**Assignment2 Question1 Expert System with Forward Chain**

Forward chaining was employed because it starts from initial state and uses facts to make a conclusion. In this assignment, the Robocup agent uses the provided rules and applies the current facts seen by it to extract additional data (facts or actions). The rules are of the form of Horn Clauses with implication, i.e. A&B&C=>D. Representing the rules in this form made it easier to develop the knowledge base and the program flow.

All functionality to implement the expert system with Forward Chain is implemented in Brain.java. Parser functions TextFileContents(), KnowledgeBaseFromTextFileAsString() parse the rules from the text file robocupRules.txt. Prior to entering the forever while loop, init() function is called that creates String ArrayList clauses to store the rules and an integer ArrayList count to store the number of predicates in the right-hand side of the rule. As discussed with the professor in one of the office hour meetings, the facts will be what the agent sees in each simulation iteration of the forever while loop. Hence the reason, no facts are mentioned in the text file. A rule's left-hand side antecedents are assertions or facts separated by an &. Right-hand side consequents are either facts/assertions or actions. Please refer to the text file for more details regarding the format of rules. Examples of rules in the text file are:

ball\_not\_visible & ball\_direction\_unknown => action,turn,40 ; rGoal\_visible => looking\_right

In each iteration of the forever while loop in Brain.java, the agent sees new facts that are stored in a String ArrayList Current\_Iteration\_Facts. In addition, a local deep copy of the count array list is also created Integer ArrayList count\_copy. Then the function Search\_Rule\_With\_Action\_In\_RHS() is called.

**Working of the forward chaining function Search\_Rule\_With\_Action\_In\_RHS():**

The function takes the following array lists as arguments:

Current\_Iteration\_Facts, Current\_Iteration\_Actions, Forward\_Chain\_Path, Rules\_visited, count\_copy.

The outerloop (a while loop) of the function loops through while there are unprocessed facts in Current\_Iteration\_Facts. It does so by removing the first fact in the Current\_Iteration\_Facts array list and storing it inside a local string variable p.

Inside this outerloop, another for loop loops over each rule in the clauses array list. In this inside for loop, helper function premiseContains() is called to determine if the right-hand side of the rule being checked contains the fact stored in local sting variable p. If it does, the corresponding element of the count\_copy arraylist for the rule being checked is decremented by one meaning that one less predicate of the same rule will be checked next time. The for loop going over all the rules continues until a rule holds that results in an action is found; once such a rule is found, the outerloop breaks. The action right-hand side is also stored in a Current\_Iteration\_Actions array list. If the right-hand side is a new fact, it is then added to the Current\_Iteration\_Facts array list and the for loop continues.

Once the Search\_Rule\_With\_Action\_In\_RHS() function ends, if the size of Current\_Iteration\_Actions is more than 1, it implies that a rule with action in the right-hand side was found. From there, the details the Current\_Iteration\_Actions array list are further parsed to let m\_krislet object perform kick, dash or turn actions. ***Hence the function goes through each rule in the knowledgebase and adds new facts when found, and keeps running until a rule is found that results in an action. If no action is found, agent does nothing in the current simulation iteration.*** ***In the next simulation iteration, agent sees new facts and the process continues.***

The original implementation of the forward chaining algorithm is available here:

<https://snipplr.com/view/56296/ai-forward-chaining-implementation-for-propositional-logic-horn-form-knowledge-bases>

**Running and testing the code**

Once the folder containing the files is unzipped, run the provided batch file (Krislet/ TeamStart.bat). The output of the agent sims shows the rules and predicates visited in each iteration of the forever while loop by printing Forward\_Chain\_Path and Rules\_visited array lists. Here is the screen shot of example output showing that the algorithm forward chains at least two rules to get to the action:

A screenshot of a computer

Description automatically generated with medium confidence

**Assignment2 Question2 Decision Network and Expected Utility**

In assignment 1 of this course, we defined an environment state "e" as combination of variables such as Ball close to agent, Opponent goal seen by agent. For this question, we introduce new evidence that the agent has no control over, that is, "Opponent player close to the agent". There are total of 4 possible environment states "e" that the agent can be in after it performs an action a [Turn, Kick or Dash]. Here are the 4 environment states and their Utilities.

|  |  |  |
| --- | --- | --- |
| **Ball close to agent** | **Opponent goal visible to agent** | **Utility U of "e"** |
| T | T | 100 |
| T | F | 80 |
| F | T | 60 |
| F | F | 0 |

|  |
| --- |
| **Opponent player close to agent** |
| P(T) = 0.70 | P(F) =0.30 |

|  |
| --- |
| **Action** |
| Turn |
| Kick |
| Dash |

|  |  |  |
| --- | --- | --- |
| **Opponent player close to agent** | **Action** | **Ball close to agent** |
| **F** | Turn | P(T) = 0.20 | P(F) =0.80 |
| **F** | Kick | P(T) = 0.80 | P(F) =0.20 |
| **F** | Dash | P(T) = 0.60 | P(F) =0.40 |
| **T** | Turn | P(T) = 0.10 | P(F) =0.90 |
| **T** | Kick | P(T) = 0.30 | P(F) =0.70 |
| **T** | Dash | P(T) = 0.40 | P(F) =0.60 |

|  |
| --- |
| T=True | F=False | P: Probability |

Diagram

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Opponent player close to agent** | **Action** | **Opponent goal seen by agent** |
| **F** | Turn | P(T) = 0.65 | P(F) =0.35 |
| **F** | Kick | P(T) = 0.70 | P(F) =0.30 |
| **F** | Dash | P(T) = 0.60 | P(F) =0.40 |
| **T** | Turn | P(T) = 0.50 | P(F) =0.50 |
| **T** | Kick | P(T) = 0.20 | P(F) =0.80 |
| **T** | Dash | P(T) = 0.30 | P(F) =0.70 |

The diagram above shows a Decision Network for the Robocup agent along with the probabilities. Note that the probabilities change in the presence of the evidence "Opponent player close to agent". Expected Utility, EU of a given the previous state 'e' that the agent is coming from is given by:

EU(α|e) = Σ i ∈ t(e,a) P(t(e,a i)) x U( t(e,a i))

Here are the calculations of EU for each action first in the absence of evidence i.e. Opponent player close to agent = F, followed by the presence of evidence i.e. Opponent player close = T.

Ball close to agent=BCA, Opponent player close to agent=OPC, Opponent goal seen=OGS for simplicity

A picture containing text

Description automatically generated

As seen from the above calculations, in the absence of evidence i.e. Opponent player close to agent OPC, Kick action is preferred by the agent as it has the highest EU of 150. Whereas in the presence of evidence i.e. OPC = T, Dash action is preferred by the agent as it has the highest EU of 140.