Borislav S. Sabotinov

bss64@txstate.edu | TXST Student ID: A04626934

Abstract

This report examines the design, implementation, and execution of an expert system, which diagnoses vehicle issues and recommends repairs to the user. Report covers how the program employs course concepts and how it makes use of data structures and algorithms.

Vehicle diagnosis and repair recommendations

Project 1 – CS4346: Advanced Artificial Intelligence

Table of Contents

[1 Introduction 4](#_Toc65526321)

[1.1 Purpose 4](#_Toc65526322)

[1.2 System Scope 4](#_Toc65526323)

[1.3 Terms and Definitions 4](#_Toc65526324)

[1.4 Overview 5](#_Toc65526325)

[2 System Description 5](#_Toc65526326)

[2.1 System Perspective 5](#_Toc65526327)

[2.2 System Features 6](#_Toc65526328)

[2.3 Design and Implementation Constraints 6](#_Toc65526329)

[2.4 Assumptions and dependencies 6](#_Toc65526330)

[3 System Design and Specification 6](#_Toc65526331)

[3.1 Decision Tree Diagram 6](#_Toc65526332)

[3.1.1 Failure to Start Diagnosis 7](#_Toc65526333)

[3.1.2 Vehicle Noise Diagnosis 8](#_Toc65526334)

[3.1.3 Vehicle Overheating Diagnosis 10](#_Toc65526335)

[3.1.5 Electrical Issues Diagnosis 11](#_Toc65526336)

[3.1.6 Power Steering Issues Diagnosis 11](#_Toc65526337)

[3.1.7 Tire Issues Diagnosis 12](#_Toc65526338)

[3.1.8 General Diagnostics 13](#_Toc65526339)

[3.2 Variables List for Decision Tree 14](#_Toc65526340)

[3.3 UML Class Diagram 17](#_Toc65526341)

[4 Classes Deep-Dive 18](#_Toc65526342)

[4.1 ClauseItem 18](#_Toc65526343)

[4.1.1 Data Structures Used 18](#_Toc65526344)

[4.2 Statement 18](#_Toc65526345)

[4.2.1 Data Structures Used 19](#_Toc65526346)

[4.3 VariableListItem 19](#_Toc65526347)

[4.3.1 Data Structures Used 20](#_Toc65526348)

[4.4 KnowledgeBase 20](#_Toc65526349)

[4.4.1 Data Structures Used 21](#_Toc65526350)

[4.5 Back Chain 21](#_Toc65526351)

[4.4.1 Data Structures Used 22](#_Toc65526352)

[4.5 ForwardChain 22](#_Toc65526353)

[4.5.1 Data Structures Used 23](#_Toc65526354)

[5 Back Chaining Deep-Dive 23](#_Toc65526355)

[6 Forward Chaining Deep-Dive 23](#_Toc65526356)

[7 Building and Running the Program 23](#_Toc65526357)

[8 Sample Runs 23](#_Toc65526358)

[8.1 Sample Run #1: Diagnosing a Tire Issue 24](#_Toc65526359)

[8.2 Sample Run #2: Diagnosing a Start-up Issue 25](#_Toc65526360)

[8.3 Sample Run #3: Repair Recommendation, Air Filter Replacement due to Noise 26](#_Toc65526361)

[8.4 Sample Run #4: Repair Recommendation, Replacing a Defective Water Pump 27](#_Toc65526362)

[8.5 Sample Run #5: Printing out the Knowledge Base when prompted 29](#_Toc65526363)

[8.6 Sample Run #6: Printing the Help Menu 31](#_Toc65526364)

[8.7 Sample Run #7: Intentionally Running with Faulty Knowledge Base File to Examine Error Handling 31](#_Toc65526365)

[8.8 Sample Run #8: Exhausting the Options – Inconclusive 31](#_Toc65526366)

[References 32](#_Toc65526367)

[Appendix A: Decision Tree Diagram 33](#_Toc65526368)

[Appendix B: Source Code 37](#_Toc65526369)

[ClauseItem.hpp 37](#_Toc65526370)

[ClauseItem.cpp 38](#_Toc65526371)

[Statement.hpp 39](#_Toc65526372)

[Statement.cpp 40](#_Toc65526373)

[VariableListItem.hpp 41](#_Toc65526374)

[VariableListItem.cpp 42](#_Toc65526375)

[KnowledgeBase.hpp 44](#_Toc65526376)

[KnowledgeBase.cpp 45](#_Toc65526377)

[BackChain.hpp 53](#_Toc65526378)

[BackChain.cpp 54](#_Toc65526379)

[ForwardChain.hpp 65](#_Toc65526380)

[ForwardChain.cpp 66](#_Toc65526381)

[*VariablesList.csv* 73](#_Toc65526382)

[KnowledgeBase.txt 74](#_Toc65526383)

[Appendix E: Complete Sample Output for Sample Run #1 76](#_Toc65526384)

List of Tables

[Table 1: Terms and Definitions 5](#_Toc65524675)

[Table 2: Variables List, Descriptions, and Node Mapping by Subsystem 15](#_Toc65524676)

List of Figures

[Figure 1: Beginning of Decision Tree 7](#_Toc65524650)

[Figure 2: Failure to Start Diagnosis Subsystem of Decision Tree 8](#_Toc65524651)

[Figure 3: Vehicle Noise Diagnosis Subsystem of Decision Tree 9](#_Toc65524652)

[Figure 4: Vehicle Overheating Issue Diagnosis Subsystem of Decision Tree 10](#_Toc65524653)

[Figure 5: Power Steering Issue Diagnosis Subsystem of Decision Tree 12](#_Toc65524654)

[Figure 6: Tire Issue Diagnosis Subsystem of Decision Tree 13](#_Toc65524655)

[Figure 7: General Diagnosis Subsystem of Decision Tree 14](#_Toc65524656)

[Figure 8: UML Class Diagram for Vehicle Diagnosis and Repair System 17](#_Toc65524657)

[Figure 9: ClauseItem UML Class Node 18](#_Toc65524658)

[Figure 10: Statement UML Class Node 19](#_Toc65524659)

[Figure 11: VariableListItem UML Class Node 20](#_Toc65524660)

[Figure 12: KnowledgeBase UML Class Node 21](#_Toc65524661)

[Figure 13: BackChain UML Class Node 22](#_Toc65524662)

[Figure 14: ForwardChain UML Class Node 23](#_Toc65524663)

[Figure 15: Sample Run #1 - Diagnosing Tire Issue 25](#_Toc65524664)

[Figure 16: Sample Run #2 – Startup Issue 26](#_Toc65524665)

[Figure 17: Sample Run #3 – Repair Recommendation, Change Air Filter 27](#_Toc65524666)

[Figure 18: Sample Run #4 - Replace Defective Water Pump 28](#_Toc65524667)

[Figure 19: Sample Run #5 - Print Knowledge Base to Console 30](#_Toc65524668)

[Figure 20: Sample Run #6 - Print the Help Menu 31](#_Toc65524669)

[Figure 21: Sample Run #7 - Intentionally Test Error Handling 31](#_Toc65524670)

[Figure 22: Sample Run #8 - Inconclusive Result 32](#_Toc65524671)

[Figure 23: Full Decision Tree Diagram 33](#_Toc65524672)

[Figure 24: Electrical Issue Diagnosis Subsystem Part 1 35](#_Toc65524673)

[Figure 25: Electrical Issue Subsystem Diagnosis Part 2 36](#_Toc65524674)

# 1 Introduction

## 1.1 Purpose

The intent of this report is to demonstrate a thorough understanding of class concepts and provide a detailed explanation of the project one vehicle diagnosis application. This report will attempt to exhaustively cover every aspect of the design, implementation, execution, and analysis of the program.

## 1.2 System Scope

The application for project one is an intelligent expert system for diagnosing vehicle issues and recommending repairs to the user. It employs Backward Chaining to diagnose the issue and Forward Chaining to identify a repair to recommend.

The system uses as much realistic, real-world vehicle and mechanical data as possible. The system is not intended to serve as an actual diagnosis and repair system for real-world applications and it is primarily intended to demonstrate a thorough understanding of advanced artificial intelligence concepts for CS5346.

The system shall:

* Read in data from external file(s).
* Prompt users for data if the required values are not available.
* Provide users the ability to print out the entire knowledge base of the expert system.
* Attempt to be robust and handle error cases, such as malformed data or missing files.

The system is designed and built to compile and run on Texas State Linux hosts. It was additionally tested on Windows – provided certain prerequisites are met, it can execute on the Windows OS as well.

## 1.3 Terms and Definitions

Table 1: Terms and Definitions

|  |  |
| --- | --- |
| **Term** | **Definition** |
| The system | “The system” in this report shall refer to the application designed and implemented for project one, an intelligent expert system for vehicle diagnostics and repair recommendations. |
| KB | Knowledge Base. KB throughout the report shall refer to the knowledge base, containing logic for the vehicle diagnosis and repair system. |

## 1.4 Overview

The table of contents shows where each section is located and contains a list of tables and figures used throughout the report for ease of reference.

**Section 1** covers general, introductory information about the system created for this project. It shows terms and definitions used in this report.

**Section 2** describes the system at a high level. It defines the intended audience and the types of users served by this system, a list of features. Any constraints encountered, or assumptions made, during design and implementation are also outlined here.

**Section 3** dives into the system design in greater detail. We examine the decision tree diagram for the expert system and all its subsystems. We look at the variables list we derive from this decision tree. Finally, the system’s UML class diagram is presented.

**Section 4** dives into the source code and outlines each class, as well as any noteworthy data structures or algorithms employed.

**Section 5** covers back chaining and how it is used to diagnose vehicle issues in this system.

**Section 6** covers forward chaining and how it is used to recommend repairs in this system.

**Section 7** shows how to build and run the program, providing a complete listing of all available commands.

**Section 8** comprehensively covers the system by exercising it against all unique available options. In addition to the minimum required three sample runs, four additional runs are performed. Two issues diagnosis, two repair recommendations, printing the knowledge base, printing the help menu, and one intentional error are all exercised. Additionally, we consider inconclusive results, where the user exhausts the available options and the system does not contain a repair recommendation. A complete text output is presented in Appendix E.

**The appendices** provide large images of all diagrams for readability, as well as the complete source code. Any diagram too large to be readable in its entirety is dissected into sections and each section is presented separately, expanded to fill the screen.

# 2 System Description

## 2.1 System Perspective

While the system is designed with a real end user in mind – someone who may need to diagnose and repair their car – the system is also tailored for computer science students and faculty. The output is verbose and intentionally signals what is going on behind the scenes during execution. Detailed sample runs may be found in section 8.

## 2.2 System Features

D

## 2.3 Design and Implementation Constraints

The system was built with Texas State Linux servers in mind. While it may operate properly in other environments (e.g., Windows-10, Ubunut, SLES, etc.) it is guaranteed to run on either the Eros or Zeus servers. Section 7 outlines detailed build and execution instructions, which are also available in a shortened format in the README.md file submitted alongside the source code.

## 2.4 Assumptions and dependencies

D

# 3 System Design and Specification

## 3.1 Decision Tree Diagram

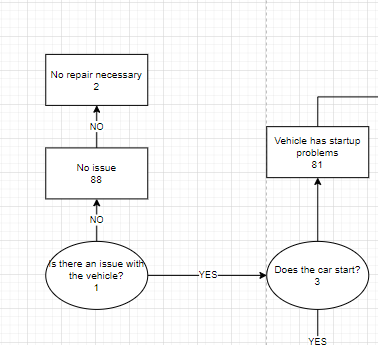


Figure 1: Beginning of Decision Tree

### 3.1.1 Failure to Start Diagnosis

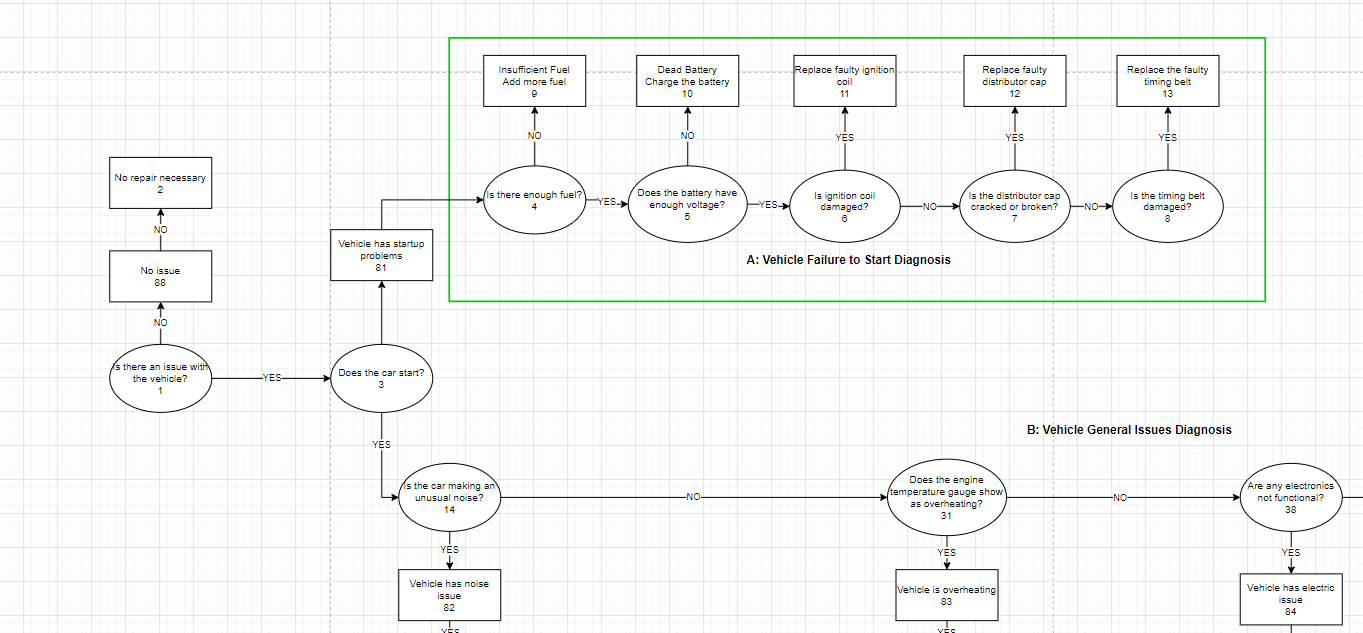


Figure 2: Failure to Start Diagnosis Subsystem of Decision Tree

### 3.1.2 Vehicle Noise Diagnosis

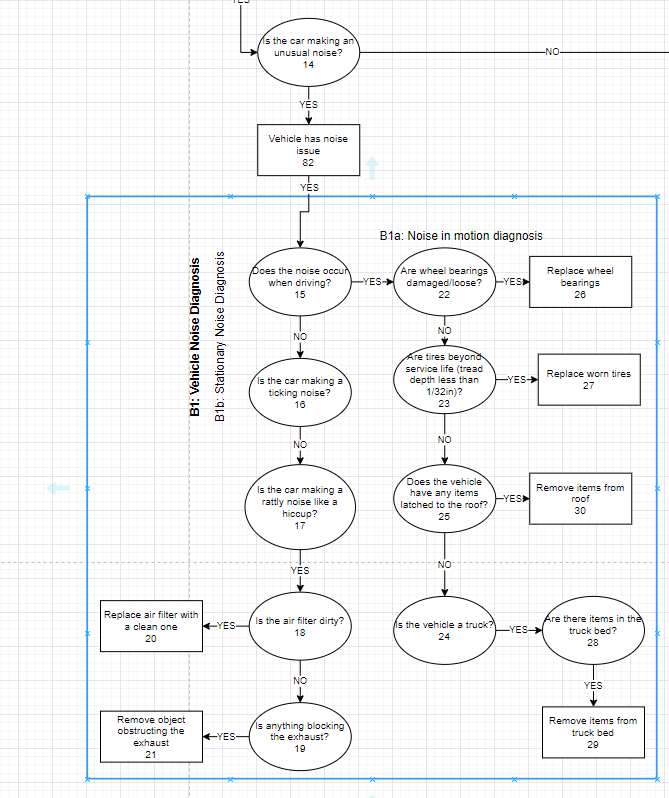


Figure 3: Vehicle Noise Diagnosis Subsystem of Decision Tree

### 3.1.3 Vehicle Overheating Diagnosis

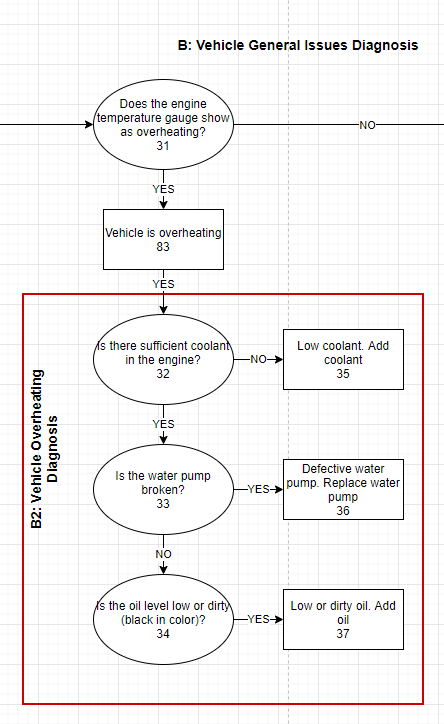


Figure 4: Vehicle Overheating Issue Diagnosis Subsystem of Decision Tree

### 3.1.5 Electrical Issues Diagnosis

The diagram for this subsystem is too large to be clearly displayed here. For readability and ease of access, it is included at the end of this report under Appendix A.

