en amarillo la negación acompañada de la sentencia en donde encontró el valor con el cual puedes realizar la resolución unitaria.

A partir de la iteración 2, se muestra la lista de valores que deben buscar en la base de conocimiento su respectivo “inverso” para aplicar la regla. “Elem” es el valor específico que está buscando su inverso para ese momento de tiempo. En amarillo se mantiene el mismo propósito.

Finalmente, cuando observamos que la lista está vacía se ha finalizado la comprobación por refutación, termina el ciclo y por lo tanto muestra el resultado de que la pregunta inicial es verdadera.

- **Proceso de unificación (Proceso que se utiliza únicamente en la implementación sin unificación):**



Acá lo que se muestra es el resultado una vez realizado el proceso de unificación. La primera lista es la que se extrae directamente del archivo de texto, mientras que la segunda es la lista resultante del proceso explicado de unificación y que posteriormente va a ser utilizada para la comprobación por refutación. Que va a generar el mismo resultado que se ve en la imagen anterior.