Temas Tratados en el Trabajo Práctico 4

- Representación del Conocimiento y Razonamiento Lógico.
- Estrategias de resolución de hipótesis: Encadenamiento hacia Adelante, Encadenamiento hacia Atrás y Resolución por Contradicción.
- Representación basada en circuitos.

Ejercicios Teóricos

1. ¿Qué es una inferencia?

Una inferencia es el proceso de razonamiento que permite a un sistema de inteligencia artificial deducir conclusiones lógicas a partir de un conjunto de hechos y reglas, aplicando estos a una base de conocimientos. Es la habilidad de generar conocimiento nuevo, analizando y combinando la informacion que el sistema ya posee, permitiéndole a este ir más allá de los hechos explícitos para descubrir conocimiento implícito.

2. ¿Cómo se verifica que un modelo se infiere de la base de conocimientos?

Verificar que un modelo se infiere de una base de conocimientos significa asegurar que es una consecuencia lógica de los hechos y reglas existentes. En otras palabras, se confirma que si toda la base de conocimientos es verdadera, la conclusión o el modelo también debe serlo. Este proceso se puede llevar a cabo de tres maneras principales: mediante la demostración lógica, construyendo una cadena de razonamiento formal desde las premisas hasta la conclusión; a través de la resolución por contradicción, asumiendo que el modelo es falso y demostrando que esta suposición lleva a una inconsistencia con la base de conocimientos; o por interpretación semántica, evaluando si el modelo es verdadero en todos los escenarios posibles donde la base de conocimientos también lo es.

3. Observe la siguiente base de conocimiento:

$R1 : b \land c \rightarrow a$

$R2 : d \land e \rightarrow b$

$R3 : g \land e \rightarrow b$

$R4 : e \rightarrow c$

$R5 : d$

$R6 : e$

$R7 : a \wedge g \rightarrow f$

3.1 ¿Cómo se puede probar que $a = True$ a través del encadenamiento hacia adelante? Este método solamente usa reglas ya incorporadas a la base de conocimiento para inferir la hipótesis, ¿qué propiedad debe tener el algoritmo para asegurar que esta inferencia sea posible?

3.2 ¿Cómo se puede probar que $a = True$ a través del encadenamiento hacia atrás? Este método asigna un valor de verdad a la hipótesis y deriva las sentencias de la base de conocimiento, ¿qué propiedad debe tener el algoritmo para asegurar que esta derivación sea posible?

3.3 Exprese la base de conocimiento en su Forma Normal Conjuntiva. A continuación, demuestre por contradicción que $a = True$.

Respuesta Ejercicio 3

3.1

Partiendo de los hechos conocidos:

- R5: d
- R6: e
- R4: e -> c.
- R2: d^e -> b
- R1: b^c -> a

De esta manera infiere que a es verdadero

3.2

Partiendo de la hipotesis a. Debemos demostrar que b y c son verdaderos

- R1: a -> b^c

Que d y e son hechos conocidos implica que b es verdadero

- R2: b -> d^e
- R5: d
- R6: e

Que e es un hecho conocido implica que c es verdadero

- R4 e -> c

Finalmente, dado que b y c son verdaderos, verifico la hipotesis, a es verdadero.

Ambos algoritmos (encadenamiento hacia adelante y hacia atras) requieren completitud para ser efectivos en la derivacion de conclusiones. Esto significa que se puede derivar cualquier hecho que sea logicamente implicado por la base de conocimientos.

3.3

Toda sentencia en logica proposicional es equivalente logicamente a una conjuncion de disyunciones de literales.