### 3.1.6 Power Steering Issues Diagnosis

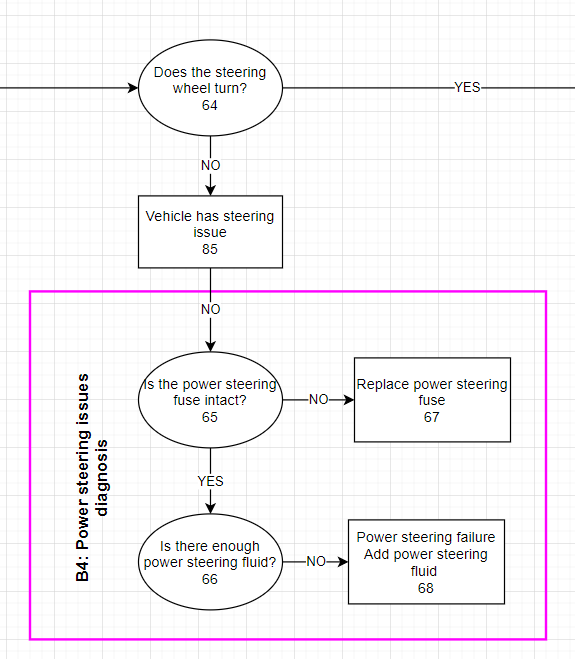


Figure 5: Power Steering Issue Diagnosis Subsystem of Decision Tree

### 3.1.7 Tire Issues Diagnosis

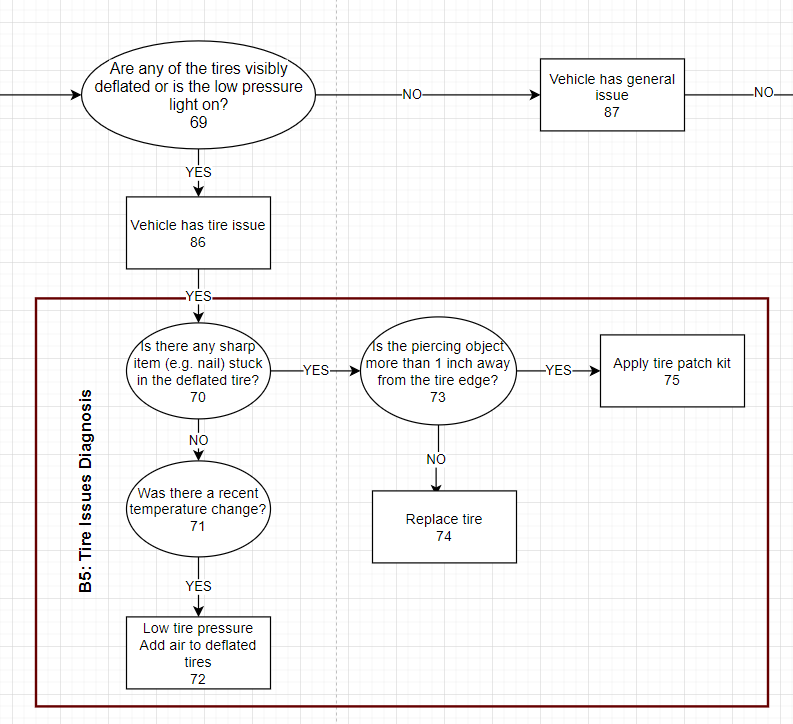


Figure 6: Tire Issue Diagnosis Subsystem of Decision Tree

### 3.1.8 General Diagnostics

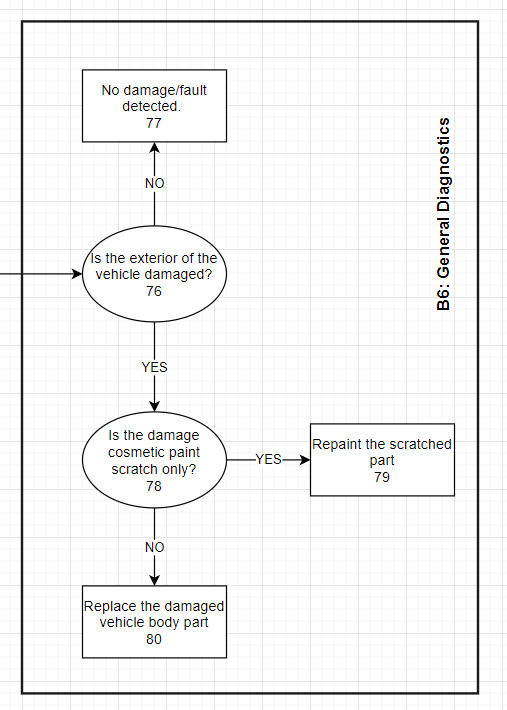


Figure 7: General Diagnosis Subsystem of Decision Tree

## 3.2 Variables List for Decision Tree

Table 2: Variables List, Descriptions, and Node Mapping by Subsystem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable Name** | **Type** | **Description / Meaning** | **Node(s)** | **Subsystem** |
| has\_issue | bool | Is there an issue with the vehicle? (y/n) | 1 | Initial Position |
| is\_starting | bool | Does the car start? (y/n) | 3 | Vehicle Failure to Start Diagnosis |
| has\_fuel | bool | Is there enough fuel? (y/n) | 4 | Vehicle Failure to Start Diagnosis |
| has\_voltage | bool | Does teh battery have enough voltage? (y/n) | 5 | Vehicle Failure to Start Diagnosis |
| is\_ignition\_coil\_damaged | bool | Is the ignition coil in good condition? (y/n) | 6 | Vehicle Failure to Start Diagnosis |
| is\_distributor\_cap\_damaged | bool | Is the distributor cap cracked or broken? (y/n) | 7 | Vehicle Failure to Start Diagnosis |
| is\_timing\_belt\_damaged | bool | Is the timing belt damaged? (y/n) | 8 | Vehicle Failure to Start Diagnosis |
| is\_making\_noise | bool | Is the car making an unusual noise? (y/n) | 14 | General - Vehicle Noise Diagnosis |
| is\_noisy\_while\_driving | bool | Does the noise occur when driving? (y/n) | 15 | General - Vehicle Noise Diagnosis |
| is\_ticking\_noise | bool | Is the car making a ticking noise? (y/n) | 16 | General - Vehicle Noise Diagnosis |
| is\_hiccup\_noise | bool | Is the car making a rattly noise like a hiccup? (y/n) | 17 | General - Vehicle Noise Diagnosis |
| is\_air\_filter\_dirty | bool | Is the air filter dirty? (y/n) | 18 | General - Vehicle Noise Diagnosis |
| is\_exhaust\_blocked | bool | Is anything blocking the exhaust? (y/n) | 19 | General - Vehicle Noise Diagnosis |
| are\_wheel\_bearings\_damaged | bool | Are wheel bearings damaged/loose? (y/n) | 22 | General - Vehicle Noise Diagnosis |
| are\_tires\_bald | bool | Are tires beyond service life (tread depth less than 1/32in)? (y/n) | 23 | General - Vehicle Noise Diagnosis |
| is\_truck | bool | Is the vehicle a truck? (y/n) | 24 | General - Vehicle Noise Diagnosis |
| has\_items\_in\_truck\_bed | bool | Are there items in the truck bed? (y/n) | 28 | General - Vehicle Noise Diagnosis |
| has\_items\_on\_roof | bool | Does the vehicle have any items latched to the roof? (y/n) | 25 | General - Vehicle Noise Diagnosis |
| is\_overheating | bool | Does the temperature gauge show as overheating? (y/n) | 31 | General - Vehicle Overheating Diagnosis |
| has\_coolant | bool | Is there sufficient coolant in the engine? (y/n) | 32 | General - Vehicle Overheating Diagnosis |
| is\_water\_pump\_broken | bool | Is the water pump broken? (y/n) | 33 | General - Vehicle Overheating Diagnosis |
| is\_oil\_low\_or\_dirty | bool | Is the oil level low or dirty (black in color)? (y/n) | 34 | General - Vehicle Overheating Diagnosis |
| has\_nonfunctional\_electronics | bool | Are any electronics not functional? (y/n) | 38 | General - Electrical Issues Diagnosis |
| does\_ac\_power\_on | bool | Does the AC power on? (y/n) | 39 | General - Electrical Issues Diagnosis |
| does\_ac\_blow\_cold | bool | Does the AC blow cold? (y/n) | 40 | General - Electrical Issues Diagnosis |
| has\_nonfunctional\_headlights | bool | Are any headlights or lights not turning on? (y/n) | 41 | General - Electrical Issues Diagnosis |
| is\_ac\_fuse\_intact | bool | Is the AC fuse intact? (y/n) | 44 | General - Electrical Issues Diagnosis |
| are\_ac\_wires\_connected | bool | Are AC ground and power wires connected? (y/n) | 53 | General - Electrical Issues Diagnosis |
| are\_therm\_settings\_correct | bool | Are the thermostat settings correct? (y/n) | 47 | General - Electrical Issues Diagnosis |
| is\_evaporator\_coil\_frozen | bool | Is the evaporator coil frozen? (y/n) | 56 | General - Electrical Issues Diagnosis |
| is\_air\_filter\_dirty | bool | Is the air filter dirty? (y/n) | 62 | General - Electrical Issues Diagnosis |
| is\_nonfunct\_light\_fuse\_intact | bool | Is the respective non-working light fuse intact? (y/n) | 49 | General - Electrical Issues Diagnosis |
| are\_nonfunct\_light\_wires\_conn | bool | Are ground and power wires connected? (y/n) | 58 | General - Electrical Issues Diagnosis |
| has\_burning\_plastic\_smell | bool | Do you smell any burning plastic / electric insulation? (y/n) | 42 | General - Electrical Issues Diagnosis |
| is\_radio\_working | bool | Does the radio power on? (y/n) | 43 | General - Electrical Issues Diagnosis |
| is\_radio\_fuse\_intact | bool | Is the radio fuse intact? (y/n) | 51 | General - Electrical Issues Diagnosis |
| are\_radio\_wires\_connected | bool | Are radio ground and power wires connected? (y/n) | 59 | General - Electrical Issues Diagnosis |
| does\_wheel\_turn | bool | Does the wheel turn stiffly or fail to turn? (y/n) | 64 | General - Power Steering Issues Diagnosis |
| is\_power\_steering\_fuse\_intact | bool | Is the power steering fuse intact? (y/n) | 65 | General - Power Steering Issues Diagnosis |
| has\_power\_steering\_fluid | bool | Is there enough power steering fluid? (y/n) | 66 | General - Power Steering Issues Diagnosis |
| are\_tires\_deflated | bool | Are any of the tires visibly deflated or is the low pressure light on? (y/n) | 69 | General - Tire Issues Diagnosis |
| has\_piercing\_object | bool | Is there a sharp item (e.g., nail) stuck in the deflated tire? (y/n) | 70 | General - Tire Issues Diagnosis |
| is\_obj\_inch\_away\_from\_edge | bool | Is the piercing object more than 1in away from the tire edge? (y/n) | 73 | General - Tire Issues Diagnosis |
| is\_recent\_temp\_change | bool | Was there a recent temperature change? (y/n) | 71 | General - Tire Issues Diagnosis |
| is\_exterior\_damaged | bool | Is the exterior of the vehicle damaged? (y/n) | 76 | General Issue Diagnosis |
| is\_damage\_cosmetic | bool | Is the damage cosmetic paint scratch only? (y/n) | 78 | General Issue Diagnosis |
| repair | string | What repair should be recommended to the user? These are the decision nodes. | 2, 9, 10, 11, 12, 13, 26, 27, 30, 20, 21, 29, 35, 36, 37, 45, 54, 46, 55, 61, 48, 57, 50, 52, 60, 63, 67, 68, 75, 74, 72, 77, 79, 80 | Conclusion variable |
| issue | string | What is the general vehicle issue detected based on user's response? | 81,82,83,84,85,86,87,88 | Conclusion variable, also used as intermediate variable in premise list |

