Knowledge-Based Systems

Adrian A. Hopgood
De Montfort University, UK

INTRODUCTION

The tools of artificial intelligence (AI) can be divided into two broad types: knowledge-based systems (KBSs) and computational intelligence (CI). KBSs use explicit representations of knowledge in the form of words and symbols. This explicit representation makes the knowledge more easily read and understood by a human than the numerically derived implicit models in computational intelligence.

KBSs include techniques such as rule-based, model-based, and case-based reasoning. They were among the first forms of investigation into AI and remain a major theme. Early research focused on specialist applications in areas such as chemistry, medicine, and computer hardware. These early successes generated great optimism in AI, but more broad-based representations of human intelligence have remained difficult to achieve (Hopgood, 2003; Hopgood, 2005).

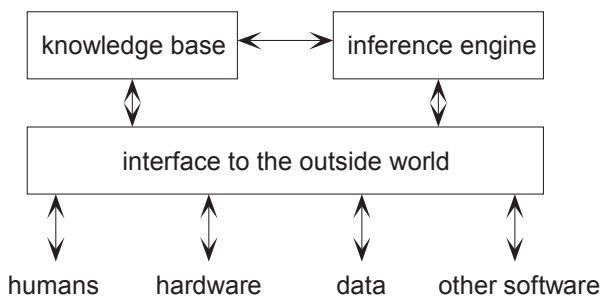
BACKGROUND

The principal difference between a knowledge-based system and a conventional program lies in its structure. In a conventional program, domain knowledge is intimately intertwined with software for controlling the application of that knowledge. In a knowledge-based system, the two roles are explicitly separated. In the simplest case there are two modules: the knowledge module is called the knowledge base and the control module is called the inference engine. Some interface capabilities are also required for a practical system, as shown in Figure 1.

Within the knowledge base, the programmer expresses information about the problem to be solved. Often this information is declarative, i.e. the programmer states some facts, rules, or relationships without having to be concerned with the detail of how and when that information should be applied. These latter details are determined by the inference engine, which uses the knowledge base as a conventional program uses a data file. A KBS is analogous to the human brain, whose control processes are approximately unchanging in their nature, like the inference engine, even though individual behavior is continually modified by new knowledge and experience, like updating the knowledge base.

As the knowledge is represented explicitly in the knowledge base, rather than implicitly within the structure of a program, it can be entered and updated with relative ease by domain experts who may not have any programming expertise. A knowledge engineer is someone who provides a bridge between the domain

Figure 1. The main components of a knowledge-based system



expertise and the computer implementation. The knowledge engineer may make use of meta-knowledge, i.e. knowledge about knowledge, to ensure an efficient implementation.

Traditional knowledge engineering is based on models of human concepts. However, it has recently been argued that animals and pre-linguistic children operate effectively in a complex world without necessarily using concepts. Moss (2007) has demonstrated that agents using non-conceptual reasoning can outperform stimulus–response agents in a grid-world test bed. These results may justify the building of non-conceptual models before moving on to conceptual ones.

TYPES OF KNOWLEDGE-BASED SYSTEM

Expert Systems

Expert systems are a type of knowledge-based system designed to embody expertise in a particular specialized domain such as diagnosing faulty equipment (Yanga, 2005). An expert system is intended to act like a human expert who can be consulted on a range of problems within his or her domain of expertise. Typically, the user of an expert system will enter into a dialogue in which he or she describes the problem – such as the symptoms of a fault – and the expert system offers advice, suggestions, or recommendations. It is often proposed that an expert system must offer certain capabilities that mirror those of a human consultant. In particular, it is often stated that an expert system must be capable of justifying its current line of inquiry and explaining its reasoning in arriving at a conclusion. This functionality can be integrated into the inference engine (Figure 1).

Rule-Based Systems

Rules are one of the most straightforward means of representing knowledge in a KBS. The simplest type of rule is called a production rule and takes the form:

```
if <condition> then <conclusion>
```

An example production rule concerning a boiler system might be:

```
/* rule1 */
if valve is open and flow is high then steam is escaping
```

Part of the attraction of using production rules is that they can often be written in a form that closely resembles natural language, as opposed to a computer language. The facts in a KBS for boiler monitoring might include:

```
/* fact1 */
valve is open

/* fact2 */
flow is high
```

One or more given facts may satisfy the condition of a rule, resulting in the generation of a new fact, known as a derived fact. For example, by applying rule1 to fact1 and fact2, fact3 can be derived:

```
/* fact3 */
steam is escaping
```

The derived fact may satisfy the condition of another rule, such as:

```
/* rule2 */
if steam is escaping or valve is stuck then outlet is blocked
```

This, in turn, may lead to the generation of a new derived fact or an action. Rule1 and rule2 are inter-dependent, since the conclusion of one can satisfy the condition of the other. The inter-dependencies amongst the rules define a network, as shown in Figure 2, known as an inference network.

It is the job of the inference engine to traverse the inference network to reach a conclusion. Two important types of inference engine can be distinguished: forward-chaining and backward-chaining, also known as data-driven and goal-driven, respectively. A KBS working in data-driven mode takes the available information, i.e. the given facts, and generates as many derived facts as it can. In goal-driven mode, evidence is sought to support a particular goal or proposition.

The data-driven (forward chaining) approach might typically be used for problems of interpretation, where the aim is to find out whatever the system can infer about some data. The goal-driven (backward chaining)