

MBAN6500_assignment1

January 18, 2021

1 Build an Expert System (ES) using experta

1.1 Submission

The assignment should be submitted as Python3 code and uploaded to Canvas. It is due by on February 1, 2021, at 8:30 am. The code will be tested and should produce results relatively well matching with reality.

1.2 Background

[Experta](#) is a Python alternative to [CLIPS](#). CLIPS is a tool/language for building ES still in use today. However, for you not to have to learn a new language for a single assignment, we stick with Python instead. Most of what you learn with experta should be transferrable to CLIPS.

1.3 Assignment

1. Implement an ES that tells whether an animal is a bird, mammal, or unknown. It should require two percepts/declarations: `cover` and `wings`. The former can take one of two values, "fur" or "feathers", and the latter `True` or `False`. Given the declarations `cover = "feathers"` and `wings = True`, it should print "bird". Similarly, given `cover = "fur"` and `wings = True`, it should print "mammal", since a bat has wing and is a mammal. For `cover = "feathers"` and `wings = False` it should print "unknown" (since I don't know of any wingless bird).
2. Implement an `AnimalIdentifier`. Given some suitable number of declarations, this ES should be able to classify an animal into the following categories: "protozoa" (a single cell animal), "invertebrate", "fish", "bird", "mammal" or "unknown".
3. Implement an ES that can answer questions about a topic of your own interest. Include an explanation and description of its intended behavior as a comment.

1.4 Getting started

Please start by checking [The Basics](#) in the GitHub documentation under docs/source/thebasics.rst, and especially **Facts** and **Rules**. After that, take your time to look through the experta [examples](#).

You can see what is available in experta by `dir(experta)`. This will list all attributes of experta, i.e. the stuff you access by `experta.<attribute>`.

To learn more about a particular attribute use `experta.<attribute>?` E.g.:

experta.KnowledgeEngine?

1.4.1 Python classes and decorators

If you have no previous experience with Python classes then the [Python documentation](#) may be helpful. Beyond that, there are plenty of tutorials on the web.

To form rules (i.e. sentences in FOL), experta uses decorators (i.e. the @-syntax). See [here](#) for a quick intro, or [here](#) for a more in-depth explanation.

1.4.2 The first example

Let's look at the first example in the [experta documentation](#).

```
from random import choice
from experta import Fact, KnowledgeEngine

class Light(Fact):
    """Info about the traffic light."""
    pass

class RobotCrossStreet(KnowledgeEngine):
    @Rule(Light(color='green'))
    def green_light(self):
        print("Walk")

    @Rule(Light(color='red'))
    def red_light(self):
        print("Don't walk")

    @Rule(AS.light << Light(color=L('yellow') | L('blinking-yellow'))))
    def cautious(self, light):
        print("Be cautious because light is", light["color"])
```

In the class RobotCrossStreet, the rules (@Rule) are essentially FOL sentences. For example,

```
@Rule(Light(color='green'))
def green_light(self):
    print("Walk")
```

can be read as follows:

$Light(green) \rightarrow Walk$

or “if the light is green, then walk” (or rather, print the string 'Walk').

The third rule does two things:

```
@Rule(AS.light << Light(color=L('yellow') | L('blinking-yellow'))))
def cautious(self, light):
    print("Be cautious because light is", light["color"])
```

First it says “if the light is yellow or blinking-yellow, then be cautious”

or,

$Light(yellow) \vee Light(blinking - yellow) \rightarrow BeCautious.$

Second, the `AS.light << ...` binds the value of `color` (i.e. ‘yellow’ or ‘blinkingyellow’) to `light` so that the value of `color` can be accessed by `light["color"]`.

To use `RobotCrossStreet` we run the following:

```
engine = RobotCrossStreet()    # instantiates the class
engine.reset()                 # resets to default values (if any)

# randomly pick a color from the list
light_color = choice(['green', 'yellow', 'blinking-yellow', 'red'])

# give this light color as input to the engine (the instantiation of RobotCrossStreet).
# This would be equivalent to TELL(KB, MAKE-PERCEPT-SENTENCE(percept)) in Fig 7.1 (AIMA or cla
engine.declare(Light(color=light_color))
engine.run() # This is equivalent to ASK(KB, ...).
```

Good luck!

[]: