**CS5346**

**Project 1:**

**Expert Profession Recommender System**

By:

Gentry Atkinson

Vishal Kumar M. G.

**Table of Contents:**

1. Introduction 2
   1. Problem Statement
   2. Solution
2. Contributors 3
3. Analysis of the Problem 4
   1. Domain and Goal
   2. Problems with existing code
   3. Proposed Solution
4. Knowledge Base Design 5
   1. Decision Tree
   2. Rules Formation
5. Inference Engine Design 6
   1. Backward Chaining Tree
   2. Rules For Backward Chaining
   3. Forward Chaining Tree
   4. Rules For Forward Chaining
6. Methodology 7
   1. Algorithm
   2. Data Structures
7. Program Implementation 8
8. Sample Run 9
   1. Sample 1
   2. Sample 2
   3. Sample 3
9. Analysis of Program Results 10
10. Comparison of Systems 11

**Introduction**

**1.1 Problem Statement**

According to the National Center for Education Statistics, the current cost of attending a 4-year institution of higher education is currently more than $26,000 annually in the United States. With these high costs being placed of students it is imperative that we give students the tools to complete their educations as efficiently as possible and to realize the greatest result from their education after they’ve graduated.

Giving students expert guidance as early as possible in their educations will help them realize the greatest benefit from the cost of their education. But our nation collectively also invests a great amount in the education of our students on top of their contribution of tuition. We rely on students being properly placed in positions were they can excel. Properly pairing well educated students with high-demand jobs assures the greatest return to the community for the expense of educating a young adult.

The earlier a student can be given the expert guidance the better. Early guidance will allow a student to shape their education most effectively. But human guidance is often an expensive commodity. With college attendance rates steadily rising, undergraduate counselors find themselves advising larger groups of students without additional resources. It is therefore greatly advantageous to students, institutions, and society that an automated career recommender be developed.

**1.2 Solution**

Modern students have nearly continuous exposure to computer and network resources. By placing an automated recommender system at their disposal, students will be given access to to early and regular career guidance.

Any automated that is applied in this problem domain will need to be able to recommend a large variety of professions to a student based on his or her interests. Furthermore it would be beneficial if this system could recommend not only a profession but a specific area within a profession.

The focus of this project will be the implementation of a system which fits these criteria.

**Contributors:**

**2.1 Gentry Atkinson**

* Graphics and Report
* Rules Base
* Variable Lists
* Definition of Profession Conditions

**2.2 Vishal Kumar Mainka Ganeshbapu**

* Object Oriented code re-factoring
* Development of Data Structures
* Industry Research
* Definition of Area Conditions

**2.3 Outside Contributors**

This project would like to thank Dr. Moonis Ali of Texas State University for providing the example code that this project was built from.

**Analysis of the Problem**

**3.1: Domain and Goal**

* Develop Rules based on the ideas and concepts we knew.
* Develop decision tree for both forward chaining and backward chaining.
* Converting the rules which we have proposed into the rules which are used as the knowledge base for both backward and forward chaining.
* Develop a program for both backward chaining and forward chaining and make it work individually.
* Then eliminate all the poorly written code and make it an efficient code.
* Develop the code into which it can be easily reused.
* Making sure both backward chaining program and Forward chaining program work in a single run.

**3.2: Problems with existing code**

* Existing program has a lot of syntax errors.
* The program had a lot of variables declared globally.
* There were a couple of GOTO (Jump statements) which is not considered to be efficient while writing the code.
* The program had arrays declared in char datatype, which would not work efficiently for an Intelligent system.
* The program was written in a very old style programming concepts and would lead to complex problems. There was no scope of reusing the code.
* All the information such as the knowledge base and inference engine data was dumped into a single code.
* Backward chaining and forward chaining was written in two separate files which is not how the intelligent system has to work!

**3.3: Proposed Solution**

* Eliminated all the syntax error.
* Modified the program without any global variables and jump statements.
* The Jump statements are converted into separate functions and made it work appropriately.
* Converted char’s of Array into String’s of Array.
* We have changed the code by implementing the OOPS concepts (Classes) to make the code more reusable and easier for understanding purpose.
* The knowledge base and Inference engine where separated into separate functions which helped us in reusing the code easily.
* Constants are used in the program to set the sizes of the array.
* Eliminated redundancy.
* Made both Backward chaining and Forward chaining program run in a single instance.

**6. Methodology**:

**Backward chaining:**

Backward chaining (or backward reasoning) is an [inference](https://en.wikipedia.org/wiki/Inference) method described colloquially as working backward from the goal. It is used in [automated theorem provers](https://en.wikipedia.org/wiki/Automated_theorem_prover), [inference engines](https://en.wikipedia.org/wiki/Inference_engine), [proof assistants](https://en.wikipedia.org/wiki/Proof_assistant), and other [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) applications

Backward chaining starts with a list of [goals](https://en.wikipedia.org/wiki/Goal) (or a [hypothesis](https://en.wikipedia.org/wiki/Hypothesis)) and works backwards from the [consequent](https://en.wikipedia.org/wiki/Consequent) to the [antecedent](https://en.wikipedia.org/wiki/Antecedent_(logic)) to see if any [data](https://en.wikipedia.org/wiki/Data) supports any of these consequents. An [inference engine](https://en.wikipedia.org/wiki/Inference_engine) using backward chaining would search the [inference](https://en.wikipedia.org/wiki/Inference) rules until it finds one with a consequent (Then clause) that matches a desired goal. If the antecedent (If clause) of that rule is not known to be true, then it is added to the list of goals (for one's goal to be confirmed one must also provide data that confirms this new rule).

