CS323: Nonmonotonic Reasoning

Chapter 1

Lecture 1:

1.1 Introduction

1.1.1 A Brief History of Logic

The roots of logic can be traced back to Aristotle's work, where he developed a number of syllogisms which were supposed to capture human reasoning. The most famous example is:

> All men are mortal;
> Socrates is a man;
> from this can be deduced that Socrates is mortal.

In today's terms we would classify this logic as a subset of propositional logic with some elements of predicate logic.

Two thousand years later, Leibniz attempted to use a logical framework he developed for "computing arguments". His theory was that when people had disagreements, instead of discussing it they could compute the correct answer and thus prove who is right.

Although his logic introduced 0 and 1 as truth values, it was 150 years later when Boole showed that truth values do not add up like numbers, and introduced logical operators in his book "Laws of Thought". These are the familiar logical operators 'and', 'or', and 'not'.

Thirty years later Frege set up the ground for modern logic while studying both human reasoning and mathematical reasoning. His work in mathematical reasoning started a branch of mathematics called mathematical logic. This shows that the problem of formalizing reasoning is not a new one, and that much effort and ingenuity has been expended in this pursuit.

This course will concentrate on more recent developments in the use of logic, in particular, non-monotonic formalisms will be addressed. In the late 1970's and early 1980's the development of nonmonotonic logical formalisms began. Three frameworks for nonmonotonic reasoning were developed in parallel. This is not something professional mathematical logicians had expected, their interests having remained in other well established areas.

1.1.2 Logic and AI

After Turing's paper "Computing Machinery and Intelligence", which can be seen as the AI manifesto, AI has gone in two directions.

The first has biological motivation: humans are intelligent, so why not copy them to achieve intelligence. One biological approach was to directly imitate the neurons in the human brain, but this turned out to be a very uninformative, since it was proved that every neuron is a universal computational element. Another approach which became popular in the 1980s uses the idea of "neurons in the brain" only as a motivation for building distributed networks of simple computing elements. Nowadays this school of thought is usually referred to as connectionism.

The second has psychological motivation. It was started by Allen Newell in the 1950s with the famous "Logic Theory Machine". This was a program that analysed logic. In contrast, this course will use logic as the representation language for knowledge.

The primary distinction between the two approaches is that in their search for intelligence, the biological approach studies man while the the approach we take is based on studying the world. The relation between common sense reasoning and formal common sense reasoning is like the relation between mathematical reasoning and formal mathematical reasoning.

The AI problem can be divided into the epistemological problem and heuristic problem. During the its early stages, AI researchers were worried only with the heuristic aspects. When writing a chess program they would take a fixed representation and just try to improve the playing strategy by using better search techniques or heuristic rules. They were not concerned with the epistemological problem, of which facts should be known, and what they represent. A discussion of these problems can be found in the 1969 paper Some Philosophical Problems from the Standpoint of Artificial Intelligence[11].

1.1.3 Monotonicity and its limitations

Ordinary logic is monotonic. If A is a collection of sentences, we define $Th(A)$ to be the set of sentences logically deducible form A[1]. The monotonicity property is:

$$\frac{\begin{array}{l} Th(A) \\ A \subset B \end{array}}{Th(B)}$$

which states that the set of theorems is a monotonically increasing function of the set of sentences. Another way of saying this is:

$$\frac{\begin{array}{l} A \vdash p \\ A \subset B \end{array}}{B \vdash p}$$

stating that if p can be proved from the sentences in the set A, and B contains all of the sentences of A then p can be proved from the sentences in B. This is a purely syntactic definition. It only depends on the form of the sentences not on their truth nor on what they mean. Monotonicity can also be viewed form a semantic perspective. This can be stated as:

$$\frac{\begin{array}{l} A \models p \\ A \subset B \end{array}}{B \models p}$$

which states that the models of B are a subset of the models of A if B contains all the sentences is A, or if the sentence p is true in all models of A and B contains all the sentences of A then p will be true in all models of B.

[1] We will use "deducible" for standard monotonic deduction. When talking about nonmonotonic reasoning we use the term "inferrable".

1.1.4 Motivations for Non-monotonic Reasoning

Common sense reasoning is nonmonotonic. Imagine that we need to travel from Glasgow to Moscow. We can construct a plan of buying a ticket from Glasgow to London, and from London to Moscow, and prove that a certain plan will work in getting us to Moscow. But if we now assume that we lose the ticket while we are in London, the plan no longer works. Using monotonic reasoning, if we asserted that we can no longer get to Moscow, the system would be inconsistent, which would allow us to prove anything; not a desirable property. Nonmonotonic reasoning will enable us to deal with this situation, as the deduction that we can get to Moscow can be dropped.

Another motivation for nonmonotonic reasoning can be illustrated on the same example. When buying the ticket, the travel agent will tell us that we need to change planes in London, but will not tell us that if we lose the ticket in London we will have to buy a new ticket. However if we ask him "what will happen if we lose the ticket in London", he will know the answer, replying that we need to buy a new one. To capture communicational conventions of this sort also requires a nonmonotonic reasoning formalism.

Puzzles also have their conventions which are nonmonotonic. Although everyone knows that a river can be crossed by using a bridge, that is clearly not the correct answer when presented with the missionaries and cannibals problem 2.2.4 And even if we mention that there are no bridges, a skeptical adversary could come up with a helicopter, an airplane, etc. We cannot possibly list all the alternatives. Furthermore, when we give him the correct solution which involves crossing the river 8 times, he could always ask, "how do you know the boat has oars?" The defaults that people seem to use when solving puzzles include: nothing that is not mentioned can be used, and a tool works for what it is commonly intended for. To use express conventions in a logic, the logic must allow for nonmonotonicity. A discussion of these issues can be found in Circumscription—A Form of Nonmonotonic Reasoning and Applications of Circumscription to Nonmonotonic Reasoning[2].

Marvin Minsky argued that these and similar problems can not be solved in logic. The development of non-monotonic formalisms shows that logic can be extended or modified to be powerful enough to solve them.

Chapter 2

Lecture 2:

2.1 Introduction

The topic of this section are a collection of ideas first presented in the paper “Programs with Common Sense”.[13] This paper was written in 1958 but unfortunately it is not nearly as obsolete as it should be. The paper makes various claims as to what problems need to be overcome to achieve this area of AI, which it calls common sense reasoning.

2.2 The Advice Taker

One of the proposals of this paper was to create a program called the **Advice Taker**. If computers are to be intelligent they must be able to learn from their experience. But before any learning can be done there must be a way for the computer to represent what it has learned. The question of how to represent knowledge was what the **Advice Taker** addressed. This is a problem that while essential for the success of AI is clearly an easier problem that the entire enterprise. Why this was, and still remains an important approach to the problem can best be seen from an historical perspective.

2.2.1 Learning and Samuel's Checkers Program

The first attempt at making a computer learn was Samuel's checkers program. It was a program that played checkers against an opponent. It had the ability to modify its behaviour based on past experience. It had two basic ways of remembering or representing what it had learned.

1.It had a set of evaluation functions of positions. These were combined together in a linear polynomial with various weights. The total was an evaluation function of the entire position. The various functiqons computed such things as the number of single men, the number of kings, how much control of the centre the player had, and other values.

2. The program also remembered many specific positions. When it came across a position it had seen already it would use the evaluation function with look-ahead that was stored rather than compute it again.

The program learned by adjusting values. The naive way of doing this would be to play games and to adjust the values of the coefficents when there was a loss. However this brings up a credit assignment problem. There is no obvious way to decide which of the moves was the bad one. Samuel avoided this problem by training the parameters on games played by checkers masters. The values that made the computer play most like a master in a given situation were used. Using these methods the program had some success.

2.2.2 Representation of Knowledge

This method of representing past knowledge has some difficulties however. In checkers there is the following stratagem. If two men are trapped by one king on one side of the board, and the total number of men is equal, then the player who has trapped the two has a piece advantage on the other side. It can win on the other side with this piece advantage, and then the other player if forced to move and be taken on the first side. This is a well known strategy by checkers players, but there is no way of setting Samuel's coefficients to avoid losing to this behaviour. It is also difficult for it to use look ahead against this strategy, for the disaster happens a long way in the future.

This is an example of a type of knowledge that Samuel's program cannot learn because it is unable to represent it. As a first step in solving this problem therefore, one can ignore the problems of learning the information and settle for the ability to tell the program any strategy.

2.2.3 Adding Information

Changing the behaviour of many programs can be compared to education by brain surgery. If a student has problems in a class because he always forgets to take into account some type of fact, it can be imagined that a surgeon could tamper with his synapses to correct the behaviour. This seems to be neither the most desirable nor the easiest way to teach. It is preferable that the correct course of action be told to the student without having to tinker with his internals.

In the same way it is desirable that a program's actions should be modifiable by telling it information rather than rewriting its code. This also has the advantage that one need not know the internal structure of the Advice Taker in order to give it advice.

2.2.4 Elaboration Tolerance

An important feature of these instructions is that they are elaboration tolerant. The missionaries and cannibals problem is the following:

> There are three missionaries and three cannibals on one side of a river, a boat that fits two is on the same side as them. They must cross the river, but if the missionaries are ever outnumbered by the cannibals on either bank they will be eaten.

There is a simple solution to this problem as a eighteen state graph. An elaboration of this might be adding:

> There is an oar on each side of the river and one person can travel in the boat with one oar, but for two to travel two oars are needed.

It should be possible to add new information without changing the old sentences. If the problem had been stored as an eighteen state graph it would not be elaboration tolerant.

It is possible to be slightly more tolerant than this. An ability to cancel old information could be added. Here it should not be necessary to give the information to be deleted in exactly the form it is stored, conclusions drawn from it should also be deleted.

2.3 Declarative representation of Heuristics

It was envisaged that the **Advice Taker** would have a database of facts, specific situations and goals. It was to reason by forward chaining. If it inferred "I should do x", then it would carry out that action. This behaviour was to be reflexive, so that it could control its own retrieval operations from memory, and cause itself to make observations. The basic inference mechanism was to be fixed and the heuristics were to be introduced by sentences.

Unfortunately this idea of representing heuristics declaratively is still unfinished business today. Heuristics in problem solvers are as a rule built in. A resolution theorem prover might use the unit clause method, or a set of support, but they do not have a way of declaratively associating a particular heuristic with a class of problems.

2.3.1 Examples of Declarative Heuristics

Heuristics are difficult to represent declaratively. Examples of this is the heuristic to prove that two triangles are congruent. If one can find a place where two of the conditions for congruency are met, this is a better place to look to prove the third. This is not expressible in terms of any of the presently used ways of controlling resolution. This is not to say that some resolution strategy would not do the same searching as this heuristic, merely that there is no way to tell a resolution prover to act in this way, if it does not already do so.

Another example of this is discussed in "Map Colouring and the Kowalski Doctrine"[15]. The problem is to colour a map of the United

States with four colours, so that no adjacent states have the same colour. California has only three neighbours, therefore no matter what colour the other three states are given there will always be a colour for California. California may therefore be removed. Now Arizona has only three neighbours, this continues until only Kansas is left. The map can now be coloured by choosing a colour for Kansas and working outwards. This method works for many maps, in fact I know of no terrestrial map for which it does not work.

2.3.2 The Kowalski Doctrine

These are examples of how to guide reasoning. Thus they are the second part of Kowalski's doctrine **Algorithm = Logic + Control**. The class of answers is uniquely specified by the logic of the problem, how to find this class is given by the control rules or heuristics. A close connection between this and the control of Prolog programs can be seen. Most work on Prolog programs has been for general heuristics. No current work allows specification that is particular to a small class of problems. For instance the heuristic above is limited to map-colouring. How to include knowledge of this sort remains an open problem, nor does it seem a problem that can be avoided.

As before the problem of how to find and when to use heuristics can be simplified to the question of how to give heuristic advice. When simple knowledge was being given as advice, it was seen that having to rewrite the program was like education by brain surgery, trying to teach new search procedure is far worse in this respect.

2.3.3 Postponability

The solution that was given to the problem of map colouring used a heuristic that can be generalized. The solution of the problem was to choose values for a set of variables subject to certain constraints. Certain variables (the colour of California) were found to be postponable, that is the rest of the problem could be solved, and then, no matter how the other variables had been set, there would remain an acceptable value for the postponed variable.

A second example of this is the problem of going to the airport. The plan will consist of two variables, the first is the time of the flight, and the second the time the taxi that is to bring us to the airport is called. A search program could first choose a value for the time of the taxi, the see how long the wait until the next flight was. If as is likely this was unacceptable it would retry another value, and so on.

