Solving Cave Puzzles Using First-Order Logic and Forward Chaining

Anthony (AJ) Mann ANTHONYSTEVENMANN@GMAIL.COM
Artificial Intelligence Student
Montana State University
Bozeman, MT 59718, USA

Jackson Thorn JTHORN2448@GMAIL.COM
Artificial Intelligence Student
Montana State University
Bozeman, MT 59718, USA

Abstract

This paper aims to explore the use of first-order logic (FOL) and forward chaining with the goal of determining whether an agent's next move can be classified as SAFE, UNSAFE, or RISKY. The agent exists in a cave that contains two hazards: a bottomless pit and Wumpus monsters. The agent has already taken a safe path into the cave and collected information about the surrounding area: a breeze (B) is present if the path borders a pit; a stench (S) is present if the path borders a Wumpus. This information is used to build a knowledge base of facts and rules, from which forward chaining is applied to unify and resolve literals. Our inference engine correctly determined the classification of the agent's next move across all puzzle difficulties.

1 Introduction

1.1 Toy Problem

The Toy Problem is used as a testing ground for the Wumpus World problem. An initial knowledge base and query are provided. The algorithm must use first-order logic and resolution to prove the query by contradiction. New facts are produced through resolution and added to the knowledge base. This process continues until an empty clause is found, or the knowledge base has been saturated. If an empty clause is encountered at any point, the proof is found to be true. If no new clauses can be found, the knowledge base is saturated and therefore doesn't contain enough information to prove the query. The process used for solving the toy problem is linear to that of the Wumpus World, even though its knowledge base is dramatically simpler and resolution is performed quicker.

1.2 Wumpus World

The Wumpus World cave is an n x n grid that contains exactly one gold cell. An agent is tasked with finding the gold, but may run into hazards: a bottomless pit and a Wumpus. If the agent enters a cell containing either it dies, and the game ends. Each cell entered by the agent contains clues about adjacent cells. A breeze (B) is present if the cell borders a

pit; a stench (S) is present if the cell borders a Wumpus. The agent has already taken a safe path through the cave and collected information about the surrounding cells.

The agent is considering moving to a new cell. Our goal is to determine if the agent's next move can be classified as SAFE (no Pit, no Wumpus), UNSAFE (Pit OR Wumpus) or RISKY (possible Pit or Wumpus). Using known information about cells adjacent to the path already taken by the agent, a knowledge base of facts and rules can be constructed. The knowledge base is represented through literals, a statement that provides information about a cell, and clauses, a list of literals.

Depending on the size of the cave and its complexity (easy, medium, hard), the knowledge base may contain between a few dozen and a few hundred clauses. With this knowledge base, forward chaining is applied with the goal of unifying and resolving literals within clauses to create new clauses. Doing this allows the inference engine to create proof by contradiction, and ultimately devise new clauses that give the engine the ability to properly classify the agent's next move.

1.3 Hypothesis

The inference engine's performance is tracked by the total number of unifications and resolutions taken to reach its result. We hypothesize that our inference engine will perform more unifications and resolutions as the complexity of the cave increases. The total number of unifications and resolutions may depend on the knowledge base's creation, while poor initialization may lead to improper classification of the agent's next move.

2 Descriptions of Implemented Algorithms

2.1 Resolution

Resolution is the backbone of the inference engine. Its objective is to complete a proof by contradiction, based on the knowledge base and a negated query. The core idea of resolution is to select two clauses that contain complementary literals in order to derive new facts. By complementary literals, we are referring to any pair of literals that share the same predicate and arguments but have opposite signs (one is negated and the other is not). The process of identifying such a pair is known as unification.

Once a unifying pair is found between two parent clauses, a new clause—the resolvent—is constructed from all the remaining, non-unifying literals. This new fact is then added to the knowledge base. This resolution process continues until one of two conditions is met. The primary goal is to derive an empty clause, which occurs when two clauses containing only a single, complementary pair are resolved (e.g., [Wumpus(1,2), False] and [Wumpus(1,2), True]). The generation of an empty clause signifies a logical contradiction, which proves the original negated query to be true.

