ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

“INTELIGENCIA ARTIFICIAL (ICCD523”

PROYECTO BIMESTRAL ROMPECABEZAS LÓGICO

Francis Alian Bravo Aleman

Brandon Ismael Freire Muesmeran

Johann Vladimir Pasquel Montenegro

Jorge Alejandro Torres Maldonado

Docente: Dr. Myriam Hernández

NOVIEMBRE 2025

Rompecabezas Lógico Usando Lógica Proposicional y Resolución con tabla de verdad.

OBJETIVOS

- Desarrollar e implementar un sistema computacional completo y verificable que automatice la resolución de rompecabezas lógicos mediante la aplicación rigurosa de los principios de lógica proposicional y algoritmos de model checking, demostrando la viabilidad de técnicas de inteligencia artificial simbólica para problemas de razonamiento deductivo.

- Desarrollar algoritmos de generación exhaustiva de tablas de verdad y verificación de entailment que garanticen corrección semántica y completitud lógica, optimizando el rendimiento para espacios de búsqueda de tamaño moderado.

- Diseñar e implementar un motor de lógica proposicional en Python que soporte de manera nativa todos los conectivos lógicos estándar (negación, conjunción, disyunción, implicación, bicondicional) con evaluación semántica eficiente.

PRESENTACIÓN DEL PROBLEMA

Este proyecto se centra en la aplicación de la lógica proposicional computacional para la resolución automatizada de acertijos lógicos. El problema específico para abordar consiste en determinar las propiedades ontológicas de un sujeto hipotético (un unicornio) y si este es mítico, mágico y si posee cuernos, basándose en un conjunto estricto de premisas condicionales.

Para lograr esto, se implementará una solución algorítmica en Python que modele las proposiciones lógicas y genere una tabla de verdad exhaustiva. Este enfoque permitirá evaluar todas las interpretaciones posibles de las premisas para deducir, mediante prueba formal, la validez de los argumentos y obtener una respuesta concluyente sobre los atributos del unicornio

DESCRIPCIÓN DEL MODELO

En este proyecto el rompecabezas del unicornio se modela como un problema de lógica proposicional clásica. La idea central es pasar del texto en lenguaje natural a un conjunto de proposiciones y reglas lógicas que luego podamos evaluar de forma sistemática mediante una tabla de verdad generada en Python.

Primero se definen las variables proposicionales básicas asociadas a las características del unicornio:

- **Mi**: "el unicornio es mítico"
- **I**: "el unicornio es inmortal"
- **Ma**: "el unicornio es mamífero"
- **Mo**: "el unicornio es mortal"
- **H**: "el unicornio tiene cuernos"
- **Mg**: "el unicornio es mágico"

Luego se traduce cada premisa del enunciado a fórmulas lógicas:

"Si el unicornio es mítico, entonces es inmortal"

$$Mi \rightarrow I$$

"Si no es mítico, entonces es un mamífero mortal"

$$\neg Mi \rightarrow (Ma \wedge Mo)$$

"Si el unicornio es inmortal o mamífero, entonces tiene cuernos"

$$(I \vee Ma) \rightarrow H$$

"El unicornio es mágico si tiene cuernos"

$$H \rightarrow Mg$$

DESCRIPCIÓN DEL MÉTODO UTILIZADO PARA RESOLVER EL ROMPECABEZAS

Se construye una base de conocimiento con las premisas del enunciado y luego se exploran todos los modelos posibles mediante una tabla de verdad. A partir de esos modelos, y del uso de un verificador de modelos (model checking), se decide qué cosas se pueden probar formalmente sobre el unicornio.

Se definen símbolos lógicos para cada propiedad importante: que el unicornio sea mítico, inmortal, mamífero, mortal, que tenga cuernos y que sea mágico. Después, las frases del problema se traducen a implicaciones lógicas usando las clases de logic.py (And, Or, Not, Implication). El conjunto completo de reglas se agrupa en una sola base de conocimiento (knowledge).

- Se genera una tabla de verdad que recorre todas las combinaciones posibles de verdad y falsedad para los símbolos. Para cada combinación se evalúa si la base de conocimiento es verdadera en ese "mundo". Solo se muestran o guardan las filas donde la base de conocimiento se cumple, porque son los modelos que respetan todas las premisas.

- Por otro lado, se usa model_check(knowledge, consulta) para hacer preguntas directas del tipo "¿se puede probar que el unicornio es mítico?", "¿se puede probar que tiene cuernos?", "¿se puede probar que es mágico?". Esta función revisa todos los modelos que satisfacen la base de conocimiento y responde True solo si la proposición consultada es verdadera en todos ellos.