The time of the taxi is a postponable variable. No matter what time the flight leaves at there is a time for the taxi that is correct. Therefore in this problem the time of the taxi can be postponed, and the time of the flight can be chosen first.

Definition: A postponable variable is one that no matter how the goals not involving that variable are fulfilled, there always is a value for the variable that fulfills the rest of the goals.

2.3.4 Using Information

It is not always easy to turn the advice one is given into a program or plan. Work in this area has been done by Barbara Huberman/Liskov [14]. Her Ph.D. thesis was to take advice from a chess book, and see how to use this information to play chess. The information she chose to use was Capablanca's[1] instructions on how to win certain endgames. These were king and rook against king, king and two bishops against king and king, bishop and knight against king. The last of these is very difficult, and can take up to 34 moves.

The task was not to hack up a program that would play these ending but to develop a fair system that would take the information and use it in a uniform way. The method that was chosen was to introduce a *better* predicate. Look-Ahead was done until a better position was found and a move was made. If the *better* predicate was easily computable, and did not need too much look ahead, and guaranteed a win if a long enough series of better positions were found, then this method would always work.

This was considered fair because Capablanca's advice was a way to improve the position. In this way the program took the advice in the way it was originally presented, rather than the program being changed

[1] Capablanca was the World Chess Champion in the 1920's

to fit the problem. More will be said on fairness later.

The *better* predicate was not attaching numbers to positions[2]. It is common in many programs to use numerical values for judging states. However humans rarely can compare to entirely different states and say which is better. If Grand Masters are asked to compare two very different chess positions they rarely find the question meaningful. What they are good at is judging two similar positions. In this way a *better* predicate is more like the way humans act, and closer to the kind of advice that could be given to an **Advice Taker**.

2.4 The Common Sense Informatic Situation

In the paper [13] there is a definition of what common-sense reasoning is, "A program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows.". The experience of the last thirty years has shown however that while this is clearly one of the aspects of common-sense it is not the main point. Recently the Common Sense Informatic Situation has been as source of interest. This is where the system does not know which information in its large database is relevant. It must decide which data to use in solving its problems. It can not make the the trivial assumption that all the information is relevant as the computational explosion is insurmountable.

The problem of planning to go from Glasgow to Moscow, is an example of this problem. When the original plan is made one needs only to consider buying a ticket at Glasgow. The consideration of what action to take in the event that the ticket is lost should only occur if some evidence that this may occur is present. [3]

[2]There is a mathematical theorem that says any set of allowable **better** predicates can be encapsulated in a evaluation function. However there is no guarantee that the function will be readily computable.

[3]In the paper [13] the term 'deduce' is used which suggests monotonic reasoning. However this problem has been addressed since then with various forms of non-monotonic reasoning. In these notes deduce will be used for monotonic deduction, while infer will be used for non-monotonic deduction.

2.5 Objects and Fairness

When common sense is formalized in sentences there is knowledge that seems difficult to capture, or which seems unreasonable to have to mention explicitly. In this section two methods for approaching these problems are discussed. Both of these ideas come from [13] but the ideas there have been improved on to a certain extent, though the problems are far from solved.

2.5.1 Objects

An object is a symbolic expression. It has the property that some of it properties are inferrable from its form, but some are not. An example of this is the number "1776". For Americans, this is an object, the date of the Revolution. For others it might not be an object. A more obscure example of this is the number "1729". This was an object for Ramamujan, the Indian mathematician. He was in hospital in England, and Hardy, his mentor called to visit him. Hardy said that he had come in a taxi with the number 1729, and hoped that this was not a bad omen, as this is such a boring number. Ramunujan disagreed, as he immediately said that it was the smallest number that is the sum of two cubes in two different ways.

An object therefore is an expression that we remember something about. On solution to their treatment has been property lists. Consider the previous problem of going to the airport. If the problem was to be posed to the **Advice Taker** it would be given some sentences and the goal;

$$Want(At(I, Airport))$$

The **Advice Taker** would start with the goal. It would recognise that this is an object, it has thought about going to airports before, perhaps it has some heuristics that it knows apply, or some memory of what action it took last time.

