Laboratorio 1: Lógica en Inteligencia Artificial

Curso:Inteligencia Artificial

---

1 Anotaciones

En este laboratorio, estudiaremos casos prácticos de lógica y la inteligencia artificial. Usaremos la lógica para representar el significado de oraciones en lenguaje natural y cómo se puede usar para resolver puzzle y demostrar teoremas. La mayor parte de esta tarea consistirá en traducir el inglés a fórmulas lógicas. En futuras evaluaciones usaremos otros lenguajes.

Para comenzar, iniciamos un shell de Python y escribimos los siguientes comandos para agregar expresiones lógicas a la base de conocimientos.

```
from logic import *
Rain = Atom('Rain')
Wet = Atom('Wet')
kb = createResolutionKB()     # Crea la base de conocimientos
kb.ask(Wet)
kb.ask(Not(Wet))
kb.tell(Implies(Rain, Wet))
kb.ask(Wet)
kb.tell(Rain)
kb.tell(Wet)
kb.ask(Wet)
kb.ask(Not(Wet))
kb.tell(Not(Wet))
```

1.1 Ejercicios

1. Imprime los resultados de cada uno de los resultados en un cuaderno jupyter llamado `CuadernoPrueba.ipynb`
2. Imprime el contenido de la base de conocimiento.
3. ¿ Que significan los símbolos `'*'` y `'-'` en los resultados obtenidos?.

Aquí hay una tabla que describe cómo se representan las fórmulas lógicas en el código.

Nombre	Notacion matematica	Codigo
Constante	uni	Constant('uni') (minusculas)
Variable	x	Variable('$x') (minusculas)
Fórmula Atómica (atom)	Rain	Atom('Rain') (predicado mayuscula)
	LocatedIn(uni,x)	Atom('LocatedIn', 'uni , '$x') (argumentos son simbolos)
Negación		Not(Atom('Rain'))
Conjunción		And(Atom('Rain'), Atom('Snow'))
Disyunción		Or(Atom('Rain'), Atom('Snow'))

Implicación	`Implies(Atom('Rain'), Atom('Wet'))`
Equivalencia	`Equiv(Atom('Rain'), Atom('Wet'))`
Cuantificador Existential	`Exists('$x', Atom('LocatedIn', 'uni', '$x'))`
Cuantificador Universal	`Forall('$x', Atom('MadeOfAtoms', '$x'))`

Las operaciones `AND` y `OR` solo toman dos argumentos. Si queremos tomar una conjunción o disyunción de más de dos, usamos `AndList` y `OrList`. Por ejemplo: `AndList([Atom ('A'), Atom ('B'), Atom ('C')])` es equivalente a `And(And(Atom('A'), Atom ('B')), Atom('C'))`.

1.2 Problemas

1. Escribe una fórmula de lógica proposicional para cada una de las siguientes oraciones en inglés en la función dada en `ejercicios.py`. Por ejemplo, si la oración es "If it is raining, it is wet,", entonces escribirías `Implies (Atom('Rain'), Atom ('Wet'))`, que sería $Rain \Rightarrow Wet$ en símbolos (revisa `ejemplos.py`). No te olvides de retornar la fórmula construida!.

   - `"If it's summer and we're in California, then it doesn't rain"`.
   - `"It's wet if and only if it is raining or the sprinklers are on"`.
   - `"Either it's day or night (but not both)"`.

   Puedes ejecutar el siguiente comando para probar cada fórmula:

   ```
   python prueba.py 1a
   ```

   Si tu fórmula es incorrecta, el evaluador proporcionará un contraejemplo, que es un modelo con el que su fórmula y la fórmula correcta no concuerdan. Por ejemplo, si accidentalmente escribiste `And(Atom('Rain'), Atom('Wet'))` para "If it is raining, it is wet", entonces el calificador generaría lo siguiente:

   ```
   Tu formula (And(Rain,Wet)) dice que este modelo es FALSE, pero debe ser TRUE:
   * Rain = False
   * Wet = True
   * (otros atomos si los hay) = False
   ```

   En este modelo que cumple la fórmula correcta $Rain \Rightarrow Wet(TRUE)$ pero no satisface la fórmula incorrecta $Rain \wedge Wet(FALSE)$. Utiliza estos contraejemplos como guía en el resto de los ejercicios.

