MSc

Knowledge Representation[1]

Autumn 2003

Qiang Shen[2] and Alan Smaill[3]

[1]These notes are the main part of the lecture notes for the KR module; the whole set of notes consists of these and others to be handed out incrementally.

[2]Email: Q.Shen@ed.ac.uk

[3]Email: A.Smaill@ed.ac.uk

Contents

1 Introduction 2

2 Intelligence and Representation Hypotheses 3

3 The Predicate Calculus as a Representation Language 7

4 Search in Inference Systems 16

5 Probabilistic Approach to Uncertain Reasoning 20

6 Fuzzy Logic and Approximate Reasoning 25

7 Representing Ignorance: Dempster-Shafer Theory of Evidence 36

8 Constraint Satisfaction 42

9 Reasoning Maintenance 56

10 Model-Based Reasoning 62

11 A Brief Overview of Case-Based Reasoning 72

Chapter 1

Introduction

This module provides the basis for the understanding and use of Knowledge Representation techniques in intelligent reasoning systems. The module covers notions of representation and the relationship between representation and that which is represented, along with techniques required to maintain and manipulate such representations. It loosely complements the Fundamentals of AI module.

The main topics covered are:

- Representation and the relationship between symbolic representations and represented structures.
- Logic as a representation language, and deduction as inference in a logical calculus.
- Representations of uncertain and vague concepts; approximate (fuzzy) and probabilistic reasoning.
- Meta-reasoning, inference control, reasoning maintenance, hypothesis-making and consequence exploration.
- Representations of experience and expertise; rule- and case-based reasoning.
- Representation of constraints; constraint propagation and constraint satisfaction techniques.

We welcome comments and corrections on these notes, which have benefitted from such feedback from previous students.

Chapter 2

Intelligence and Representation Hypotheses

2.1 A provisional definition of intelligence

To situate discussion of intelligent systems and the role of reasoning, let's consider the following characterisation, due to Aaron Sloman:

> Intelligence consists of the principled manipulation of representations, in order to achieve some goal.

Notice that according to this view, given some *goal*, it should be achieved by some *manipulations* (ie computations ?) applied to some *representation*. Furthermore, The idea of representation has a mental component — "X represents Y *for* Z". The definition also encompasses the idea that the manipulations are *principled* (they are structured in some way, and not simply done at random); and they are done in order to achieve some goal.

What sorts of things are these representations, and where can we find them? In natural intelligence, we get a picture as in figure 2.1.

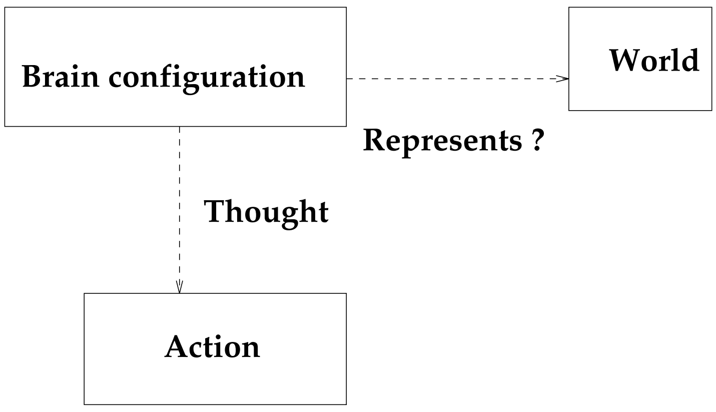


Figure 2.1: Natural Representation

Should we look to find the representations at the hardware, or at the structures and processes implemented on the hardware? *Symbolic AI* and *Connectionism* take different answers to this question.

This takes us to an old philosophical question about the nature of *meaning* — what do the representations mean? (Think for example of program code in a declarative language.) New ideas have appeared, based on (eg)

- abstract machines;
- compilation;
- computer languages.

A more explicit account of the role of representation and reaoning in intelligent systems is given in the next section.

2.2 The Knowledge Representation Hypothesis

Let's look at a claim concerning the implementation of Artificial Intelligence, made by Brian Smith in [Smi85] (as a hypothesis he adopted for his own work).

The claim is that for a system to be an implementation of an artificial intelligence, it must

1. contain a component that can be understood as linguistic (ie expressed in some language) such that
2. this component contains the knowledge of the system, and
3. this component drives the intelligent behaviour of the system.

Of these, 2 says that there should be semantics for the system; and 3 that the computation should be guided by the semantics (that deduction in the system should respect the semantics).

The verbatim quote is:

> Any mechanically embodied intelligent process will be comprised of structural ingredients that
> a) we as observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and
> b) independent of such external semantic attributes, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge.
>
> [Smi85]

Here we take a) and b) to correspond to 2) and 3) above. By *causal role* for knowledge, we mean that this is what gives the system the behaviour it has. So, for example, this feature of the system is not just some comments to the code. Note that the behaviour of the system is *independent* of the attribution of semantic content to the knowledge in the system.

We can take as an example the two programs in figure 2.2.

In the second program, knowledge is *explicitly* represented in the colour/1 clauses. These are the structures that the KR hypothesis talks of. We can interpret these clauses as propositions about the world (about football clubs, in this case). And these statements cause the program to give the behaviour it does – they are not just comments. They act as a mini Knowledge Base.

In the first program, we do *not* have this explicit knowledge in the program, so we will not count it as giving a propositional account of the knowledge involved.

Note that these programs have the same input/output behaviour. The difference between them is not the use of a specific language or data structure – it's a question of being able to understand the program in a certain way.

```
% Program 1

printColour(hearts) :- !, write('They're maroon').
printColour(hibs)   :- !, write('They're green').
printColour(X)      :- write('Beats me').

% Program 2

printColour(X) :- colour(X,Y),!,
                  write('They're '),write(Y).
printColour(X) :- write('Beats me').
colour(hearts,maroon).
colour(hibs,green).
```

Figure 2.2: 2 programs

2.2.1 Procedural and Declarative Knowledge

Recall the distinction between two sorts of knowledge, knowing *that* and knowing *how*.

The first is **declarative knowledge**: this is the sort of knowledge that refers to some domain of interest, expressed in some appropriate language (*eg* "Vaduz is the capital of Lichtenstein").

The second is **procedural knowledge**: this is the sort of knowledge we have when we are able to perform some task without necessarily being able to express this knowledge verbally (*eg* how to tie a shoelace). This also covers knowledge of how declarative knowledge should be used in a given situation – how to *use* the knowledge we have.

We have a better idea of how to represent and make use of declarative declarative knowledge than for procedural knowledge, which often appears in knowledge bases in the form of annotations to the KB, or implicitly on the form of clause ordering. Brian Smith's reflection hypothesis gives us another line of attack on this problem.

2.3 Reflection Hypothesis

If we can build a system that reasons about the world by manipulating representations of the world, then we can build a system that reasons about itself by manipulating a representation of itself.

The verbatim quote is:

> Inasmuch as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of the world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of *its own* operations and structures.
>
> [Smi85]

Let's say that a computational process could can reason about itself is *reflective*.

If we accept this hypothesis, it will help us in dealing with procedural knowledge in the following way:

procedural knowledge of how to use some declarative knowledge can be represented as declarative knowledge *about* **how** *an implementation of the procedural knowledge is to be used*; in turn this is declarative knowledge in a *reflective* system (which is capable of representing its own operations).

So far this possibility has not been followed up in earnest. Nevertheless it is an important avenue for the future development of more intelligent systems. We have seen a version of this when we looked at meta-interpreters – the control knowledge that is embodied in using a declarative object program/theory in a certain way can be turned into more declarative knowledge about the interpretation process itself. This does not completely accord with the Reflection Hypothesis, since we used non-declarative features in writing the meta-interpreter. But it does bear out the hypothesis to an extent, since it shows that a general purpose symbolic reasoning system (Prolog, in this case) *can* be given access to its own reasoning patterns, by formulating them in the same way it formulates representations of other domains.

2.4 Is the Knowledge Representation Hypothesis true

We can summarise the Knowledge Representation Hypothesis by the diagram in figure 2.3. For an intelligent process, the following components should be present.

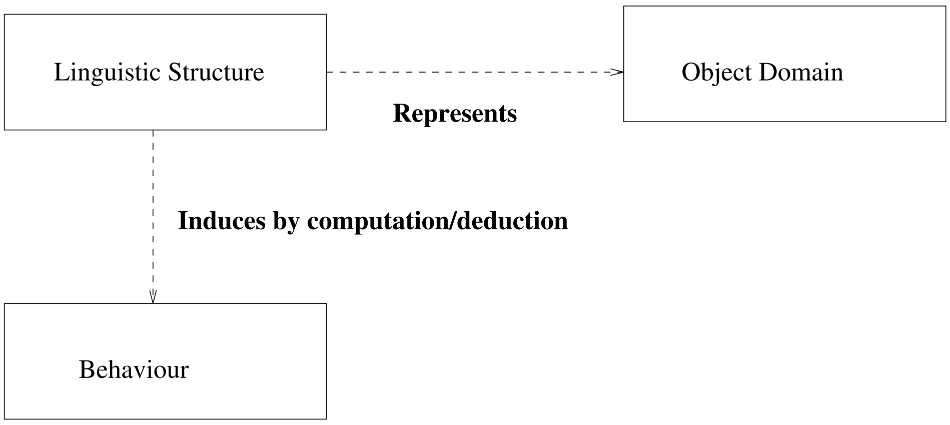


Figure 2.3: Knowledge Representation Hypothesis

To ask if knowledge is *representational* is to ask whether having knowledge of a certain domain consists in having available an *explicit* representation in the way the Knowledge Representation Hypothesis suggests; in this case reasoning can be carried out by manipulations of that representation, and intelligent behaviour can be regarded as manipulation of representations in order to achieve some goal.

The Knowledge Representation Hypothesis claims that knowledge of this sort is *essential* to an artificially intelligent process. Procedural knowledge, on the other hand, does not seem to be of this form – the ability to perform an act may be present without any explicit notions of the significance of the action.

Certain AI systems, such as neural nets, do not take such a representational view of knowledge. Such a system may be able to perform a task such as facial recognition without having any component that we can distinguish as constructing or manipulating representations.

Chapter 3

The Predicate Calculus as a Representation Language

We now consider the predicate calculus as an example of a knowledge representation formalism. A more detailed account of these notions can be found, for example, in [GN87].

We give semantics for the calculus, that is we say what the formulae of the calculus are to *mean*. We do this by giving a way of relating *linguistic formulae* to (conceptualisations of) an object domain, and in particular we say what it is for a formula to be *true* relative to the state of the domain.

The object domain is whatever we are making statements about – for mathematics this could involve numbers, geometrical objects and so on; for a medical expert system this could cover bones, cells, drugs *etc*.

Such domains could be described in many ways. For our purposes, we suppose we have a description of the domain in terms of

- A set of objects (the universe of discourse). We insist that this set is non-empty.
- Some functions from objects to objects.
- Some relations defined over the objects.

(Here functions and relations apply to an appropriate number of arguments. An n-place relation simply specifies whether it holds or not for any n input objects.)

Such a description is called a *conceptualisation* of the domain, or a *structure*.

Example

Suppose we have blocks arranged on a table as in figure 3.1.

We could take here the set of objects to be

$$\mathtt{O} = \{\mathtt{base}, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}.$$

We can take a function defined over these objects, say the function `next` defined by

$$\begin{array}{ll} next(base) = a & next(a) = c \\ next(c) = e & next(e) = d \\ next(d) = b & next(b) = base \end{array}$$

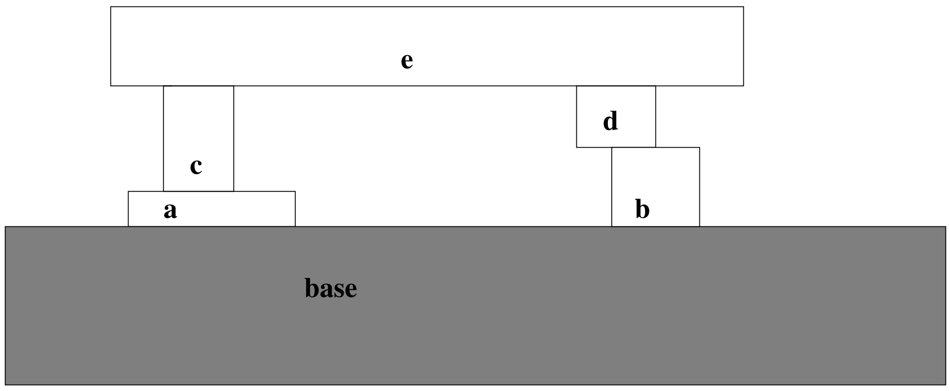


Figure 3.1: Arrangement of blocks

We can also consider the relation *on* defined as follows. It is a two place relation, which we give by specifying those pairs of objects for which the relation holds, namely

$$\mathtt{on} = \{(\mathtt{e},\mathtt{c}),(\mathtt{c},\mathtt{a}),(\mathtt{e},\mathtt{d}),(\mathtt{d},\mathtt{b}),(\mathtt{b},\mathtt{base}),(\mathtt{a},\mathtt{base})\}.$$

(An alternative way of specifying a relation is by associating one of `true` or `false` to each possible combination of input values.)

We can write $\mathtt{on}(\mathtt{e},\mathtt{c})$ for the statement $(\mathtt{e},\mathtt{c}) \in \mathtt{on}$, where $\in$ indicates set membership.

In this way, we can give many different *conceptualisations* of a situation we are interested in; it is modeled in terms of the mathematical notions of set, function and relation.

3.1 Syntax for the Predicate Calculus

We now describe the formulae of the predicate calculus that we use to talk about structures such as the one we have just discussed.

Syntactic Primitives take the form

constants:	$c_1, c_2, c_3, \ldots$
variables:	$v_1, v_2, v_3, \ldots$
function symbols:	$f_1, f_2, f_3, \ldots$
predicate symbols:	$p_1, p_2, p_3, \ldots$

Again, predicate and function symbols take some specified number of arguments. The exact form of symbols used for each of these roles of course varies in different presentations of the predicate calculus. We usually use (possibly subscripted) letters from the start of the alphabet as constants ($a, b, c_{12}, \ldots$) and from the end as variables ($u, v_{176}, x, y, \ldots$).

We now define terms. *Terms* are of one of the following forms:

- c_n (a constant)
- v_n (a variable)
- $f_n(t_1, t_2 \ldots t_k)$ where each t_i is a term, and f_n is a function symbol that takes k arguments.

Terms are going to denote *objects* in the object domain, as we shall see.

Basic Formulae are of the form $p_n(t_1, t_2 \ldots t_k)$ where p_n is a k-place predicate symbol (or of the form p_n when p_n is a zero-place predicate). We will interpret basic formulae as making assertions about the object domain. *Formulae* (also called *well-formed formulae*) are formed from basic formulae by closing up under the logical connectives:

$\neg$	negation (not)
$\wedge$	conjunction (and)
$\vee$	disjunction (or)
$\rightarrow$	implication (implies)
$\forall v_n$	universal quantification (for all)
$\exists v_n$	existential quantification (for some)

In the last two, v_n is a variable from the language. We use brackets to determine an unambiguous reading, or define precedences appropriately between the connectives. We will assume that $\neg$ binds more tightly than $\wedge$, $\vee$, which bind more tightly than $\rightarrow$.

It is important to note the phenomenon of *scope* of a quantifier. Just as in a programming language like Pascal different occurrences of an identifier do not necessarily correspond to the same program variable, so in a logical formula the meaning of a variable depends on which quantifier (if any) it is bound by. For example, in the formula

$$foo(x) \rightarrow \forall x\ (foo(x) \rightarrow \exists x\ foo(x)),$$

the x in the first occurrence of $foo(x)$ is free (*ie* bound by no quantifier), in the second occurrence it is bound by the universal quantifier, and in the third by the existential quantifier. To see the meaning of such formulas, it is a good idea to rename the *bound* variables so that each binding by a quantifier uses a different variable. Thus the formula

$$foo(x) \rightarrow \forall y\ (foo(y) \rightarrow \exists z\ foo(z))$$

has the same meaning as the formula above.

Example

Consider formulae where there is a single one-place function symbol $next$, and single two-place predicate symbol on, and constants $base, a, b, c, d, e$.

Formulae we can form are, for example:

$$on(a, base$$
$$\neg(on(a, b) \vee on(a, c))$$
$$\exists v_3\ (on(a, v_3) \rightarrow on(a, next(v_4)))$$

3.2 Interpretation

We now start to relate formulae to structures. By a *language* for the predicate calculus, we mean some choice of constants, function and predicate symbols. An *interpretation* of a such a language is a mapping form *elements of the language* to *elements of a structure*.

Suppose this interpretation is given by a function I, where the set of objects of the structure in question is S. Then we require

For a constant $\boldsymbol{c_n}$, $\mathtt{I}(\boldsymbol{c_n})$ is an object of S;
For a function symbol $\boldsymbol{f_n}$, $\mathtt{I}(\boldsymbol{f_n})$ is an function $F : S \times \ldots \times S \rightarrow S$ (with n arguments, depending on how many arguments $\boldsymbol{f_n}$ has);
For predicate symbol $\boldsymbol{P}$, $\mathtt{I}(\boldsymbol{P})$ is a relation on S (n-place depending on how many arguments P takes)

Note that we do *not* give an interpretation for the variables.

Example

To continue our running example, we could define an interpretation of the language we gave into the earlier structure as follows.

For constants, let $\mathtt{I}(\boldsymbol{base}) = \mathtt{base}$, $\mathtt{I}(\boldsymbol{a}) = \mathtt{a}$, $\mathtt{I}(\boldsymbol{b}) = \mathtt{b}$ and so on.

For the function symbol, let $\mathtt{I}(\boldsymbol{next}) = \mathtt{next}$, and map the predicate symbol by $\mathtt{I}(\boldsymbol{on}) = \mathtt{on}$

Many other interpretations are possible apart from this natural one. For example, we could take $\mathtt{I}(\boldsymbol{base}) = \mathtt{a}$, $\mathtt{I}(\boldsymbol{a}) = \mathtt{base}$, ...

3.2.1 Satisfaction

Having given an interpretation of a language we can now define what it is for a formula to be *true* in a structure under a given interpretation. (We also say in this case that the structure and interpretation together form a *model* of the formula.)

The terms of the language denote objects of the structure. We first extend the interpretation from constants to terms (without variables, for the moment) as follows.

Let
$$\mathtt{I}(\boldsymbol{f_n}(\boldsymbol{t_1}, \boldsymbol{t_2} \ldots \boldsymbol{t_k})) = \mathtt{I}(\boldsymbol{f_n})(\mathtt{I}(\boldsymbol{t_1}) \ldots, \mathtt{I}(\boldsymbol{t_k})).$$

Thus we interpret compound terms by interpreting the subterms recursively (with the constants as the base case), and applying the function which interprets the function symbol in question to the interpretation of the subterms.

We now say what it is for a formula to be satisfied (to be true) in a structure under an interpretation. Write $\mathcal{S}$ for a structure together with an associated interpretation $\mathtt{I}$. Satisfaction (written $\models$) is defined as follows.

$$\begin{array}{rll} \mathcal{S} \models \boldsymbol{p_n}(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots \boldsymbol{t_k}) & \text{if and only if} & I(\boldsymbol{p_n})(I(\boldsymbol{t_1}), \ldots I(\boldsymbol{t_k})) \\ \mathcal{S} \models \boldsymbol{A} \wedge \boldsymbol{B} & \text{if and only if} & \mathcal{S} \models \boldsymbol{A} \;\; \textit{and} \;\; \mathcal{S} \models \boldsymbol{B} \\ \mathcal{S} \models \boldsymbol{A} \vee \boldsymbol{B} & \text{if and only if} & \mathcal{S} \models \boldsymbol{A} \;\; \textit{or} \;\; \mathcal{S} \models \boldsymbol{B} \\ \mathcal{S} \models \boldsymbol{A} \rightarrow \boldsymbol{B} & \text{if and only if} & (\textit{not} \; (\mathcal{S} \models \boldsymbol{A})) \;\; \textit{or} \;\; \mathcal{S} \models \boldsymbol{B} \\ \mathcal{S} \models \neg \boldsymbol{A} & \text{if and only if} & \textit{not} \; (\mathcal{S} \models \boldsymbol{A}) \end{array}$$

This recursive definition is due to Tarski. Satisfaction is defined using the recursive structure of formulae as we defined them. In the base case (for basic formulae), we look at the interpretation of the predicate symbol, which is some relation defined over the objects, and see whether or not it holds of the objects that interpret the terms that appear in the argument places. For the propositional connectives, it can be seen that the definition corresponds to the usual truth tables for these connectives.

Example

In our example, let B be the blocks world structure together with the natural interpretation we gave.

Then

$$\begin{array}{rcl} \mathcal{B} & \models & on(c,a) \wedge on(e,c) \\ \mathcal{B} & \models & \neg(on(c,a) \rightarrow on(e,a)) \\ not(\mathcal{B} & \models & on(a,c) \vee on(c,c)) \end{array}$$

For quantified formulae, we need more machinery. The idea is that variables can be interpreted freely as any member of the objects in the domain. We must also distinguish carefully when variables are bound by some quantifier. For a full description of this, see [GN87], section 2.3.

Roughly, the idea is that

$$\begin{array}{lll} \mathcal{S} \models \forall v_n\ (\Phi(v_n)) & \text{if and only if} & \mathcal{S} \models \Phi(v_n) \text{ whatever object } v_n \text{ is interpreted as} \\ \mathcal{S} \models \exists v_n\ (\Phi(v_n)) & \text{if and only if} & \mathcal{S} \models \Phi(v_n) \text{ for some interpretation of } v_n. \end{array}$$

Here $\Phi(v_n)$ is an arbitrary formula in which v_n may occur free – this definition only uses the *free* occurrences of v_n.

Example

In our usual example, we have

$$\begin{array}{rcl} \mathcal{B} & \models & \exists v_1\ on(a,v_1) \\ \mathcal{B} & \models & \forall v_1\ \neg on(table,v_n) \end{array}$$

Some formulae are true in *all* interpretations (for *all* structures); they are called *valid*. For example:

$$\begin{array}{rcl} \mathcal{M} & \models & foo(a) \rightarrow foo(a) \\ \mathcal{M} & \models & (\forall v_3\ baz(v_3)) \rightarrow baz(b) \end{array}$$

are true for any structures and interpretations.

A formula is said to be *consistent* is it is true in *some* structure and interpretation. For example, $foo(a,b) \wedge \neg(foo(b,a))$ is consistent since it has a model given by

- objects $\{1,2\}$
- relation $<$
- interpretation $\mathtt{I}(a) = 0, \mathtt{I}(b) = 1, \mathtt{I}(foo) = <$.

For another example, consider $p(a) \wedge \exists v_1\ \neg p(v_1)$.

This has a model with domain $\{0,1\}$ and the one-place relation r for which $r(0)$ is true and $r(1)$ is false. Interpret a as 0 and p as r. Letting $\mathcal{S}$ stand for this interpretation, we have from our definition of satisfaction:

$$\mathcal{S} \models p(a) \wedge \exists v_1\ \neg p(v_1) \leftrightarrow\ \mathcal{S} \models p(a) \textit{ and } \mathcal{S} \models \exists v_1\ \neg p(v_1)$$

Now

$$\mathcal{S} \models p(a)$$

since the interpretation of p (namely `r`) holds of the interpretation of a (namely `0`). Similarly the second half of the conjunction can also be shown to hold using our definitions.

Example

We can extend these notions to several formulae: we require that *all* the formulae are satisfied.

For example, the formulae

$$\begin{array}{rlcl} \forall x \forall y \forall z & x * (y * z) & = & (x * y) * z \\ \forall x \forall y & x * y & = & y * x \\ \forall x & x * 1 & = & x \end{array}$$

have a model given by taking

- objects: the integers
- function: addition
- interpretation $\mathtt{I}(\mathbf{1}) = \mathtt{0}, \mathtt{I}(*) = \mathtt{addition}$

3.3 Why semantics?

We have described how formulae of the predicate calculus can be related to structures, and what it is for a formula to be *satisfied* in a structure under an interpretation (equivalently, for a structure and interpretation to form a model of a formula).

In general we cannot implement this semantics – if the domains are infinite, the predicates and functions may well be uncomputable. Even if we just look at small finite structures, the computational complexity will defeat us.

However, the semantics is useful in the following ways. It allows us to

- validate inference rules;
- show that derivations do *not* exist (*ie* give counterexamples);
- characterise other inference systems;
- compare formalisms (*eg* database languages).

3.4 Inference Systems

Our semantics gives us a notion of *logical consequence*.

We say that a formula $\boldsymbol{G}$ is a logical consequence of formulae $\boldsymbol{F_1}, \boldsymbol{F_2} \ldots \boldsymbol{F_n}$ (meaning that it follows logically) if and only if, for all structures with interpretation $\mathcal{S}$,

$$\textit{if } \mathcal{S} \models \boldsymbol{F_1} \textit{ and} \ldots \textit{and } \mathcal{S} \models \boldsymbol{F_n}, \textit{ then } \mathcal{S} \models \boldsymbol{G}.$$

When this is true, we write

$$\boldsymbol{F_1}, \boldsymbol{F_2} \ldots \boldsymbol{F_n} \models \boldsymbol{G}.$$

Note than the symbol $\models$ is overloaded – we use it both for logical consequence and for satisfaction.

So to ask if $\boldsymbol{G}$ follows from some knowledge base in the predicate calculus $\boldsymbol{K}$ is to ask if

$$\boldsymbol{K} \models \boldsymbol{G}.$$

The intuition is that we assume that the statements in the knowledge base are true in the object domain, and we want to know if the conclusion is true as well. We need an effective way of computing this - using the semantics directly is not a feasible approach. Instead we need to use an inference system.

3.4.1 Inference Rules

An inference system uses repeated applications of inference rules, for example the rule modus ponens:

$$\frac{\mathtt{P} \qquad \mathtt{P} \rightarrow \mathtt{Q}}{\mathtt{Q}}$$

The general form is

- a set of sentence patterns called *conditions* (P and $P \rightarrow Q$ above)
- a patterns called *conclusion* (Q above)

When we have formulae that match the conditions, we can *infer* a formula that match the conclusions. Inference rules are intended to take us from true conditions to true conclusions. An inference rule with no conditions is usually called an *axiom* – these are therefore intended to be true statements.

Some other examples are:

$$\frac{\mathtt{A} \rightarrow \mathtt{B} \qquad \neg\mathtt{B}}{\neg\mathtt{A}}$$

$$\frac{\forall \mathtt{v_1}\ \mathtt{P(v_1)}}{\mathtt{P(t)}} \text{ for any term } t$$

$$\frac{\mathtt{P_1} \vee \ldots \vee \mathtt{P_n} \vee \mathtt{R} \qquad \mathtt{Q_1} \vee \ldots \vee \mathtt{Q_m} \vee \neg\mathtt{R}}{\mathtt{P_1} \vee \ldots \vee \mathtt{P_n} \vee \mathtt{Q_1} \vee \ldots \vee \mathtt{Q_m}}$$

(the second rule is called *instantiation*, and the last is a version of the *resolution* rule of inference.)

A *derivation* of a formula $\mathtt{G}$ from formulae $\mathtt{F_1}, \mathtt{F_2} \ldots \mathtt{F_n}$ consists of a (finite) sequence of formulae

$$\begin{array}{c} \mathtt{D_1} \\ \mathtt{D_2} \\ \vdots \\ \mathtt{D_r} \end{array}$$

where $\mathtt{D_r} = \mathtt{G}$ and each $\mathtt{D_i}$ is either

- one of $\mathtt{F_1}, \mathtt{F_2} \ldots \mathtt{F_n}$ (called axioms),
- the conclusion of an inference rule, with conditions occurring earlier in the list.

Some thought shows that we could use a *tree* data structure here, rather than sequence, where instances of the inference rules correspond to nodes in a tree. (This may require duplication of some formulas in the list in oreder to construct a correponding tree.)

If we can find appropriate inference rules, we can automate the process of searching for such derivations.

Example Given

$$\forall \mathtt{v_1}\ \mathtt{red(v_1)}, \qquad \mathtt{red(a)} \rightarrow \mathtt{bow\_group(a)}$$

we give a derivation for $\mathtt{bow\_group(a)}$ using the instantiation and modus ponens rules above.

1	$\forall \mathtt{v_1}\ \mathtt{red(v_1)}$	*axiom*
2	$\mathtt{red(a)}$	*instance 1*
3	$\mathtt{red(a)} \rightarrow \mathtt{bow\_group(a)}$	*axiom*
4	$\mathtt{bow\_group(a)}$	*modus ponens 2,3*

Notice the formulae are labelled with justifications corresponding to our definition of a derivation.

Given an inference system I with a number of rules of inference, we use the notation

$$\mathtt{F_1} \ldots \mathtt{F_N} \vdash_I \mathtt{G}$$

to indicate that there is some derivation of the conclusion using the rules of inference of the system I in this way, taking $\mathtt{F_1} \ldots \mathtt{F_n}$ as axioms. We write simply $\vdash$ when the choice of inference system is clear.

An *inference procedure* is a way of selecting which rule to apply in the course of constructing a derivation, where several rules are applicable. For example, the Prolog strategy of searching depth-first and always resolving the left-most literal first is an inference procedure.

3.4.2 Relating inference rules and semantics

Our semantics gives us a way of validating proposed inference rules. For example, we don't want something like

$$\frac{\mathtt{A}}{\neg \mathtt{A}}$$

to be an inference rule. (Why not?)

We say an inference rule

$$\frac{\mathtt{A_1} \ldots \mathtt{A_n}}{\neg \mathtt{B}}$$

is *sound* if for every set of formulae $\mathtt{F_1} \ldots \mathtt{F_n}, \mathtt{G}$ that match the inference rule, we have

$$\mathtt{F_1} \ldots \mathtt{F_n} \models \mathtt{G}$$

We say an inference system containing several inference rules is sound if every inference rule in the system is sound.

In a sound inference system, we know that the conclusion of any derivation will follow according to our semantics. Conversely, we say an inference system is *complete* if whenever a formula (with no free variables) follows semantically, there is some derivation that has the formula as its conclusion. (It would not make sense to expect that a formula with free variables like `even(x)` would be decided in this way.)

Thus an inference system `I` is complete iff

$$\text{if } \mathtt{F_1} \ldots \mathtt{F_n} \models \mathtt{G} \text{ then } \mathtt{F_1} \ldots \mathtt{F_n} \vdash_I \mathtt{G}$$

Complete inference systems for the predicate calculus are well-known (for details, see [Gal86]).

The resolution rule on its own is *not* complete[1]; for example there is no derivation of the formula $\mathtt{a} \vee \neg \mathtt{a}$ where there are no input clauses; yet the formula is always satisfied.

There is a further distinct notion of completeness that applies to search procedures rather than to inference systems. We say a search procedure is *complete* if whenever there is a derivation of some formula, then the search procedure will find a derivation for it. The usual Prolog search procedure is not complete, since it may well loop even when there is a derivation using resolution.

There is a stronger notion, that of a *decision procedure* – this is an algorithm which takes formulas as input, which always terminates, indicating whether or not the input formula has a derivation or not.

Looking at the various inference systems that exist for the predicate calculus, and for susbsystems of the predicate calculus, such as the Horn clauses that Prolog uses, we may say that

- **good news:** They give us an approach to automating inference, in a way we can relate to the semantics.
- **bad news:** The task of automating such inference is *impossible* in general. The relation between formulae we have defined

  $$\mathtt{F_1} \ldots \mathtt{F_n} \models \mathtt{G}$$

  is *not* decidable – there is no algorithm that always terminates and which correctly determines whether one formula follows from other formulae in the standard predivate calculus. This means also that there is no decision procedure for derivability in the predicate calculus.
- **bad news:** Even if we look at a small part of the calculus, just looking at the connectives $\wedge$, $\vee$, $\neg$ and throwinq away quantifiers, and so just using so-called *propositional reasoning* the problem is still computationally intractable. (Technically, determining if a formula in this form is valid is known to be a co-NP-complete problem.)

We will see later how the logic-based approach attempts to soften the blow of the last two points.

[1]However resolution *is* complete for goals of the form of a contradiction.

Chapter 4

Search in Inference Systems

Given a particular inference system, and a goal formula, the system can give rise to a search procedure in many ways.

The first distinction is between using the inference rules *bottom up* and *top down*. In the *bottom up* case, we construct a derivation starting with some axioms and successively applying inference rules until the goal formula is found. In the *top down* case we start with the goal formula, and apply the inference rules backwards, to see what subgoals we have to prove in order to establish the initial goal. We recurse until each branch has reached an axiom.

4.1 Hilbert systems

For an example of top down and bottom up use, consider the modus ponens rule:

$$\frac{\mathtt{P} \qquad \mathtt{P} \to \mathtt{Q}}{\mathtt{Q}}\textit{modus ponens}$$

In a bottom up system, we use this to construct a derivation for $\mathtt{Q}$ given derivations for $\mathtt{P} \to \mathtt{Q}$ and $\mathtt{P}$. In a top down system, we use it in the opposite direction: to prove $\mathtt{Q}$, we search for proofs of $\mathtt{P} \to \mathtt{Q}$ and $\mathtt{P}$. Notice that in this top down use, the subgoals are not completely determined by the goal: given the goal $\mathtt{Q}$, we can use this rule for *any choice* of the formula $\mathtt{Q}$ — eg we could prove $\mathtt{p} \to \mathtt{p} \to \mathtt{p}$ by showing

$$(\mathtt{p} \to \mathtt{q} \to \mathtt{r}) \to (\mathtt{p} \to \mathtt{q} \to \mathtt{r})$$

and also from

$$((\mathtt{p} \to \mathtt{q} \to \mathtt{r}) \to (\mathtt{p} \to \mathtt{q} \to \mathtt{r})) \to (\mathtt{p} \to \mathtt{p} \to \mathtt{p}).$$

This means that the search for derivations has an infinite branch point here, which is a Bad Thing.

Here are some axioms which, together with modus ponens, give an inference system for formulas that just use the connective $\to$. (This does not prove all the true formulas according to our semantics, by the way, but only some of them.) Any formulas which match the following patterns are axioms.

$$\begin{array}{ll} \mathtt{A} \to \mathtt{B} \to \mathtt{A} & \textit{Ax1} \\ (\mathtt{A} \to \mathtt{B} \to \mathtt{C}) \to (\mathtt{A} \to \mathtt{B}) \to (\mathtt{A} \to \mathtt{C}) & \textit{Ax2} \end{array}$$

This is part of an inference system that was used by logicians (including Hilbert) before computational and search issues were important for inference systems. You may like to convince yourself that these two axioms together with modus ponens are not a good basis for the automation of proof search. Here is a proof in this system of $\mathtt{p} \to \mathtt{p}$. (We take $\to$ to associate to the right here.)

1	$(\mathtt{p} \to (\mathtt{p} \to \mathtt{p}) \to \mathtt{p}) \to (\mathtt{p} \to (\mathtt{p} \to \mathtt{p})) \to (\mathtt{p} \to \mathtt{p})$	*Ax2*
2	$\mathtt{p} \to (\mathtt{p} \to \mathtt{p}) \to \mathtt{p}$	*Ax1*
3	$(\mathtt{p} \to (\mathtt{p} \to \mathtt{p})) \to (\mathtt{p} \to \mathtt{p})$	*modus ponens 1,2*
4	$\mathtt{p} \to (\mathtt{p} \to \mathtt{p})$	*Ax1*
5	$\mathtt{p} \to \mathtt{p}$	*modus ponens 4,3*

This proof can be constructed from 1 to 5 (bottom up) or from 5 to 1 (top down). Either way, some thought will show that there are many possbile choices allowed by the inference system, even for such a simple formula.

4.2 Sequent Calculus

We now give a different inference system which captures the same logic as in the previous case, but where the inference system is more helpful for purposes of proof search.

Instead of looking at formulae, we manipulate *sequents*, that is a pair of two sets of formulae. We use the symbol $\Longrightarrow$ to separate the two sets in the pair. So a sequent takes the form

$$\mathtt{F}_1, \ldots \mathtt{F}_n \Longrightarrow \mathtt{G}_1, \ldots \mathtt{G}_m$$

Taking *sets* rather than *lists* of formulae means that we do not distinguish the order of formulae, nor how often a formula occurs.

The intended reading of a sequent is that if all the formulae on the left of the sequent are true, then at least one of the formulae on the right is true. Thus the single formula $\mathtt{F}$ corresponds in the sequent calculus to the sequent $\Longrightarrow \mathtt{F}$ with an empty set of formulae on the left and a single formula on the right.

The axioms are all sequents

$$\mathtt{F}_1, \ldots, \mathtt{H}, \ldots \mathtt{F}_n \Longrightarrow \mathtt{G}_1, \ldots, \mathtt{H}, \ldots \mathtt{G}_m$$

where some formula H appears on the left and on the right.

Here are two inference rules:

$$\frac{\mathtt{F}, \mathtt{F}_1 \ldots \mathtt{F}_n \Longrightarrow \mathtt{G}, \mathtt{H}_1, \ldots, \mathtt{H}_m}{\mathtt{F}_1 \ldots \mathtt{F}_n \Longrightarrow \mathtt{F} \to \mathtt{G}, \mathtt{H}_1, \ldots, \mathtt{H}_m} \quad \textit{impI}$$

$$\frac{\mathtt{F}_1 \ldots \mathtt{F}_n \Longrightarrow \mathtt{F}, \mathtt{H}_1, \ldots, \mathtt{H}_m \qquad \mathtt{G}, \mathtt{F}_1 \ldots \mathtt{F}_n \Longrightarrow \mathtt{H}_1, \ldots, \mathtt{H}_m}{\mathtt{F} \to \mathtt{G}, \mathtt{F}_1 \ldots \mathtt{F}_n \Longrightarrow \mathtt{H}_1, \ldots, \mathtt{H}_m} \quad \textit{impE}$$

Now, while we now have two inference rules instead of one, there are many fewer choices to be made when these rules are used backwards. The proof of $\mathtt{p} \to \mathtt{p}$ (taken as $\Longrightarrow \mathtt{p} \to \mathtt{p}$) falls out quickly from a blind search. Nearly as simple is the proof of $\Longrightarrow \mathtt{p} \to \mathtt{q} \to \mathtt{p}$:

1	$\mathtt{q}, \mathtt{p} \Longrightarrow \mathtt{p}$	*Ax*
2	$\mathtt{p} \Longrightarrow \mathtt{q} \to \mathtt{p}$	*impI 1*
3	$\Longrightarrow \mathtt{p} \to \mathtt{q} \to \mathtt{p}$	*impI 2*

4.3 Search Spaces for Top Down Inference Systems

Given that our problem is to find derivations in some inference system, we now describe a search space that is defined by the rules. For efficient inference, we want the search space to be as small as possible (but still for every formula that should be derivable to have a derivation).

Given inference rules (as for the sequent calculus) we can describe a search space as a tree of derivations as follows.

The formula we want to prove is set up as the initial goal. Its immediate children in the search space are all derivations got by application of some rule of inference to the initial goal. (Note that some rules of inference may apply backwards in several ways, for example, the modus ponens rule. Each different application gives a different node in the search space.

Subsequent nodes are defined the same way: the children of a node are all derivations got by extending the parent derivation backwards by some inference rule application. A complete proof is one where all the leaves are axioms – these are the success nodes in the search space.

This is a general description of a search space generated by a set of inference rules. Usually we are not attached to any inference system in particular – we will change to a different inference system that has proofs for the same set of formulas, if the new inference system has better search properties.

Here we need to know about the inference rules orI and andE:

$$\frac{\mathtt{F_1 \ldots F_n} \Longrightarrow \mathtt{F, G, H_1, \ldots, H_m}}{\mathtt{F_1 \ldots F_n} \Longrightarrow \mathtt{F \lor G, H_1, \ldots, H_m}} \quad \textit{orI}$$

$$\frac{\mathtt{F, G, F_1 \ldots F_n} \Longrightarrow \mathtt{H_1, \ldots, H_m}}{\mathtt{F \land G, F_1 \ldots F_n} \Longrightarrow \mathtt{H_1, \ldots, H_m}} \quad \textit{andE}$$

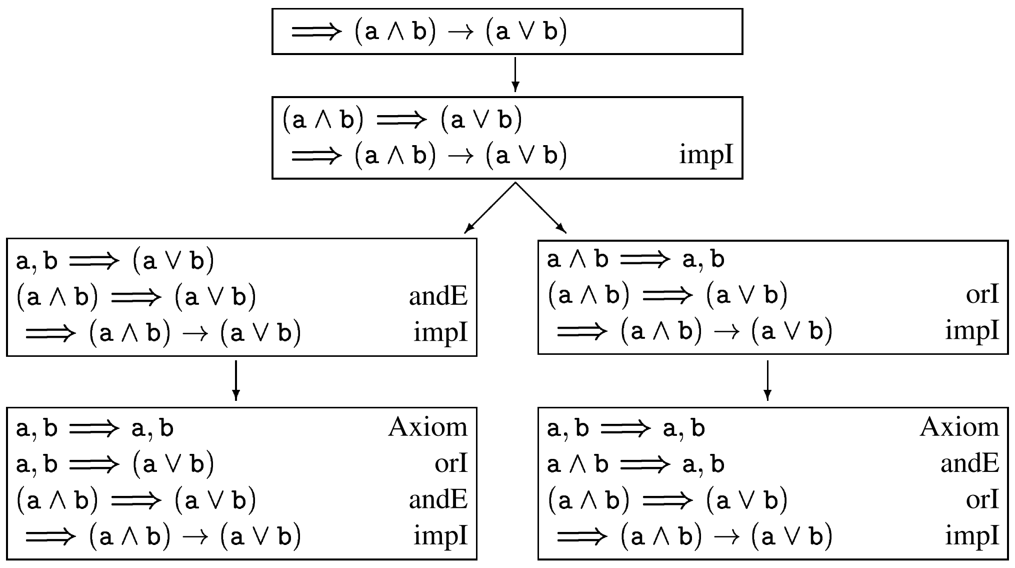


Figure 4.1: Search Space

An example of search space (for the sequent $[\,] \Longrightarrow (\mathtt{a} \land \mathtt{b}) \to (\mathtt{a} \lor \mathtt{b})$ in the sequent calculus) is shown in Figure 4.1.

Chapter 5

Probabilistic Approach to Uncertain Reasoning

5.1 Introduction

Traditionally, uncertain knowledge representation and reasoning has been treated as a problem of mathematical probability. Probability theory is indeed a popular tool in AI problem solving. Its roots were dated in the 17th century when it was invented to increase calculated odds of winning in gambling, though the idea of probability was first framed even earlier in the 16th century.

This chapter looks at the basic issues involved in probability theory, including the concepts of sample space, prior and posterior probabilities, and joint and conditional probabilities. It also introduces the Bayes theorem which supports inexact inference, and which has been widely applied in various problem domains, covering science, engineering, business, economics and medicine, etc. A comprehensive introduction to probability theory and Bayesian statistics can be found in [Sav54].

5.2 Basics of the Theory

Probability theory concerns with the properties of random events. A random event e represents a subset of possible outcomes of a certain experiment. Different outcomes may result from different experiments which are carried out under the same conditions. The set of all possible outcomes of one experiment is called a *sample space* and is denoted Ω hereafter. The theory proposes the existence of a number $p(e), \forall e \subseteq \Omega$, which is regarded as the likelihood of e occurring from an experiment. It is this number $p(e)$ that is referred to as the *probability* of e.

The present discussion addresses the probability of a given event from a discrete set of possible outcomes. The ideas covered can be extended to coping with continuous sample spaces, via the use of so-called probability distribution functions. This is however beyond the scope of this introductory material.

To start with, probability represents the expected likelihood of a given event from some experiment and is required to satisfy the following axioms:

- $p(e_i) \in [0, 1],\ \ e_i \subseteq \Omega,\ \ p(\phi) = 0,\ \ p(\Omega) = 1$
- $p(e_i \cup e_j) = p(e_i) + p(e_j),\ \ e_i \cap e_j = \phi,\ \ e_i, e_j \subseteq \Omega$

where ϕ stands for empty set.

The first axiom above indicates that a definitely certain event is assigned the probability value of 1, an impossible event assigned probability 0, and other uncertain events take a likelihood value within the range of 0 to 1. The second states that the probability of two mutually exclusive events is simply the sum of their individual probabilities. These axioms put the probability theory on a sound theoretical basis. However, they do not specify the probabilities of any uncertain event. The actual determination of such probability values is done by other methods. Following the experimental approach, probabilities are defined in the limit of an infinite number of trials:

$$p(e) = lim_{N \to \infty} \frac{n(e)}{N}$$

where $n(e)$ denotes the number of event e occurring in a total of N experiments. When experimental trials are not possible, probabilities may be subjectively assigned as degrees of belief.

In many applications of the theory, it is needed to consider the combinations of various random events. For instance, in medical diagnosis, it is often useful to see what symptoms may occur together. Alternatively, in the case of throwing a perfect die, it might be interesting to predict what would be the probability of the resulting number being either of the three even numbers. For the first case, which involves multiple events jointly occurring, the likelihood of such a combined event is called the *joint probability*. This is determined by finding the intersection of all the relevant events concerned. In particular, the joint probability of two events e_i ande_j is given by

$$p(e_i \cap e_j) = \frac{n(e_i \cap e_j)}{n(e)}$$

where $n(e)$ denotes the cardinality (i.e. number of elements) of the entire sample space Ω, and $n(e_i \cap e_j)$ denotes the number of elements in Ω which are shared by the events e_i and e_j. Note that this assumes that the individual elements of Ω are equally probable.

Consider a simple example, of throwing a fair die. The joint probability of two events: e_1 indicating the number achieved being an even number and e_2 indicating the number being divisible by 3, is

$$p(e_1 \cap e_2) = \frac{1}{6}$$

since $\Omega = \{1, 2, ..., 6\}, e_1 = \{2, 4, 6\}$, and $e_2 = \{3, 6\}$.

In some cases, there exist events that do not affect each other in any way. Such events are termed *independent events*. For two independent events their joint probability is simply the product of their individual probabilities, that is,

$$p(e_i \cap e_j) = p(e_i)p(e_j)$$

For the above example, there is no interdependency between the achieved number being even and that being divisible by 3, thus

$$p(e_1 \cap e_2) = p(e_1)p(e_2) = \frac{n(e_1)}{n(e)} \frac{n(e_2)}{n(e)} = \frac{3}{6}\frac{2}{6} = \frac{1}{6}$$

Now, consider the probability of either of a given number of events occurring. For simplicity, examine the case where either of two random events may take place. In this case, the probability is calculated by

$$p(e_i \cup e_j) = p(e_i) + p(e_j) - p(e_i \cap e_j)$$

The subtraction of the joint probability is added in order to avoid counting those elements shared by the two events twice.

Clearly, random events that are not mutually exclusive influence each other. Knowing the occurrence of one event may lead to the revision of the prior probabilities of some other events. The probability of an event e_i occurring, given that an event e_j has already taken place, is referred to as the *conditional probability* of e_i given e_j, which is defined by

$$p(e_i|e_j) = \frac{p(e_i \cap e_j)}{p(e_j)}, \ \ p(e_j) \neq 0$$

When e_i and e_j are independent of each other, $p(e_i|e_j) = p(e_i)$, and hence $p(e_i \cap e_j) = p(e_i)p(e_j)$, which has been established before.

Back to the die example. Suppose that the achieved number after rolling the die is divisible by 3 (event b). What would be the probability of event a that represents the achieved number being 3? It follows from the above definition that

$$p(a|b) = \frac{\frac{n(a\cap b)}{n(e)}}{\frac{n(b)}{n(e)}} = \frac{n(a \cap b)}{n(b)} = \frac{1}{2}$$

This shows that the expected likelihood of getting the number 3 has been revised from the original prior probability of $\frac{1}{6}$ to a probability of $\frac{1}{2}$.

Note that to avoid terminological confusion, when used in association with a conditional probability, the normal probability is interchangeably called the unconditional or absolute probability in the literature.

5.3 Bayes' Theorem and Bayesian Inference

What a conditional probability allows is to obtain the probability of event a given that event b has occurred. In many applications of probability theory, e.g. in medical diagnosis, it is often the reverse of this situation that is required. For instance, having observed the symptom of someone having a running nose, what would be the likelihood of which the person has been suffering from flu? In more general terms, what is the probability of an earlier event given that some later one has occurred? Such a probability is referred to as *a posteriori* probability (whilst those discussed above are termed *a priori* probabilities), and the method for solving such reverse problems is the well-known Bayes theorem:

$$p(h|e) = \frac{p(h)p(e|h)}{p(e)}$$

where $p(h|e)$ denotes the probability of event (say, hypothesis) h being true given event (say, evidence) e, $p(h)$ is the (unconditional) probability of h being true, $p(e|h)$ denotes the probability of event e (say, being observed) when h is true, and $p(e)$ is the (unconditional) probability of e.

Given the definition of conditional probability, it is not difficult to derive the above theorem. This is due to the fact that the joint probability

$$p(h \cap e) = p(h|e)p(e) = p(e|h)p(h)$$

This theorem, however, shows the way in which prior probabilities can be used to interpret the present situation. If historical information is rich enough, such statistical information can be used to estimate the prior probabilities which, in turn, can be used to determine the likelihood of some hypothesis being true, given some evidence about the problem at hand.

Bayes theorem supports inference under uncertain circumstances, typically by exploring domain knowledge structured in production form. For instance, given a rule like

$$\text{IF evidence } e \text{ THEN hypothesis } h$$

the theorem can then be employed to provide the probability of the hypothesis h given the evidence e. Such use could be applied for a sequence of rules like

$$\text{IF } e_1 \text{ THEN } e_2, \text{ IF } e_2 \text{ THEN } e_3, \text{ ..., IF } e_{N-1} \text{ THEN } e_N, \text{ IF } e_N \text{ THEN } h$$

In addition to helping establish a probability *for* a hypothesis being true given some evidence, the theorem can also support establishing a probability *against* the hypothesis being true for the same evidence. This is implemented by straightforward substitution of h with $\bar{h}$:

$$p(\bar{h}|e) = \frac{p(\bar{h})p(e|\bar{h})}{p(e)}$$

where $p(e|\bar{h})$ denotes the probability of e being true when h is false, and $p(\bar{h})$ is the (unconditional) probability of h being false, with $\bar{h}$ standing for the complement of h.

It is easy to see that the term $p(e)$ in the theorem can be rewritten as

$$p(e) = p(e|h)p(h) + p(e|\bar{h})p(\bar{h})$$

This may help perform certain inference tasks.

5.4 Example Use of Bayes' Theorem

To illustrate the basic ideas of using Bayes theorem to support inexact inferences, consider a simple example. Parents often give their child a temperature check if the child has a running nose. Test results are classified as having a fever (above 38 ^oC), which suggests the child may have got flu, and not having a fever, which suggests the child does not have flu. Suppose that a child has got a fever. How likely has he caught a flu? To apply the Bayes theorem, assume that the following prior probabilities have been obtained from a (large) group of children having running noses:

$p(FLU) = 0.65$, meaning that 65% of children have flu (if they have running noses)

$p(FEVER|FLU) = 0.95$, meaning that 95% of children who have flu may have a fever

$p(FEVER|\overline{FLU}) = 0.15$, meaning that 15% of children who do not have flu may have a fever

From this, it is known that

$p(\overline{FLU}) = 1 - p(FLU) = 0.35$, meaning that 35% of children do not have flu

(and that

$p(\overline{FEVER}|\overline{FLU}) = 1 - p(FEVER|\overline{FLU}) = 0.85$, meaning that 85% of children who do not have flu do not have a fever although this result is not necessary for finding the answer to the present question). The answer is itself computed by

$$\begin{aligned} p(FLU|FEVER) &= \frac{p(FLU)p(FEVER|FLU)}{p(FEVER|FLU)p(FLU) + p(FEVER|\overline{FLU})p(\overline{FLU})} \\ &= \frac{0.65 \times 0.95}{0.95 \times 0.65 + 0.15 \times 0.35} \approx 0.92 \end{aligned}$$

This result can be interpreted such that slightly more than 90% of the children having a running nose and a fever are suffering from flu.

5.5 Discussions

Probability theory is a very powerful tool for representing and reasoning with uncertain problems where information is random. This is evident from its widepsread applications. However, any successful use of this theory requires a fundamental assumption in that the sample space is well defined and that the prior probabilities can be obtained from a set of past data. Yet, providing reliable data and thence estimating the probabilities may present challenging issues for real-world applications.

In building an intelligent system, e.g. for disease diagnosis, various things may stop the attempt to use a good probabilistic basis for handling uncertain information. In the medical context, there are a large number of diseases and a very large number of bits of evidence to consider. It is almost impossible to collect good data on the association between symptoms and diseases from medical records. Diseases come and go, there is a high error rate in the recording of symptoms, and so on. It is, moreover, of little use trying to use some experts' intuitive assessment of the probabilistic figures; experiments have shown that such estimates vary very widely [BS84]. Also, using the Bayesian approach may be computationally very expensive, although work on graphical models (e.g. Bayesian nets [Pea88]) helps reduce such difficulties. That said, due to its formal mathematical foundation, probability theory and techniques based upon this theory are still the most popular in building AI systems.

Chapter 6

Fuzzy Logic and Approximate Reasoning

6.1 Introduction

A distinct approach to coping with reasoning under uncertain circumstances is to use the theory of fuzzy sets [Zad65]. The main objective of this theory is to develop a methodology for the formulation and solution of problems that are too complex or ill-defined to be suitable for analysis by conventional Boolean techniques. It deals with subsets of a universe of discourse, where the transition between full membership of a subset and no membership is gradual rather than abrupt as in the classical set theory. Such subsets are called fuzzy sets.

Fuzzy sets arise, for instance, when mathematical descriptions of ambiguity and ambivalence are needed. In the real world, the attributes of, say, a physical system often emerge from an elusive vagueness or fuzziness, a readjustment to context or an effect of human imprecision. The use of the "soft" boundaries of fuzzy sets, namely the graded memberships, allows subjective knowledge to be utilised in defining these attributes. With the accumulation of knowledge the subjectively assigned memberships can, of course, be modified. Even in some cases where precise knowledge is available, fuzziness may be a concomitant of complexity involved in the reasoning process.

The adoption of fuzzy sets helps to ease the requirement for encoding uncertain domain knowledge. For example, labels like $small, medium$ and $large$ have an intuitive appeal to represent values of some physical attributes. However, if they are defined with a semantic interpretation in terms of crisp intervals, such as

$$small = \{x | x > 0, x \ll 1\}$$

$$medium = \{x | x > 0, x \approx 1\}$$

the representation may lead to rather non-intuitive results. This is because, in practice, it is not realistic to draw an exact boundary between these intervals. When encoding a particular real number it may be difficult to decide which of these the number should, or should not, definitely belong to. It may well be the case that what can be said is only that a given number belongs to the $small$ set with a possibility of A and to the $medium$ with a possibility of B. The avoidance of this problem requires gradual membership and hence the break of the laws of excluded-middle and contradiction in Boolean logic. This forms the fundamental motivation for the development of fuzzy logic.

6.2 Basics of the Theory

Speaking more formally, let X be an ordinary (crisp) set of objects, called the universe (of discourse), whose generic elements are denoted by x. A *fuzzy set*, A, in X is a set of pairs

$$A = \{(x, \mu_A(x)) | \mu_A(x) \in [0, 1], x \in X\}$$

where $\mu_A(x)$ is called the grade of membership of x in A, or sometimes, the membership distribution/function of A. Intuitively, the closer the value of $\mu_A(x)$ is to 1, the more x belongs to A. When $\mu_A(x)$ is restricted to the values 0 and 1, the fuzzy set A degenerates to an ordinary set and its membership distribution becomes the characteristic function of the classical set.

Fuzzy sets defined on the real number line are most often used, these fuzzy sets are termed fuzzy numbers. As a simple example, an uncertain concept "approximately 4" can be described by a fuzzy number, as shown in figure 6.1. In this figure, $\mu_{Four}(x)$ denotes the degree of membership to which a real number x ($x \in R$) is regarded to be part of "approximately 4". It has a natural appeal that 4 belongs to this vague set with a full membership $\mu_{Four}(4) = 1$ and that, for instance, 3 belongs to it with a membership value of 0.8, whereas 10 does not belong to this set, i.e. $\mu_{Four}(10) = 0$.

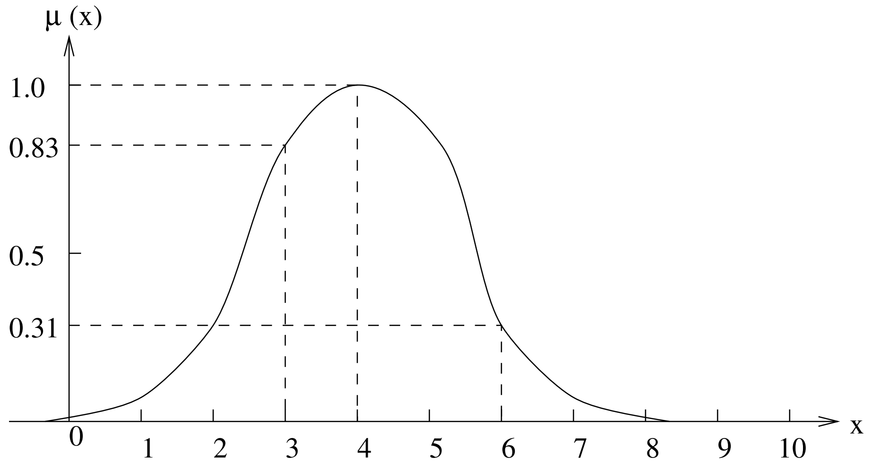


Figure 6.1: Approximately four

In fuzzy logic, the truth value of a statement is linguistic (and no longer Boolean), of the form *very true, true, more or less true, not very false, false*. These logic values are themselves fuzzy sets; some may be compounded fuzzy sets from other atomic ones, by the use of certain operators. As with ordinary crisp sets, different operations can be defined over fuzzy sets. The following presents three basic set operators that are the most commonly used for propagating fuzzy values across logical connectives, where x belongs to a universe X:

- Set complement (logical negation: *not*) $\neg A$

$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

- Set union (logical disjunction: *or*) $A \cup B$

$$\mu_{A \cup B}(x) = max(\mu_A(x), \mu_B(x))$$

- Set intersection (logical conjunction: and) $A \cap B$

$$\mu_{A\cap B}(x) = min(\mu_A(x), \mu_B(x))$$

It should not be difficult to see that these definitions cover the conventional operations on crisp sets as their specific cases. Taking the set intersection as an example, when both sets A and B are crisp, x is a member of $A \cap B$ if and only if (abbreviated to iff hereafter) it belongs to both A and B, and vice versa. This is covered by the definition of fuzzy set intersection because $\mu_{A\cap B}(x) = 1$ (or $min(\mu_A(x), \mu_B(x)) = 1$), iff $\mu_A(x) = 1$ and $\mu_B(x) = 1$. Otherwise, $\mu_{A\cap B}(x) = 0$, given that $\mu_A(x)$ and $\mu_B(x)$ take values only from $\{0, 1\}$ in this case.

An example should help with the understanding of the basic concepts and operators introduced above. Suppose that the universe of discourse, X, is a class of students and that a group, A, of students within this class are said to be *tall* in height. Thus, A is a fuzzy subset of X (but X itself is not fuzzy), since the boundary between *tall* and *not tall* cannot be naturally defined with a fixed real number. Rather, describing this vague concept using a gradual membership function as characterised in figure 6.2 is much more appealing. Similarly, fuzzy term *very tall* can be represented by another fuzzy (sub-)set as also shown in this figure. Given such a definition of the fuzzy set $A = tall$, a proposition like "student x is $tall$" can be denoted by $\mu_A(x)$.

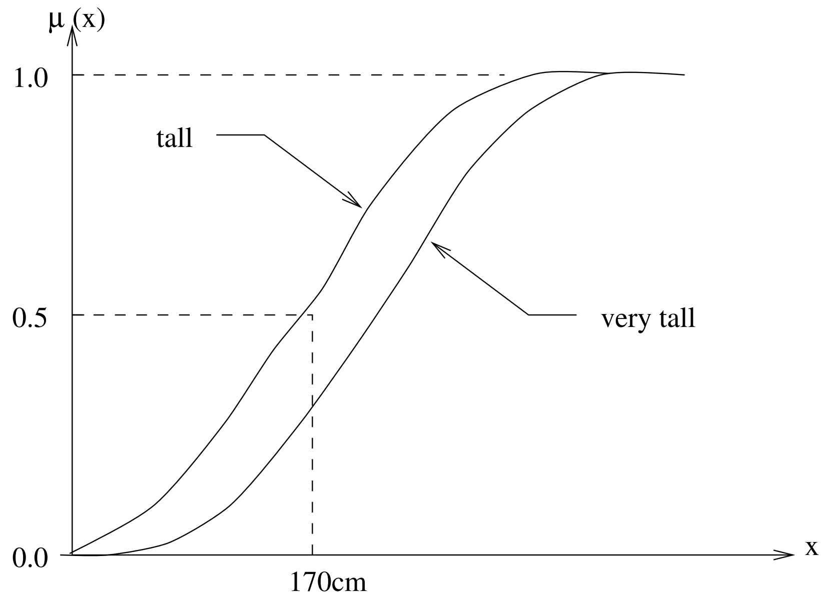


Figure 6.2: Representation of fuzzy sets "tall" and "very tall"

Assume that Andy is a student in the class and that he has a 80% possibility of being considered as a *tall* student. This means that $\mu_A(Andy) = 0.8$. Also, suppose that another fuzzy set B is defined on the same universe X, whose members are said to be *young* in age; and that $\mu_B(Andy) = 0.7$, meaning that Andy is thought to be *young* with a possibility of 70%. From this, using the operations defined above, the following can be derived that is justifiable with respect to common-sense intuitions:

- $\mu_{\neg A}(Andy) = 1 - 0.8 = 0.2$, indicating the possibility of Andy being *not tall*

- $\mu_{A \cup B}(Andy) = max(\mu_A(Andy), \mu_B(Andy)) = max(0.8, 0.7) = 0.8$, indicating the possibility of Andy being *tall or young*
- $\mu_{A \cap B}(Andy) = min(\mu_A(Andy), \mu_B(Andy)) = min(0.8, 0.7) = 0.7$, indicating the possibility of Andy being *both tall and young*

Not only having an intuitive justification, these operations are also computationally very simple. However, care should be taken not to explain the results from the conventional Boolean logic point of view. The laws of excluded-middle and contradiction do not hold in fuzzy logic anymore. To make this point clearer, let us attempt to compare the results of $A \cup \neg A$ and $A \cap \neg A$ obtained by fuzzy logic with those by Boolean probabilistic logic (although such a comparison is itself a common source for debating in the literature). Applying fuzzy logic to the above case of Andy gives that

- $\mu_{A \cup \neg A}(Andy) = max(\mu_A(Andy), \mu_{\neg A}(Andy)) = max(0.8, 0.2) = 0.8$
- $\mu_{A \cap \neg A}(Andy) = min(\mu_A(Andy), \mu_{\neg A}(Andy)) = min(0.8, 0.2) = 0.2$

However, using the theory of probability, different results would be expected such that

- p("Andy is *tall*" *or* "Andy is *not tall*") $= 1$
- p("Andy is *tall*" *and* "Andy is *not tall*") $= 0$

This important difference is caused by the deliberate avoidance of the excluded-middle and contradiction laws in fuzzy logic. This avoidance enables fuzzy logic to represent vague concepts that are difficult to capture otherwise. If the memberships of x belonging to A and $\neg A$ are neither 0 nor 1, then it should not be surprising that x is also of a non-zero membership belonging to A *and* $\neg A$ at the same time. Yet, this claim does not stop the debate about this issue in the literature (see [Sha94, Nov96]).

In addition to the three operators defined above, many conventional mathematical functions can be extended to be applied to fuzzy values. This is possible by the use of so-called *extension principle*, which provides a general extension of classical mathematical concepts to fuzzy environments. For those interested, this principle is stated as follows (though it is beyond the scope of this module): If an n-ary function f maps the Cartesian product $X_1 \times X_2 \times \cdots \times X_n$ onto a universe Y such that $y = f(x_1, x_2, \ldots, x_n)$, and $A_1, A_2, \ldots, A_n$ are n fuzzy sets in $X_1, X_2, .., X_n$, respectively characterised by membership distributions $\mu_{A_i}(X_i), i = 1, 2, \ldots, n$, a fuzzy set on Y can then be induced as given below, where ϕ is the empty set:

$$\mu_B(y) = \begin{cases} max_{\{x_1,\ldots,x_n,y=f(x_1,\ldots,x_n)\}} min(\mu_{A_1}(x_1), \ldots, \mu_{A_n}(x_n)) & \text{if } f^{-1}(Y) \neq \phi, \\ 0 & \text{if } f^{-1}(Y) = \phi \end{cases}$$

A crucial concept in fuzzy set theory is that of *fuzzy relations*, which is a generalisation of the conventional crisp relations. An n-ary fuzzy relation in $X_1 \times X_2 \times \cdots \times X_n$ is, in fact, a fuzzy set on $X_1 \times X_2 \times \cdots \times X_n$. Fuzzy relations can be composed (and this composition is closely related to the extension principle shown above). For instance, if U is a relation from X_1 to X_2 (or, equivalently, a relation in $X_1 \times X_2$), and V is a relation from X_2 to X_3, then the composition of U and V is a fuzzy relation from X_1 to X_3 which is denoted by $U \circ V$ and defined by

$$\mu_{U \circ V}(x_1, x_3) = max_{x_2 \in X_2} min(\mu_U(x_1, x_2), \mu_V(x_2, x_3)), \qquad x_1 \in X_1, x_3 \in X_3$$

A convenient way of representing a binary fuzzy relation is to use a matrix. For example, the following matrix, P, can be used to indicate that a computer-game addict is much more fond of multimedia games than conventional ones, be they workstation or PC-based:

$$P = \begin{array}{c|cc} & Workstation & PC \\ \hline Multimedia & 0.9 & 0.8 \\ Conventional & 0.3 & 0.2 \end{array}$$

Suppose that the addict fancies keyboard-based games more than mouse-based, another relation, Q, may then be used to describe this preference such that

$$Q = \begin{array}{c|cc} & Keyboard & Mouse \\ \hline Workstation & 0.7 & 0.5 \\ PC & 0.8 & 0.3 \end{array}$$

Given these two relations, a composed relation, R, can be obtained as shown below:

$$\begin{aligned} R = P \circ Q &= \begin{pmatrix} 0.9 & 0.8 \\ 0.3 & 0.2 \end{pmatrix} \circ \begin{pmatrix} 0.7 & 0.5 \\ 0.8 & 0.3 \end{pmatrix} \\ &= \begin{pmatrix} max(min(0.9, 0.7), min(0.8, 0.8)) & max(min(0.9, 0.5), min(0.8, 0.3)) \\ max(min(0.3, 0.7), min(0.2, 0.8)) & max(min(0.3, 0.5), min(0.2, 0.3)) \end{pmatrix} \\ &= \begin{array}{c|cc} & Keyboard & Mouse \\ \hline Multimedia & 0.8 & 0.5 \\ Conventional & 0.3 & 0.3 \end{array} \end{aligned}$$

This composition indicates that the addict enjoys multimedia games, especially keyboard-based multimedia games.

6.3 Approximate Reasoning

Fuzzy relations and fuzzy relation composition form the basis for approximate reasoning, sometimes termed fuzzy reasoning. Informally, approximate reasoning means a process by which a possibly inexact conclusion is inferred from a collection of inexact premises. Systems performing such reasoning are built upon a collection of fuzzy production (*if-then*) rules, which provide a formal means of representing domain knowledge acquired from empirical associations or experience. Such systems run upon a given set of facts that allows a (usually partial) instantiation of the premise attributes in some of the rules.

For example, a rule like

if x is A_i and y is B_i then z is C_i

which governs a particular relationship between the premise attributes x and y and the conclusion attribute z, can be translated into a fuzzy relation R_i:

$$\mu_{R_i}(x, y, z) = min(\mu_{A_i}(x), \mu_{B_i}(y), \mu_{C_i}(z))$$

Where A_i, B_i and C_i are fuzzy sets defined on the universes X, Y and Z of the attributes x, y and z, respectively. Provided with this relation, if the premise attributes x and y actually take the fuzzy values A' and B', a new fuzzy value of the conclusion attribute z can then be obtained by applying the *compositional rule of inference*:

$$C' = (A' \times B') \circ R_i$$

Or,

$$\mu_{C'}(z) = max_{x \in X, y \in Y} min(\mu_{A'}(x), \mu_{B'}(y), \mu_{R_i}(x, y, z))$$

Of course, an approximate reasoning system would not normally function using only one specific rule, but a set. Given a set of production rules, two different approaches may be adopted to implement the reasoning in such a system; both rely on the use of the above compositional rule of inference. The first is to apply the compositional rule after an overall relation associated with the entire rule set is found. The second is to utilise the compositional rule locally to each individual production rule first and then to aggregate the results of such applications to form an overall result for the consequent attribute.

Given a set of K *if-then* rules, basic algorithms for these two approaches are summarised below. For simplicity, it is herein assumed that each rule contains a fixed number of N premise attributes $x_1, x_2, \ldots, x_N$ and one conclusion attribute y. Also, the logical connectives are to be applied strictly from left to right.

Method 1: Inference with overall interpreted relation

Step 1: For each rule $k, k \in \{1, 2, \ldots, K\}$, compute its corresponding fuzzy relation.
For instance, given the following two rules ($K = 2, N = 3$):

If x_1 is A_1 and x_2 is not B_1 or x_3 is C_1 then y is D_1

If x_1 is not A_2 or x_2 is B_2 and x_3 is not C_2 then y is not D_2

their associated fuzzy relations are respectively calculated by

$$R_1(x_1, x_2, x_3, y) = min\{max\{min\{\mu_{A_1}(x_1), 1 - \mu_{B_1}(x_2)\}, \mu_{C_1}(x_N)\}, \mu_{D_1}\}$$

$$R_2(x_1, x_2, x_3, y) = min\{min\{max\{1 - \mu_{A_2}(x_1), \mu_{B_2}(x_2)\}, 1 - \mu_{C_2}(x_N)\}, 1 - \mu_{D_2}\}$$

Step 2: Combine individual fuzzy relations, $R_k(x_1, x_2, \ldots, x_N, y)$, $k \in \{1, 2, \ldots, K\}$ to form the overall integrated relation, such that

$$R(x_1, x_2, \ldots, x_N, y) = max_{k \in \{1,2,\ldots,K\}} R_k(x_1, x_2, \ldots, x_N, y)$$

Step 3: Apply the compositional rule of inference to obtain the inferred fuzzy value D of the conclusion attribute, given a fuzzy value for each antecedent attribute (say, A for x_1, B for $x_2, \ldots,$ and C for x_N). That is,

$$\mu_D(y) = max_{x_1, x_2, \ldots, x_N} min\{\mu_A(x_1), \mu_B(x_2), \ldots, \mu_C(x_N), \mu_R(x_1, x_2, \ldots, x_N, y)\}$$

Method 2: Inference via locally interpreted relations

Step 1: This is the same as the first step of method 1.

Step 2: For each rule k, given a fuzzy value for each premise attribute (again, say, A for x_1, B for x_2, ..., and C for x_N), compute the inferred fuzzy value D_k of the conclusion attribute by applying the compositional rule of inference, such that

$$\mu_{D_k}(y) = max_{x_1,x_2,\ldots,x_N} min\{\mu_A(x_1), \mu_B(x_2), \ldots, \mu_C(x_N), \mu_{R_k}(x_1, x_2, \ldots, x_N, y)\}$$

Step 3: Calculate the overall inferred fuzzy value D of the conclusion attribute by aggregating the inferred values derived from individual rules. That is,

$$\mu_D(y) = max_{k,k\in\{1,2,\ldots,K\}} \mu_{D_k}(y)$$

6.4 Fuzzy Control: An Application Case Study

Since the middle of the seventies, fuzzy logic has been applied to many different fields of information processing. Most remarkably, it has been incorporated as a method of representing and handling domain knowledge for intelligent systems control. Indeed, approximate reasoning systems are commercially very successful; many industrial plants and household apparatus are controlled by such systems. The basic ideas of fuzzy logic based control systems, which are often termed fuzzy logic controllers (FLCs), are illustrated below. A detailed overview of fuzzy logic control techniques can be found in [Lee90].

The purpose of a fuzzy logic controller is first to relate the state variables of the physical system under control to the control input variables and then, based on the resulting relationships to compute the values of the control variables using the observations from the system. This is, of course, the same as with any other approaches to systems control. Two important justifications, among others, for the use of fuzzy techniques are: a) it is not unusual that the expressed relationships between the state and control variables are vague and qualitative, and b) the controller of a physical system need not be physical itself but may be purely logical. Within an FLC, domain-dependent heuristics, which are usually linguistically expressed, are typically given in terms of production rules. The system variables are designed to take linguistic values which are represented as fuzzy subsets of the non-fuzzy universes (from which the underlying physical variables take their real values).

Clearly, within a fuzzy logic based control mechanism, both the state variables and the control variables take fuzzy sets as their values. However, in a real application, observations and control actions are all represented in real numbers. Therefore, to use an FLC, the observations obtained from the physical system must be mapped onto fuzzy set representation and the outputs of the FLC must be mapped back to real-valued control signals. Hence, two supplementary processes are necessary to enable the use of an FLC, which are respectively called the fuzzification and the defuzzification. Fuzzification is, however, largely a task of domain-specific data interpretation and, thus, is left out from the present discussion.

There exist a number of methods for defuzzification [Lee90, Run97], though there is no clear winner. Two widely used ones are briefly summarised below.