## 3.3 UML Class Diagram

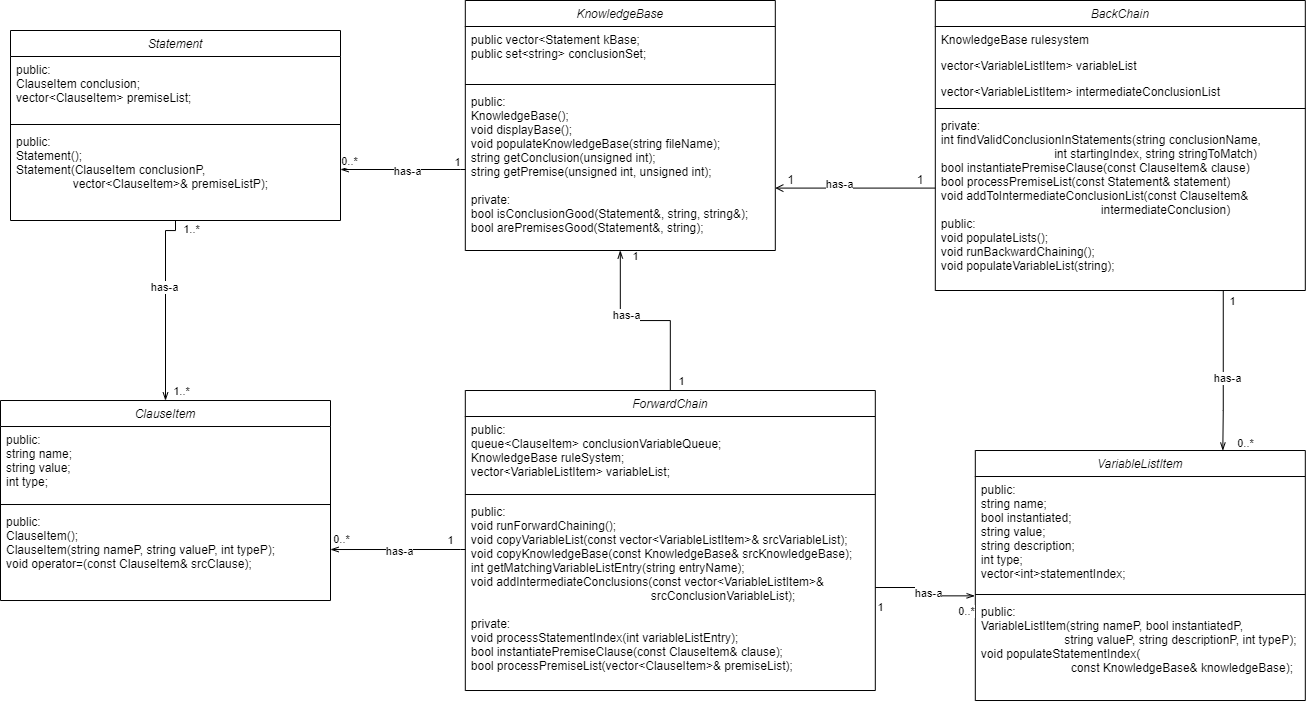


Figure 8: UML Class Diagram for Vehicle Diagnosis and Repair System

# 4 Classes Deep-Dive

## 4.1 ClauseItem

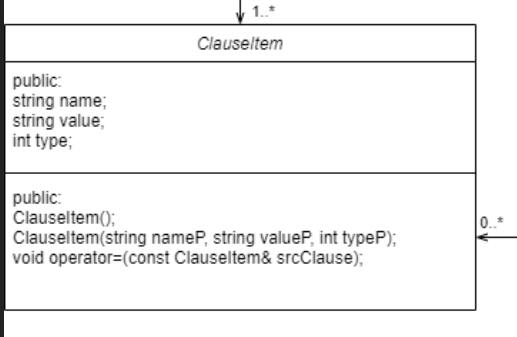


Figure 9: ClauseItem UML Class Node

### 4.1.1 Data Structures Used

D

## 4.2 Statement

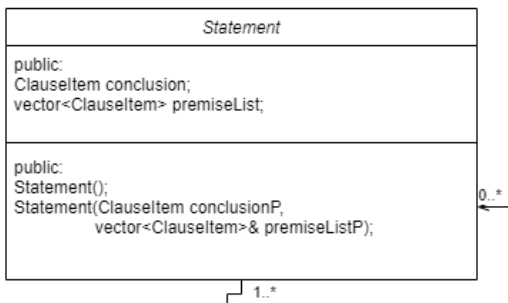


Figure 10: Statement UML Class Node

### 4.2.1 Data Structures Used

D

## 4.3 VariableListItem

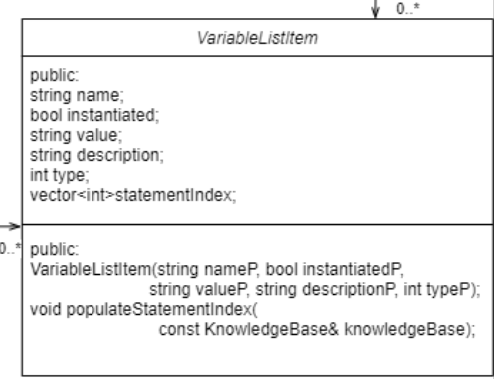


Figure 11: VariableListItem UML Class Node

### 4.3.1 Data Structures Used

D

## 4.4 KnowledgeBase

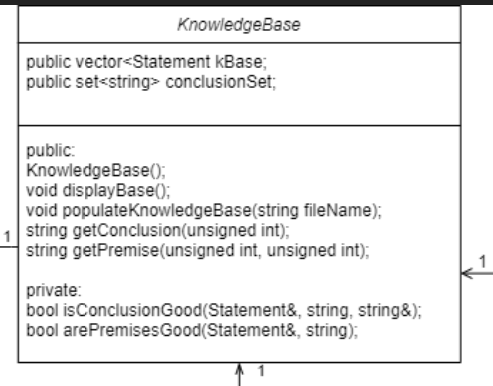


Figure 12: KnowledgeBase UML Class Node

### 4.4.1 Data Structures Used

D

## 4.5 Back Chain

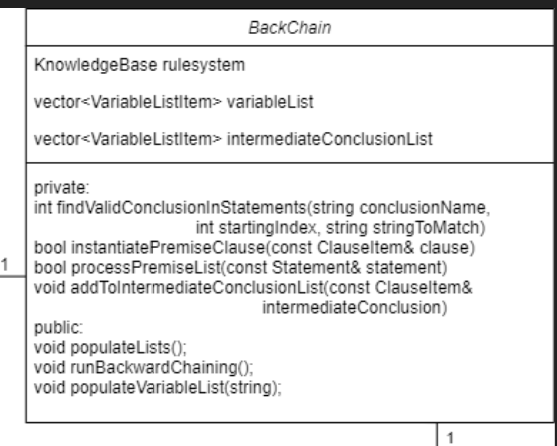


Figure 13: BackChain UML Class Node

D

### 4.4.1 Data Structures Used

D

## 4.5 ForwardChain

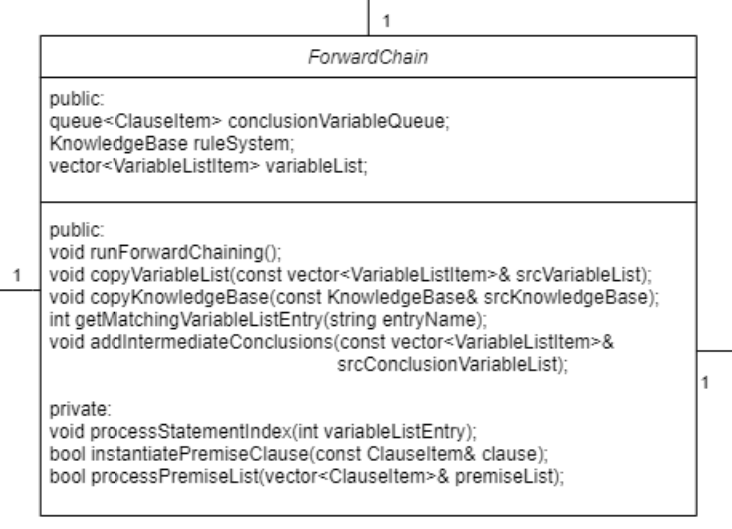


Figure 14: ForwardChain UML Class Node

### 4.5.1 Data Structures Used

D

# 5 Back Chaining Deep-Dive

D

# 6 Forward Chaining Deep-Dive

D

# 7 Building and Running the Program

D

8 Sample Runs

The program prints the same content to the console at the beginning of each run. A complete listing of this output may be found at the end of this report, under “**Appendix E: Complete Sample Output for Sample Run #1**.” The program always displays a welcome message to the user, parses the KB text file and prints contents to the console as it does so, and pauses. After the user hits Enter, it asks if the user wants to display the KB in human readable output. Tee is a command which reads our standard I/O and writes it to both standard output and a text file, in our case a log.

Complete output of this program was obtained on eros.cs.txstate.edu. Program was invoked normally but standard output was piped to tee. For a complete listing of this and other commands, refer to Section 7.

This section aims to comprehensively cover all options of the implemented system. Not every available repair recommendation will be shown here, for example, but every uniquely available option will be exercised and presented.

8.1 Sample Run #1: Diagnosing a Tire Issue

In this sample run we impersonate a user experiencing a vehicle that is not operating properly. Our hypothetical user does not know it, but the vehicle has a flat tire. Through a careful application of back chaining, we ask the user questions (when appropriate) to determine what issue their vehicle is experiencing. They are not able to go down the road properly, so they leverage our system for diagnostics.

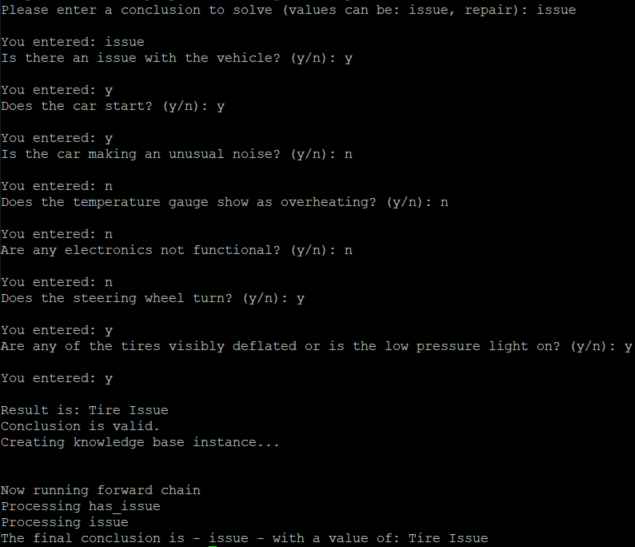


Figure 15: Sample Run #1 - Diagnosing Tire Issue

## 8.2 Sample Run #2: Diagnosing a Start-up Issue

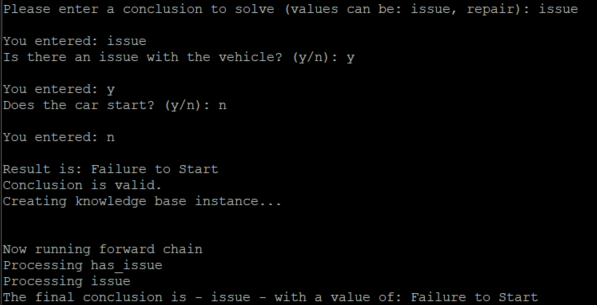


Figure 16: Sample Run #2 – Startup Issue

## 8.3 Sample Run #3: Repair Recommendation, Air Filter Replacement due to Noise

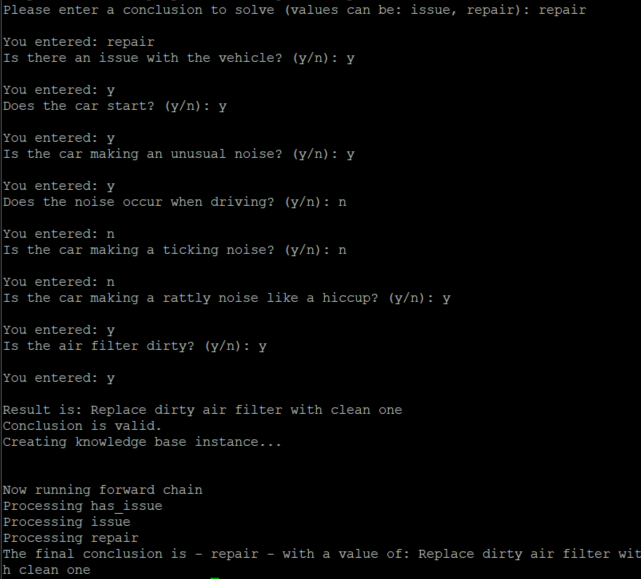


Figure 17: Sample Run #3 – Repair Recommendation, Change Air Filter

## 8.4 Sample Run #4: Repair Recommendation, Replacing a Defective Water Pump

D

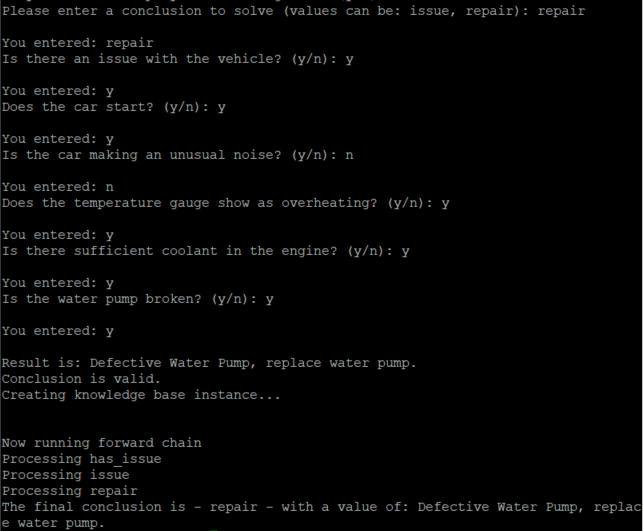


Figure 18: Sample Run #4 - Replace Defective Water Pump

## 8.5 Sample Run #5: Printing out the Knowledge Base when prompted

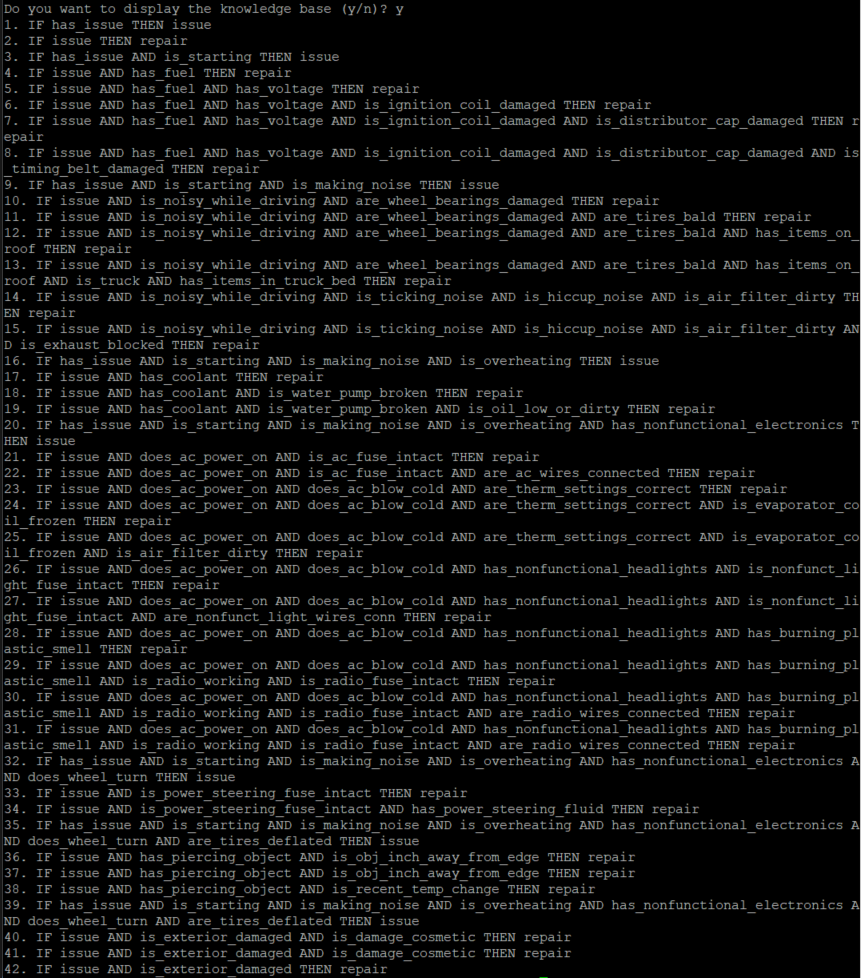


Figure 19: Sample Run #5 - Print Knowledge Base to Console

## 8.6 Sample Run #6: Printing the Help Menu

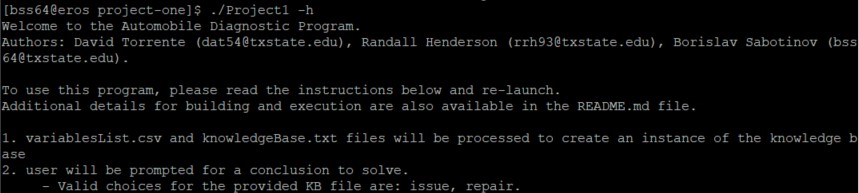


Figure 20: Sample Run #6 - Print the Help Menu

## 8.7 Sample Run #7: Intentionally Running with Faulty Knowledge Base File to Examine Error Handling

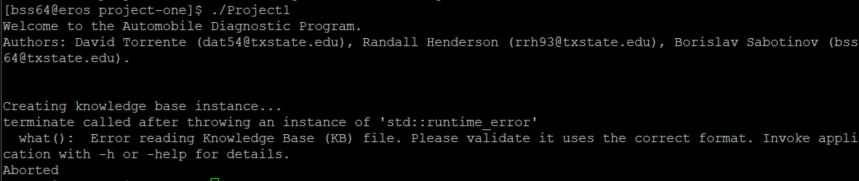


Figure 21: Sample Run #7 - Intentionally Test Error Handling

## 8.8 Sample Run #8: Exhausting the Options – Inconclusive

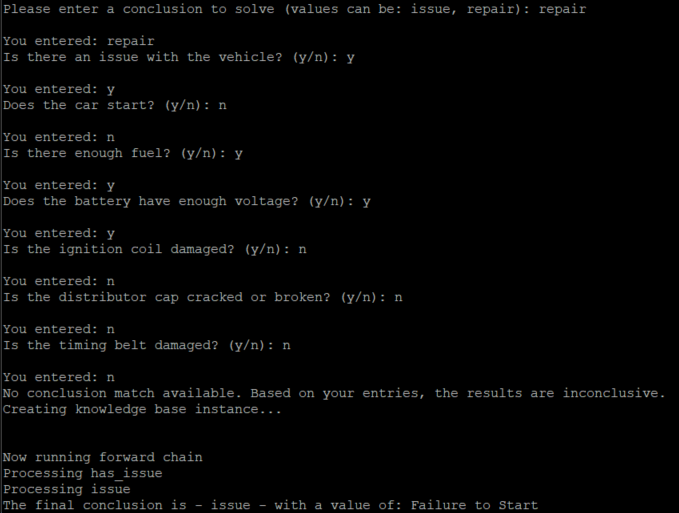


Figure 22: Sample Run #8 - Inconclusive Result

# References

D

# Appendix A: Decision Tree Diagram

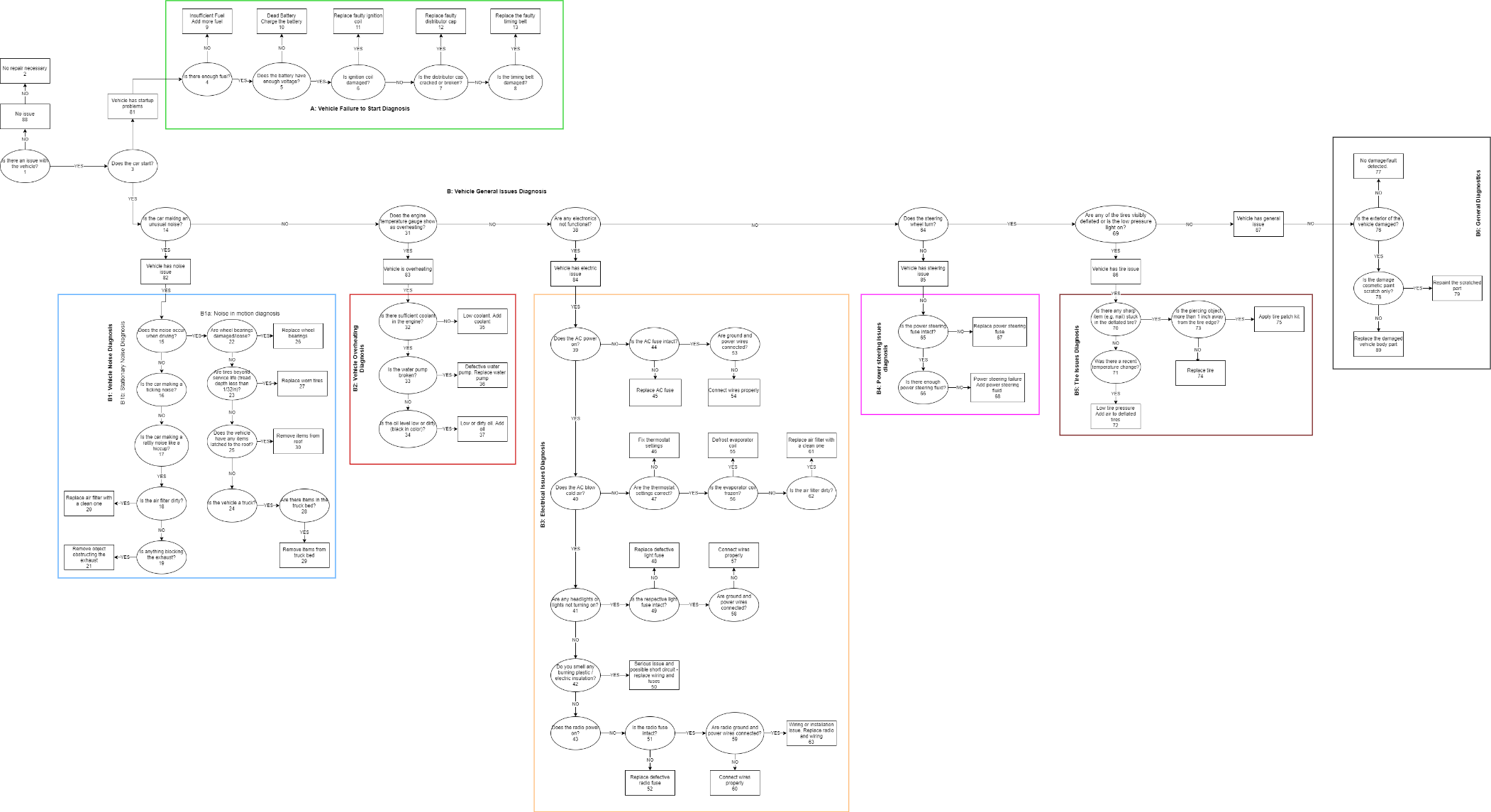


Figure 23: Full Decision Tree Diagram

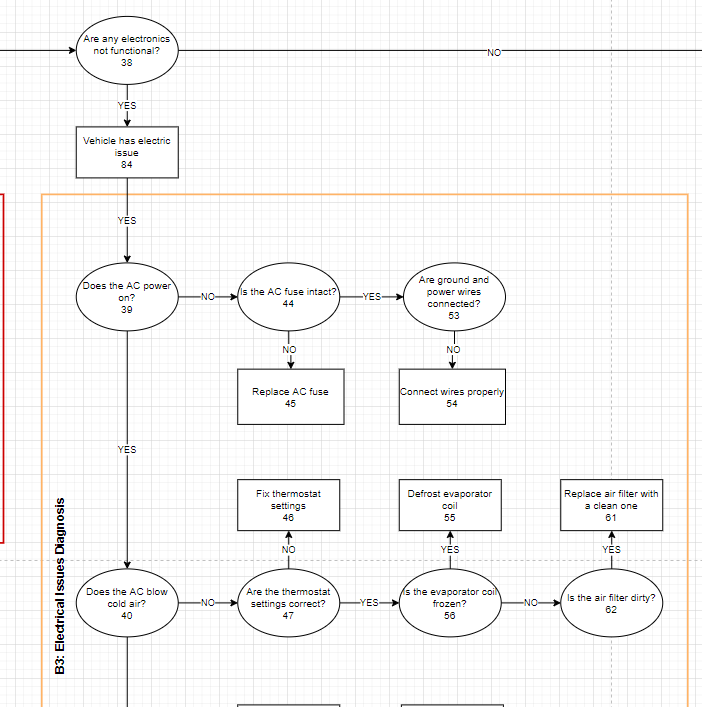


Figure 24: Electrical Issue Diagnosis Subsystem Part 1

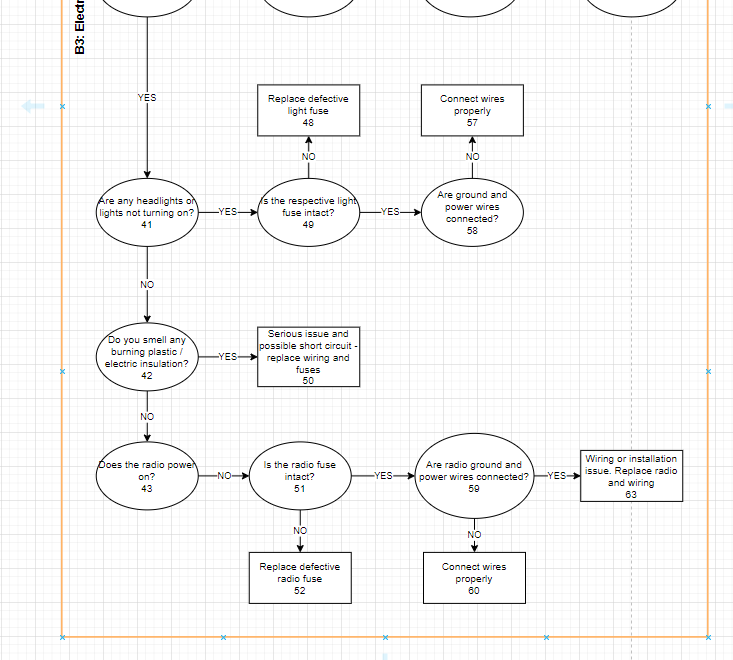


Figure 25: Electrical Issue Subsystem Diagnosis Part 2

# Appendix B: Source Code

## ClauseItem.hpp



## ClauseItem.cpp



## Statement.hpp



## Statement.cpp

1. #include "Statement.hpp"
2. /\*\*
3. \* Constructor | Statement | Statement
4. \*
5. \* Summary: Instantiates a default statement with NULL values. Note that if the
6. \*          knowledge base contains the value of NULL, some care must be taken.
7. \*          However, a guard is in place as each index begins at element 1.
8. \*
9. \*/
10. Statement::Statement()
11. {
12. conclusion.name = "NULL";
13. conclusion.value = "NULL";
14. conclusion.type = STRING;
15. premiseList.push\_back(ClauseItem());
16. }
17. /\*\*
18. \* Constructor | ClauseItem | ClauseItem
19. \*
20. \* Summary: Instantiates a statement item with the specified values. Statements
21. \*          follow the pattern of premiseList then conclusion.
22. \*          All types are currently set to type string.
23. \*
24. \* @param ClauseItem conclusionP: The conclusion portion of the statement
25. \* @param std::vector<ClauseItem>& premiseListP> :    A vector of premise
26. \*                                  clauses to be used in the statement.
27. \*
28. \*/
29. Statement::Statement(ClauseItem conclusionP, std::vector<ClauseItem>& premiseListP)
30. {
31. conclusion = conclusionP;
32. premiseList = premiseListP;
33. }

## VariableListItem.hpp

1. #ifndef VARIABLE\_LIST\_ITEM\_H
2. #define VARIABLE\_LIST\_ITEM\_H
3. #include <string>
4. #include <vector>
5. #include "KnowledgeBase.hpp"
6. /\*\*
7. \* A list of variables and their associated values that must accompany the knowledgeBase.txt file
8. \* The variablesList.csv file contains a comma delimited list of variables and their contents to display.
9. \* For example:
10. \* {@code is\_distributor\_cap\_damaged,Is the distributor cap cracked or broken? (y/n),STRING}
11. \*
12. \* first value is the variable name, next is the description that will be displayed to the user if we need to obtain their
13. \* input about this variable, and last is the type. Currently all variables are of string type, as we read them in via text file.
14. \*/
15. class VariableListItem
16. {
17. public:
18. VariableListItem(std::string nameP, bool instantiatedP, std::string valueP, std::string descriptionP, int typeP);
19. void populateStatementIndex(const KnowledgeBase& knowledgeBase);
21. std::string name;
22. bool instantiated;
23. std::string value;
24. std::string description;
25. int type;
26. std::vector<int>statementIndex;
27. };
28. #endif // !VARIABLE\_LIST\_ITEM\_H

## VariableListItem.cpp

1. #include <vector>
2. #include <string>
3. #include "VariableListItem.hpp"
4. /\*\*
5. \* Constructor | VariableListItem | VariableListItem
6. \*
7. \* Summary: Instantiates a variable item with the specified values. Note that no
8. \*          values are passed in to the statement index. This is because
9. \*          this index is only used for forward chaining and is populated
10. \*          at that time.
11. \*
12. \* @param string nameP:   Name of the clause variable .
13. \* @param bool instantiatedP: If it has been given a value. Will start as false.
14. \* @param string valueP:  Initial value of the clause.
15. \* @param string descriptionP: A prompt to display when assigning a value.
16. \* @param int typeP:  The type of clause, usually set to STRING.
17. \*
18. \*/
19. VariableListItem::VariableListItem(std::string nameP, bool instantiatedP, std::string valueP, std::string descriptionP, int typeP)
20. {
21. name = nameP;
22. instantiated = instantiatedP;
23. value = valueP;
24. description = descriptionP;
25. type = typeP;
26. statementIndex.push\_back(-1); // To keep in line with the other indexes. But use -1 to note bad values
27. }
28. /\*\*
29. \* CMember function | VariableListItem | populateAtatementIndex
30. \*
31. \* Summary: Creates an inverted index to allow for quicker searching when using
32. \*          forward chaining. Also helps prevent the need to double search or
33. \*          process needless entries.
34. \*
35. \* @param const KnowledgeBase& knowledgeBase: Used to find entries that match
36. \*          the name for this variable list entry.
37. \*/
38. void VariableListItem::populateStatementIndex(const KnowledgeBase& knowledgeBase)
39. {
40. bool isFound = false;
41. for (int statementIter = 1; statementIter < knowledgeBase.kBase.size(); statementIter++)
42. {
43. isFound = false;
44. for (int premiseIter = 1; (!isFound && premiseIter < knowledgeBase.kBase.at(statementIter).premiseList.size()); premiseIter++)
45. {
46. if (name == knowledgeBase.kBase.at(statementIter).premiseList.at(premiseIter).name)
47. {
48. isFound = true;
49. statementIndex.push\_back(statementIter);
50. }
51. }
52. }
53. }

