Artificial Intelligence – Programming Assignment 2

UNIZG FER, academic year 2016/17

Handed out: 28.3.2017. Due: 9.4.207. by 23.59.

Introduction

In this lab assignment you will help agent Pacard to find the exit from the labyrinth of the evil GhostWumpus. You will solve two problems, one of which is refutation resolution and the other map search supported by refutation resolution. You will implement the algorithm of refutation resolution as well as the logic behind exploring the labyrinth of GhostWumpus.

The code describing the behavior of Pacard's world is already written and functional, and your task is to simply implement the algorithms which will control Pacard's movement and logical thinking. The code you will be using can be downloaded as a zip archive at the pages of the faculty here, or at the github repository of the class here.

The code for the lab assignment consists of Python files, a part of which you are required to read and understand in order to implement the lab assignment, a part that you will need to edit yourselves and a part that you can ignore. After unpacking the zip archive, a brief description of the files and directories you will find is as follows:

Files you will edit:	
pacard.py	File where you will implement Pacard's movement
logic.py	File where you will implement refutation resolution
Files you should read:	
logicAgents.py	Search agents and the problem definition
pacman.py	The main file that runs Pacman games
game.py	The logic behind how the Pacman world works
util.py	Useful data structures
commands.txt	Commands for running the code in txt format
Supporting files you can ignore:	
graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman

In the context of this lab assignment there will be no autograder, and we encourage you to come up with your own tests for your implementations. At the bottom of *logic.py* there is a basic test of refutation resolution, while the mini-map for testing search is defined in *layouts/miniWumpus.lay*. Larger maps you can write yourselves by creating a new file with the extension *.lay* and pass its name as an argument to pacman - *-l myWumpusLayout*.

The version of the game you control manually can be run by:

```
python pacman.py -l miniWumpus -g WumpusGhost
```

While the version of the map that is solved in the method miniWumpusSearch can be run as follows:

```
python pacman.py -l miniWumpus -p PacardAgent -a fn=miniWumpusSearch -g WumpusGhost
```

You can test your implementation of logicBasedSearch by running the following:

```
python pacman.py -l miniWumpus -p PacardAgent -a fn=logicBasedSearch -g WumpusGhost
```

The lab assignment is worth 6.25 points in total, from which an equal amount (3.125) is divided between the first part - implementing refutation resolution, and the second part - where you implement the logic based search.

Pacman GhostWumpus propositional logic problem

As you surely know, in the previous years of Artificial Intelligence captain Jean-Luc Pacard had a rough time on the planet Wumpusworld, where the vicious creature called Wumpus the Hutt teleported Pacard in his cave and tried to eat him. Thanks to the last year's students, Pacard managed to escape the evil Wumpus and lived through many adventures deep into his old age. In one of those adventures, he ended up on the planet Pacearth, where the inhabitats Pacpeople accepted him as one of their own and gave him an honorary name 'Pacard'.

What Pacard did not know is that the Pacpeople are extremely well developed scientifically, not easy to spot from their rough and primitive outside. They secretly acquired Pacard's DNA sample and by combining it with the Paco Hungreus chromosome, they created half Pacman half Pacard - who they gave a name of Lean-Juc Pacard. However, through these reckless experiments they created an anomaly in space-time which separated a part of the notorious Wumpus's soul and bound it to the vicious enemies of the Pacpeople, the ghosts. This created GhostWumpus, a creature combining the worst of the Wumpus as well as the ghosts. The creature, having learned nothing from his previous encounter with Pacard, tried to capture Pacard in his cave on the planet GhostWorld.

Lean-Juc Pacard has found himself in trouble. By using ghost magic, GhostWumpus teleported him in his cave, where he had previously laid poison capsules knowing that the biggest weakness of Pacard is his insatiable hunger. However, he did not account for the keen sense of smell of the young Pacard nor the fact that Wumpus himself will be tired from the teleporting magic to the point of being unable to move. Luckily, he managed to turn off the light in his cave on time - and the only thing Pacard can count on are his senses of smell and touch. Unluckily, Pacard provided teleportation technology go the people of Wumpusworld during his stay, and they managed to deduce the location of GhostWumpus's cave and open a teleportation exit at a random place inside of it. Help Pacard to safely reach the teleporter back to Pacearth by evading poison capsules and GhostWumpus.

