Introduction à l'IA / Intelligent Systems

Symbolic AI – Logic Reasoning

Sylvain Bouveret (sylvain.bouveret@grenoble-inp.fr)
April 2024

Introduction

This document contains several practical exercises on logical reasoning. We will use *pyDatalog*[1] for this session. Only Sections 2 and 3 will be graded. Section 1 only aims at making you discover the library and how datalog rules can be written and used to infer new facts. All sections are independent on each other, but I strongly advise you to start with Section 1 unless you already perfectly know the language.

This practical work has to be made by teams of 2 or 3 students.
Only Sections 2 and 3 will be graded.
Your work has to be submitted at the end of the session. No report is needed. Just send your Python code.

1 Discovering Datalog

1.1 Installing the environment

The *pyDatalog* library is not installed by default, but you can simply install it using *pip*. We advise you to use a Python virtual environment to keep the installation local.

```
$ python3 -m venv venv
[...]
$ . venv/bin/activate
[...]
$ pip3 install pyDatalog
[...]
$ # Run your programs here, and when you are done:
$ deactivate
```

1.2 Learning pyDatalog

pyDatalog aims at embedding logic programming in Python. It is based on *Datalog*, which is a declarative logic programming language (like Prolog), made of a subset of first order logic. Like a deductive knowledge base, a Datalog program is made of:

- *facts*, that are statements that are known to be true, like:

$$mother(john, mary)$$

- *rules*, in the form of first order definite clauses:

$$\begin{aligned} parent(X,Y) &\leftarrow mother(X,Y) \\ parent(X,Y) &\leftarrow father(X,Y) \\ grandParent(X,Y) &\leftarrow parent(X,Y) \wedge parent(Y,Z) \end{aligned}$$

[1] https://sites.google.com/site/pydatalog

Datalog is closely related to relational algebra for databases, and is often used as a query language for deductive databases. It has a somewhat limited expressivity, but it is still powerful enough to be of interest in our context.

pyDatalog mixes elements of Datalog logic programming with Python programming, which means that pyDatalog programs have both a declarative and an imperative flavour.

Question 1 After this short introduction, it is time for you to experiment. To learn how *pyDatalog* works, follow the online tutorial at:

https://sites.google.com/site/pydatalog/Online-datalog-tutorial.

Question 2 Now that you have followed the tutorial, try to implement the small example described above, with parents, mothers, fathers, and grand-parents. Experiment it with facts from your own family or a fictive one. Do not spend to much time on it now; we will extend this example in Section 2.

1.3 A more comprehensive example

In this exercise, we will learn to use *pyDatalog* on a more comprehensive example, based on rules that could be used by a bank to accept or refuse a loan.

Question 3 Download from Caseine the file `test-pyDatalog.py` and observe its output.

Question 4 Extend the file by encoding the other rules, described in file `datalogRules.pdf`, available on Caseine as well.

Once you are done, the output of your program should be something like:

```
X  | Y
---|-----
p2 | no
p1 | yes
```

In other words, the decision for `p1` is `no`, while it is `yes` for `p2`. Why that? A great advantage of symbolic reasoning like logic programming is that decisions can be explained. For that, we only have to annotate each rule with its name, by adding an argument to each rule. For instance, rule 1:

```
ageRange(X, 'young') <= age(X, Y) & (Y < 30)
```

could become:

```
ageRange(X, 'young', 'R1') <= age(X, Y) & (Y < 30)
```

Question 5 Revisit your rules by annotating each of them with a name (R1, R2, etc.) that should help the user in having an explanation for the final output. The explanation names should be propagated for the rules that call other rules.

Once you are done, the output of your program should be something like:

```
X  | Y   | E1
---|-----|----------------------------------------
p2 | no  | ('R44', ('R19', '~', 'R33'))
p1 | yes | ('R37', (('R22', ('R16', 'R5')), 'R31'))
```

which can be read as: `p2` was inferred as "no" because of R44, which called R19, some rule that failed, and R33; moreover, `p1` was inferred as "yes" because of R37, which called R22 (which called R16 and R5) and R31.

2 Winter is coming...

Remember *Game of Thrones*?[2] No? It does not matter. Your grade will not depend on your knowledge of the characters. When I used to watch this series, I sometimes had trouble understanding all the complicated family relations between the characters and the different houses. And if you do not understand these family relations, you miss a big part of the story. Can you help me getting my bearings about this?

To help you, I have found a whole database of characters of *Game of Thrones*, with their family relations and houses. You can download it on Caseine (the file is called `got.csv`).[3]

In this section, we will use *pyDatalog* in an object-oriented programming way, which means the syntax will slightly differ from what we saw before. Do not worry, there is an entire example you can adapt to suit your needs.