IMPLEMENTACIÓN DEL CÓDIGO

Módulo logic.py

Se define una clase abstracta Sentence, de la cual heredan todas las otras sentencias lógicas. A partir de ella se implementan los distintos tipos de conectores: Symbol para las variables proposicionales, Not para la negación, And para la conjunción, Or para la disyunción, Implication para la implicación y Biconditional para la bicondicional.

Cada clase sabe cómo evaluarse sobre un modelo (un diccionario que asigna True o False a cada símbolo), cómo devolver su fórmula en forma de texto y qué símbolos contiene.

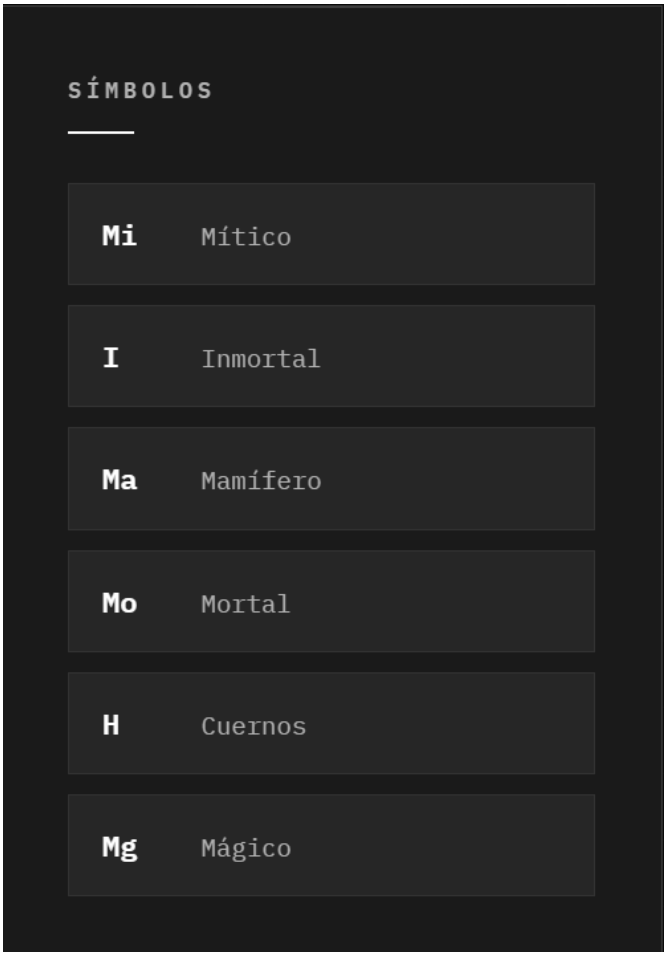
Se implementa la función model_check, que es la encargada de realizar el "model checking": a partir de una base de conocimiento (knowledge) y una consulta (query), recorre recursivamente todos los modelos posibles y devuelve True solo si en todos los modelos que satisfacen la base de conocimiento la consulta también se cumple.

Módulo proyecto3.py

```
from logic import *
import itertools

# 1. Definición de Símbolos (Átomos)
Mi = Symbol("Mítico")
I  = Symbol("Inmortal")
Ma = Symbol("Mamífero")
Mo = Symbol("Mortal")
H  = Symbol("Cuernos")
Mg = Symbol("Mágico")

# 2. Construcción de la Base de Conocimiento (KB)

knowledge = And(
    # Premisa 1:
    Implication(Mi, I),
    Implication(Not(Mi), And(Ma, Mo)),

    # Premisa 2:
    Implication(Or(I, Ma), H),

    # Premisa 3:
    Implication(H, Mg)
)
```

