Toxic Diagnose Expert System

An Expert System used for Emergency Room

Designed by:
Po-Teng Tseng
Thomas Lynn
Vimmi Taneja

Table of Content

1. Introduction

1.1. Problem Statement

In emergency rooms, knowing the source of poison is very important when they get a poisoned patient. Sometimes it is a problem since it's not likely all staff will have knowledge for all kinds of poisons and their responded symptoms.

1.2. Solution

Having a software to verify which poison patients possibly have would be very useful. By using a artificial intelligence system asking staff input symptoms, the software can easily searchs its knowledge base and gives out recommended treatments.

2. Contribution

2.1 Team Members:

2.1.1. Po-Teng Tseng

- Information search and analysis
- Created Forward Chaining Decision Tree
- Created Backward Chaining Decision Tree
- Backward Chaining implementation
- Combined Forward Chaining and Backward Chaining
- Created separated knowledge base
- Bug fixing

2.1.2. Thomas Lynn

- Information search and analysis
- Forward Chaining Decision Tree
- Backward Chaining Decision Tree
- Forward Chaining implementation
- Bug fixing

2.1.3. Vimmi Taneja

- Information search and analysis
- Forward Chaining Decision Tree
- Backward Chaining Decision Tree
- Bug Testing

3. Analysis of Problem

We need to have the information of the symptoms of poisons, and they should be at the knowledge base, so the program can verify the conditions based on the knowledge it has.

4. Knowledge Base Design

We made knowledge bases for both backward chaining and forward chaining process. Information are searched online. After we decided which poisons and symptoms we want to put in current version of software, we drew the decision trees for both of them, and created rule lists for both of them. The rules will be input into the software after. The rule list is in Appendix.

5. Inference Engine

There are two main parts for inference process: backward chaining and forward chaining. The program uses backward chaining to collect needed information from the user, and concludes the possible poison. The poison information will then be transferred to the forward chaining part. The forward chaining process will then find out the corresponding treatments from it's knowledge base.

6. Program Implementation

We designed two core functions: backward_chaining() and forward_chaining(). After we finished implementation, we combined them into a cpp file. forward_chaining() will get information from backward_chaining().

7. Improvement of Programs – What we did for improving the program

One goal of this project was to improve upon the existing code provided. The original codes had very bad coding styles. Using goto is generally not allowed in most conditions nowadays, due to the difficulty of reading and maintaining. Also, The codes had many global variables and fixed-size conditional statements, which makes the difficulty of changing size of arrays and reusing some elements very high. Also, the knowledge base should be separated from the program so that it's easy to extend the knowledge base from outside. By rewriting the whole program from scratch, we improved many things.

7.1. Goto statements

Theoretically all goto statements could be replaced with while loop and if statements, and become more readable. By fully understanding the algorithm we need to use, it's not difficult to rewrite the whole things by our own. The final product doesn't have any goto statements in it, and the number of loops are also reduced.

7.2. Global variables

Using too many global variables is bad for maintainability. It would be better to just declare variables in functions if they are not used everywhere. For example, the number of clause variables per statement for backward chaining is not used in forward chaining part but they might share the same name, so it's better to not declare it as global variables. Most of variables, lists and structs are only declared in the functions which need them, and then passed as arguments. Though it sometimes increases the number of arguments needed to be delivered, overall it's still increases the readability and maintainability.

For example, the variable list, conclusion list, file object and clause variable list are only declared in backward_chaining() function to reduce confusiness:

```
string backward_chaining()
{
    Variable variable_list[num_variable];
    Conclusion conclusion_list[num_statement];

    fstream knowledgeFile;
    Question clause_list[num_statement * num_clause_per_statement];
```

7.3 Not separated knowledge base

The original code hardcoded the knowledge base into the code, which is bad for extending its content in the future. By designing a new system to input all data from another file outside the program, it becomes very easy to add new rules, variables and conclusions.

7.3.1 Concept

A special format of input file should be introduced. We designed a new format for our knowledge base rules, and our program can recognize them and translate them into real rules.

For the rules like this:

```
If AnimalPoison = No and
If OnlyEyeExp = Yes and
If IrritationEYes = No and
If EyePainRednessBurns = Yes then
Poison = Sodium_Cyanide
```

We transform them in this format:

```
AnimalPoison
No
OnlyEyeExp
Yes
IrritationEYes
No
EyePainRednessBurns
Yes
then
Poison
Sodium_Cyanide
```

And our program should recognize those rules and put them into clause variable list and conclusion list.

The first line of file would be the total length of the file, so we don't have to code it in the program, which increases more flexibility. We don't have to worry about changing the size every time when we change the knowledge base file.

7.3.2 Implementation

The core part to build clause variable list and conclusion list:

```
while (countLine < sizeFile)
{
   file >> bufferInput;
   if (bufferInput == "then") //then part
   {
      file >> listConclusion[countStatement].name; //answer
      file >> listConclusion[countStatement].possibleAnswer; //answer value
      countLine++; //as it reads one more line for "then"
      countStatement++;
      countClause = 0;
   }
   else //if part
   {
      listQuestion[countStatement * clause_per_statement + countClause].name1 = bufferInput;
      file >> listQuestion[countStatement * clause_per_statement + countClause].value1;
      countClause++;
   }
   countLine+=2;
}
```

It puts the names of variables/sub-conclusions and their “correct values (for a certain conclusion)” (IF part) from the outside file into clause variable list. All elements in clause variable list are structs which includes the name of variables/sub-conclusions and their corresponding correct values.

It also put the names of conclusions and their “possible values (if all conditions are satisfied)” (THEN part) from the outside file into conclusion list.

The struct is:

```
struct Question
{
   string name1; //the name of variable or sub-conclusion
   string value1; //the correct value to satisfy this conclusion
};
```

That way, we stored the information of our rules into clause variable list. It would be better to do that instead of putting all these information into conclusion list so that conclusion list can remain easier.

The core part to build variable list from the built clause variable list:

```
int sizeVariableReal = 0;
for (int i = 0; i < sizeClause; i++)    //go through clause list
{
   if (listClause[i].name1 != "")      //if it's not a empty clause
   {
      //don't put it if it's a conclusion
      if ( check_conclusion(listClause[i].name1, listConclusion, sizeConclusion) == -1 )
      {
         for (int j = 0; j < sizeClause; j++)
         {
            //only put every variable once
            if (listVariable[j].name == listClause[i].name1)
            {
               break;
            }
            else if (listVariable[j].name == "")
            {
               listVariable[j].name = listClause[i].name1;
               sizeVariableReal = j + 1;
               break;
            }
         }
      }
   }
}
```

It basically reads the clause variable list we just created, verifies all distinct variables (not conclusions), and put them into variable list

7.3.3 Result

After above steps, we created variable list, conclusion list and clause variables list (they are all we need) by just inputting one file, and it's totally automatic. It means we successfully separated the knowledge base from the program, and it's ready to be extended anytime without changing the source code.

We didn't create the conclusion stack in this step because we don't need it. Please read the following section.

7.4 Conclusion Stack

Conclusion stack is one of data structures in the original code, but can we not to use it and still get the same result while not changing the whole concept about backward chaining? The answer is yes and it will improve not only readability but also memory consuming.

7.4.1 The original use of conclusion stack

The conclusion stack was in the original code to achieve “First in, last out”, because we might need to get sub-conclusions (or middle conclusion) first before we get the final conclusion. However, it's somehow not that easy to track the program flow comparing with recursive function, so why not to change it into recursive function?

7.4.2 Recursive function

Recursive function is known by it's easy structure. The function calls itself again and again. The first called function will be the last one to return the value, and the last one called will be the first one to finish. It's the same as stack.

7.4.3 Implementation

By creating the get_conclusion() recursive function, the whole concept of backward chaining becomes more intuitive. It will get the conclusion type you feed it, and return the value of that conclusion. If the conclusion you want to get has sub-conclusions, it calls itself again to get those sub-conclusions, return them to the first function, and return the final conclusion to you.

The recursive function get_conclusion() gets the type of conclusion and returns the answer as string:

```
string get_conclusion(string conclusion, Conclusion* listConclusion, Question* listClause,
                Variable* listVariable, int sizeConclusion, int clausePerStatement, int sizeVariable)
```

It calls it self when it's needed to get the sub-conclusion first (related lines in the function):

```
//check if variables are initiated or any of them is conclusion
int indexSubConclusion = 0;
indexSubConclusion = check_conclusion(listClause[i].name1, listConclusion, sizeConclusion);
if (indexSubConclusion != -1) //it's a conclusion
{
   listConclusion[indexSubConclusion].value = get_conclusion(listClause[i].name1,
      listConclusion, listClause, listVariable, sizeConclusion, clausePerStatement, sizeVariable);
   cout << "sub conclusion " << listConclusion[indexSubConclusion].name
      << " now is " <<listConclusion[indexSubConclusion].value << endl;
}
```

7.4.4 Result

The readability increased because you have less thing to bother when you track the program flow (conclusion stack is disappeared) by your eyes, and the implementation becomes easier.

