Farmer Problem Expert System Documentation

Introduction

This expert system is designed to solve the classic farmer problem, where the objective is to transport a farmer, a goat, a wolf, and a cabbage across a river, subject to specific constraints. The system employs Prolog, a logic programming language ideal for representing and solving problems with well-defined rules and states.

System Overview

The system models the problem as a state space where each state is defined by the positions (either east or west shore) of the farmer, goat, wolf, and cabbage. The initial state has all subjects on the east shore, and the goal is to get them all safely to the west shore.

Key Components:

Initial and Goal States: Defined as `initial_state(state(e, e, e, e))` and `goal_state(state(w, w, w, w))`, indicating the start and end conditions.

State Transition Rules: Encoded via `rule/1` predicates, specifying legal moves from one state to another.

Illegal States: Conditions under which the goat cannot be left alone with the wolf or cabbage, implemented in the `illegal/1` predicate.

Search Strategy: A Depth-First Search (DFS) algorithm (`dfs/4`) explores the state space to find a solution path from the initial to the goal state.

Implementation Details

State Representation

Each state is represented as `state(FP, GP, WP, CP)`, where `FP`, `GP`, `WP`, and `CP` indicate the positions (either `e` for east or `w` for west) of the farmer, goat, wolf, and cabbage, respectively i.e. in any state predicate, The farmer is always the first argument, second is always the goat, third is always the wolf and the fourth is always the cabbage. E.g. state(e,w,e,w) indicates that the farmer is east, goat is west, wolf is east and cabbage is west.

Transition Rules and newLocation

Transition rules (`rule/1`) define possible moves. The `newLocation/2` predicate identifies opposite shores, facilitating rule definition based on the current state. All the transition rules are used as arguments for the rule/1 predicate just to formally define them as rules. E.g.
“transport(state(X, X, Wolf, Cabbage), state(Y, Y, Wolf, Cabbage))”
is enclosed in the rule predicate as
“rule(transport(state(X, X, Wolf, Cabbage), state(Y, Y, Wolf, Cabbage)))”
This rule indicates that the farmer and the goat are moving from location X to Y.
When this rule or any other rule is triggered, the corresponding action is a new Location predicate “newLocation(X, Y)” which says the new location of X is Y. the values of X and Y are either east or west.

Hence, the rule and action :
“rule(transport(state(X, X, Wolf, Cabbage), state(Y, Y, Wolf, Cabbage))) :- newLocation(X, Y).”
will transition the “state(X, X, Wolf, Cabbage)” to state(Y, Y, Wolf, Cabbage) moving the farmer and goat from location X to location Y

Illegal States

The `illegal/1` predicate checks for states where the goat is left with the wolf or cabbage without the farmer, preventing these configurations. “illegal(state(F, G, W, C))’ is the rule that will check if a state predicate is illegal, the action triggered will be a check to make sure that if the Goat and Wolf are together without Farmer or the Goat and Cabbage are together without Farmer “(G = W, F \= G) ; (G = C, F \= G)” then it is an illegal state.

Depth-First Search (DFS)

The `dfs/4` predicate recursively explores moves, adhering to transition rules and avoiding illegal states. It backtracks upon reaching dead ends or illegal states, systematically exploring all possibilities. The dfs will first trigger the transport rule of a possible move from the current state to the next state while making sure that state has not been visited and it is not an illegal state.

Explanation of how the solution was derived
When we transition from a state to another state, it is because a rule-action was triggered. In the program I attempt to show the rule that was triggered to explain how we transitioned from one state to another. This is done in the DFS algorithm after we have checked that the next state is not illegal and not visited, we use the variable RuleText to manually construct the rule that was triggered in the DFS algorithm. It uses pattern matching to identify which transport rule is applicable based on the current state's configuration. Once a match is found and a next state (NextState) is determined, RuleText captures this rule application in a structured form, such as “RuleText = rule(transport(State, NextState))”. This serves as a clear, textual representation of the action taken, enabling the tracking of actions and decisions made by the algorithm in progressing from the initial state towards the goal state.
E.g. a single line of the output:
“Moving from state(e,w,e,e) to state(w,w,e,w)-rule(transport(state(e,w,e,e),state(w,w,e,w)))”
indicates that we are moving from a “state(e,w,e,e)” to “state(w,w,e,w)” with the rule “transport(state(e,w,e,e),state(w,w,e,w))” and this is know because of the pattern matching explained earlier, the only pattern that matches the movement from a state where the first and last argument in the state predicate (the difference between state(e,w,e,e) and state(w,w,e,w) is the first and last argument changed to w) are the only changes is the rule “transport(state(X, Goat, Wolf, X), state(Y, Goat, Wolf, Y))”

User Interaction

Running the System: The `run/0` predicate prompts for the knowledge base file, the user loads it, and then it initiates the solving process.

