Representación del conocimiento

Práctica 1

Rubén García
Yuhua Zhan
Javier Mier

Octubre 2023

1 Introducción

En la siguiente práctica se va a implementar el encadenamiento hacia delante en lógica proporcional y se va a desarrollar un algoritmo que decida si el encadenamiento es completo o no, con las siguientes características:

- No trata con paréntesis, ni con negaciones.
- En las implicaciones, primero se aborda el conjunto de cláusulas de la parte derecha de la implicación, luego la izquierda y finalmente la propia implicación.
- Cada elemento de la base de conocimiento, sea regla o hecho, se trata como si estuviesen unidas por conjunciones.
- Si un grupo de clausulas se encuentran unidas por conjunciones y disyunciones, se priorizan primero las disyunciones.

El diccionario utilizado es el siguiente:

$+$	Disyunción
$*$	Conjunción
$:$	Implicación

2 Planteamiento del problema

2.1 Clase `reader`

```
class reader:
    def __init__(self, file):
        self.file = file
        self.data = self.read_file()

    def read_file(self):
        with open(self.file, 'r') as f:
            data = f.readlines()
        for i in range(len(data)):
            data[i] = data[i].replace('\n', '')
        return data

    def get_data(self):
        return self.data
```

La base de conocimiento se encuentra en un fichero `BC_X.txt`, donde la `X` es un número entero. La lectura de este fichero, siguiendo las indicaciones anteriores, para el programa pueda trabajar con la base de conocimiento. Es la clase `reader` la que se encarga de leer los ficheros que contienen la base de conocimiento. Para que pueda leer el fichero de la BC correctamente, es necesario que éste se encuentre en el mismo directorio que el código.

2.2 Clase `rule`

```
class rule:
    def __init__(self, clausulas, consecuente):
        self.clausulas = clausulas
        self.consecuente = consecuente

    def disparar(self):
        if self.consecuente in base_conocimiento:
            return False
        for clausula in self.clausulas:
            if clausula not in base_conocimiento:
                return False
        base_conocimiento.update({self.consecuente:True})
        return True
```

La clase `rule` es una clase especial utilizada para guardar las reglas que se leen de la BC. La clase almacena en su interior una lista de todas las clausulas y el consecuente. La función `disparar` trata de comprobar si se puede añadir su consecuente a la BC. Si el consecuente ya se encuentra en la BC o no todas las clausulas se encuentran en la BC, se devuelve False; en caso contrario, se añade el consecuente a la BC y se devuelve True.

2.3 `main`

2.3.1 Ejecución del `main`

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

2.3.2 Código del `main`

```
print("Encadenamiento hacia delante")
fichero = reader("BC_6.txt")
# Se recorre todas las filas del fichero
for fila in fichero.get_data():
    # Busqueda de reglas y hechos
    if (re.search("^.*:.$",fila)):
        #divido en clausulas y consecuente
        division = fila.split(":")
        clausulas, consecuente = division[0], division[1]
        clausulas = clausulas.split("*")
        reglas.append(rule(clausulas, consecuente))
    else:
        base_conocimiento.update({fila:True})
        hechos.append(fila)

    for char in fila:
        if (re.search("[a-z]", char) and (not char in
proposiciones)):
            proposiciones.append(char)
```

En este fragmento de código se lee la base de conocimiento. Dividimos cada linea de clausula y consecuente, la cláusula es la parte de la sentencia que está a la "izquierda" de la implicación (que en nuestro caso se representará con el símbolo **:**). De la siguiente forma, guardaremos todas las proposiciones en una lista para ellas, definiéndolas como cualquier letra del alfabeto.

```
loop = True
while loop:
    loop = False
    for regla in reglas:
        loop = loop or regla.disparar()

base_completa = []
for elem in base_conocimiento.keys():
    base_completa.append(elem)

print("BC encadenamiento hacia delante:\t", base_completa)
```

En el bucle `while loop` se disparan todas las reglas y en caso de que alguna haya sido disparada se reevaluan todas las reglas en caso de que haya habido algún cambio. En el siguiente bucle añadimos todo el conocimiento que ha sido inferido por el encadenamiento hacia delante, imprimiendo el resultado.Faltaría comprobar que la base de conocimiento es completa.

```
valores_proposiciones_bool = [[] for i in range(len(
proposiciones))]
for i in range(2 ** len(proposiciones)):
    for j in range(len(proposiciones)):
        valores_proposiciones_bool[j].append(bool(i & (2 ** j))
)
```

En este trozo de código, asignamos a cada proposición una lista de valores 0 y 1 que se tratarán como valores falso y verdadero respectivamente. Un ejemplo de como resulta es el siguiente:

	p	q	r
μ_1	0	0	0
μ_2	1	0	0
μ_3	0	1	0
μ_4	1	1	0
μ_5	0	0	1
μ_6	1	0	1
μ_7	0	1	1
μ_8	1	1	1

```
    def obtener_valor(elem):
        if re.search("^.{1}$", elem):
            return valores_proposiciones_bool[proposiciones.index(
    elem)].copy()
        else:
            lista_or = elem.split("+")
            lista_valores = []
            for elem in lista_or:
                lista_valores.append(obtener_valor(elem))

            lista_retorno = lista_valores[0]

            for i in range(len(lista_retorno)):
                for lista in lista_valores[1:]:
                    lista_retorno[i] = lista_retorno[i] or lista[i]
            return lista_retorno
```

