Knowledge Representation and Inference Two

Lecture Three:

Non-Monotonic Reasoning and Truth Maintenance Systems

1 Introduction

This set of notes leads on from the non-classical logics discussed in lecture two and presents a discussion of some aspects of non-monotonic reasoning. First, a development of first-order predicate logic called circumscription is presented. This was developed to meet some of the inadequacies of the *situational logic* presented in lecture two, and thus represents an attempt to formalise non-monotonic reasoning. The second part of the notes presents three kinds of *Truth Maintenance* techniques, which are used to support the workings of a problem-solver which reasons non-monotonically. A bibliography is included at the end for further reading on the material covered, together with this week's reading (which includes *no* required reading), and the next small group tutorial preparation.

2 Monotonic Reasoning

Classical propositional logic is an example of a monotonic logic. If S is the set of formulas provable from a set of axioms A, then if the set A is enlarged by the addition of one or more axioms, the size of the set of provable formulas S increases (or possibly stays the same size). There are no axioms which can be added to A which will result in the number of provable formulas in S being reduced. The set S is thus said to monotonically increase as the number of axioms in A increases. In other words, the addition of new knowledge (in the form of axioms in A) will never cause a previously derived fact (in the form of formulas in S) to become invalid, or untrue.

There are several advantages to using monotonic reasoning systems based upon one of the classical logics. These include:

1. When a new fact is added to the system, no checks need to be carried out to ensure that it is consistent with those already known to the system.
2. It is not necessary to record for each derived fact introduced the facts upon which its truth fundamentally depends, since there is no danger of previously added facts being removed.

Monotonic reasoning is, however, not able to deal with three kinds of situations which often arise in the real world. These are:

[illegible] 26/1/88.

disadvantages.

i) Situations in which only incomplete knowledge is available, or in which knowledge about the situation is discovered as a result of carrying out the actions that have to be reasoned about. sequentially.

ii) Situations in which the knowledge being reasoned about changes due to changes in the world.

iii) Situations in which the reasoning system generates *assumptions* as part of its problem-solving strategy — generate and test, for example.

3 Non-Monotonicity and Non-Monotonic Reasoning

Consider the following example from the records of an agents reasoning:

1)	**Given fact:**	Robin lives in the United States.
2)	**Already known:**	Boston is in the United States.
3)	**Infer:**	From (1) and (2) it is possible that Robin lives in Boston.
4)	**Given fact:**	Robin lives in Amherst.
5)	**Already known:**	Amherst is not a part of Boston.
6)	**Infer:**	It is not possible that Robin lives in Boston.

In this example inference (6) can be seen to be the negation of inference (3). Thus the reasoning processing illustrated involves the reversal in the truth value of a possibility-qualified proposition. This type of reasoning is called *non-monotonic reasoning*, and is to be contrasted with the strictly *monotonic reasoning* supported by classical first order logics.

3.1 Default Reasoning and Circumscription

3.1.1 Default Reasoning

One type of non-monotonic reasoning is called *default reasoning*. It occurs in situations in which a fact, or belief, is derived from a given fact because it usually follows and because there is no evidence to believe it does not in the particular case. However, if, at some later stage, evidence for the negation of this derived belief is presented the derived *default* fact is withdrawn. Such beliefs are said to be *defeated* in such situations, and a belief (sometimes called a conclusion) which may later be defeated is called *defeasible*. For example:

- At an important public lecture Robin (the invited speaker) is introduced as a Professor of Computing Science. You therefore concluded that Robin has a

PhD, which need not necessarily be the case. Some way through the Chairman's introduction of Robin, while he is summarising his career, you realise that Robin never actually obtained a PhD. You therefore withdraw your previous conclusion that he has one. [1]

This pattern of reasoning can be generalised as follows:

If	X is a professor of Computing Science
and	there is no evidence that X does not have a PhD
then	conclude X has a PhD.

The conclusion of this rule is a *default.*

The novel part of this rule is the *there is no evidence that X does not have a PhD* part. Formalising what this means requires more subtlety than it might seem. One important attempt at doing so is due to McCarthy and is called *Circumscription.* Building computer programs to reason in a non-monotonic way is not so hard though. How this can be done is the subject of the second part of these notes on *Truth Maintenance Systems.*

3.1.2 Circumscription

In reasoning about some problem we often assume that the problem involves only those objects and relationships that are of immediate relevance to the particular problem, and that no others are involved. For example, in the well known *missionaries-and-cannibals* problem of how to get the missionaries across the river without being eaten by the cannibals, we do not usually think of solutions involving bridges, rockets, handcuffs, murder of the cannibals, or holes in the boat, etc.

Circumscription is a formal method for representing default reasoning using predicate calculus. It uses a *mechanism* for adding to a set of predicates one or more new formulas which express a default, or closed world, assumption. In other words, it formalises a mechanism in which it is assumed that all qualifications to the solution to a problem are stated explicitly.

Let a be a formula containing an n-ary predicate symbol p. If Ψ represents a formula with n designated free variables, then we use $a[\Psi/p]$ to denote the result of substituting Ψ for each occurrence of p in a, such that the *k*th free variable in Ψ is replaced by the *k*th argument of p in the occurrence. If the occurrence of p has the form $p(X_1, ..., X_n)$, which is abbreviated to $p(\mathcal{X})$, then the occurrence of Ψ that replaces it can be denoted $\Psi(\mathcal{X})$.

Then the circumscription of p in a is the following *schema* for generating formulas :

$$\{a[\Psi/p] \wedge \forall \mathcal{X}[\Psi(\mathcal{X}) \rightarrow p(\mathcal{X})]\} \rightarrow \forall \mathcal{X}[p(\mathcal{X}) \rightarrow \Psi(\mathcal{X})].$$

This schema represents the assertion that the only objects $\mathcal{X}$ that satisfy p are those which *must* satisfy p, to avoid inconsistency, assuming a is true.

[1] The reason why Robin does not have a PhD is now one of the legends of the AI Department, and beyond belief, were it not for the fact it is true, and only to be related late at night after a suitable intake of some alcoholic beverage at some favourite hostelry.

To illustrate how circumscription can be used consider the following scenario:

> Anne has been invited by Simon to dinner. On arriving at Simon's flat Anne discovers that he has forgotten to buy any wine. Simon asks Anne to go round to the off-licence to buy some, adding that she may use his car, and hands her the keys saying the car is just outside. Anne agrees to go, concluding that she would have to pay for the wine. After failing to start Simon's car and concluding that it has a flat battery, Anne walks round to the off-licence to buy the wine, and on the way back decides that Simon will pay for it.

In order to drive a car there are a number of requirements, two of which are:

k: having the *keys* to the car, and

c: being able to get to the car. ¬ a

These requirements or prerequisites for driving a car are expressed by the formula:

a: $\text{prerequisite}(k) \wedge \text{prerequisite}(c)$.

In the above scenario Anne assumed that if the prerequisites were satisfied (that is that there were *no problems* concerning these aspects of driving a car) she could and would drive the car round to the off-licence. This belief is expressed as follows:

α: $\forall X[\text{prerequisite}(X) \rightarrow \text{noproblems}(X)] \rightarrow \text{buy}(\text{anne}, \text{wine})$.

When Anne went to drive Simon's car she had two prerequisites:

$$\text{noproblem}(k) \wedge \text{noproblem}(c).$$

The circumscription of *prerequisite* in a is the schema:

$$\{[\Psi(k) \wedge \Psi(c)] \wedge \forall X[\Psi(X) \rightarrow \text{prerequisite}(X)]\} \rightarrow \forall X[\text{prerequisite}(X) \rightarrow \Psi(X)].$$

From this circumscription we can derive Anne's first conclusion. We begin by taking for Ψ the expression $(X = k) \vee (X = c)$.

The antecedent of the circumscription is true, so we may conclude the right-hand side:

$$\forall X[\text{prerequisite}(X) \rightarrow ((X = k) \vee (X = c))]$$

which says that the keys and access to the car are the only prerequisites. This formula is then added to the set of theorems, and it can be used to deduce new theorems.

Now since k and c are the only prerequisites, and for Anne both of them are satisfied, the antecedent of (α) is true, thus the consequence that buy(anne, wine) follows. Note that without circumscription, there would be no basis for proving the antecedent of (α).

The non-monotonic negation of Anne's initial conclusion that she would pay for the wine occurred after two further facts became known:

b: battery functional

prerequisite(b):	the battery of the car must not be flat, and
¬noproblem(b):	the battery of the car is not functional.

A way of dealing with Anne's change of mind is thus to remove from the set of theorems the right-hand side of the circumscription of *prerequisite* in a as soon as the new facts involving *prerequisite* become known, and to also remove all the formulas derived from the circumscription. The new fact may be conjoined with a to form a new formula:

a′: noproblem(k) ∧ noproblem(c) ∧ noproblem(b).

A new circumscription of *prerequisite* in a′ may be constructed in the hope of deriving new useful conclusions, but it is no longer possible to prove the formula buy(anne, wine), since ¬noproblem(b) prevents the antecedent of (α) being true.

3.2 Some Comments

An advantage of circumscription, over some other methods for dealing with non-monotonic reasoning, is that it is an extension of first-order predicate logic, which allows all the reasoning, except circumscription itself and the instantiation of the resulting schemata, to be handled by methods already available, such as resolution, for example. However, the practical application of circumscription is awkward in comparison with the adoption of specific defaults or the use of negation-as-failure assumptions in logic programming.

Since 1980, when McCarthy first introduced circumscription as a formal way of dealing with the problems of using *situational* logic — the *frame* problem and the *qualification* problem, it has developed into almost a subject in its own right. There are now different types of circumscription, starting with the *predicate circumscription* presented above, there followed *domain circumscription* which was an attempt to improve the generality of predicate circumscription. *Formula circumscription* came next as another improvement, and more recently there has been something called *prioritized circumscription*, which is intended to be more general yet. All this work on circumscription, mostly reflects one of McCarthy's firm and long held beliefs, that trying to develop a formal language for expressing general commonsense knowledge for inclusion in a general database is the key problem of generality in AI.

The use of non-monotonic reasoning techniques is not limited to those classes of problems which specifically require this type of reasoning. Non-monotonic reasoning may also be the only way of building a practical system, ie one which produces results in a reasonable amount of computational time, for example, to solve types of problems which are theoretically monotonic but which when expressed in first-order predicate logic are undecidable in finite time. Playing chess, for example, can be formulated as a monotonic reasoning problem, but in practice this is of little value, and chess playing programs usually have to choose a move after a certain time interval; increasing this interval for consideration might change its decision as to what move to make next, thus introducing a degree of non-monotonicity in its behaviour. Such systems can be thought of as using non-monotonic reasoning to approximate the *theoretically possible* monotonic reasoning.

"Reasoning and Problem Solving programs must eventually allow the full use of quantifiers and sets, and have strong enough control methods to use them without combinatorial explosion" — McCarthy.

4 Truth Maintenance

4.1 Introduction

When problem-solving systems which reasoned non-monotonically were first built they could be viewed as programs which performed a mixture of *house-keeping* and deduction, with much of the house-keeping serving to decide what deductions to make. This typically resulted in bits and pieces of deductions being strewn throughout the program execution (as Drew McDermott appropriately put it). This somewhat ad hoc state of affairs continued until *Truth Maintenance* systems were invented, first by Doyle and McAllester, and more recently by de Kleer, to collect and maintain the bits of deductions produced and used by a problem-solving system.

Truth maintenance systems record and maintain proofs. The proofs are made up of *justifications* connecting data structures called *nodes*. Nodes typically represent assertions, rules, or other types of system beliefs. They may be the consequence of several justifications, each of which represents a different method of proving belief in the node. Also each justification may have several nodes as its antecedents. Some nodes may be designated hypothesis, or assumptions. For each node, the truth maintenance systems computes whether or not belief in the node is supported by a non-circular proof from the basic hypotheses and the set of recorded justifications. The set of such non-circular proofs is recorded as the *well-founded support* of the believed nodes. In other words, a truth maintenance system is used to maintain a proof structure about the beliefs of a problem-solving system over its database of facts, or rules, etc. It is therefore not a reasoning, or problem-solving, system in itself but a reason support system, or belief support system; they are used to provide an important service to problem-solving systems. Truth Maintenance systems are therefore sometimes called *Reason Maintenance Systems* or *Belief Revision Systems*. The term Truth Maintenance System (TMS) will be used in these notes and is taken to be synonymous with the other two terms.

4.2 Chronological Backtracking

The simplest way of providing truth maintenance to a problem-solver is like that provided by the prolog language. This is straightforward tree searching with backtracking. In such a system all hypotheses and the consequences inferred from them are recorded at the search node that created them. When an inconsistency is generated, the system simply backtracks to the next node from which there remain unexplored search paths. The hypotheses and their consequences disappear automatically. This mechanism of maintaining a consistent database of hypotheses and consequences makes no attempt to record node dependencies. It consequently operates by removing facts in the order in which they were generated by the problem-solver, the prolog theorem prover for example.

For example, suppose our problem-solver were to generate the following sequence of hypotheses (Hi) and consequences (Ci) as a result of some reasoning:

H1: The next King's Cross to Waverley train leaves at 14:00 hours.

– "In my opinion getting a language for expressing general commonsense knowledge for inclusion in a general database is the key problem of generality in AI" – McCarthy.

C1: The next train from King's Cross will arrive at Waverley at 19:00 hours.

H2: The walk from Waverley station to Royal Terrace will take 20 minutes.

C2: The time of arrival at Royal Terrace on catching a train from King's Cross to Waverley will be 20 minutes after arriving at Waverley Station.

And then, on receiving news that the departure of the 14:00 hours King's Cross to Waverley train has been delayed by 30 minutes, introduces a new hypothesis and consequence:

H3: The next King's Cross to Waverley train leaves at 14:30 hours.

