Artificial Intelligence Laboratory 3: First Order Logic

DT8012 (HT16) Halmstad University
28th of November 2016

This lab is designed to introduce you to the First Order Logic (FOL). By the end of this lab session you will:

- Learn how to use FOL to create knowledge base and use it for reasoning
- Learn how to use FOL instead of if/then statements to create multi-step inferences

For this lab, you will use FOL in two different domains:

- **Smart Home:** to find out the general rules to explain typical behavior of a resident in a simulated smart home environment
- **Poker player:** to evaluate your hand and improve your strategy against your opponents

Before you arrive at the lab:

You should read through the entire document and look up anything you don't quite remember from the lectures.

After the lab:

Within two weeks after this lab session you must hand in your **report** and **source code**. Your report should explain your results, and **how** you achieved them. Make sure you include any relevant information to your explanation. Your code, developed in order to solve the exercises proposed here, should be well commented and self-explanatory.

The deadline for your report and code submission is **December 13**. Send your report as a pdf document and your code as one zip file to hassan.nemati@hh.se. The name of your files should include your first and last names, e.g. HassanNematiReport.pdf, HassanNematiCode.zip. Write the name of the lab in the title of your email e.g. Lab 3. Do not forget to write your name and your group mate full name.

Introduction

Environment setup

The only required software tool is Python. You need to install 'numpy' and 'pyDatalog' libraries in python. For more information about pyDatalog read: https://sites.google.com/site/pydatalog/Online-datalog-tutorial

Note that the 'pyDatalog' library is not FOL, but you can use it to create variables and rules. For example, terms can be defined like "indianFood('Curry')", "mildFood('Tandoori')", and sentence like likes('Sam', X) <= (indianFood(X) & mildFood(X)), where "X" is a variable. This example can be represented by FOL as:

$$(\text{indianFood}(X) \wedge \text{mildFood}(X)) \Rightarrow \text{likes}('\text{Sam}', X)$$

Some of the syntaxes than can be used in this library are listed:

- To declare the variables they must start with an upper-case letter
- Logic function names starts with a lower case
- You can assert logic clauses of the form head <= body, where head is a single literal with one or more variables, and body is a list of literals separated by '&' : p(X) <= q(X) & R(X,Y)
- Each variable in the head must also appear in the body. The body may include equality and comparison predicates: N1 == N-1, or N > 0, and contain lambda expressions
- You can negate a literal in the body: ~(p(X))
- The argument of a literal cannot be another predicate : p(q(a)) is not allowed
- You can define logic functions by a formula: p[X]=expression (similar to an assignment)
- You can define aggregate functions, e.g. p[X]=len(Y) <= body : len, sum, concat, min, max
- You can use prefixed literals and functions (e.g. Employee.name[X]==Y) to refer to python objects in logic clauses, or to run logic queries in python programs

Take a look at **Logic_Example.py** file in the **Lab3Code.zip** file to figure out the library usage.

Task 1: Smart Home

In this task you will use FOL to explain typical behavior of a resident in a simulated Smart Home (Fig. 1). In this Smart Home, 11 different sensors are mounted in different areas. Some of these sensors are PIR (motion detector) which produce a signal whenever they detect that a person is in a room. Other sensors (e.g. door or pressure sensors) can produce two different signals, e.g. for the bed sensor, one signal is for when a person is lying down on the bed and a separate signal is for when he or she rises from the bed.

In the Lab3.zip you will find the "SmartHomeData.npy" file, which contains simulated data for 1010 days. The data file is a 3-dimensional matrix: 60 (number of time steps, one minute each) ^{x} 18 (number of signals from sensors) ^{x} 1010 (number of days). Thus, the columns are the observations (60, one per minute), the rows are the 18 signals as described below, and the aisle are the days of experiment (1010 aisles). Timestamps start at 00:00, so the first column is time 00:00, the second one is 00:01, etc.