## KnowledgeBase.hpp



## KnowledgeBase.cpp

1. #include <iostream>
2. #include <fstream>
3. #include <stdexcept>
4. #include "KnowledgeBase.hpp"
5. KnowledgeBase::KnowledgeBase()
6. {
7. std::cout << "\nCreating knowledge base instance..." << std::endl;
8. }
9. /\*\*
10. \* populateKnowledgeBase - reads in a data file and creates the knowledge base (KB)
11. \* accordingly, provided the right format is used.
12. \* File format example
13. \* issue = Failure to Start ^ has\_fuel = n : repair = Insufficient Fuel, Add more fuel.
14. \*
15. \* = is used to separate variable and value
16. \* ^ is logical AND
17. \* : separates clause and conclusion
18. \*
19. \* The above example should be read as follows. If issue is equal to failure to start, and has fuel is false
20. \* then repair conclusion equals "Insufficient Fuel, Add more fuel."
21. \*
22. \* @param string fileName - the name of the file containing the knowledge base. knowledgeBase.txt
23. \*
24. \* @return none
25. \*
26. \*/
27. void KnowledgeBase::populateKnowledgeBase(std::string fileName)
28. {
29. int total\_good = 0, total\_bad = 0;
30. std::string inputBuffer;
31. std::ifstream inputFile;
32. inputFile.open(fileName);
33. if (inputFile)
34. {
35. while (getline(inputFile, inputBuffer))  // while read was successfull
36. {
37. std::cout << "Processing:  " << inputBuffer << std::endl;
38. if (inputBuffer.size() > 0)
39. {
40. Statement lList;
41. std::string listPremise;
42. int indicatorLocation;
43. if (isConclusionGood(lList, inputBuffer, listPremise))
44. {
45. if (arePremisesGood(lList, listPremise))
46. {
47. std::cout << "Conclusion and premise(s) are good => List Updated\n";
48. kBase.push\_back(lList);
49. ++total\_good;
50. }
51. else
52. {
53. std::cout << "Premise List is formatted incorrectly.  List NOT updated\n";
54. ++total\_bad;
55. }
56. }
57. else
58. {
59. std::cout << "Conclusion is formatted incorrectly.  List NOT updated\n";
60. ++total\_bad;
61. }
62. }
63. std::cout << std::endl;
64. }
65. }
66. else {
67. throw std::runtime\_error("Error reading Knowledge Base (KB) file. Please validate it uses the correct format. Invoke application with -h or -help for details.");
68. }
69. std::cout << "\nKnowledge Base finished Loading.\n" << total\_good << " items were loaded into the KnowledgeBase\n";
70. if ( total\_bad > 0 )
71. std::cerr << "\nWARNING! " << total\_bad << " malfromed item(s) were not loaded into the Knowledge Base. " <<
72. "\nPlease check output above for items not loaded and inspect data file.\n";
73. std::cout << "<CR/Enter> to continue  ";
74. std::cin.ignore();
75. }
76. /\*\*
77. \* isConclusionGood - helper function to check if the conclusion in the premise is valid and trim any white spaces
78. \*
79. \* @param Statement& lList - a statement containing the premise list and the conclusion it leads to
80. \* @param std::string iBuffer - the entire statement (premise and conclusions)
81. \* @param std::string& listPremise - a list of the premises (left side of expression)
82. \*
83. \* @return bool - if the conclusion is valid, true. Otherwise false.
84. \*/
85. bool KnowledgeBase::isConclusionGood(Statement& lList, std::string iBuffer, std::string& listPremise)
86. {
87. ClauseItem nClause;
88. std::string listConclusion;
89. int indicatorLocation = iBuffer.find(':', 0);  // THEN symbol
90. if (indicatorLocation == -1)
91. return false;
92. listPremise = iBuffer.substr(0, indicatorLocation);
93. listConclusion = iBuffer.substr(indicatorLocation + 1, iBuffer.size());
94. indicatorLocation = listConclusion.find('=', 0);  // Assignment Symbol
96. if (indicatorLocation == -1)
97. return false;
99. lList.conclusion.name = listConclusion.substr(0, indicatorLocation);
101. // trim white spaces in conclusion name, first front, then back
102. if (lList.conclusion.name.front() == ' ')
103. {
104. lList.conclusion.name = lList.conclusion.name.substr(1);
105. }
106. if (lList.conclusion.name.back() == ' ')
107. {
108. lList.conclusion.name = lList.conclusion.name.substr(0, lList.conclusion.name.size() - 1);
109. }
110. lList.conclusion.value = listConclusion.substr(indicatorLocation + 1, listConclusion.size());
111. // trim white spaces in conclusion value, first front, then back
112. if (lList.conclusion.value.front() == ' ')
113. {
114. lList.conclusion.value = lList.conclusion.value.substr(1);
115. }
116. if (lList.conclusion.value.back() == ' ')
117. {
118. lList.conclusion.value = lList.conclusion.value.substr(0, lList.conclusion.value.size() - 1);
119. }
120. lList.conclusion.type = STRING;
121. conclusionSet.insert(lList.conclusion.name); // add conclusion to set, maintaining unique list of conclusions
122. std::cout << "Conclusion is good; ";
123. return true;
124. }
125. /\*\*
126. \* isConclusionGood - helper function to check if the conclusion in the premise is valid and trim any white spaces
127. \*
128. \* @param Statement& lList - a statement containing the premise list and the conclusion it leads to
129. \* @param std::string& listPremise - a list of the premises (left side of expression)
130. \*
131. \* @return bool - if the premise is valid, true. Otherwise false.
132. \*/
133. bool KnowledgeBase::arePremisesGood(Statement& lList, std::string listPremise)
134. {
135. ClauseItem nClause;
136. std::string listRight,
137. listLeft,
138. tmpList;
139. do
140. {
141. int andLocation = listPremise.find('^', 0),
142. equalsLocation;
143. if (andLocation == -1 && listPremise.size() == 0) // no premise
144. return false;
145. if (andLocation == -1)
146. andLocation = listPremise.size();
147. tmpList = listPremise.substr(0, andLocation);
148. listPremise.erase(0, andLocation + 1);
149. equalsLocation = tmpList.find('=', 0);  // assignment Symbol
150. if (equalsLocation == -1)
151. return false;
152. listRight = tmpList.substr(0, equalsLocation);
153. listLeft = tmpList.substr(equalsLocation + 1, tmpList.size());
154. if (listLeft.size() == 0 || listRight.size() == 0)  // missing something
155. return false;
156. // trim white space from clause name, first front then back
157. nClause.name = listRight;
158. if (nClause.name.front() == ' ')
159. {
160. nClause.name = nClause.name.substr(1);
161. }
162. if (nClause.name.back() == ' ')
163. {
164. nClause.name = nClause.name.substr(0, nClause.name.size() - 1);
165. }
166. // trim white space from clause value, first front then back
167. nClause.value = listLeft;
168. if (nClause.value.front() == ' ')
169. {
170. nClause.value = nClause.value.substr(1);
171. }
172. if (nClause.value.back() == ' ')
173. {
174. nClause.value = nClause.value.substr(0, nClause.value.size() - 1);
175. }
176. nClause.type = STRING;
177. lList.premiseList.push\_back(nClause);
178. } while (listPremise.size() != 0);
179. std::cout << "Premise list is good!  " << lList.premiseList.size() - 1 << " premise(s) loaded\n";
180. return true;
181. }
182. /\*\*
183. \* displayBase - prints out the imported knowledge base to the screen
184. \* We iterate over the knowledge base vector and print the premise list
185. \* and the conclusion they lead to in a human readable format.
186. \*/
187. void KnowledgeBase::displayBase()
188. {
189. Statement n;
190. unsigned int pntr = 1;
191. while (pntr < kBase.size())
192. {
193. unsigned int pPntr = 1,
194. lastPntr = kBase.at(pntr).premiseList.size();
195. std::cout << pntr << ". IF ";
196. while (pPntr < kBase.at(pntr).premiseList.size())
197. {
198. std::cout << kBase.at(pntr).premiseList.at(pPntr).name;
199. pPntr++;
200. if (pPntr < lastPntr)
201. std::cout << " AND ";
202. }
203. std::cout << " THEN " << kBase.at(pntr).conclusion.name << std::endl;
204. pntr++;
205. }
206. }
207. /\*\*
208. \* getConclusion - helper function, which allows to get a conclusion of a specific index in the KB
209. \*
210. \* @param unsigned int index - position of the statement in the knowledge base vector, whose conclusion we want
211. \*
212. \* @return std::string the string representation of the conclusion
213. \*/
214. std::string KnowledgeBase::getConclusion(unsigned int index)
215. {
216. return kBase.at(index).conclusion.name;
217. }
218. /\*\*
219. \* getConclusion - helper function, which allows to get a conclusion of a specific index in the KB
220. \*
221. \* @param unsigned int index - position of the statement in the knowledge base vector, whose conclusion we want
222. \* @param unsigned int index\_1 - once we know which statement, we need to know which premise
223. \*                               as a statement may contain one or more premises.
224. \*
225. \* @return std::string the string representation of the premise
226. \*/
227. std::string KnowledgeBase::getPremise(unsigned int index, unsigned int index\_1)
228. {
229. return kBase.at(index).premiseList.at(index\_1).name;
230. }

## BackChain.hpp

1. #ifndef BACK\_CHAIN\_H
2. #define BACK\_CHAIN\_H
3. #include <string>
4. #include <vector>
5. #include "Statement.hpp"
6. #include "VariableListItem.hpp"
7. #include "ClauseItem.hpp"
8. #include "KnowledgeBase.hpp"
9. class BackChain
10. {
11. public:
12. void populateLists();
13. void runBackwardChaining();
14. void populateVariableList(std::string);
15. KnowledgeBase ruleSystem;
16. std::vector<VariableListItem> variableList;
17. // Due to the design of the system (use the info from the backward
18. // chain to populate the forward chain), this list is needed
19. // to keep track of the conclusions that were set.
20. std::vector<VariableListItem> intermediateConclusionList;
21. private:
22. int findValidConclusionInStatements(std::string conclusionName, int startingIndex, std::string stringToMatch);
23. bool instantiatePremiseClause(const ClauseItem& clause);
24. bool processPremiseList(const Statement& statement);
25. void addToIntermediateConclusionList(const ClauseItem& intermediateConclusion);
26. };
27. #endif // !BACK\_CHAIN\_H