C3: The next train from King's Cross will arrive at Waverley at 19:30 hours.

which contradicts consequence C1, based upon hypothesis H1. The backtracking mechanism of our simple truth maintenance system will remove all database entries made subsequent to and including H1, leaving the new hypothesis and consequence about when the train from King's Cross will depart and arrive. The database of the problem-solving system will thus becomes:

H3: The next King's Cross to Waverley train leaves at 14:30 hours.

C3: The next train from King's Cross will arrive at Waverley at 19:30 hours.

and all the reasoning about how long it takes to walk from Waverley Station to Royal Terrace will be lost, despite the fact that it is still valid, and could be used to infer the new time of arrival at Royal Terrace having travelled from King's Cross on the delayed King's Cross to Waverley train.

4.2.1 Some Comments

This kind of truth maintenance is usually referred to as *Chronological Backtracking.* Although it is the simplest to implement, it comes free with prolog, and the cheapest in terms of storage space required, it is typically the most expensive in terms of computation. For anything but very simple problems it usually proves to be unacceptably expensive, and is thus rarely used in practice. Because of this, and because it does not attempt to keep any records of the dependencies between the nodes in the database, it is not usually called a Truth Maintenance System, but just Chronological Backtracking.

4.3 Dependency-Directed Backtracking and the Doyle Type TMS

fewer rules required

In 1979 Doyle proposed a truth maintenance system which made use of a more efficient method of modifying the contents of a database called *dependency-directed backtracking*, first proposed by Stallman in 1977. Using dependency-directed backtracking enables the hypothesis nodes in the database, and any nodes that have been inferred from them, to be added and removed in any order. Thus, for example, if the reasoning system which produced the sequence of hypotheses and consequences described in the previous section had been supported by a dependency-directed backtracking system the database, after the entry of hypothesis H3 and its consequence C3, would have become:

H2: The walk from Waverley station to Royal Terrace will take 20 minutes.

C2: The time of arrival at Royal Terrace on catching a train from King's Cross to Waverley will be 20 minutes after arriving at Waverley station.

H3: The next King's Cross to Waverley train leaves at 14:30 hours.

C3: The next train from King's Cross will arrive at Waverley at 19:30 hours.

Thus saving it from having to re-make hypothesis H2, and infer the consequence C2.

In the Doyle Truth Maintenance System, a node justification may be either *valid* or *invalid*, usually invalid justifications start off being valid, but become invalid as a result of some action of the problem-solving system. Any node which has at least one valid justification is said to be IN, that is, *in the set of believed nodes.* Nodes with no valid justifications are said to be **OUT**, that is, *not in the set of believed nodes.* Note that an **OUT** node may have a use in the reasoning system, it may, for example, be mentioned in the valid justification of some IN node which is believed because the **OUT** node is not believed.

In the Doyle type TMS there are two kinds of justification:

- *Support List* justifications, of the form:

  believe this node as long as believe Inlist nodes but don't believe Outlist nodes.

  (SL INlist OUTlist)

  where INlist and **OUT**list are each lists of nodes. The justification is valid if all the nodes in the INlist are IN, and all the nodes in the **OUT**list 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 the nodes in OUTlist are out. This can be thought of as being (CP *consequent IN-hypotheses OUT-hypotheses*), or the *consequent* node has a valid justification when all the nodes in the *IN-hypotheses* list are in and all the nodes in the *OUT-hypotheses* list are out.

The above type of justifications enable some useful kinds of nodes to be built in the following ways:

- A fact or rule can be made an axiom, ie something to always be believed, by giving the justification of the node an INlist and **OUT**list which are both empty so that it is always IN.

- Nodes that have a non-empty **OUT**list are hypotheses or assumptions; there are conditions under which they could be contradicted.

- CP nodes can be used to support deduced rules, for example:

Node	*Fact or Rule*	*Justification*
n1	A	possibly none valid
n2	B	possibly none valid
n3	if A then B	(CP **n2** (**n1**) ())
n4	C	possible none valid
n5	if C and A then B	(CP **n2** (**n3** **n4**) ())

The algorithm used by a Doyle type TMS to move nodes **IN** and **OUT** as the validity of their justifications change typically works as follows. Assume that **N** is some node whose contents are discovered by the reasoning of the problem solver to be inconsistent with some other node, and that it has decided that **N** is the cause of the trouble. The TMS then proceeds by:

1. Tracing the dependencies backward from **N**, which is currently **IN**, to find the set of hypotheses that support it. Find the maximal set, ie those that do not support other hypotheses, denoted $\mathbf{H}_1$ ... $\mathbf{H}_n$.
2. Creates a new node called *no-good*, or **NG** to represent the occurrence of the contradiction, by defining the rule:

   not($\mathbf{H}_1$ and ... and $\mathbf{H}_n$)

   giving it the justification:

   (CP **N** ($\mathbf{H}_1$ ... $\mathbf{H}_n$) ())

   which means that it is currently **IN**.
3. Pick, possibly at random, one of the set $\mathbf{H}_1$ ... $\mathbf{H}_n$, denoted by $\mathbf{H}_i$, so that if it were forced **OUT**, **N** would also be forced **OUT**.
4. Since $\mathbf{H}_i$ is a hypothesis it has a non-empty **OUT**list in a valid justification. Find the set of nodes $\mathbf{D}_1$ ... $\mathbf{D}_m$ which are currently **OUT**, such that if any one came **IN** then $\mathbf{H}_i$ would go **OUT**. Pick one, denoted $\mathbf{D}_j$.
5. Give $\mathbf{D}_j$ a new valid justification so that it comes **IN**. A suitable justification would be:

   (SL (NG $\mathbf{H}_1$ ... $\mathbf{H}_i - 1$ $\mathbf{H}_i + 1$ ... $\mathbf{H}_n$) ($\mathbf{D}_1$ ... $\mathbf{D}_j - 1$ $\mathbf{D}_j + 1$ ... $\mathbf{D}_m$))

   $\mathbf{D}_j$ will become **IN** so $\mathbf{H}_i$ will become **OUT**, and **N** become **OUT**. If any other **D** comes **IN**, or if any other **H** goes **OUT** then $\mathbf{D}_j$ will be **OUT**. This may bring $\mathbf{H}_i$ back **IN** without bringing **N** back **IN**, so the arbitrary choice of $\mathbf{H}_i$ is not irrevocable. Also, the **NG** node records the nature of the contradiction so that the reasoning system will not be troubled by this particular inconsistency again, at least not in that particular form.

To illustrate this procedure, suppose, for example, that we have a reasoning system trying to solve a timetabling problem, and in the process assumes that the KR+I-2 lectures will be held at 2pm in Appleton Tower (AT). The database might thus look like this:

Node	*Content*	*Justification*
n1	time = 2pm	(SL () (**n2**))
n2	time ≠ 2pm	...
n3	place = AT	(SL () (**n4**))
n4	place = FH	...

and that at this stage the **IN** set is {**n1 n3**}. Now suppose that the reasoning system discovers a problem, such as:

n5	the problem	(SL (**n1 n3**) ())

that is, some node **n5** is inferred, based upon time = 2pm and place = AT, which turns out not to be possible. The **IN** set is now {**n1 n3 n5**}. To clear the problem, introduce the *no-good* node:

n6	not(**n1** and **n3**)	(CP **n5** (**n1 n3**) ())

The maximal hypotheses are **n1** and **n3**. Pick **n3**. It has one **OUT** node, **n4**, which can be brought **IN** to take it out. Thus, give **n4** the justification:

n4	place = FH	(SL (**n6 n1**) ())

The **IN** set becomes {**n1 n4 n6**} and the problem is cured. The existence of **n5** and **n6** prevent the TMS allowing the problem-solver creating the same problem for itself in future. (The problem-solver then goes on to discover that FH does not have a room large enough to hold the KR+I-2 class!)

4.3.1 Some Comments

In a more sophisticated system, the problem-solver may be used to help the TMS make a more rational choice about which node is causing the problem, rather than just picking one at random. This algorithm is not without its problems. For example, suppose that the TMS tries to force **OUT** some newly-discovered contradiction C, and the cause of the contradiction is a hypothesis ¬C, which is currently **IN**. In this case the TMS may get into an infinite loop. Although it is not difficult to patch up the system to deal with this kind of failure, there are more complex versions of it which are harder to recognise and deal with.

This kind of TMS can become very inefficient if it has to move large numbers of nodes **IN** and **OUT** all the time. Under such circumstances the problem-solving support system, the TMS, takes up more computational power than the problem-solver itself. It's a bit like a virtual memory paging system continually swapping memory segments between main memory and disc while the application program hardly crunches anything. However, for problems whose search spaces are such that this kind of behaviour is not easily provoked by a problem solver, and for classes of problems for which only one or a relatively few solutions are required, this type of TMS can be effective.

4.4 De Kleer's Assumption-Based Truth Maintenance — MANY SOLUTIONS REQUIRED.

In de Kleer's TMS, justifications are always considered to be valid, if they are made from rules and facts which hold true for some set of *assumptions*. The task of his Assumption-based TMS (ATMS) is to record under what sets of assumptions any conclusion holds. Each node has, in addition to its contents, indicating what it denotes, a *label* which contains a set of sets of assumptions. Each of the sets of assumptions in the set denoted by a nodes label can be used to *prove* the contents of the node. If a node turns out to be false, according to the reasoning of the supported problem-solver, the ATMS notes all the sets of assumptions labelling the node as *no-good* sets. The convenient way of doing this noting is to have a node whose content is *false*, and to add any sets of no-good assumptions to its label. A check must then be made to see that none of the sets in the label of the false node is a superset of any other; if there are any they should be deleted. For this to work properly the system must also take care that no superfluous assumptions appears anywhere in a set in a label. In other words, that only assumptions directly involved in the proof of a nodes contents are included in the label sets. Whenever the label of the false-node is changed, due to a new no-good set of assumptions having been discovered, the new false-node label entry, or any superset of it, must be removed from any other node label containing it.

For example, suppose that at some stage in the reasoning of our problem-solver we have nodes for five assumptions $a_1 \ldots a_5$, and three nodes for some derived facts A, B, and C with the labels shown below.

Node	*Contents*	*Label*
n1	assumption 1	$\{a_1\}$
n2	assumption 2	$\{a_2\}$
n3	assumption 3	$\{a_3\}$
n4	assumption 4	$\{a_4\}$
n5	assumption 5	$\{a_5\}$
n6	A	$\{a_1\ a_2\}\ \{a_2\ a_5\}$
n7	B	$\{a_1\}\ \{a_2\ a_3\}\ \{a_4\}$
n8	C	$\{a_1\ a_3\ a_4\}$

so 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 found so far is that a_4 and a_5 can be used to prove false. So that we have a *false* node entry, typically numbered node zero, looking like this:

n0	false	$\{a_4\ a_5\}$

It is important to remember that the ATMS does not use a concept of believed nodes and unbelieved nodes in the Doyle sense; there is no question about nodes being IN or OUT. Instead, it records only how nodes, whose contents are inferences from other nodes, would depend upon any assumptions were they to be believed. In other words, the ATMS only seeks to record all possible proof paths from consistent sets of assumptions to the contents of each node in the database.

Continuing with our example, suppose our problem-solver wants to add another justification to the database, that A and B imply C. The ATMS records the justification,

and from the labels of A and B creates a new label for C as follows (note if C were a new deduction it would first have created a new node for it):

i) Form the cross-product of the labels of A and B from the unions of all the pairs of sets; these are:

1. $\{a_1\ a_2\}$
2. $\{a_1\ a_2\ a_3\}$ SUPERFLUOUS
3. $\{a_1\ a_2\ a_4\}$
4. $\{a_1\ a_2\ a_5\}$
5. $\{a_2\ a_3\ a_5\}$
6. $\{a_2\ a_4\ a_5\}$ — CONTRADICTORY.

ii) Remove any resulting set which has another as a subset. In this case, sets (2), (3), and (4) all have (1) as a subset. This is to remove any superfluous dependencies upon assumptions. Thus only sets (1), (5), and (6) remain.

iii) Remove any set which has a contradictory set of assumptions as a subset of it. Since $\{a_4\ a_5\}$ is already know to be a *no-good* set, this disposes of set (6), leaving sets (1) and (5) to form the new label for C.

iv) If C forms a new node (1) and (5) become its label. If it already exists, as in this example, (1) and (5) are *added* to its existing label, having first checked that it is either not a superset of an existing set in the label, or that it is not a subset of one, in which case the older superset is removed from the label.

v) If the [redacted] whole process changes the label for C, and the recorded justifications show that other nodes depend upon C, then the changes made to the label of C have to be *propagated* forward to those nodes, and from these nodes on to any that depend upon them, and so on. Note that this process does terminate, as a little thought will make clear.

By this stage the database of our problem-solver in this example looks like this:

Node	*Contents*	*Label*
n0	false	$\{a_4\ a_5\}$
n1	assumption 1	$\{a_1\}$
n2	assumption 2	$\{a_2\}$
n3	assumption 3	$\{a_3\}$
n4	assumption 4	$\{a_4\}$
n5	assumption 5	$\{a_5\}$
n6	A	$\{a_1\ a_2\}\ \{a_2\ a_5\}$
n7	B	$\{a_1\}\ \{a_2\ a_3\}\ \{a_4\}$
n8	C	$\{a_1\ a_3\ a_4\}\ \{a_1\ a_2\ a_3\ a_5\}$

4.4.1 Some Comments

The de Kleer type ATMS has two advantages over the Doyle TMS. First, all consequences of a set of assumptions can be explored together; no backtracking is involved, and the system does not have to attempt to maintain a single consistent set of assumptions, as in Doyle's TMS. As a result it avoids the problem of having to try to decide which assumption, or assumptions, should be put **OUT**, and thus the possibility of getting into some complex **IN**ing and **OUT**ing loop. Second, it can be efficiently implemented, since the main operations involved are set union and superset/subset checking operations. If the sets of assumptions for each node are represented as bit strings these kinds of operations can be performed directly by hardware.

