Department of Computer Science and Engineering
University of California at San Diego

CSE 255
Winter 1998

Project 3

DUE IN CLASS, TUESDAY FEBRUARY 17.

The assignment here is to write a small expert system that diagnoses faults in a simple electronic circuit. The particular circuit to diagnose is a set/reset latch with enable. This circuit is shown in Figure 7–10(a) in the attached handout from *Digital Design: Principles & Practices*, second edition, by John F. Wakerly, Prentice-Hall. The correct behavior of the circuit is specified in Figures 7–10(b) and 7–11.

You can assume that the circuit to be diagnosed is built from discrete NAND gates as shown in Figure 7–10(a). Any or all of these gates may be faulty, and any or all connections between gates may be broken. You can assume that each signal entering or leaving each NAND gate is observable.

Any other simplifying assumptions that you make in developing your expert system should be stated precisely, with a discussion of their implications. For example, you may wish to assume that two or more faults never happen simultaneously.

The expert system should be implemented in CLIPS version 6.0 and follow a standard architecture. In particular, your should distinguish between knowledge in the form of rules used for reasoning, factual knowledge organized in an object-oriented way, and temporary working knowledge. The user interface component of your software should be distinct from its knowledge-based component. Do not use CLIPS to simulate a traditional imperative program.

You should submit a printout of your software, with comments and documentation of professional quality. This documentation should be sufficient for another knowledge engineer to maintain the knowledge base. To this end, the knowledge base itself should be organized thoughtfully. For software in general, good documentation is necessary but not sufficient: it cannot alleviate bad engineering.

You should also submit sample sessions showing how a user interacts with your system and solves a problem. No fancy user interface is necessary, but you should devote thought to making the interaction convenient. Two issues to consider here are:

- Exactly what inputs should be requested from the human user about the current problem to be solved? Exactly what outputs does s/he want to see?
- Should the human take the initiative in supplying inputs, or should the system ask the human questions, or should the system operate in a "mixed initiative" mode?

In addition, you should submit short written answers to the following questions.

- How could you provide assurance that your expert system is *sound*, i.e. that it never gives a wrong answer?
- How could you provide assurance that your expert system is *complete*, i.e. that it always gives a correct answer, if sufficient information is available to do so?
- What did the experience of doing this assignment teach you about knowledge-based systems that you did not realize previously?

Guidelines for Project 3

Understanding the intended environment

Be sure to describe clearly the intended environment in which your expert system is to be used. The intended environment determines what type of user interface is appropriate. The type of user interface, in particular whether the system or the user takes the initiative in requesting or supplying information about the problem to be solved, determines what software architecture is most appropriate.

One possible environment is fully automated testing with full instrumentation. In this environment one can assume that the expert system can give test inputs of its choice to the device, and for each set of test inputs, the expert system can observe all intermediate points in the device.

Interactive trouble-shooting is a very different environment. Here an expert system should minimize the number of observations that it requires. A simple system would instruct the operator to apply a fixed set of test inputs, and ask questions about individual observables.

A more complicated system would ask the operator what test inputs he or she had chosen to apply, and then ask what measurements he or she had made.

Diagnosis versus testing

Diagnosing a device is more than just testing it and reporting whether or not it passes. Diagnosis means inferring the basic causes of directly observable failures.

Diagnosis requires a carefully chosen ontology. This ontology says what the immediately observable symptoms are, versus the possible fundamental causes.

Often the possible fundamental causes are failures in components of the device. If component-level symptoms are directly observable, then diagnosis can be done by component-level testing.

Knowledge-based programming

Expert systems are software, knowledge engineering is a special case of software engineering, and knowledge-based programming resembles conventional programming. However it is a mistake to do knowledge-based programming as if it were the same as standard imperative programming.

In knowledge-based programming, all knowledge should be made as explicit as possible. Using knowledge implicitly, i.e. without stating it explicitly, should be avoided. In a typical program the flow of control is highly constrained and sequential, and the program only works because the flow of control is correct. This means that knowledge about the task to be solved is built into the flow of control. This should be avoided in knowledge-based programming.

Remember an important general principle of software engineering: always keep software as short and simple as possible. It is always worthwhile to spend extra time in the design phase of a project simplifying the design. In the coding phase, whenever you notice that your code is repeating some pattern, you should turn the pattern into an abstraction, and make each repetition into an instantiation of the abstraction.

One common way of abstracting a pattern is to encapsulate it in a procedure (or function or subroutine). For example, the operation of asking the user a question where the answer must be in a certain set of allowed answers should be made into a reusable procedure with the set of allowed answers as a parameter.

Another common way of abstracting a pattern is to store all the variants of the pattern in a table, and to have a loop of some sort that iterates through the table constructing or executing the instances of the pattern. This method of abstraction is a special case of the idea of separating knowledge bases from inference engines.

Notes on writing

Be sure to give the reader of any report exactly what s/he has requested. Make sure you discuss explicitly each one of the issues raised in the project description.

Pages should be numbered and your report should be divided into numbered sections with section headings. At each level of organization, the most important material should come first. Section and subsection titles should be specific. In particular they should not be single words. Avoid redundancy in individual sentences and at higher levels of organization.

Every sentence should follow standard rules of grammar. For clarity, the subjects and objects of sentences should be stated explicitly. Pronouns such as *it* and *this* should usually be avoided. Pronouns are especially pernicious when they refer to abstract ideas or to entities named in a different sentence. In general, using *this* as an adjective is fine.

> Knowledge is power. (BAD) This is very true. (BETTER) This principle is very true.
>
> A latch is a sequential device. (BAD) This makes it difficult to understand. (BETTER)
> The sequential nature of latches makes them difficult for humans to understand.

It is very important to use words only with their precise standard meanings. For example "maximize" is used incorrectly twice in the sentence "Maximizing this knowledge maximizes the market segment of the technology we propose below." Note that according to the standard Unix online dictionary, the word *diagnosis* means "investigation or analysis of the cause or nature of a condition, situation, or problem." The word *diagnostic* is normally an adjective. As a plural noun, *diagnostics* means "the art or practice of diagnosis," which is not the same thing as diagnosis itself. The difference between diagnosis and diagnostics is analogous to the difference between cooking and cookery.

Avoid long compound nouns, e.g. "process root cause". Avoid acronyms, especially ones that are not explained, not reused often, or that do not have a specific meaning. Avoid jargon, e.g. "core competencies", unless it is used in a very precise sense. In technical writing, only conventional metaphors are acceptable. For example an issue can cause a "headache" but not "sore muscles." Remember that *cannot* is one word, the difference between *its* and *it's*, and that slashes are not conjunctions.

Figures should only be used if they give significant quantities of information. In figures showing system architecture, use thin arrows to show the flow of control information, and thick arrows for data or material flow.