Another plus is, it reduces the memory use because we have no stack right now and the resources used to write/read/delete the stack become zero. For maintainability, it's also a plus because we have less thing to bother. I feel it's easier to maintain it after we made the change.

7.5 Check possibilities when asking for information

In many cases, when the user gives some conditions, it's already impossible to reach certain goals, so it would be nice if we can just abandon that path and go for next possible conclusion.

7.5.1 Concept

The concept is quite easy. Every time we check a variable's value for a conclusion or ask the user for a not initiated variable, we check if the value is against the current conclusion. If any of the variable is against the current conclusion, we stop asking more variables, end this conclusion, and continue with the next conclusion

7.5.2 Implementation

This small feature is achieved by just a small function. It's executed every time when the program checks variables.

```
bool check_stop_path(Variable* listVariable, int indexVariable, Question* listClause, int indexClause)
{
   if (listVariable[indexVariable].value != listClause[indexClause].value1) { return true; }
      else { return false; }
}
```

7.5.3 Result

All unnecessary questions are abandoned, and a lot of time is saved, which increases the efficiency significantly.

7.6 Some user friendly designs

There are some designs we made for users' convenience. At the beginning, though the main use of this software is to find “Poison”, we still prepare a way for users to check other sub-conclusion like “Animal Poison”, “Snake Bite”, which provide more information users might need. Thanks for the design of recursive function, it's pretty easy to get such information with sub-conclusion without changing many things. At the end, the program also asks users if they would like to try the diagnosis again

8. Sample Run of Application

8.1 Colchicine Poison

For reaching Colchicine, the condition would be:

```
If
AnimalPoison = No
OnlyEyeExp = No
NauseaVomiting = Yes
ENTIrritationCoughSuffocation = No
RapidBreathing_LightHeadedness_Anxiety = No
RunnyNose = No
AbdPainDirr = Yes
JointPainFever = Yes
then
Poison = Colchicine
```

Animal Poison is a sub-conclusion, and its rule is:

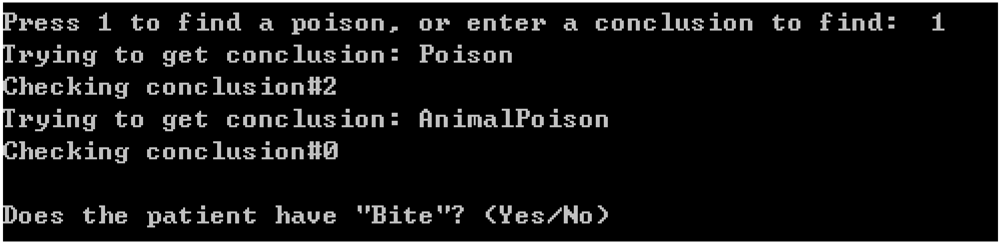
```
If
Bite = No
then
AnimalPoison = No
```

By starting the program, first we see the message:

```
Inputting file. The file size is 386
```

And we will see three lists displayed, which means all information is loaded.

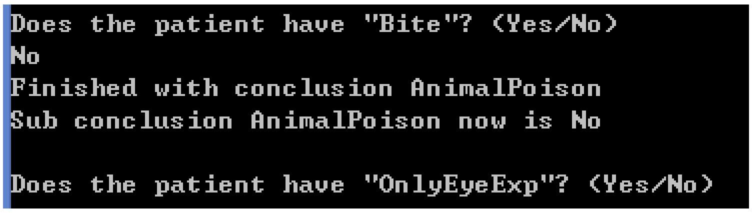
By pressing 1, we will see the question asking if the patient has bite on his/her body. It's the first question of our decision tree so it will always be asked.



```
Press 1 to find a poison, or enter a conclusion to find:  1
Trying to get conclusion: Poison
Checking conclusion#2
Trying to get conclusion: AnimalPoison
Checking conclusion#0

Does the patient have "Bite"? (Yes/No)
```

It also tells you that it's checking conclusion#2 and conclusion#0. The conclusion #2 is the closest possible conclusion in current condition, and the conclusion #0 is the sub-conclusion #2 needs, so it check #2 first then jump to #0. The question regarding bite is the question for #0.

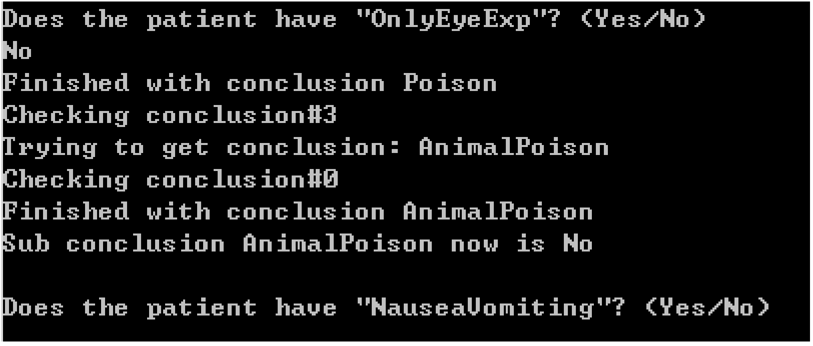
We need to answer No to it:



```
Does the patient have "Bite"? (Yes/No)
No
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No

Does the patient have "OnlyEyeExp"? (Yes/No)
```

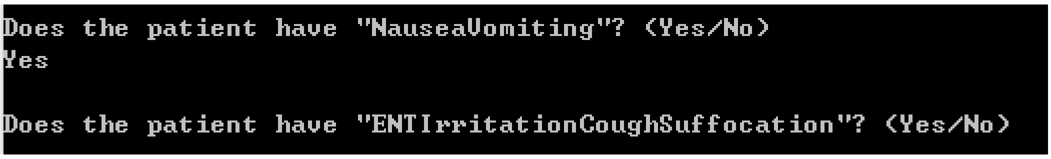
It tells you a sub-conclusion is reached, and the part with AnimalPoison is finished, which means it will continue with #2

After we answer No to it, it shows:



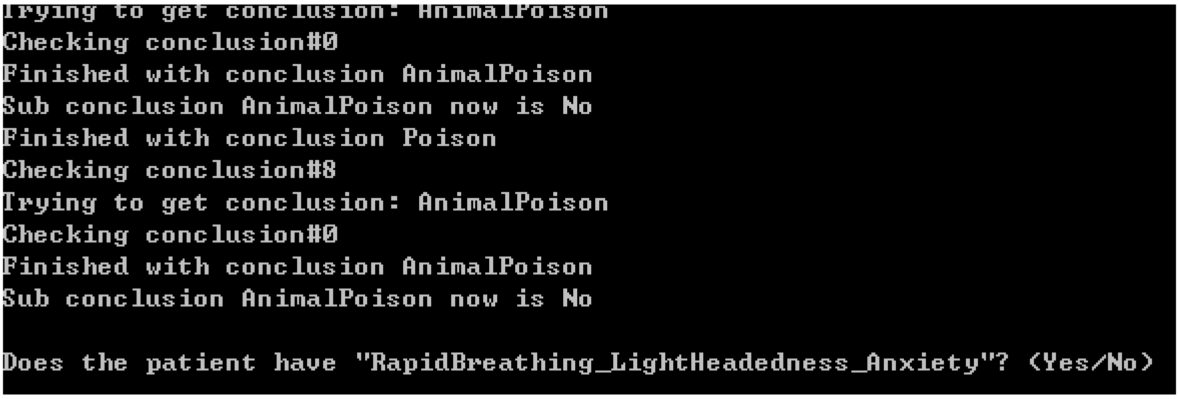
```
Does the patient have "OnlyEyeExp"? (Yes/No)
No
Finished with conclusion Poison
Checking conclusion#3
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No

Does the patient have "NauseaVomiting"? (Yes/No)
```

At here, we can see it finishes with the current conclusion, because by getting the last answer from us, it knows it's impossible to get the current conclusion #2. It continues with conclusion #3, which also needs #0 as its sub-conclusion, so it checks #0, gets it's value, comes back to the conclusion #3, and asks the next question.



```
Does the patient have "NauseaVomiting"? (Yes/No)
Yes

Does the patient have "ENTIrritationCoughSuffocation"? (Yes/No)
```