Alternatively, the process may continue until saturation, a state where applying the resolution rule no longer generates any new clauses. If the knowledge base reaches saturation

before an empty clause is found, it indicates that the available information is insufficient to complete the proof.

2.2 Unification

Unification is a critical part of the resolution process. In the context of our system, we employ a direct form of propositional unification. Its primary goal is not to find substitutions, but rather to act as a strict pattern-matching algorithm that verifies if two literals are a complementary pair.

Unification receives two literals and succeeds only if three conditions are met: the literals must share the same predicate (e.g., both are 'Wumpus'), they must refer to the exact same arguments (e.g., both concern the cell (1,2)), and they must have opposite negation. If any of these conditions fail, unification fails, and the resolution engine moves on to the next pair of clauses. This form of unification provides a computationally efficient and direct method for identifying the contradictions needed for resolution.

3 Experimental Approach

3.1 Environment and Data

The algorithm was implemented in Python using a Jupyter Notebook. Each cave puzzle is provided as a .txt file and contains the size of the cave, the number of arrows available to the agent, the path the agent has already taken, the agent's proposed next move and the expected outcome of that move. The puzzle is loaded directly into a dictionary that captures these variables. The number of arrows available to the agent is directly translated into the total number of Wumpuses that exist in the cave. The path the agent has already taken includes the cell's coordinates along with breeze (B) and stench (S) perceptions. For example: (1,0) B:F S:T indicates that there is no breeze at that cell, but there is a stench.

3.2 Implementation

3.2.1 Knowledge Base

The knowledge provided by the agent's path is used to derive facts about the surrounding cells. Each literal is created with a predicate, the cell's coordinates, and a negation. The predicate can be Breeze, Pit, Stench, or Wumpus, while the negation indicates the truth value of the statement. These literals are combined into clauses. For example:

- $[(\text{Pit}, (1, 0), \text{True})]$ is a fact, meaning there is no pit at $(1, 0)$
- $[(\text{Stench}, (0, 0), \text{False}), (\text{Wumpus}, (0, 1), \text{True})]$
  If there is no stench at $(0, 0)$ then there is no Wumpus at $(0, 1)$

The construction of the knowledge base starts with asserting the foundational facts of the already taken path. Visited cells cannot contain a Pit or Wumpus and are added as absolute truths. The same logic applies to any cell that contains a stench or breeze.

These facts are used to derive logical rules about cells adjacent to the agent's path. If

there is no perception of stench or breeze, all neighbors must be free of a Wumpus or Pit. If there is a perception of stench or breeze, one of the nearby neighbors must have a Wumpus or Pit. This holds if and only if at least one adjacent, unvisited cell contains a Wumpus or Pit. Additionally, for any given neighbor, if it has a Wumpus or Pit, it must cause a stench or breeze at the already visited cell.

While the agent can deduce rules from its immediate surroundings, local knowledge is often insufficient. This creates ambiguous scenarios where the agent cannot make a definitive conclusion. To resolve this ambiguity, the knowledge base is expanded with global constraints derived from the total Wumpus count. For a set of candidate squares where N Wumpuses are known to exist, clauses are added to enforce this exact count. "At-most-N" clauses make it a contradiction for any combination of N+1 Wumpuses to exist, while "at-least-N" clauses ensure that any conclusion resulting in fewer than N Wumpuses also leads to a contradiction. This allows for the resolution engine to prove the status of a square by invalidating assumptions that would violate the total Wumpus count for the entire cave.

Before the inference process can begin, the knowledge base must be simplified. Unit propagation is an effective strategy that resolves facts and rules to create simpler, new facts. Unit clauses (clauses containing one literal) that already exist in the knowledge base are compared with rules (clauses with two or more literals). If the rule contains a literal opposite to the selected unit clause, a simpler clause is generated. This persists across the entire knowledge base until no more simplifications can be found. Simplification reduces the search space, therefore dramatically reducing the time taken for the inference engine to find proofs.