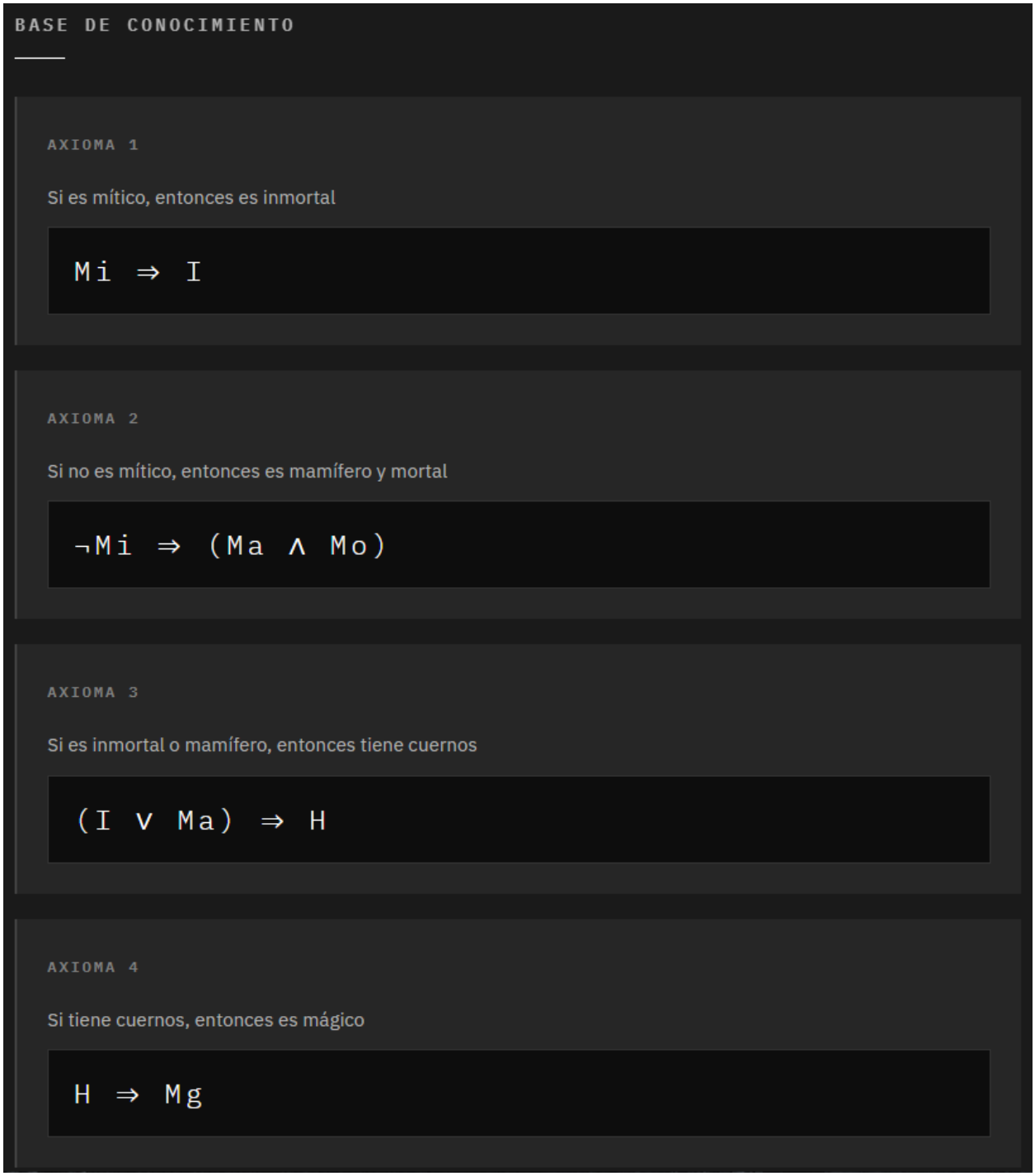
```python
# 3. Función para Generar e Imprimir la Tabla de Verdad
def imprimir_tabla_verdad(kb, simbolos):
    encabezados = [s.name for s in simbolos] + ["KB (Es válida?)"]
    print(f"{' | '.join(encabezados)}")
    print("-" * (len(encabezados) * 12))

    # Generar todas las combinaciones posibles de verdad (True/False)
    combinaciones = list(itertools.product([True, False], repeat=len(simbolos)))

    modelos_validos = 0

    for valores in combinaciones:
        modelo = dict(zip([s.name for s in simbolos], valores))

        # Evaluar si la Base de Conocimiento es verdadera en este modelo
        es_verdad = kb.evaluate(modelo)

        if es_verdad:
            modelos_validos += 1
            fila = [str(modelo[s.name]) for s in simbolos] + [str(es_verdad)]
            print(f"{' | '.join(f'{val:<5}' for val in fila)}")

    print(f"\nNro de modelos donde la KB se cumple: {modelos_validos}")

# 4. Resolución del Problema (Inferencia)
def resolver_preguntas():
    print("\n--- RESULTADOS DE INFERENCIA ---")

    # Pregunta 1: ¿Es mítico?
    es_mitico = model_check(knowledge, Mi)
    print(f"¿Se puede probar que es Mítico? {es_mitico}")

    # Pregunta 2: ¿Es mágico?
    es_magico = model_check(knowledge, Mg)
    print(f"¿Se puede probar que es Mágico? {es_magico}")

    # Pregunta 3: ¿Tiene cuernos?
    tiene_cuernos = model_check(knowledge, H)
    print(f"¿Se puede probar que tiene Cuernos? {tiene_cuernos}")

# --- EJECUCIÓN PRINCIPAL ---
if __name__ == "__main__":
    # Lista de símbolos para la tabla
    lista_simbolos = [Mi, I, Ma, Mo, H, Mg]

    print("--- TABLA DE VERDAD (Solo filas válidas/consistentes) ---")
    imprimir_tabla_verdad(knowledge, lista_simbolos)

    resolver_preguntas()
```

Aquí se importan las clases de logic.py y se definen los símbolos atómicos del enunciado: Mi para "Mítico", I para "Inmortal", Ma para "Mamífero", Mo para "Mortal", H para "Cuernos" y Mg para "Mágico". Estos símbolos son los que representan, dentro del programa, las propiedades que queremos analizar.