Displaying Solutions: Solutions are displayed through `display_solution/1` and `display_transitions/1`, illustrating the path from the initial to the goal state, including the rules applied at each step.

Asking for More Solutions: After displaying a solution, the system prompts the user if they wish to see another solution, allowing exploration of alternative paths.

Execution Examples

Assuming the expertSystem.pl file is consulted into GNU prolog and the user invokes “ run.”

User is asked to enter the knowledge base file name. eg 'knowledgeBase.pl'.

After displaying a solution, the system asks if the user wants to see another solution. If yes, then user enters “yes.” and the system prints any other solution and if user enters “no.” the program ends.

Illustration

```
| ?- run.

Enter the knowledge base file name: 'knowledgeBase.pl'.

compiling C:/Users/veekt/Desktop/project/knowledgeBase.pl for byte code...

C:/Users/veekt/Desktop/project/knowledgeBase.pl compiled, 20 lines read - 3351 bytes written, 4 ms
```

```
Solution Path:
Moving from state(e,e,e,e) to state(w,w,e,e)-rule(transport(state(e,e,e,e),state(w,w,e,e)))
Moving from state(w,w,e,e) to state(e,w,e,e)-rule(transport(state(w,w,e,e),state(e,w,e,e)))
Moving from state(e,w,e,e) to state(w,w,w,e)-rule(transport(state(e,w,e,e),state(w,w,w,e)))
Moving from state(w,w,w,e) to state(e,e,w,e)-rule(transport(state(w,w,w,e),state(e,e,w,e)))
Moving from state(e,e,w,e) to state(w,e,w,w)-rule(transport(state(e,e,w,e),state(w,e,w,w)))
Moving from state(w,e,w,w) to state(e,e,w,w)-rule(transport(state(w,e,w,w),state(e,e,w,w)))
Moving from state(e,e,w,w) to state(w,w,w,w)-rule(transport(state(e,e,w,w),state(w,w,w,w)))
Do you want to see another solution? (yes/no): yes.
Solution Path:
Moving from state(e,e,e,e) to state(w,w,e,e)-rule(transport(state(e,e,e,e),state(w,w,e,e)))
Moving from state(w,w,e,e) to state(e,w,e,e)-rule(transport(state(w,w,e,e),state(e,w,e,e)))
Moving from state(e,w,e,e) to state(w,w,e,w)-rule(transport(state(e,w,e,e),state(w,w,e,w)))
Moving from state(w,w,e,w) to state(e,e,e,w)-rule(transport(state(w,w,e,w),state(e,e,e,w)))
Moving from state(e,e,e,w) to state(w,e,w,w)-rule(transport(state(e,e,e,w),state(w,e,w,w)))
Moving from state(w,e,w,w) to state(e,e,w,w)-rule(transport(state(w,e,w,w),state(e,e,w,w)))
Moving from state(e,e,w,w) to state(w,w,w,w)-rule(transport(state(e,e,w,w),state(w,w,w,w)))
Do you want to see another solution? (yes/no): yes.
No more solutions or search stopped by the user.


(47 ms) yes
| ?-
```

Principles and Theories Applied

State Space Representation: The problem is modeled as a graph where nodes represent states, and edges represent legal transitions.

Depth-First Search (DFS): A classic algorithm for exploring paths through a graph, suitable for problems with a finite and manageable number of states.

Backtracking: DFS inherently supports backtracking, enabling exploration of alternative paths after reaching a dead end or an illegal state.

Rule-Based System: The system's logic is encoded as rules, a hallmark of expert systems, allowing declarative representation of problem constraints and transitions.

Pattern Matching: A fundamental feature in logic programming and Prolog, pattern matching is used to select appropriate rules for state transitions based on the current situation.

Conclusion

This expert system provides a structured and interactive solution to the farmer problem, illustrating the power of Prolog for rule-based reasoning and state space exploration. By leveraging logical predicates for state representation, transition rules, legality checks, and a depth-first search strategy and pattern matching, the system effectively navigates the problem's constraints to find viable solutions.