Recordando la Base de Conocimiento:

$R1 : b \land c \rightarrow a$

1. Eliminar la implicacion logica sustituyendo a->b por -a v b

   ```
   -(b^c) v a
   ```
2. Aplicar negacion solo a literales utilizando Morgan:

   ```
   (-b v -c) v a
   ```

De manera similar:

$R2 : d \land e \rightarrow b$

```
(-d v -e) v b
```

$R3 : g \land e \rightarrow b$

```
(-g v -e) v b
```

$R4 : e \rightarrow c$

```
-e v c
```

$R5 : d$

$R6 : e$

$R7 : a \land g \rightarrow f$

```
(-a v -g) v f
```

Demostracion por Contradiccion que a = true. Para ello **asumimos: a = Falso**

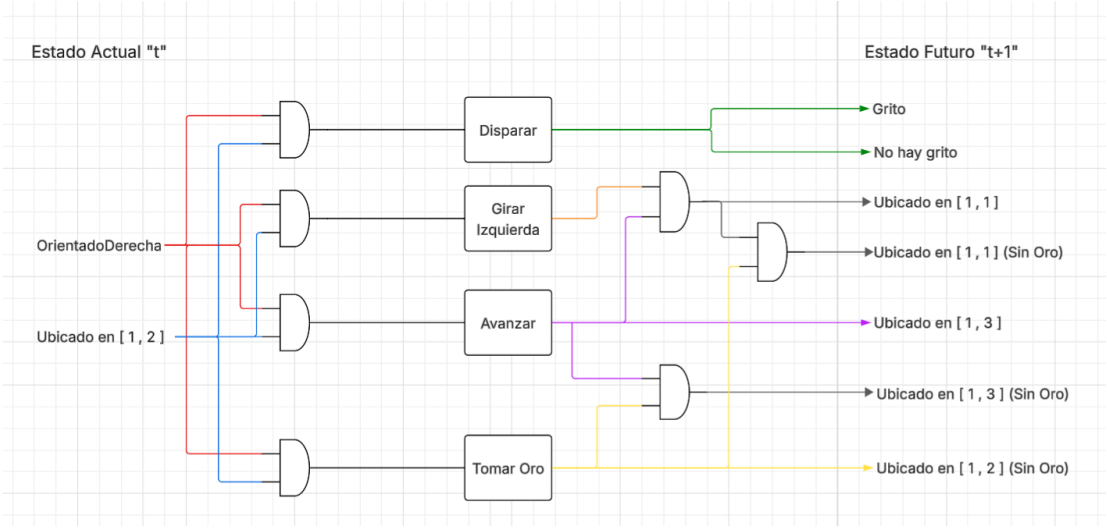
Partiendo de la base de conocimiento en su forma normal conjuntiva:

- De la cláusula 4 (-e v c). Por R6 'e' es verdadero, por lo tanto '-e' es falso asi deducimos que 'c' es verdadero.
- De la cláusula 1 (-b v -c v a). Asumiendo 'a' falso y sabemos que 'c' es verdadero. para que la clausula sea verdadera. '-b' debe ser verdadero, es decir, 'b' es falso.

- De la cláusula 2 (-d v -e v b). Por R5 'd' es verdadero, Por R6 'e' es verdadero y anteriormente concluimos que 'b' es falso. Por tanto todos los literales de la clausula 2 son falsos. Por lo tanto nuestra suposicion de que a es falso es incorrecta.

4. Diseñe con lógica proposicional basada en circuitos las proposiciones *OrientadoDerecha* y *Agente ubicado en la casilla [1,2]* para el mundo de wumpus de 4x4. Dibuje el circuito correspondiente.

Para esta consigna, utilizamos las percepciones de "OrientadoDerecha" y "Agente ubicado en la casilla [1,2]", además de las acciones "Disparar", "TomarOro", "Avanzar" y "GirarIzquierda", evaluando el estado actual "t" y un estado futuro "t+1".



5. El nonograma es un juego en el cual se posee un tablero en blanco y cada fila y columna presenta información sobre la longitud de un bloque en dicha fila/columna. Además, la leyenda puede indicar más de un número, indicando esto que existen varios bloques de las longitudes mostradas por la leyenda y en el mismo orden, separados por al menos un espacio vacío.

Resuelva el nonograma de la imagen de abajo escribiendo en primer lugar cada regla que puede incorporarse a la base de conocimientos inicial e incorporando cada inferencia que realice.