## BackChain.cpp

1. #include <iostream>
2. #include <fstream>
3. #include "ClauseItem.hpp"
4. #include "BackChain.hpp"
5. /\*\*
6. \* Member Function | BackChain | populateLists
7. \*
8. \* Summary: Populates the knowledge base and variable lists for the back
9. \*          chaining portion of the program. Typically read from a csv or text
10. \*          file. Takes the outside representation of the knowledge base and
11. \*          allows the inference engine to act on it.
12. \*/
13. void BackChain::populateLists()
14. {
15. // To offest the vectors by 1, populate index 0 with NULL or Empty elements.
16. ruleSystem.kBase.push\_back(Statement());
17. variableList.push\_back(VariableListItem("Empty", false, "", "This is an error string", STRING));
18. intermediateConclusionList.push\_back(VariableListItem("Empty", false, "", "This is an error string", STRING));
19. // Populate the knowledge base and variable list.
20. ruleSystem.populateKnowledgeBase("knowledgeBase.txt");
21. populateVariableList("variablesList.csv");
22. }
23. /\*\*
24. \* Member Function | BackChain | populateVariableList
25. \*
26. \* Summary: Populates the variable list for back chaining. This initial list
27. \*          contains only the premises. Note that is is later passed on
28. \*          to forward chaining in a modified format.
29. \*
30. \* @param string fileName: The name of the file to read entries from. This file
31. \*                  is in a CSV format of name, prompt, type.
32. \*
33. \*/
34. void BackChain::populateVariableList(std::string fileName)
35. {
36. std::string csvLine;
37. std::string name;
38. std::string prompt = " ";
39. int type = 1;
40. int startParseLocation = 0;
41. int endParseLocation = 0;
42. bool isValid = true;
43. std::ifstream variableListFile;
44. variableListFile.open(fileName);
45. int varCount = 0;
46. std::cout << "List of variables: ";
47. if (variableListFile)
48. {
49. while (getline(variableListFile, csvLine))
50. {
51. startParseLocation = 0;
52. endParseLocation = (csvLine.find(',', startParseLocation) - startParseLocation);
53. if (endParseLocation <= -1)
54. {
55. isValid = false;
56. }
57. name = csvLine.substr(startParseLocation, endParseLocation);
58. std::cout << name << ", ";
59. startParseLocation = endParseLocation + 1;
60. endParseLocation = (csvLine.find(',', startParseLocation) - startParseLocation);
61. if (endParseLocation <= -1)
62. {
63. isValid = false;
64. }
65. prompt = csvLine.substr(startParseLocation, endParseLocation);
66. startParseLocation = endParseLocation + 1;
67. endParseLocation = (csvLine.find(',', startParseLocation) - startParseLocation);
68. if (endParseLocation <= -1)
69. {
70. isValid = false;
71. }
72. if (isValid)
73. {
74. variableList.push\_back(VariableListItem(name, false, "", prompt, type));
75. varCount++;
76. }
77. else
78. {
79. std::cout << "\nInvalid entry, line " << csvLine << " not added." << std::endl;
80. }
81. }
82. std::cout << "\nNumber of variables: " << varCount << std::endl;
83. }
84. else
85. {
86. std::cout << "Could not find the file" << std::endl;
87. }
88. }
89. /\*\*
90. \* Member Function | BackChain | processPremiseList
91. \*
92. \* Summary: Processed the premise list of a given statement. This statement
93. \*          can lead to a recursive call if the statement contains a conclusion
94. \*          in its premise list.
95. \*
96. \* Preconditions:   The statement parameter was found to have a valid conclusion.
97. \*
98. \* @param Statement& statement   :   An individual statement taken from the
99. \*                                      knowledge base. It typically includes
100. \*                                      premise clauses that will be processed
101. \*                                      in order to see if they are conclusions,
102. \*                                      premise clauses, or just invalid.
103. \*
104. \*/
105. bool BackChain::processPremiseList(const Statement &statement)
106. {
107. int solution = 0;
108. bool isValid = true;
109. int location = 0;
110. int conclusionLocation = 0;
111. std::string valueToMatch = "";
112. // Process the premise list for a conclusion that was found to be valid.
113. for (int premiseIter = 1; (isValid && premiseIter < statement.premiseList.size()); premiseIter++)
114. {
115. // Go through and if it is a conclusion on the premise side,
116. // back chain with it.
117. // This will cause another recursive call by adding a conclusion
118. // to the stack. It is this step that allows the removal of the actual
119. // stack in back chaining.
120. conclusionLocation = findValidConclusionInStatements(statement.premiseList.at(premiseIter).name, 1,
121. statement.premiseList.at(premiseIter).value);
122. // It is a conclusion but not valid
123. if (conclusionLocation == -1)
124. {
125. isValid = false;
126. }
127. // It is a conclusion and valid
128. if (conclusionLocation > 0)
129. {
130. isValid = true;
131. // This step is not needed for the backward chaining portion. It is used
132. // when farward chaining is to immediately follow backward chaining and
133. // use values that have already been instantiated.
134. addToIntermediateConclusionList(statement.premiseList.at(premiseIter));
135. }
136. // It was not a conclusion. Go to the clause variable list and
137. // check if it is instantiated as well as what the value was.
138. if (conclusionLocation == 0)
139. {
140. isValid = instantiatePremiseClause(statement.premiseList.at(premiseIter));
141. }
142. }
143. return isValid;
144. }
145. /\*\*
146. \* Member Function | BackChain | instantiatePremiseClause
147. \*
148. \* Summary: Checks to see if the single premise clause passed in has a matching
149. \*          value to what exists in the clause variable list. If not, ask for
150. \*          a value and then check.
151. \*
152. \* Postcondition:   The clause variable list, if it matches, will have the
153. \*                  matching entry instantiated.
154. \*
155. \* @param  const ClauseItem& clause:   An individual premise clause. Contains
156. \*                                      a name and potentially a value.
157. \*
158. \* @return bool isValid:   Returns if the individual premise clause was found
159. \*                          to be valid by matching the name and value of the
160. \*                          premise clause in the knowledge base to what is
161. \*                          in the clause variable list.
162. \*
163. \*/
164. bool BackChain::instantiatePremiseClause(const ClauseItem &clause)
165. {
166. bool isValid = false;
167. bool isFound = false;
168. // Go through the entire clause variable list and look for the matching
169. // Entry. It has to find a match. If not, it could be the case that
170. // The two are out of sync with eachother (the knowledge base and clause
171. // variable list).
172. for (int premiseClauseIter = 1; (!isFound && premiseClauseIter < variableList.size()); premiseClauseIter++)
173. {
174. // If the premise clause we are looking to resolve matches, check
175. // its status.
176. if (clause.name == variableList.at(premiseClauseIter).name)
177. {
178. isFound = true;
179. // This means that it is the first time we encountered this
180. // premise. We need more info and will get it in this step.
181. if (!variableList.at(premiseClauseIter).instantiated)
182. {
183. std::cout << variableList.at(premiseClauseIter).description << ": ";
184. std::cin >> variableList.at(premiseClauseIter).value;
185. std::cout << "\nYou entered: " << variableList.at(premiseClauseIter).value << std::endl;
186. variableList.at(premiseClauseIter).instantiated = true;
187. }
188. // Clause variable list is guaranteed to be updated here.
189. // It can now be safely compared to the incoming premise clause
190. // value.
191. if (variableList.at(premiseClauseIter).value == clause.value)
192. {
193. isValid = true;
194. }
195. }
196. }
197. // The premise clause we just looked at will either be good or bad.
198. // One thing to note is that this is for an individual premise.
199. return isValid;
200. }
201. /\*\*
202. \* Member Function | BackChain | findValidConclusionInStatements
203. \*
204. \* Summary: Takes a conclusion name and value and tries to find a statement
205. \*          that matches up to both. If it finds one and the recursive stack
206. \*          is done, that will be the solution. If the stack is not empty,
207. \*          that means that we just completed an intermediate step in the process.
208. \*
209. \* @param string conclusionName: The name of a conclusion to match up to. Used
210. \*                                  As the first part in checking if a statement
211. \*                                  is valid or not.
212. \* @param int startingIndex: The first index location to begin searching
213. \*                                  from. Typically a 1, but can be adjusted.
214. \*          string  stringToMatch:  The value portion to be matched when searching
215. \*                                  for a valid conclusion. It will take on
216. \*                                  a value when this function is called
217. \*                                  recursively.
218. \*
219. \* @return int location:   Specifies the location of a conclusion.
220. \*
221. \*/
222. int BackChain::findValidConclusionInStatements(std::string conclusionName, int startingIndex, std::string stringToMatch)
223. {
224. int location = 0;
225. bool isConclusion = false;
226. bool isValid = false;
227. // This loop will go through the knowledge base and look for a matching
228. // conclusion in all of the statements. It initially is not trying to
229. // find a match to the conclusion value, as the first inquiry will be
230. // the open ended question that the user wants the system to solve.
231. // It also begins at index 1 for the first run.
232. for (int conclusionIter = startingIndex; (conclusionIter < ruleSystem.kBase.size() && !isValid); conclusionIter++)
233. {
234. // Check that the conclusion name matches what the user entered or
235. // if the back chain is recursing, see if it matches the conclusion
236. // Next in the list.
237. if (conclusionName == ruleSystem.kBase.at(conclusionIter).conclusion.name)
238. {
239. // It matched the conclusion name, just that at this point.
240. isConclusion = true;
241. // Note the DONTCARE here. This allows the initial inquiry to go through
242. // Since it is open ended. However, if not DONTCARE, the stringToMatch
243. // Parameter that was passed in must match. This is due to the multi
244. // purposing of this function.
245. if (stringToMatch == ruleSystem.kBase.at(conclusionIter).conclusion.value || stringToMatch == "DONTCARE")
246. {
247. // It matched the conclusion name (above) and now it also matched the
248. // value in the knowledge base. This needs to be fully processed.
249. // Process premiseList will do just that for this statement.
250. // If everything lines up, we are good.
251. isValid = processPremiseList(ruleSystem.kBase.at(conclusionIter));
252. if (isValid)
253. {
254. // Everything matched up, conclusion name, conclusion value
255. // and the premises all were good.
256. location = conclusionIter;
257. }
258. }
259. }
260. }
261. /\*
262. \* There are actually three options here. -1 means that the conclusion name
263. \* was found, but it was not valid. This could happen if there is a bad
264. \* knowledge base or perhaps the user entered in a bad value, such as an x
265. \* instead of a y or n.
266. \*
267. \* The second option is 0, which means there was no match, no nothing.
268. \* This can happen if the user enters in a bad inquiry to start.
269. \*
270. \* The third option is the actual index of where a valid conclusion was found.
271. \*/
272. if (isConclusion && !isValid)
273. {
274. location = -1;
275. }
276. // See comment right above this one for info on this return value.
277. return location;
278. }
279. /\*\*
280. \* Member Function | BackChain | runBackwardChaining
281. \*
282. \* Summary: The entry point for starting the backward chain process. Asks
283. \*          the user to enter the conclusion to solve and then runs.
284. \*
285. \*/
286. void BackChain::runBackwardChaining()
287. {
288. std::string conclusionToSolve = "";
289. int conclusionLocation = 0;
290. bool isSolvedStatement = false;
291. std::cout << "Please enter a conclusion to solve (values can be: ";
292. int tmpSetCounter = 0;
293. for (auto f : ruleSystem.conclusionSet)
294. {
295. std::cout << f;
296. tmpSetCounter++;
297. if (tmpSetCounter != ruleSystem.conclusionSet.size())
298. {
299. std::cout << ", ";
300. }
301. }
302. std::cout << "): ";
303. std::cin >> conclusionToSolve;
304. std::cout << "\nYou entered: " << conclusionToSolve << std::endl;
305. conclusionLocation = findValidConclusionInStatements(conclusionToSolve, 1, "DONTCARE");
306. //is a conclusion but not valid
307. if (conclusionLocation == -1)
308. {
309. std::cout << "No conclusion match available. Based on your entries, the results are inconclusive. ";
310. }
312. //is a conclusion and valid
313. if (conclusionLocation > 0)
314. {
315. std::cout << "\nResult is: " << ruleSystem.kBase.at(conclusionLocation).conclusion.value << std::endl;
316. std::cout << "Conclusion is valid. ";
317. }
318. //not a conclusion
319. if (conclusionLocation == 0)
320. {
321. std::cout << "No conclusion. ";
322. }
323. }
324. /\*\*
325. \* Member Function | BackChain | addToIntermediateConclusionList
326. \*
327. \* Summary: This function allows for intermediate conclusion clauses to be added
328. \*          to the forward chaining variable list. This is needed in order to
329. \*          run the forward chaining portion immediately after the backward
330. \*          chaining portion. It allows for all entries including the resolved
331. \*          intermediate conclusion clauses to be preserved. Note that the
332. \*          description field is not specific. This is entered in here only to
333. \*          keep in line with the other entries. It will not actually be seen.
334. \*
335. \*/
336. void BackChain::addToIntermediateConclusionList(const ClauseItem &intermediateConclusion)
337. {
338. intermediateConclusionList.push\_back(
339. VariableListItem(intermediateConclusion.name,
340. true,
341. intermediateConclusion.value,
342. (intermediateConclusion.name + "(y/n)"),
343. intermediateConclusion.type));
344. }

## ForwardChain.hpp

1. #ifndef FORWARD\_CHAIN\_H
2. #define FORWARD\_CHAIN\_H
3. #include <string>
4. #include <vector>
5. #include <queue>
6. #include "Statement.hpp"
7. #include "VariableListItem.hpp"
8. #include "ClauseItem.hpp"
9. #include "KnowledgeBase.hpp"
10. /\*\*
11. \* Through forward chaining, we provide repair recommendations to the user based on their input.
12. \* We pass a copy of the KnowledgeBase from BackChain to ForwardChain.
13. \*/
14. class ForwardChain
15. {
16. public:
17. void runForwardChaining();
18. void copyVariableList(const std::vector<VariableListItem>& srcVariableList);
19. void copyKnowledgeBase(const KnowledgeBase& srcKnowledgeBase);
20. int getMatchingVariableListEntry(std::string entryName);
21. void addIntermediateConclusions(const std::vector<VariableListItem>& srcConclusionVariableList);
23. std::queue<ClauseItem> conclusionVariableQueue;
24. KnowledgeBase ruleSystem;
25. std::vector<VariableListItem> variableList;
26. private:
27. void processStatementIndex(int variableListEntry);
28. bool instantiatePremiseClause(const ClauseItem& clause);
29. bool processPremiseList(std::vector<ClauseItem>& premiseList);
30. };
31. #endif // !FORWARD\_CHAIN\_H

## ForwardChain.cpp

1. #include <iostream>
2. #include "ForwardChain.hpp"
3. /\*\*
4. \* Member Function | ForwardChain | copyVariableList
5. \*
6. \* Summary: Copies over all values that were used to find the backward chaining
7. \*          conclusion from the clause variable list. This is part one of two.
8. \*          The second part will also bring over the intermediate conclusions.
9. \*
10. \* @param  const vector<VariableListItem>& srcVariableList:    The variable
11. \*                  list to be copied over. Each element is copied with the
12. \*                  current values instantiated by backward chaining.
13. \*
14. \*/
15. void ForwardChain::copyVariableList(const std::vector<VariableListItem> &srcVariableList)
16. {
17. //do start at 0 in this case, might as well copy over the NULL.
18. for (int varListiter = 0; varListiter < srcVariableList.size(); varListiter++)
19. {
20. variableList.push\_back(srcVariableList.at(varListiter));
21. variableList.back().populateStatementIndex(ruleSystem);
22. }
23. }
24. /\*\*
25. \* Member Function | ForwardChain | addIntermediateConclusions
26. \*
27. \* Summary: Takes the intermediate conclusion list that was populated during
28. \*          backward chaining and adds it to the clause variable list. This is
29. \*          needed due to forward chaining requiring all premise items to be in
30. \*          the clause variable list, which is different from backward chaining.
31. \*
32. \* @param  const vector<VariableListItem>& srcConclusionVariableList:  The
33. \*                  additional conclusion variable list items to copy into the
34. \*                  forward chaining clause variable list.
35. \*
36. \*/
37. void ForwardChain::addIntermediateConclusions(const std::vector<VariableListItem> &srcConclusionVariableList)
38. {
39. //do start at 0 in this case, might as well copy over the NULL.
40. for (int conclListIter = 0; conclListIter < srcConclusionVariableList.size(); conclListIter++)
41. {
42. variableList.push\_back(srcConclusionVariableList.at(conclListIter));
43. variableList.back().populateStatementIndex(ruleSystem);
44. }
45. }
46. /\*\*
47. \* Member Function | ForwardChain | copyKnowledgeBase
48. \*
49. \* Summary: Copy the internal representation of the knowledge base over to be used
50. \*          inforward chaining.
51. \*
52. \* @param  const KnowledgeBase& srcKnowledgeBase:  The internal representation
53. \*          of the knowledge base to be used. Similar to what was done for
54. \*          back chaining.
55. \*/
56. void ForwardChain::copyKnowledgeBase(const KnowledgeBase &srcKnowledgeBase)
57. {
58. //do start at 0 in this case, might as well copy over the NULL.
59. for (int kBaseIter = 0; kBaseIter < srcKnowledgeBase.kBase.size(); kBaseIter++)
60. {
61. ruleSystem.kBase.push\_back(srcKnowledgeBase.kBase.at(kBaseIter));
62. }
63. }
64. /\*\*
65. \* Member Function | ForwardChain | runForwardChaining
66. \*
67. \* Summary: Entry point for running forward chaining. This is expected to run
68. \*          after backward chaining, as part of the suggested fix step.
69. \*
70. \*/
71. void ForwardChain::runForwardChaining()
72. {
73. ClauseItem queueTopPtr;
74. int variableListEntry;
75. int initialRepairEntry;
76. queueTopPtr.name = "inconclusive";
77. queueTopPtr.value = "no valid solution.";
78. std::cout << std::endl
79. << std::endl
80. << "Now running forward chain" << std::endl;
81. // Start the chain by looking for the very first prompt, does it have an issue.
82. // Note that this will also prevent forward chaining from running if the user
83. // entered in a bad value to resolve while back chaining.
84. initialRepairEntry = getMatchingVariableListEntry("has\_issue");
85. if (variableList.at(initialRepairEntry).instantiated)
86. {
87. conclusionVariableQueue.push(ClauseItem(variableList.at(initialRepairEntry).name,
88. variableList.at(initialRepairEntry).value,
89. variableList.at(initialRepairEntry).type));
90. }
91. while (!conclusionVariableQueue.empty())
92. {
93. queueTopPtr = conclusionVariableQueue.front();
94. std::cout << "Processing " << queueTopPtr.name << std::endl;
95. // Note that this is the only location where the queue is reduced.
96. conclusionVariableQueue.pop();
97. //get the matching entry in the variable list, the value does not matter at this time.
98. variableListEntry = getMatchingVariableListEntry(queueTopPtr.name);
99. //go through the variable list's inverted index of statements and push any valid conclusions.
100. //make sure to prompt for entry of any non instantiated.
101. if (variableListEntry != -1)
102. {
103. processStatementIndex(variableListEntry);
104. }
105. }
106. std::cout << "The final conclusion is - " << queueTopPtr.name << " - with a value of: " << queueTopPtr.value << std::endl;
107. }
108. /\*\*
109. \* Member Function | ForwardChain | processStatementIndex
110. \*
111. \* Summary: Runs through an inverted index of the current variable list entry and
112. \*          checks to see which statements are to be processed due to its value.
113. \*          This step is part of the BFS, where each matching item
114. \*          is added to the queue for this particular entry before it is
115. \*          popped off the queue and the next one is processed.
116. \*
117. \*  @param int variableListEntry: A numeric value of which variable list entry
118. \*          is being processed.
119. \*
120. \*  @return    None - note that this is indeed the case since the queue will be
121. \*              added to if there is a valid value. If it is not valid, it is not
122. \*              added.
123. \*/
124. void ForwardChain::processStatementIndex(int variableListEntry)
125. {
126. int curStatement = 0;
127. for (int variableListIter = 1; variableListIter < variableList.at(variableListEntry).statementIndex.size(); variableListIter++)
128. {
129. //           = The matching variable list entry  . The individual statment number
130. curStatement = variableList.at(variableListEntry).statementIndex.at(variableListIter);
131. if (true == processPremiseList(ruleSystem.kBase.at(curStatement).premiseList))
132. {
133. // Everything matched up, so move forward on adding it to the queue to be
134. // processed.
135. conclusionVariableQueue.push(ruleSystem.kBase.at(curStatement).conclusion);
136. }
137. }
138. }
139. /\*\*
140. \* Member Function | ForwardChain | processPremiseList
141. \*
142. \* Summary: Takes a conclusion name and value and tries to find a statement
143. \*          that matches up to both. If it finds one and the recursive stack
144. \*          is done, that will be the solution. If the stack is not empty,
145. \*          that means that we just completed an intermediate step in the process.
146. \*
147. \* @param  vector<ClauseItem>& premiseList: the premise list of a particular
148. \*                      statement.
149. \*
150. \* @return bool isValid: Specifies if the premise clauses were all found
151. \*                  to be valid for a particular statement..
152. \*
153. \*/
154. bool ForwardChain::processPremiseList(std::vector<ClauseItem> &premiseList)
155. {
156. bool isValid = true;
157. for (int premiseIter = 1; (isValid && premiseIter < premiseList.size()); premiseIter++)
158. {
159. isValid = instantiatePremiseClause(premiseList.at(premiseIter));
160. }
161. return isValid;
162. }
163. /\*\*
164. \* Member Function | ForwardChain | instantiatePremiseClause
165. \*
166. \* Summary: Takes a single clause and verifies that it has been instantiated and
167. \*          that it matches up to the back chaining portion of diagnostics.
168. \*          Note that this varies slightly from back chaining and the typical
169. \*          behavior of the instantiation step.
170. \*
171. \* Preconditions: Backchaining has been ran and the variable list is populated
172. \*                  with the needed results.
173. \*
174. \* @param  string  conclusionName: The name of a conclusion to match up to. Used
175. \*                                  As the first part in checking if a statement
176. \*                                  is valid or not.
177. \*
178. \* @return isFound:    Specifies if the incoming clause is found within the
179. \*                      variable list.
180. \*/
181. bool ForwardChain::instantiatePremiseClause(const ClauseItem &clause)
182. {
183. bool isFound = false;
184. // Look through the variable list and see if this particular clause has a valid match.
185. for (int varListIter = 1; (!isFound && varListIter < variableList.size()); varListIter++)
186. {
187. if (variableList.at(varListIter).instantiated && clause.name == variableList.at(varListIter).name && clause.value == variableList.at(varListIter).value)
188. {
189. isFound = true;
190. }
191. }
192. return isFound;
193. }
194. /\*\*
195. \* Member Function | ForwardChain | getMatchingVariableListEntry
196. \*
197. \* Summary: Takes a conclusion name that was popped off of the queue and tries to
198. \*          locate it in the variable list. Note that the variable list will only
199. \*          contain unique values.
200. \*
201. \* @param  string entryName: The name of the variable list entry to match up.
202. \*
203. \* @return int matchingEntryIndex: The location of the matching entry. Returns
204. \*                  -1 if it is not found.
205. \*
206. \*/
207. int ForwardChain::getMatchingVariableListEntry(std::string entryName)
208. {
209. int matchingEntryIndex = -1;
210. bool isFound = false;
211. for (int variableListIter = 1; (!isFound && variableListIter < variableList.size()); variableListIter++)
212. {
213. if (entryName == variableList.at(variableListIter).name)
214. {
215. isFound = true;
216. matchingEntryIndex = variableListIter;
217. }
218. }
219. return matchingEntryIndex;
220. }