2. Escribe una fórmula lógica de primer orden para cada una de las siguientes oraciones en inglés en la función dada en `ejercicios.py`. Por ejemplo, si la oración es "There is a light that shines,", entonces puedes escribir

   ```
   Exists('$x', And (Atom('Light', '$x'), Atom('Shines', '$x '))),
   ```

   que sería $\exists x Light(x) \wedge shines(x)$ en símbolos (ver `ejemplos.py`).

   Sugerencia : las tuplas de Python pueden abarcar varias líneas, lo que ayuda a mejorar la legibilidad cuando escribes expresiones lógicas (algunas de ellas en esta tarea pueden ser bastante grandes).

   - `"Every person has a mother."`.
   - `"At least one person has no children"`.
   - Crea una fórmula que define `Daughter(x,y)` en términos de `Female(x)` y `Child(x,y)`.
   - Crea una fórmula que define `Grandmother(x,y)` en términos de `Female(x)` y `Parent(x,y)`.

3. Alguien bloqueó el servidor y las acusaciones están volando. Para este problema, codificaremos la evidencia en fórmulas lógicas de primer orden para averiguar quién bloqueó el servidor. Se ha reducido a cuatro sospechosos: John, Susan, Mark y Nicole.

   Tu tienes la siguiente información:

   ```
   John dice: "It wasn't me!"
   Susan dice:"It was Nicole!"
   Mark dice: "No, it was Susan!"
   Nicole dice:"Susan's a liar."
   Sabes que exactamente una persona dice la verdad.
   También sabe exactamente que una persona bloqueó el servidor.
   ```

   Completa la función `liar()` para devolver una lista de 6 fórmulas, una para cada uno de los hechos anteriores. Debes asegurarte de que tus fórmulas estén exactamente en el orden especificado.

   Puedes probar tu código usando los siguientes comandos:

   ```
   python prueba.py 3a-0
   python prueba.py 3a-1
   python prueba.py 3a-2
   python prueba.py 3a-3
   python prueba.py 3a-4
   python prueba.py 3a-5
   python prueba.py 3a-all  # Prueba la conjuncion de todas las formulas
   ```

   Para resolver el puzzle y encontrar la respuesta, `tell` las fórmulas a la base de conocimientos y `ask` la consulta

   ```
   CrashedServer('$x')
   ```

   ejecutando:

   ```
   python python.py 3a-run
   ```

4. En este problema, veremos cómo usar la lógica para probar automáticamente teoremas matemáticos. Nos centraremos en la codificación de un teorema y dejaremos la parte de prueba al algoritmo de inferencia lógica. Aquí está el teorema:

   Si se cumplen las siguientes restricciones:

   - Cada número x tiene exactamente un sucesor, que no es igual a x.
   - Cada número es par o impar, pero no ambos.
   - El sucesor de un número par es impar.
   - El sucesor de un número impar es par.
   - Para cada número x el sucesor de x es mayor que x.
   - Propiedad transitiva: si x es mayor que y e y es mayor que z, entonces x es mayor que z.

   Entonces tenemos la siguiente consecuencia: `para cada número, hay un número par mayor que él`.

   Nota: en este problema, `mayor que` es simplemente una relación arbitraria y no debes asumir que tiene un significado previo. En otras palabras, no asuma cosas como `un número no puede ser mayor que él mismo` a menos que se indique explícitamente.

(a) Completa `ints()` para construir 6 fórmulas para cada una de las restricciones. La consecuencia se ha completado para ti (consulta el código). Puedes probar tu código usando los siguientes comandos:

```
python prueba.py 5a-0
python prueba.py 5a-1
python prueba.py 5a-2
python prueba.py 5a-3
python prueba.py 5a-4
python prueba.py 5a-5
python prueba.py 5a-all  # Prueba la conjuncion de todas las formulas
```

Para probar el teorema, `tell` las fórmulas a la base de conocimientos y `ask` la consulta ejecutando la verificación del modelo (en un modelo finito):

```
python python.py 5a-run
```

(b) Supongamos que agregamos otra restricción: `un número no es mayor que él mismo`. Demuestra que no existe un modelo finito no vacío para el que el conjunto resultante de 7 restricciones sea consistente.