- The *centre of gravity* method finds the geometrical centre $\hat{y}$ in the inferred fuzzy value D of the conclusion attribute y as the defuzzified value, such that

$$\hat{y} = \frac{\sum y\mu_D(y)}{\sum \mu_D(y)}$$

- The *mean of maxima* method finds the value whose membership degree is the largest in D as the defuzzified value; if there are more than one value which has the maximum degree, then the average of them is taken as the defuzzified value.

The present FLC example considers the task of controlling the heat-pressure loop of a steam engine. Suppose that this system is described by two state variables pe and cpe, and a control variable hc:

- pe – the pressure error between the actual and desired values
- cpe – the change in pressure error between the present pe and the last (corresponding to the last sampling instant)
- hc – the heat input change

These variables are quantised into a number of points within the ranges of their magnitude. The linguistic values of each variable are then (often subjectively) defined using several fuzzy numbers. For instance, the variable pe may be quantised into 11 points, corresponding to its universe of discourse, say, $[-5, 5]$, ranging from maximum negative error through zero error to maximum positive error, such that

$$\begin{array}{c|ccccccccccc|}
 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 \\
L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0.8 & 1 \\
M & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.6 & 1 & 0.6 & 0.1 \\
S & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.8 & 0.1 & 0 & 0 \\
O & 0 & 0 & 0 & 0 & 0.1 & 1 & 0.1 & 0 & 0 & 0 & 0 \\
-S & 0 & 0 & 0.1 & 0.8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-M & 0.1 & 0.6 & 1 & 0.6 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-L & 1 & 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Where L stands for the linguistic value $positive\ large$, $-M$ for $negative\ medium$, and O for $zero$, etc. Each linguistic value is in this case a discrete fuzzy number, e.g.

$$L = \{(-5,0), (-4,0), \ldots, (2,0), (3,0.2), (4,0.8), (5,1)\}$$

For presentational simplicity, this is usually represented by

$$\mu_L(pe) = \frac{0.2}{3} + \frac{0.8}{4} + \frac{1}{5}$$

From these primitive fuzzy sets, values such as "*not L or M*" can be assigned to the variables using the relevant logic connectives defined earlier. Further, much more sophisticated values can be compounded using so-called linguistic hedges (which may be thought of as a kind of quantifiers) [Zad 6], though this latter issue is beyond the scope of this introductory discussion.

Control rules are assumed to be expressed in the fuzzy if-then form; a collection of such rules for the present example are given as follows:

If $pe = M$ or L then if $cpe = -S$ then $hc = -M$
If $pe = S$ then if $cpe = -S$ or O then $hc = -M$
If $pe = O$ then if $cpe = M$ or L then $hc = -M$
If $pe = O$ then if $cpe = -M$ or $-L$ then $hc = M$
If $pe = O$ then if $cpe = O$ then $hc = O$

If $pe = O$ then if $cpe = -M$ or $-L$ then $hc = -M$
If $pe = O$ then if $cpe = M$ or L then $hc = M$
If $pe = -S$ then if $cpe = S$ then $hc = M$
If $pe = -M$ or $-L$ then if $cpe = -S$ then $hc = M$

Each of these rules can be interpreted as a fuzzy relation, between the state variables pe, cpe and the control input hc. Then, either of the two inference methods introduced before may be used to derive a desired control signal when given actual values for the two state variables.

To illustrate the basic idea let us simplify the problem, assuming that the purpose of the fuzzy logic controller is to maintain a preset pressure in the engine, which it tries to achieve by controlling the heat input. Suppose that inference method 1 and the mean of maxima defuzzification are adopted. Also, suppose that both variables take their values as defined above and that there are only 2 rules available:

R_1: If $pe = -S$ then $hc = M$
R_2: If $pe = M$ or L then $hc = -M$

Following step 1 of the inference method, consider the representation of rule R_1 first. Since

$$\mu_{-S}(pe) = \frac{0.1}{-3} + \frac{0.8}{-2} + \frac{1.0}{-1}$$

(that is, $-3, -2, -1$ belonging to $-S$ with a membership of $0.1, 0.8, 1.0$ respectively, whilst the other elements of the domain do not belong to $-S$), and

$$\mu_M(hc) = \frac{0.1}{1} + \frac{0.6}{2} + \frac{1.0}{3} + \frac{0.6}{4} + \frac{0.1}{5}$$

so the interpreted fuzzy relation for R_1 is

$$\mu_{R_1}(pe, hc) = min(\mu_{-S}(pe), \mu_M(hc)) =$$

$$\begin{array}{c|ccccccccccc}
 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
-5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\
-2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.6 & 0.8 & 0.6 & 0.1 \\
-1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 \\
0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{array}$$

For the slightly more complicated rule R_2, given its joint condition being

$$\begin{aligned}
\mu_{M \vee L}(pe) &= max(\mu_M(pe), \mu_L(pe)) \\
&= max((\frac{0.1}{1} + \frac{0.6}{2} + \frac{1.0}{3} + \frac{0.6}{4} + \frac{0.1}{5}), (\frac{0.2}{3} + \frac{0.8}{4} + \frac{1.0}{5})) \\
&= \frac{0.1}{1} + \frac{0.6}{2} + \frac{1.0}{3} + \frac{0.8}{4} + \frac{1.0}{5}
\end{aligned}$$

and its conclusion being

$$\mu_{-M}(hc) = \frac{0.1}{-5} + \frac{0.6}{-4} + \frac{1.0}{-3} + \frac{0.6}{-2} + \frac{0.1}{-1}$$

the corresponding fuzzy relation is

$$\mu_{R_2}(pe, hc) = min(\mu_{M \vee L}(pe), \mu_{-M}(hc)) =$$

$$\begin{array}{c|ccccccccccc} & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline -5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2 & 0.1 & 0.6 & 0.6 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4 & 0.1 & 0.6 & 0.8 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}$$

From step 2 of inference method 1, the overall integrated relation R is obtained by joining these two algorithms as follows:

$$\mu_R(pe, hc) = \mu_{R_1 \vee R_2}(pe, hc) = max(\mu_{R_1}(pe, hc), \mu_{R_2}(pe, hc)) =$$

$$\begin{array}{c|ccccccccccc} & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline -5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ -2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.6 & 0.8 & 0.6 & 0.1 \\ -1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 \\ 0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2 & 0.1 & 0.6 & 0.6 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4 & 0.1 & 0.6 & 0.8 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5 & 0.1 & 0.6 & 1.0 & 0.6 & 0.1 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}$$

This gives the final result of fuzzy interpretation for the given set of control rules. It is ready for use to perform approximate reasoning, in order to generate necessary control signals when certain observations are obtained. For instance, suppose that a newly obtained fuzzy value of the state variable pe is

$$\mu_{New}(pe) = \frac{0.3}{-3} + \frac{0.8}{-2} + \frac{0.9}{-1}$$

Applying the compositional rule of inference (step 3 of inference method 1), the following new value of the control variable hc results:

$$\begin{aligned} \mu_{New}(hc) &= max(min(\mu_{New}(pe), \mu_R(pe, hc))) \\ &= \frac{0.1}{1} + \frac{0.6}{2} + \frac{0.9}{3} + \frac{0.6}{4} + \frac{0.1}{5} \end{aligned}$$

where, for instance, the membership value of the element 2 is calculated as follows:

$$max(min(0,0), min(0,0), min(0.3,0.1), min(0.8,0.6), min(0.9,0.6),$$

$$min(0,0), min(0,0), min(0,0), min(0,0), min(0,0), min(0,0))$$
$$= max(0,0,0.1,0.6,0.6,0,0,0,0,0,0) = 0.6$$

Note that the newly obtained value of pe is similar to, but different from, its value used in the condition of rule R_1, and that the new value of hc, resulting from applying the control mechanism (or firing the rules), is correspondingly similar to its counterpart in the conclusion of R_1. This property of fuzzy inference is important to producing robust control actions, a factor making FLCs popular.

As the mean of maxima is adopted to produce defuzzified control signals and there is a single maximum membership value (0.9), the underlying element 3 is therefore returned as the defuzzified real value. Note that, in general, a defuzzified value need not be an integer but can be any real number within the underlying real value range of the variable, even though its original fuzzy value may be based on a range of integers. Due to its simplicity, the present example does not show such a case, however.

6.5 Discussions

Before closing this chapter, it is worth emphasising that the concept of fuzzy sets is not merely a disguised form of subjective probability. The theory of fuzzy sets and the theory of probability are two approaches to attaining realistic solutions to problems in reasoning with uncertainty. In essence, fuzzy set theory is aimed at dealing with sources of uncertainty or imprecision that are inherently vague and non-statistical in nature. For example, the proposition "X is a *large number*," in which *large number* is a label of a fuzzy subset of non-negative integers, defines a *possibility distribution* rather than the probability distribution of X. This implies that, if the degree to which an integer I fits the subjective perception of *large number* is $\mu(I)$, then the possibility that X may take I as its value is numerically equal to $\mu(I)$. Consequently, such a proposition conveys no information about the probability distribution of the values of X, as argued by Zadeh [Zad86].

A simple example to show the difference between the two theories is throwing a dice: probability distribution indicates the random number achieved after each throw, whilst fuzzy membership functions describe whether the achieved number is small, medium or large. Thus, probability theory does not always provide an adequate tool for the analysis of problems in which the available information is ambiguous, imprecise or even unreliable. An interesting combination of fuzziness and probability would be the chance for the number achieved to be small. Attempts to combine probability theory and fuzzy logic exist, aiming at providing powerful inference mechanisms for problems involving such ill-defined random variables [PG98]. Details of this are beyond the scope of these introductory notes of course.

Chapter 7

Representing Ignorance: Dempster-Shafer Theory of Evidence

7.1 Introduction

Many approaches to uncertainty representation and reasoning (e.g. certainty-factor based [BS84] and Bayesian techniques) deal with uncertainty measures attached to individual hypotheses. However, for certain applications, estimating the uncertainty measures of every individual hypothesis can be very difficult. It may only be possible to obtain a broader notion of uncertainty measure associated with sets of hypotheses instead. Dempster-Shafer Theory of Evidence (DST) provides a method for performing inference with uncertain knowledge in such circumstances [Wal96].

DST can be thought of as a mathematical theory of belief. This theory was originally designated by Dempster, in an attempt to represent uncertainty by a range of probabilities instead of a single probabilistic value [Dem67]. DST was formally described in its present form in the book entitled "A Mathematical Theory of Evidence" [Sha76] by Shafer, who refined and extended Dempster's original work. A brief account of DST is given next and an example of this theory in use is provided in section 7.3, where an important conceptual difference between DST and the conventional probability theory is also addressed. Finally, some discussions about DST are presented in section 7.4.

7.2 Basics of the Theory

Suppose that an exhaustive set of mutually exclusive hypotheses is denoted by

$$H = \{h_1, h_2, ..., h_N\}$$

For simplicity, this set of hypotheses is assumed to be finite, though DST can be similarly applied to infinite domains. Given the hypothesis set H being mutually exclusive and exhaustive, the probability of the correct hypothesis existing within this set must be 1; whilst the probability of the correct hypothesis lying within the empty set $\phi = \{\}$ is 0. For any subset of H, say, a (sub-)set H_i of hypotheses $\{h_{i1}, h_{i2}, ..., h_{iM}\}$ ($h_{ij} \in H, j = 1, 2, ..., M, M \leq N$), a value between 0 and 1 can be assigned to express the belief that the correct hypothesis may exist in it. An estimate of such a degree of belief, for a subset to contain the correct hypothesis, is denoted by $m(H_i)$, with $\sum_i m(H_i) = 1$ and $m(\phi) = 0$.

The measure of belief defined above is the belief in evidence assigned to a single (sub-)set of hypotheses and is called the *local belief* of that set. However, knowledge of the subsets of a hypothesis set may imply additional information about that set. To reflect the difference between a local belief and a measure of belief that applies to information contained within a hypothesis set and all its subsets, DST defines a further belief measure, termed the *total belief*, of a given hypothesis set. More formally, given a hypothesis (sub-)set S and all its subsets, the total belief of S is defined by

$$Bel(S) = \sum_{T \in P(S)} m(T)$$

where $P(S)$ denotes the power set of S. The power set of a given set consists of all possible subsets of that set. For example, the power set of $S = \{a, b, c\}$ is

$$P(S) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{c, a\}, \{a, b, c\}\}$$

This total belief is regarded as the minimum belief of S based on given evidence. Another type of numerical measure is introduced in DST to reflect the plausible maximum belief that may be assigned to a hypothesis set. This numerical estimate of belief is termed the *plausibility*, of the hypothesis set, and is defined as the degree to which given evidence fails to refute that set of hypotheses:

$$Pls(S) = 1 - Bel(\bar{S}) = 1 - \sum_{T \in P(\bar{S})} m(T)$$

where $\bar{S} = H - S$ is the competing set of S, denoting the union of those subsets of H that have no intersection with S.

The degree of belief assigned to a subset of hypotheses may change in response to the acquisition of further evidence. In particular, as a piece of new evidence is gathered, the local belief in the hypothesis subset Z, of an exhaustive hypothesis set H, is revised by the use of the *rule of combination*:

$$m(Z) = (\sum_{X \cap Y = Z} m(X) * m(Y))/(1 - \sum_{X \cap Y = \phi} m(X) * m(Y))$$

In this rule, $X, Y \in P(H)$, with $m(X)$ representing a local belief due to the existing (previously combined) evidence and $m(Y)$ representing that exclusively due to the new evidence, and the denominator is used for normalisation purposes (see later).

Note that once the local beliefs get revised, the total beliefs and plausibilities can easily be updated accordingly.

7.3 An Example

To illustrate how these definitions can be utilised in practice to perform reasoning with uncertain knowledge, consider the following example. Imagine that only computers made by four giant computer manufacturers are possibly deemed to be popular in the market. These are SUN and HP workstations, and Compaq (CP) and Packard Bell (PB) PCs. There exists considerable debate regarding which computers are the most popular. Some views or evidence only relates to types of computer (i.e., workstations or PCs), rather than to a particular brand. Suppose that it is now required to estimate the strength of belief in various top-popularity hypotheses.

To be concise, a set of popularity hypotheses is hereafter referred to the set of computer brands or types concerned unless otherwise stated. For instance, the hypothesis set of PCs being the most popular is simply written to be the set of PCs, i.e. $\{CP, PB\}$. The entire set of mutually exclusive hypotheses is denoted by $H = \{SUN, HP, CP, PB\}$.

To start with, the measure of local belief in the entire hypothesis set H, i.e. $m(H)$, is set to 1, assuming that the only information initially available is that the correct hypothesis lies within this exhaustive set. Under this initial condition, where no evidence about other hypothesis subsets is provided, the measures of local belief in any other elements of the power set of H are set to 0. That is,

$$m(S) = 0, S \in P(H), S \neq H$$

Suppose that the first piece of information obtained suggests that the score of PCs being the most popular is 0.4 (on a scale of 0 to 1, with 0 being the score for absolutely not the most popular and 1 for the definitely most popular). This weight of evidence is assigned to the set of PCs as its local belief and the remaining belief is assigned over the whole set H. Thus,

$$m(\{CP, PB\}) = 0.4$$

$$m(H) = 0.6$$

It is important to note that this assignment of (local) beliefs indicates a major difference between DST and the conventional probability theory. Using DST, the rest of the belief apart from that already assigned to the set of PCs is left with the entire hypothesis set. It stands for the amount of belief uncommitted to any specific hypothesis subset other than the entire set itself. This means that neither belief nor disbelief in PCs being the most popular is to a degree of 0.6, which is the degree of *nonbelief* in the evidence. However, from the available information, probability theory would assume that

$$p(\{CP, PB\}) = 0.4$$

$$p(H - \{CP, PB\}) = p(\{SUN, HP\}) = 1 - p(\{CP, PB\}) = 0.6$$

That is to say, in probability theory, given the belief in PCs being the most popular to be 0.4, the disbelief in the set of PCs must be 0.6 and this degree of disbelief must be the same as the belief in the set of those computer brands other then PCs, i.e. in the set of workstations. This difference between the two approaches highlights the most significant property of DST in that committing some belief to a hypothesis set (or even to a single hypothesis if the hypothesis set contains only one hypothesis) does not require committing the rest of one's belief to the negation of that hypothesis set (or to the negation of that single hypothesis).

Return to the example. Given the local beliefs, the total belief in the set of PCs is computed as follows:

$$Bel(\{CP, PB\}) = m(\{CP, PB\}) + m(\{CP\}) + m(\{PB\}) = 0.4 + 0 + 0 = 0.4$$

The plausibility of the set of hypotheses indicating PCs to be the most popular is 1 minus the present total belief of workstations being regarded the most popular:

$$Pls(H - \{SUN, HP\}) = Pls(\{CP, PB\}) = 1 - Bel(\{SUN, HP\}) = 1$$

Suppose that the second piece of evidence is now provided, indicating that the score of the set of workstations plus Packard Bell PCs being the most popular is 0.8. In DST, this new piece of information is represented by

$$m(\{SUN, HP, PB\}) = 0.8$$

$$m(H) = 0.2$$

Clearly, presented with this new evidence, the belief in the hypothesis set of PCs being the most popular should be revised accordingly. How? DST works this out by first finding the local beliefs of all the intersections between the two batches of evidence. This is indicated in table 7.1, where entries show the results of a) multiplying the local belief of each hypothesis set, from the first batch of evidence on the left hand side, with that from the second batch on the top, and b) assigning each resultant product to the intersection of the two sets whose local beliefs led to that product. For example, the belief that $m(\{PB\}) = 0.32$ is obtained by multiplying $m(\{SUN, HP, PB\}) = 0.8$ with $m(\{CP, PB\}) = 0.4$, and by intersecting $\{SUN, HP, PB\}$ with $\{CP, PB\}$. Then, from these newly calculated local beliefs, DST updates the total belief and plausibility of the hypothesis set of PCs as follows:

$$Bel(\{CP, PB\}) = m(\{CP, PB\}) + m(\{CP\}) + m(\{PB\}) = 0.08 + 0 + 0.32 = 0.4$$

$$Pls(\{CP, PB\}) = 1 - Bel(\{SUN, HP\}) = 1 - (0 + 0 + 0) = 1$$

From this, it can be concluded that, although the second piece of evidence does not alter the total belief in PCs being the most popular, it has shifted much of this overall belief to the popularity of Parkard Bell PCs (given that $m(\{PB\})$ has increased from 0 to 0.32 while $m(\{CP, PB\})$ has decreased from 0.4 to 0.08).

Combination	$m(\{SUN, HP, PB\}) = 0.8$	$m(H) = 0.2$
$m(\{CP, PB\}) = 0.4$	$m(\{PB\}) = 0.32$	$m(\{CP, PB\}) = 0.08$
$m(H) = 0.6$	$m(\{SUN, HP, PB\}) = 0.48$	$m(H) = 0.12$

Table 7.1: Combination of local beliefs given the second piece of evidence

The above process of calculating local beliefs is a simplification of what would actually be performed, with respect to the rule of combination. This simplification is permitted at this stage, due to the fact that a) no intersections between those two lots of evidence would result in the same hypothesis set and hence, the numerator in the formula for combining local beliefs involves only one intersection and multiplication, and b) for any $S_1, S_2 \in P(H)$, $m(S_1) * m(S_2) = 0$ if $S_1 \cap S_2 = \phi$, hence the denominator in the formula happens to be 1 and no normalisation is required.

Now, suppose that a further piece of evidence is available such that the score of SUN workstations being the most popular is 0.3. Again, this score can be seen as the local belief in the set of $\{SUN\}$ and this measure is combined with the existing local beliefs. Then, the result can be used to revise the total belief and plausibility of the set of hypotheses indicating that PCs are the most popular. However, when updating the local beliefs, the normalising denominator appearing in the rule of combination is no longer to be 1 this time since, for instance, $m(\{PB\}) = 0.32, m(\{SUN\}) = 0.3$, whilst $\{PB\} \cap \{SUN\} = \phi$. Also, the intersections between this batch of evidence and the existing one may result in more than one set of hypotheses to be the same, e.g.

$$\{SUN\} \cap \{SUN, HP, PB\} = \{SUN\}$$

$$\{SUN\} \cap H = \{SUN\}$$

Table 7.2 shows the results of evidence combination. To give a flavour of how the entries in this table are obtained and, therefrom, how the local beliefs get combined, some computational details are given below.

Take the entries for $\{SUN\}$ as an example. Given that both $\{SUN\} \cap \{SUN, HP, PB\}$ and $\{SUN\} \cap H$ lead to $\{SUN\}$, according to the rule of combination, the following needs to be calculated:

$$m(\{SUN\}) * m(\{SUN, HP, PB\}) = 0.3 * 0.48 = 0.144$$

$$m(\{SUN\}) * m(H) = 0.3 * 0.12 = 0.036$$

From this, the following *partial* local beliefs (of $\{SUN\}$ being the most popular, as listed in the table) can be obtained:

$$m_a(\{SUN\})/D = 0.144/0.88 = 0.164$$

$$m_b(\{SUN\})/D = 0.036/0.88 = 0.041$$

where the normalising denominator is computed such that

$$\begin{aligned} D &= 1 - \sum_{X \cap Y = \phi} m(X) * m(Y) \\ &= 1 - [m(\{PB\}) * m(\{SUN\}) + m(\{CP, PB\}) * m(\{SUN\})] \\ &= 1 - (0.32 * 0.3 + 0.08 * 0.3) = 0.88 \end{aligned}$$

Thus, with respect to the rule of combination, the updated local belief in the set of SUN workstations is

$$m(\{SUN\}) = 0.164 + 0.041 = 0.205$$

Combination	$m(\{SUN\}) = 0.3$	$m(H) = 0.7$
$m(\{PB\}) = 0.32$	$m_u(\phi) = 0$	$m(\{PB\}) = 0.254$
$m(\{CP, PB\}) = 0.08$	$m_v(\phi) = 0$	$m(\{CP, PB\}) = 0.064$
$m(\{SUN, HP, PB\}) = 0.48$	$m_a(\{SUN\}) = 0.164$	$m(\{SUN, HP, PB\}) = 0.382$
$m(H) = 0.12$	$m_b(\{SUN\}) = 0.041$	$m(H) = 0.095$

Table 7.2: Combination of local beliefs given the third piece of evidence

The newly obtained local beliefs are, once again, used to update the total belief and plausibility of the popularity hypothesis on PCs as follows:

$$\begin{aligned} Bel(\{CP, PB\}) &= m(\{CP, PB\}) + m(\{CP\}) + m(\{PB\}) \\ &= 0.064 + 0 + 0.254 = 0.318 \end{aligned}$$

$$\begin{aligned} Pls(\{CP, PB\}) &= 1 - Bel(\{SUN, HP\}) \\ &= 1 - (m(\{SUN, HP\}) + m(\{SUN\}) + m(\{HP\})) \\ &= 1 - (0 + 0.205 + 0) = 0.795 \end{aligned}$$

It can be seen that the most recent evidence has reduced the total belief in the set of hypotheses indicating that PCs are the most popular computers in the market, although the reduction from 0.4 to 0.318 is not very significant. This piece of new information has also reduced the difference between the measure of the total belief and that of the plausibility, from 0.6 to 0.477. The gap between the minimum and maximum beliefs in the hypothesis set is therefore narrowed, implying a better allocation of the "correct" belief.

7.4 Discussions

As described above, the Dempster-Shafer Theory has a good theoretical foundation. It has been shown that the certainty-factor based approach to uncertainty handling, which was originally developed upon an *ad hoc* basis and used in many rule-based reasoning systems, can be formalised and seen as a special case of this theory [GS85]. However, DST is computationally much more expensive than certainty factor techniques. Also, for any practical application, the definition of the entire mutually exclusive and collectively exhaustive hypothesis set must be carefully tailored to the domain, so that every new piece of evidence can be easily translated into a local belief in a subset of hypotheses. In addition, the original DST did not provide a mechanism for belief propagation through logical connectives, though this was addressed in a later proposal (see [GLF81]).

Chapter 8

Constraint Satisfaction

8.1 Introduction

Many AI applications involve knowledge that can be expressed as relations on facts, events and/or some other sorts of terms. For example, mass and energy balance relations often appear in physical reasoning systems; length and angle relations often appear in spatial reasoning systems; and instance and duration relations often appear in temporal reasoning systems. These relations, which must be obeyed by variables in a given problem description, are generally referred to as constraints. They are a natural means of knowledge representation. A constraint often takes the form of an equation or inequality, but in the most abstract sense is simply a logical relation amongst several variables expressing a set of admissible value combinations.

The process of identifying a solution to a problem which satisfies all specified constraints is termed constraint satisfaction. Programs developed to maintain constraints and to perform the solution process are called constraint satisfaction systems.