Aquí identificamos si el elemento pasado es una proposición o una disyunción de ellas. Si es una proposición, se obtiene su valor a partir de la tabla de verdad de ellas, en caso contrario, se obtienen las tablas de verdad de todos los elementos de la disyunción y se hace la operación lógica `or`

```
    #Lista que contendra los valores booleanos de todos los
    elementos de la base de conocimiento
    valores_tablas_elementos = []
    #Obtener los valores de verdad de los hechos y anhadirlos a la
    lista
    for elem in hechos:
        valores_tablas_elementos.append(obtener_valor(elem))

    #Obtener los valores de verdad de las reglas y anhadirlos a la
    lista
    for regla in reglas:
        #Lista que contendra los valores booleanos de todas las
    clausulas de la regla (parte izquierda)
        lista_valores_clausulas = []
        #Valores booleanos del consecuente (parte derecha)
        valores_consecuente = obtener_valor(regla.consecuente)
        #Obtener los valores booleanos de las clausulas y
    anhadirlos a la lista
        for clausula in regla.clausulas:
            lista_valores_clausulas.append(obtener_valor(clausula))

        #Obtener la tabla de verdad de la regla
        for i in range(len(valores_consecuente)):
```

```
            valor_clausulas = True
            for lista in lista_valores_clausulas:
                valor_clausulas = valor_clausulas and lista[i]

            valores_consecuente[i] = (not valor_clausulas) or
valores_consecuente[i]

        valores_tablas_elementos.append(valores_consecuente)
```

En éste ultimo fragmento de código, se obtienen las tablas de verdad de todos los elementos de la BC, ya sean hechos o reglas.

```
    BC_tabla_verdad_bool = []
    #Obtener la tabla de verdad de la base de conocimiento
    for i in range(len(valores_proposiciones_bool[0])):
        valor_tabla = True
        for elem in valores_tablas_elementos:
            valor_tabla = valor_tabla and elem[i]
        BC_tabla_verdad_bool.append(valor_tabla)
```

En esta sección de código obtenemos la tabla de verdad de toda la BC, equivalente a un and lógico con todos los valores de verdad de todos los elementos del fragmento de código anterior.

```
    BC_tabla_verdad = proposiciones.copy()

    # Comprobar cuales de las proposiciones estan en la base de
    conocimiento
    for i in range(len(valores_proposiciones_bool)):
        for j in range(len(BC_tabla_verdad_bool)):
            # Si en la tabla de verdad es cierto un valor y en la
    proposicion es falso,
            # se elimina de la base de conocimiento
            if (not valores_proposiciones_bool[i][j] and
    BC_tabla_verdad_bool[j]):
                BC_tabla_verdad.remove(proposiciones[i])
                break

    print("BC tabla de verdad:\t\t\t", BC_tabla_verdad)
```

Aquí obtenemos todos los elementos deducibles de la base de conocimiento a través de la tabla de verdad. En el inicio asumimos que todas las proposiciones forman parte de la BC, pero en caso de que alguna tenga valor falso cuando en la BC es verdadero, se elimina de ésta.

```
    # Comprobar que todos los elementos de la tabla de verdad estan
     en
    # la base de conocimiento que hemos obtenido por el
    encadenamiento hacia delante
    completa = True
    for elem in BC_tabla_verdad:
        if not elem in base_completa:
            completa = False
            break

    if completa:
        print("La base de conocimiento es completa")
    else:
```

```
12          print("La base de conocimiento no es completa")
```

Comprobamos que todos los elementos deducidos a través de la tabla de verdad han sido deducidos mediante encadenamiento hacia delante. Si todos se encuentran contenidos, la BC es completa, y en caso contrario no lo es.

3 Eficacia

Para comprobar el correcto funcionamiento del código planteado se ha utilizado las siguientes bases de conocimiento:

1. Reglas clausula de Horn y hechos clausula de Horn. Este caso tiene que ser siempre completo.

   `BC_3.txt` es completo.

   ```
   p
   p*q:r
   ```

2. Reglas no clausula de Horn y hecho clausula de Horn. Este caso puede ser incompleto.

   `BC_1.txt` es incompleto.

   ```
   p+q+r:s
   p*q:r
   p
   p:q
   p+q:r
   ```

   `BC_2.txt` es incompleto.

   ```
   p+q:r
   p
   ```

   `BC_5.txt` es completo.

   ```
   p+q:r
   s
   t
   s*t:u
   ```

3. Reglas no clausula de Horn y hechos no clausula de Horn. Este caso puede ser incompleto.

   `BC_4.txt` es completo.

   ```
   p
   p+q
   p+q:r
   ```

   `BC_6.txt` es incompleto.

   ```
   p+q+r:s
   p+q
   ```

Debido a que las bases de conocimiento eran pequeñas y simples, hemos comprobado en primera instancia si las BCs eran completas o incompletas y después hemos ejecutado el programa. En todos los casos, el resultado alcanzado por el programa es el mismo que el obtenido haciéndolo manualmente