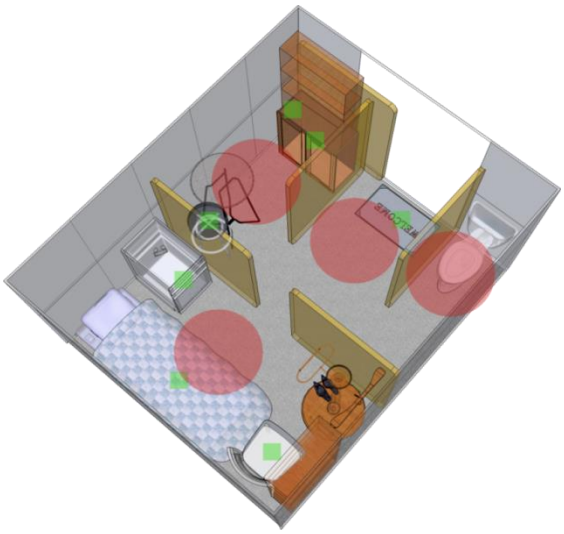

Fig. 1- Smart Home environment, red circles are the PIR sensors (4 sensors), and the green boxes are the door or pressure sensors (7 sensors)

Rows 0 & 1: Pressure_sensor_S1, Entrance door [In to Home row 0 and Out of Home row 1]
Row 2 & 3: Pressure _sensor_S2, Bedroom [In to Bed row 2 and Out of Bed row 3]
Row 4 & 5: Pressure _sensor_S3, Chair in bedroom
Row 6 & 7: Pressure _sensor_S4, Chair in kitchen

Row 8: PIR_sensor_1, Hallway [Enter Hallway row 8]
Row 9: PIR_sensor_2 , Kitchen [Enter Kitchen row 9]
Row 10: PIR_sensor_3, Bathroom [Enter Bathroom row 10]
Row 11: PIR_sensor_4 , Bedroom [Enter Bedroom row 11]

Row 12 & 13: DOOR_sensor_D1, Bedroom drawer [Open Drawer row 12, close Drawer row 13]
Row 14 & 15: DOOR_sensor_D2 , Water kettle
Row 16 & 17: DOOR_sensor_D3, Kitchen cabinet

For example, the result of printing a part of the sensors readouts for one hour during a night is shown in Table. 1. According to this table, the person was at the kitchen at time 00:56, then he was at the bedroom at time 00:57, and finally went to bed at time 00:58.

	Time 00:00	Time 00:01	Time 00:02	Time 00:03	…	Time 00:56	Time 00:57	Time 00:58	Time 00:59
Sensor S1	0	0	0	0	…	0	0	0	0
Sensor S1	0	0	0	0	…	0	0	0	0
Sensor S2	0	0	0	0	…	0	0	1	0
Sensor S2	0	0	0	0	…	0	0	0	0
Sensor S3	0	0	0	0	…	0	0	0	0
Sensor S3	0	0	0	0	…	0	0	0	0
Sensor S4	0	0	0	0	…	0	0	0	0
Sensor S4	0	0	0	0	…	0	0	0	0
Sensor S5	0	0	0	0	…	0	0	0	0
Sensor S6	0	0	0	0	…	1	0	0	0
Sensor S7	0	0	0	0	…	0	0	0	0
Sensor S8	0	0	0	0	…	0	1	0	0

Sensor S9	0	0	0	0	…	0	0	0	0
Sensor S9	0	0	0	0	…	0	0	0	0
Sensor S10	0	0	0	0	…	0	0	0	0
Sensor S10	0	0	0	0	…	0	0	0	0
Sensor S11	0	0	0	0	…	0	0	0	0
Sensor S11	0	0	0	0	…	0	0	0	0

Table. 1- Example sensor readouts for one night

Task 1

a. Your task is to create rules, based on the available data, to represent typical daily routine (behavior with high frequency) in the Smart Home environment. You need to create at least 4 rules with the corresponding FOL.

For example one of the common daily routine is: getting up from the bed in bedroom "**inBedroom**", then going to the kitchen "**EnterKitchen**", and then going back to the bedroom "**inBedroom**". One way to create this rule is by using the following sentences:

Bed_to_Kitchen(Y, X) <= (inBedroom(Y, X) & inKitchen(Y, Z) & nextNr(X,Z))

Kitchen_to_Bed(Y, X) <= (inKitchen(Y, X) & inBedroom(Y, Z) & nextNr(X,Z))

Bed_Kitchen_Bed(Y) <= (Bed_to_Kitchen(Y, X) & Kitchen_to_Bed(Y, Z) &

~EnterBathroom(Y) & ~ OutofHome(Y))

where "inBedroom(Y, X)" corresponds to: the person is in the bedroom at time "X" day "Y", and "nextNr(X, Z)" specifies that "Z" is on second after "X". You also need to specify the person did not do any other activity during these days by checking the negation of predicates such as: **~EnterBathroom(Y) & ~ OutofHome(Y).**

Note that these sentences are not available in 'pyDatalog' and you have to define them, see "Example 2" in Logic_Example.py.