The obvious disadvantage of an ATMS is that the system can only really support forward-chaining reasoning by a problem-solver. There is no natural way in which it can be made to support reasoning towards a particular goal. This characteristic reflects its origins. De Kleer first proposed his ATMS as a more efficient truth maintenance system for his *Qualitative Reasoning* systems, which typically work by *envisioning* the way the world might develop over time, ie forward inferencing from some initial state of the world to all the possible future world states. De Kleer has, however, recently published a proposal for a variant of the basic ATMS which combines the advantages of the Doyle type dependency-directed backtracking systems with those of the ATMS, and which avoids the disadvantages of each, see [dK86].

This forward inferencing support capability tends to mean that an ATMS is best for problems in which most, or all, possible solutions are required to be found by a problem-solver.

One application, not anticipated by de Kleer, but for which an ATMS has some important advantages is in *AI-based Design Support Systems.* These are systems which aim to support a broad range of the activities engaged in by Designers during the design process. By characterising the design process as the exploration of the space of possible designs, an ATMS can be used to support effective and efficient explorations of a particular design space. (More on this will be said in lecture 9 when the Edinburgh Designer System will be discussed.)

4.5 The Scaffolding Analogy

One way of comparing the three truth maintenance mechanisms described above is to draw an analogy between the proof structures built by each of them — to support the construction of solutions by the problem-solver — and the kinds of structures which can be built using scaffolding — to support buildings during their construction. This is referred to as the *scaffolding analogy*, inspiration for which came from Peter Ross.

Chronological backtracking does not build any scaffolding since it does not attempt to maintain any proof structure ovaer the database of the problem solver. It simply dynamically builds and pulls down a tower representing the current proof being considered by the problem-solver. At the end of the task no record is kept of how any solutions found were derived.

In the Doyle type TMS which uses dependency-directed backtracking a scaffolding structure is built from a number of *roots* which form a set of consistent hypotheses, or assumptions. When an inconsistency between two, or more, hypotheses is found - by deriving a false node from them, a hypothesis from the root set is selected and removed in order to re-establish consistency. All the scaffolding built up from the hypothesis

removed is taken down, but a record of how to re-build it is kept, so that if the removed hypothesis is ever brought back into the consistent set, as a result of another hypothesis with which it is inconsistent being removed for example, the scaffolding it supported can be recreated.

An ATMS builds a scaffolding from a set of roots which represent all the assumptions made by the problem-solver. If any set, or sets, of these assumptions becomes inconsistent - by the problem-solver deriving a false node from them, the assumptions are marked as being inconsistent and no further scaffolding is built from this combination of roots. At no time is any of the scaffolding taken down.

5 Required Reading

No required reading is set for this lecture due to the shortage of Big Red KR books. Recommended reading is [dK84] and [Doy79]. This is designed to give you more opportunity to read what has has been set for lectures one and two.

6 Next Small Group Tutorial Preparation

Using the material contained in the required readings for lectures one and two (Chapters 1, 2, 3, 5, 14, and 18 from the Big Red KR book) prepare a defence of logic as a representation of commonsense knowledge and commonsense reasoning. You should aim to present your defence in about five minutes, and be prepared to take questions specifically on your case for a further five minutes. This is to be done for the next small group tutorials to be held on Thursday 4 February (for MSc students) and Friday 5 February (for AI/CS-3 students).

You are reminded that all full class and small group tutorials are to be considered mandatory, and that material covered in small group tutorials is classed as examinable.

7 Date For Handing In Class Work Exercise One

The date for handing in the first class work exercise set at the end of the notes for lecture two is Tuesday 2 February, ie by the end of lecture three.

References

[dK84] J. de Kleer. Choices without bactracking. In *Proceedings of AAAI*, 1984.

[dK86] J. de Kleer. Back to backtracking: controlling the ABTMS. In *Conference Proceedings of AAAI*, 1986.

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

[McC80] J. McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13, 1980.

[Non80] Non-monotonic reasoning. *Artificial Intelligence*, 13, 1980. Collection of papers.

[SS77] R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer aided circuit analysis. *Artificial Intelligence*, 9, 1977.

Tim Smithers
January 1988

Knowledge Representation and Inference Two

Lecture Four:

Uncertainty: A Question of Belief

or

Not Magic Numbers Again

"They say that Understanding ought to work by the rules of right reason. These rules are, or ought to be, contained in Logic; but the actual science of logic is conversant at present only with things either certain, impossible, or entirely doubtful, none of which (fortunately) we have to reason on. Therefore the true logic for this world is the calculus of Probabilities, which takes account of the magnitude of the probability which is, or ought to be, in a reasonable man's mind."

— James Clerk Maxwell.

1 Some Introductory Comments and Questions

What is uncertainty? Is it different from not knowing? Can you be uncertain about something you don't know? Perhaps it would be better to ask what is certainty? Is there a difference between knowing and being certain about something? Can you be certain about something you know? If we are certain or uncertain about something what is it that we are certain or uncertain of — what we know, or the knowing of something? Where does the degree of belief of something, or the confidence in a belief come in to all this?

It seems to be hard to be certain about what uncertainty is! For example, you are probably certain of your age, but not of when the next general election in this country is going to take place. We can describe your certainty about your age as a certain belief, that is something you believe in with no reason to think that your belief might be mistaken, and probably you have never believed otherwise. You may have a belief about when the next general election will be, but you can probably think of many reasons as to why you could be mistaken in that belief. Perhaps the only thing you can be certain of in believing when the next general election will be is that you cannot be certain about it. Can we say by how much we are uncertain about something, or how uncertain our belief in something is? For example, is it meaningful to talk of being 90% certain of when the next general election will be, or 50% certain, or 10% certain. If so, what does it mean to say this? Are these absolute values? If so how are they to be obtained? Or are they relative numbers? If so what are they relative to, and again how are they to be obtained?

Despite this uncertainty about what uncertainty is we can know a great deal more about uncertain situations besides the degree or strength of belief. There are many different kinds of uncertainty and a large number of approaches to resolving or discounting it. Some forms of uncertainty are preferable to others , and certainty depends not only on evidence, but also on the importance of the uncertain situation. The *utility* of evidence can be estimated, and then decisions can be made as to whether it is worth collecting any.

'Which way ought I to go to get from here?'
'That depends a good deal on where you want to get to.' said the Cat.
'I don't much care where —' said Alice.
'Then it doesn't matter which way you go,' said the Cat.

Lewis Carroll — *Alice in Wonderland.*

Another example, consider the following scenario:

John and Sally are lying dead in the middle of the living room surrounded by broken glass and water. The window is open. Who murdered them?

This is the scene for one of those puzzles in which you are allowed to ask the person who described the scene questions that can be answered yes or no. Is it from a position of uncertainty that you start to try to solve the puzzle, or one of not knowing, ie ignorance, or both? Do you try to formulate questions in order to try to reduce your uncertainty, or your ignorance, or both, or uncertainty with some questions and ignorance with others.

Often when faced with these kinds of situations we consider the evidence and then form hypotheses or conjectures for their explanation, which we then seek to prove or disprove by the collection of further evidence. But suppose that on the basis of what we are told in the scenario about John and Sally we form a hypothesis which after some questioning of the presenter results in additional knowledge, or evidence, which all confirms our hypothesis. Does this mean that we should adopt the hypothesis as a certain explanation? Probably not, since it may just be the next question that uncovers some contrary evidence that defeats the hypothesis. How long do we need to go on collecting supportive evidence for a hypothesis before we become certain of it?

Remarkably, in a world in which almost nothing is *certain*, our knowledge of uncertainty enables us to act as if almost nothing is *uncertain*. How can we represent and reason with knowledge about uncertainty in artificial systems?

tIn these notes four approaches to representing and reasoning about uncertainty are described: Bayes' rule, the Dempster-Schafer theory of belief, Alan Bundy's incidence calculus, and Paul Cohen's hueristic approach based upon his model of endorsements.

2 Bayes' Rule

2.1 Introduction

Thomas Bayes was a nonconformist minister who lived in Tunbridge Wells and was a Fellow of the Royal Society. He lived in the first half of the 18th century. In a memoir, published posthumously, Bayes showed how you could calculate the probability of a hypothesis, or explanation, in the light of the evidence available, it provides a simple mathematical method for updating the probability of a hypothesis as evidence is accumulate. It is therefore of potential use in *diagnosis* type problems.

2.2 Theoretical Development

Suppose you want to know the probability of Liverpool winning the first division of the English football league if by halfway through the season they have not lost a game. The probability we want is called the *conditional probability*, because it is the probability that Liverpool will win the league *in the event* of them having not lost a game in the first half of the season. If **win** denotes Liverpool winning the league and **notLostInFirstHalf** denotes the event of Liverpool not losing a game in the first half of the season, then $\mathbf{P(win|notLostInFirstHalf)}$ denotes the conditional probability that Liverpool will win the league in the event of them not having lost a game in the first half of the season.

The conditional probability of a hypothesis given some evidence is given by:

$$(2.1) \quad \mathbf{P(h|e)} = \mathbf{P(h \cap e)/P(e)}, \text{ where } \cap \text{ denotes set intersection,}$$

similarly

$$(2.2) \quad \mathbf{P(e|h)} = \mathbf{P(h \cap e)/P(h)},$$

from which we can derive one form of Bayes' rule:

$$(2.3) \quad \mathbf{P(h|e)} = \mathbf{P(e|h)P(h)/P(e)}.$$

Thus if we know the *prior probability*, $\mathbf{P(h)}$, that Liverpool will win the league; the *likelihood*, $\mathbf{P(e|h)}$, that not losing any games in the first half of the season will result in Liverpool winning the league; and the probability of the evidence $\mathbf{P(e)}$; we can calculate the *conditional probability*, $\mathbf{P(h|e)}$, of Liverpool winning the league given that they have not lost a game in the first half of the season. Using this form of Bayes' rule it is straightforward to revise the conditional probability of a hypothesis in the light of one piece of evidence.

This form is, however, not suitable if we want to deal with more than one competing hypothesis. For this situation we can extend equation (2.3), noting that the denominator of (2.3) can be written in terms of the conditional probabilities of **e** given **h** and $\neg$**h**:

$$(2.4) \quad \mathbf{P(e)} = \mathbf{P(e|h)P(h) + P(e|\neg h)P(\neg h)}.$$

Then Bayes' rule can be written as:

$$(2.5) \quad \mathbf{P(h|e)} = \frac{\mathbf{P(e|h)P(h)}}{\mathbf{P(e|h)P(h) + P(e|\neg h)P(\neg h)}}$$

This can be generalised for the case where h_i is one of n hypotheses:

$$(2.6) \quad \mathbf{P(h_i|e)} = \frac{\mathbf{P(e|h_i)P(h_i)}}{\sum_{j=1}^{n}\mathbf{P(e|h_j)P(h_j)}}$$

Note that the denominator of (2.6) involves the assumption that the n hypotheses are mutually exclusive and exhaustive, just as h and ¬h were mutually exclusive and exhaustive in (2.5).

The next extension of Bayes' rule is to deal with the situation in which there are several pieces of evidence for a hypothesis, instead of the single piece of evidence, e, of the forms presented so far. Equation (2.6) is extended to cope with two or more pieces of evidence per hypothesis as follows:

$$(2.7) \quad \mathbf{P(h_i|e_1 \;\&\ldots\&\; e_m)} = \frac{\mathbf{P(e_1 \;\&\ldots\&\; e_m|h_i)P(h_i)}}{\sum_{j=1}^{n}\mathbf{P(e_1 \;\&\ldots\&\; e_m|h_j)P(h_j)}}$$

But, a difficulty arises in trying to use this form of Bayes' rule to combine an arbitrary number of pieces of evidence in favour of a hypothesis. The denominator of (2.7) implies that we need to know the conditional probabilities of all possible combinations of evidence for all possible hypotheses. This requirement is clearly impractical for most applications, it is discussed further in section 4.2 when talking about Bundy's Incidence Calculus. To reduce the problem, by reducing the amount of information required, an assumption of *conditional independence* is often made. Formally, this is:

$$\mathbf{P(e_i \& e_j|h)} = \mathbf{P(e_i|h)P(e_j|h)}$$

In other words, if **h** is the true state of the world, then the observation of evidence $\mathbf{e}_i$ is independent of the observation of evidence $\mathbf{e}_j$. This assumption allows us to rewrite (2.7) as:

$$(2.8) \quad \mathbf{P(h_i|e_1 \;\&\ldots\&\; e_m)} = \frac{\mathbf{P(e_1|h_i) \times\ldots\times P(e_m|h_i)\times P(h_i)}}{\sum_{j=1}^{n}\mathbf{P(e_1|h_j) \times\ldots\times P(e_m|h_j)\times P(h_j)}}$$

This is the form of Bayes' rule usually used to revise the probability of one of a number of exhaustive hypotheses by *pooling* the weights of the pieces of evidence. We have seen that it is derived from the definition of conditional probability, and that it involves three assumptions:

1. that the n hypotheses $\mathbf{h}_i$ are mutually exclusive,
2. that the n hypotheses $\mathbf{h}_i$ are exhaustive, and
3. that there is conditional independence of the evidence for each hypothesis.

2.3 An Example

The use of Bayes' rule will now be illustrated by an example. Consider two boxes, each containing a set of red and yellow bricks, set A in the proportion of two red bricks for every yellow brick, and set B in the proportion one red brick for every two yellow bricks. To start with we don't know which box contains which set. Suppose we choose either one box or the other according to whether an ordinary coin, when flipped, lands heads or tails. Now select a brick from the chosen box, without looking, note its colour, and put it back; do this, say, four times. Suppose the result of our four selections, which we will call our evidence are:

selection	1	=	red
	2	=	red
	3	=	yellow
	4	=	red

In the light of the evidence, what is the probability of the chosen box containing one or other of the sets of bricks?

Having chosen one of the boxes we can form two mutually exclusive and exhaustive hypotheses:

$\mathbf{h}_1$: that the chosen box contains the set A.
$\mathbf{h}_2$: that the chosen box contains the set B.

