Automatic Reasoning and Learning - Homework Assignment 1: Knowledge Based Agents with CPo Logic

Ramón Béjar

April 8, 2021

Goal

The goal of this work is to develop an intelligent agent for the Envelope World problem. In this world, we have a grid of $n \times n$ cells. Some cells contain an envelope (with some money inside), and we know that there is at least one envelope in the world. We have an *envelope finder agent*, that uses a sensor to discover the position of the envelopes. If the agent is located at position (x, y), it can receive any of these readings from the sensor (it can even receive **many** of these readings depending on the number and positions of envelopes in the 3x3 square around its current position):

1. If there are some envelopes at positions $\{(x+1, y-1), (x+1, y), (x+1, y+1)\}$, it will receive the reading: 1.
2. If there are some envelopes at positions $\{(x+1, y+1), (x, y+1), (x-1, y+1)\}$, it will receive the reading: 2.
3. If there are some envelopes at positions $\{(x-1, y-1), (x-1, y), (x-1, y+1)\}$, it will receive the reading: 3.
4. If there are some envelopes at positions $\{(x+1, y-1), (x, y-1), (x-1, y-1)\}$, it will receive the reading: 4.
5. If there is an envelope at position (x, y), it will receive the reading: 5.

If there are no envelopes in any of these positions, the sensor will give no readings. Observe that the presence of envelopes in the *corner* positions of the 3x3 square, will produce the sensor to give two different readings. For example, an envelope at $(x+1, y+1)$ will give the sensor readings: 1 and 2. We assume the sensor works perfectly at every position where the finder agent moves to.

You have to develop your program using the java classes I have provided with this assignment, where some functions must be implemented, the finished ones can be modified, and new ones added. BEWARE: Everything that I ask in this document that appears in red is something that is mandatory to satisfy in order to have your project evaluated. So, any not satisfied red point will make the grade of this project equal to 0. So, before delivering your project, check that you satisfy all these minimal points. Check it with me, before delivering it, if you want to be sure. You can work in groups of two or three.

Your application works with two main objects: the envelope finder agent and the world environment. The world environment is simply an interface that is used to model what the agent performs in the world and the information it receives from the world with its sensor. The application has to work with the following input:

1. The dimension of the world (the value of n for the $n \times n$ Envelope World) (this information is needed by both the finder agent and the world environment object).
2. This information is used only for the finder agent:
   (a) Number of steps (l) to perform.

(b) A sequence of l steps of this form:

$$x_1, y_1\ x_2, y_2\ \ldots\ x_l, y_l$$

where x_t, y_t indicates that at time step t the agent moves to position (x, y). This sequence of steps will be stored in an text file, in a single line.

3. This information is used only for the world environment object:

(a) The list of envelope locations, following the same format that the sequence of steps: in a file with a single line with the positions of m envelopes (where always $m \geq 1$) as:

$$ex_1, ey_1\ ex_2, ey_2\ \ldots\ ex_m, ey_m$$

With that input, your finder agent should print at the standard output (the screen) the knowledge state for possible locations of envelopes that the agent has **after it processes each step of the agent**. This knowledge state will be presented as the $n \times n$ matrix with the ? and X symbols, where ? indicates a possible location and X a **not possible location**. For using this representation of knowledge states, your agent will use the class `EFState`, that is already implemented, but it can be modified if you need to.

Remark: Notice that when the sensor returns the value 5, it means there is an envelope at current position (x, y). In that case, the agent could infer that (x, y) is not only a possible position, but a secure one. However, to simplify the problem and do not make the agent to perform more than one kind of inference questions (questions of the kind $\models e^{t+1}_{x',y'}$ in addition to the kind $\models \neg e^{t+1}_{x',y'}$), we will consider that position simply as a possible one for envelopes. So, for example, imagine that at position (x, y) the only value readed from the sensor is 5. Then, the agent should infer, when doing questions of the kind $\models \neg e^{t+1}_{x',y'}$, that any position in the 3x3 square around (x, y) is a not possible location, except position (x, y) where the query $\models \neg e^{t+1}_{x,y}$ should fail. So, use only queries of the kind $\models \neg e^{t+1}_{x',y'}$ (discover only not possible locations and consider the rest as possible locations).