3.2.2 Unification & Resolution

In order for the agent to decide if its next move can be classified as SAFE, UNSAFE, or RISKY, it requires a mechanism to perform logical deduction. A resolution-based inference engine systematically derives new knowledge from existing facts and rules through the process of unification and resolution.

From two existing clauses, a resolvent can be generated. In order for this to occur, unification must first be satisfied. Unification is a pattern-matching process that succeeds if two literals are opposing: they share the same predicate and arguments but have opposite negation. For instance, the literal ('Wumpus', (1, 2), False) unifies with ('Wumpus', (1, 2), True).

When such a pair is found, resolution produces a new clause containing all literals from both parents, excluding the unified pair. For example, resolving the clause [('Stench', (0,1), True), ('Wumpus', (0,2), False)] with the fact [('Stench', (0,1), False)] yields the simpler fact [('Wumpus', (0,2), False)]. Resolution allows for the engine to chain deductions together, progressively simplifying complex rules into facts.

3.3 Inference

To determine if the agent's next move is SAFE, UNSAFE, or RISKY, the engine uses unification & resolution to prove a hypothetical goal. The strategy implemented is proof-

by-contradiction: instead of trying to prove that a statement is true, the engine attempts to prove that the opposite is false. The negation of the statement to be proven is added as a temporary assumption to the knowledge base, where the engine then applies the resolution rule repeatedly to the entire set of clauses. If this process ever generates an empty clause, a logical contradiction has been found. This empty clause proves that the initial assumption was false, thereby proving its opposite must be true. The resolution process is constrained with an iteration limit of 10,000 to mitigate the effect of becoming stuck in a local maxima.

To prove the cell is SAFE, the engine must prove that there is no Pit AND no Wumpus. It first assumes that there is a Pit and applies the inference process. If a contradiction is found, it has proven there is no Pit. Separately, the process repeats for Wumpus. If both proofs succeed, the agent's next move is proven SAFE.

If the proof for SAFE fails, the engine proceeds to prove if the cell is UNSAFE. It makes the opposite assumption: that the cell is SAFE, which corresponds to asserting no Pit AND no Wumpus as facts. If running the inference engine with this assumption leads to a contradiction, it proves the cell cannot be safe, and it is therefore declared UNSAFE. If both proofs for SAFE and UNSAFE fail to produce a contradiction, it means the knowledge base contains insufficient information to logically determine the cell's status. In this case, the cell is concluded to be RISKY.

4 Results

Path	**Clauses**	**Resolutions**	**Unifications**	**Result**	**Expected**
P1	26	2	2	UNSAFE	UNSAFE
P2	57	7	7	RISKY	RISKY
P3	99	2	2	SAFE	SAFE
P4	95	88	93	RISKY	RISKY

Table 1: Performance of Resolution on Easy Paths

Path	**Clauses**	**Resolutions**	**Unifications**	**Result**	**Expected**
P1	321	7	7	RISKY	RISKY
P2	104	20	20	UNSAFE	UNSAFE
P3	177	2,146,428	2,336,381	RISKY	RISKY
P4	218	12	12	RISKY	RISKY

Table 2: Performance of Resolution on Medium Paths

Path	Clauses	Resolutions	Unifications	Result	Expected
P1	189	7	7	RISKY	RISKY
P2	427	3	3	SAFE	SAFE
P3	271	7	7	RISKY	RISKY
P4	576	7	7	RISKY	RISKY

Table 3: Performance of Resolution on Hard Paths

Our inference engine correctly determined if the agent's next move was SAFE, UNSAFE or RISKY for all caves. As with the Sudoku project, some of the hard paths proved to be the easiest for the inference engine to complete. One example is Hard-P4: though tasked with resolving a knowledge base 22 times larger, the engine performed fewer unifications and resolutions than it did for Easy-P4. Additionally, puzzles with an expected resolution of RISKY performed more unifications and resolutions than for SAFE and UNSAFE. This can be attributed to the engine attempting both proofs for SAFE and UNSAFE, thereby performing more unifications and resolutions.