If we have made our choice of box on a purely random basis the prior probabilities for each of these hypotheses will be the same, that is 1/2. This is written:

$$\mathbf{P}(\mathbf{h}_1) = 1/2 \text{ and } \mathbf{P}(\mathbf{h}_2) = 1/2.$$

Next we need to calculate the likelihood of the evidence given each hypothesis, that is $\mathbf{P}(\mathbf{e}|\mathbf{h}_1)$ and $\mathbf{P}(\mathbf{e}|\mathbf{h}_2)$. This calculation is presented in the table below: note the chances of selecting a red brick from set A is 2/3 and a yellow brick 1/3, and visa versa for set B.

Evidence e			Likelyhood	
selection			Set A	Set B
1	=	red	2/3	1/3
2	=	red	2/3	1/3
3	=	yellow	1/3	2/3
4	=	red	2/3	1/3
			$\mathbf{P}(\mathbf{e}|\mathbf{h}_1) = 8/81$	$\mathbf{P}(\mathbf{e}|\mathbf{h}_2) = 2/81$

Table 2.3.1 — Evidence and Likelihood for first four selections.

We can now calculate the probability of the evidence, $\mathbf{P}(\mathbf{e})$, using equation (2.4) above:

$$\mathbf{P}(\mathbf{e}) = (8/81)\times(1/2) + (2/81)\times(1/2) = 10/162$$

The conditional probabilities for each hypothesis, $\mathbf{P}(\mathbf{h}_1|\mathbf{e})$ and $\mathbf{P}(\mathbf{h}_2|\mathbf{e})$, can thus be calculated using the form of Bayes' rule in equation (2.5) or (2.6):

$$\mathbf{P}(\mathbf{h}_1|\mathbf{e}) = \frac{\frac{8}{81}\times\frac{1}{2}}{\frac{10}{162}} = \frac{8}{10}$$

$$\mathbf{P}(\mathbf{h}_2|\mathbf{e}) = \frac{\frac{2}{81}\times\frac{1}{2}}{\frac{10}{162}} = \frac{2}{10}$$

From which we might conclude that the selected box contains the set A, and not the set B. So, although $\mathbf{h}_1$ and $\mathbf{h}_2$ are equally probable initially, after the evidence is in, $\mathbf{h}_1$ has become more likely to be true than $\mathbf{h}_2$.

If you were still not certain enough of this conclusion you can go on to collect some more evidence. For example, suppose you make four further selections. The evidence and new likelihoods might look like table 2.3.2 below.

Evidence e			Likelihood	
selection			Set A	Set B
1	=	red	2/3	1/3
2	=	red	2/3	1/3
3	=	yellow	1/3	2/3
4	=	red	2/3	1/3
5	=	yellow	1/3	2/3
6	=	red	2/3	1/3
7	=	red	2/3	1/3
8	=	red	2/3	1/3
			$\mathbf{P}(\mathbf{e}|\mathbf{h}_1)' = 64/6561$	$\mathbf{P}(\mathbf{e}|\mathbf{h}_2)' = 4/6561$

Table 2.3.2 — Evidence and Likelihood for the eight selections.

The new evidence probability is given by:

$$\mathbf{P}(\mathbf{e})' = (64/6561)\times(1/2) + (4/6561)\times(1/2) = 168/13122,$$

and the new conditional probabilities become:

$$\mathbf{P}(\mathbf{h}_1|\mathbf{e})' = \frac{\frac{64}{6561}\times\frac{1}{2}}{\frac{68}{13122}} = \frac{32}{34} = \frac{16}{17}$$

$$\mathbf{P}(\mathbf{h}_2|\mathbf{e})' = \frac{\frac{4}{6561}\times\frac{1}{2}}{\frac{68}{13122}} = \frac{2}{34} = \frac{1}{17}$$

From which we might become more confident that our first conclusion, that the selected box contains set A, is true.

2.4 Some Comments

This example is an uncontroversial application of Bayes' rule. Controversy arises when we cannot fix the prior probabilities objectively by the spinning of a coin, for example. For instance, there is no objective way of fixing the prior probabilities of different possible causes of the result of some clinical trial. In such cases, when forming prior probabilities, one would have to rely, in part, on subjective impressions and untested ideas. This suggests that hoped for scientific conclusions, as expressed by conditional probabilities, would be similarly subjective.

More recent work on Bayes' rule has shown, however, that prior probabilities normally have relatively little influence on the corresponding conditional probabilities, and what influence they have diminishes rapidly as the evidence accumulates. This means that even if two scientists started out with very different initial probabilities for various hypotheses, sufficient evidence would soon bring about virtually indistinguishable conditional probabilities. For example, if, instead of making $P(h_1)$ and $P(h_2) = 1/2$ in the above example, we had started of with $P(h_1) = 2/3$ and $P(h_2) = 1/3$, and we had collected the evidence after eight selections, the posterior probabilities for h_1 and h_2 turn out as:

$$P(h_1|e)'' = \frac{32}{33},$$
$$P(h_2|e)'' = \frac{1}{33},$$

which compares well with the $\frac{32}{34}$ and $\frac{2}{34}$, obtained above for $P(h_1)$ and $P(h_2)$ respectively.

You will find that the larger the sample you take, the more similar are the two sets, of posterior probabilities. So, although there is a subjective element in the Bayesian approach to uncertainty reasoning, this does not imply that scientific conclusions in the Bayesian scheme are subject to the overriding influence of purely objective evidence.

There are however other problems which must be faced in any application of Bayes' rule in a reasoning system intended to cope with uncertainty. The most obvious derive from the assumptions which were made in the development of the general form of the Bayes' rule presented in equation (2.8) above. It is not always possible to satisfy the exclusive, exhaustive constraints on the set of hypotheses, and the requirement for conditional independence of evidence for each hypothesis.

Other difficulties include the problem that the Bayesian view of probability does not allow a distinction to be made between uncertainty and ignorance. In other words, it is not possible to tell whether a degree of belief has been directly calculated from evidence, or indirectly inferred from an absence of evidence. A related problem is that the single degree of belief is a point estimate, and we know nothing about its precision. An assessed degree of belief of 0.5, say, might mean that the probability of an event is exactly 0.5. Alternatively it might mean that the odds of the event occurring are somewhere between 4/6 and 6/4. Another problem is that single degrees of belief combine reasons for believing with reasons for disbelieving, when one might want them kept separate.

For a more extended analysis of the more practical problems associated with using Bayes' rule see [Cohen 85].

3 Dempster-Schafer Theory

3.1 Introduction

The Dempster-Schafer (D-S) theory, like Bayes' rule, relies on degrees of belief to represent uncertainty. Unlike Bayes' rule, however, it permits the assigning of degrees of belief to subsets of hypotheses. Using Bayes' rule a probability distribution is constructed over all individual *singleton* hypotheses, but in the D-S theory, a distribution is constructed over all *subsets* of hypotheses. This is of great advantage. For example,

imagine a world in which there are four car manufacturers: Nissan Toyota, General Motors (GM) and Ford. Imagine also a small, newly developing country, and the hypotheses about which car manufacture will come to dominate this country's car sales. In addition to the four singleton hypotheses there are three others: that Japanese car makers will dominate, that US car makers will dominate, and that one of the four will dominate. The D-S theory provides us with a way of assigning probabilities to all the hypotheses in this scenario; some of the intuitive assignments permitted by D-S theory are not handled cleanly by Bayes' rule.

3.2 Theory and Example

Suppose we obtain evidence that the probability of Japanese domination is 0.4. In the D-S theory we can assign the probability 0.4 to the set {Nissan, Toyota} without committing any of the 0.4 to either one of the sets. Using Bayes' rule we would have to, possibly artificially, divide the 0.4 between Nissan and Toyota, with the consequence that we *appear* to know more about their position in the market than we do. Furthermore, in the Bayesian scheme, we would be forced to assign the remaining 0.6 of the probability distribution (1.0 − 0.4) to the US car makers where it, too, would have to be divided between GM and Ford. In contrast, the D-S theory allows us to assign the remaining 0.6 to the set {Nissan Toyota GM Ford}, that is to the set that reflects our ignorance about who will come to dominate the market.

The property of the D-S scheme illustrated in the above example is that it distinguishes between uncertainty and ignorance. Evidence that Japanese car makers will dominate the market is just that — it is *not* evidence about Nissan or Toyota individually, and it is *not* evidence that US car makers will *not* dominate. Any claim about US car makers is made out of ignorance since we have only one piece of evidence, and that is about Japanese car makers. Ideally we want to limit our claims to the set {Nissan Toyota}, for which we have evidence of dominance, and to the set {Nissan Toyota GM Ford}, that is, the set reflecting out ignorance. The D-S theory permits this assignment because it permits probability distributions over subsets of {Nissan Toyota GM Ford}, whereas the Bayes' rule does not because its probability distribution is constructed over the individual hypotheses {Nissan}, {Toyota}, {GM}, and {Ford}.

The largest set from which we construct subsets, the set of all the singleton hypotheses we know, the set that reflects our ignorance about which singleton is true, is called the *frame of discernment*, denoted by Θ. The singletons in Θ are assumed to be mutually exclusive and exhaustive. Thus for our example, the frame of discernment Θ = {Nissan Toyota GM Ford}. If we have no evidence about which subset of Θ will dominate the market, then the probability assigned to Θ is 1.0. To the extent that evidence favours hypotheses constructed from subsets of Θ, such as {Nissan Toyota}, to that extent will the probability assigned to Θ be diminished. The D-S theory introduces a function, called a ***basic probability assignment***, denoted by m, to assign probability to the subsets of Θ. Thus, if our only evidence favours Japanese domination of a market to degree 0.4, them $m(\{\text{Nissan Toyota}\}) = 0.4$ and $m(\Theta) = 0.6$.

It is desirable to be able to *sum up* the probability assigned to subsets to determine the degree of belief in a superset. For example, suppose we obtain evidence about dominance of the car market in our imaginary developing country by GM and Ford such that $m(\{GM\}) = 0.45$ and $m(\{Ford\}) = 0.15$. Then the degree of belief in US dominance of the market is:

$$Bel(\{GM\ Ford\}) = m(\{GM\}) + m(\{Ford\}) = 0.45 + 0.15 = 0.6.$$

Obviously, $Bel(\Theta) = 1.0$, since it is the sum of the basic probability assignments of all subsets of Θ.

To use the D-S theory in an inference system, we need a method to incrementally update the values of belief functions Bel_i as evidence becomes available and directs the assignment of probability to subsets of Θ. The mechanism for this is called *Dempster's rule of combination.* Given two pieces of evidence and their probability assignments m_1 and m_2, Dempster's combination rule provides a means of computing a new probability assignment $m_1 \oplus m_2$ and a new belief function $Bel_1 \oplus Bel_2$. The method works as follows. Suppose we have two pieces of evidence about dominance of the market by members of $\Theta \stackrel{def}{=} \{Nissan\ Toyota\ GM\ Ford\}$. The first evidence is summarised by $m(\{Nissan\ Toyota\}) = 0.4$, and the second by $m(\{Toyota\ GM\ Ford\}) = 0.8$ [1]. Intuitively, it seems that since we have two pieces of evidence about Toyota, we should be able to make a probability assignment to {Toyota} alone; in fact, it seems we should assign a good probability to Toyota, since the assignment made by m_2 does not favour Nissan. Also, if we are able to make a probability assignment to Toyota, we should *reduce* the probability assigned to {Toyota GM Ford}, because a good part of the 0.8 assigned by m_2 we now know was assigned on behalf of Toyota. Similarly, we should reduce the probability assigned to {Nissan Toyota}. Table 2.1 shows an arrangement that captures these intuitive arguments.

cline2-4		m_2	
		{Toyota GM Ford}(0.8)	Θ(0.2)
m_1	{Nissan Toyota}(0.4)	{Toyota}(0.32)	{Nissan Toyota}(0.08)
	Θ(0.6)	{Toyota GM Ford}(0.48)	Θ(0.12)

Table 2.1 — Table Form of Dempster's Rule of Combination

In each cell of the above table we put the intersection of two sets: The intersection of {Toyota GM Ford} and {Nissan Toyota} is {Toyota}; the intersection of Θ and Θ is Θ; the intersection of Θ and any subset of Θ is that subset. We then simply multiply the probability assignments of the original sets to obtain the probability assignment for their intersection. Thus:

$m_1 \oplus m_2(\{Toyota\}) = 0.32$,
$m_1 \oplus m_2(\{Nissan\ Toyota\}) = 0.08$,
$m_1 \oplus m_2(\{Toyota\ GM\ Ford\}) = 0.48$, and
$m_1 \oplus m_2\ (\Theta) = 0.12$.

[1]Note: this latter hypothesis illustrates how the D-S theory is used to represent *negation*: $m(\{Toyota\ GM\ Ford\}) = m_1$ makes assignments not only to its subset of Θ, but also to Θ itself, that is, prior to combination, m_1 assigns 0.4 to {Nissan Toyota} and 0.6 to Θ; and m_2 assigns 0.8 to {Toyota GM Ford} and 0.2 to Θ.

Belief functions are calculated as before. Thus, belief that the Japanese will dominate a market is given by:

$$
\begin{aligned}
&Bel_1 \oplus Bel_2(\{Nissan\ Toyota\})\\
&= m_1 \oplus m_2(\{Nissan\ Toyota\}) + m_1 \oplus m_2(\{Nissan\}) + m_1 \oplus m_2(\{Toyota\})\\
&= 0.08 + 0.0 + 0.32\\
&= 0.4
\end{aligned}
$$

This simple scheme for combining probability assignments gives rise to a number of problems, but these will not be discussed here. See [BS84] for more details.

4 Incidence Calculus

4.1 Introduction

Incidence Calculus is a mechanism for reasoning about uncertain knowledge proposed by Alan Bundy in 1984. It attempts to overcome some of the problems associated with Bayes' rule and the Dempster-Schafer theory, and those of other numerical uncertainty representation schemes.