GhostWumpus's cave can be represented as a 2D grid (matrix) of size N×M. Each cell in the grid is marked with exactly one of the following: *"W"* (the cell which contains GhostWumpus), *"P"* (a cell containing a poisoned capsule), *"T"* (a cell containing a teleporter) and *"O"* (regular cell – no GhostWumpus, no capsule, and no teleporter). On all cells adjacent to the cell where GhostWumpus is (we consider only edge adjacency,

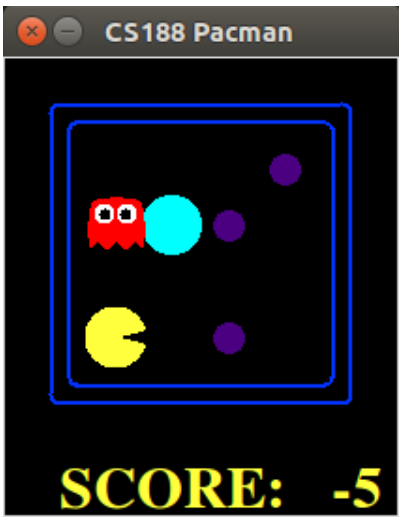

Figure 1. GhostWumpus's cave

not vertex adjacency Pacard can smell obnoxious stench of GhostWumpus, whereas on all cells adjacent to any of the capsules Pacard can sense a breeze of chemicals from them. The cell containing a teleporter emits glow to all its adjacent cells (but the cell containing a teleporter does not glow itself). An example of the GhostWumpus's cave is given in Figure 1.

Your task is to help Pacard find a safe way to the teleporter by applying the refutation resolution in propositional logic (finding a safe way, depending on the cave configuration, may or may not be achievable). In each step, Pacard tries to infer the marks for cells adjacent to the one he is currently at by using the information collected before (e.g., marks of cells he already visited) and the information stemming from his senses, i.e., information about the stench, chemicals, or glow he notices at the current cell. Sometimes Pacard may not be able to infer the mark for some adjacent cell because of insufficient information. Pacard moves across the cave (i.e., performs transitions from one cell to another) according to the following rules (rules are listed according to their priority):

1. If there is an adjacent cell for which Pacard is able to infer the mark *"T"*, Pacard moves to that cell because it contains a telerpoter and allows him to get back to Pacearth;

2. If there is an adjacent cell for which Pacard is able to infer the mark *"O"* (equivalently, Pacard can infer that *not "T" and not "P" and not "W"*), Pacard moves to that safe cell. In case of more than one safe cell surrounding the current cell, Pacard chooses to move to the one with the lowest position index (assuming the position is a pair (x, y), Pacard moves to the cell for which the value $20x + y$, i.e. position index, is the smallest);

3. If there is an adjacent cell for which Pacard is unable to infer the mark, Pacard moves to that cell. If there is more than one such cell, Pacard moves to the one with the lowest position index;

4. Pacard realizes that he cannot move anywhere (if none of the previous rules is satisfied, this means Pacard is surrounded with cells leading to certain death – cells with poisoned capsules and a cell with GhostWumpus), thus he stays at his current position, waiting to be rescued by Pacpeople.

The module performing the refutation resolution for propositional logic should be implemented independently of its application to the Wumpusworld problem, as described beneath.

Refutation resolution

You may assume that all the formulas of prepositional logic that will be given to the refutation resolution module will already be in conjunctive normal form. Put differently, you do not need to implement transformation of an arbitrary propositional logic formula to CNF.

Implement a theorem prover in propositional logic based on refutation resolution. Use the set-of-support strategy as the resolution strategy. The prover takes as input a set of premises $F_1, \ldots, F_n$ and a goal formula G. The function returns *true* if the goal formula is deductively follows from the premises, and *false* otherwise.
Make sure that you never resolve the same pair of clauses more than once and that you don't generate clauses that are already generated.
You should also implement a simplification strategy as follows. The strategy should eliminate all redundant and irrelevant clauses from the set of clauses after each application of the resolution rule. Removing redundant clauses is based on the absorption equivalence $F \wedge (F \vee G) \equiv F$. Assuming that clauses are represented as sets of literals, if the set of clauses contains a pair of clauses C_1 and C_2 for which $C_1 \subseteq C_2$, then clause C_2 may be removed from the set. Removing irrelevant clauses amounts to removing all clauses that are valid formulae. A clause is valid iff it contains a complementary pair of literals.

Putting Wumpusworld and refutation resolution together

Implement a wrapper function that takes as input the set of premises and the goal formula (in CNF format) and outputs whether the goal can be inferred from the premises. You'll implement calls to this wrapper function from the main program which navigates Pacard from through the cave.