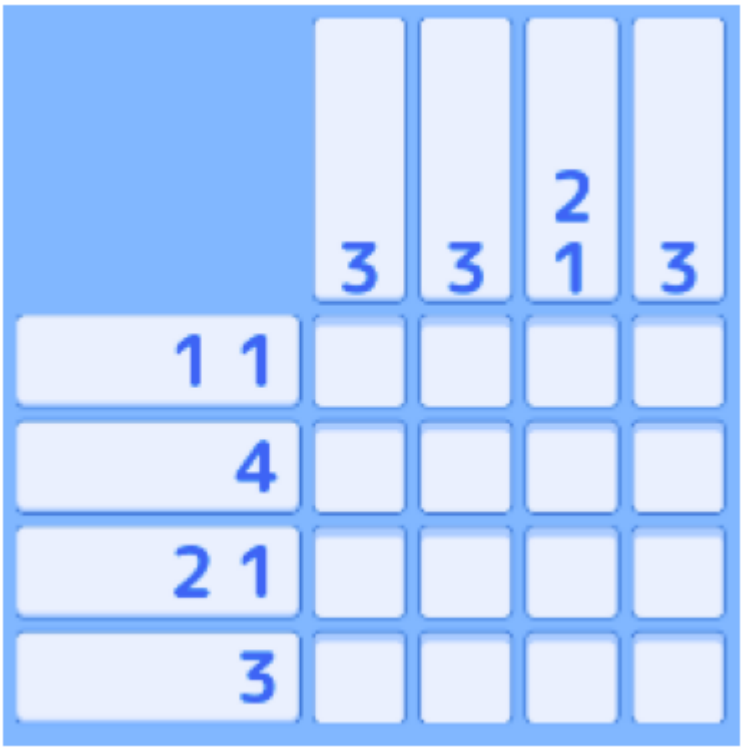
In [1]:
```
import requests
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt

# URL directa de Google Drive
url = "https://drive.google.com/uc?export=view&id=1SKiXvrI_TX-U4sbw60TYSRmaNYyFi

# Descargar la imagen
response = requests.get(url)
img = Image.open(BytesIO(response.content))

# Mostrar la imagen
plt.imshow(img)
```

```
plt.axis('off')  # Ocultar ejes
plt.show()
```



Base de conocimiento - Reglas (R)

R1 : [(1,1) ^ (1,3)] v [(1,2) ^ (1,4)] v [(1,1) ^ (1,4)] //Fila 1

R2 : (2,1) ^ (2,2) ^ (2,3) ^ (2,4) //Fila 2

R3 : [((3,1) ^ (3,2))^(3,4)] v [(1,1) ^ ((3,3) ^ (3,4))] //Fila 3

R4 : [(4,1) ^ (4,2) ^ (4,3)] v [(4,2) ^ (4,3) ^ (4,4)] //Fila 4

R5 : [(1,1) ^ (2,1) ^ (3,1)] v [(2,1) ^ (3,1) ^ (4,1)] //Columna 1

R6 : [(1,1) ^ (1,2) ^ (1,3)] v [(1,2) ^ (1,3) ^ (1,4)] //Columna 2

R7 : [((1,3) ^ (2,3))^(4,3)] v [(1,3) ^ ((3,3) ^ (4,3))] //Columna 3

R8: [(1,4) ^ (2,4) ^ (3,4)] v [(2,4) ^ (3,4) ^ (4,4)] //Columna 4

R9 : (2,1)

R10 : (2,2)

R11 : (2,3)

R12 : (2,3)

Nuevos conocimientos o reglas a través de inferencia

R13: R2 ^ R10 ⇒ [(1,3) ^ (2,3)] ^ (4,1)