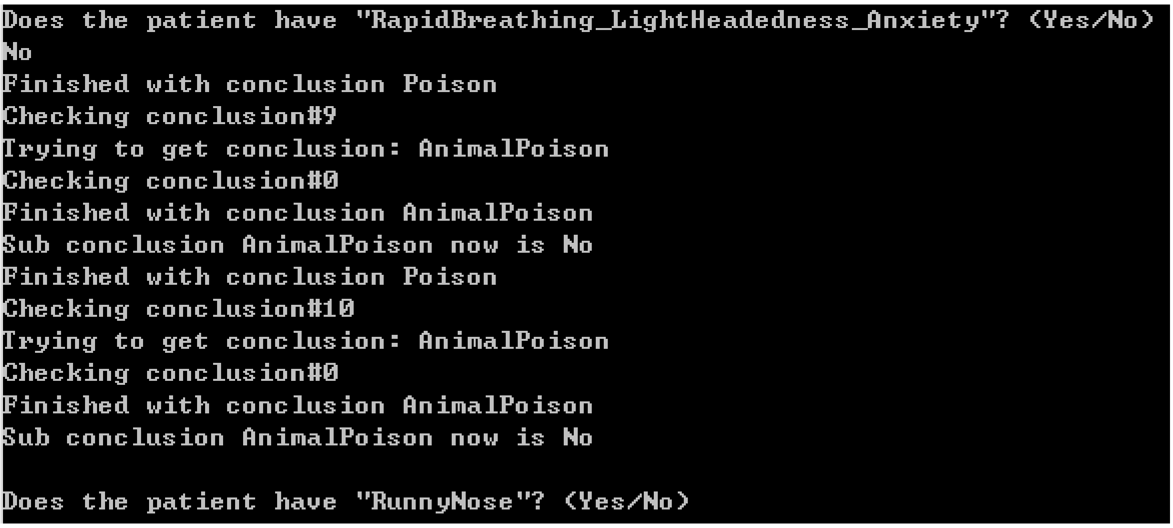
By answering No to this question, we will see it skipping many conclusions:
(The image was cut because it's too long)



```
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No
Finished with conclusion Poison
Checking conclusion#8
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No

Does the patient have "RapidBreathing_LightHeadedness_Anxiety"? (Yes/No)
```

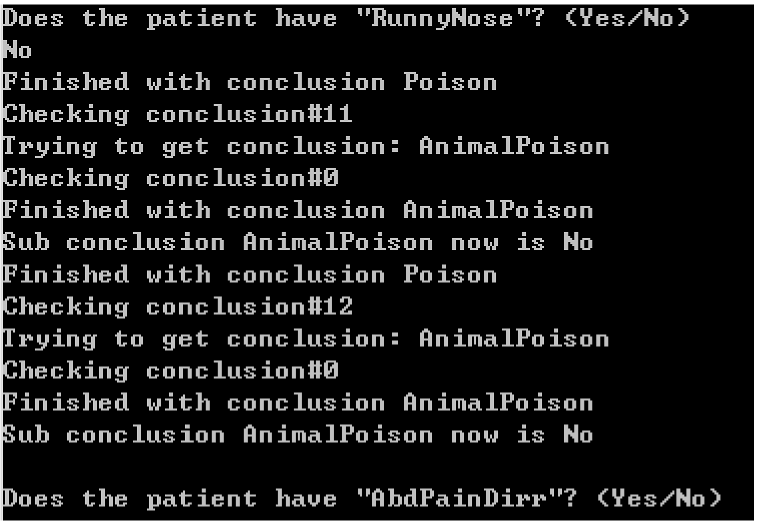
It skipped many conclusions which the current conditionals are against with, and find the next possible conclusion in this condition.

Continue by answering No to this:



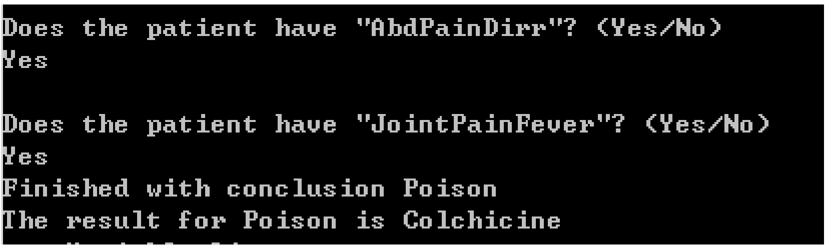
```
Does the patient have "RapidBreathing_LightHeadedness_Anxiety"? (Yes/No)
No
Finished with conclusion Poison
Checking conclusion#9
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No
Finished with conclusion Poison
Checking conclusion#10
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No

Does the patient have "RunnyNose"? (Yes/No)
```

Answering No to it generates:



```
Does the patient have "RunnyNose"? (Yes/No)
No
Finished with conclusion Poison
Checking conclusion#11
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No
Finished with conclusion Poison
Checking conclusion#12
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No

Does the patient have "AbdPainDirr"? (Yes/No)
```

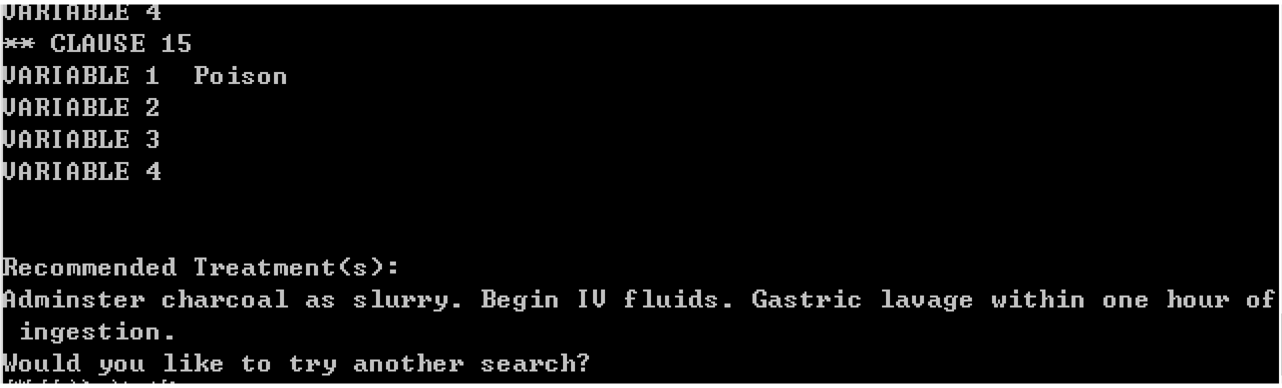
Answering Yes to it generates:



```
Does the patient have "AbdPainDirr"? (Yes/No)
Yes

Does the patient have "JointPainFever"? (Yes/No)
Yes
Finished with conclusion Poison
The result for Poison is Colchicine
```

At this moment, the Poison conclusion has been reached. The program verify all the conditions and knowing the Poison is Colchicine. This information will be delivered to forward chaining function.

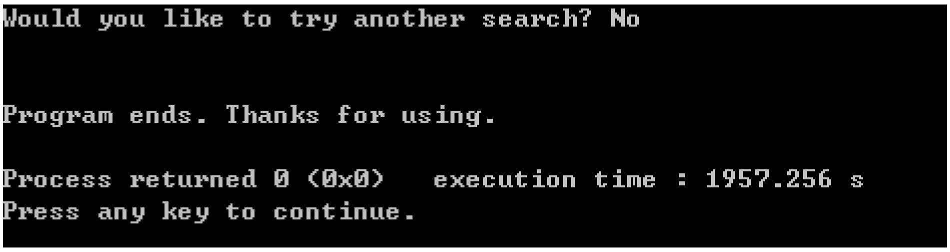
Now, the forward chaining part will work. It gives out the content of the clause variable list, and searches for the treatment corresponding to the information it gets from backward chaining.



```
VARIABLE 4
** CLAUSE 15
VARIABLE 1  Poison
VARIABLE 2
VARIABLE 3
VARIABLE 4


Recommended Treatment(s):
Adminster charcoal as slurry. Begin IV fluids. Gastric lavage within one hour of
 ingestion.
Would you like to try another search?
```

It successfully gave out the treatments for Colchicine poison.

After that, the program will ask if you would like to try it again. By inputting No, the program ends correctly.