Formally speaking, a classical constraint satisfaction problem (CSP) involves a set of n variables, $X = \{x_1, ..., x_n\}$, and a set of constraints, C_{set}, over these variables. Each x_i has an associated domain, D_i, describing its potential values. A variable *assignment* is an assignment to a variable, x_i, of one of the values from its associated domain, D_i. Table 8.1 lists the four variables in an example problem on lexicographic ordering and their associated domains.

Variable	Domain
x_1	$\{P, K, B\}$
x_2	$\{U, L, I\}$
x_3	$\{V, U, N\}$
x_4	$\{K, J, E, B\}$

Table 8.1: Variables and associated domains

A classical constraint $C(x_i, ..., x_j) \in C_{set}$ specifies a subset of the Cartesian product $D_i \times ... \times D_j$, indicating variable assignments that are compatible with each other. Table 8.2 shows the constraints that exist in the example problem. They specify a lexicographic ordering such that, for instance, $x_1 < x_2$ requires that the assignment to x_1 is strictly lexicographically less than that to x_2. A *solution* to a CSP is a complete value assignment to the problem variables satisfying all constraints simultaneously. A CSP may contain several solutions, and the task for a constraint satisfaction system is to find one, more than one or all of these.

Constraint	Definition
$C(x_1, x_2)$	$x_1 < x_2$
$C(x_1, x_3)$	$x_1 < x_3$
$C(x_1, x_4)$	$x_1 < x_4$
$C(x_2, x_3)$	$x_2 < x_3$
$C(x_2, x_4)$	$x_2 < x_4$
$C(x_3, x_4)$	$x_3 > x_4$

Table 8.2: Constraints for the example problem of table 8.1

The question is how to determine whether a solution exists and, if so, how to find one or more solutions? The next section provides a brief account for answering this question.

8.2 Basic Solution Methods

A simple (yet computationally very expensive) approach to solving a CSP is known as *generate-and-test*. This approach assumes finite discrete domains and so the solution space is finite. Each possible combination of variable assignments is generated and subsequently tested to see whether any of the specified constraints are violated. A little thought should reveal that generate-and-test is sound and complete, but can be very slow in finding a solution. In the worst case, the number of combinations to be generated and tested is the size of the Cartesian product of all variable domains. This brute force attempt will, therefore, be very time-consuming on problems of any reasonable size.

The search for a solution may be viewed as tree traversal. Figure 8.1 shows the search tree produced by generate-and-test in finding the solution $\{x_1 = B, x_2 = I, x_3 = V, x_4 = K\}$ to the example problem above. Nodes are annotated with the domain values that were assigned to the current variable at this point in the search. The search path is indicated by the dashed, directed line. For clarity, an unsuccessful assignment to x_4 of each of the elements in D_4 has been replaced by a triangle. Each branch represents a partial assignment. Since this algorithm generates complete assignments before any constraint checks are made, each branch represents a complete assignment in this case. The ith level of the tree represents choices made for the ith variable, with level 0 (the root) denoting the empty set of assignments.

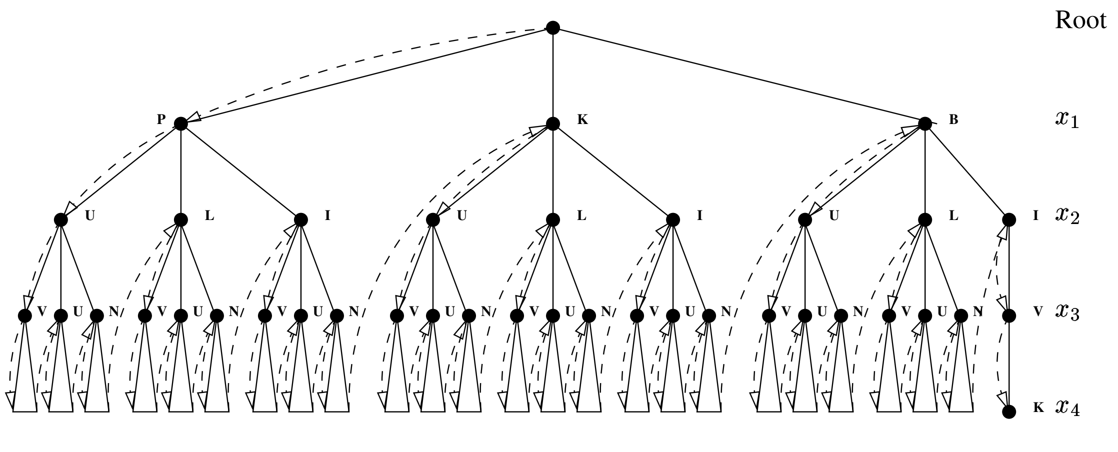


Figure 8.1: Generate-and-test as a process of tree search

An alternative approach for solving CSPs with finite domains is *backtracking* search. This approach considers the variables in some order. Starting with the first, it assigns a possible value to each successive variable in turn so long as the assigned values are consistent with those assigned in the past. During this sequential value assignment process (which is sometimes called the instantiating process), if a variable is encountered such that none of its domain values is consistent with previous (partial) assignments, backtracking takes place. The value assigned to the last variable is replaced with the next unassigned value remaining in its domain (if any). The search continues in the same way until a solution is found or until it may be concluded that no solution of the problem exists.

More formally, suppose that the following variable numbering scheme is adopted. At a level, i, of the instantiation order (in which value assignments are being made), past variables have indices less than i and have been assigned values, and future variables have indices greater than i and do not yet have value assignments. The backtracking algorithm makes assignments sequentially such that x_i is instantiated to extend a partially complete assignment involving past variables, i.e. $x_1, ..., x_{i-1}$. The constraints dictate the subset of D_i that can be assigned to x_i given previous variable assignments. If there are no possibilities for x_i, then it backtracks, making a new choice for x_{i-1}. If there are no new choices for x_{i-1}, the algorithm backtracks until a variable is found with an untested domain element or it fails.

Figure 8.2 shows the search tree generated by backtracking in producing the same solution to the example problem as that found by generate and test. Black nodes represent a successful assignment at this search point, and white nodes represent a failed assignment. The efficiency gained from performing backtracking is obvious. Once any partial assignment of variables' values fails to satisfy a constraint it cannot be part of any total consistent assignment or of a solution sought. Thus, a potentially very large subspace of the product space of the currently unassigned variable domains is eliminated by a single violation.

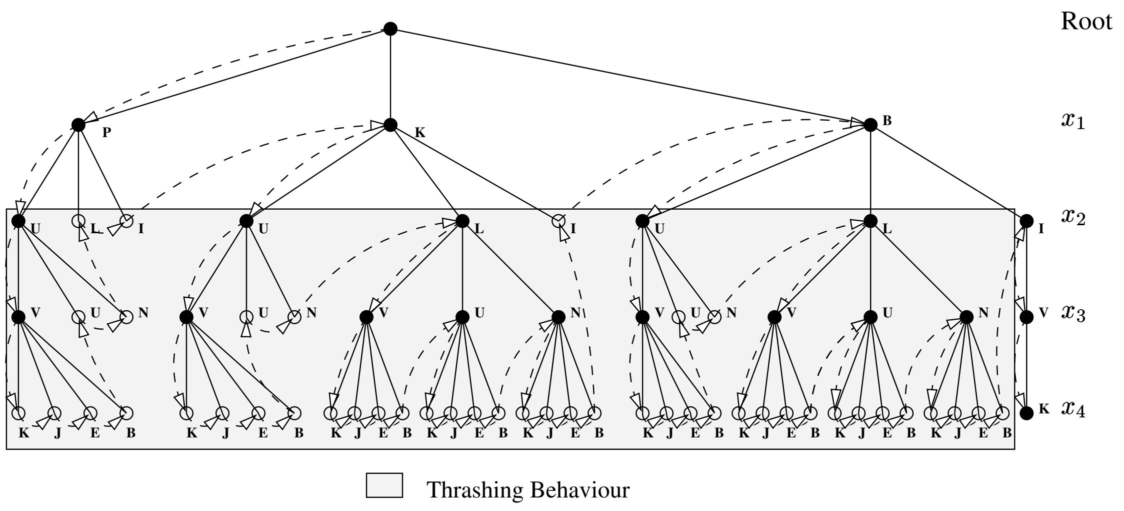


Figure 8.2: Improvement enabled by backtracking

The worst case complexity of backtracking is exponential in the number of variables, however. A major factor in this behaviour is that it suffers from *thrashing*: repeated failure of the search process due to the same reason. Thrashing is present in backtracking's attempt to solve the example problem. It is caused here because of $C(x_2, x_4)$: when $x_2 = U$, or $x_2 = L$ there is no element of D_4 that can be assigned to x_4 such that the constraint will be satisfied. This is not discovered by backtracking until all possible assignment combinations to x_3 and x_4 have been tried. Several improvements to the backtracking algorithm have been proposed to alleviate thrashing. These are beyond the scope of this module but an overview of them can be found in [MS99].

8.3 Constraint Propagation

Instead of making improvements directly on tree search, modifications can be made to reduce the size of the search space that the backtracking algorithm has to handle in the first place. The idea is to pre-process a given CSP to create a simpler, equivalent (i.e. with the same solution set) problem. Most of these modifications utilise a common core method, referred to as *constraint propagation*.

In constraint propagation, information deduced from a local group of constraints is recorded as a change in the given CSP. Such changes are used to make further deductions and therefore further changes. This causes the effects of originally locally constrained information to spread gradually throughout the entire CSP.

An efficient implementation of constraint propagation is the Waltz algorithm, originally developed for line labelling in early computer vision programs [Wal75]. This algorithm works by applying the following *refinement* on each constraint and each variable in the problem iteratively until no further changes are produced:

$$Refine(C, v_j) = \{d_j \in D_j | \exists(d_i \in D_i, i = 1, ..., k, i \neq j), C(d_1, ..., d_j, ..., d_k)\}$$

where C is a constraint on variables $v_i, i = 1, ..., k$ that are defined on domains D_i. $Refine(C, v_j)$ represents the set of values for v_j that is consistent with constraint C and the domains $D_i, i \neq j$. In other words, a value d_j is in $Refine(C, v_j)$ if it is in D_j and it is part of some k-tuple assignment d_1, ..., d_k which satisfy C with d_i belonging to D_i. Clearly, $Refine(C, v_j) \subseteq D_j$.

Given this notion of refinement, the Waltz algorithm can be presented as the combination of the two procedures shown in figure 8.3 (modified from [Dav87]). Note that in this algorithm, no specification is given as to where a constraint should be placed in the queue when it is added. The user of the algorithm has the freedom of deciding where it should go.

Waltz algorithm reflects a number of important properties that are shared by many follow-up constraint propagation techniques, including:

- Simplicity of the control strategy involved, making it easy to code, to understand and to modify.
- Gradualness of performance degradation under time limitations, offering partial results if the execution is interrupted before completion.
- Locality of enforcing constraints, performing deduction normally within a small portion of the CSP.
- Incrementability in describing a CSP, allowing new constraints to be added during the execution.

To illustrates the Waltz algorithm at work, two examples are provided here.

Example One

The first CSP consists of two constraints imposed over three variables a, b, c:

$$\begin{array}{ll} C_1: & b = c - a \\ C_2: & b \geq a \end{array}$$

```
**Procedure** $REVISE(C(v_1, ..., v_k))$

    **begin** $CHANGED \leftarrow \phi$
        **for** each variable $v_i$ **do**
            **begin** $D \leftarrow REFINE(C, v_i)$
                **if** $D = \phi$ **then halt**
                **else if** $D \neq D_i$ **then**
                    **begin** $D_i \leftarrow D$
                        add $v_i$ to $CHANGED$
                    **end**
            **end**
        **return** $CHANGED$
    **end**
```

```
**Procedure** WALTZ

    **begin** $Q \leftarrow$ a queue of all given constraints
        **while** $Q \neq \phi$ **do**
            **begin** remove constraint $C$ from $Q$
                $CHANGED \leftarrow REVISE(C)$
                **for** each $v_i$ in $CHANGED$ **do**
                  **for** each $C' \neq C$ which involves $v_i$ **do**
                      **if** $C' \notin Q$ **then** add $C'$ to $Q$
            **end**
    **end**
```

Figure 8.3: Waltz algorithm

These variables are defined on the following integer domains:

$$\begin{array}{lcl} D_a & = & \{4,5,6,7,8,9\} \\ D_b & = & \{2,3,4,5,6,7,8,9,10,11\} \\ D_c & = & \{3,4,5,6,7,8\} \end{array}$$

Suppose that the algorithm starts with the given constraint queue: (C_1, C_2), and that the refinement is carried out over variables (related by a constraint) from left to right. Look at, say, the first constraint $C_1 : b = c - a$ first. In this constraint, the first variable whose domain is to be refined is b. Given that $c \in \{3, 4, ..., 8\}$ and $a \in \{4, 5, ..., 9\}$, any value in the set $\{-6, -5, ..., 4\}$ might be a valid assignment to b with respect to C_1, since such an assignment would be consistent with the constraint. However, it is already known that $b \in \{2, 3, ..., 11\}$. Hence, $Refine(C_1, b) = \{2, 3, 4\}$. This means that the domain of b is reset to $\{2, 3, 4\}$. Due to this change, b becomes an element of the temporary collection of those variables involved in the current constraint whose domain has been revised. Similarly, $Refine(C_1, c) = \{6, 7, 8\}$ and $Refine(C_1, a) = \{4, 5, 6\}$. Therefore, the temporary collection of the revised variables contains all the three variables $\{a, b, c\}$.

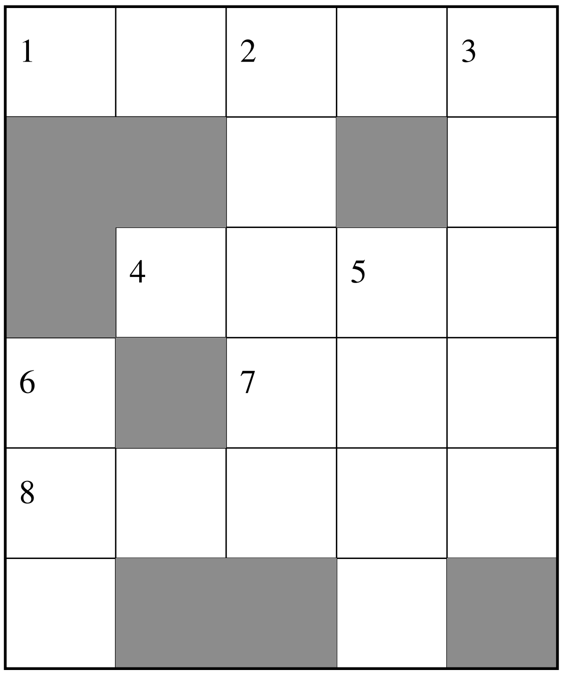
As D_a and D_b have been changed, the (only) other constraint C_2 which involves a and b should be added to the constraint queue if it is not already there. However, in this case, C_2 is already in the queue and therefore, no changes are made to the queue (instead of making the queue to become (C_2, C_2)). Although the domain of c has also been changed, this was not mentioned above since c does not appear in the other constraint.

Now, look at the second constraint $C_2 : b \geq a$. This time, $D_a = \{4, 5, 6\}, D_b = \{2, 3, 4\}$ (and $D_c = \{6, 7, 8\}$). Since $a \geq 4,\ b \geq 4$, the domain of b is reset to $\{4\}$. Also, since $b = 4,\ a \leq 4$, the domain of a is reset to $\{4\}$. As D_a and D_b have been changed, the temporary collection of changed variables returns $\{a, b\}$. This causes C_1 to be added to the constraint queue.

Look at C_1 again (due to its addition back to the queue). At this moment, $D_a = \{4\}, D_b = \{4\}$ and $D_c = \{6, 7, 8\}$. When the refinement is applied to b and, similarly, to a no changes take place to their domains. When it is applied to c, since $a = 4, b = 4$, constraint C_1 requires that $c = 8$ and hence that the domain of c be reset to $\{8\}$. Because only D_c has been changed and no other constraints involve c apart from C_1, nothing is added to the queue.

Given an empty constraint queue, the iterative refinement process terminates, returning the reduced domains of a, b, c as $\{4\}, \{4\}, \{8\}$, respectively. It happens that there is only one valid value for each variable, no application of any tree search mechanism is needed. The solution to the given CSP is $(4, 4, 8)$. Note that, in general, the Waltz algorithm will not return the exact solution but a reduced solution space for backtracking to search.

Example Two



AFT	LASER
ALE	LEE
EEL	LINE
HEEL	SAILS
HIKE	SHEET
HOSES	STEER
KEEL	TIE
KNOT	

(Mackworth, 92)

Figure 8.4: A crossword puzzle

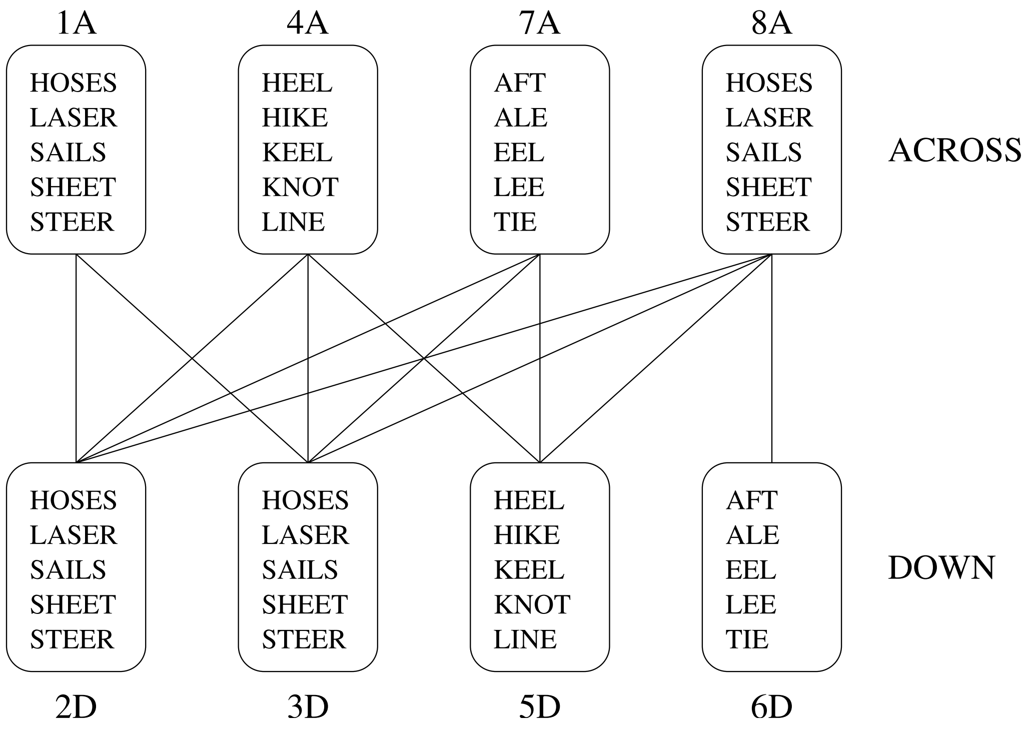


Figure 8.5: The crossword constraint network

This example is borrowed from *Encyclopaedia of AI* [Mac92]. Consider the puzzle in figure 8.4, in which the task is to fit some of the fifteen words given into the eight positions in a consistent way. A word can be used more than once. As it happens there is only one solution to this problem (as with the previous example). There are 5 five-letter words and 4 five-letter positions; 5 four-letter words and 2 four-letter positions; 5 three-letter words and 2 three-letter positions; so there are at most $5^4 \times 5^2 \times 5^2 = 5^8 = 390625$ combinations to consider. The number can, however, be vastly reduced by observing that there is a constraint wherever a vertical meets a horizontal. For example, 1-across cannot be `LASER` because then there would be no candidate for 3-down. Nevertheless, attempting to solve this problem would be difficult should generate-and-test be used.

Figure 8.5 gives a graphical representation of this CSP. For example, the line between `8A` and `6D` denotes the constraint between 8-across and 6-down. This constraint, as figure 8.4 should indicate, requires that the first letter of 8-across must also be the middle letter of 6-down. Thus all but `ALE` for 6-down and all but `LASER` for 8-across can be eliminated. Any change to a node (or variable) in figure 8.5 requires that every other node which is attached to that one be reconsidered. The process of re-applying the constraints continues recursively, triggered each time by any node change, until no further change needs to be made to any node.

The process of solving this CSP must terminate because each change is a deletion of one or more words at a node, and the worst that may happen is that no words get deleted. If all the words associated with a node are deleted, the original CSP is an over-constrained one – there is no solution possible for the problem. If, however, every node contains just one word when the process ends, then there is one solution as it is the case in this example. If some nodes contain more than one word there will be more than one solution. For example, if the words `VISOR` and `IVY` were added to the word list (these two words containing the only occurrences of the letter `V`) there would be two solutions: one in which 8-across was `LASER` and 6-down was `ALE`, the other in which 8-across was `VISOR` and 6-down was `IVY`. In order to keep track of such correspondences it is common to also label each line in the graph with the pairs of words from the nodes at the two ends of the line which together obey the constraint, and to modify the line label every time a node label is changed.

8.4 Qualitative Simulation as a CSP

Knowledge embedded in any reasoning system is always incomplete, as the real world is infinitely complex. A way to enable intelligent agents to cope with the demands of the real world, with incomplete knowledge, is to capture only the important distinctions in the description of the physical mechanisms while ignoring others. To address this, a fundamental review of the theories of describing physical systems and simulating their behaviour has been carried out. A seminal work for this is QSIM, or qualitative simulation [Kui86, Kui94], which works essentially by representing a physical system as a network of qualitative constraints, and by solving the resulting problem using the standard CSP techniques.

Note that although QSIM is herein given as a case of CSP applications, it itself offers an important method for knowledge representation. Thus, the following discussion about QSIM is presented in a fairly self-contained manner.

In QSIM, each system variable takes values from a domain termed *quantity space*, which is defined as a finite, alternating sequence of distinguished points and intervals between these points, on the real line. Such distinguished points are called landmarks and they are totally ordered within a quantity space. Thus, a quantity space is fixed if the landmarks involved are all identified (not necessarily in terms of their real value but symbolic name). That is to say, when defining a quantity space, only landmarks need to be explicitly mentioned.

Informally speaking, a landmark stands for a physically distinct value, e.g. a set-point or a maximum, which indicates the natural joint that breaks a continuous set of values into qualitatively different regions. A quantity space often includes three specific landmarks: $-\infty$, 0 and ∞, to represent limits of the domain, though they can be omitted if their explicit presence is not necessary.

At a given time t the qualitative state (QS) of a variable v is described by a pair of values:

$$QS(v,t) =< qval(v,t), qdir(v,t) >$$

The value $qval(v,t)$ represents the magnitude of the variable and can be a particular landmark or an open interval between two landmarks. That is, given L being the set of landmarks, $qval(v,t) = l_j$ if $v(t) = l_j \in L$, and $qval(v,t) = (l_j, l_{j+1})$ if $v(t) = l \in (l_j, l_{j+1})$. The value $qdir(v,t)$ represents the direction of change of v, with respect to time, and can take one of the three values dec, std and inc that stand for decreasing, steady and increasing respectively:

$$qdir(v,t) = \begin{cases} dec, & \text{if } v'(t) < 0 \\ std, & \text{if } v'(t) = 0 \\ inc, & \text{if } v'(t) > 0 \end{cases}$$

In summary, the qualitative state of a variable is represented by two values: a qualitative magnitude of a finite, but variable, precision and a qualitative derivative with a fixed precision of three symbols.

QSIM defines three classes of constraint primitives (or operators) as follows:

- arithmetic constraints:

$$ADD(x, y, z) : x(t) + y(t) = z(t)$$
$$MULT(x, y, z) : x(t) \times y(t) = z(t)$$
$$MINUS(x, y) : x(t) = -y(t)$$

- derivative constraint:

$$DERIV(x, y) : x'(t) = y(t)$$

- functional constraints:

$$M^+(x, y) : y(t) = f(x(t)), f \in \{M^+\}$$
$$M^-(x, y) : y(t) = -f(x(t)), f \in \{M^+\}$$

  where $\{M^+\}$ is, loosely speaking, the set of monotonically increasing functions over a certain open interval on the real line. The more rigid definition of $\{M^+\}$ is left out due to its mathematical sophistication.

Additionally, constraints can have one or more *correspondences* associated. Each correspondence is a pair or a triple of landmarks, depending on how many variables there are in the constraint it is associated with. A correspondence explicitly represents interrelated values for the constrained variables that are given *a priori*. For instance, the function relation $y(t) = \sqrt{x(t)}$ can be qualitatively represented as $M^+(x, y)$ with correspondences $(0, 0)$ and (∞, ∞). Also, an addition constraint always has an implicit correspondence associated with it, where each of the three variables involved takes the value of zero.