## VariablesList.csv

1. has\_issue,Is there an issue with the vehicle? (y/n),STRING
2. is\_starting,Does the car start? (y/n),STRING
3. has\_fuel,Is there enough fuel? (y/n),STRING
4. has\_voltage,Does the battery have enough voltage? (y/n),STRING
5. is\_ignition\_coil\_damaged,Is the ignition coil damaged? (y/n),STRING
6. is\_distributor\_cap\_damaged,Is the distributor cap cracked or broken? (y/n),STRING
7. is\_timing\_belt\_damaged,Is the timing belt damaged? (y/n),STRING
8. is\_making\_noise,Is the car making an unusual noise? (y/n),STRING
9. is\_noisy\_while\_driving,Does the noise occur when driving? (y/n),STRING
10. is\_ticking\_noise,Is the car making a ticking noise? (y/n),STRING
11. is\_hiccup\_noise,Is the car making a rattly noise like a hiccup? (y/n),STRING
12. is\_air\_filter\_dirty,Is the air filter dirty? (y/n),STRING
13. is\_exhaust\_blocked,Is anything blocking the exhaust? (y/n),STRING
14. are\_wheel\_bearings\_damaged,Are wheel bearings damaged/loose? (y/n),STRING
15. are\_tires\_bald,Are tires beyond service life (tread depth less than 1/32in)? (y/n),STRING
16. is\_truck,Is the vehicle a truck? (y/n),STRING
17. has\_items\_in\_truck\_bed,Are there items in the truck bed? (y/n),STRING
18. has\_items\_on\_roof,Does the vehicle have any items latched to the roof? (y/n),STRING
19. is\_overheating,Does the temperature gauge show as overheating? (y/n),STRING
20. has\_coolant,Is there sufficient coolant in the engine? (y/n),STRING
21. is\_water\_pump\_broken,Is the water pump broken? (y/n),STRING
22. is\_oil\_low\_or\_dirty,Is the oil level low or dirty (black in color)? (y/n),STRING
23. has\_nonfunctional\_electronics,Are any electronics not functional? (y/n),STRING
24. does\_ac\_power\_on,Does the AC power on? (y/n),STRING
25. does\_ac\_blow\_cold,Does the AC blow cold? (y/n),STRING
26. has\_nonfunctional\_headlights,Are any headlights or lights not turning on? (y/n),STRING
27. is\_ac\_fuse\_intact,Is the AC fuse intact? (y/n),STRING
28. are\_ac\_wires\_connected,Are AC ground and power wires connected? (y/n),STRING
29. are\_therm\_settings\_correct,Are the thermostat settings correct? (y/n),STRING
30. is\_evaporator\_coil\_frozen,Is the evaporator coil frozen? (y/n),STRING
31. is\_air\_filter\_dirty,Is the air filter dirty? (y/n),STRING
32. is\_nonfunct\_light\_fuse\_intact,Is the respective non-working light fuse intact? (y/n),STRING
33. are\_nonfunct\_light\_wires\_conn,Are ground and power wires connected? (y/n),STRING
34. has\_burning\_plastic\_smell,Do you smell any burning plastic / electric insulation? (y/n),STRING
35. is\_radio\_working,Does the radio power on? (y/n),STRING
36. is\_radio\_fuse\_intact,Is the radio fuse intact? (y/n),STRING
37. are\_radio\_wires\_connected,Are radio ground and power wires connected? (y/n),STRING
38. does\_wheel\_turn,Does the steering wheel turn? (y/n),STRING
39. is\_power\_steering\_fuse\_intact,Is the power steering fuse intact? (y/n),STRING
40. has\_power\_steering\_fluid,Is there enough power steering fluid? (y/n),STRING
41. are\_tires\_deflated,Are any of the tires visibly deflated or is the low pressure light on? (y/n),STRING
42. has\_piercing\_object,Is there a sharp item (e.g. nail) stuck in the deflated tire? (y/n),STRING
43. is\_obj\_inch\_away\_from\_edge,Is the piercing object more than 1in away from the tire edge? (y/n),STRING
44. is\_recent\_temp\_change,Was there a recent temperature change? (y/n),STRING
45. is\_exterior\_damaged,Is the exterior of the vehicle damaged? (y/n),STRING
46. is\_damage\_cosmetic,Is the damage cosmetic paint scratch only? (y/n),STRING

## KnowledgeBase.txt

1. has\_issue = n : issue = No issue
2. issue = No issue : repair = No Repair Required
3. has\_issue = y ^ is\_starting = n : issue = Failure to Start
4. issue = Failure to Start ^ has\_fuel = n : repair = Insufficient Fuel, Add more fuel.
5. issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = n : repair = Dead Battery, Change the battery.
6. issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = y : repair = Bad Ignition Coil, Replace faulty ignition coil.
7. issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = n ^ is\_distributor\_cap\_damaged = y : repair = Bad Distributor Cap, Replace faulty distributor cap.
8. issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = n ^ is\_distributor\_cap\_damaged = n ^ is\_timing\_belt\_damaged = y : repair = Bad Timing Belt, Replace faulty timing belt.
9. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = y : issue = Noise Issue
10. issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = y : repair = Faulty bearings, replace wheel bearings
11. issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = y : repair = Tires are worn, replace with new tires
12. issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = n ^ has\_items\_on\_roof = y : repair = Remove items from roof
13. issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = n ^ has\_items\_on\_roof = n ^ is\_truck = y ^ has\_items\_in\_truck\_bed = y : repair = Remove items from truck bed
14. issue = Noise Issue ^ is\_noisy\_while\_driving = n ^ is\_ticking\_noise = n ^ is\_hiccup\_noise = y ^ is\_air\_filter\_dirty = y : repair = Replace dirty air filter with clean one
15. issue = Noise Issue ^ is\_noisy\_while\_driving = n ^ is\_ticking\_noise = n ^ is\_hiccup\_noise = y ^ is\_air\_filter\_dirty = n ^ is\_exhaust\_blocked = y : repair = Remove object obstructing exhaust
16. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = y : issue = Overheating Issue
17. issue = Overheating Issue ^ has\_coolant = n : repair = Low Coolant, add coolant.
18. issue = Overheating Issue ^ has\_coolant = y ^ is\_water\_pump\_broken = y : repair = Defective Water Pump, replace water pump.
19. issue = Overheating Issue ^ has\_coolant = y ^ is\_water\_pump\_broken = n ^ is\_oil\_low\_or\_dirty = y : repair = Low or Dirty Oil, replace oil
20. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = y : issue = Electronics Issue
21. issue = Electronics Issue ^ does\_ac\_power\_on = n ^ is\_ac\_fuse\_intact = n : repair = Replace defective AC fuse
22. issue = Electronics Issue ^ does\_ac\_power\_on = n ^ is\_ac\_fuse\_intact = y ^ are\_ac\_wires\_connected = n : repair = Connect AC wires properly
23. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = n : repair = Fix thermostat settings
24. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = y ^ is\_evaporator\_coil\_frozen = y : repair = Defrost evaporator coil
25. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = y ^ is\_evaporator\_coil\_frozen = n ^ is\_air\_filter\_dirty = y : repair = Replace dirty air filter with clean one
26. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = y ^ is\_nonfunct\_light\_fuse\_intact = n : repair = Replace defective light fuse
27. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = y ^ is\_nonfunct\_light\_fuse\_intact = y ^ are\_nonfunct\_light\_wires\_conn = n : repair = Connect light wires properly
28. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = y : repair = Serious issue and possible short circuit, replace wiring and fuses.
29. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = n : repair = Replace defective radio fuse
30. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = y ^ are\_radio\_wires\_connected = n : repair = Connect radio wires properly
31. issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = y ^ are\_radio\_wires\_connected = y : repair = Wiring or installation issue, replace radio and wiring
32. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = n : issue = Steering Issue
33. issue = Steering Issue ^ is\_power\_steering\_fuse\_intact = n : repair = Replace power steering fuse
34. issue = Steering Issue ^ is\_power\_steering\_fuse\_intact = y ^ has\_power\_steering\_fluid = n : repair = Power steering failure, add power steering fluid
35. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = y ^ are\_tires\_deflated = y : issue = Tire Issue
36. issue = Tire Issue ^ has\_piercing\_object = y ^ is\_obj\_inch\_away\_from\_edge = n : repair = Replace deflated tire
37. issue = Tire Issue ^ has\_piercing\_object = y ^ is\_obj\_inch\_away\_from\_edge = y : repair = Apply tire patch kit to deflated tire
38. issue = Tire Issue ^ has\_piercing\_object = n ^ is\_recent\_temp\_change = y : repair = Low tire pressure, add air to deflated tires
39. has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = y ^ are\_tires\_deflated = n : issue = General Issue
40. issue = General Issue ^ is\_exterior\_damaged = y ^ is\_damage\_cosmetic = y : repair = Repaint the scratched part
41. issue = General Issue ^ is\_exterior\_damaged = y ^ is\_damage\_cosmetic = n : repair = Replace the damaged vehicle body part
42. issue = General Issue ^ is\_exterior\_damaged = n : repair = No damage/fault detected