Requirements

You have to satisfy the following requeriments:

- Use a clean TOP-DOWN design, with small member functions in your agent classes, such that each function performs a well defined function. You must comment each member function, explaining what the function does, its input arguments and its output (if any). Comment all the function headers and relevant class variables using javadoc comments. All your code must contain enough comments so that you can convince me that you really understand how your program works.
- Present all your code well organized, using a consistent style of indentation. Use clear and informative names for the variables you use in your class and class functions.
- Your finder agent must use **propositional logic** to reason about the possible locations of envelopes. So, the architecture of your agent will follow the one we have seen at the classroom (check the slides about knowledge based systems with propositional logic) for intelligent agents based on propositional logic, so the main process that your finder agent must implement is:
  1. When no inputs have been processed, the only knowledge of the agent is the original formula Γ you have created for the $n \times n$ world.
  2. Each time t the agent receives new information (detector sensor information), your agent must:

(a) For any location (x', y') of the world, ask whether it is not possible that an envelope is in that location, that is, the agent checks if:

$$\Gamma \cup E \models \neg e^{t+1}_{x',y'}$$

holds for its current knowledge formula Γ, where E represents the information the agent has obtained from the information of the sensor, but expressed in propositional logic (the evidence about the world), and assuming $e^{t+1}_{x',y'}$ is the variable used to encode that an envelope is located at (x', y'). As you know, you can perform the inference questions using a SAT solver. Use the `sat4j` library (mainly using the ISolver interface at org.sat4j.specs), or any other external SAT solver that you want.

(b) Update the knowledge Γ of the agent that is true so far incorporating all the clauses corresponding to the positions that have been inferred as **not possible locations**. That is, add all the clauses of the set:

$$\{ (\neg e^{t-1}_{x',y'}) \mid \Gamma \cup E \models \neg e^{t+1}_{x',y'} \}$$

So at the end of the iteration the knowledge formula Γ is updated with new information (or just before performing the next one). Observe that any location (x', y') that was previously not possible for an envelope (so $\neg e^{t-1}_{x',y'}$ was already a clause in Γ at the beginning of the iteration), will be also not possible at time step $t + 1$.

- You must implement a minimal set of testing functions in the class `EnvelopeFinderTest.java`, for testing all the example step sequences I will provide, using `junit4`. This class has some functions implemented, but some must be finished and you can add any other functions you need for this testing class.

Minimal set of functions

This is the minimal mandatory set of functions that you must implement in the class EnvelopeFinder (check the javadoc comments at the headers of such functions for explanations):

- `public void processDetectorSensorAnswer( AMessage ans ).`
- `public void addLastFutureClausesToPastClauses() .`
- `public void performInferenceQuestions().`

In the class `EnvelopeWorldEnv` you must implement:

- `acceptMessage(AMessage msg)`. Because the provided implementation only works with the `moveto` message. You must extend it to accept and answer to the other message: `detectsat`.
- `loadEnvelopeLocations( String EnvelopeFile ).`

In the class `EnvelopeWorld` (main class of the program) you must implement:

- `runStepsSequence( int wDim, int numSteps, String stepsFile, String envelopesFile ).`
- `void main ( String[] args).`

And this for the test class EnvelopeFinderTest:

- `public void testMakeSimpleStep( EnvelopeFinder eAgent, EFState targetState ).`
- `testMakeSeqOfSteps( int wDim, int numSteps, String stepsFile, String fileStates, String envelopesFile ).`

In the test class, there is an example test (function `EWorldTest1()`), that uses `testMakeSeqOfSteps` to implement one step sequence test. You can use this example to build the other tests I will ask, or use some kind of parameterized tests to implement all of them. All the other existing functions can be modified to fit the needs of your design, and you can add any additional functions you need.

What you Have to Deliver

You must deliver:

- **maven build file**. I have to be able to build your application with the maven file I have included with the initial code, or with a modified version of it. Even if you modify the maven file (with more dependencies or plugins), this is the set of maven commands that need to work OK (as I will use them when checking your application):
  1. `mvn package`: Build jar file but first execute all the unit tests found in the test subfolder
  2. `mvn test`: execute all the unit tests found in the test subfolder
  3. `mvn exec:java -Dexec.args="dim numsteps stepsfilename envelopesfilename"`: execute the main class of the program passing to the main function the required arguments.
  4. `mvn javadoc:javadoc`. Generate in html files all the documentation at the level of classes, class functions and package general documentation.
- **Documentation**. A document where you explain the design of your program. The documentation must also contain an explanation of the propositional logical formula that you have used to encode the inference rules of the agent. You can write all this documentation using javadoc comments in the different classes, and in the documentation at the level of the application package (file `package-info.java`). Or you can provide a separate PDF file for this, using javadoc comments only for class and class functions, as I have requested in the Requirements. All the javadoc html documents obtained, will be found in the folder: `target/site/apidocs/apryraz/eworld/`.
- **Code**. Give all the needed code for running your program. Use the same folder structure you have in the initial code I have provided, or modify if you think that this is needed, but then make sure that the maven build file works OK. **Do not give a program** that needs the installation of special libraries that cannot be found in maven repositories (or include the needed jar files in your code but then add the appropriate dependencies in the maven file). So, check that you use only standard java libraries or that you have provided the needed maven dependencies, so that they will be downloaded automatically by maven if needed. If you use a satsolver different from sat4j, you must, of course, include it also in your code.