Another thing to note is the statistical anomaly for Medium-P3. The engine performed roughly 2 million unifications and resolutions for this puzzle. When performing resolution to prove if the agent's next move can be determined UNSAFE, the knowledge base grew to 2,443 clauses and took roughly 2,000 iterations to find no contradictions. The large number of unifications and resolutions can be explained by an exhaustive exploration of the search space until no new clauses could be created. The fact that other puzzles with expected resolutions of RISKY did not perform nearly as many unifications and resolutions is due to the query cell being logically isolated: the engine could quickly confirm it had no information to work with.

We cannot reliably state that our initial hypothesis was proven correct. We found that the number of unifications and resolutions is not necessarily tied to puzzle difficulty, but rather the expected outcome of the query cell and the design of the experiment. This conclusion is difficult to come to due to the fact that of the twelve puzzles, eight of them resolved RISKY. The low sample size of SAFE and UNSAFE resolutions does not provide enough data for this experiment to be considered complete, though the inference engine does its job and correctly classifies the agent's next move in each of the cases.

5 Discussion

5.1 Challenges

There are many factors at play that could affect the performance of the algorithm. During development, we did not initially simplify our knowledge base after constructing it. This led to an extremely large search space, and the inference engine was just combining clauses and growing the support set until it could not anymore- the puzzle could not be proved SAFE/UNSAFE and was just deemed RISKY. The case of Medium-P3 is similar to how our

implementation looked before simplifying the knowledge base, and even though the count of unifications and resolutions is extremely high, the engine is working as designed- the query square had a deep logical entanglement within the knowledge base.

The method in which the knowledge base was constructed also affected the performance of the algorithm. Initially, the number of possible Wumpuses within a cave was not considered. It was found that by ignoring this information, the engine would have to rely on the "at-most-N" and "at-least-N" clauses. This led to a large number of long clauses, and when given to the inference engine, we observed an explosion of unifications and resolutions with no real proofs being found. By at least finding some of the Wumpuses, we simplified the global problem. For example, if there are 2 Wumpuses in a cave with 15 candidate cells, finding one dramatically prunes the search space. The global problem becomes "find 2 Wumpuses among 15 candidates" to "find 1 Wumpus among 14 candidates", or "at-most-two" to "at-most-one". Furthermore, the fact made by finding the Wumpus can be used in both simplifying the knowledge base and resolution. Something that we did not do, but should have, was simplify the knowledge base again after attempting to find Wumpus locations- though the performance of our engine did not show that this was necessary (except for Medium-P3, maybe).

5.2 Limitations

For RISKY puzzles, the agent is effectively paralyzed. Since it cannot reliably prove that the query cell is SAFE or UNSAFE, it cannot move. The scope of this project did not require the agent to actively traverse through the cave and find gold, but if it did, the inference engine would need to become more complex. One such approach is that the limitation of moving to just one square would be removed, and it would be able to move up, down, left or right (including the ability to backtrack). Of course, the possibility that all possible moves being RISKY exists. In this case, probabilistic reasoning could be an optimization where the engine computes the probability of a cell containing a Pit or Wumpus, and making its next move based on that probability- with the underlying risk that it may die and lose all progress. Though this is beyond the scope of what we implemented, it is an interesting point of discussion.

6 Summary

It was very interesting learning about resolution, unification, and the unique challenges that come with developing an inference engine. With more time, it would be interesting to compare different types of resolution and examine how their performance varies. It would also be interesting to add a probabilistic factor when the agent deems a cell as RISKY to decide if the agent should traverse to it, ultimately solving the puzzle by finding the gold. The majority of the building process was satisfying and rewarding; especially bringing everything together in the end. We also recognize the importance of hypothesis testing. While it may seem obvious that "easy" paths would be easier than "hard" paths, intuition can lead us astray. Developing an agent that has the capability to reason can contain many components- the biggest challenge is properly connecting them.

7 Appendix

Jackson wrote the following:

- Wumpus World Code
- Paper - abstract, 1.2, 1.3, 3.1, 3.2, 3.3, 4, 5

AJ wrote the following:

- Toy Problem Code
- Paper, 1.1, 2.1, 2.2, 4, 6