Assignment 2 - FOL Theory
Mihael Zlatev - 1MI3400543

This report presents a logical theory approach for solving a maze problem using a Forward Chaining and Backward chaining algorithm. The problem is described in terms of First-Order Logic (FOL), and the process of inference through logical rules is demonstrated. The maze is represented as a grid with cells that may have obstacles (cells in the maze which can't be visited)

Forward Chaining

Facts are defined to represent the state of the maze:

- **MazeCell(row, col)**
  Example for the 2 x 2 maze:
  MazeCell(0, 0), MazeCell(0, 1), MazeCell(1, 0), MazeCell(1, 1)
- **Obstacle(row, col)** – each MazeCell which contains an obstacle.
- **Start(n, m) ↔ (MazeCell(n, m) ∧ ¬Obstacle(n, m))**
  For example Start(0, 0)
- **End(n, m) ↔ (MazeCell(n, m) ∧ ¬Obstacle(n, m))**
  For example Start(4, 5)

Rules describe how new facts are inferred:

1. **ValidMove** - ∀x,y,dx,dy (InMaze(x,y) ∧ InMaze(x+dx, y+dy) ∧ ¬Obstacle(x+dx, y+dy)) → ValidMove(x+dx, y+dy)

2. **Neighbours** - ∀x1,y1,x2,y2 (ValidMove(x1,y1) ∧ ValidMove(x2,y2) ∧ (|x1-x2| + |y1-y2| = 1)) → Neighbours(x1,y1,x2,y2)

3. **Explorable:**
   ∀x,y (Start(x,y) → Explorable(x,y))

   ∀x1,y1,x2,y2 (ValidMove(x1,y1) ∧ ValidMove(x2,y2) ∧ Neighbours(x1,y1,x2,y2) ∧ Explorable(x1, y1)) → Explorable(x2, y2)

Forward chaining - Pseudo code:

The Forward Chaining algorithm iteratively applies rules to infer new facts until no new facts can be derived, or the goal is reached:

1. Initialize:
   - Mark Start cell as Explorable
2. Inference Process:
   - Repeat until no new Reachable cells can be found: - For each known Explorable cell (x1,y1) - Examine all neighboring cells (x2,y2) - If (x2,y2) is a ValidMove add (x2,y2) to Explorable set
3. Termination:
   - Stop when no new Explorable cells can be added or Path is constructed through Explorable

Backward Chaining

Facts

- Reusing the same facts as forward chaining

Rules

1. **Predecessor:**
   ∀x1,y1,x2,y2 (ValidMove(x1,y1) ∧ ValidMove(x2,y2) ∧ (x2 = x1 - dx ∧ y2 = y1 - dy) ∧ ((dx == 1 ∧ dy= 0) ∨ (dx == 0 ∧ dy= 1) ∨ (dx == -1 ∧ dy= 0) ∨ (dx == 0 ∧ dy == -1))) → Predecessor(x2,y2,x1,y1)

2. **Path Existence:**
   ∀x,y (End(x,y) → PathExists(x,y))

   ∀x1,y1,x2,y2 (ValidMove(x1,y1) ∧ ValidMove(x2,y2) ∧ Predecessor(x1,y1,x2,y2) ∧ PathExists(x2,y2) → PathExists(x1,y1))

Pseudo code:

1. Initialize:
   - Set that Path exists to goal
2. Inference process:
   - Recursively check if PathExistence rule applies for predecessors (Build path backwards from goal to start)
3. Termination:
   - stop when current cell is Start and path is constructed though the PathExistence rule

Unification of formulas

Unification is the process of finding a consistent substitution of variables in logical formulas, enabling general rules to match specific facts. However, **unification is not necessary in this maze-solving problem** because the algorithm works by directly matching specific facts and rules without needing to generalize them.