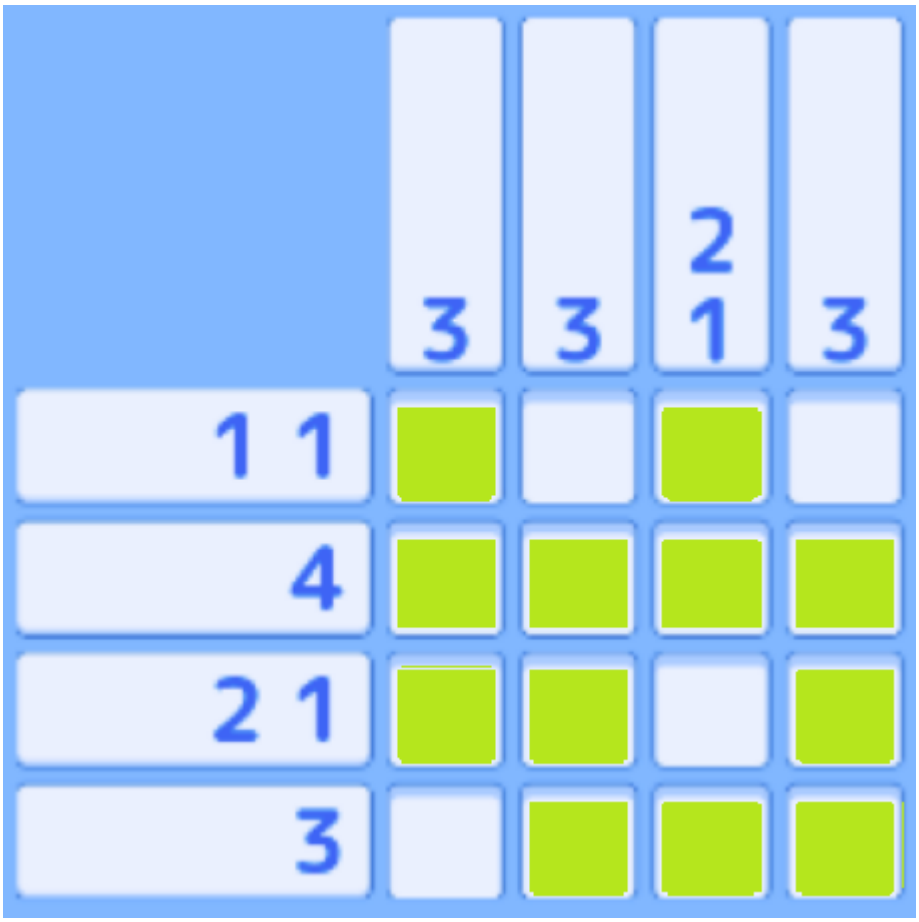
R14 : R13 ⇒ [(1,1) ^ (1,3)]

R15 : R14 ⇒ [(1,1) ^ (2,1) ^ (3,1)]

R16 : R13 ^ R14 ⇒ (2,4) ^ (3,4) ^ (4,4)

R17 : R16 ⇒ (4,2) ^ (4,3) ^ (4,4)

R18 : R16 ^ R17 ⇒ [((3,1) ^ (3,2))^(3,4)] // FIN



Ejercicios de Implementación

6. Implementar un motor de inferencia con encadenamiento hacia adelante. Pruébelo con las proposiciones del ejercicio 3.

7. Implementar un motor de inferencia con encadenamiento hacia atrás. Pruébelo con las proposiciones del ejercicio 3.

8. Implementar un motor de inferencia por contradicción que detecte si el conjunto de proposiciones del ejercicio 3 es inconsistente.