QSIM takes as input a set of variables, a set of constraints relating the variables and an initial qualitative state for each variable, and produces as output a tree of system states with each path representing a possible behaviour of the system under consideration. The inference process of producing system behaviours is termed qualitative simulation, and is implemented by repeated use of basic CSP solution techniques.

A system state consists of one qualitative state per variable, with all such variable states defined over the same time point or interval. The time used to describe a system behaviour is herein represented by an alternating sequence of distinguished time points and open intervals between a pair of such points. A distinguished time point is one at which there is at least one variable taking a landmark as its qualitative magnitude.

In terms of a single variable, say variable v, a qualitative behaviour that QSIM may generate, from a given initial state $QS(v, t_0)$, comprises an alternating sequence of qualitative states at its distinguished time points $(t_i, i = 1, 2, ...n)$ or over the time intervals:

$$QS(v, t_0), QS(v, t_0, t_1), QS(v, t_1), QS(v, t_1, t_2), ..., QS(v, t_n)$$

Here, $QS(v, t_i, t_{i+1})$ represents the qualitative state of v for any $t \in (t_i, t_{i+1})$, namely,

$$QS(v, t_i, t_{i+1}) = QS(v, t), t \in (t_i, t_{i+1})$$

How does QSIM generate successor states from a given initial state? It does this in a different way from what conventional numerical simulation methods would do. Given an initial system state, a set of all possible next states per variable is determined, forming the domains of each variable in terms of CSPs. The determination of such sets of possible next states is done by the use of versions of the intermediate value and mean value theorems, based on the assumption of system variables being continuous and differentiable functions of time. These versions of the theorems form a set of "transition rules" depending upon whether the system is currently at a distinguished time point, or at an open interval between time points. When transiting from a time point to an interval the set of P-transition rules applies whilst, when transiting from an interval to a time point the set of I-transition rules applies. These two sets of rules are shown in tables 8.3 and 8.4 [Kui86].

Rule-id	$QS(v, t_i)$	$QS(v, t_i, t_{i+1})$
P1	$< l_j, std >$	$< l_j, std >$
P2	$< l_j, std >$	$< (l_j, l_{j+1}), inc >$
P3	$< l_j, std >$	$< (l_{j-1}, l_j), dec >$
P4	$< l_j, inc >$	$< (l_j, l_{j+1}), inc >$
P5	$< (l_j, l_{j+1}), inc >$	$< (l_j, l_{j+1}), inc >$
P6	$< l_j, dec >$	$< (l_{j-1}, l_j), dec >$
P7	$< (l_j, l_{j+1}), dec >$	$< (l_j, l_{j+1}), dec >$

Table 8.3: P-transition rules

Rule-id	$QS(v, t_i, t_{i+1})$	$QS(v, t_{i+1})$
I1	$< l_j, std >$	$< l_j, std >$
I2	$< (l_j, l_{j+1}), inc >$	$< l_{j+1}, std >$
I3	$< (l_j, l_{j+1}), inc >$	$< l_{j+1}, inc >$
I4	$< (l_j, l_{j+1}), inc >$	$< (l_j, l_{j+1}), inc >$
I5	$< (l_j, l_{j+1}), dec >$	$< l_j, std >$
I6	$< (l_j, l_{j+1}), dec >$	$< l_j, dec >$
I7	$< (l_j, l_{j+1}), dec >$	$< (l_j, l_{j+1}), dec >$
I8	$< (l_j, l_{j+1}), inc >$	$< l, std >$
I9	$< (l_j, l_{j+1}), dec >$	$< l, std >$

Table 8.4: I-transition rules

It can be seen from these tables that in some cases the next state is uniquely defined. For example, if a variable is increasing (or decreasing) at a distinguished time point then it must transit into the next region depending upon the direction of change (see transition rules P4 and P6). Also, if a variable is between landmarks at a time point it must persist in the same interval until the following time point is reached.

In other cases, however, multiple possible next states exist, and it is impossible to discriminate between them based on assumed continuity and differentiability alone. For example, when a variable is increasing during a time interval, four possible next states are generated, namely I2, I3, I4 and I8. The first three of these correspond to whether it hits a landmark and stays steady (I2) or continues to increase (I3), or it continues to increase but still remains between landmarks (I4). The case of the variable becoming steady between known landmarks (I8) is catered for by creating a new landmark which splits the open interval into two. In fact, within rule I8 or I9 in table 8.4, l represents a previously unknown landmark which is just discovered and which satisfies that $l_j < l < l_{j+1}$. Thus, QSIM can generate new landmarks at run time as the behaviour evolves to previously unknown critical positions. This ability of QSIM to dynamically create new landmarks significantly extends the precision of the variable at the expense of complicating the inference algorithm; in the worst case, QSIM may never terminate.

Having generated the possible next states using the transition rules, restrictions over these possible states are then imposed by the use of the set of constraints which represent the relationships between the system variables. For each constraint, a set of magnitude-derivative value (i.e. state) pairs or triples is generated by computing the cross-product of the possible next states of the variables involved. Those pairs or triples inconsistent with the constraint are eliminated. Consistency between constraints, which share a variable, is then checked to further eliminate inconsistent states. For efficiency purposes, the Waltz algorithm is employed to perform such consistency checking between constraints. QSIM actually uses a modified version of the original Waltz algorithm, as summarised in figure 8.6.

```
Procedure $REVISE(C(v_1, ..., v_k))$

    begin $CHANGED \leftarrow \phi$
        for each constraint $C_j(v_{j1}, ..., v_{jn}) \neq C(v_1, ..., v_k)$
            and $\{v_{j1}, ..., v_{jn}\} \cap \{v_1, ..., v_k\} \neq \phi$ do
            begin $S \leftarrow REFINE(C, C_j)$
                if $S = \phi$ then halt
                else if $S \neq S_C$ then
                    begin $S_C \leftarrow S$
                        add $v_1, ..., v_k$ to $CHANGED$
                    end
            end
        return $CHANGED$
    end


Procedure WALTZ

    begin $Q \leftarrow$ a queue of all given constraints
        while $Q \neq \phi$ do
            begin remove constraint $C$ from $Q$
                $CHANGED \leftarrow REVISE(C)$
                for each $v_i$ in $CHANGED$ do
                    for each $C' \neq C$ which involves $v_i$ do
                        if $C' \notin Q$ then add $C'$ to $Q$
            end
    end
```

Figure 8.6: Waltz algorithm in QSIM

Within this modified algorithm, the notion of *refinement* is defined by

$$Refine(C, C_j) = \{(d_1, ..., d_k) \in S_C | \exists\, (d_{j1}, ..., d_{jn}) \in S_{C_j}, \exists\, i \in \{1, ..., k\},$$
$$\exists\, m \in \{1, ..., n\}, v_i = v_{jm} \rightarrow d_i = d_{jm}\}$$

where, for example, C is a constraint on variables $v_i, i = 1, ..., k$, with k being 2 or 3 (namely, the total number of the variables that C involves); S_C is the set of combinations of the possible states of these variables (at a time instance or interval), with each combination being consistent with C; and d_i is a state of v_i, represented in terms of its magnitude and derivative values. Thus, $Refine(C, C_j)$ denotes the set of state pairs or triples that are consistent with the constraint C and with another constraint C_j which shares a variable with C. Whether it is a set of pairs or triples depends on how many variables C involves.

After constraint propagation, complete system state descriptions are generated from the remaining pairs or triples and become the immediate successor states of the current state. As, at worst, each variable may have four next possible states and each constraint may include three variables, the next possible system states can be obtained by generate-and-test.

In general, the set of the resulting successor states contains more than one possible next state, due to the inherent ambiguity of qualitativeness. This ambiguity may lead to the generation of "spurious" behaviours that do not correspond to any physically possible behaviours of the real system. Some of these behaviours can be eliminated by applying so-called global filters derived from system theories or from other (external) knowledge sources. Typical global filters include check for no-change or divergence of a variable to infinity. Filters can also be used to check for periodic oscillation by looking for cycles of the state values. Detailed descriptions of various kinds of global filter are, however, beyond the scope of this module.

To illustrate how QSIM represents incomplete knowledge and performs qualitative simulation, consider a simple physical system (modified from [Kui86]) consisting of a ball thrown upwards in a constant gravitational field. This system can be modelled by the following three constraints relating three system variables: the height x, the velocity v and the constant acceleration a of the ball such that

$$DERIV(x, v), DERIV(v, a), a = g$$

where g is a fixed, negative real number.

For simplicity, the quantity space for both variables x and v is defined as $\{-\infty, 0, \infty\}$ (note that, here, only landmarks get mentioned). Due to the constant gravity, the quantity space for a is fixed to the single-valued set $\{g\}$. Suppose that the initial condition is that, at time t_0, the ball is just released from the hand and thrown up. That is, the initial system state can be described by

$$QS(x, t_0, t_1) =< (0, \infty), inc >, QS(v, t_0, t_1) =< (0, \infty), dec >, QS(a, t_0, t_1) =< g, std >$$

According to the I-transition rules given in table 8.4 (think why no P-transition rules are applicable), the set of possible next states for the system variables are obtained as shown in table 8.5. Again, for simplicity, it is (correctly) assumed that the possibility that $x = \infty$ is excluded, and hence transition rules I2 and I3 are not used to produce the possible next states of x nor listed in this table.

Given these possible transitions, cross-products of the possible next states are found with respect to the variables involved in each constraint. The results are presented in table 8.6, with the result concerning the third constraint which only involves one constant omitted. In this table, those marked with 'c' are removed by checking for constraint self consistency, and those marked with 'p' are deleted by checking for pairwise consistency between constraints. For instance, the state tuple $\{< (0, \infty), inc >, < 0, std >\}$ for variables x and v is eliminated by self consistency filtering with regard to the constraint $DERIV(x, v)$, since otherwise it would require x to increase while $v = 0$, an obvious inconsistency. The state tuple $\{< (0, \infty), inc >, < H_v, std >\}$ is eliminated by pairwise consistency filtering, since it finds no remaining tuple associated with the constraint $DERIV(v, a)$ where the variable v might take the state $\{< H_v, std >\}$.

Variable	**Rule-id**	Current State	Next State
a	I1	$< g, std >$	$< g, std >$
v	I5	$< (0, \infty), dec >$	$< 0, std >$
	I6	$< (0, \infty), dec >$	$< 0, dec >$
	I7	$< (0, \infty), dec >$	$< (0, \infty), dec >$
	I9	$< (0, \infty), dec >$	$< H_v, std >$
x	I4	$< (0, \infty), inc >$	$< (0, \infty), inc >$
	I8	$< (0, \infty), inc >$	$< H_x, std >$

Table 8.5: State transitions

$DERIV(x, v)$	$DERIV(v, a)$
$\{< (0, \infty), inc >, < 0, std >\}$ c	$\{< 0, std >, < g, std >\}$ c
$\{< (0, \infty), inc >, < 0, dec >\}$ c	$\{< 0, dec >, < g, std >\}$
$\{< (0, \infty), inc >, < (0, \infty), dec >\}$	$\{< (0, \infty), dec >, < g, std >\}$
$\{< (0, \infty), inc >, < H_v, std >\}$ p	$\{< H_v, std >, < g, std >\}$ c
$\{< H_x, std >, < 0, std >\}$ p	
$\{< H_x, std >, < 0, dec >\}$	
$\{< H_x, std >, < (0, \infty), dec >\}$ c	
$\{< H_x, std >, < H_v, std >\}$ c	

Table 8.6: Constraint filtering

After constraint filtering, the remaining state tuples can be formed into two global interpretations as presented in table 8.7. Each interpretation stands for a possible system state. The first is the same as the given initial state and, therefore, no further state generation needs to be done from it, as the same generation-and-filtering process just described would otherwise be repeated. It is this type of filtering that is called the “no-change” global filtering. This filtering method may be applied after a number of further states have been generated where a possible system behaviour cycles, though not cycling just one state ahead.

As every time interval must have an endpoint, after the no-change filtering, the solo remaining possible next state becomes the unique successor of the initial state. This defines a new landmark value H_x for the variable x, reflecting the maximum height that the ball can reach (at some time t_1).

x	v	a
$< (0, \infty), inc >$	$< (0, \infty), dec >$	$< g, std >$
$< H_x, std >$	$< 0, dec >$	$< g, std >$

Table 8.7: Global interpretations

Although only one next state is generated from the initial one for this toy system, it is very important to reiterate that, in general, QSIM generates as output a tree of possible behaviours, including spurious ones, due to qualitative ambiguity. It is also important to note that the behaviour tree does, however, include the ‘real’ underlying behaviour of the system, if this real behaviour is captured by a well-posed numerical model and if QSIM’s qualitative constraints are abstracted from the numerical one (as often the case). A detailed discussion about the spurious behaviours and a full, up-to-date treatment of QSIM can be found in [Kui94].

8.5 Discussions

The techniques of constraint satisfaction are ubiquitous in artificial intelligence, mainly due to the simplicity of the structure underlying a constraint-based representation. A variety of different problems can, when formulated appropriately, be seen as *classical* CSPs which are imperative and which are either fully satisfied or fully violated. For such problems, the solution methods introduced above can be successfully applied.

As the techniques of constraint satisfaction have been applied to real-world problems, it has become increasingly clear that classical constraints are not able to capture the full subtlety of such problems. In light of this, recent approaches have extended the classical CSP framework to create *flexible* constraint satisfaction techniques. A simple example is the expression of preferences among the set of assignments that constitute a solution. A preference is not a hard constraint, since it is ignored if it cannot be satisfied. Furthermore, classical constraint satisfaction cannot efficiently support problems whose structure is subject to change (though the Waltz algorithm may support this for constraint propagation in principle). To address this type of problem, the techniques of *dynamic* constraint satisfaction have also been proposed. This allows constraints to be added to and removed from a given problem, with as much as possible of the (partial) solution obtained before it changed being reused. See [MS99] for an introduction to work on such extensions.

Chapter 9

Reasoning Maintenance

9.1 Introduction

Sometimes it is necessary to make guesses or assumptions in one's reasoning, but those guesses or assumptions might turn out to be wrong so that all that followed from them becomes potentially invalid. In what is called a *monotonic* logic, new conclusions can never invalidate what is already known or has been deduced; in *non-monotonic* systems they can. How then, does one keep track of how new deductions depend on earlier information?

The cheap and nasty, but sometimes workable, approach is to attach a 'time tag' - usually this just means a sequence number - to each deduced fact or rule. (Note that a rule here does not have to be a production rule but can be a procedure, a constraint or some other atomic form of knowledge representation.) If a fact/rule added first at time T is, in subsequent reasoning, invalidated, then just erase it and also anything deduced since then. Just restart all reasoning from time $T-1$, adding just enough to stop following the same line of reasoning as before. This is clearly uneconomic:

```
Time T: add "earth is flat"
Time T+1: add "Edinburgh is the capital of Scotland"
Time T+2: contradict "earth is flat"
```

– it is wasteful to erase the second fact which does not depend on the first. Better approaches to the problem are described below.

In a truth or reasoning maintenance system (alias TMS or RMS), facts and rules are recorded as labels of nodes. The node names are a convenient shorthand for these facts and rules. A node can have zero or more justifications, a justification being an associated record describing how that fact or rule came to be known. Thus, a network of nodes and the interlinking justifications gets built up, like a tower of scaffolding. Some of this tower is rooted in unequivocal facts, but partly it is rooted in potentially invalid assumptions.

There are two main flavours of TMS. In Doyle's formulation [Doy79], a tower of deductions gets built, but sometimes the reasoning engine may signal that a certain deduction cannot be sustained; that part of the tower must fall. The TMS's job is to decide which piece of the scaffolding might have given way to cause this, and usually there are several assumptions that might take the blame. The TMS blames one and then demolishes such of the scaffolding that depends on it, in such a way that if the blame is later found to have been wrongly assigned all the demolished scaffolding can be resurrected without further work by the reasoning engine. In de Kleer's formulation [dK86], all possible scaffolding is built up from the initial facts and assumptions by forward chaining, so that the

TMS discovers all the deductions that might be supportable by some kind of scaffolding. For each such deduction it records all the sets of assumptions that could be used to support it. This chapter introduces these two reasoning maintenance systems, whilst the module focuses on the latter which is far more popular in use now.

9.2 Doyle's TMS

In this version, a justification may be valid or invalid – usually invalid justifications started life as valid ones. Any node that has at least one valid justification is said to be IN (that is, 'in the set of believed nodes') and those with no valid justification are said to be OUT ('of the set of believed nodes'). In practical systems, OUT nodes that have no use are usually erased after a short while to save space. Note that an OUT node may have a use, for instance it may be mentioned in a valid justification of an IN node, since it may happen that one thing is believed because another is not.

In this type of TMS, there are two kinds of justification:

- *Support List* justifications, of the form

  (SL *INlist OUTlist*)

  where *INlist* and *OUTlist* are each lists of nodes. The justification is valid if all the nodes in the *INlist* are IN, and all those in the *OUTlist* are OUT.

- *Conditional Proof* justifications, of the form

  (CP *node INlist OUTlist*)

  which is valid if *node* is IN whenever all the nodes in *INlist* are IN and all those in *OUTlist* are out.

Here are some observations about these:

- A fact or rule can be made an axiom by giving the node a justification of

  (SL () ())

  so that it is permanently IN.

- Nodes that have a non-empty *OUTlist* are assumptions; there are (at least apparent) conditions under which they could be contradicted.

- CP nodes are useful in supporting deduced rules, e.g.

Node	*Fact/Rule*	*Justifications*
n_1	A	maybe none valid
n_2	B	maybe none valid
n_3	if A then B	(CP n_2 (n_1) ())
n_4	C	maybe none valid
n_5	if C and A then B	(CP n_2 (n_3 n_4) ())

- A reasoning maintenance system is *not* a reasoning system; it is a slave of a reasoning system.

A TMS allows default reasoning. For example, suppose that $F_1 \dots F_n$ are nodes representing a range of choices, e.g. F_1 represents 'lecture on Monday' and F_7 represents 'lecture on Sunday'. Imagine that node G is a rule that calls for the making of a default choice. Node F_i can be made the default by giving it a justification of the form

(SL (G) $(F_1\ F_2 \dots F_{i-1}\ F_{i+1} \dots F_n)$)

so F_i will be IN if G is IN and there is no reason to believe in the competitors of F_i (that is, they are OUT). Sometimes it is impossible or inconvenient to enumerate the competing choices like this. Then a node F can be made the default by a simpler expedient. Create, if it doesn't exist, a node to represent the negation of what F represents; call this E. Then F is made the default by giving it the justification

(SL (G) (E))

and giving E various justifications such as

(SL (F_i) ())

for those F_i explicitly known to be competitors of F. Thus, if any F_i is IN, then E will be IN and F will be OUT. Otherwise, F will be IN whenever G is IN and E is OUT. As an example, consider the question of a person's salary:

Node	*Fact*	*Justifications*
n_1	salary=10000	(SL (...) (n_2))
n_2	salary≠10000	(SL (n_3) ()) (SL (n_4) ())
n_3	salary=9314	...
n_4	salary=11206	...

Therefore n_1 will be IN whenever a choice is required, unless any of n_2, n_3 or n_4 is IN for some other reason.

A TMS permits dependency-directed backtracking, that is, it allows the reasoning system to decide to retract some assumption without setting up contradictions caused by continuing to believe any consequence of a retracted assumption. It is worth noting that the TMS itself has no built-in notion of inconsistency; it is up to the reasoning system to detect them and to let the TMS know.

The algorithm typically used to implement this type of TMS is mildly ingenious. Imagine that N is some node representing a fact or rule that the reasoning system finds does not hold, that is, it is a contradiction. The algorithm is:

- Trace the dependencies backward from N (which is currently IN and causing trouble) to find the set of assumptions that support it. Find the maximal ones – those that do not support other assumptions. Suppose that these are nodes $A_1 \dots A_n$.

- Create a new node NG (called a "no good") to represent the occurrence of the contradiction, e.g. the rule

  not (A_1 and ... and A_n)

  Give it justification

  (CP N $(A_1 \dots\dots A_n)$ ())

which means it is currently IN.

- Rather than being sophisticated, just pick a culprit at random from the set of maximal assumptions $A_1 \ldots A_n$. The aim is to force it OUT, so that N will also be OUT. Suppose that the culprit chosen is A_i.
- A_i is an assumption, so has a non-empty *OUTlist* in a valid justification. Find the set of nodes $D_1 \ldots D_k$ which are currently OUT, such that if any one came IN than A_i would be OUT. Pick one, say D_j.
- Give D_j a new valid justification, so it comes IN. A suitable justification would be

  $$\text{(SL } (NG\ A_1 \ldots A_{i-1}\ A_{i+1} \ldots A_n)\ (D_1 \ldots D_{j-1}\ D_{j+1} \ldots D_k))$$

  Now D_j will be IN, so A_i will be OUT, so N will be OUT. If any other D comes in, or if any other A goes OUT then D_j will be OUT. This may bring A_i back IN without bringing N back IN, so the arbitrary choice of A_i is not irrevocable. Also, the node NG records the essence of the contradiction, so that the reasoning system will not be troubled with that particular one again, at least in that form.

A small example may give the flavour. Suppose that a reasoning system is trying to solve a timetabling problem, and assumes that some lecture is going to happen at 10am at Forrest Hill:

Node	Fact	Justifications
n_1	time=10am	(SL () (n_2))
n_2	time$\neq$10am	...
n_3	place=FH	(SL () (n_4))
n_4	place=HPSq	...

Currently, the IN set is $\{n_1, n_3\}$. Suppose that the reasoning system now finds the problem:

n_5	(the snag)	(SL (n_1 n_3) ())

That is, some node n_5 turned up, assuming 10am at Forrest Hill, and the reasoning system then found out that this was not possible. The IN set is now $\{n_1, n_3, n_5\}$. The maximal assumptions for n_5 are n_1 and n_3. To clear up the trouble, introduce the "no good"

n_6	not (n_1 and n_3)	(CP n_5 (n_1 n_3) ())

Pick on n_3. It has one OUT node, n_4, which can be brought IN to take it out. Thus, give n_4 the justification

n_4	place=HPSq	(SL (n_6 n_1) ())

The IN set is now $\{n_1, n_4, n_6\}$; the problem has been cured. The existence of n_5 and n_6 stops the reasoning maintenance system from allowing the reasoning system to make the same mistake in future.

In fancier applications, the reasoning system may be used to help the TMS make a more rational choice of culprit than just picking one at random. However, the entire algorithm is not flawless. Suppose that the TMS has been asked to force some newly-discovered contradiction C to be OUT, and the nature of the contradiction is that it depends on, say, an assumption (not C) which is currently IN. Arranging for (not C) to be OUT may cause C to go OUT, by the algorithm, but then this may cause C to come back IN thanks to the rule $C \vee$ (not C) being always IN. Thus the TMS may loop forever. Although it would be easy to build in a check against this simple kind of flaw, there are much more elaborate forms of it that can be very hard to spot.

9.3 de Kleer's ATMS

In de Kleer's version, named assumption-based TMS (ATMS), justifications are always valid. They are made from rules and facts which hold with some sets of assumptions. ATMS does not care about whether any node or assumption is actually believed – there is no question of IN and OUT. Instead, it records only how deductions would depend on any assumptions. That is, the job of ATMS is to record under what sets of assumptions any conclusion holds.

In ATMS, each node has, in addition to the tag showing what it stands for (what you would normally call its label), something else which de Kleer confusingly terms a *label*; this is a set of sets of assumptions. The node follows from any of such sets of assumptions. If a node turns out to be contradictory, according to the reasoning system, then the ATMS notes that all the sets of assumptions labelling it lead to a contradiction. The convenient way to do this noting is to have a node which stands for 'false', and to add any such contradictory sets to its label. For each node, a check is made such that none of its sets of assumptions is a superset of any other; if so, it should be deleted. The ATMS does this to ensure that no superfluous assumption appears anywhere in a set within any label.

To have a look at the idea of how ATMS functions, suppose that early on in the deductive process (performed by a reasoning engine) there have been nodes for assumptions $a_1 \dots a_5$, and that there are already two nodes A and B, with labels as follows

A	$\{a_1, a_2\}$ $\{a_2, a_5\}$
B	$\{a_1\}$ $\{a_2, a_3\}$ $\{a_4\}$

For instance, the label of node B means that B holds if a_1 does, or if a_2 and a_3 do, or if a_4 does. Suppose also that the only contradiction so far discovered is that a_4 and a_5 together lead to an inconsistency. This means that the label of the specific 'false' node currently contains just one set of contradictory assumptions: $\{a_4, a_5\}$.

