CS7050 Artificial Intelligence

Submitted to: Prof Vassil Vassilev

Submitted by

Muhammad Sajjad Hussain-24014153

Muhammad Usama Fiaz-23029706

Muhammad Umer-23035882

Usama Hussain-23030627

Problem Solving using State Space Search *and Knowledge-based Inference*

December 2, 2024

London Metropolitan University ,
London

Contents

Maze Solving by State Space Search and Knowledge-based Inference

1 Heuristic Using FOL

There are some First-Order Logic (FOL) rules to solve the behavior of BFS-based maze-solving algorithm.

1.1 Start Position Rule

- **FOL Statement:**
$$\exists x, y\,[\text{Start}(x, y)]$$
- **Explanation:** This rule asserts that a starting position exists in the maze. The predicate $\text{Start}(x, y)$ is used to specify this unique position, which is the point from where the search begins (e.g., $(0, 0)$). The rule itself doesn't assign specific values to x and y, but it declares that such a position exists somewhere in the grid.

1.2 Goal Position Rule

- **FOL Statement:**
$$\exists x, y\,[\text{Goal}(x, y)]$$
- **Explanation:** This rule indicates the presence of a goal position in the maze, represented by $\text{Goal}(x, y)$. In our case, the goal is $(4, 5)$ in the maze. Similar to the start position, this rule states that a goal position exists.

1.3 Path Validity

- **FOL Statement:**
$$\forall u, v, x, y\ [\text{Path}(u, v, x, y) \Rightarrow \text{Adjacent}(u, v, x, y) \wedge \neg\text{Obstacle}(x, y)]$$
- **Explanation:** This rule defines when a move from one cell (u, v) to another cell (x, y) is valid. For a position to be a valid step:
  - The destination cell (x, y) must be adjacent to the current cell (u, v), meaning the move is only one cell away.
  - The destination cell (x, y) must not contain an obstacle, ensuring the path only moves through open spaces.

  The predicate $\text{Path}(u, v, x, y)$ represents a valid path from one position to another, constrained to adjacent, non-obstructed positions only.

1.4 Adjacency Rules

- **FOL Statement:**
$$\forall x, y, x', y'\ \big[\text{Adjacent}(x, y, x', y') \iff \big((x' = x + 1 \wedge y' = y)$$
$$\vee (x' = x - 1 \wedge y' = y) \vee (x' = x \wedge y' = y + 1) \vee (x' = x \wedge y' = y - 1)\big)$$
- **Explanation:** This rule defines what it means for two cells to be "adjacent":

  - Two cells are adjacent if they are exactly one unit apart, either horizontally or vertically.
  - This accounts for movement in four directions: up, down, left, or right. Diagonal moves are not allowed.

This rule defines adjacency based on grid coordinates, used in the path validity rule to ensure only legal moves are made in the maze.

1.5 Non-Revisitation Rule

- **FOL Statement:**

$$\forall \text{path}, x, y\ [\text{Visited}(\text{path}, x, y) \Rightarrow \neg\text{Visited}(\text{path}, x, y)]$$

- **Explanation:** This rule enforces that each position in a path should be unique, meaning the algorithm should not revisit cells within a single path. The predicate $\text{Visited}(\text{path}, x, y)$ means that a cell (x, y) has already been visited in a particular path sequence. The rule ensures that if a position (x, y) is already part of the path, it cannot be visited again, helping prevent cycles in the path and ensuring each possible path is explored independently.

1.6 Path Completion Rule

- **FOL Statement:**

$$\forall \text{path}, x, y\ [\text{Path}(\text{Start}, (x, y)) \wedge \text{Goal}(x, y) \Rightarrow \text{Completed}(\text{path})]$$

- **Explanation:** This rule states that if a path reaches the goal position, it is marked as a completed path. The predicate Completed(path) indicates that the given path sequence starts from the start position and successfully reaches the goal.

1.7 Optimal Path Rule

- **FOL Statement:**

$$\begin{aligned}\forall\, \text{path}_1, \text{path}_2\ [&\text{Completed}(\text{path}_1) \wedge \text{Completed}(\text{path}_2)\\ &\wedge \text{Length}(\text{path}_1) \leq \text{Length}(\text{path}_2)\\ &\Rightarrow \text{Optimal}(\text{path}_1)\,]\end{aligned}$$

- **Explanation:** This rule defines the "optimal" path as the shortest path to the goal. If path_1 and path_2 both reach the goal, then path_1 is optimal if its length is less than or equal to the length of path_2. The predicate Optimal(path) identifies the shortest path, as found by BFS in the above code.