Each cell of the cave grid, (x, y), can be described using seven propositional logic literals:

- $S_{(x,y)}$ – the stench of GhostWumpus is felt at the cell (x, y);
- $C_{(x,y)}$ – the smell of chemicals is felt at the cell (x, y);
- $G_{(x,y)}$ – a glow of the teleporter is seen at the cell (x, y);
- $W_{(x,y)}$ – GhostWumpus is found at the cell (x, y);
- $P_{(x,y)}$ – A pit is found at the cell (x, y);
- $T_{(x,y)}$ – The teleporter is found at the cell (x, y).
- $O_{(x,y)}$ – the cell (x, y) is safe.

The following logical formulas (valid for each cell of the cave grid) stem directly from the previously described rules of the GhostWumpusworld:

- If Pacard smells stench at a cell then one of the adjacent cells contains Wumpus, i.e., $S_{(x,y)} \rightarrow (W_{(x-1,y)} \vee W_{(x+1,y)} \vee W_{(x,y-1)} \vee W_{(x,y+1)})$;
- If Pacard doesn't smell stench at a cell then none of the adjacent cells contain Wumpus, i.e., $\neg S_{(x,y)} \rightarrow (\neg W_{(x-1,y)} \wedge \neg W_{(x+1,y)} \wedge \neg W_{(x,y-1)} \wedge \neg W_{(x,y+1)})$
- If Pacard feels the smell of chemicals breeze at a cell then at least one of adjacent cells contains a poisoned pill, i.e., $B_{(x,y)} \rightarrow (P_{(x-1,y)} \vee P_{(x+1,y)} \vee P_{(x,y-1)} \vee P_{(x,y+1)})$;
- If Pacard doesn't feel the smell of chemicals breeze at a cell then at none of adjacent cells contain a poisoned pill, i.e., $\neg C_{(x,y)} \rightarrow (\neg P_{(x-1,y)} \wedge \neg P_{(x+1,y)} \wedge \neg P_{(x,y-1)} \wedge \neg P_{(x,y+1)})$
- If Pacard sees glow at a cell then one of the adjacent cells contain the teleporter, i.e., $G_{(x,y)} \rightarrow (T_{(x-1,y)} \vee T_{(x+1,y)} \vee T_{(x,y-1)} \vee T_{(x,y+1)})$
- If Pacard doesn't see glow at a cell then none of the adjacent cells contain the teleporter, i.e., $\neg G_{(x,y)} \rightarrow (\neg T_{(x-1,y)} \wedge \neg T_{(x+1,y)} \wedge \neg T_{(x,y-1)} \wedge \neg T_{(x,y+1)})$
- If a cell contains Wumpus then no other cell contains Wumpus, i.e., $W_{(x,y)} \rightarrow (\neg W_{(x',y')})$ (for every (x', y') other than (x, y)).
- If Pacard doesn't feel the stench of Wumpus or the chemicals, then all the cells around that cell are safe (*"O"*), i.e. $(\neg C_{(x,y)} \wedge \neg S_{(x,y)}) \rightarrow (O_{(x-1,y)} \wedge O_{(x+1,y)} \wedge O_{(x,y-1)} \wedge O_{(x,y+1)})$
- If neither the Wumpus or a poison pill are present on a field, than that field is safe ((*"O"*)), i.e. $(\neg W_{(x,y)} \wedge \neg P_{(x,y)}) \rightarrow (O_{(x,y)}$

You can convert the above rules to CRF manually. The above set contains only the basic rules that hold in GhostWumpusworld. You are free to design other formulas which hold in GhostWumpusworld and could potentially facilitate inference, but this is not a mandatory part of the assignment. For example, what might we infer from the fact that Pacard can smell stench on two vertex-adjacent cells (knowing that GhostWumpus can only be found in one cell)?

Additional instructions, clarifications and example runs

In the second subtask, you need to help Jean-Luc Pacard to reach the teleporter by using the rules of propositional logic. As a part of the assignment, you are given a set of rules of propositional logic which hold true for each point in the cave we are in – something we might consider *the laws of the world*. In order to use those rules, however, we need some additional information, which we acquire by navigating through the cave. Jean-Luc can use his senses at every step to reach the following information: if he is at the point (x, y), he can conclude $S_{(x,y)}$ or $\neg S_{(x,y)}$, $C_{(x,y)}$ or $\neg C_{(x,y)}$, and $G_{(x,y)}$ or $\neg G_{(x,y)}$.

