CSCI 446 — Artificial Intelligence

Project #2:
Local Inference and the Wumpus World

1 Project Outline

First Order Logic System

This assignment aims to give you a chance to implement a basic, First-Order Logic (FOL) reasoning system. Develop a system that can take a set of rules, automatically create new arguments, and performs unification and resolution automatically. Your system should be built so that the main algorithmic processes are independent of a given problem formulation and can be expanded when new rules are developed and added to the system.

Toy example

The following knowledge base is given in Section 9.5.3 of Artificial Intelligence: a Modern Approach by Russell and Norvig. The problem describes how FOL and resolution can be used for proofs. For more detail, refer to the text and the related explanation. The power of your FOL system is that the process of deriving new facts is problem independent, given a knowledge base to process. The following is a knowledge base to implement:

1. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
2. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$
3. $\forall x\ Animal(x) \Rightarrow Loves(Jack, x)$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x\ Cat(x) \Rightarrow Animal(x)$

or, written in normal form:

1. $Animal(F(x)) \vee Loves(G(x), x)$
2. $\neg Loves(x, F(x)) \vee Loves(G(x), x)$
3. $\neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kills(x, z)$
4. $\neg Animal(x) \vee Loves(Jack, x)$
5. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
6. $Cat(Tuna)$
7. $\neg x \vee Animal(x)$

If you run your FOL system on this rule set, you should find a collection of new facts can be added to the knowledge base, specifically about Tuna and Jack. Someone killed Tuna, however, and we want to ask if it was Curiosity or Jack (the only possible suspects). To do this, let us add the rule

“$\neg Kills(Curiosity, Tuna)$” ,

and see if it is consistent with our knowledge base. After you run the FOL system, you should see that your knowledge base of truths does not contain “$Kills(Jack, Tuna)$”. This serves as the logical contradiction that we would see from a traditional proof by contradiction.

Look at the sections in the book under section 8.4.2, “Pose queries to the inference procedure” and “Debug the knowledge base”. The concepts described there may give you some hints on how to incorporated an agent into the mix as we do in the next section, and how to understand and improve your proposed knowledge base.

Wumpus World

You will also be testing your system on various instances of the Wumpus world problem. You will be provided with ten pre-built caves that range from trivially easy to very difficult. A simple cave generator is provided in Python if you decide to use randomly generated caves in your experimental design (this code is given as purely supplemental material if you decide your experimental design will benefit from using it, it is not required to complete this assignment). This script does not account for solvability and only imposes a uniform probability of adding obstacles and is free to be expanded upon if so desired.

The Wumpus world cave is an $n \times n$ grid containing exactly one gold cell. The problem's goal is to reach the gold (the trial is considered a success when the agent enters the cell with the gold). The other cells can contain either a Wumpus, a ferocious beast that does not move, a pit, a treacherous fall to the death, a wall, an obstacle to bump into, or nothing at all. Each of the Wumpus and pit give a sensory effect to each cell adjacent to the object of stench and breeze, respectively. If the agent enters the cell with either a Wumpus or a pit, the trial is terminated and considered a failure. If the agent enters the cell with a wall, they are bounced back to the previous cell, knowing they ran into a wall. If the agent enters a cell with the gold, then it will see a glitter, indicating that the gold is present in that cell.

In addition to information gained at runtime, a scout has been sent ahead, and the agent knows the number of Wumpus in the cave. The agent is equipped with the same number of arrows as Wumpus in the cave (the arrow economy is rough, and the kingdom cannot spare any extra). At any time, the agent may shoot an arrow, permanently removing it from the count, in a straight direction (along an axis and for the length of the board). If the arrow strikes a Wumpus, the agent hears a squeal, and the Wumpus is killed. The agent can NOT tell how far the Wumpus killed was. If the agent enters the cell of a dead Wumpus, it is considered a wall, returning the agent to the previous cell. The cells next to the dead Wumpus still smell.

The agent starts in the bottom left corner of the board and should control with first-order rules (not propositional!). As the agent moves around, you can assume it know how to backtrack to any previously reached point. This effectively means that the agent is allowed to teleport around the cave to explore any cell on the frontier. The more difficult the cave, the more important it is that your set of rules is complete. Most of the caves focus on specific use cases that could be encountered. Consider testing your model against these boards and monitoring the behavior to understand what rules should be incorporated. Consider some of the following problems that you could write your first edition of rules for:

- How does the agent respond when the gold is seen - is it eager, or does it ensure that it knows exactly where the gold is before stepping?
- What does the agent do when there is no way to access the gold?
- Are there rules for backtracking?
- How are ties (equally good/bad moves) broken?
- What arrow shooting rules do you have? Do you assume that when you hear a squeal, is the Wumpus adjacent to you? How conservative/eager is the agent about shooting arrows?