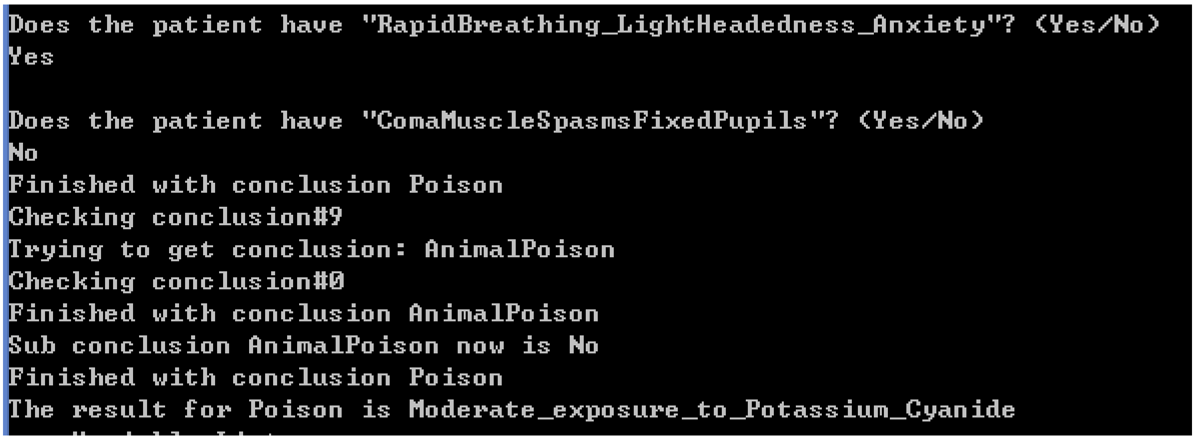


```
Would you like to try another search? No


Program ends. Thanks for using.

Process returned 0 (0x0)   execution time : 1957.256 s
Press any key to continue.
```

8.2 Moderate exposure to Potassium Cyanide

If
AnimalPoison = No
OnlyEyeExp = No
NauseaVomiting = Yes
ENTIrritationCoughSuffocation = No
RapidBreathing_LightHeadedness_Anxiety = Yes
ComaMuscleSpasmsFixedPupils = No
then
Poison = Moderate_exposure_to_Potassium_Cyanide

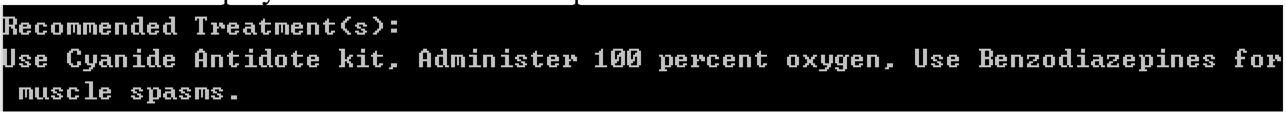
After answer “No” to “Bite”, “No” to “OnlyEyeExp”, “Yes” to “NauseaVomiting”, “No” to “ENTIrritationCoughSuffocation”, “Yes” to “RapidBreathing_LightHeadedness_Anxiety ”, and “No” to “ComaMuscleSpasmsFixedPupils”, we get:



```
Does the patient have "RapidBreathing_LightHeadedness_Anxiety"? (Yes/No)
Yes

Does the patient have "ComaMuscleSpasmsFixedPupils"? (Yes/No)
No
Finished with conclusion Poison
Checking conclusion#9
Trying to get conclusion: AnimalPoison
Checking conclusion#0
Finished with conclusion AnimalPoison
Sub conclusion AnimalPoison now is No
Finished with conclusion Poison
The result for Poison is Moderate_exposure_to_Potassium_Cyanide
```

It shows the correct conclusion “Poison = Moderate_exposure_to_Potassium_Cyanide”.

It then displays the treatment for this poison:



```
Recommended Treatment(s):
Use Cyanide Antidote kit, Administer 100 percent oxygen, Use Benzodiazepines for
 muscle spasms.
```

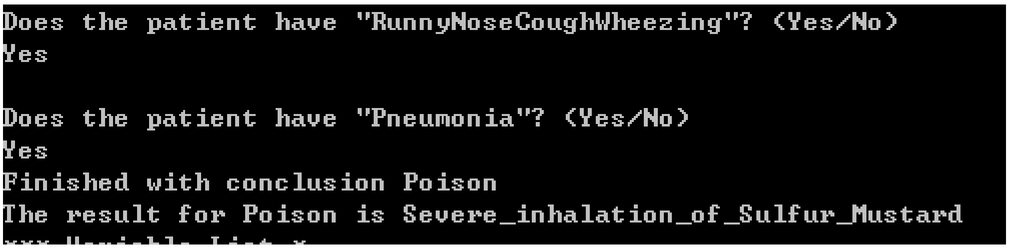
8.3 Severe_inhalation_of_Sulfur_Mustard

```
If
AnimalPoison = No
OnlyEyeExp = No
NauseaVomiting = No
MuscleSpasms = No
RunnyNoseCoughWheezing = Yes
Pneumonia = Yes
then
Poison = Severe_inhalation_of_Sulfur_Mustard
```

Again, Animal Poison is a sub-conclusion, and its rule is:

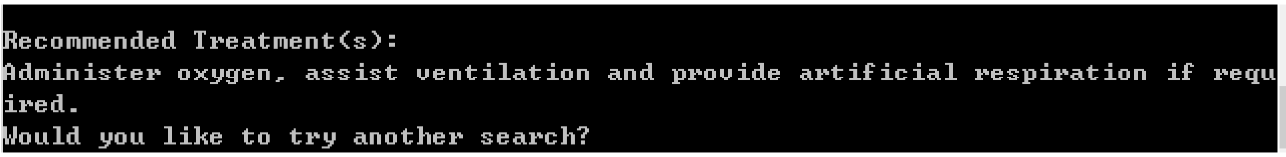
```
If
Bite = No
then
AnimalPoison = No
```

After answering “No” to “Bite”, “No” to “OnlyEyeExp”, “No” to “NauseaVomiting”, “No” to “MuscleSpasms”, “Yes” to “RunnyNoseCoughWheezing”, and “Yes” to “Pneumonia”, we get:



```
Does the patient have "RunnyNoseCoughWheezing"? (Yes/No)
Yes

Does the patient have "Pneumonia"? (Yes/No)
Yes
Finished with conclusion Poison
The result for Poison is Severe_inhalation_of_Sulfur_Mustard
```

It shows the correct conclusion “Poison = Severe_inhalation_of_Sulfur_Mustard”.
It then displays the treatment for this poison:



```
Recommended Treatment(s):
Administer oxygen, assist ventilation and provide artificial respiration if requ
ired.
Would you like to try another search?
```

9. Conclusion

As long as the symptoms are correct, we can get correct poison names and treatments. The very complicated rules we choose also helps us to prove the reliability this program has. Although we've failed once, we improved the way we code and rewrote the whole program, and finally we succeed.

