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

Ramón Béjar

April 1, 2023

Goal

The goal of this work is to develop an intelligent agent for this **modification of the Treasure World problem**:

> Consider an agent in a grid world with n × n cells (like in the Barcenas Word). The agent tries to find a hidden treasure. It uses a metal sensor to try to locate it. At every location (x, y) of the world, the sensor can give three different readings:
>
> 1: means the treasure is located in one of the five cells $\{(x,y-1),(x,y),(x,y+1),(x-1,y),(x+1,y)\}$. So, it is NOT in any other cells of the world.
>
> 2: means the treasure is located in one of the four cells $\{(x+1,y+1),(x+1,y-1),(x-1,y-1),(x-1,y+1)\}$. So, it is NOT in any other cells of the world.
>
> 3: means the treasure is NOT located in any of the nine cells of the filled square centered around (x,y): $\{(x+1,y-1),(x+1,y),(x+1,y+1),(x,y-1),(x,y),(x,y+1),(x-1,y-1),(x-1,y),(x-1,y+1),\}$.
>
> We assume there is exactly one treasure in the world.

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 of course new ones can be 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.

Your application works with two main objects: the treasure finder and the world environment. It has to work with the following input:

1. Information used by the finder agent:
   (a) The dimension of the world (the value of n of the $n \times n$ world).
   (b) Number of steps (l) to perform.
   (c) A sequence of l steps of this form:

   $$x_1, y_1 \; x_2, y_2 \; ... \; x_l, y_l$$

   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.
2. Information used by the world environment object:
   (a) The dimension of the world (the same value given to the finder agent).

(b) The position x, y where the treasure is located.

With that input, your finder agent should print at the standard output (the screen) the knowledge state for possible locations of the treasure 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 `TFState`, that is already implemented, but it can modified if you need to.

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 the treasure. 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 the treasure is in that location, that is, the agent checks if:
     $$\Gamma \cup E \models \neg t^{t+1}_{x',y'}$$
     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). 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:
     $$\{ (\neg t^{t-1}_{x',y'}) \mid \Gamma \cup E \models \neg t^{t+1}_{x',y'} \}$$
     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 the treasure (so $\neg t^{t-1}_{x',y'}$ 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 `TreasureFinderTest.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 TreasureFinder (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 `TreasureWorldEnv` 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`.

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

- `runStepsSequence( int wDim, int tX, int tY, int numSteps, String fileSteps ).`
- `void main ( String[] args).`

And this for the test class TreasureFinderTest:

- `public void testMakeSimpleStep( TreasureFinder tAgent, TFState targetState ).`
- `testMakeSeqOfSteps( int wDim, int tX, int tY, int numSteps, String fileSteps ).`

In the test class, there is an example test (function `TWorldTest1()`), 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 tx ty numsteps stepsfilename"`: 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/tworld/`.
- **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.