The two approaches described so far, and others like the system of *confidence factors* used in MYCIN, involve the assignment of numerical values to hypotheses, and the definition of arithmetic functions to the connectives AND, OR, and NOT in the case of rules, such as those used in MYCIN. The problem with such assignments is that the final computed value may not represent a meaningful probability of a hypothesis or assertion. They can often only be taken to be rather ill-defined measures of some kind of strength or cost. As such it is difficult to use these techniques for true probabilistic reasoning. Bundy argues (see [Bun84]) that no method based purely upon assigning numerical values can be so described.

4.2 Some Limitations of Purely Numerical Approaches

The laws of probability theory can be presented as:

LP1: $P(A \wedge B) = P(A) \times P(B)$, if A and B are independent
LP2: $P(\neg A) = 1.0 - P(A)$
LP3: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

These laws are contradictory if the condition of independence of A and B in LP1 is ignored. In probability theory, a *correlation* value, corl(A,B), between two events is used to denote their degree of dependence. This correlation function is defined by:

LP1′: $P(A \wedge B) = P(A) \times P(B) + corl(A,B) \times P(A) \times P(\neg A) \times P(B) \times P(\neg B)$.

The formula LP1′ can be used to replace LP1 when A and B are dependent. However, to use LP1′ in general requires us to know the values for the correlations of all combinations of the evidence involved. This would be reasonable if we had rules which allowed us to calculate corl([A ∧ B], C) from P(A), P(B), P(C), corl(A,B), corl(B,C). However, Bundy has shown that it is impossible to develop such a calculus. Thus, the "expert" knowledge provider would have to provide correlations for all combinations of evidence and hypotheses, which is impractical for all but the most trivial of problems.

4.3 Incidents

Incidence calculus is based upon the set-theoretic roots of probability theory, in which the probability of a formula is based upon a set of situations (interpretations, or possible worlds). Bundy calls these situations *incidents*. For example, imagine that we have a six-sided die and a coin. There are twelve possible incidents:

Die	Coin
1	head
1	tail
2	head
2	tail

etc.

Let W denote the set of all possible worlds, or incidents, in which the formulas of a theory are to be evaluated. So, from the above example,

$$W = \langle 1,\text{head}\rangle, \langle 1,\text{tail}\rangle, \langle 2,\text{head}\rangle, \langle 2,\text{tail}\rangle, \text{etc...}$$

The incidence $\imath(A)$ of a formula A with respect to W is the subset of W containing all those incidents in which A is true. In continuing the above example, if A represents face 2 being up on the die, then:

$$\imath(A) = \langle 2,\text{head}\rangle, \langle 2,\text{tail}\rangle$$

The dependence or independence of two formulas is the amount of intersection between their incidences. If two formulas are *independent*, then this intersection would be that obtained from a random assignment of values to the elements of their incidences.

4.4 Incidence Calculus

The following set of formulas define incidence calculus:

1. $\imath(\top) = \mathbf{W}$, ie $\top$, denoting true, is true in all incidents in $\mathbf{W}$.
2. $\imath(\bot) = \{\}$, ie false is true in no incidents.
3. $\imath(\neg A) = \mathbf{W} \setminus \imath(A)$, where $\setminus$ denotes set difference.
4. $\imath(A \wedge B) = \imath(A) \cap \imath(B)$, where $\cap$ denotes set intersection.
5. $\imath(A \vee B) = \imath(A) \cup \imath(B)$, where $\cup$ denotes set union.

To illustrate the use of this calculus, we will use the example introduced above about a die and coin. In this example:

- $\mathbf{W} = \langle 1,\text{head}\rangle, \langle 1,\text{tail}\rangle, \langle 2,\text{head}\rangle, \langle 2,\text{tail}\rangle$, etc...
- A = die with face 2 up
- $\imath(A) = \langle 2,\text{head}\rangle, \langle 2,\text{tail}\rangle$
- B = coin with head facing up
- $\imath(B) = \langle 1,\text{head}\rangle, \langle 2,\text{head}\rangle, \langle 3,\text{head}\rangle, \langle 4,\text{head}\rangle$, etc...
- $\imath(A \wedge B) = \langle 2,\text{head}\rangle$

4.5 Weighted probabilities

If w is an incident, let P(w) denote the probability of w occurring. If $\mathbf{W}$ is a set of incidents, then the weighted probability of $\mathbf{W}$, denoted $P(\mathbf{W})$ is given by:

$$p(\mathbf{W}) = \textstyle\sum_j P(w_j),\ w_j \in \mathbf{W}$$

In other words, the weighted probability of $\mathbf{W}$ is the sum of the probabilities of the incidents of $\mathbf{W}$.

In our example, since the incidents of $\mathbf{W}$ are disjoint, $P(\mathbf{W}) = 1.0$. If A is a formula, then P(A) is the probability of A being true and:

$$P(A) = P(\imath(A))$$

For example, if A represents die with face 2 up then

$$P(A) = \textstyle\sum_{w \in \imath(A)} P(w) = P\langle 2,\text{hed}\rangle + \langle 2,\text{tail}\rangle.$$

4.6 Representing Incidents

Incidents can be represented by bit strings. For example, using the die and coin again, we can represent the twelve disjoint incidents as follows:

w = 1 1 1 1 1 1 1 1 1 1 1 1
where the first 1 denotes ⟨1,head⟩, the second 1 denotes ⟨1,tail⟩ etc.

Now suppose that A denotes die with face 2 up, then $\imath(A)$ may be represented by:

$\imath(A)$ = 0 0 1 1 0 0 0 0 0 0 0 0

Similarly,

$\imath(\neg A)$	=	1 1 0 0 1 1 1 1 1 1 1 1
$\imath(A \wedge \neg A)$	=	0 0 0 0 0 0 0 0 0 0 0 0
$\imath(A \vee \neg A)$	=	1 1 1 1 1 1 1 1 1 1 1 1

Hence $P(A \wedge \neg A) = 0$ and $P(A \vee \neg A) = 1$. Thus, the method gives the required results for the two related events *die with face 2 up* and *die not with face 2 up*.

Suppose now that B denotes *coin with head up*. We can assign to B the following incidence:

$$B = 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0$$

giving:

$$\imath(A \wedge B) = 0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0$$
$$\imath(A \vee B) = 1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0$$

hence:

$P(A \wedge A) = 1/12$
$P(A \vee B) = 9/12 = 3/4$
assuming that each incident is equally probable.

4.7 Use of Incidences

Given $\imath(A)$ and a rule: [if A then X], we need to be able to calculate $\imath(X)$. There is, however, a problem: from [if A then X], all we can infer is that $\imath(A) \subseteq \imath(X)$ since X might also be true in incidents in which A is false. Thus, we can only calculate a lower bound on the incidence of X. But, if we have several rules each with the same consequent X, then we can take the *union* of the lower bounds as the lower bound for $\imath(X)$.

For example, suppose that we have a set of rules:

1. if A then X
2. if B then X
3. if C then X

we can then compute a lower bound for $\imath(X)$ as follows:

$$(\imath(A) \cup \imath(B) \cup \imath(C)) \subseteq \imath(X).$$

Continuing with our die and coin example, suppose we have the three rules:

R1: if⟨3,tail⟩ then ⟨odd,tail⟩
R2: if⟨5,tail⟩ then ⟨odd,tail⟩
R3: if⟨7,tail⟩ then ⟨odd,tail⟩

A lower bound for an ⟨odd, tail⟩ event can be computed as using:

$$\imath(\langle 3,\text{tail}\rangle) \cup \imath(\langle 5,\text{tail}\rangle) \cup \imath(\langle 5,\text{tail}\rangle) = 0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,1$$

giving a lower bound for $P(\langle\text{odd,tail}\rangle) = 3/12 = 1/4$.

That is, if we knew nothing about a particular die/coin throw other than the facts:

1. that each of the eight die faces were equally likely,
2. that each of the two coin faces were equally likely,
3. that (head∧tail) had probability 0.0, and (dieFace1 ∧ dieFace2) had probability 0.0, etc.,
4. that the die and coin events were independent, and
5. that the three rules R1, R2, and R3 applied,

then we would have a lower bound of $3/12 = 1/4$ for $P(\langle\text{odd,tail}\rangle)$. However, if we had another rule R4: if ⟨1,tail⟩ then ⟨odd,tail⟩, then we could improve upon this lower bound, it would become $4/12 = 1/3$.

4.8 Some Comments

For a more detailed presentation of Bundy's Incidence Calculus see [Bundy 84]. This approach to representing and reasoning about uncertainty has not received much attention by implementers of systems, and it needs more experimental testing to gauge how useful it is. One particular problem illustrated above which requires further investigation is that only being able to calculate a lower bound on the probabilities of formulae may not be strong enough in practice. However, it does offer some important advantages over other approaches: it captures the properties of probabilistic reasoning which purely numeric mechanisms can not, and it can be efficiently implemented using bit strings and set operations.

5 Heuristic Reasoning about Uncertainty

'What's one and one and one and one and one and one and one and one and one and one?'
'I don't know,' said Alice, 'I lost count.'
'She can't do addition,' said the Red Queen.

Lewis Carroll — *Through the Looking-Glass.*

In this last section a rather different approach to the representation and reasoning about uncertainty is briefly described. It is based upon the work of Paul Cohen who was concerned to develop a heuristic approach, as opposed to one based on numerical or probabilistic methods. Cohen's method is based upon what he calls a *model of endorsement*. Endorsements are structured objects which represent reasons for believing and disbelieving the propositions, of hypotheses, with which they are associated. Reasons for believing are called *positive* endorsements, and reasons for disbelieving are called *negative* endorsements.

5.1 The Model of Endorsement

The explicit marking of factors relating to one's certainty is called *endorsements*. The records of these factors are called endorsements. First, in order to set up a comparison of a probabilistic approach and one based upon endorsements, a semantic interpretation of probability theory is introduced.

The semantics of a probability theory provides us with, among other things, *mental devices* to facilitate comparisons of the situations we are interested in, or concerned with. For example, if we adopt a frequency semantics for probability, we might employ the mental device of a spinning pointer that comes to rest in some randomly determined sector of a disk. We can use this device to help interpret the statement P(A) = 0.9, by imagining that the disk is painted red in one sector and blue in the other, and that the red sector is nine times the size of the blue sector. Then the interpretation of P(A) = 0.9 becomes: my support for A is comparable to what would be provided by associating A with the red sector and spinning the pointer. This mental device not only helps interpretation of P(A) = 0.9 with frequency semantics, but it also helps us think about whether the frequency semantics is *appropriate* for the situation we are trying to represent.

An analogy, or metal device, with which to introduce the model of endorsements is that of a bureaucracy. A piece of work in a bureaucracy proceeds from one stage to the next contingent upon the endorsement of a bureaucrat. The job must satisfy certain, often formal, requirements before it is endorsed at any stage. Imagine, in place of bureaucrats watching over a job, a collection of rules watching over the development of a line of reasoning. Each rule endorses a step in the argument if it satisfies certain requirements. Thus, *endorsements* are just records that a particular kind of inference has taken place, and *endorsers* are just the computations that assert the records. Bureaucrats can require a job to be cleared by lesser bureaucrats before they even consider it; for example, a city council won't consider a proposal for some development project unless it is cleared by the planning department, and the planning department won't look at a proposal until it has been checked that it complies with the regulations first. Similarly, an endorser may require the conditions of a rule to have a certain level of endorsement before it will endorse the conclusion of the rule. For example, one endorser might endorse the conclusion of a rule only if the conditions were themselves endorsed as parameters derived from rules that do not introduce uncertainty, such as simple arithmetic transformations. Most conclusions accrue several, more or less stringent endorsements. The certainty of a hypothesis can be represented at its strongest endorsement. In terms of the bureaucracy analogy, one's confidence in a job is proportional to the degree of scrutiny and stages through which it has passed.

Accrual of endorsements as if by independent bureaucrats is a useful way to view the evidence for and against a hypothesis, but we need to extend this analogy to deal with the *weighting* of evidence. To do this we can imagine a bureaucrat using something rather like a ledger book in which there are three columns; one for evidence in support of a hypothesis, one for evidence which contradicts a hypothesis, and one for irrelevant evidence. Weighing evidence then has two stages: First, decide which columns each piece of evidence belongs, and , second, do the *accounting* of the evidence for and against. Endorsements are structures associated with evidence which provide the information

necessary for carrying out such operations.

5.2 Some Comments

Cohen has implemented his model of endorsements approach to reasoning about uncertainty in a system called SOLOMON, more details about which can be found in [Coh85]. Again, this technique has yet to be widely applied and so it is difficult to assess how effective it is in general. However, it has one attractive aspect to it which should make it appeal to those who do not like using any of the various *magic number* techniques, in that it attempts to model the heuristic reasoning it seems we adopt in reasoning about situations in which we are in some way uncertain.

6 Required Reading

The required reading for this lecture is all taken from the *Big Red KR book*:

1. Chapter 21, page 371 — Davis, R., Buchanan, B.G., and Shortliffe, E., *Production Rules as a Representation for a Knowledge-Based Consultation Program.*
2. Chapter 22, page 389 — Davis, R. and Buchanan, B.G., *Meta-Level Knowledge: Overview and Applications.*
3. Chapter 27, page 457 — Garvey, T.D., Lowrance, J.D., and Fischler, M.A., *An Inference Technique for Integrating Knowledge from Disparate Sources.*

References

[BS84] B. G. Buchanan and E. H. Shortliffe. *Rule-based Expert Systems*, chapter 13, pages 272–287. Addison-Wesley, 1984. *The Dempster-Schafer Theory of Evidence* written by J. Gordon and E. H. Shortliffe.

[Bun84] A. Bundy. *Incidence Calculus: A Mechanism for Probabilistic Reasoning.* DAI Research Paper No. 216, University of Edinburgh, Department of Artificial Intelligence, 1984.

[Coh85] P. R. Cohen. *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach. Research Notes in Artificial Intelligence 2*, Pitman Publishing, 1985.

[DB85] R. Davis and B. G. Buchanan. Meta-level knowledge: overview and applications. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, chapter 22, pages 389–397, Morgan Kaufmann, 1985.