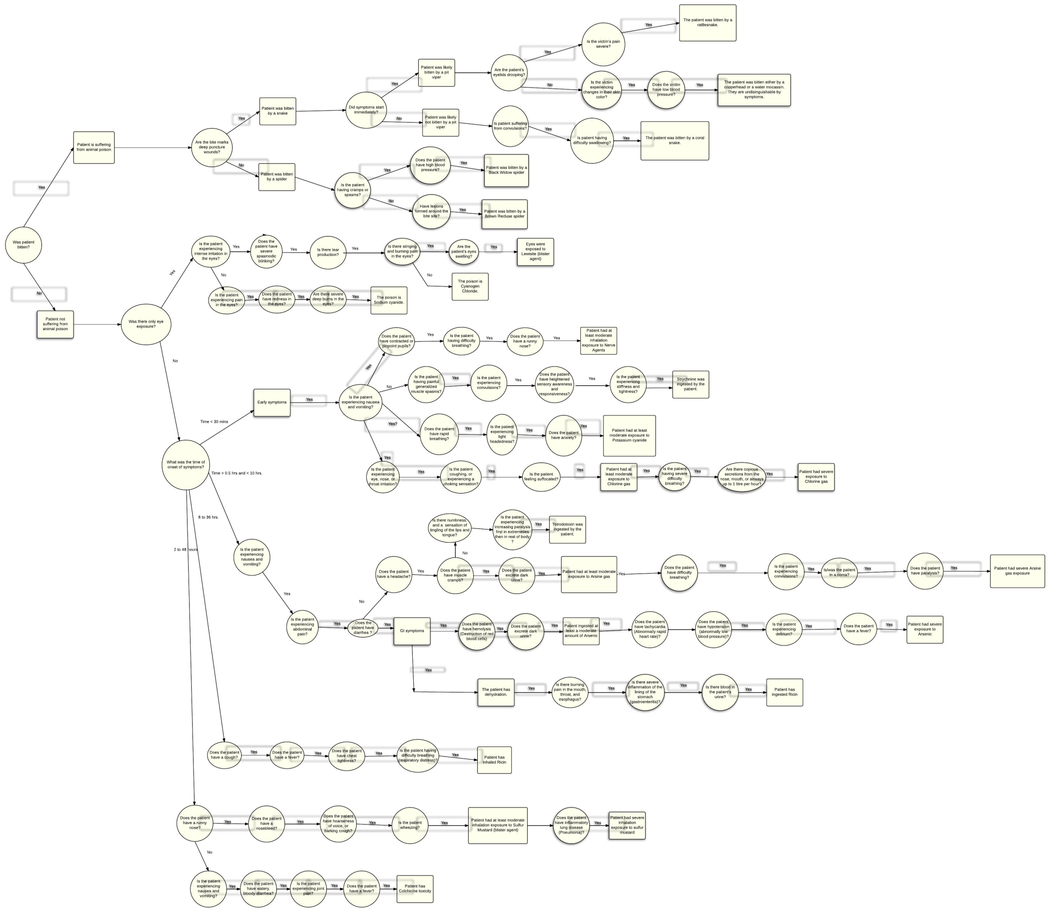
10. Appendix

10.1 backward chaining rule list

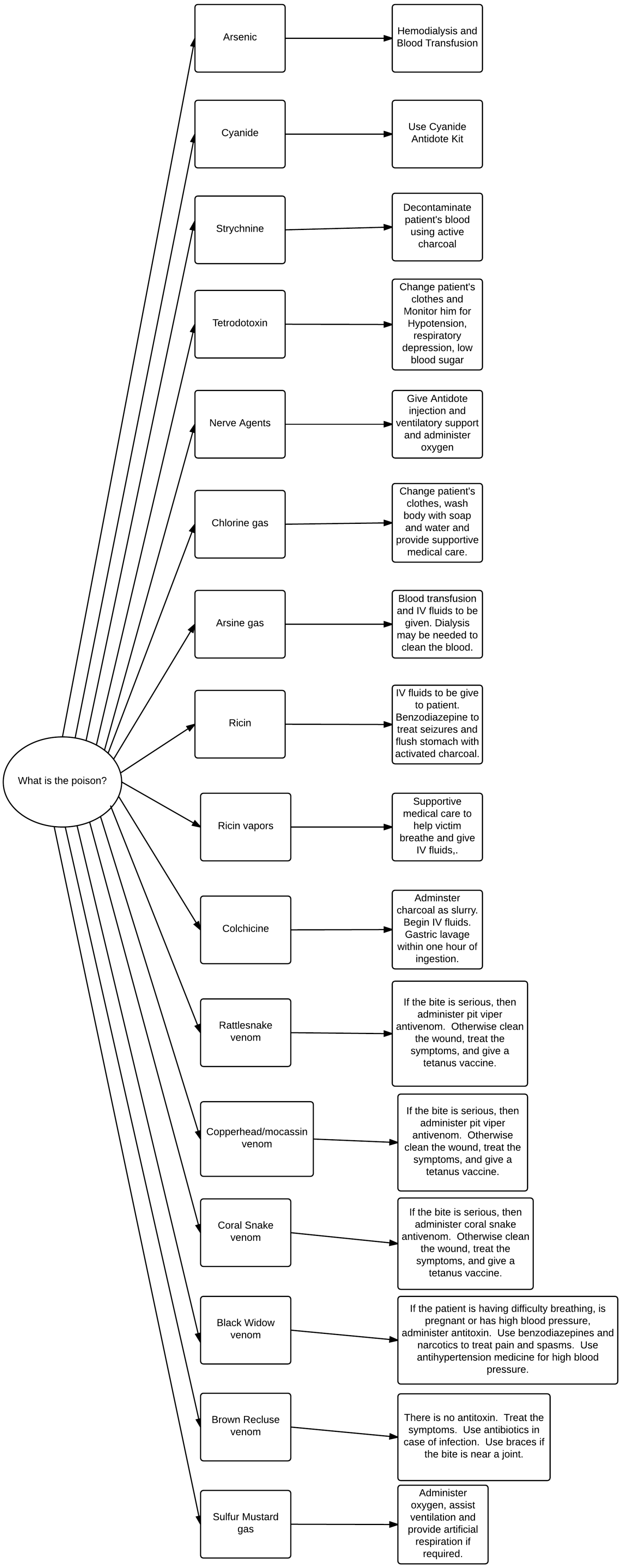
1. If Bite = No then AnimalPoison = No
2. If Bite = Yes then AnimalPoison = Yes
3. If AnimalPoison = No and OnlyEyeExp = Yes and IrritationEyes = Yes and SpasmodicBlinkingTearProd = Yes and StingingPain = No then Poison = Cyanogen Chloride.
4. If AnimalPoison = No and OnlyEyeExp = Yes and IrritationEyes = Yes and SpasmodicBlinkingTearProd = Yes and StingingPain = Yes and Swelling = Yes then Lewisite.
5. If AnimalPoison = No and OnlyEyeExp = Yes and IrritationEyes = No and EyePainRednessBurns = Yes then Poison = Sodium Cyanide
6. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = Yes and DifBreSecretion = No Then Poison = Moderate exposure to Chlorine gas
7. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = Yes and DifBreSecretion = Yes Then Poison = Severe exposure to Chlorine gas
8. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = No and MuscleSpasms = Yes and ConvRespStiff = yes then Poison = Strychnine
9. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = No and MuscleSpasms = No and RunnyNoseCoughWheezing = yes and Pneumonia = yes then Poison = Severe inhalation of Sulfur Mustard
10. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = No and MuscleSpasms = No and RunnyNoseCoughWheezing = yes and Pneumonia = No then Poison = Moderate inhalation of Sulfur Mustard
11. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathing LightHeadedness Anxiety = Yes and ComaMuscleSpasmsFixedPupils = yes Then Poison = Severe exposure to Potassium Cyanide
12. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathing LightHeadedness Anxiety = Yes and ComaMuscleSpasmsFixedPupils = No Then Poison = Moderate exposure to Potassium Cyanide
13. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathing LightHeadedness Anxiety = No and RunnyNose = yes and PinpointPupilsDiffBreath = Yes and ComaSeizuresFluidAccumulation = Yes then Poison = Severe exposure to Nerve Agents.
14. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadedness Anxiety = No and RunnyNose = yes and PinpointPupilsDiffBreath = Yes and ComaSeizuresFluidAccumulation = No then Poison = Moderate exposure to Nerve Agents.
15. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathing LightHeadedness Anxiety = No and RunnyNose = No and AbdPainDirr = Yes and JointPainFever = Yes then Poison = Colchicine
16. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and JointPainFever = No and NumbLipsTongueIncreasingParalysis = Yes then Poison = Tetrodotoxin

17. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and JointPainFever = No and NumbLipsTongueIncreasingParalysis = No and MuscleCrampsDarkUrine = Yes and ComaDiffBreConv = Yes then Poison = Severe Arsine gas exposure
18. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and JointPainFever = No and NumbLipsTongueIncreasingParalysis = No and MuscleCrampsDarkUrine = Yes and ComaDiffBreConv = No then Poison = Moderate Arsine gas exposure
19. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and Hemolysis = No and BurningPainMouthBlUrGI = Yes then Poison = Ricin
20. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and Hemolysis = Yes and TachycardiaHypotensionFever = Yes then Poison = Severe exposure to Arsenic
21. If AnimalPoison = No and OnlyEyeExp = No and NauseaVomiting = Yes and ENTIrritationCoughSuffocation = No and RapidBreathingLightHeadednessAnxiety = No and RunnyNose = No and AbdPainDirr = Yes and Hemolysis = Yes and TachycardiaHypotensionFever = No then Poison = Moderate exposure to Arsenic
22. If AnimalPoison = yes, and Puncture = yes, Then SnakeBite = yes
23. If AnimalPoison = yes, and Puncture = no, Then SpiderBite = yes
24. If SnakeBite = yes, and ImmedSympt = yes, Then PitViper = yes
25. If SnakeBite = yes, and ImmedSympt = no, Then PitViper = no
26. If PitViper = yes, and DroopingEyes = yes, and SeverePain =yes, Then Poison = Rattlesnake
27. If PitViper = yes, and DroopingEyes = no, and SkinColorChange = yes, and LowBloodPressure = yes, Then Poison = Copperhead/Moccasin
28. If PitViper = no, and Convulsions = yes, and BreathingDifficulty = yes, Then Poison = Coral Snake
29. If SpiderBite = yes, and CrampsSpasms = yes, and HighBloodPressure = yes, Then Poison = Black Widow
30. If SpiderBite = yes, and CrampsSpasms = no, and Lesions = yes, Then Poison = Brown Recluse