Notice that not all these rules are about the direction to take a step. Your agent should be able to process when to shoot an arrow and when to take a step, as both require resources, but only one can end the trial. The advantage of a FOL system is that it is expandable once we know the rules to be added. A large challenge is developing the correct set of rules.

After building the Wumpus worlds, you will use your rule set with your reasoning system and have the explorer explore each cave. You should keep track of the following statistics: The number of times the gold is found, the number of Wumpus killed, the number of times the explorer falls in a pit, the number of times the Wumpus kills the explorer, and the total number of cells explored. You should also create a simple reactive explorer that does not use the reasoning system. Instead, it decides which cell to enter randomly based on whether or not it believes the neighboring cell is safe. So, it selects first from safe neighboring cells and next from unsafe neighboring cells. Record the same statistics for this reactive explorer.

2 Deliverable Requirements

- Implement a first-order logic system that will take a given rule set and the system's current state of known truths and update the set of truths by performing unification and resolution.

- Apply your first-order logic system to the toy problem and ensure it provides the correct updated set of truths after providing knowledge.
- Implement a resolution-based inference system that will analyze each state the explorer is in, update the explorer's knowledge of the world, and use that knowledge to have the explorer make a decision. You should keep track of known safe cells and a "frontier" of cells for exploration and use those to guide the decision-making process. This system should implement your first-order logic system.
- Develop a collection of FOL rules that will drive your explorer that your resolution-based inference system can use. You should test these rules to ensure they are complete enough to solve the system reliably (when a solution is probable to find).
- Write a design document that outlines the architecture of your software. In the architecture design requirements, you should add a preliminary set of rules that you will fill the system with before expanding the rule set from problem exploration. Refer to the Design Document directions document.
- Run experiments, keeping track of the main decisions being made for each algorithm and the statistics highlighted at the end of the last section. To test stability or your code, you should run each of the provided caves ten times. You can assume a maximum number of iterations equal to the number of free cells in the cave.
- Write a paper that incorporates the following elements, summarizing the results of your experiments. Ensure you explain the experimental setup, tuning process, and final parameters for each algorithm. Remember that your paper must be in JMLR style, at least five pages, and up to ten.
    1. Title and author names
    2. A brief, one-paragraph abstract summarizing the results of the experiments
    3. Problem statement, including hypothesis (how do you expect the algorithms to do?)
    4. Description of the problem
    5. Description of your experimental approach
    6. Presentation of the results of your experiments
    7. A discussion of the behavior of your algorithms, combined with any conclusions you can draw. This needs to include a reasonable analysis and discussion of the caves you were not able to reliably solve; this may include why you were not able to solve it and what steps may need to be taken to solve such a cave, or if better performance many not be reasonably achieved
    8. Summary
    9. References to any external sources that helped in the development of your code and analysis
- Create a video demonstrating the functioning of your code. This video should focus on behavior and not on walking through the code. You need to show input, data structure, and output. How you do this is entirely up to you, but be sure it will convince the grader that your program works. This video can be recorded by any number of members of the team. If multiple individuals are in the video, please do what you can to manage the volume levels (so as not to "jump scare" our grader).
    1. The video is to be at most 5 minutes long
    2. The video should be provided in mp4 format. Alternatively, it can be uploaded to a streaming service such as YouTube with a link provided
    3. Fast forwarding or jumps are permitted through long computational cycles. Fast forwarding is *not permitted* whenever there is a voice-over or when results are presented
    4. Be sure to provide verbal commentary or explanation on all the elements you are demonstrating
    5. Demonstrate your code reading and parsing a toy problem rule.
    6. Show the process of resolution and demonstrate that it is done correctly
    7. Show updating the agent's knowledge base when presented with new information (in the Wumpus world problem)

8. Show the selection of two "next actions" by the agent: one for making a move and one for shooting an arrow (in the Wumpus world problem)
9. Show the steps of the knowledge, from start to finish, of the logic system for the toy problem.
10. Show the trace of the knowledge base for the agent, from start to finish, for a cave larger than 5×5. Show the complete knowledge base, previous step, and next step for at least gfour points in the traversal process.

- Submit your fully documented code, the video demonstrating the proper functioning of each algorithm, your design document, and your paper (which will include the results of the experiments).

Notice that the design document is due earlier than the rest of the project. Refer to the class calendar for due dates.

3 Project Evaluation

Your grade will be broken down as follows:

- Code structure - 10%
- Code documentation/commenting - 10%
- Proper functioning of your code, as illustrated by a 5-minute video - 30%
- Summary paper - 50%