[DBS85] R. Davis, B. B. Buchanan, and E. Shortliffe. Production rules as a representation for a knowledge-based consultation program. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, chapter 21, pages 371–387, Morgan Kaufmann, 1985.

[GLF85] T. D. Garvey, J. D. Lowrance, and M. A. Fischler. An inference technique for integrating knowledge from disparate sources. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, chapter 27, pages 457–464, Morgan Kaufmann, 1985.

Tim Smithers
February 1988

16/2/88

Knowledge Representation and Inference Two

Lecture Six:

Planning and Planning Systems

1 Introductory Comments

Planning: deciding upon some necessary actions before executing any of them.

Research into planning and planning systems constitutes a major subject of research in AI, and has done so since its beginnings. There is therefore a large amount of work reported in the literature. These notes aim simply to outline the major distinctions between the different approaches that have been developed, and to introduce some of the representation and reasoning issues involved by describing two of the earliest planning systems.

A final section reviews the relationship between the actions represented and reasoned about by planning systems and how they are actually executed in the real world. This leads to the identification of an important difference in approach to planning, and AI as a whole, being adopted by members of the robotics group in the Department. This approach is referred to as the *Task-structured* approach, or sometimes the *Behaviour-based* approach, since it uses *task-achieving behaviours.*

2 Approaches To Planning

Five approaches to planning problems can be identified. They are:

1. non-hierarchical planning,
2. hierarchical planning,
3. script-based planning,
4. opportunistic planning,
5. non-linear planning.

Before outlining each of these approaches the meaning of the term *hierarchical* as used in planning needs to be explained. Most plans have nested subgoal structures, and so might be described as having *hierarchical structures.* However, *hierarchical* has another interpretation, and it is this second one which is used to distinguish between approaches to planning problems. The intended distinction is that hierarchical planners

divide? [illegible]?

generate a hierarchy of *representations* of a plan as they are built, in which the highest level is a simplification, or *abstraction*, of the plan and the lowest level is a detailed plan, sufficient to solve the particular problem. In contrast, non-hierarchical planners have only one representation level of a plan. This kind of planner is typified by the STRIPS system (described in section 3). Both kinds of planner can generate plans with hierarchical subgoal structures, but only hierarchical planners use a hierarchy of representations of the plan.

2.1 Non-Hierarchical Planning

Non-hierarchical approaches do not distinguish between problem-solving actions that are critical to the success of a plan and those that are simply details of the plan. As a result, plans developed by non-hierarchical planners often get bogged down in unimportant details. For example, in planning a shopping trip to buy food etc. for a dinner party you are giving, you do not need to worry about the details of how you are going to negotiate your way around each shop. But you do want to worry about making sure you get everything you need, so that you do not end up having forgotten to buy the wine. Planning with too many details is a waste of effort, but plans that are too vague may not specify all the problem-solving operators necessary to satisfactorily solve a problem; a balance between these two extremes needs to be found for efficient planning. This balance is typically both problem and domain dependent.

Non-hierarchical planning is still useful in domains where there is very little or no difference between the importance (level of detail) of the actions that must be carried out to solve a problem.

2.2 Hierarchical Planning

The approach of *hierarchical planning* first seeks to construct a basic plan consisting of only the important, or critical, subgoals: buying both the food and the wine for your dinner party, for example. This high-level plan is then *refined* into a more detailed plan which contains more detailed subgoals. This refinement process is repeated until the initial plan has been refined into a complete sequence of subgoals expressed at a level which can be achieved using known operators. The advantage of this approach is that the plan is first built at a level at which details are not computationally overwhelming, and at which unsuitable plans can be identified without first going into detail.

Hierarchical planning can take several forms. One approach, typified by the AB-STRIPS system (described in section 4), is to determine which subgoals are critical to the success of the plan and to ignore, at least initially, all others. For example, the problem of acquiring a piano cannot be solved unless the two subgoals *select a piano* and *arrange for its delivery* are successfully accomplished. Thus, an initial plan for acquiring a piano might simply be described as: *select a piano; arrange for its delivery.* This plan might be refined to include more details that are important at the next level so that the plan becomes *select suitable musical excerpts with which to assess a piano in a few minutes; locate a good piano store; select a piano; select a good piano removal company; arrange for its delivery.* The advantage of considering the critical subgoals at a number of levels before going in to details is that it reduces search: by ignoring details, one effectively reduces the number of subgoals to be accomplished at any given level, or *abstraction space* as it is often called.

The first hierarchical planner was implemented by Newell and Simon in their GPS (General Problem Solver) theorem prover. The GPS approach to hierarchical planning differs from the ABSTRIPS approach. In the ABSTRIPS system, a hierarchy of abstraction spaces (plan representation levels) is defined in terms of some subgoals being more important than others. In the GPS approach there is only a single abstraction space defined by treating one representation of the problem as more general than others. GPS planned in an abstraction space defined by replacing all logical connectives by a single abstract symbol. The original *problem space*, or *ground level* of GPS defined four logical connectives, but many problem-solving operators were applicable to any of the connectives. Thus, they could be treated as details and abstracted out of the initial formulation of the problem. A problem was then solved in the abstraction space (the space with only one connective) and the solution mapped back into the original four-connective space.

Other approaches to hierarchical planning are different again from the ABSTRIPS and GPS systems. The ABSTRIPS system abstracted critical goals and the GPS system abstracted a more general representation of an aspect of its problem space. Another approach is to abstract problem-solving operators: plan initially with generalised operators that are subsequently refined to problem-solving operators required in the problem space. This approach can be taken one step further by abstracting both the operators and the objects in the problem space. In all cases, however, hierarchical planning involves defining and planning in one or more abstraction spaces. A plan is first generated in the highest, most abstract space. This constitutes a *skeleton* or *outline* plan to which details are added as lower abstraction spaces are searched. Hierarchical planning provides a means of ignoring the details that might obscure or complicate the search for a solution to a problem.

2.3 Script-Based Planning

A third approach to planning makes use of outline plans, but unlike hierarchical planning, these outlines are recalled from a store of plans, rather than being generated. The stored plans contain the outlines, or strategies, for solving different kinds of problems. They range in detail from specific plans for particular common problems to general plans for whole classes of problems. In this approach planning proceeds in two steps: first an appropriate outline plan is selected from the store for the given problem, and then the abstract steps in the plan are filled in with problem-solving operators for the particular problem. This *instantiation* process involves large amounts of domain-specific knowledge, often working through several levels of generality, and thus outline subplans, until a problem-solving operator is found to accomplish each outline step. If a suitable instantiation is found for each abstracted step, the plan as a whole will be successful.

This approach has much in common with that of *scripts* developed by Schank et al for natural-language understanding. Like the hierarchical planning approaches described above it provides a top-down approach to the creation of plans. It does this by building into the outline plans it stores *expectations* about the way problems can be solved, and thus about what a plan to solve a problem must contain.

"BLACKBOARD SYSTEMS"

2.4 Opportunistic Planning

A fourth approach to planning is described as *opportunistic planning* and has been developed by Hayes-Roth and Hayes-Roth in their studies of how human planning is done. Opportunistic planning is characterised by using a more flexible control strategy than in the previously described approaches. It is usually implemented using the *Blackboard model* of problem-solving: the blackboard being a "clearinghouse" for suggestions about plan steps, these suggestions being made by a set of planning *specialists* (Knowledge Sources). Each specialist is designed to make a particular kind of planning decision at some level of abstraction in the hierarchically structured blackboard datastructure[1]. Specialists do not operate in any particular order; the asynchrony of planning decisions being made only when there is reason to do so gives rise to the term *opportunistic*. In the Hayes-Roth's model of planning, which they suggest is like human planning, the ordering of operators that characterise a plan is developed piecewise. In other words, the plan *grows out* from concrete clusters of problem solving operators. For example, it is only after the subgoal *select a piano* has been identified, as a subgoal of the overall task of acquiring a piano, that the subgoal *select excerpts of music ...* is identified. In other words, if the subgoal *select a piano* had not been identified, then the subgoal *select music ...* would not have been identified.

Opportunistic planning includes a *bottom-up* element, since it is driven by opportunities to include detailed problem-solving actions in the developing plan. For example, having identified the overall task of acquiring a piano, a planning specialist may have recognised the need to pay for it and introduced in to the developing plan detailed actions such as going to the bank first, or testing for the presence of your cheque book in your jacket, etc.. It contrasts with the *top-down* refinement process characteristic of hierarchical planning, in which detailed problem-solving actions are not decided upon until the last stages of building a plan. Another difference between opportunistic planning and other approaches is that it can develop *islands* of plan actions, ie parts of a plan independently, while hierarchical planners try to develop an entire plan at each level of abstraction. This means that discovering a problem at one level of abstraction does not necessarily prevent the planning system from going on to do useful plan building at more detailed levels in other, unrelated, parts of the overall plan.

2.5 Non-Linear Planning

Two related problems recur in all approaches to planning. They are how to limit search and how to deal with interacting subgoals. The problem of search is concerned with finding an ordering of subgoal actions which will achieve the overall goal from the potentially very large number of subgoals that could be considered. This problem is sometimes referred to as the problem of *combinatorial explosion*, since the number of possible combinations of subgoals increases exponentially with the number of possible subgoals.

The problem of interacting subgoals occurs whenever a task has *conjunctive goals*, in other words, more than one goal has to be achieved to complete the overall task. The

[1]Note: this is a different use of the term hierarchical, it refers to structure levels, not representation levels.

order in which conjunctive goals are to be achieved is not always specified (it may not be known), but it can critically affect the finding of a solution. Sometimes interactions of this sort prevent any solution being found, for example, if the task of painting a room involves the conjunctive goals of painting the walls and the ceiling, the second goal *should* be achieved first, otherwise it becomes very hard to paint the ceiling without splashing the walls (which we are assuming are to be painted a different colour). If you did not know this you might find it impossible to paint the walls and ceiling of a room different colours, or you would have to work it out by some form of commonsense reasoning — which people are ironically typically not very good at.

The problem of search is related to the problem of interacting goals or subgoals because additional search results from premature commitment to an arbitrary ordering of them. In the painting the room example, a planner which arbitrarily decided to paint the walls first and then discovered that it would have difficulty painting the ceiling[2] would need to *backtrack* to the decision in the plan to paint the walls first. Backtracking involves replanning from the *decision point* that failed, in this case to paint the walls first. The disadvantage of backtracking is that it can be computationally expensive, depending upon the backtracking technique used and how much has to be done. This problem of not knowning how to order conjunctive goals and subgoals is a kind of uncertainty, which is why backtracking can become involved.

The interactions between subgoals have been called *constraints*, by Stefik. They can be inferred from the *preconditions* of operators if the preconditions are represented explicitly. For example, if the operator *PaintCeiling* has several preconditions such as *HavePaint*, *HaveBrush*, and *HaveLadder*, a planner might be able to infer that painting the ceiling can proceed, and suggest arbitrarily that it is done after painting the walls, despite the fact that it is not a good idea to do so if the walls are also to be painted. If the constraint *WallsNotAlreadyPainted* was added to the precondition list, our planner, if it suggested painting the ceiling second, would discover that this would contradict one of the preconditions of *PaintCeiling* operator. It should therefore be able to construct a plan which does not suggest painting the ceiling second.

Some of the early planning systems generated plans that violated such ordering constraints and then tried to go back and fix the plans. These systems applied a powerful heuristic called the *linear assumption* which says:

subgoals are independent and thus can be sequentially achieved in any order.

In an historical perspective, this can be seen to be an important heuristic. The number of orderings of the problem-solving operators is the factorial of the number of available operators, so clearly a problem-solver cannot reasonably examine *all* orderings in the hope of finding one which does not fail because of interacting operators, at least not without being computationally very expensive. The linear assumption says that in the absence of any knowledge about necessary orderings of operators, assume that any ordering that achieves the goal or subgoal will work. Notice that this is an example of a type of default reasoning. In this case it is a strong default assumption about the way in which subgoal operators can be combined to achieve higher level goals.

The linear assumption is used in cases where there is no a priori reason to order

[2]The planner having some yet to be realised commonsense reasoning capability to infer this.

operators. An alternative assumption is that it is better *not* to order operators at all, than to order them arbitrarily. This assumption has been applied in two different ways in planning systems. *Partial orders* of subgoal operators can be established by considering the interactions, if any, of their preconditions. For example, an agent taking the Knowledge Representation and Inference II course may know that it has two subgoals *attend the lectures* and *do the required reading set each week*, but initially it may not commit itself to an ordering of these operators. However, when it expands each of these goals, it realises that: i) the lectures can only be attended in the order they are given by the lecturer, and ii) a precondition of understanding successive lectures depends upon having done the reading set in previous lectures, it thus decides to order its subgoals as alternate *attend each lecture one*, *do required reading*, *attend lecture 2*, *do required reading*, ..., etc.. A planner can order operators only to eliminate problems that might arise from choosing an arbitrary ordering, or it can also not order operators until it knows how to order them. This second approach is called *least-commitment* planning, and planning systems which adopt it have to be able to represent, and perhaps express in the final plan, partially ordered operators. Such planning systems are called *non-linear* planning systems. An important aspect of this type of approach is that it is *constructive*; since planning decisions are made only when the planner knows they will not interfere with past or future decisions, the planner need never *backtrack* and undo a bad decision. In fact even non-linear planners use backtracking to some extent since it can turn out to be cheaper than trying not to make *bad* decisions.

3 STRIPS: A Non-Hierarchical Planner

3.1 Introduction and Historical Note

The STRIPS planning system was built as part of the mobile robot project carried out at the Stanford Research Institute (SRI) between 1966 and 1972. It is normally referred to as the Shakey project since this was the name of the mobile robot system used. In those days small portable computers were not available so the Shakey robot was controlled by a fixed PDP-10 computer which communicated with the mobile robot by a radio link. It was driven by two large stepper motors driving two wheels. It also had two load bearing castors, front and back. The main sensors were a camera (with motorised pan, tilt, focus, and aperture control), mechanical touch sensors, drive wheel shaft encoders, and a rangefinder. The robot moved about in an environment specially designed and constructed for it which consisted of four rooms connected by doorways and containing a few large, regular objects such as cubes and wedges, some of which were movable and some of which were fixed.