Resolucion Ejericios de Implementacion

In [ ]:
```
class MotorInferencia:
    def __init__(self, base_conocimiento):
```

```
        """
        Inicializa el motor de inferencia con una base de conocimiento.

        Args:
            base_conocimiento (list): Lista de reglas en forma de cláusulas Horn
                Cada regla es una tupla (antecedentes, consecuente), donde:
                - antecedentes: lista de símbolos (str) que deben ser verdaderos
                - consecuente: símbolo (str) que se infiere si los antecedentes
                Los hechos se representan como reglas con antecedentes vacíos.
        """
        self.base_conocimiento = base_conocimiento
        self.hechos = set()
        self.fnc = self._convertir_a_fnc()

    def _convertir_a_fnc(self):
        """
        Convierte la base de conocimiento a Forma Normal Conjuntiva (FNC).

        Returns:
            list: Lista de cláusulas en FNC, donde cada cláusula es una lista de
        """
        fnc = []

        for antecedentes, consecuente in self.base_conocimiento:
            if not antecedentes:  # Hecho
                fnc.append([consecuente])
            else:  # Regla: antecedentes → consecuente
                # Convertir a: ¬antecedente1 ∨ ¬antecedente2 ∨ ... ∨ consecuente
                clausula = [f"¬{antecedente}" for antecedente in antecedentes]
                clausula.append(consecuente)
                fnc.append(clausula)

        return fnc

    def agregar_hecho(self, hecho):
        """Agrega un hecho a la base de conocimientos."""
        self.hechos.add(hecho)

    def encadenamiento_adelante(self):
        """
        Realiza encadenamiento hacia adelante para derivar todos los hechos posi

        Returns:
            set: Conjunto de todos los hechos derivados (incluyendo los iniciale
        """
        # Inicializar con los hechos conocidos
        nuevos_hechos = set(hecho for antecedentes, hecho in self.base_conocimie

        # Aplicar reglas hasta que no se puedan derivar nuevos hechos
        cambios = True
        while cambios:
            cambios = False
            for antecedentes, consecuente in self.base_conocimiento:
                # Si todos los antecedentes están en los hechos y el consecuente
                if all(antecedente in nuevos_hechos for antecedente in anteceden
                    nuevos_hechos.add(consecuente)
                    cambios = True

        self.hechos = nuevos_hechos
        return self.hechos
```

```
    def encadenamiento_atras(self, objetivo, objetivos_visitados=None):
        """
        Realiza encadenamiento hacia atrás para determinar si un objetivo es ver

        Args:
            objetivo (str): El símbolo que se desea probar.
            objetivos_visitados (set): Conjunto de objetivos ya visitados (para

        Returns:
            bool: True si el objetivo puede ser probado, False en caso contrario
        """
        if objetivos_visitados is None:
            objetivos_visitados = set()

        # Evitar ciclos infinitos
        if objetivo in objetivos_visitados:
            return False

        # Si el objetivo ya es un hecho conocido, retornar True
        if objetivo in self.hechos:
            return True

        # Buscar reglas que tengan el objetivo como consecuente
        reglas_aplicables = []
        for antecedentes, consecuente in self.base_conocimiento:
            if consecuente == objetivo:
                reglas_aplicables.append(antecedentes)

        # Si no hay reglas que lleven a este objetivo, no se puede probar
        if not reglas_aplicables:
            return False

        # Para cada regla aplicable, verificar si todos sus antecedentes pueden
        for antecedentes in reglas_aplicables:
            # Verificar cada antecedente recursivamente
            todos_probados = True
            nuevos_objetivos_visitados = objetivos_visitados.copy()
            nuevos_objetivos_visitados.add(objetivo)

            for antecedente in antecedentes:
                if not self.encadenamiento_atras(antecedente, nuevos_objetivos_v
                    todos_probados = False
                    break

            if todos_probados:
                return True

        return False

    def _evaluar_clausula(self, clausula, asignacion):
        """
        Evalúa una cláusula con una asignación de valores de verdad.

        Args:
            clausula (list): Lista de literales.
            asignacion (dict): Diccionario con los valores de verdad de las vari

        Returns:
            bool: True si la cláusula es verdadera, False en caso contrario.
```

```
        """
        for literal in clausula:
            if literal.startswith("¬"):
                variable = literal[1:]
                valor = asignacion.get(variable, None)
                if valor is False:
                    return True  # La cláusula es verdadera si un literal es ver
            else:
                valor = asignacion.get(literal, None)
                if valor is True:
                    return True  # La cláusula es verdadera si un literal es ver
        return False

    def _es_consistente(self, asignacion):
        """
        Verifica si una asignación de valores es consistente con la base de cono

        Args:
            asignacion (dict): Diccionario con los valores de verdad de las vari

        Returns:
            bool: True si la asignación es consistente, False en caso contrario.
        """
        for clausula in self.fnc:
            if not self._evaluar_clausula(clausula, asignacion):
                return False
        return True

    def inferencia_por_contradiccion(self, objetivo):
        """
        Realiza inferencia por contradicción para probar un objetivo.

        Args:
            objetivo (str): El símbolo que se desea probar.

        Returns:
            bool: True si el objetivo puede ser probado por contradicción, False
        """
        # Crear una asignación inicial con los hechos conocidos
        asignacion = {}
        for hecho in self.hechos:
            asignacion[hecho] = True

        # Asumir que el objetivo es falso
        asignacion[objetivo] = False

        # Verificar si esta asignación es consistente con la base de conocimient
        consistente = self._es_consistente(asignacion)

        # Si es inconsistente, entonces la suposición es falsa y el objetivo es
        return not consistente
```