10.2 Backward Chaining Decision Trees


Was patient bitten?
Yes
No
Patient is suffering from animal poison
Are the bite marks deep puncture wounds?
Patient was bitten by a snake
Did symptoms start immediately?
Patient was likely bitten by a pit viper
Are the patient's eyelids drooping?
Is the victim's pain severe?
The patient was bitten by a rattlesnake.
Is the victim experiencing changes in their skin color?
Does the victim have low blood pressure?
The patient was bitten either by a copperhead or a water mocassin. They are undisinguishable by symptoms.
Patient was likely not bitten by a pit viper
Is patient suffering from convulsions?
Is patient having difficulty swallowing?
The patient was bitten by a coral snake.
Patient was bitten by a spider
Is the patient having cramps or spasms?
Does the patient have high blood pressure?
Patient was bitten by a Black Widow spider
Have lesions formed around the bite site?
Patient was bitten by a Brown Recluse spider
Patient not suffering from animal poison
Was there only eye exposure?
Is the patient experiencing intense irritation in the eyes?
Does the patient have severe spasmodic blinking?
Is there tear production?
Is there stinging and burning pain in the eyes?
Are the patient's eyes swelling?
Eyes were exposed to Lewisite (blister agent)
The poison is Cyanogen Chloride.
Is the patient experiencing pain in the eyes?
Does the patient have redness in the eyes?
Are there severe deep burns in the eyes?
The poison is Sodium cyanide.
What was the time of onset of symptoms?
Time < 30 mins
Early symptoms
Is the patient experiencing nausea and vomiting?
Does the patient have contracted or pinpoint pupils?
Is the patient having difficulty breathing?
Does the patient have a runny nose?
Patient had at least moderate inhalation exposure to Nerve Agents
Is the patient having painful generalized muscle spasms?
Is the patient experiencing convulsions?
Does the patient have heightened sensory awareness and responsiveness?
Is the patient experiencing stiffness and tightness?
Strychnine was ingested by the patient.
Yes?
Does the patient have rapid breathing?
Is the patient experiencing light headedness?
Does the patient have anxiety?
Patient had at least moderate exposure to Potassium cyanide
Is the patient experiencing eye, nose, or throat irritation?
Is the patient coughing, or experiencing a choking sensation?
Is the patient feeling suffocated?
Patient had at least moderate exposure to Chlorine gas
Is the patient having severe difficulty breathing?
Are there copious secretions from the nose, mouth, or airways up to 1 litre per hour?
Patient had severe exposure to Chlorine gas
Time > 0.5 hrs and < 10 hrs.
8 to 36 hrs.
2 to 48 hours
Is the patient experiencing nausea and vomiting?
Is the patient experiencing abdominal pain?
Does the patient have diarrhea ?
Does the patient have a headache?
Does the patient have muscle cramps?
Is there numbness, and a sensation of tingling of the lips and tongue?
Is the patient experiencing increasing paralysis first in extremeties then in rest of body ?
Tetrodotoxin was ingested by the patient.
Does the patient excrete dark urine?
Patient had at least moderate exposure to Arsine gas
Does the patient have difficulty breathing?
Is the patient experiencing convulsions?
Is/was the patient in a coma?
Does the patient have paralysis?
Patient had severe Arsine gas exposure
GI symptoms
Does the patient have hemolysis (Destruction of red blood cells)
Does the patient excrete dark urine?
Patient ingested at least a moderate amount of Arsenic
Does the patient have tachycardia (Abnormally rapid heart rate)?
Does the patient have hypotension (abnormally low blood pressure)?
Is the patient experiencing delirium?
Does the patient have a fever?
Patient had severe exposure to Arsenic
The patient has dehydration.
Is there burning pain in the mouth, throat, and esophagus?
Is there severe inflammation of the lining of the stomach (gastroenteritis)?
Is there blood in the patient's urine?
Patient has ingested Ricin
Does the patient have a cough?
Does the patient have a fever?
Does the patient have chest tightness?
Is the patient having difficulty breathing (respiratory distress)?
Patient has inhaled Ricin
Does the patient have a runny nose?
Does the patient have a nosebleed?
Does the patient have hoarseness of voice, or barking cough?
Is the patient wheezing?
Patient had at least moderate inhalation exposure to Sulfur Mustard (blister agent)
Does the patient have inflammatory lung disease (Pneumonia)?
Patient had severe inhalation exposure to sulfur mustard
Is the patient experiencing nausea and vomiting?
Does the patient have watery, bloody diarrhea?
Is the patient experiencing joint pain?
Does the patient have a fever?
Patient has Colchicine toxicity

10.3 Forward Chaining Decision Trees



What is the poison?

Arsenic → Hemodialysis and Blood Transfusion

Cyanide → Use Cyanide Antidote Kit

Strychnine → Decontaminate patient's blood using active charcoal

Tetrodotoxin → Change patient's clothes and Monitor him for Hypotension, respiratory depression, low blood sugar

Nerve Agents → Give Antidote injection and ventilatory support and administer oxygen

Chlorine gas → Change patient's clothes, wash body with soap and water and provide supportive medical care.

Arsine gas → Blood transfusion and IV fluids to be given. Dialysis may be needed to clean the blood.

Ricin → IV fluids to be give to patient. Benzodiazepine to treat seizures and flush stomach with activated charcoal.

Ricin vapors → Supportive medical care to help victim breathe and give IV fluids,.

Colchicine → Adminster charcoal as slurry. Begin IV fluids. Gastric lavage within one hour of ingestion.

Rattlesnake venom → If the bite is serious, then administer pit viper antivenom. Otherwise clean the wound, treat the symptoms, and give a tetanus vaccine.

Copperhead/mocassin venom → If the bite is serious, then administer pit viper antivenom. Otherwise clean the wound, treat the symptoms, and give a tetanus vaccine.

Coral Snake venom → If the bite is serious, then administer coral snake antivenom. Otherwise clean the wound, treat the symptoms, and give a tetanus vaccine.

Black Widow venom → If the patient is having difficulty breathing, is pregnant or has high blood pressure, administer antitoxin. Use benzodiazepines and narcotics to treat pain and spasms. Use antihypertension medicine for high blood pressure.

Brown Recluse venom → There is no antitoxin. Treat the symptoms. Use antibiotics in case of infection. Use braces if the bite is near a joint.

Sulfur Mustard gas → Administer oxygen, assist ventilation and provide artificial respiration if required.