3.2 The Basic STRIPS System

The world model. The STRIPS system represented the world of rooms, doorways, and objects that the Shakey robot inhabited as a set of well-formed formulas in first-order predicate calculus. Some formulas represented constant facts, such as which objects were pushable and which rooms were connected by doorways. Other facts, such as the current location of movable objects, were modified to reflect the actions executed by the robot, and the consequent changes that occurred in its world.

Operators. The actions available to the robot for changing the world were described, for the purpose of finding a plan of actions, by *operators*. Typically operators described actions for going somewhere and for pushing an object somewhere, the locations being given as parameters. Each operator had *preconditions* to its application; to push a box, for example, the robot had first to be next to the box. The application of an operator was represented, in the world model, by making appropriate changes to it. These changes were defined by a *delete list* and an *add list*, which specified the formulas to be removed from and added to the world model as a result of the operation, respectively. Thus, each operator explicitly describes what it changed in the world model.

An example of a STRIPS operator is gotoB[3], which denotes an action for the robot to go to an object in the same room:

```
gotoB(Bx) ;;; goto object Bx
        preconditions: type(Bx,object),
                       thereExists(Rx) such that
                           [inRoom(Bx,Rx) and inRoom(robot,Rx)]
        deleteList   : at(robot,_,_), nextTo(robot,_)
        addList      : nextTo(robot,Bx)
```

The precondition statement requires that Bx be an object and that both Bx and the robot be in the same room, Rx. the "_" in the delete list represents arguments with any values whatever.

Method of Operation. The STRIPS system proceeded by searching a space of world models to find one in which the current goal could be achieved. It used a *state-space representation* in which each state was a pair (world model, list of goals to be achieved). The initial state was denoted (W_0, (G_0)), where W_0 is the initial world model and G_0 the given goal. A terminal state denoted a world model in which no unsatisfied goals remained.

Given a goal G, stated as a formula in the predicate calculus, the STRIPS system first tried to prove that G was satisfied in the current world model. It did this using a modified version of the resolution-based theorem prover QA3. Typically the proof would fail, within a prespecified resource limit, because no more resolvents can be formed. The STRIPS system then had to find a different world model that the robot could achieve and which satisfied the desired goal. Since such a task is not suited to a simple theorem prover a *means-end analysis* strategy was used for this stage of the planning problem, similar to the one used in the GPS of Newell and Simon.

To do the means-end analysis, the system extracted a *difference* between the goal and the current world model and selected a relevant operator to remove, or reduce, this difference. The difference consisted of any formulas from the goal that remained outstanding when the proof attempt was abandoned (possibly pruned if this set was large). A relevant operator was defined as one whose add list contained formulas that

[3]Note: a Prolog style of notation is adopted in these notes: predicates and constants are given names beginning with lower-case letters, and variables are given names beginning with upper-case letters.

would remove some part of the difference, thereby allowing the proof to continue.

If the operator was applicable, the system applied it and tried to achieve the goal in the resulting modified world model; otherwise, the chosen operator's preconditions became new subgoals to be achieved. Since there could be several relevant operators at each step, this procedure generated a tree of world models and subgoals. The STRIPS system used a number of heuristics to control the search through this tree.

3.3 An Example of the Basic STRIPS System's Operation

As an example, suppose the robot is in room1 and that the goal is for it to be next to box1, which is in the adjacent connected room, room2. The initial world model W_0 would then include clauses of the form:

```
inRoom(robot,room1),
inRoom(box1,room2),
type(box1,object),
connects(door12,room1,room2),
status(door12, open), ...
```

and the goal G_0 is given by:

```
go = nextTo(robot,box1).
```

G_0 is not satisfied by the initial state of the world model, and the difference between it and the initial model is ¬nextTo(robot,box1). The STRIPS system would then determine that gotoB(Bx), defined above, is a relevant operator, with Bx instantiated as box1. The operator instance goto(box1), denoted OP1, is not immediately applicable because the robot is not in the correct room, so its precondition,

```
G$_1$ = type(box1,object) and
    thereExists(Rx) such that
        [inRoom(box1,Rx) and inRoom(robot,Rx)]
```

becomes a new subgoal. Relevant operators for reducing the difference between G_1 and the initial world model W_0 are: OP2 = goThroughDoor(Dx,room2) and OP3 = pushThroughDoor(box1,Dx,room1), ie., move the robot to the room with the box in it, or move the box to the room with the robot in it. If the former operator is selected (as one would hope it would be), the precondition

```
G2 = status(Dx,open) and nextTo(robot,Dx) and
    thereExists(Rx) [inRoom(robot,Rx) and connects(Dx,Rx,room2)]
```

is a new subgoal. The difference ¬nextTo(robot,door12) can be reduced by the operator OP4 = gotoDoor(door12), which is applicable immediately. Applying OP4 adds the clause nextTo(robot,door12) to the world model, creating a new model W_1. G_2 is now satisfied with Dx = door12, so OP2 can be instantiated as got-through(door12,room2) and applied. This deletes the clause inRoom(robot,room1) and adds inRoom(robot,room2). G_1 is now satisfied, so OP1 is applied, deleting nextTo(robot,door12) and adding nextTo(robot,box1), the desired goal. The generated plan is thus:

```
OP4:  gotoDoor(door12)
OP2:  goThroughDoor(door12,room2)
OP1:  gotoB(box1)
```

The corresponding solution path through the state space tree is as follows:

$$\begin{tikzcd}
(W_0, (G_0)) \arrow[d] \\
(W_0, (G_1, G_0)) \arrow[d] \\
(W_0, (G_2, G_1, G_0)) \arrow[d, "\text{OP4}"] \\
(W_1, (G_1, G_0)) \arrow[d, "\text{OP2}"] \\
(W_2, (G_0)) \arrow[d, "\text{OP1}"] \\
(W_3, ())
\end{tikzcd}$$

3.4 Generalisation of Plans

In the basic STRIPS systems, each new problem had to be solved from scratch. Even if the system had produced a plan for solving a similar problem already, it was not able to make use of this fact. A later version of the STRIPS system incorporated a scheme for generalising plans and storing them, to assist both in the solution of subsequent problems and in the monitoring of the robot's execution of particular plans.

Triangle tables. A specific plan to be generalised, say, (OP1, OP2, ..., OPn), was first stored in a data structure called a *triangle table*. This is a lower triangle of an array representing the preconditions for and effects of each operator in a plan. Some of its properties are:

1. Cell$(i,0)$ contains clauses from the original world model that are still true when operator i is to be applied and that are preconditions for operator i, OPi.
2. Marked clauses (with a leading *) elsewhere in row i are preconditions for operator i added to the world model by previous operators.

3. The effects of applying operator i are shown in row $i+1$. The operator's add list appears in cell $(i+1,i)$. For each previous operator, say, operator j, clauses added by operator j and not yet deleted are copied into cell $(i+1,j)$.
4. The add list for a sequence of operators 1 to i, taken as a whole, is given by the clauses in row $i+1$, excluding column 0.
5. The preconditions for a sequence of operators i to n, taken as a whole, are given by marked clauses in the rectangular subarray containing row i and cell$(n+1,0)$. This rectangle is called the i-th *kernel* of the plan.

The triangle table for the previous example is given below, where the number of the operators has been changed to reflect the order in which they are executed:

1	*inRoom (robot,room1) *connects (door12,room1,room2)	OP1 goToDoor (door12)		
2	*inRoom (robot,room1) *connects (door12,room1,room2) *status) (door12,open)	*nextTo (robot,door12)	OP2 goThroughDoor (door12,room2)	
3	*inRoom (box1,room2) *type (box1,object)	nextTo (robot,door12)	*inRoom (robot,room2)	OP3 gotoB
4			inRoom (robot,room2)	nextTo (robot,box1)
	0	1	2	3

Method of generalisation. The plan is generalised by replacing all constants in each of the clauses in column 0 by distinct parameters and the rest of the table with clauses that assume that no argument to an operator has been instantiated. The result may be too general, so the proof of the preconditions for each operator is run again, noting any substitutions for parameters that contain the generality of the plan. Some further corrections may need to be made for any remaining overgeneralisation, which might make the plan either inconsistent or inefficient in use. Finally, the generalised plan, termed a MACROP, is stored away for future use.

In the above example, the generalised plan would be:

```
gotoDoor(Dx)
goThroughDoor(Dx,Rx)
gotoB(Bx)
```

with preconditions:

```
inRoom(robot,Rx2)
connects(Dx,Rx2,Rx1)
status(Dx,open)
inRoom(Bx,Rx1)
type(Bx,object)
```

and add list:

```
nextTo(robot Bx)
inRoom(robot,Rx1)
```

That is, the generalised plan sends the robot from any room through a connecting doorway to an object in the adjacent room.

Using the MACROP to guide execution. When the STRIPS system produces a detailed plan to achieve a goal, it did not necessarily follow that the robot executed the plan exactly as generated. One possibility was that some action failed to achieve its intended effect, or goal, so that the corresponding step of the plan needed to be repeated. Another was that the plan was found to be less than optimal and could be improved by omitting some steps entirely. The necessary flexibility during execution was provided using the MACROP rather than the detailed plan in monitoring the robot's actions.

At the beginning of execution, the parameters of the MACROP were partially instantiated to the case at hand. The robot then attempted, at each stage, to execute the highest numbered step of the plan whose preconditions were satisfied. This procedure omitted unnecessary steps and allowed repeated execution, possibly with changed parameters, on a step that had failed. If there was no step whose preconditions were satisfied, replanning occurred. Determining which step could be done next was accomplished by a scan that exploited the design of the triangle table.

Using MACROPs in planning. When the STRIPS system was given a new problem, the time taken to produce a plan was considerably reduced if a MACROP could be incorporated into its solution. The MACROP given above, for example, could be used as the first part of a plan to fetch a box from an adjacent room. The part of the MACROP consisting of its first two suboperators, if used alone, could also give a ready-made solution to the problem *go to an adjacent room* or it could be used repeatedly in solving *go to a distant room*.

The triangle table provided a means of determining whether a relevant macro operator existed. To determine whether the sequence of operators to i of the MACROP was relevant, the STRIPS system checked the add list of this sequence as given by the row $(i+1)$ of the table. Once a MACROP was selected, irrelevant operators were removed leaving an economical, possibly parameterised operator for achieving the desired add list. The operator's preconditions were taken from the appropriate cells of column 0. Thus, almost any subsequence of operators from a MACROP could become a

macro operator in a new plan. To keep new MACROPs from producing an overwhelming number of different operators to be considered during planning, the system contained provisions for preventing consideration of redundant parts of overlapping MACROPs and for deleting MACROPs that have been completely subsumed by new ones.

4 ABSTRIPS: A Hierarchical Planner

A combinatorial explosion faces all problem-solvers that attempts to use heuristic search in a sufficiently complex problem space. A technique called *hierarchical search* or *hierarchical planning*, implemented in a system called ABSTRIPS by Sacerdoti, represented an early attempt to reduce the combinatorial problem. The ABSTRIPS system used an approach which tried to recognise the most significant features of a problem, develop an outline solution in terms of these features, and then deal with the more detailed features of the problem only after the outline plan had been shown to be adequate.

The implementation of this approach involved two distinct levels of problem representation. A simplified version of the problem, from which details had been omitted, was used to represent the *high level problem space* or *abstraction space*, and the detailed version, in a *ground space*. By an extension which allowed for several levels of abstraction instead of just two, a hierarchy of problem spaces is obtained. In general, each space in the hierarchy serves both as an abstraction space for the more detailed space just below it and as a ground space with respect to the less detailed space just above it.

4.1 Abstraction Spaces

Given the world models and operator descriptions of the basic STRIPS system, the first question is how to define the "details" that are to be ignored in the first pass at a solution. Sacerdoti's answer was to treat as detail certain parts of the operator preconditions. At all levels of abstraction, the world models and the add and delete lists of operators remain exactly the same. Such a definition of "details" was found to be strong enough to produce useful improvements in the problem-solving performance, while keeping a desirable simplicity in the relationship between each abstraction space and its adjacent ground space.

The preconditions for an operator are stated as a list of preconditions, or *literals* (as Nilsson often calls them), concerning the world model to which the operator is to be applied. The relative importance of literals is indicated by attaching to each a number called its *criticality value*. The hierarchy of problem spaces is then defined in terms of levels of criticality: in the space of criticality n, all operator preconditions with criticality less than n are ignored.

The assignment of criticality values is done only once for a given problem domain. The general ideas which should be reflected in the assignment are:

1. If the truth value of a literal cannot be changed by an operator in the problem domain, it should be assigned the highest criticality value.
2. If the preconditions for an operator include a literal L that can be readily achieved once other preconditions for the same operator have been satisfied, then L should

be less critical than these other preconditions.

3. If the possibility of satisfying literal L depends upon additional preconditions apart from those referred to in (2), then L should have a high, but less than maximum, criticality.

The actual assignment of criticalities is done by a combination of manual and automatic means. First, the system builder supplies partial ordering information, a ranking, for all the predicates that can appear in operator preconditions. The partial ordering serves two purposes: it supplies an initial criticality value for all instances of each predicate, and it governs the order in which the system will consider literals for possible increases (but not decreases) in criticality.