Now, the reasoner wants to add another justification, that A and B imply C. The ATMS records the justification, and from the labels for A and B creates a new label for C as follows (if C is new, a node for it is first created):

- Find the collection of pairwise unions of sets of assumptions, with one set per pair from each label (akin to the cross-product of the labels of A and B); this is

(1)	$\{a_1, a_2\}$
(2)	$\{a_1, a_2, a_3\}$
(3)	$\{a_1, a_2, a_4\}$
(4)	$\{a_1, a_2, a_5\}$
(5)	$\{a_2, a_3, a_5\}$
(6)	$\{a_2, a_4, a_5\}$

- Remove any which has another as a subset. In this case, sets (2), (3) and (4) each have (1) as a subset. This step removes any superfluous dependence on assumptions. Thus only sets (1), (5) and (6) remain.

- Remove any set which has a contradictory set of assumptions as a subset of it. Since $\{a_4, a_5\}$ is already known to be contradictory, this disposes of set (6), leaving sets (1) and (5) to form the new label for C.

- If C is new, this becomes its label. If not, this label is blended with the previous label: take the union of the two labels, delete any set in this union which has another member in the union as a subset of it. Since the new and old labels were already consistent there is no need to go looking for sets which have contradictory subsets, this time.
- If the whole process changes the label for C, and recorded justifications show that other nodes depended on C, then the label changes have to be propagated forward to those other nodes, and on from them, and so on.

In the above illustration, if, at a certain stage, C is found to be a nogood by the reasoning system (that is, every set of assumptions labelling it indicates a contradiction), then each member of its label is added to the label of the specific node standing for 'false', and all such members, and their supersets, are removed from the label of every other node. Any set of (inconsistent) assumptions being a superset of another within the 'false' node's label itself is also removed.

ATMS has two advantages. First, all the consequences of a set of assumptions can be explored together; no backtracking is involved, and the system is not striving to maintain one mutually consistent set of assumptions as in Doyle's TMS. This suits certain kinds of application. Second, it can be very efficiently implemented, since the main operations involved are set operations such as union and subset checking. If sets are represented as bit strings these kinds of operations can be performed directly by hardware. The obvious disadvantage is that the system is driven by forward-chaining reasoning, so there is no natural progression towards a particular desired goal.

There is considerably more to ATMS than this; this section has only covered the basic idea. For example, in some applications it might be desirable to constrain the ATMS to considering only those sets of assumptions which contain at least one, or perhaps exactly one, of a given set (of particular assumptions). This calls for adding new justifications automatically whenever the ATMS throws up a set that violates such constraints – this is much more economical than adding all the justifications that these constraints entail right from the beginning. See various papers by de Kleer in the *AI Journal* for further details.

9.4 A Caution

In practice, whatever the flavour of TMS, there are further complications to consider. For example, consider the following rule:

```
if X is burning
and X is on Y
and <..various physical conditions
       indicating Y will char..>
then Y is charred
```

Even when 'X is burning' is no longer true, the conclusion 'Y is charred' will be true. However, this conclusion might be marked as OUT in Doyle's TMS or retracted in de Kleer's ATMS along with the retraction of 'X is burning'. In this case the trouble really lies with the rule rather than the TMS, but the example does suggest that there are special difficulties in handling time-based rules in which the effect lasts even when the cause does not. Some solutions to this can be taken from recent work on automated planning and/or temporal reasoning. They are not described here, however.

Chapter 10

Model-Based Reasoning

10.1 Introduction

One of the recent trends in developing intelligent problem solvers has been the building of *model-based reasoning* systems. Imagine that you have built a knowledge-based system to diagnose faults in coin-in-the-slot telephones and to suggest cures. If the symptom is that your money is not returned after an unsuccessful call, bash the telephone sharply on its left-hand side at a point about two inches lower than the level of the coin slot. However, when BT[1] replaces these machines with slightly more modern coin-in-the-slot boxes you would need to amend the knowledge-based system. It would be nicer if the system could reason out for itself where you should hit the box, by considering a model of the mechanism. This, crudely, is the idea behind the model-based systems.

The term 'model' may refer to the description of the structure and behaviour of a physical mechanism. It could equally be a description of a social or business process, or that of biochemical balances within the human body. The common idea behind all such descriptions is that a model represents a simulation of something, a simulation which is faithful and complete at some sensible level of detail. For example, a model of how an electric refrigerator works need not be at the level of the physical layout of the parts. Rather, it might be in terms of how an electric motor compresses a gas, whose later expansion causes the gas to cool and the cooled gas then circulates through pipes within the refrigerator drawing heat out of it by conductivity through the walls of the pipes, and how convective cooling cools those parts not close to the internal pipes.

Diagnosis has been, and still remains, the largest application category of most intelligent application systems. This chapter is, therefore, focused on discussions of model-based systems which perform diagnostic tasks. Although it addresses the issues of finding faults in physical systems the techniques introduced can be applied to other domains.

To determine why a physical system has not worked correctly compared to its design intention, it is useful to know how it was supposed to work in the first place. Reflecting this viewpoint, the basic paradigm of model-based diagnosis can be best seen as the interaction of observations, obtained from the physical system under diagnosis, and predictions, generated from the system's model (which is usually qualitative). Observations indicate what the system is actually doing whilst predictions indicate what it is intended to do. Thus, this approach essentially depends on the use of an explicit structured model for behaviour prediction. Whenever there exists a difference, or a discrepancy, between observations and predictions, it is presumed that the system is not working correctly and certain faults have occurred within the system. Given detected discrepancies a model-based diagnostic

[1]BT = British Telecom

system is set to identify, ultimately, which of the system components could have failed and led to such discrepancies.

It is not unusual for there to be multiple faults simultaneously. One of the drawbacks of early rule-based approaches was an unwarranted assumption that any one symptom had a single cause, so when one cause had been diagnosed all the symptoms it accounted for could be ignored when trying to account for the remaining symptoms. Many model-based approaches, however, provide a reasonably successful method for trying to handle multiple faults. A representative of these, the General Diagnostic Engine (GDE) [dW87] is introduced in the next section, in terms of isolating multiple fault candidates efficiently via the use of an ATMS. GDE also provides a mechanism for differentiating among the set of possible candidates, which incorporates probabilities and information theory into a unified framework. An introduction to this second part of GDE is given in section 10.3.

Since its birth, GDE has been widely applied and tested in a variety of problem domains, and its limitations as well as its strengths in performing diagnosis have been extensively studied in the literature. As a result, many improvements and extensions have been made to GDE (see [WHdK92]). Among them, one of the best known is the GDE+ system [SD89]. An overview of this system is presented in section 10.4, where another extended system, Sherlock [dKW89] is also outlined to show the diversity of different modifications possible.

10.2 Postulation of Multiple Faults

GDE is intended to isolate or locate multiple simultaneous faults in a device consisting of interconnected components. It makes a number of important assumptions:

- Faults are in components rather than in any interconnections (though interconnections could be represented as components themselves).
- The device representation is faithful; there are for instance no interconnections in the real device which are not also present in the model.
- Faults are not intermittent.

To illustrate the kind of problem it can deal with, consider a much-used example system as shown in figure 10.1.

According to the design of this device, the three multipliers, M_1, M_2 and M_3 are each supposed to multiply their two inputs, and the two adders, A_1 and A_2 are supposed to add their two inputs. Thus, given the inputs:

$$A = 3, B = 2, C = 2, D = 3, E = 3$$

to a working version the result will be:

$$F = 12, G = 12$$

However, if $F = 10$ instead it might be suggested that the problem was that M_1 was only outputting 4 instead of 6. The trouble is that this is not the only diagnosis, even though it makes good sense to consider it first. The difficulty then is to find the others. Another might be that M_2 was outputting 4, A_1 was working whilst A_2 was faulty but just compensating for the error made by M_2. Another could be that M_2 was erroneously outputting 4 and M_3 was erroneously outputting 8, but the two adders were working properly. Of course it is even possible that every component is faulty!

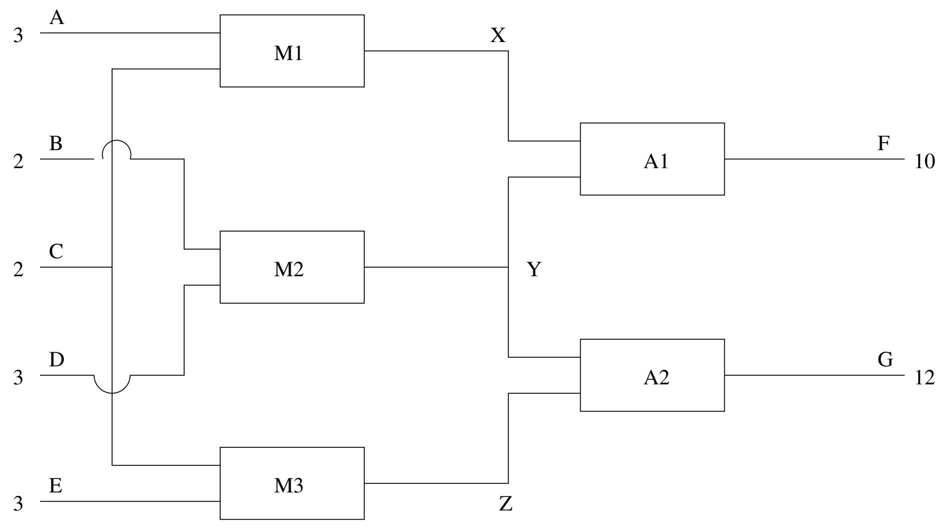


Figure 10.1: A familiar circuit

The idea behind GDE is to treat the initial observations:

$$A = 3, B = 2, C = 2, D = 3, E = 3, \quad F = 10, G = 12$$

as facts and to treat statements such as 'M_1 is working correctly' as assumptions (with a set of assumptions termed an environment for easy reference). An ATMS can then be used to keep track of what deductions depend on what environments. For example, for notational simplicity (and hopefully without causing confusion), let M_1 stand for the assumption that the multiplier M_1 is working, let A_2 stand for the assumption that the adder A_2 is working and so on. Thus, the following extra 'nodes' can be generated under the assumptions given in $\{\ldots\}$, though not necessarily in this order:

n1: $X = 6\ \{M_1\}$

n2: $Y = 6\ \{M_2\}$

n3: $Z = 6\ \{M_3\}$

n4: $Z = 6\ \{M_2, A_2\}$ (from **n2** and $G = 12$)

n5: $X = 4\ \{M_2, A_1\}$ (from $F = 10$ and **n2**)

Note that already **n5** conflicts with **n1**, so it cannot be the case that all of M_1, M_2 and A_1 are working. From this, GDE deduces that at least one of the components M_1, M_2, and A_1 is faulty and introduces the notion of *minimal candidates*: $\{M_1\}, \{M_2\}, \{A_1\}$. The process can continue:

n6: $Y = 4\ \{M_1, A_1\}$ (from $F = 10$ and **n1**)

This produces the same *conflict* as before. The process continues:

n7: $Z = 8\ \{M_1, A_1, A_2\}$ (from $G = 12$ and **n6**)

Now **n7** disagrees with **n3**, so it cannot be the case that all of M_1, A_1, A_2 and M_3 are working. This is a different conflict. Given the following two assertions:

- By **n5** and **n1** or **n6** and **n2**, at least one of $\{M_1, M_2, A_1\}$ is faulty.
- By **n7** and **n3**, at least one of $\{M_1, M_3, A_1, A_2\}$ is faulty.

GDE deduces that at least one of the following sets of components is faulty, and treats each of them as an updated minimal candidate:

$$\{A_1\}, \{M_1\}, \{A_2, M_2\}, \{M_2, M_3\}$$

The reason behind this deduction can be best shown by examples. The set $\{A_1\}$ is considered a minimal candidate because A_1 belongs to both conflicts in the above two assertions, if it is faulty both assertions are explained. Similarly, the set $\{A_2, M_2\}$ is considered a minimal candidate because if A_2 and M_2 are faulty simultaneously they jointly explain both assertions. However, unlike A_1, one and only one of $\{A_2, M_2\}$ being faulty does not form a consistent explanation for these two assertions.

For this simple device, given the above minimal candidates, it should be clear that attention is to focus on M_1 or A_1. This suggests that the value at X should be measured, in order to better isolate the possible faults. The measured value of X becomes a new fact, and the process continues.

Electrical engineers will realise that this whole process is a bit suspect. To start with, a common kind of fault is for there to be an open (broken) connection somewhere or for there to be a spurious extra connection caused by faulty manufacture; the underlying assumption that any fault lies within a component is too restrictive. Further, even if the fault is in a component it can happen that components fail in more interesting ways than just producing the wrong output for given inputs. For example, if M_3 fails it may drag input C down to 0, causing an input to M_1 to be 0 too. This could perhaps be handled within GDE using a more elaborate scheme of representation and assumptions. A more serious problem is that the number of deductions from assumptions and other intermediate deductions is going to proliferate exponentially as the circuits get more complicated. GDE brings in component failure probabilities in an attempt to deal with this, by ignoring any combination of faults which seems too unlikely to pursue in the meantime [dK90].

10.3 Choosing Measurements for Candidate Discrimination

Using the above method for fault candidate generation, or indeed one of the many other model-based techniques, can lead to a quite large number of postulated faults. In order to reduce the remaining candidates to make the diagnostic result practically more useful, a diagnostic system may use further measurements to differentiate amongst the candidates. GDE offers a method for choosing the best measurements which would most effectively distinguish between the remaining candidates. A best measurement in this context is one which will, on average, allow the discovery of the actual fault(s) in the physical system using a minimum number of subsequent measurements [dKW89].

Before showing how to choose the best measurements, it is useful to examine what consequences a possible measurement may bring to the set of generated candidates. Recall that, in GDE, each model-based prediction is treated as a fact and stored as a node in the ATMS with the set of minimal environments supporting it as its label. Here, each environment is a set of assumptions made to support a partial, behavioural description of the physical system. For instance, a value v of a variable x, which is predicted under the environments $e_1, e_2, ..., e_m$, can be represented by

$$< x = v, \{e_1, e_2, ..., e_m\} >$$

In general, a variable x may have more than one predicted value, each having a set of supporting environments. If x is measured to be v then the supporting environments of any value different from it are necessarily conflicts, since they did not lead x to taking v as its true value. If, however, the measurement is not equal to any of the predicted values of x, then every supporting environment for each predicted value of x must be a conflict.

Consider the electrical device shown in figure 10.1 again. Recall that, given the facts that $A = 3, B = 2, C = 2, D = 3, E = 3, F = 10, G = 12$, and the identified conflicting environments that led to contradictory predicted and/or measured values for individual variables (e.g. environment $\{A_1, M_2\}$ or $\{A_1, A_2, M_3\}$ led to $X = 4$, while environment $\{M_1\}$ resulted in $X = 6$), GDE deduces the following minimal candidates, meaning that at least one of these sets of components must be faulty:

$$\{A_1\}, \{M_1\}, \{A_2, M_2\}, \{M_2, M_3\}$$

Suppose that a new measurement is made at point X. This measurement will result in three possible outcomes:

- $X = 4$, in this case environments $\{A_1, M_2\}$ and $\{A_1, A_2, M_3\}$ do not lead to a contradictory value for X, but $\{M_1\}$ does and forms a conflict (as it supports $X = 6$ rather than $X = 4$), the minimal candidate now becomes $\{M_1\}$.
- $X = 6$, in this case $\{A_1, M_2\}$ and $\{A_1, A_2, M_3\}$ are conflicts and the new minimal candidates are $\{A_1\}$, $\{M_2, M_3\}$ and $\{A_2, M_2\}$.
- $X \neq 4$ and $X \neq 6$, in this case $\{A_1, M_2\}$, $\{A_1, A_2, M_3\}$ and $\{M_1\}$ are conflicts and the minimal candidates are $\{A_1, M_1\}$, $\{M_1, M_2, M_3\}$ and $\{A_2, M_1, M_2\}$.

The above example illustrated the effect of an appropriate new measurement upon the discrimination between possible fault candidates. For this example, measurement at X is selected to explain the potential consequences since, intuitively, measuring X is likely to produce the most useful information to further isolate the faults. This is because the two (more probable) singleton candidates $\{A_1\}$ and $\{M_1\}$ are only differentiable when such a measurement is made. For this toy device, it is feasible to recognise this by looking at the circuit diagram. However, when the system structure becomes more complex, how can a best measurement point be automatically chosen?

GDE gives an answer to this question by adopting a one-step lookahead strategy based on Shannon's information theory. This strategy requires the consequences of each possible measurement to be analysed with respect to a given set of (minimal) fault candidates, in order to determine which measurement to actually make next. The following entropy function is used as the basis for the analysis of the possible consequences of *hypothesised* measurements:

$$-\sum_i p_i log(p_i)$$

where p_i denotes the probability that the i-th (remaining) candidate would be the actual culprit given the hypothesised measurement outcome.

This entropy function has several important properties. In particular, if every candidate is equally likely then it will reach a maximum value, indicating that little information is available for differentiating among the candidates. If, however, one candidate is more likely than the rest this function will approach its minimum. Thus, the best next measurement should allow the minimisation of the expected entropy of candidate probabilities. From this, and by substantial algebraic manipulations, it can be derived that the best measurement is the one which minimises (over all possible measuring points x_i within the model of the physical system):

$$\sum_{k=1}^{m}[p(S_{ik}) + \frac{p(U_i)}{m}]log[p(S_{ik}) + \frac{p(U_i)}{m}] - p(U_i)log\frac{1}{m}$$

Where m is the number of possible values for variable x_i, S_{ik} is the set of candidates in which x_i must be of a certain value v_{ik} (equivalently, the set of candidates which will be eliminated if x_i is measured not to be v_{ik}), and U_i is the set of candidates which will not be eliminated no matter what value is measured for x_i (equivalently, the set of candidates which do not predict a value for x_i).

This method relies on the availability of failure probabilities for system components, in order to calculate the probabilities $p(S_{ik})$ and $p(U_i)$. However, in many cases this information is unknown. In addition, this method can be computationally very expensive if there are many points in a device that could be measured and/or if each variable may take on many possible values. To avoid these problems, de Kleer proposed a considerably simplified version of this minimum entropy technique [dK90], assuming that all components fail independently with equal probability (which is somewhat suspect since dependent failures can be very common in some domains and certain components may fail more often than others), and that components fail with a very small probability.

Supported with strong assumptions, the simplified version states that the best measurement is the one which minimises:

$$\sum_k c_{ik}log(c_{ik})$$

where c_{ik} is the number of those remaining fault candidates which have the minimum cardinality N *and* which predict that the value of variable x_i is v_{ik}. Here, N being the minimum cardinality means that all candidates with less than N components have been exonerated from suspicion.

What this method implies is that, to determine the best next measurement, it is only necessary to consider the fault candidates of minimum cardinality, i.e. those candidates containing the least number of components. In fact, given the minimum cardinality being N, every candidate with less than N possibly faulty components has been eliminated and every candidate having more than N components has a negligible probability in comparison with any candidate of cardinality N. Thus, this one-step lookahead strategy always first proposes measurements which will differentiate amongst single-component fault candidates (with $N = 1$ being the smallest cardinality possible). If all single-component candidates are eliminated, it proposes measuring points which differentiate amongst double-fault candidates, and so on.

It is important to notice that calculations involved in this method are very simple. More importantly, no actual probability values are required to perform such calculations. Only the counts of the number of candidate faulty components that support each hypothesised measurement outcome are needed. Therefore, this method can be applied even when the exact probability values are unknown, provided that assumed conditions are satisfied such that faults occur independently and with a very small, equal probability.

Now, go back to the example of figure 10.1. Given the inputs, $A = 3, B = 2, C = 2, D = 3, E = 3$, and the measured outputs $F = 10$ and $G = 12$, there are two single-component fault candidates: $\{M_1\}$ and $\{A_1\}$. In other words, $\{M_1\}$ and $\{A_1\}$ are the remaining candidates which have the minimum cardinality of $N = 1$. Possible measurement outcomes divide these candidates in the way as summarised in table 10.1. For instance, as discussed earlier, if the hypothesised measurement results in $X = 4$ then M_1 is faulty (since it would otherwise lead to $X = 6$) and A_1 is not (because it is part of both environments $\{A_1, M_2\}$ and $\{A_1, A_2, M_3\}$, which support the predicted value $X = 4$). Note that, although it is possible for, say, Y to take a predicted value of 4 rather than 6, it could not have a measured value of 4 given the *present* consideration of, at least, one of A_1 and M_1 being faulty. In fact, the supporting environment of the prediction $Y = 4$ is $\{A_1, M_1\}$, an obvious contradiction (unless both single-component candidates have been eliminated). For similar reasons, the predicted value $Z = 8$ under the environment $\{A_1, A_2, M_1\}$ is excluded as a hypothesised measurement from this table.

Hypothesised Measurement	Candidates
$X = 4$	$\{M_1\}$
$X = 6$	$\{A_1\}$
$Y = 6$	$\{A_1\}, \{M_1\}$
$Z = 6$	$\{A_1\}, \{M_1\}$

Table 10.1: Fault candidates divided by possible measured values

Given this table, it is easy to count that at point X, the number of single-component candidates which predict either of X's two possible values is 1, and that at point Y or Z, the number of single-component candidates which predict Y or Z's only possible value is 2. This leads to the following:

$$\begin{array}{ll} X: & 1 log 1 + 1 log 1 = 0 \\ Y: & 2 log 2 = 1.4 \\ Z: & 2 log 2 = 1.4 \end{array}$$

where the base for the logarithm is set to *e*; a different base may be used, if preferred, as the logarithm is a monotonic function and the use of a different base will not alter the order of the results. Thus, these results show that X is the best next measuring point for candidate discrimination. This matches well with common-sense intuition.

10.4 Extending GDE with Fault Models

GDE uses only one fixed model to predict the normal behaviour of the physical system under diagnosis. It determines a component to be faulty if a retraction of its corresponding correctness assumption makes the predicted behaviour consistent with the observations.

GDE does not make use of knowledge about how components may behave when they fail. As a result, the fault candidates isolated by GDE are logically consistent with the observations, but may well be physically implausible or even wrong. A typical example for this involves a simple electrical circuit consisting of a battery and several bulbs wired in parallel with some being lit and some not. Intuitive diagnoses in such a situation are those bulbs which are not lit. However, in addition to finding these faulty bulbs, GDE may generate many other physically impossible, though logically consistent, fault candidates, unless further measurements are made. For instance, a resulting candidate, consisting of the battery and lit bulbs, might suggest that the battery is broken and provides no power (therefore, some bulbs are not lit), and those lit bulbs are faulty since they produce light without power supply. This is a rather weird explanation, of course.

Fortunately, the GDE system has been extended to exploit fault models (when available) so as to identify the actual misbehaviour of the faulty components and, thus, to limit the occurrence of implausible or incorrect diagnoses. A good example of such extensions is the GDE+ system [SD89], in which each physical component is characterised by the description of its normal behaviour plus a set of possible descriptions of fault behaviour and in which each component is assumed not to fail outside these fault descriptions.

More concretely, within GDE+, a component, C, is allowed to fail in a certain number of ways, and each possible way is explicitly described by a fault model of the behaviour of that component. Each fault model is, in turn, represented by two nodes in the ATMS: C_i and $\neg C_i$, with C_i denoting the i-th potential fault behaviour of C and $\neg C_i$ denoting the negation of C_i. At any time, there must be one and only one of C_i and $\neg C_i$ which is consistent with the observations, with the other contradicting the observations. A fault model of a component, for instance the i-th model C_i of C, is said to contradict an observation, if there is a (possibly empty) set of other components such that C_i cannot be consistently joined with any combination of the normal and fault representations of those components when presented with that observation. The introduction of fault models in GDE+ supports the following *inference rule*: If each of the known possible faults of a component contradicts the observations, then the component is not faulty and should be exonerated from suspicion.

Consider a simple circuit as shown in figure 10.2, where a battery S is connected, in parallel, to three bulbs B_1, B_2 and B_3 of the same type via pieces of wire $W_i, i = 1, 2, ..., 6$. Observations indicate that only B_3 is lit whilst B_1 and B_2 are off. The problem is given such observations to determine what went wrong in the circuit.

