Exercise Normative Programming

B. Testerink, M. Dastani

April 10, 2012

1 Introduction

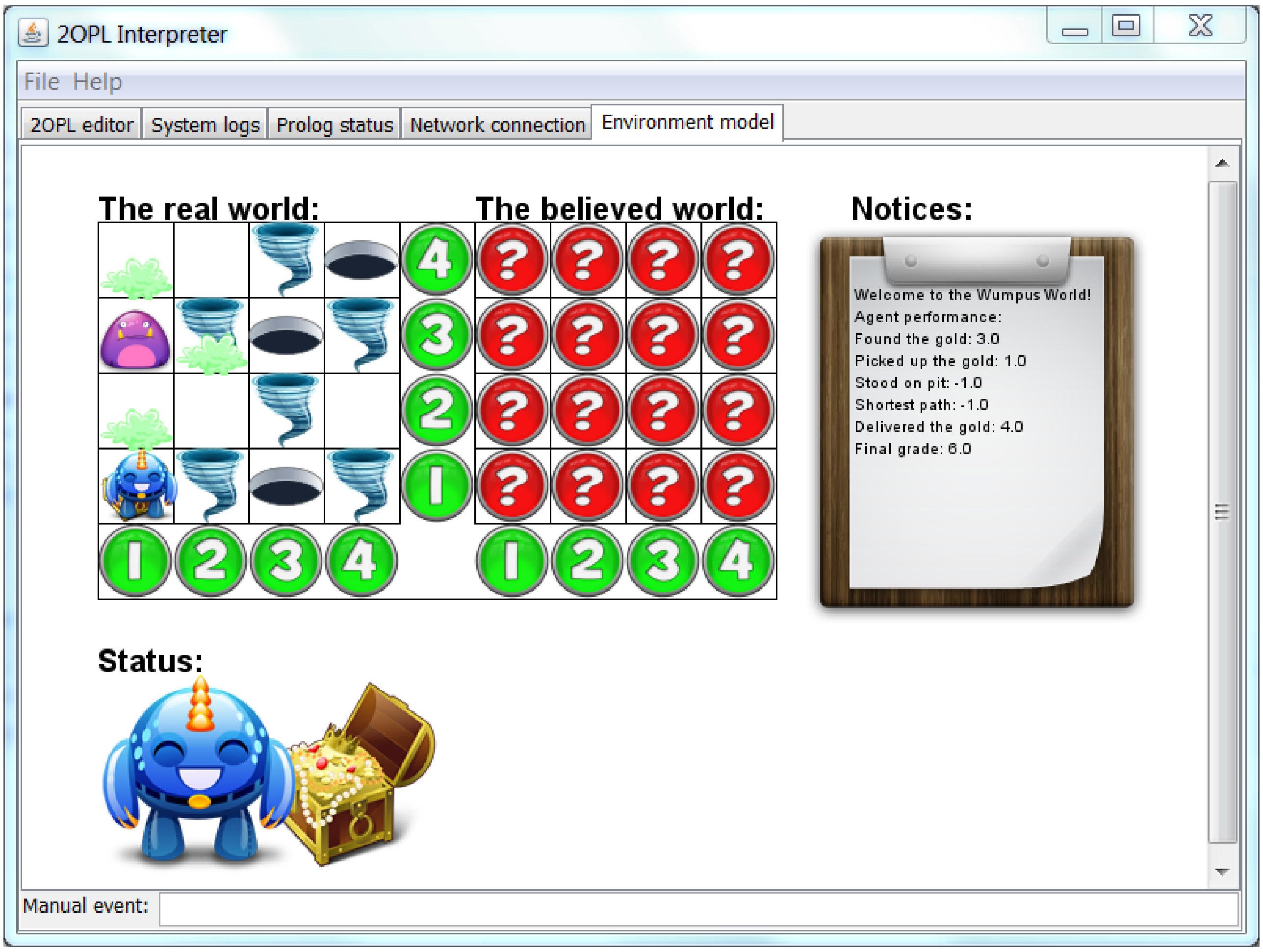
In this exercise we will explore normative systems. The main part of the exercise consists of 2OPL and the temporal norm extension. As a scenario we take the Wumpus world which is depicted in figure 1. Sometimes the Wumpus world is used as a testbed for agents (c.f. *Artificial Intelligence: A Modern Approach* by Russel & Norvig). We are not interested in the A.I. for the agent (the blue thingy) now, so we can use our mouse to control it. What we want is to monitor, punish and reward the behavior of the agent.

