**Backward chaining Algorithm**

**Define the Following Functions:**

1. **search\_con** (*double* *variable*): with passing the variable as a parameter (variable). This function will find the matching variable in the conclusion list and the corresponding rule number, Ri.

Then it will call **rule\_to\_clause** (*Ri*):

1. **rule\_to\_clause** (*integer* *variable*): - This function will convert Rule # Ri to clause number, Ci, using the following formula. If the rule numbers are sequenced like 1,2,3,4,5, ……), formula is:

CLAUSE NUMBER (Ci) = 4\* (RULE NUMBER (Ri) - 1) + 1

If the rule numbers are sequenced like 10,20,30,40,50, ……), formula is:

CLAUSE NUMBER = 4\* (RULE NUMBER / 10 - 1) + 1

It has been assumed that four slots have been assigned for each rule in the Clause Variable list. If other than four has been assigned replace 4 by that number.

It will first call **update\_VL** (*Ci).*

1. **update\_VL** (*integer* *variable*): with passing Ci to it. It will all questions and will instantiate all (maximum of 4) variables starting from the location Ci in the variable list. If the variable is not in the variable list, it could be in the then clause of some rule. call Process (variable) to find its value-it will be a recursive call.
2. **validate\_Ri** (*integer* *variable, string conclusion*): with passing, Ri, to it. It will check if the values of the variables in the ‘if’ clauses of the rule, Ri, are satisfied with the values in the variable list and the derived global variable list. If they do, it will assign conclusion of the rule to the variable conclusion; otherwise, t will not assign any value and then it will return.
3. Process (variable)
   * Start a loop. In each cycle of the loop, do the following:
   * **call search\_con (variable)** with passing the goal variable as a parameter (variable). It will find the matching variable in the conclusion list and the corresponding rule number, Ri.
   * Initialize conclusion to null and call **Validate\_Ri (Ri, Conclusion)** *with passing, Ri, to it*. This function will assign the value to the conclusion variable and return. If the conclusion variable is not null, save the conclusion in the derived global variable list. If it is null, you continue the loop which will repeat the process with the next entry of the conclusion list.
4. main function

Declaration

(you are allowed to declare variables and lists global)

* Write functions’ prototypes
* Declare variables, and arrays
* Create list of rules. It may need some organization. You have to determine which format you want to use for efficient processing by **Validate\_Ri** function.
* Create Variable List (could be an array).
* Create conclusion list (could be an array).
* Create Clause Variable list (could be an array).
* Create the Derived Global Variable list.

**Processing in the main function:**

* + Identify the Goal variable (the variable whose value need to be determined)
  + call Process ( Goal variable)
  + After the program is complete, check if the derived global variable list contains, Goal variable and it is not null. If it does, print the value of the variable and pass it to Forward it chaining. That is the answer. If it is null, print a comment that Goal cannot be determined.
  + An example of derived global variable list is given below:

|  |  |  |
| --- | --- | --- |
| Variable | Instantiated | Value |
| discovery | I | yes |
| Position | I | Service Engineer |
|  |  |  |
|  |  |  |