10.4 Source Code

```
#include <cstdlib>
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
#include <fstream>

using namespace std;




struct Variable
{
    string name;
    string value;
    bool instantiate;
};

struct Conclusion
{
    string name;
    string value;
    string possibleAnswer;
};

//This struct converts "If name1 == value1 then nameConclusion = D"
struct Question
{
    string name1;
    string value1;
};


//input knowledge base from file
bool openKnowledgeBaseFile(fstream&, const char*);

//display 3 lists
void display_clauseList(Question*, int);
void display_conclusion_list(Conclusion*, int);
void display_variable_list(Variable*, int);

//build 3 lists
void build_clause_list(fstream&, Question*, Conclusion*, int);
int build_variable_list(Variable*, Question*, int, Conclusion*, int);
```

```
//check if a variable is in variable list or conclusion list
//check_variable also asks for information from the user
int check_conclusion (string, Conclusion*, int);
int check_variable(string, Variable*, int);

//check current conditions to see if more questions are not needed
bool check_stop_path(Variable*, int, Question*, int);

//get conclusion by giving the conclusion name and 3 lists and their size
string get_conclusion(string, Conclusion*, Question*, Variable*, int, int, int);

string backward_chaining();




//======================================

void IdentifyPoison();
void Treatment();
void InitializeList();
void InitializeList2();
void Instantiate(string question, string &value, bool instantiate);
void ExecRule(int rule_num);
void forward_chaining(string);




struct Variable_f
{
    int number;

    string name;

    string value;

    string question;

    bool instantiate;
};

const int conclist_size = 30,

          varlist_size = 14,

          varlist2_size = 1,

          clausevarlist_size = 330,

          clausevarlist2_size = 64,

          statement_size = 30,

          statement2_size = 15;
```

```
Variable_f varlist[varlist_size];
Variable_f conclist[conclist_size];
Variable_f clausevarlist[clausevarlist_size];
Variable_f varlist2[varlist2_size];
Variable_f clausevarlist2[clausevarlist2_size];

int conc_counter = 0,
    clause_counter = 0,
    clause_pointer = 0;

Variable_f treatment;

string arsenic_treat = "Hemodialysis and Blood Transfusion.",
        cyanide_treat = "Use Cyanide Antidote kit, Administer 100 percent oxygen, Use
Benzodiazepines for muscle spasms.",
        strychnine_treat = "Decontaminate patient's blood using active charcoal.",
        tetrodotoxin_treat = "Change patient's clothes and Monitor him for Hypotension, respiratory
depression, low blood sugar.",
        nerve_agents_treat = "Give Antidote injection and ventilatory support and administer
oxygen.",
        chlorine_treat = "Change patient's clothes, wash body with soap and water and provide
supportive medical care.",
        arsine_treat = "Blood transfusion and IV fluids to be given.  Dialysis may be needed to clean
the blood.",
        ricin_treat = "IV fluids to be give to patient. Benzodiazepine to treat seizures and flush
stomach with activated charcoal.",
        colchicine_treat = "Adminster charcoal as slurry. Begin IV fluids. Gastric lavage within one
hour of ingestion.",
        sulfur_mustard_gas_treat = "Administer oxygen, assist ventilation and provide artificial
respiration if required.",
        rattlesnake_treat = "If the bite is serious, then administer pit viper antivenom.  Otherwise clean
the wound, treat the symptoms, and give a tetanus vaccine.",
        copperhead_mocassin_treat = "If the bite is serious, then administer pit viper antivenom.
Otherwise clean the wound, treat the symptoms, and give a tetanus vaccine.",
        coral_snake_treat = "Administer Coral Snake Antivenom.",
        black_widow_treat = "Administer Black Widow Antitoxin.",
        brown_recluse_treat = "There is no antitoxin.  Treat the symptoms.  Use antibiotics in case of
infection.  Use braces if the bite is near a joint.";


Variable_f poison;

int main()
```

```
{
    string result = "";
    string inputUser = "";
    do
    {
        result = backward_chaining();

        if (result != "wrong")
        {
            forward_chaining(result);
        }
        cout << "Would you like to try another search? ";
        cin >> inputUser;
    } while (inputUser == "Yes");
    cout << "\n\nProgram ends. Thanks for using.\n";
    return 0;
}

string backward_chaining()
{
    const char *FILE_NAME_STA = "statement.txt";

    int get;
    const int num_statement = 30;
    const int num_clause_per_statement = 10;
    const int num_variable = num_statement * num_clause_per_statement;
    int num_variable_real = 0;

    Variable variable_list[num_variable];
    Conclusion conclusion_list[num_statement];

    fstream knowledgeFile;
    Question clause_list[num_statement * num_clause_per_statement];

    bool displayList = false;
    string inputUser = "";
    string result = "";
    //set default values for lists
    for (int i = 0; i < num_variable; i++)
    {
        variable_list[i].name = "";
        variable_list[i].value = "";
        variable_list[i].instantiate = false;
    }
    for (int i = 0; i < num_statement * num_clause_per_statement; i++)
    {
        clause_list[i].name1 = "";
        clause_list[i].value1 = "";
    }

    //open file
    if (openKnowledgeBaseFile(knowledgeFile, FILE_NAME_STA) == false)
    {
            cout << "There is no file existing. Please check." << endl;
            return "wrong";
    }


    build_clause_list(knowledgeFile, clause_list, conclusion_list, num_clause_per_statement);
    num_variable_real = build_variable_list(variable_list, clause_list, num_statement *
num_clause_per_statement,
                                            conclusion_list, num_statement);

    display_clauseList(clause_list, num_statement * num_clause_per_statement);
    display_variable_list(variable_list, num_variable_real);
    display_conclusion_list(conclusion_list, num_statement);


    //get conclusion from the user
    cout << "Press 1 to find a poison, or enter a conclusion to find:  ";
    cin >> inputUser;

    if (inputUser == "1") { inputUser = "Poison"; }
    result = get_conclusion(inputUser, conclusion_list, clause_list, variable_list,
                            num_statement, num_clause_per_statement, num_variable_real);
    if (result == "No Match")
    {
        cout << "There is no conclusion for this condition." << endl;
    }
    else //give out the result
    {
        cout << "The result for " << inputUser << " is " << result << endl;
    }

    return result;
    cout << "\n\nBackward Chaining Ends.\n\n\n";
}


bool openKnowledgeBaseFile(fstream& file, const char* name)
{
    file.open(name, ios::in);
    if (!file)
    {
        cout << "Knowledge base file does not exist." << endl;
        return false;
    }
    return true;
}

void display_clauseList(Question* listQuestion, int size)
{
    cout << "\n====Building clause list...==== \nClause list:\n\n";
    for (int i = 0; i < size; i++)
    {
        cout << "clause number "<< i << "  " <<listQuestion[i].name1
            << " = " << listQuestion[i].value1 << endl;
        if ( (i % 100 == 0) && (i != 0) )
        {
            cout << "HIT RETURN KEY TO CONTINUE";
            cin.get();
        }
    }
}

void display_conclusion_list(Conclusion* listConclusion, int sizeList)
{
    cout << "\nConclusion list:\n\n";
    for (int i = 0; i < sizeList; i++)
    {
        cout << i << "  " << listConclusion[i].name << " = " << listConclusion[i].value << endl;
    }
}

//It gets a fstream object and put information to clause list
void build_clause_list(fstream& file, Question* listQuestion, Conclusion* listConclusion,
                        int clause_per_statement)
{
    int sizeFile = 0;
    int countLine = 0;
    int countStatement = 0;
    int countClause = 0;
    string bufferInput;

    file >> sizeFile;
    cout << "Inputing file. The file size is " << sizeFile << endl;

    while (countLine < sizeFile)
    {
        file >> bufferInput;
        if (bufferInput == "then") //then part
        {
            file >> listConclusion[countStatement].name; //answer
            file >> listConclusion[countStatement].possibleAnswer; //answer value
            countLine++; //as it reads one more line for "then"
            countStatement++;
            countClause = 0;
        }
        else //if part
        {
            listQuestion[countStatement * clause_per_statement + countClause].name1 = bufferInput;
            file >> listQuestion[countStatement * clause_per_statement + countClause].value1;
            countClause++;
        }
        countLine+=2;
    }
}


void display_variable_list(Variable* variable_list, int sizeVariable)
{
    cout << "\nPrinting variable list:" << endl;
    for (int i = 0; i < sizeVariable; i++)
    {
        cout << variable_list[i].name << " = " << variable_list[i].value << endl;
    }
};


//return the index of conclusion if the given variable name is one of conclusions
//otherwise, return -1
int check_conclusion (string nameConclusion, Conclusion* listConclusion, int sizeConclusion)
{
    for (int i = 0; i < sizeConclusion; i++)
    {
        if (listConclusion[i].name == nameConclusion) { return i; }
    }
    return -1;
}


//It build the variable list based on clause variable list.
int build_variable_list(Variable* listVariable, Question* listClause, int sizeClause,
                        Conclusion* listConclusion, int sizeConclusion)
{
    int sizeVariableReal = 0;
    for (int i = 0; i < sizeClause; i++)    //go through clause list
    {
        if (listClause[i].name1 != "")      //if it's not a empty clause
        {
            //don't put it if it's a conclusion
            if ( check_conclusion(listClause[i].name1, listConclusion, sizeConclusion) == -1 )
            {
                for (int j = 0; j < sizeClause; j++)
                {
                    //only put every variable once
                    if (listVariable[j].name == listClause[i].name1)
                    {
                        break;
                    }
                    else if (listVariable[j].name == "")
                    {
                        listVariable[j].name = listClause[i].name1;
                        sizeVariableReal = j + 1;
                        break;
                    }
                }
            }
        }
    }
    return sizeVariableReal;
}


//check the given variable is initiated or not, and ask the user if not
//return the index of that variable for following use
int check_variable(string nameVariable, Variable* listVariable, int sizeVariable)
{
    for (int i = 0; i < sizeVariable; i++)
    {
        if (listVariable[i].name == nameVariable)
        {
            if (listVariable[i].instantiate == false)
            {
                cout << "\nDoes the patient have \"" << listVariable[i].name << "\"? (Yes/No)" << endl;
                ///////////////////need to deal with input
                cin >> listVariable[i].value;
                listVariable[i].instantiate = true;
            }
            return i;
        }
    }
    cout << "Why here???" << endl;
    return -1;
}

bool check_stop_path(Variable* listVariable, int indexVariable, Question* listClause, int
indexClause)
{
    if (listVariable[indexVariable].value != listClause[indexClause].value1) { return true; }
    else { return false; }
}


//try to get the value of given conclusion
string get_conclusion(string conclusion, Conclusion* listConclusion, Question* listClause,