# Appendix E: Complete Sample Output for Sample Run #1

1. Welcome to the Automobile Diagnostic Program.
2. Authors: David Torrente (dat54@txstate.edu), Randall Henderson (rrh93@txstate.edu), Borislav Sabotinov (bss64@txstate.edu).
3. Creating knowledge base instance...
4. Processing: has\_issue = n : issue = No issue
5. Conclusion is good; Premise list is good! 1 premise(s) loaded
6. Conclusion and premise(s) are good => List Updated
7. Processing: issue = No issue : repair = No Repair Required
8. Conclusion is good; Premise list is good! 1 premise(s) loaded
9. Conclusion and premise(s) are good => List Updated
10. Processing: has\_issue = y ^ is\_starting = n : issue = Failure to Start
11. Conclusion is good; Premise list is good! 2 premise(s) loaded
12. Conclusion and premise(s) are good => List Updated
13. Processing: issue = Failure to Start ^ has\_fuel = n : repair = Insufficient Fuel, Add more fuel.
14. Conclusion is good; Premise list is good! 2 premise(s) loaded
15. Conclusion and premise(s) are good => List Updated
16. Processing: issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = n : repair = Dead Battery, Change the battery.
17. Conclusion is good; Premise list is good! 3 premise(s) loaded
18. Conclusion and premise(s) are good => List Updated
19. Processing: issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = y : repair = Bad Ignition Coil, Replace faulty ignition coil.
20. Conclusion is good; Premise list is good! 4 premise(s) loaded
21. Conclusion and premise(s) are good => List Updated
22. Processing: issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = n ^ is\_distributor\_cap\_damaged = y : repair = Bad Distributor Cap, Replace faulty distributor cap.
23. Conclusion is good; Premise list is good! 5 premise(s) loaded
24. Conclusion and premise(s) are good => List Updated
25. Processing: issue = Failure to Start ^ has\_fuel = y ^ has\_voltage = y ^ is\_ignition\_coil\_damaged = n ^ is\_distributor\_cap\_damaged = n ^ is\_timing\_belt\_damaged = y : repair = Bad Timing Belt, Replace faulty timing belt.
26. Conclusion is good; Premise list is good! 6 premise(s) loaded
27. Conclusion and premise(s) are good => List Updated
28. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = y : issue = Noise Issue
29. Conclusion is good; Premise list is good! 3 premise(s) loaded
30. Conclusion and premise(s) are good => List Updated
31. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = y : repair = Faulty bearings, replace wheel bearings
32. Conclusion is good; Premise list is good! 3 premise(s) loaded
33. Conclusion and premise(s) are good => List Updated
34. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = y : repair = Tires are worn, replace with new tires
35. Conclusion is good; Premise list is good! 4 premise(s) loaded
36. Conclusion and premise(s) are good => List Updated
37. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = n ^ has\_items\_on\_roof = y : repair = Remove items from roof
38. Conclusion is good; Premise list is good! 5 premise(s) loaded
39. Conclusion and premise(s) are good => List Updated
40. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = y ^ are\_wheel\_bearings\_damaged = n ^ are\_tires\_bald = n ^ has\_items\_on\_roof = n ^ is\_truck = y ^ has\_items\_in\_truck\_bed = y : repair = Remove items from truck bed
41. Conclusion is good; Premise list is good! 7 premise(s) loaded
42. Conclusion and premise(s) are good => List Updated
43. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = n ^ is\_ticking\_noise = n ^ is\_hiccup\_noise = y ^ is\_air\_filter\_dirty = y : repair = Replace dirty air filter with clean one
44. Conclusion is good; Premise list is good! 5 premise(s) loaded
45. Conclusion and premise(s) are good => List Updated
46. Processing: issue = Noise Issue ^ is\_noisy\_while\_driving = n ^ is\_ticking\_noise = n ^ is\_hiccup\_noise = y ^ is\_air\_filter\_dirty = n ^ is\_exhaust\_blocked = y : repair = Remove object obstructing exhaust
47. Conclusion is good; Premise list is good! 6 premise(s) loaded
48. Conclusion and premise(s) are good => List Updated
49. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = y : issue = Overheating Issue
50. Conclusion is good; Premise list is good! 4 premise(s) loaded
51. Conclusion and premise(s) are good => List Updated
52. Processing: issue = Overheating Issue ^ has\_coolant = n : repair = Low Coolant, add coolant.
53. Conclusion is good; Premise list is good! 2 premise(s) loaded
54. Conclusion and premise(s) are good => List Updated
55. Processing: issue = Overheating Issue ^ has\_coolant = y ^ is\_water\_pump\_broken = y : repair = Defective Water Pump, replace water pump.
56. Conclusion is good; Premise list is good! 3 premise(s) loaded
57. Conclusion and premise(s) are good => List Updated
58. Processing: issue = Overheating Issue ^ has\_coolant = y ^ is\_water\_pump\_broken = n ^ is\_oil\_low\_or\_dirty = y : repair = Low or Dirty Oil, replace oil
59. Conclusion is good; Premise list is good! 4 premise(s) loaded
60. Conclusion and premise(s) are good => List Updated
61. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = y : issue = Electronics Issue
62. Conclusion is good; Premise list is good! 5 premise(s) loaded
63. Conclusion and premise(s) are good => List Updated
64. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = n ^ is\_ac\_fuse\_intact = n : repair = Replace defective AC fuse
65. Conclusion is good; Premise list is good! 3 premise(s) loaded
66. Conclusion and premise(s) are good => List Updated
67. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = n ^ is\_ac\_fuse\_intact = y ^ are\_ac\_wires\_connected = n : repair = Connect AC wires properly
68. Conclusion is good; Premise list is good! 4 premise(s) loaded
69. Conclusion and premise(s) are good => List Updated
70. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = n : repair = Fix thermostat settings
71. Conclusion is good; Premise list is good! 4 premise(s) loaded
72. Conclusion and premise(s) are good => List Updated
73. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = y ^ is\_evaporator\_coil\_frozen = y : repair = Defrost evaporator coil
74. Conclusion is good; Premise list is good! 5 premise(s) loaded
75. Conclusion and premise(s) are good => List Updated
76. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = n ^ are\_therm\_settings\_correct = y ^ is\_evaporator\_coil\_frozen = n ^ is\_air\_filter\_dirty = y : repair = Replace dirty air filter with clean one
77. Conclusion is good; Premise list is good! 6 premise(s) loaded
78. Conclusion and premise(s) are good => List Updated
79. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = y ^ is\_nonfunct\_light\_fuse\_intact = n : repair = Replace defective light fuse
80. Conclusion is good; Premise list is good! 5 premise(s) loaded
81. Conclusion and premise(s) are good => List Updated
82. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = y ^ is\_nonfunct\_light\_fuse\_intact = y ^ are\_nonfunct\_light\_wires\_conn = n : repair = Connect light wires properly
83. Conclusion is good; Premise list is good! 6 premise(s) loaded
84. Conclusion and premise(s) are good => List Updated
85. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = y : repair = Serious issue and possible short circuit, replace wiring and fuses.
86. Conclusion is good; Premise list is good! 5 premise(s) loaded
87. Conclusion and premise(s) are good => List Updated
88. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = n : repair = Replace defective radio fuse
89. Conclusion is good; Premise list is good! 7 premise(s) loaded
90. Conclusion and premise(s) are good => List Updated
91. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = y ^ are\_radio\_wires\_connected = n : repair = Connect radio wires properly
92. Conclusion is good; Premise list is good! 8 premise(s) loaded
93. Conclusion and premise(s) are good => List Updated
94. Processing: issue = Electronics Issue ^ does\_ac\_power\_on = y ^ does\_ac\_blow\_cold = y ^ has\_nonfunctional\_headlights = n ^ has\_burning\_plastic\_smell = n ^ is\_radio\_working = n ^ is\_radio\_fuse\_intact = y ^ are\_radio\_wires\_connected = y : repair = Wiring or installation issue, replace radio and wiring
95. Conclusion is good; Premise list is good! 8 premise(s) loaded
96. Conclusion and premise(s) are good => List Updated
97. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = n : issue = Steering Issue
98. Conclusion is good; Premise list is good! 6 premise(s) loaded
99. Conclusion and premise(s) are good => List Updated
100. Processing: issue = Steering Issue ^ is\_power\_steering\_fuse\_intact = n : repair = Replace power steering fuse
101. Conclusion is good; Premise list is good! 2 premise(s) loaded
102. Conclusion and premise(s) are good => List Updated
103. Processing: issue = Steering Issue ^ is\_power\_steering\_fuse\_intact = y ^ has\_power\_steering\_fluid = n : repair = Power steering failure, add power steering fluid
104. Conclusion is good; Premise list is good! 3 premise(s) loaded
105. Conclusion and premise(s) are good => List Updated
106. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = y ^ are\_tires\_deflated = y : issue = Tire Issue
107. Conclusion is good; Premise list is good! 7 premise(s) loaded
108. Conclusion and premise(s) are good => List Updated
109. Processing: issue = Tire Issue ^ has\_piercing\_object = y ^ is\_obj\_inch\_away\_from\_edge = n : repair = Replace deflated tire
110. Conclusion is good; Premise list is good! 3 premise(s) loaded
111. Conclusion and premise(s) are good => List Updated
112. Processing: issue = Tire Issue ^ has\_piercing\_object = y ^ is\_obj\_inch\_away\_from\_edge = y : repair = Apply tire patch kit to deflated tire
113. Conclusion is good; Premise list is good! 3 premise(s) loaded
114. Conclusion and premise(s) are good => List Updated
115. Processing: issue = Tire Issue ^ has\_piercing\_object = n ^ is\_recent\_temp\_change = y : repair = Low tire pressure, add air to deflated tires
116. Conclusion is good; Premise list is good! 3 premise(s) loaded
117. Conclusion and premise(s) are good => List Updated
118. Processing: has\_issue = y ^ is\_starting = y ^ is\_making\_noise = n ^ is\_overheating = n ^ has\_nonfunctional\_electronics = n ^ does\_wheel\_turn = y ^ are\_tires\_deflated = n : issue = General Issue
119. Conclusion is good; Premise list is good! 7 premise(s) loaded
120. Conclusion and premise(s) are good => List Updated
121. Processing: issue = General Issue ^ is\_exterior\_damaged = y ^ is\_damage\_cosmetic = y : repair = Repaint the scratched part
122. Conclusion is good; Premise list is good! 3 premise(s) loaded
123. Conclusion and premise(s) are good => List Updated
124. Processing: issue = General Issue ^ is\_exterior\_damaged = y ^ is\_damage\_cosmetic = n : repair = Replace the damaged vehicle body part
125. Conclusion is good; Premise list is good! 3 premise(s) loaded
126. Conclusion and premise(s) are good => List Updated
127. Processing: issue = General Issue ^ is\_exterior\_damaged = n : repair = No damage/fault detected
128. Conclusion is good; Premise list is good! 2 premise(s) loaded
129. Conclusion and premise(s) are good => List Updated
130. Knowledge Base finished Loading.
131. 42 items were loaded into the KnowledgeBase
132. <CR/Enter> to continue List of variables: has\_issue, is\_starting, has\_fuel, has\_voltage, is\_ignition\_coil\_damaged, is\_distributor\_cap\_damaged, is\_timing\_belt\_damaged, is\_making\_noise, is\_noisy\_while\_driving, is\_ticking\_noise, is\_hiccup\_noise, is\_air\_filter\_dirty, is\_exhaust\_blocked, are\_wheel\_bearings\_damaged, are\_tires\_bald, is\_truck, has\_items\_in\_truck\_bed, has\_items\_on\_roof, is\_overheating, has\_coolant, is\_water\_pump\_broken, is\_oil\_low\_or\_dirty, has\_nonfunctional\_electronics, does\_ac\_power\_on, does\_ac\_blow\_cold, has\_nonfunctional\_headlights, is\_ac\_fuse\_intact, are\_ac\_wires\_connected, are\_therm\_settings\_correct, is\_evaporator\_coil\_frozen, is\_air\_filter\_dirty, is\_nonfunct\_light\_fuse\_intact, are\_nonfunct\_light\_wires\_conn, has\_burning\_plastic\_smell, is\_radio\_working, is\_radio\_fuse\_intact, are\_radio\_wires\_connected, does\_wheel\_turn, is\_power\_steering\_fuse\_intact, has\_power\_steering\_fluid, are\_tires\_deflated, has\_piercing\_object, is\_obj\_inch\_away\_from\_edge, is\_recent\_temp\_change, is\_exterior\_damaged, is\_damage\_cosmetic,
133. Number of variables: 46
134. Do you want to display the knowledge base (y/n)? 1. IF has\_issue THEN issue
135. 2. IF issue THEN repair
136. 3. IF has\_issue AND is\_starting THEN issue
137. 4. IF issue AND has\_fuel THEN repair
138. 5. IF issue AND has\_fuel AND has\_voltage THEN repair
139. 6. IF issue AND has\_fuel AND has\_voltage AND is\_ignition\_coil\_damaged THEN repair
140. 7. IF issue AND has\_fuel AND has\_voltage AND is\_ignition\_coil\_damaged AND is\_distributor\_cap\_damaged THEN repair
141. 8. IF issue AND has\_fuel AND has\_voltage AND is\_ignition\_coil\_damaged AND is\_distributor\_cap\_damaged AND is\_timing\_belt\_damaged THEN repair
142. 9. IF has\_issue AND is\_starting AND is\_making\_noise THEN issue
143. 10. IF issue AND is\_noisy\_while\_driving AND are\_wheel\_bearings\_damaged THEN repair
144. 11. IF issue AND is\_noisy\_while\_driving AND are\_wheel\_bearings\_damaged AND are\_tires\_bald THEN repair
145. 12. IF issue AND is\_noisy\_while\_driving AND are\_wheel\_bearings\_damaged AND are\_tires\_bald AND has\_items\_on\_roof THEN repair
146. 13. IF issue AND is\_noisy\_while\_driving AND are\_wheel\_bearings\_damaged AND are\_tires\_bald AND has\_items\_on\_roof AND is\_truck AND has\_items\_in\_truck\_bed THEN repair
147. 14. IF issue AND is\_noisy\_while\_driving AND is\_ticking\_noise AND is\_hiccup\_noise AND is\_air\_filter\_dirty THEN repair
148. 15. IF issue AND is\_noisy\_while\_driving AND is\_ticking\_noise AND is\_hiccup\_noise AND is\_air\_filter\_dirty AND is\_exhaust\_blocked THEN repair
149. 16. IF has\_issue AND is\_starting AND is\_making\_noise AND is\_overheating THEN issue
150. 17. IF issue AND has\_coolant THEN repair
151. 18. IF issue AND has\_coolant AND is\_water\_pump\_broken THEN repair
152. 19. IF issue AND has\_coolant AND is\_water\_pump\_broken AND is\_oil\_low\_or\_dirty THEN repair
153. 20. IF has\_issue AND is\_starting AND is\_making\_noise AND is\_overheating AND has\_nonfunctional\_electronics THEN issue
154. 21. IF issue AND does\_ac\_power\_on AND is\_ac\_fuse\_intact THEN repair
155. 22. IF issue AND does\_ac\_power\_on AND is\_ac\_fuse\_intact AND are\_ac\_wires\_connected THEN repair
156. 23. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND are\_therm\_settings\_correct THEN repair
157. 24. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND are\_therm\_settings\_correct AND is\_evaporator\_coil\_frozen THEN repair
158. 25. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND are\_therm\_settings\_correct AND is\_evaporator\_coil\_frozen AND is\_air\_filter\_dirty THEN repair
159. 26. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND is\_nonfunct\_light\_fuse\_intact THEN repair
160. 27. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND is\_nonfunct\_light\_fuse\_intact AND are\_nonfunct\_light\_wires\_conn THEN repair
161. 28. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND has\_burning\_plastic\_smell THEN repair
162. 29. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND has\_burning\_plastic\_smell AND is\_radio\_working AND is\_radio\_fuse\_intact THEN repair
163. 30. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND has\_burning\_plastic\_smell AND is\_radio\_working AND is\_radio\_fuse\_intact AND are\_radio\_wires\_connected THEN repair
164. 31. IF issue AND does\_ac\_power\_on AND does\_ac\_blow\_cold AND has\_nonfunctional\_headlights AND has\_burning\_plastic\_smell AND is\_radio\_working AND is\_radio\_fuse\_intact AND are\_radio\_wires\_connected THEN repair
165. 32. IF has\_issue AND is\_starting AND is\_making\_noise AND is\_overheating AND has\_nonfunctional\_electronics AND does\_wheel\_turn THEN issue
166. 33. IF issue AND is\_power\_steering\_fuse\_intact THEN repair
167. 34. IF issue AND is\_power\_steering\_fuse\_intact AND has\_power\_steering\_fluid THEN repair
168. 35. IF has\_issue AND is\_starting AND is\_making\_noise AND is\_overheating AND has\_nonfunctional\_electronics AND does\_wheel\_turn AND are\_tires\_deflated THEN issue
169. 36. IF issue AND has\_piercing\_object AND is\_obj\_inch\_away\_from\_edge THEN repair
170. 37. IF issue AND has\_piercing\_object AND is\_obj\_inch\_away\_from\_edge THEN repair
171. 38. IF issue AND has\_piercing\_object AND is\_recent\_temp\_change THEN repair
172. 39. IF has\_issue AND is\_starting AND is\_making\_noise AND is\_overheating AND has\_nonfunctional\_electronics AND does\_wheel\_turn AND are\_tires\_deflated THEN issue
173. 40. IF issue AND is\_exterior\_damaged AND is\_damage\_cosmetic THEN repair
174. 41. IF issue AND is\_exterior\_damaged AND is\_damage\_cosmetic THEN repair
175. 42. IF issue AND is\_exterior\_damaged THEN repair
176. Please enter a conclusion to solve (values can be: issue, repair):
177. You entered: issue
178. Is there an issue with the vehicle? (y/n):
179. You entered: y
180. Does the car start? (y/n):
181. You entered: y
182. Is the car making an unusual noise? (y/n):
183. You entered: n
184. Does the temperature gauge show as overheating? (y/n):
185. You entered: n
186. Are any electronics not functional? (y/n):
187. You entered: n
188. Does the steering wheel turn? (y/n):
189. You entered: y
190. Are any of the tires visibly deflated or is the low pressure light on? (y/n):
191. You entered: y
192. Result is: Tire Issue
193. Conclusion is valid.
194. Creating knowledge base instance...
195. Now running forward chain
196. Processing has\_issue
197. Processing issue
198. The final conclusion is - issue - with a value of: Tire Issue