Utilizamos la clase MotorInferencia para resolver los enunciados 6, 7 y 8 del trabajo practico.

In [ ]:
```
def main():
    # Base de conocimiento
    base_conocimiento = [
        (['b', 'c'], 'a'),    # R1: b ∧ c → a
```

```
        (['d', 'e'], 'b'),    # R2: d ∧ e → b
        (['g', 'e'], 'b'),    # R3: g ∧ e → b
        (['e'], 'c'),         # R4: e → c
        ([], 'd'),            # R5: d
        ([], 'e'),            # R6: e
        (['a', 'g'], 'f')     # R7: a ∧ g → f
    ]

    # Instanciar el motor de inferencia
    motor = MotorInferencia(base_conocimiento)

    # Establecer objetivo
    objetivo = 'a'
    print(f"Objetivo a probar: {objetivo}\n")

    # --- PRINTS ---
    # ENCADENAMIENTO ADELANTE
    print("--- Demostración con encadenamiento hacia adelante ---")

    # Ejecutar encadenamiento hacia adelante
    hechos_derivados = motor.encadenamiento_adelante()

    print(f"Hechos derivados: {hechos_derivados}")
    if objetivo in hechos_derivados:
        print(f"✅ ¡El objetivo '{objetivo}' es verdadero!")
    else:
        print(f"❌ El objetivo '{objetivo}' no se puede probar.")

    # ENCADENAMIENTO ATRAS
    print("\n--- Demostración con encadenamiento hacia atrás ---")

    # Ejecutar encadenamiento hacia atrás
    es_verdadero_atras = motor.encadenamiento_atras(objetivo)

    if es_verdadero_atras:
        print(f"✅ ¡El objetivo '{objetivo}' es verdadero!")
    else:
        print(f"❌ El objetivo '{objetivo}' no se puede probar.")

    # INFERENCIA POR CONTRADICCION
    print("\n--- Verificación de consistencia por contradicción ---")

    motor_contradiccion = MotorInferencia(base_conocimiento)
    # Se simula la inicialización con los hechos
    motor_contradiccion.agregar_hecho('d')
    motor_contradiccion.agregar_hecho('e')

    # Ejecutar inferencia por contradicción
    es_verdadero_contradiccion = motor_contradiccion.inferencia_por_contradiccio

    if es_verdadero_contradiccion:
        print(f"✅ La base de conocimiento es inconsistente si asumimos que '{ob
    else:
        print(f"❌ La base de conocimiento es consistente si asumimos que '{obj

if __name__ == "__main__":
    main()
```

```
Objetivo a probar: a

--- Demostración con encadenamiento hacia adelante ---
Hechos derivados: {'e', 'c', 'b', 'a', 'd'}
✅ ¡El objetivo 'a' es verdadero!

--- Demostración con encadenamiento hacia atrás ---
✅ ¡El objetivo 'a' es verdadero!

--- Verificación de consistencia por contradicción ---
✅ La base de conocimiento es inconsistente si asumimos que 'a' es falso, por lo tanto, 'a' es verdadero.
```

Bibliografía

Russell, S. & Norvig, P. (2004) *Inteligencia Artificial: Un Enfoque Moderno*. Pearson Educación S.A. (2a Ed.) Madrid, España

Poole, D. & Mackworth, A. (2023) *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press (3a Ed.) Vancouver, Canada