Simple Automobile Tire Expert System Built in C++ with Embedded SWI-Prolog

Whetzel Thornsbury
wst12@my.fsu.edu

Abstract

Expert Systems are an interesting part of Artificial Intelligence that focuses on mimicking the decision making ability of human experts. This is accomplished by using formal logic to examine facts and then to make decisions on those facts, similar to how a human would. This system uses two separate parts, the inference engine and the knowledge base. The inference engine is the system that examines the rules and knowledge and then comes to some logical conclusion. The knowledge base is where all the facts and rules are stored. For this paper, a simple Expert System will be created based on knowledge and rules of an automobile tire professional. In order to achieve this, C++ will be used as the interface between the user, Inference Engine, and knowledge based. SWI-Prolog will be used as the Inference Engine.

Introduction

There are many types of expert systems in use today in different industries from medical to automotive. These systems are both used by experts and non-experts to help make decisions based on the facts they have about a particular domain of knowledge.

> An expert system is a program designed to solve problems at a level comparable to that of a human expert in a given domain.
> (Ogu, 2013)

This knowledge is stored in the knowledge base of an expert system. Formal logic is the language that is used to store this knowledge, and will be used to retrieve and processed information as it is queried by the user.

Before that can happen, a user interface will need to take the information from the user in order to compare to the knowledge base to their query. An easy way to do this is to have the user give facts about a particular problem. The program will then package this information and send it to the inference engine.

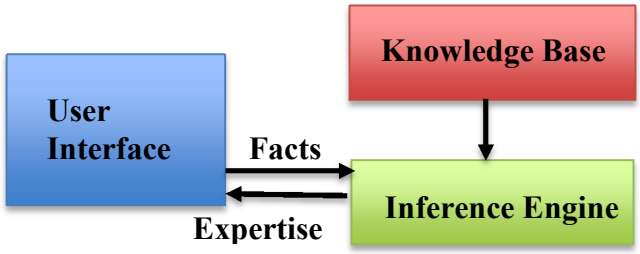
The inference engine is where the magic happens when it comes to expert systems. With the user's information at hand, the inference engine will use this to come up with a decision to a problem. When it comes to inference engines, there are two schools of thought as to how the information is processed and a decision is made; forward chaining and backward chaining.

Forward chaining is very basic in how it operates. "The system simply tests the rules in the order that they occur... If the system needs a variable to determine if a rule is true or false, and the value of the variable is unknown, the system immediately asks the user for the value". (Huntington, 2002) The facts are processed by finding an appropriate rule where the fact is in the IF portion of a predicate, and the THEN portion is output as a solution.

Backwards chaining works in a reverse way by first looking at the THEN portion of the rule. "This requires an IF/THEN rule that assigns a value to the variable in the THEN part of the rule. If such a rule is found, that rules IF portion is tested to determine if it is true." (Huntington, 2002) Unlike forward chaining requiring all variables to have a rule which it is matched to, backward chaining will take an unknown value and perform another backward chain query to find a value that it can use to complete the original query. "If no rule is available to assign a value …, the system asks the user directly." (Huntington, 2002) The Expert system that will be described in this paper will use backwards chaining.

Although there is a more complex "shell" that is used for most expert systems, in this paper I will only describe and build a basic system that will look like Figure 1.

Figure 1. Expert System
Based on Fig. 0.1 of Noran. The Evolution of Expert Systems



Background

Back in the early part of Artificial Intelligence research, one of the projects that researchers worked on was a General Problem Solver, the precursor of modern day expert systems. The first system, called Post, was first derived from mathematics being a "set of rules specifying how to change a string of symbols into another set of symbols." (Noran) The drawback of this system was using symbols for knowledge and manipulating the symbols to form new symbols, "the manipulation of strings is only based on syntax and not on any semantics or understanding of the underlying knowledge." (Noran) What researches wanted was a way for the computer to reason like humans, and this was not possible if a computer does not have rules to help it "understand" the decision.

Newell and Simon demonstrated that much of human problem-solving could be expressed as IF..THEN production rules. (Noran) These rules help the computer understand the decision, not like we understand, but the semantics of the decision process and how knowledge is represented.

Still researchers still focused on creating a general problem solver, but there was too much knowledge to model, and this wasn't efficient. They needed to break up how much knowledge needed to be represented, and this is where the focus of expert systems came from. It was easier to model domain knowledge, then have a computer keep track of all knowledge.

Building a Basic Expert System

So let's break down what will be used to create the expert system created based on the domain knowledge of an automobile tire expert.

First, let's take a look at the inference engine that will be used, Prolog. Prolog is a logic programming language that uses backwards chaining in order to process decisions. As described by A. Aaby, in *Prolog Tutorial*, a Prolog program consists of a data base of facts and logical relationships that is referenced when a question is asked, and the program uses logical deduction to find the answer. This makes it perfect to use as an inference engine, because logical deduction is similar to how experts filter through their own knowledge. A knowledge base will need to be created that the user can query against to produce an answer.