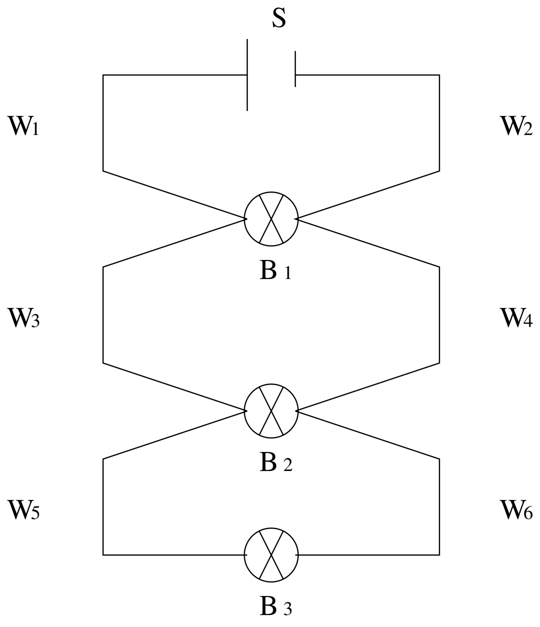


Figure 10.2: A circuit of a battery and three bulbs

Using GDE's candidate generation method, for this simple diagnostic problem, a total of 22 minimal candidates are generated, including $\{B_1, B_2\}$, $\{S, B_3\}$, $\{S, W_5\}$, $\{W_1, B_3\}$, $\{S, W_6\}$, $\{W_1, W_5\}$, $\{W_1, W_6\}$, $\{W_2, W_5\}$, $\{W_2, W_6\}$, $\{W_2, B_3\}$, ... (Note that these candidates are not necessarily generated in this order.) Although GDE does return the plausible diagnosis, i.e. the first candidate, $\{B_1, B_2\}$, in the above list, it also generates many physically implausible diagnoses. Here, the first one is plausible because B_3 being lit indicates that the battery does supply sufficient voltage, and a bulb that is off despite this fact is considered to be broken. However, all the other candidates are not acceptable diagnoses, since they imply either a wire producing voltage out of nothing or a bulb being lit without voltage. Only further measurements would enable GDE itself to rule out such implausible diagnoses.

This problem can be solved by GDE+ without requiring further measurements, however. Let each type of component have exactly one fault model, informally as listed in table 10.2, where $i = 1, 2, 3$ and $j = 1, ..., 6$. The only fault model for each bulb, and hence the one for B_3, directly contradicts

Fault	Predicted Value
battery empty	voltage(S) $= 0$
bulb broken	light(B_i) $=$ off
wire broken	resistance(W_j) $= \infty$

Table 10.2: Fault models for the battery-bulb example

the observation that shows B_3 being lit. B_3 is thus inferred to be not faulty and removed from any minimal candidates (e.g. $\{S, B_3\}$ and $\{W_1, B_3\}$) which would otherwise contain it. This is because GDE+ works based on the strict, though retractable, assumption that a component fails in no other way than those specified in its fault models. Any minimal candidate with part of it being deleted (e.g. $\{S\}$ or $\{W_1\}$) will no longer be regarded as a candidate, since whatever remain in that set of inconsistent assumptions are not sufficient to explain all the observed discrepancies (the very reason that the original candidate was referred to as a minimal candidate).

The elimination of B_3 being a possible faulty component means that there is current passing through it given the fact that it is lit. Having current at B_3 contradicts the only fault behaviour of each broken wire and the empty battery. Therefore, candidates involving pieces of broken wire and/or empty battery are also eliminated. In so doing, GDE+ is able to conclude that the only minimal candidate is $\{B_1, B_2\}$, as desired.

The price paid for this important capability of GDE+ for candidate elimination is the significant increase of the complexity of the original GDE system. GDE+ worsens the combinatorial problem of GDE: There are already $O(2^n)$ possible diagnoses for a physical system of n components without fault models in GDE, with k possible fault descriptions per component, there may be $O(k^n)$ possible diagnoses for GDE+ to examine. Therefore, there exists a potential risk in utilising GDE+ of creating a vast number of combinations of normal and fault component models. In general, certain control strategies are necessary to attain a trade-off between its discrimination power and efficiency.

Another representative of extensions made to GDE is the Sherlock system [dKW89]. Sherlock views the central task of diagnosis as identifying the behavioural description of each system component which is consistent with the observations. Whilst allowing every component to have different behavioural descriptions, normal or faulty, it maintains the basic intuition behind the GDE system: To determine faulty components without necessarily knowing how they fail. It ensures this by assigning each component with an unknown mode.

As with GDE+, however, each component now has multiple models and each model predicts a different behaviour which must be considered. There are far more behaviours to reason about and the useful concepts of minimal conflicts and minimal candidates in GDE becomes difficult to handle (if not virtually meaningless). To circumvent this problem, Sherlock resorts to probabilistic information about the likelihood of each behavioural description, introducing the most probable fault behaviours to components in order to restrict candidate generation. In particular, to keep the combinatorics in check, Sherlock performs best-first search to generate the diagnoses in decreasing likelihood and utilises heuristics to decide when enough diagnoses have been found. In this way, Sherlock avoids considering most of the possible candidates that GDE has to deal with and performs significantly better than GDE.

Chapter 11

A Brief Overview of Case-Based Reasoning

11.1 Introduction

Much of human knowledge comes from past experience. People solve new problems often by reasoning through analogy, based on their own or others' experience, to similar problems encountered before. For example, in legal reasoning and within the context of sentencing convicted criminals, it is quite common for a judge to decide what sentence should be imposed in the present case by comparing it against previous similar cases.

Similarly (though in a very different problem domain), if you would like to prepare a genuine Chinese meal for entertaining friends in Edinburgh, you would start thinking of some typical Chinese dishes. This would remind you of the recipe of a certain delicious dish. However, you might not be able to find all of the appropriate ingredients for that dish locally; you have to use some substitute ingredients and hence modify the recipe. Then, you might like to try out the new recipe. If the dish does not taste as good as expected you would need to further modify the recipe to suit the ingredients available, so long as your local friends would not feel that the dish becomes a Scottish one! You never know, the final outcome could taste better than that made using the original recipe. If successful, you would try to remember this modified recipe, comprising locally available ingredients, for the use in future social do's. It is this kind of observation that inspired the development of the techniques for case-based reasoning (CBR) [Kol93, Sla91], meaning to deal with new cases by reasoning with old ones.

This chapter presents an outline of CBR. The next section shows a generic architecture of CBR systems. This is followed by a description of two CBR programs, a legal judging supportive system [Bai91] and an architectural design assistant system [Kol91], to illustrate CBR approaches at work.

11.2 Basic Ideas of Case-based Reasoning

The reasoning process of a case-based system can be outlined as shown in figure 11.1 (after [RB87]), though there may be certain variations when implementing a particular system for a given application. Within this generic CBR system architecture, boxes represent sub-processes involved and octagons represent knowledge elements required to facilitate the reasoning. Every case-based system contains a library of past cases, called case memory. Usually, each stored case is represented by a pair of components: one being a problem description and the other being the corresponding successful solution.

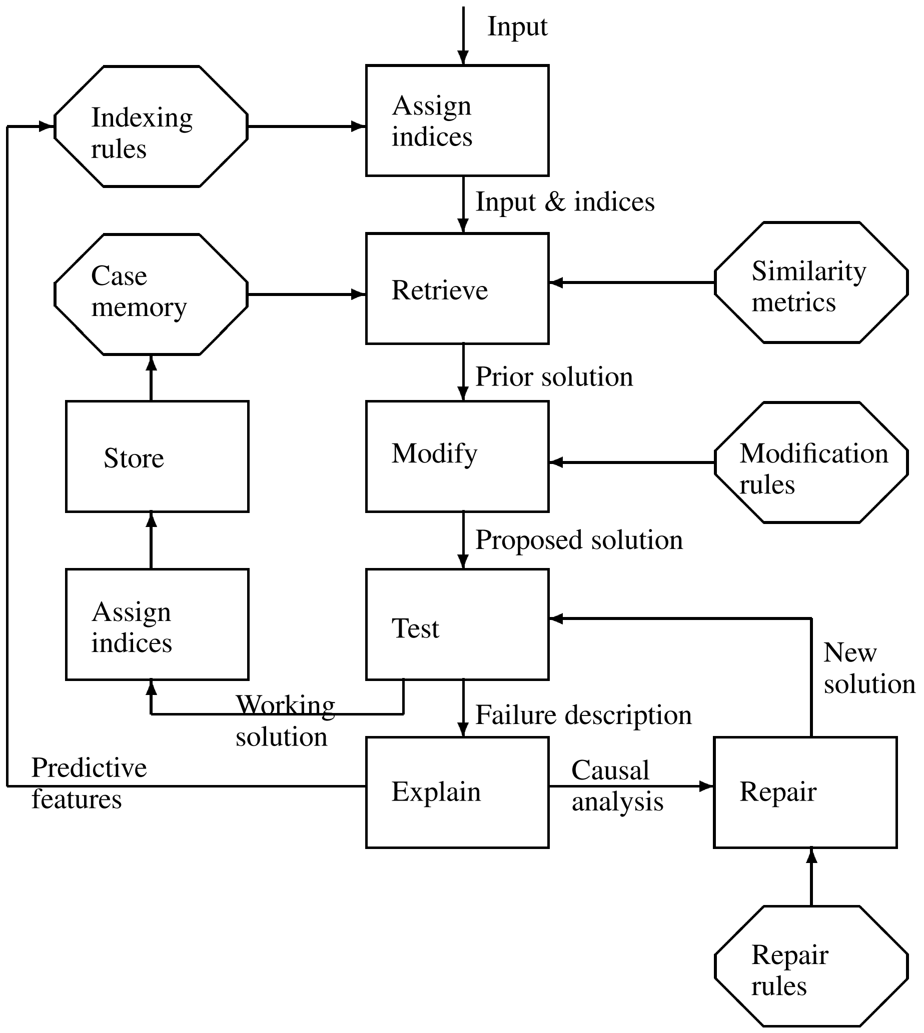


Figure 11.1: Generic architecture of case-based reasoning systems

This case memory forms the basis upon which to perform inferences. Those pairs of problem and solution may be represented in one of the many standard knowledge representation forms, e.g. using first-order predicate representation or fuzzy logic representation, and hence their actual representation is not a major concern in CBR nor in the present discussion. What is important is how a case-based system finds a solution through the use of such stored past experience.

As reflected by figure 11.1, a case-based system carries out reasoning through the following stages:

- *Assignment of Indices*: Important features of a given problem are extracted as indices characterising the problem. They are used to enable the retrieval (see next stage) of certain previously stored cases that may have similar indices assigned when they were entered into the case memory. It is hoped that the retrieved cases would have similar problem characteristics and, therefore, possible solutions to the given problem. Since indices determine under what circumstances a case might be appropriately retrieved, the success of assigning indices significantly affects a CBR system's reasoning performance. Below are three commonly used criteria for extracting useful indices:

  - Predictiveness: The indices should have an influence upon the possible solution for the given problem.
  - Abstractness: The indices should be abstract enough so that the case they associate with can match as broad a choice of problems as appropriate.
  - Concreteness: The indices should be concrete enough so that the case they associate with can be recognised without involving much inference.

  Obviously, the last two criteria compete with each other. How to determine significant input features is indeed a persistent problem in CBR (referred to as the indexing problem). Abstract features are more predictive, but concrete features are easier to compute. Thus, a trade-off between abstractness and concreteness is necessary. The following lists some indexing schemes which are often used in the literature:

  - Indexing by exceptional features
  - Using a fixed set of indices known to be predictive
  - Indexing by differences between what is already in memory and the case being indexed.

  In practice, a set of indexing rules that reflect domain heuristics are usually used to help the identification of appropriate indices for the problem at hand.

- *Retrieval of a Previous Case*: The indices assigned to the given problem are matched against the problem components of those cases stored in the memory, in order to retrieve a similar past case and hence a potentially useful solution to the present problem. The case that best matches the present problem is selected. In implementation, similarity metrics are frequently used to decide which case is more like the current one. During the matching process, parts of the solution description of the selected case may be instantiated, thereby obtaining a partial solution to the present problem.

- *Modification of Solution*: The solution component of the selected case may not have been sufficiently instantiated in the retrieval stage. In fact, few or perhaps no old cases would exactly, or even very closely, match a new situation. Therefore, adaptation is often necessary for the (partial) solution to suit the new problem. For this, knowledge about what factors in the original solution description can be changed and how to change them is needed. Such knowledge is domain-specific and is usually described in terms of modification rules that govern the way of substituting or transforming the selected solution.

- *Testing of the Modified Solution*: The modified solution is then tested to check if it would produce acceptable results for the given problem. The outcome is of course either successful or not. Depending on which outcome a modified solution leads to, one of the following two processes is executed:

  - Storing the working solution: If the solution succeeded, appropriate indices which are necessary for retrieving it in the future are assigned to the present problem and, then, the pair of the problem and the found solution is incorporated into the case memory.
  - Repairing the failed solution: If the solution failed, the system tries to form an explanation of why it failed and, based on the explanation, to repair the solution using domain-specific repair rules (showing what sorts of change are permissible). The repaired solution is then tested again, and the predictive features of the failed case may be incorporated into the indexing rules to anticipate the same failure in future.

11.3 Application Examples

Although a number of issues remain as ongoing research topics, there have been quite a few commercial as well as prototypical systems developed to try out the ideas of CBR. A categorised bibliography of these systems can be found in [MW94]. The following sketches out the basic working processes of two existing systems, in an effort to show the way of a case-based system solving given problems. An in-depth treatment of CBR in action is provided in [Wat97].

Example One

One of the early CBR programs is the Judge system, built to perform legal reasoning in the context of sentencing convicted criminals [Bai91]. The system simulates the process of a judge deciding the appropriate sentence for a criminal based on his/her experience. In particular, it uses cases as a way of establishing legal precedents, through records of past cases and their outcomes. The system is invoked when given a new case, say, CRIME999, which has features that remind it of a previous assault case, CRIME911. The Judge system compares the two cases to decide what sentence should be imposed in the new case. It explains its reasoning as it proceeds:

> CRIME999 Facts:
>
> First, Andy struck Mary with his fists several times.
>
> > Mary was only slightly hurt.
> > [Indexed as unprovoked-violation]
>
> Next, Mary struck Andy with her fists several times.
>
> > Andy was only slightly hurt.
> > [Indexed as parity-self-defence with an achieved-result]
>
> Finally, Andy struck Mary with his fists several times.
>
> > Mary received a concussion.
> > [Indexed as parity-retaliation with a violative-accidental-result]
>
> The only thematically similar previous situation in memory was CRIME911. CRIME911 contained similar intention and result indices as CRIME999.
>
> Comparing CRIME911 from the memory with the current input, CRIME999 gives:
>
> The old crime, CRIME911, involved the following:
>
> First, Mark struck Fiona with his fists several times.
>
> > Fiona was only slightly hurt.
> > [Indexed as unprovoked-violation]
>
> Next, Fiona struck Mark with her fists several times.

```
        Mark was only slightly hurt.
        [Indexed as parity-self-defence with an achieved-result]
Then, Mark slashed at Fiona with a knife one time.
        Fiona's skin was slightly cut.
        [Indexed as escalated-retaliation with a bungled-result]
Next, Fiona slashed at Mark with a knife one time.
        Mark's skin was slightly cut.
        [Indexed as parity-self-defence with a bungled-result]
Finally, Mark stabbed Fiona with a knife several times.
        Fiona died.
        [Indexed as escalated-retaliation with an achieved-result]

The result where Fiona was killed was worse than the one in which Mary received a
concussion. Not only was the outcome in the newer situation accidental, but Mark both
intended and caused more harm than Andy.

The intent of Mark, the offender in the old crime, to act repeatedly to stab Fiona to death
was more severe than that of Andy, the present offender, to act repeatedly to hit Mary.
In addition, although neither actor's intentions were justified, Mark escalated the level of
violence, while Andy did not.

Comparison finished with the result that the old crime, CRIME911, was worse.

The sentence originally given for CRIME911, for violation of murder, was a term of
imprisonment of not less than 40 years nor more than 50 years to be served in the state's
prison. Since the circumstances of CRIME911 are found to be worse than CRIME999,
due to more severe actions and results at the end of the crime, the present offender will
get a sentence which is more lenient than the sentence for CRIME911, relative to the
crime of assault in the second degree.

The sentence to be given for CRIME999, for violation of assault in the second degree
as defined in Section 53A-60 in the Connecticut penal code, will be a term of imprison-
ment of not less than 3 years to be served in the state's prison. The maximum period of
incarceration allowed for violation of this statute is 5 years.
```

The above execution result of the Judge system with respect to a new crime, CRIME999, illustrates some of the key stages of CBR. First of all, the system has a case memory consisting of previously adjudicated crimes to start with. Then, the newly presented case is analysed to provide indices in order to retrieve similar past cases. Next, it compares the new case with the retrieved (single) old case to determine how the new crime might require modification of the sentence given to the old case. After this, the system provides an explanation of the reasoning done by itself through explicit analysis and comparison of the two cases. Finally, the new case can be incorporated into the case memory with the sentence period meted out.

Example Two

The Judge system uses the CBR approach to solve problems without interactions with a user, once a new case has been presented to it. CBR can also be utilised to develop systems for providing assistance to human user's decision making via interactions during an execution process. This kind of application is motivated by the observation that people are good at using analogy to solve new problems but they are not always good at remembering right past cases, whilst computers are good at remembering. The basic idea behind this is that the CBR system arguments the user's memory by providing cases for him/her to use in solving a problem, and the user makes actual decision using these cases as guidelines. The following example (modified from [Kol91]) shows such an application system, which helps an architect in completing a design task.

The problem considered is to design a geriatric hospital on a site of x-acre wooded sloping square. The hospital should: a) be able to serve a certain number of inpatients and outpatients daily and provide offices for a group of doctors, b) include both long-term and short-term facilities, and c) be more like a home than an institution and allow easy visitation by patients' family members.

To initiate the system, the following features that the user feels need to be addressed early in the design are entered into the system (as indices): hospital, homeyness, provision of visitation, long-term and short-term care, and geriatric; together with the physical constraints of the site, namely its contour, size and shape. The system then uses these features as preferences in finding useful cases in the memory. The most useful cases will match all these features; less useful ones will match some subset of them (and will, therefore, probably address only some part of the problem). The system finds many matched or partially matched cases and partitions these cases according to how well they match those features. From this, it presents an ordered partitions and asks the user which case should be shown in more detail:

> I know of many long-term facilities for the elderly, many acute-care hospitals, and one geriatric hospital: Wesley Woods in Atlanta. Of the acute-care hospitals, Piedmont Hospital in Atlanta is the most homey. Of the long-term facilities, the Jewish Home in Fairfield County, Connecticut, is the closest in the type of care it provides and in its ability to provide opportunities for family visitation. Which would you like to see?

The user chooses to see Wesley Woods, the only geriatric hospital with both long-term and short-term facilities, by entering: "Show Wesley Woods". The system shows a picture of the layout of Wesley Woods and provides a commentary:

> The goal in designing Wesley Woods was to create a homey atmosphere. To achieve this goal, they used residential, rather than institutional, furnishings and forms with cheery colours, etc. They made all the corridors short so that patients would feel they were in a home-sized place. The design did a very good job of providing homeyness. However, there was one big problem: it is hard for patients and staff to find their way around because the views give no distinguishing landmarks or orienting features.

Having noticed both the successes and the failures of Wesley Woods, the user decides to borrow large portions of this design and starts manipulating the design to make it fit the new site while avoiding its pitfalls. For the new hospital to fit the site, the user adapts the original one-floor design by adding a lower floor and putting one main entrance on each level. These two new features, i.e. two floors and each having an entrance, are then entered into the CBR system. The system accordingly retrieves cases that show how to use two entrances on two levels and interrupts the user by making the suggestion below:

> I have several examples of hospital buildings with entrances on two levels. Georgie Baptist Hospital has lobbies on two levels, one for inpatients, one for outpatients. Brighams and Womens in Boston has entry doors on two levels with a two-floor lobby that combines inpatient and outpatient functions. The two floors within the lobby are connected by a ceremonial escalator.

The user at this stage requires that the outpatient lobby be put on the bottom level and the inpatient one on the second level. In response to this new requirement, the CBR system tries to warn a potential violation of the initial design goal that the user entered before, by presenting the following to the user:

> One of the nice features of the Jewish Home in Fairfield is the access to the outside that patients have. That is a potential problem if the inpatient lobby is on the upper floor.

Interestingly, this system is able to detect possible violations, because it keeps track of the design goals *and* how some of these goals were achieved in the cases matched or partially matched (which the user might not have seen yet). In this case, it noticed that the inpatient entrance was required to be above the main usable ground. This would lead to the inconvenience for the inpatients and their visitors to reach outside, whilst the Jewish Home which allows for easy visitation provides easy access to the usable outdoors. The CBR system could go beyond this warning and suggest an alternative solution, and then the interaction process would carry on until the user is satisfied with the final outcome.

In short, this case-based architect's assistant is used to suggest possible solutions to a design problem and provide warnings of potential failures. It does these mainly by retrieving previous cases which best match the features presented by the user, and by tracking which cases within the matched ones possess which required features. The human user is, however, responsible for hard adaptation of the suggested solutions, for the choice of the features to consider, for the evaluation of the suggestions and warnings and, ultimately, for the final decision making.

Bibliography

[Bai91] W. Bain. *Case-Based Reasoning: A Computer Model of Subjective Assessment*. Ph.D. Dissertation, Yale University, 1991.

[BS84] E. Buchanan and E. H. Shortliffe. *Rule-based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[Dav87] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[Dem67] A. Dempster. Upper and lower probabilities induced by multivalued mappings. *Annals of Math. State.*, 38:325–329, 1967.

[dK86] J. de Kleer. An assumption-based tms. *Arificial Intelligence*, 28:127–162, 1986.

[dK90] J. de Kleer. Using crude probability estimates to guide diagnosis. *Arificial Intelligence*, 45:381–392, 1990.

[dKW89] J. de Kleer and B. Williams. Diagnosis with behavioural modes. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 2, pages 1324–1330, 1989.

[Doy79] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[dW87] J. deKleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–129, 1987.

[Gal86] J. Gallier. *Logic for Computer Science*. Harper Row, 1986.

[GLF81] T. Garvey, J. Lowrance, and M. Fischler. An inference technique for integrating knowledge from disparate sources. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981.

[GN87] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[GS85] J. Gorden and E. Shortliffe. The Dempster-Shafer theory of evidence. In B. Buchanan and E. Shortliffe, editors, *Rule-based Expert Systems*, pages 272–292. Addison-Wesley, 1985.

[Kol91] J. Kolodner. Improving human decision making through case-based decision aiding. *AI Magazine*, pages 52– 68, 1991.

[Kol93] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[Kui86] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[Kui94] B. Kuipers. *Qualitative Reasoning: Modelling and Simulation with Incomplete Knowledge*. MIT Press, 1994.

[Lee90] C. Lee. Fuzzy logic in control systems: Fuzzy logic controller - parts i and ii. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.

[Mac92] A. Mackworth. *Encyclopaedia of AI*. Wiley, 1992.

[MS99] I. Miguel and Q. Shen. Hard, flexible and dynamic constraint satisfaction. *Knowledge Engineering Review*, 14(3):199–220, 1999.

[MW94] F. Marir and I. Watson. Case-based reasoning: a categorised bibliography. *Knowledge Engineering Review*, 9, 1994.

[Nov96] V. Novak. Paradigm, formal properties and limits of fuzzy logic. *General Systems*, 24:337–405, 1996.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[PG98] W. Pedrycz and F. Gomide. *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, 1998.

[RB87] C. Riesbeck and W. Bain. *A Methodology for Implementing Case-Based Reasoning Systems*. Lockheed, 1987.

[Run97] T. Runkler. Selection of appropriate defuzzification methods using application specific properties. *IEEE Transactions on Fuzzy Systems*, February, 1997.

[Sav54] L. J. Savage. *The Foundations of Statistics*. Wiley, 1954.

[SD89] P. Struss and O. Dressler. Physical negation - integrating fault models into general diagnostic engine. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 2, pages 1318–1323, 1989.

[Sha76] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[Sha94] L. Shastri, editor. *Fuzzy Logic Symposium*, IEEE Expert, 1994. Thematic issue, 9(4).

[Sla91] S. Slade. Case-based reasoning: a research paradigm. *AI Magazine*, pages 42–55, 1991.

[Smi85] B.C. Smith. Prologue to 'Reflection and semantics in a procedural language'. In R.J. Brachman and H.J. Levesque, editors, *Readings in Knowledge Representation*, pages 31–40. Morgan Kaufmann, 1985.

[Wal75] D. Waltz. Understanding line drawings of scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.

[Wal96] P. Walley. Measures of uncertainty in expert systems. *Artificial Intelligence*, 83:1–58, 1996.

[Wat97] I. Watson. *Applying Case-Based Reasoning: techniques for enterprise systems*. Morgan Kaufmann, 1997.

[WHdK92] L. Console W. Hamscher and J. de Kleer. *Readings in Model-based Diagnosis*. Morgan Kaufmann, 1992.

[Zad65] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad86] L. Zadeh. Is probability theory sufficient for dealing with uncertainty in AI: A negative view. In *Uncertainty in Artificial Intelligence*. Elsevier, 1986.

[Zad 6] L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, pages (8):199–249/301–357, (9):43–80, 1975-6.