2 Actions and events

The organization allows three actions which are depicted below. The effect of these actions can be described by effect rules (i.e.: `{pre} move(Dx,Dy) {post}`), which bear a great resemblance to the belief update rules from 2APL.

Name:	Moving
Description:	Moves the agent one square. The arguments are Δ_x and Δ_y, for which holds that one of the two is zero and the other is either 1 or -1. For instance if $\Delta_x = 0$ and $\Delta_y = 1$, then the agent moves one square to the north.
Manual:	Click on a square that is adjacent to the agent.
Performed action:	`move(DeltaX,DeltaY)`

Name:	Picking up
Description:	Try to grab gold.
Manual:	Click on the agent while it holds nothing.
Performed action:	`gripper(pickup)`



Welcome to the Wumpus World!
Agent performance:
Found the gold: 3.0
Picked up the gold: 1.0
Stood on pit: -1.0
Shortest path: -1.0
Delivered the gold: 4.0
Final grade: 6.0

Figure 1: Screenshot of the wumpus environment

Name:	Dropping
Description:	Drop the gold.
Manual:	Click on the agent while it holds the gold.
Performed action:	`gripper(drop)`

You can also let the organization perform actions. The first action is that you can let the organization bombard the wumpus. An example sanction rules that uses this: `p => do_bombard`. Another action is updating the notices. You can use `show(Topic,Grade)` for this. Notices are ordered by topics. Using this call you can override a topic. Examples are provided in the norm files.

3 The exercise

Our organization will implement an automatized grading system for the agent. To prevent ambiguity we refer to the agent as 'he' instead of 'it'. We want the following behavior:

- If the agent finds the gold, then he is rewarded 3 points (we assume that the agent cannot loose the gold such that he can find it more than once).
- If the agent immediately picks up the gold after he finds it, then the agent is rewarded another point.
- If the agent neglected to immediately pick up the gold after he found it, then we subtract 4 points.
- If the agent returns to the chest and drops the gold, then he is rewarded 4 points. You may assume that the chest is always located at (1,1).
- If the agent delivered the gold within three steps (three move actions) after he found it, then he is rewarded another 2 points
- If the agent did not deliver the gold within three steps, then we subtract a point.
- If the agent stands on a living wumpus, then we murder the wumpus by bombarding it and subtract 4 points.
- Each time the agent stands on a pit we subtract a point.
- Only the above situations change the grade of the agent (for instance standing on a dead wumpus does not have any effect).
- When the agent is finished (the gold is dropped at (1,1)), then we display his total grade.

You have to implement this by programming one organization with 2OPL in its original form ("2OPL Original.2opl"), and one organization with the temporal extension (temporal norms instead of counts-as. "2OPL with Temporal.2opl"). You only need to hand in the .2opl files. Just like the other exercises we will also look at the code style (comments, and whether it is concise). Remember that 2OPL uses 'and' for conjunction. Temporal norms are notated with brackets as the tuple beginning/end in order to avoid ambiguity with less than and greater than operators. So you get for instance: `lbl(Y):[p(X) and q(Y),O(r and not s), X > Y]`. This norm can generate either `viol(lbl(Y))` or `obey(lbl(Y))`. To determine the consequences of this example norm you get for instance: `obey(lbl(Y)) => not obey(lbl(Y)) and t.` in the sanctions section.

4 GUI

To start the program, use `java -jar "2OPL Interpreter.jar"`. You can use the GUI that is provided to edit your norm file. But perhaps it is easier to use your favorite text editor, as the GUI does not yet provide functionalities like ctrl+z and parenthesis highlighting. The only kind of possible debugging is logging the Prolog states of the organization and querying/viewing it on runtime. You can enable logging in the logging tab, and the Prolog state can be kept up to date as well (btw, clicking the 'keep updated' button in the Prolog tab refreshes its textarea). In the environment tab it is possible to manually send an event to the organization. This can be used to play around with the language.

5 Concerning bugs and issues

Should you encounter a bug, something which is quite common in prototypes, then notify this ASAP (B.J.G.Testerink at uu.nl). Do not forget to include your norm file, and the exact sequence of events that you performed when the bug occurred. We have our own solution to this exercise so it is possible to implement it. However, if a bug slows you down considerably then you are entitled to some more in-depth help to get you going. Any issues other than critical bugs can be reported in your read me file. We will appreciate feedback on the practical side of normative programming (such as suggestions for functionality or syntax).