2.1 Data is coming

Question 6 Adapt the example from:
`https://sites.google.com/site/pydatalog/Datalog-in-python`
to create a class `Character` with four attributes: `name`, `mother`, `father` and `house`.

Now that you have a class representing the characters, it is time to load the data in the CSV file.

Question 7 Write a Python program to read the CSV file and load the data on it: for each row of the CSV file, the program should create an instance of `Character` with the appropriate fields.

Be careful, the two fields `mother` and `father` should be existing instances of `Character` (not strings), or `None` if the string is empty in the CSV file.

2.2 Queries are coming

Now that we have loaded our data as Python objects (and have corresponding *pyDatalog* predicates thanks to `pyDatalog.Mixin`), it is time to write some rules and to query the database.

Question 8 Adapt the predicates `parent` and `grandParent` you have written in Question 2 to work with your instances of `Character`.

Who are Arya Stark's parents and grand-parents?[4]

Question 9 Add a predicate `sibling` and a predicate `cousin` to your Datalog program.

Who are Arya Stark's cousins and siblings?[5]

Question 10 Now add a predicate `ancestor` to your Datalog program.

Who are Jon Snow's ancestors?[6] Who are Jon Snow's ancestors *that do not belong to the same house as him*?

Question 11 In *Game of Thrones*, strange family things can happen. Are there any pairs of characters that are both siblings and cousins?

[2] `https://en.wikipedia.org/wiki/Game_of_Thrones`

[3] This file has been adapted by myself from a file provided by Shirin Elsinghorst (`https://shiring.github.io/networks/2017/05/15/got_final`), itself adapted from an initial Kaggle dataset (`https://www.kaggle.com/mylesoneill/game-of-thrones`).

[4] You are not supposed to answer to this question directly, but to give the program that gives the answer.

[5] Same remark as Footnote 4 basically.

[6] The answer might surprise you.

3 Ticket to ride

Congrats, you have obtained an internship in a research lab, and you have a research paper that you want to present in a conference. Your lab, which has a bold carbon emission reduction policy, prevents researchers from traveling by plane if the same journey can be made in less than 6 hours by train.[7] You want to figure out where you can go by train in France, and how much time the journey will take.

3.1 A train dataset

I have a dataset for you: download the file `trains.csv` on Caseine. This file contains a set of journeys between pairs of cities in France, with travel times expressed in minutes.[8]

Question 12 Adapt the example from:
https://sites.google.com/site/pydatalog/Datalog-in-python
to create a class `Journey` with three attributes: `departure`, `arrival`, and `time`.

Question 13 Write a Python program to read the CSV file and load the data on it: for each row of the CSV file, the program should create an instance of `Journey` with the appropriate fields.

3.2 Where should I go now?

Now it is time to write the rules and query the dataset...

Question 14 First create a predicate `link(X, Y, T)` that models the fact that there is a direct journey between `X` and `Y`, with travel time `T`.

NB: We assume in this exercise that this predicate is symmetric in `X` and `Y`, meaning that if `link(X, Y, T)` is true, then `link(Y, X, T)` should also be.

Test your predicate by listing all the possible direct journeys from (and to) Grenoble.

Question 15 Create a predicate `connected(X, Y, T)` that models the fact that `X` and `Y` are connected (directly or indirectly) and that it takes time `T` to travel between these two cities.

Question 16 Enhance your predicate `connected(X, Y, T)` to create a predicate `path(X, Y, P, T)` which models that `X` and `Y` are connected (directly or indirectly) via the cities `P` (`T` is still the total travel time like before). Here, `P` is a `list`.

Question 17 The previous predicate lists all the paths between two cities. Write a new function `shortest_path_value[X, Y]` that maps the pair of cities `X`, `Y` to the shortest travel time between them.[9]

Then write a predicate `shortest_path(X, Y, P, T)` which is the analogous of `path(X, Y, P, T)`, but only outputs the shortest path between `X`, `Y`.

Question 18 Finally, write a predicate `transport(X, Y, Tr)` which indicates which means of transportation we should use to travel from `X` to `Y`. `Tr` is `'train'` if the (shortest) travel time is less than 6 hours, and `'plane'` otherwise.

[7] You are lucky, because your lab goes beyond the legal constraint, which only applies to journeys of less than 4 hours.

[8] Adapted from https://ressources.data.sncf.com/explore/dataset/meilleurs-temps-des-parcours-des-trains. It seems largely incomplete, but it does not matter much for this exercise.

[9] Catch a glimpse at the tutorial if you do not remember how to use aggregate functions.