The information he gathers, once discovered, holds true for the entire game and can be stored in some sort of a knowledge base. We can use the knowledge to conclude something about the surrounding fields by using the laws of the world along with the information. The eight clauses we wish to prove for each neighboring state are:

1. Is GhostWumpus here or not ($W_{(x,y)}$, $\neg W_{(x,y)}$)?
2. Is a poison pill here or not ($P_{(x,y)}$, $\neg P_{(x,y)}$)?
3. Is a teleporter here or not ($T_{(x,y)}$, $\neg T_{(x,y)}$)?
4. Is the field safe or not ($O_{(x,y)}$, $\neg O_{(x,y)}$)?

Note that the order in which we attempt to prove these rules is also important, since the knowledge we would gain by proving something from the first three pairs of clauses directly influences the outcome of resolution for the fourth pair of clauses (a state is considered safe only if it does not contain Wumpus or a poison pill).

Now, we will go through an example run of the program for the map in Figure 1 with a detailed output of the resolution process.

```
Visiting:  (1, 1)
Sensed:  ~s(1, 1)
Sensed:  ~b(1, 1)
Sensed:  ~g(1, 1)
Concluded:  ~w(1, 2)
Concluded:  ~p(1, 2)
Concluded:  ~t(1, 2)
Concluded:  o(1, 2)
Concluded:  ~w(2, 1)
Concluded:  ~p(2, 1)
Concluded:  ~t(2, 1)
Concluded:  o(2, 1)
```

In state $(1, 1)$ we don't feel anything through the senses, and we conclude that all the neighboring cells are safe. We add states $(1, 2)$ and $(2, 1)$ to our list of safe states, and we visit state $(1, 2)$ as the next one due to priority.

```
Visiting:  (1, 2)
Sensed:  s(1, 2)
Sensed:  ~b(1, 2)
Sensed:  ~g(1, 2)
Concluded:  ~p(1, 3)
Concluded:  ~t(1, 3)
Concluded:  ~p(2, 2)
Concluded:  ~t(2, 2)
```

In the state $(1, 2)$ we feel the stench of GhostWumpus, and although we can't conclude on which field he is, we know that none of the neighboring fields contain either the teleporter or the poison pill. We add the neighboring states to the list of unsure states since we are unable to prove them safe or unsafe. We stil have one safe state $(2, 1)$, and we will visit it next. Notice that even though states $(1, 2)$ and $(2, 1)$ aren't nighbors, we can freely cross from one to the other – as soon as we were able to reach a state in one point, we know there is a way to get back there. The "reconstructPath()" method will handle the trip between the states.

```
Visiting:  (2, 1)
Sensed:  ~s(2, 1)
Sensed:  b(2, 1)
Sensed:  ~g(2, 1)
Concluded:  ~w(2, 2)
Concluded:  o(2, 2)
Concluded:  ~w(3, 1)
Concluded:  p(3, 1)
Concluded:  ~t(3, 1)
```

In state $(2, 1)$ we feel the smell of chemicals, and we know that one of the neighboring fields contains a poison pill. However, we don't smell the stench of GhostWumpus and we successfully conclude that neither of the neighboring fields contains GhostWumpus! This solved our problem that we had with state $(2, 2)$, where we were not sure if it contained GhostWumpus in the previous step. Now, we can conclude that the state is safe, and add it to our list of safe states to visit next.

Notice that in this step (with additional laws of the world) we could have concluded that the state $(1, 3)$ contains GhostWumpus, since all other neighboring states of state $(1, 2)$ are safe, and we felt the stench of GhostWumpus there. In the same way, we can conclude that the state $(3, 1)$ contains a poison pill, however this is not necessary for the lab assignment.

```
Visiting:  (2, 2)
Sensed:  ~s(2, 2)
Sensed:  ~b(2, 2)
Sensed:  g(2, 2)
Concluded:  ~w(2, 3)
Concluded:  ~p(2, 3)
Concluded:  o(2, 3)
Concluded:  ~w(3, 2)
Concluded:  ~p(3, 2)
Concluded:  o(3, 2)
```

We now visit state $(2, 2)$, and feel the glow of the teleporter. As we don't consider the teleporter dangerous, but we wish to reach it (corrected from the previous version of the clauses), we conclude that all the neighboring states are safe and cross to $(2, 3)$ as the next state to be teleported home.

```
Visiting:  (2, 3)
Game over: Teleported home!
```