                       Variable* listVariable, int sizeConclusion, int clausePerStatement, int sizeVariable)
{
    int indexConclusion = -1;
    int indexVariable = -1;
    bool stopPath = false;
    //if the answer from the user is already against the current rule, stop it.

    cout << "Trying to get conclusion: " << conclusion << endl;


    //search conclusion to find which clause has it
    //num_statement is the size of conclusion list
    for (int i = 0; i < sizeConclusion; i++)
    {
        if (conclusion == listConclusion[i].name)
        {
            indexConclusion = i;
            cout << "Checking conclusion#" << indexConclusion<<endl;

            if (indexConclusion == -1)
            {
                return "No Match";
            }

            else //examine this one conclusion and see if we can get the possible answer
            {
                //Find clause which can generate the variable/conclusion the last clause need
                for (int i = indexConclusion * clausePerStatement; i < (indexConclusion + 1) *
clausePerStatement; i++)
                {
                    if ( listClause[i].name1 != "" )
                    {
                        //check if variables are initiated or any of them is conclusion
                        int indexSubConclusion = 0;
                        indexSubConclusion = check_conclusion(listClause[i].name1, listConclusion,
sizeConclusion);
                        if (indexSubConclusion != -1) //it's a conclusion
                        {
                            listConclusion[indexSubConclusion].value =
get_conclusion(listClause[i].name1,
                                listConclusion, listClause, listVariable, sizeConclusion, clausePerStatement,
sizeVariable);
                            cout << "Sub conclusion " << listConclusion[indexSubConclusion].name
                                << " now is " <<listConclusion[indexSubConclusion].value << endl;
                        }
                        else if (listClause[i].name1 != "") //it's a variable
                        {
                            indexVariable = check_variable(listClause[i].name1, listVariable, sizeVariable);

                            if (check_stop_path(listVariable, indexVariable, listClause, i) == true)
{ break; }
                        }
                    }
                }

                cout << "Finished with conclusion " << conclusion << endl;

                //go through all conditions
                bool satisfied = true;
                for (int i = indexConclusion * clausePerStatement; i < (indexConclusion + 1) *
clausePerStatement; i++)
                {
                    if (listClause[i].name1 != "")
                    {
                        //check conclusions
                        for (int j = 0; j < sizeConclusion; j++)
                        {
                            //Need to care about not choosing the same name conclusions with empty
values
                            if ( (listConclusion[j].value != "") && (listClause[i].name1 ==
listConclusion[j].name)
                                && (listConclusion[j].value != listClause[i].value1) )
                            {
                                satisfied = false;
                            }
                        }
                        //check variables
                        for (int j = 0; j < sizeVariable; j++)
                        {
                            if ( (listClause[i].name1 == listVariable[j].name) && (listVariable[j].value !=
listClause[i].value1) )
                            {
                                //cout <<"clause = "<<listClause[i].name1<<" variable =
"<<listVariable[j].name<<endl;
                                //cout << "var = "<<listVariable[j].value << " but need to be
"<<listClause[i].value1 << endl;
                                satisfied = false;
                            }
                        }
                    }
                }
                if (satisfied == true) { return listConclusion[indexConclusion].possibleAnswer; }
            }
        }
    }
    return "No match";
```

```
}



//=======================================================


void forward_chaining(string valuePoison)
{
    Variable_f Empty;
    Empty.name = "  ";

    for(int i = 0; i < clausevarlist_size; i++)
        clausevarlist[i] = Empty;

    for(int i = 0; i < clausevarlist2_size; i++)
        clausevarlist2[i] = Empty;

    poison.name = "Poison";
    poison.value = valuePoison;       // Enter the poison type here
    poison.instantiate = true;

    varlist2[0] = poison;

    for(int i = 0; i < clausevarlist2_size; i++)
    {
        if (i % 4 == 0)
            clausevarlist2[i] = poison;
    }

    InitializeList2();

    Treatment();
}


void Treatment()
{
    treatment.name = "Treatment";
    treatment.instantiate = false;
    clause_pointer = 0;
    queue<Variable_f> cond;
    cond.push(poison);

    bool stop = false;

    while (cond.empty() == false && treatment.instantiate == false)
    {
        for (int i = 0; i < clausevarlist2_size; i++)
        {
            if (i % 4 == 0)
                clause_pointer = i;
            if (clausevarlist2[i].name == cond.front().name)
            {
               for (int j = clause_pointer; j < clause_pointer + 4; j++)
               {
                  if (clausevarlist2[i].name != cond.front().name && clausevarlist2[i].name != "  " &&
                      clausevarlist2[i].instantiate == false)
                     Instantiate(clausevarlist2[i].question, clausevarlist2[i].value,
clausevarlist2[i].instantiate);
               }
               ExecRule(i / 4 + 1);
            }
        }
        cond.pop();
    }
    if (treatment.instantiate == true)
        cout << endl << "Recommended Treatment(s):" << endl << treatment.value << endl;
}

void InitializeList()
{
    cout << "*** Conclusion List *" << endl;
    for(int i = 0; i < conclist_size; i++)
    {
        cout << "CONCLUSION " << (i + 1) << "  " << conclist[i].name << endl;
    }
    cout << "HIT RETURN KEY TO CONTINUE";
    cin.get();
    cout << endl;

    cout << "*** Variable List *" << endl;
    for (int i = 0; i < varlist_size; i++)
    {
        cout << "VARIABLE " << (i + 1) << "  " << varlist[i].name << endl;
    }
    cout << "HIT RETURN KEY TO CONTINUE";
    cin.get();
    cout << endl;


    cout << "*** Clause Variable List *" << endl;
    for(int i = 0; i < statement_size ; i++)
    {
        cout << "** CLAUSE " << (i+1) << endl;
        for(int j = 0; j < 11; j++)
        {
            cout << "VARIABLE " << (j + 1) << "  "
                 << clausevarlist[(i * 11) + j].name << endl;
        }
        if (i < 2)
        {
            cout << "HIT RETURN KEY TO CONTINUE";
            cin.get();
        }
    }
    cout << endl;
}

void InitializeList2()
{

    cout << "*** Variable List *" << endl;
    for (int i = 0; i < varlist2_size; i++)
    {
        cout << "VARIABLE " << (i + 1) << "  " << varlist2[i].name << endl;
    }
    cout << "HIT RETURN KEY TO CONTINUE";
    cin.get();
    cout << endl;


    cout << "*** Clause Variable List *" << endl;
    for(int i = 0; i < statement2_size ; i++)
    {
        cout << "** CLAUSE " << (i + 1) << endl;
        for(int j = 0; j < 4; j++)
        {
            cout << "VARIABLE " << (j + 1) << "  "
                 << clausevarlist2[(i * 4) + j].name << endl;
        }
        if (i < 2)
        {
            cout << "HIT RETURN KEY TO CONTINUE";
            cin.get();
        }
    }
    cout << endl;
}
void Instantiate(string question, string &value, bool instantiate)
{
    cout << question << "  ";
    cin >> value;
    cout << endl;

    instantiate = true;
}
void ExecRule(int rule_num)
{
    switch(rule_num)
    {
        case 1: if (poison.value == "Severe_exposure_to_Arsenic" || poison.value ==
"Moderate_exposure_to_Arsenic")
                {
                   treatment.value = arsenic_treat;
                   treatment.instantiate = true;
                }
                break;

        case 2: if (poison.value == "Severe_exposure_to_Potassium_Cyanide" || poison.value ==
"Moderate_exposure_to_Potassium_Cyanide" || poison.value == "Sodium_Cyanide")
                {
                   treatment.value = cyanide_treat;
                   treatment.instantiate = true;
                }
                break;

        case 3: if (poison.value == "Strychnine")
                {
                    treatment.value = strychnine_treat;
                    treatment.instantiate = true;
                }
                break;

        case 4: if (poison.value == "Tetrodotoxin")
                {
                   treatment.value = tetrodotoxin_treat;
                   treatment.instantiate = true;
                }
                break;

        case 5: if (poison.value == "Severe_exposure_to_Nerve_Agents" || poison.value ==
"Moderate_exposure_to_Nerve_Agents")
                {
                   treatment.value = nerve_agents_treat;
                   treatment.instantiate = true;
                }
                break;
        case 6: if (poison.value == "Moderate_exposure_to_Chlorine_gas" || poison.value ==
"Severe_exposure_to_Chlorine_gas")
                {
                  treatment.value = chlorine_treat;
                  treatment.instantiate = true;
                }
                break;

        case 7: if (poison.value == "Severe_Arsine_gas_exposure" || poison.value ==
"Moderate_Arsine_gas_exposure")
                {
                  treatment.value = arsine_treat;
                  treatment.instantiate = true;
                }
                break;

        case 8: if (poison.value == "Ricin")
                {
                  treatment.value = ricin_treat;
                  treatment.instantiate = true;
                }
                break;

        case 9: if (poison.value == "Colchicine")
                 {
                   treatment.value = colchicine_treat;
                   treatment.instantiate = true;
                 }
                 break;

        case 10: if (poison.value == "Severe_inhalation_of_Sulfur_Mustard" || poison.value ==
"Moderate_inhalation_of_Sulfur_Mustard")
                 {
                   treatment.value = sulfur_mustard_gas_treat;
                   treatment.instantiate = true;
                 }
                 break;
        case 11: if (poison.value == "Rattlesnake")
                 {
                    treatment.value = rattlesnake_treat;
                    treatment.instantiate = true;
                 }
                 break;

        case 12: if (poison.value == "Copperhead/Moccasin")
                 {
                    treatment.value = copperhead_mocassin_treat;
                    treatment.instantiate = true;
                 }
                 break;

        case 13: if (poison.value == "Coral_Snake")
                 {
                    treatment.value = coral_snake_treat;
                    treatment.instantiate = true;
                 }
                 break;

        case 14: if (poison.value == "Black_Widow")
                 {
                    treatment.value = black_widow_treat;
                    treatment.instantiate = true;
                 }
                 break;

        case 15: if (poison.value == "Brown_Recluse")
                 {
                    treatment.value = brown_recluse_treat;
                    treatment.instantiate = true;
                 }
                 break;
    }
}
```