The rule "**Bed_Kitchen_Bed**" is considered as a typical daily routine since its frequency of occurrence is 156 days out of 1010 days.

>>> print(len(**Bed_Kitchen_Bed(Y)**))

156

b. Identify the exceptions in the data set. There are 10 scenarios which are significantly different from the rest, e.g., there may be a day when the person sleeps all the day in his bed and does not go out from the bedroom at all. Other examples:
   - Went out and did not come back
   - Went to bathroom and did not come back
   - The first activity in that day is enter to kitchen but the bedroom sensor did not activate

You need to find at least 6 of such examples with the corresponding day number and represent them by quantifiers "For all", "There exists".

∃ (Y) (DayNumber(Y) & OutofHome (Y) & ~IntoHome (Y))

c. (extra credit) In the Lab3.zip you will find the Sampledata.csv, which contains real data from another smart home. This data is collected during 4 days and each line contains a single sensor event in the format: date, time, area, sensor, and status of the sensor (Fig. 2).

- The date refers to the specific day on which the sensor event occurred. The date is in the format yyyy-mm-dd
- The time refers to the time of day at which the sensor event occurred. The time is in the format hh:mm:ss.x
- The area is an abstract representation of the sensor, often a room name combined with sensor type
- The Sensor is a more specific functional description of the sensor, perhaps a specific object or area in a room of the building
- The sensor which generated the current event has a value for this event. As an example, if the person is sleeping on the bed, the Bed sensor in the Bedroom records the status ON.

Date	Time	Area	Sensor	Status
2011-06-15	00:06:32.834414	Bedroom	Bed	ON
2011-06-15	00:06:33.988964	Bedroom	Bed	OFF
2011-06-15	03:37:46.585185	Bedroom	Bed	ON
2011-06-15	03:37:47.706265	Bedroom	Bed	OFF

Fig. 2 - Example sensor readouts

For this task you need to find at least 5 activity rules using FOL similar to task 1-a.

<activity1> ⟶ <activity2>

Then, you need to calculate the mean and standard deviation of the time stamps between these two activities. And finally print the results in the following format:

<activity1> ⟶ <activity2> <mean> <standard_deviation> <number of occurrence>

Where <mean> and <standard_deviation> are the mean and standard deviation of the time between these two activities in all the data set the rule is valid for them. Consider the following example as the required output for this task:

Bedroom ⟶ Bathroom 1000(sec) 200(sec) 10

This indicates the rule toEnterBath(Bedroom ∧ bathroom) occurs 10 times where the person within 1000 ± 200 seconds after getting up from the bed goes to the bathroom.
Therefore, you need to define terms e.g. "Bedroom", rules e.g. "toEnterBath(Bedroom & bathroom)", the frequency of seeing this rule, and the time difference between each activity. You can use "*datetime*" library in python to get the time difference between each activity.

Task 2: Poker

Playing your opponent is arguably more important than playing your cards in poker. You have to be able to read what your opponent is doing: in particular, to estimate how strong their hand is.

In Lab2 you have been given a strategy for an opponent, in the procedural form (as a Python code consisting of many if-then statement).

a. Rewrite part of the given strategy in Lab2 (agent strategy) using First Order Logic (use provided example in "Example 3", Logic_Example.py as a starting point).

   For example you can define the following predicates in the knowledge base to rank your hand:

   + weakHand('OnePair')
   + mediumHand('TwoPairs')
   + strongHand('ThreeOfKind')
   + strongHand('FullHouse')
   + veryweak…
   + verystrong…

   And then reasoning your action by evaluating your hand and the stack amount:

   agentAction('Raise',X) <= (agentHand(X) & mediumHand(X) & (playerStack[Stack] == 'Large'))

b. Use the resulting FOL rules to reason in the opposite direction (one of the benefits of declarative knowledge is that this is easy to do): given the actions made by the opponent, deduce the possible contents of their hand.

   For example:
   agentAction('Fold',X) <= (OpponentAction('Raise') & (OpponentStack[Stack] == 'Small') & (agentHand(X) & mediumHand(X) & (playerStack[Stack] == 'Large'))

   You need to define at least 5 rules for this task.

Grading criteria

- Pass: Complete all the tasks except 1-c.
- Extra credit: Complete the task 1-c (within the 2 weeks deadline).
- Submission deadline December 13.