Finally, the user interface will be created. This will be a texted based console that will allow a user to select an issue with their tire, and the console will either return an answer or ask a further question in order to help diagnose the problem.

Knowledge Base Creation

The expert we will be mimicking will be an automobile tire professional. To fill this knowledge base, I will be using facts and rules that deal specifically with tire repair and tire replacement, in order to keep it simple. All facts concerning car alignment and other mechanical problems that may cause tire damage (or early wear) will be ignored. The following rules will be added to the knowledge base.

Expert Rules for Tire Repair and Replacement

If tire tread <= 0.3175 cm, Then tire damaged.
If tire is flat And hole in tire, Then patch tire.
If tire is flat And Not hole or Not tear, Then air tire up.
If tire is flat and tear in tire, Then replace tire.
If tear in tire, Then flat tire.
If hole in tire, Than flat tire
If tire is damage, Then replace tire.

Unfortunately, these rules are not as cut and dry when it comes to creating a knowledge base around them using Prolog. Programming using a logical programming language, like Prolog, is very different than what most students will learn in the first couple of years at university. Although much of the inference engine rules are based on IF/THEN statements and facts, they do not work similar to how IF/THEN statements work in procedural paradigms.

Using predicates, you string together rules using formal logic, and these rules work on facts within the program. Let's say you are given the facts causes(cause, problem) and effects(cause, effect). The statement experienced problem(X,Z) :- causes(X,Y), effects(Y,Z) would mean IF experienced problem(X,Z) THEN causes(X,Y) AND effects(Y,Z). If you wanted to evaluate experienced problem(problem,Z), then Prolog would output Z=effect based on the knowledge base.

I used the rules that I defined above as a starting point for the facts and rules I would write for Prolog. I had to do this quite a few times to actually understand how to do this correctly using the new paradigm. What I ended up using was a cause and effect type rule, similar to my example above, to determine what the solution is to a particular tire problem. In order to have this rule work, I had to create a list of facts about tire problems, what causes those problems, and what would fix those particular problems.

Testing and Results

Seeing as there were not many problems for which this expert system solves for, I did not have much testing to do. I took every problem and ran it through the system many different times to see what solutions were given. I received appropriate solutions for my queries, but it did not give the option to select an intermediate problem if there were multiple issues that would cause a problem, in this case the flat tire option. It tended to always pick hole in the tire option, and selected that I patch the tire. I looked into the problem, but only figured this was a symptom of how Prolog backwards chaining works and how my facts and rule were programmed. As I did not have much experience with Prolog or any logic programming language, this problem is

still alluding me, and do not have a solution to this problem, yet.

Conclusion and Future Work

Although I did not solve the problem that I am having with my implementation of the expert system I worked on, I did learn a lot about Expert Systems and how they are created. Most of the work that a person would perform in an expert system, like the one I built, is in creating a user interface and, what I think is most difficult, engineering the knowledge base. Prolog, the logic language used, is a very different approach to programming compared to how most people program. Although not much detail was put into the knowledge base for this particular domain, there are many intricacies within the language that were not explored that could have made a much more tweaked knowledge base.

In the near future, I plan on exploring more of the prolog language within the SWI-Prolog program in order to learn how to make better defined data and rules. Once I am able to properly program in Prolog, I will be able to engineer a knowledge base with precision, and at that point I could make a more interesting expert system. With as much data and expert information out there, there is no limit to the amount of domains that expert systems can be built for.

References

Huntington, D. 2002. Back to Basics – Backward Chaining: Expert System Fundamentals. *PCAI Volume 16 Issue 4*.

Anjaneyulu, K S R. 1998. Expert Systems: An Introduction. *Resonance journal of science education Volume 3 Issue 3*. 46-58

Ogu, E. 2013. Basic Concepts of Expert System Shells and an Efficient Model for Knowledge Acquisition. *International Journal of Science and Research Volume 2 Issue 4*. 554-559

"Artificial Intelligence and Expert Systems." *Artificial Intelligence and Expert Systems*. PerfectLogic Corporation, 2013. Web. 20 July 2013. <http://www.perfectlogic.com/articles/AI/ExpertSystems/ExpertSystems.html>.

Noran, Ovidiu S. "The Evolution of Expert Systems." Griffith University School of Computing and Information Technology, n.d. Web. <http://www.ict.griffith.edu.au/noran/Docs/ES-Evolution.pdf>.

Aaby, A. "Prolog Tutorial." *Prolog Tutorial*. LIX - École Polytechnique, 2 May 1997. Web. 15 July 2013. <http://www.lix.polytechnique.fr/~liberti/public/computing/prog/prolog/prolog-tutorial.html>.