**Data Structures used in Backward Chaining:**

* Clause Variable List:

Clause variable list stores all the variables used in IF part of the rules.

These variables are stored in the array with four array slots allocated for each slot.

If only one or two array slots are filled, the remaining slots are left blank.

If all the clauses in the IF part of a rule are connected by the logical operator AND, all the variables in these clauses must be instantiated before the THEN part can be executed.

We can calculate the Clause variable number using the formula Clause\_Number=4\*((Rule\_Number/10)-1) + 1.

* Conclusion Stack:

Conclusion stack is the most important data structure in implementing backward chaining. It tells which rule contains the conclusion that we are trying to reach and which clause number in the IF portion is currently examined for instantiated.

* Conclusion List:

Conclusion list consists of the consequents (THEN part). A clause variable pointer keeps track of the current rule and currently executed clause in that rule. It consists of rule number, conclusion associated with that rule number and set of conditions which yields the conclusion. Conclusion list is complete when the THEN portion of each rule is placed in the same row as the rule number. If the IF part of a rule is true, we invoke the THEN part and instantiate the conclusion.

* Variable List:

This Data structure contains two items: one is a variable name for each variable contained in the IF part of the knowledge base rules and the other item tells us whether or not the variable is instantiated. A variable only appears once in the list no matter how many condition clauses it appears. The instantiated column is always initially set to not instantiated (NI). It will be changed to instantiated (I) as each variable is set to a value.

**ALGORITHM FOR BACKWARD CHAINING:**

Step 1: Get the Conclusion for the problem from User.

Step 2: Search for first occurrence of the conclusion which is taken as input from the user in the conclusion list.

Step 3: If found place the rule on the conclusion stack using rule number and a [1] to represent the clause number. The clause number can be found by using the formula 4\*((rule\_number/10)-1)+1. If the conclusion is not found in the conclusion list, notify the user that an answer can’t be found.

Step 4: Instantiate IF clause of the statement.

Step 5: If any of the IF clause variables are not instantiated, ask the user for input.

Step 6: If one of the clauses is conclusion variable, place the conclusion variable’s rule on the top of the stack and go back to Step 4.

Step 7: If statement on top of the stack is not satisfied, remove the rule and search for next instance of the conclusion in the conclusion list

Step 8: If found go back to Step 4.

Step 9: If rule on the top of the stack is satisfied, remove it from the stack. If another conclusion variable is underneath, increment the clause number and for remaining clauses go back to step 4.

Step 10: If no other conclusion is underneath, we have solution for which user can conclude.

**FORWARD CHAINING:**

* Forward chaining (or forward reasoning) is one of the two main methods of reasoning when using an inference engine.
* Forward chaining is a popular implementation strategy for expert systems, business and production rule systems.
* Forward chaining starts with the available data and uses inference rules to extract more data (from an end user, for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When such a rule is found, the engine can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data.

* In our case, the Backward chaining process will generate a PROFESSION by considering all the inputs given by user and pass that PROFESSION to Forward chaining. Now the forward chaining will refer the information stored in the knowledge base and apply the forward chaining techniques and will provide the solution to the PROFESSION.

**DATA STRUCTURE USED IN FORWARD CHAINING:**

* Clause Variable list:

The clause variable list stores the antecedents (IF part).

* Conclusion Variable Queue:

Variable on which we are working is placed on conclusion variable queue and are served on first come first serve basis.

* Clause Variable Pointer:

It keeps track of clause within the rule being examined. It’s used to keep track of the rule and the clause within the rule being processed.

* Variable List:

The variable list is used to know whether the variable is instantiated or not. When user enters some information for a variable, then it is instantiated, and the answer given by the user is stored.

**ALGORITHM FOR FORWARD CHAINING:**

Step 1: The condition is identified.

Step 2: The condition variable is placed on the conclusion variable queue and its value is marked on the variable list

Step 3: The clause variable list is searched for the variable whose name is the same as the one in the front of the queue.

Step 4: If found, the rule number and a 1 are placed into the clause variable pointer. If not found, go to step 7.

Step 5: Each variable in the IF clause of the rule that is not already instantiated is now instantiated. The variables are in the clause variable list. If all the clauses are true, the THEN part is invoked

Step 6: The instantiated THEN part of the variable is placed in the back of the conclusion variable queue.

Step 7: When there are no more IF statements containing the variable that is at the front of the conclusion variable queue, that variable is removed.

Step 8: If there are no more variables on the conclusion variable queue, end the session. If there are more variables, go to step 3.