A partir de ellos se construye la base de conocimiento (knowledge) usando las clases And, Or, Not e Implication. Por ejemplo, la regla "si el unicornio es mítico, entonces es inmortal" se traduce como Implication(Mi, I), mientras que la frase "si no es mítico, entonces es mamífero y mortal" se expresa combinando Not(Mi), And(Ma, Mo) y otra implicación.

De forma similar se codifican las otras dos premisas: "si es inmortal o mamífero, entonces tiene cuernos" y "si tiene cuernos, entonces es mágico". El resultado es una única expresión And(...) que agrupa todas las reglas y representa formalmente todo lo que sabemos del unicornio.



Sobre esa base de conocimiento se programan las funciones que generan resultados. La función imprimir_tabla_verdad recibe como parámetros la base de conocimiento y una lista de símbolos, y se encarga de construir la tabla de verdad. Internamente, usa itertools.product para generar todas las combinaciones posibles de valores True/False para los símbolos definidos (en este caso, 2^6 = 64 combinaciones). Para cada combinación se construye un modelo asociando el nombre de cada símbolo con su valor de verdad correspondiente y se evalúa la base de conocimiento con kb.evaluate(modelo).

PRUEBAS Y RESULTADOS

Tabla de Verdad - Modelos Consistentes

Tabla de Verdad

64	6	6	9.4%
COMBINACIONES	MODELOS VÁLIDOS	VARIABLES	CONSISTENCIA

MI	I	MA	MO	H	MG	KB
1	1	0	0	1	1	1
1	1	1	0	1	1	1
1	1	0	1	1	1	1
0	0	1	1	1	1	1
0	1	1	1	1	1	1
1	1	1	1	1	1	1

De las 64 combinaciones posibles (2^6), el sistema identificó exactamente 6 modelos que satisfacen todas las premisas de la base de conocimiento. Estos modelos representan los únicos escenarios lógicamente consistentes con las reglas del problema. Para visualizar los resultados, utilice las siguientes capturas de pantalla:

Resultados de Inferencia

¿Se puede probar que es Mítico?	¿Se puede probar que es Mágico?	¿Se puede probar que tiene Cuernos?
⊥ INDETERMINADO	T VERDADERO	T VERDADERO

CONCLUSIÓN

Mediante el método de resolución por tabla de verdad, el sistema ha demostrado formalmente que el unicornio necesariamente posee cuernos (H ⊨ KB) y es mágico (Mg ⊨ KB). La propiedad de ser mítico (Mi) no puede ser determinada unívocamente, ya que tanto Mi como ¬Mi son consistentes con la base de conocimiento, resultando en múltiples modelos válidos que satisfacen todas las premisas establecidas.

- **¿Es mítico?** False - No se puede demostrar que el unicornio sea mítico. Existen modelos válidos donde es mítico y otros donde no lo es.
- **¿Es mágico?** True - Se puede demostrar que el unicornio es necesariamente mágico. En todos los modelos consistentes, el unicornio es mágico.
- **¿Tiene cuernos?** True - Se puede demostrar que el unicornio necesariamente tiene cuernos. En todos los modelos consistentes, el unicornio tiene cuernos.

CONCLUSIONES Y DISCUSIÓN

- El sistema logró resolver de forma consistente el rompecabezas del unicornio usando únicamente lógica proposicional y tablas de verdad. A partir de las premisas del enunciado y de la base de conocimiento construida en Python, el programa encontró 6 modelos en los que todas las reglas se cumplen.

- Trabajar con tablas de verdad escala mal cuando el número de proposiciones crece, porque el número de combinaciones explota muy rápido. En este rompecabezas, con unas pocas variables, todavía es manejable y hasta didáctico ver todas las filas válidas. Pero para bases de conocimiento más grandes habría que pensar en otros enfoques más eficientes.

- Se muestra cómo un conjunto de reglas escritas en lenguaje natural puede transformarse en un sistema que razona de forma automática y transparente. A futuro, se podría extender el trabajo probando otros rompecabezas, añadiendo más reglas al mundo del unicornio o integrando el motor lógico con interfaces más completas.