For example, consider an operator `turnOnLamp(X)`, with preconditions:

```
type(X,lamp) and thereExists(R) such that
                    [inRoom(robot,R) and
                     inRoom(X,R) and
                     pluggedIn(X) and
                     nextTo(robot,X)]
```

The partial ordering of predicates, reflecting an intuitive view of their relative importance, might be as follows:

Predicates	Initial Rank
`type`	4
`inRoom`	3
`pluggedIn`	2
`nextTo`	1

The assignment algorithm would first find that the truth of `type(X,lamp)` is beyond the power of any operator to change and therefore would set its criticality to the maximum; in this case 6. The algorithm would then find that `type(X,lamp)` is an insufficient basis for achieving `inRoom(robot,R)` or `inRoom(X,R)`; so these two literals would have their criticality raised to the next highest level, 5. Next `pluggedIn(X)` is considered, and a plan to achieve `pluggedIn(X)` found using only the literals already processed as a starting point. Hence, the `pluggedIn` literal retains its initial criticality of 2, and similarly, `nextTo(robot,X)` is given criticality 1. The result, after similar processing of the preconditions of the other operators in the domain, is a hierarchy of at least four, and possibly six, distinct problem spaces. The results of the assignment algorithm so far are summaries as follows:

Literals	Assigned Criticality
`type(X,lamp)`	6
`inRoom(robot,R)`	5
`inRoom(X,R)`	5
`pluggedIn(X)`	2
`nextTo(robot,X)`	1

4.2 Control Structure

A problem statement for the ABSTRIPS system, as for the STRIPS system, consisted of a description of the state of the world to be achieved. A solution was a plan, or an ordered set of operators, for achieving the desired world state. The ABSTRIPS system proceeded by forming an outline plan at the highest level of abstraction and successively refining it. The executive controller was a recursive procedure taking two parameters: the current level of criticality, defining the abstraction space in which planning is to take place, and a list of the nodes representing the plan to be refined. Before the initial call, criticality is set to the maximum, and the outline plan initialised to a single operator — a dummy operator — whose preconditions were those of the goal to be achieved. The ABSTRIPS system computed the difference between the preconditions and the current world model, found operators relevant to reducing the difference, and if necessary, pursued subgoals to satisfy the preconditions of the selected operators. During the process, any preconditions of less than the current criticality were ignored. A search tree was built from which, if the process succeeded, a fuller operator sequence leading from the initial world model to the goal could be reconstructed. This new outline plan, together with the next level down of criticality, were passed recursively to the executive for the next round of planning.

The search strategy adopted by the ABSTRIPS system was called *length-first*, because the executive procedure formed a complete plan for reaching the goal in each abstraction space before considering plans in any lower level space. This approach has the advantage that it permits early recognition of dead ends, thus reducing the work wasted in extending the search tree along fruitless paths involving detailed preconditions. If a subproblem in any particular space cannot be solved, control is returned to its immediate abstraction space, and the search tree is restored to its previous state in that space. The node that caused the failure in the lower level space is removed from further consideration and the search continued in the higher level space for a new outline plan. This mechanism, which clearly involved backtracking, suffers from the problem that no information was available at the higher level as to what caused the plan to fail.

Since backtracking can be computationally expensive, and also because each operator in an abstraction space may be expanded to several operators in the ground space, it was important for the ABSTRIPS system to produce good plans at the highest level. Two modifications to the STRIPS system were made to try to ensure that the ABSTRIPS system would do this.

First, whereas in the STRIPS system search tended to be *depth-first* and therefore sometimes found non-optimal solutions, the ABSTRIPS system made the order of expanding nodes in the search tree dependent upon the level of abstraction. At the highest level it used an *evaluation function* that sometimes increased the search effort but which ensured that the shortest possible solution sequence was found[4].

The second modification was to do with the instantiation of operator parameters, in cases where two or more choices were equally good. While the STRIPS system made a choice arbitrarily, the ABSTRIPS system deferred the choice until a greater level of detail indicated which one was preferable. Backtracking still sometimes occurred since the choice could still turn out to be a bad one.

[4]Note its relationships to the A^* search algorithm.

4.3 Performance

The performance of the STRIPS and ABSTRIPS systems was compared over a series of problems. One of the longest, needing eleven operators for its solution, required the robot to open a door, go through the adjacent room to another room, push two boxes together, and then to go through two more doors to reach the room where it was to stop. The basic STRIPS system (ie not using MACROPS) took over 30 minutes of computer time to find the solution; the ABSTRIPS system took about five and a half minutes and generated only half the number of search-tree nodes.

5 Some Comments

5.1 The Frame Problem and Qualification Problem Again

The *frame problem*, introduced in lecture two, can be understood using the analogy of the changes between the frames of an animated film. In very simple animations, certain characters move in a fixed background in successive frames. In more complicated animations (and perhaps more realistic ones) changes occur in the background as well. The STRIPS operators can be thought of as only being able to represent the goings on in a simple animation in which the background is fixed.

The problem of specifying which formulas in a STRIPS world model should change and which should not is what is called the *frame problem* in AI. The best way of dealing with it depends upon the complexity of the world and actions being modelled. Roughly, if the components of a world model are closely coupled or unstable, then each action might have a profound effect upon the world state. In such worlds STRIPS operators would be very hard to build which adequately represented the goings on in the world. If, however, the components of the world being modelled are sufficiently decoupled to permit the assumption that the effects of actions are relatively local, STRIPS type operators may be adequate for representing the goings on in the world. Typically, the frame problem becomes more pressing as the level of detail of the world model required increases, since the representation and operators must take into account couplings between components which might be safely ignored at higher levels of description.

The Qualification problem concerns how to deal with anomalous conditions. For example, the STRIPS operator gotoB(Bx) with its defined preconditions, can be regarded as an appropriate representation of a robot action under *normal conditions*. But what happens if there is something in the path of the robot not represented in the world model, or some other unusual situation which would cause gotoB(Bx) to fail? It would not be practical to include as preconditions the negations of all the conditions which might cause the failure of the operator, since we probably cannot know them all. Yet if any of them did occur our simple model would fail.

A FAILING OF THE CLOSED WORLD ASSUMPTION.

5.2 Circumscription Again

The STRIPS world model and operators with preconditions, and add lists and delete lists, can be viewed as a weak implementation of McCarthy's Circumscription calculus. The approach attempts to deal with both the Frame problem and the Qualification

9 RORGAN.

problem in an adequate, yet tractable way. It is worth noting that although its weaknesses in this respect can clearly be seen no other approach to planning problems has demonstrated a significant improvement upon the performance of the STRIPS and ABSTRIPS systems in this respect.

5.3 Failure-Directed Backtracking

One of the weaknesses of the ABSTRIPS system identified above was that if a plan failed at one level the system had to backtrack up to the next level of abstraction to try an alternative refinement, but without knowing anything about why the previous attempt had failed. As a result the next attempt may fail for the same reason. For a different approach to this problem see Chris Malcolm's MSc Thesis, *Planning and Performing the Robotic Assembly of Soma Cube Constructions*. The approach used in this system is called *failure-directed backtracking*. It attempts to make use of knowledge about why one plan generation attempt failed to prevent others suffering from the same problem being considered.

Chris Malcolm's planner is also an example of a system which uses abstractions over operators and objects at the different levels of a hierarchical planning scheme. It introduces the various complications of getting a robot manipulator and gripper to perform certain pick and place operations gradually.[5]

5.4 Hierarchical Planning and Diagnosis Problems

It is worth noting that despite the strong similarity between generating plans of actions for robots, or people, and that of generating diagnoses of illnesses, there has been few if any attempts to use techniques typically used planning systems for diagnosis systems, or the other way around. For example, the hierarchical, least commitment approaches, and criticality value techniques used in the ABSTRIPS system could clearly benefit some of the problems of constructing a diagnosis for complex or multiple illnesses. The combined use of hierarchical representation, least commitment and criticality values, provides a much sounder approach to dealing with uncertainty than the Confidence Factors, or magic numbers, used in MYCIN type systems, for example.

6 Task-Structuring and Task-Achieving Behaviours

6.1 Historical Note

From an historical perspective it is interesting to note that many of the early approaches to AI-based planning were motivated by thoughts of programming robots. Some people even built robots to execute their plans, the Shakey robot being the most famous example. The Freddy-2 robot system built in the Department in the early 1970's was another example, this time of a robotic assembly system. Since those early days robots have been less and less visible in the thoughts, concerns, and writings of AI-based

[5] If you have not seen a demonstration of this system working, you should ask for one sometime, but not during term time. It is the first robot planning system to produce reliable plans that are robustly executed by a real robot manipulating real objects since Shakey.

planning people, though the problems of programming any sort of robot to do anything are still far from solved.

6.2 Building Robots Is Harder Than Programming Computers

There seems to be two reasons for this; one to do with timing, the other to do with a well known classical effect. First the timing. In the early 1970's it was much harder to engineer a robot system capable of doing interesting and useful tasks reliably, than it was to write computer programs. In those days LISP was quite widely available in the US, and POP-2 had been developed in Edinburgh, but the "industrial robot" had yet to be invented. This led to a situation in which the second effect became active. This is the classic effect that when faced with two problems, one of them easier than the other, it is always the easier one which is attacked. Consequently robot planning people became increasingly interested in trying to simulate the real world and to build planning systems which could construct plans that worked in these simulated worlds. The task of understanding how to build reliable robot systems was forgotten about in AI research.

6.3 The Cost Of Simulation

In order to simulate even a limited subset of the real world you have to make assumptions about the coupling between objects and events in the real world; the *causal relationships* and about the way the objects in it behave, otherwise the problem of simulating it becomes too large and complex. It is one of the great advantages of computer based simulations that you can decide how complicated it is to be, and so can always keep the scale of the problem down to one you can manage, which means you seldom choose to go and do something else — like build a real robot, for example. In making these necessary assumptions about the real world you also make, usually by implication, assumptions about the way in which tasks can actually be done in the real world. It turns out that in the case of robots these implied assumptions lead us to believe that many things are possible to do with a robot which are actually either very difficult, or impossible.

This situation is reflected by the fact that the most successful AI-based planning systems today are those that are concerned with problems of resource planning (the resources being used by people), and project planning (the project being conducted by people). The reason for this can be explained in terms of what the reliable operators are that the plans are expressed in terms of. If the operators of a plan are to be executed by people, then, unless they are particularly difficult for people to perform, they can usually be taken to represent reliable operators. They are also likely to be robust with respect to the numerous anomalous events that it is very hard to take account of in a planning system. This is because people are typically very good at correcting for such events "on-the-run", and at also recognising what to do to maintain the world in such a way that it always adequately matches the world model used in a planning system, ie., people are good at maintaining the *normality* that is necessarily assumed to a high degree in planning systems to make the planning problems tracktable.

6.4 Robot Systems Cannot Maintain The Normality Required By The Planning Systems

Robots, on the other hand are very different. Compared to our apparently trivial ability to maintain normality, robots are typically useless. They can work well if the world in which they operate stays within some tightly bounded sense of normality, but if anything happens for it to stray just a little outside these bounds, anything can happen, and often does. It is for this reason that today's robot systems can only operate successfully, and therefore economically, in highly engineered worlds, or only very poorly in unengineered natural worlds.

In this sense then, today's robot systems are very unintelligent compared to us. Their ability to cope with anything other than a highly engineered normality is almost nil.

6.5 Task-Structuring: A New Approach To AI

The problem of understanding how to build robot systems which are able to deal with a wider degree of variation in their working environments, and thus be more useful, and *intelligent*, has stimulated a number of people to adopt a new approach to building intelligent systems, and therefore to AI. This approach is perhaps best represented by Brooks [Brooks85] who advocates the building of insect-like robots as the way to understanding how to build intelligent agents which can operate usefully in the real world that we occupy, as opposed to the symbol processing worlds most people build in today's computer programs.

In the AI Department at Edinburgh a similar approach is being adopted towards the problems of planning and programming robotic assembly tasks, and to the problem of building artificially intelligent systems in general. It is referred to as the *Task-structured approach* or the *Behaviour-based approach*. Its principle aim is to understand how the task of operating reliably and robustly in a real world can be broken down into sets of subtasks which can be realised in practice using the various engineering techniques and knowledge we have today for building real artificial systems — *not simulations*. It also seeks to take advantage of work already done, and still being carried out, in various other fields of animal and human physiology, neurophysiology, cognitive psychology, philosophy of mind, and cognitive science. An important difference between the Task-structure approach, and that of the classical approach, which make assumptions about what tasks a robot system will be able to successfully execute and about how their execution will be coupled in the real world, is that it seeks to explicitly identify how tasks can be, and are, structured in terms of reliable and robust operators that can be realised by human engineered devices.

The Task-structured approach has, therefore, a strong bottom-up theme which distinguishes it from the strong top-down approach adopted by people trying to replicate and model parts of the so called higher cognitive abilities of man.

7 Required Reading

There are no papers explicitly on planning in the Big Red KR Book[6]. This weeks required reading is therefore designed to support material to be covered in the next two lectures.

1. Chapter 12, page 245 — Minsky, M., *A Framework for Representing Knowledge.*
2. Chapter 28, page 467 — Hayes, P.J., *The Second Naive Physics Manifesto.*

8 References

1. Barr, A. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence*, Vol I, 1981, Chapter D5, page 128, and Chapter D6, page 135.

   Brooks, R.A., *Achieving Artificial Intelligence Through Building Robots*, MIT AI Memo 899, May 1986.
2. Cohen, P. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence*, Vol III, 1982, Chapter 15, page 513.
3. Genesereth, M.R. and Nilsson, N.J., *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987, Chapter 12, page 285.
4. Malcolm, C.A., *Planning and Performing the Robotic Assembly of Soma Cube Constructions*, MSc Thesis, DAI, 1987.
5. Nilsson, N.J., *Principles of Artificial Intelligence*, Springer-Verlag, 1980, Chapter 7, page 275, and Chapter 8, page 321.
6. Tate, A., *Project Planning Using A Hierarchical Non-Linear Planner*, DAI Research Report No 25, 1976.

9 Class Work Exercise

Construct a suitable set of STRIPS-like operators for use in planning the repair of punctures in the back wheels of bicycles. Assign criticality values (by hand) to the literals of the preconditions and explain how many levels of abstraction you would expect a planner to use in constructing a plan using your operators and criticality values./Briefly describe how similar this STRIPS-like method of planning is to how you generally plan activities, illustrating your answer with examples. ✓

This exercise is due to be handed in by the end of the lecture on Tuesday 1 March.

[6]This is probably due more to the interests of the editors, rather than it being the case that there are no papers on planning suitable for such a collection.

Tim Smithers
February 1988