2.5.2 Formal Generalization

If the **Advice Taker** did not recognise it 2.5.1 as an object, it could do a formal generalization of the goal. The following are the formal generalizations of the above goal $want(At(I,x)), want(At(x, Airport))$, $want(x(I, Airport))$. Information could be filed under the goal of getting to any place, of getting a unspecified thing to the airport, or of achieving a relation between the user and the airport.

How much of the work to achieve this behaviour has been realized is unclear. Some methods of pattern matching have been used successfully. This is usually done by using production that matches against the goal and returns a method. This is again very much built into the program.

This is not as general as it could be. More general would be a method that could retrieve information about any object, not just a goal, whenever it is mentioned. It seems that this generalization would be worthwhile, as the retrieval of heuristics and meta-knowledge seem to need this behaviour.

2.5.3 Fairness

The problem that the treatment of object bring up are often avoided in toy domains by 'cheating'. By cheating is meant giving the computer information hat it is unreasonable to expect it would have stored in that explicit form. All information that is retrieved, whether as a result of object recognition or other methods, should be plausibly present in the database.

The example that is given in [13] is that it is possible to drive to the airport. Instead of this being given explicitly, the fact that it is possible to drive between all places in the county is given. Then, as it is known that the airport and the home are both in the county, the required knowledge is derivable.

Often toy examples are unfair, or at least they do not discuss fairness. If a system is to work with a large amount of knowledge this point will be come more important.

Chapter 3

Lecture Three:

3.1 Philosophical Problems

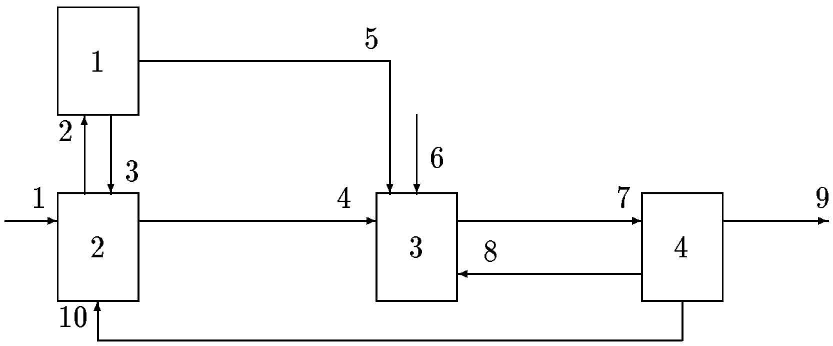
3.1.1 Free-Will and Determinism

The problem of free will is an area which philosophers have given a great deal of time to but which computer scientists, including those working in AI have by and large neglected. This is unfortunate as many of the problems in this area can be more easily stated and understood when robots or programs are considered than when the object of interest is man himself.

Consider the example of a robot who has two choices, to turn left or to turn right. The robot has to decide which of these it should do. If the robot were to say, "But I am completely deterministic, so therefore I cannot possible have a choice", it will not behave as well as we would like it to. What is wanted is the robot to consider each of the possibilities and to make a decision. For this it seems that it will be useful for the robot to consider it has free will, that is for it to consider that it has choices to make.

3.1.2 What's in a Can

The aim of this section is to formalize the notion of 'can', as in 'I can make coffee, but I won't'. It will be useful at certain times to be able to say that a certain robot could do a certain action but that it has decided not to. To examine this first we consider a very simple, deterministic model of a robot, a set of finite state automata.



The above is a picture of interconnected automata. The goal is to formalize what automata 1 *can* do. The behaviour of an automata is determined uniquely by is state and its inputs therefore the behaviour of automata 1 is determined, for a given initial state by the input from automata 2. We can represent this by the following equations.

$$
\begin{aligned}
a_1(t+1) &= A_1(a_1(t), s_2(t)) \\
a_2(t+1) &= A_2(a_2(t), s_1(t), s_3(t), s_{10}(t)) \\
a_3(t+1) &= A_3(a_s(t), s_4(t), s_5(t), s_6(t), s_8(t)) \\
a_2(t+1) &= A_4(a_4(t), s_7(t)) \\
s_2(t) &= S_2(a_2(t)) \\
s_3(t) &= S_3(a_1(t)) \\
s_4(t) &= S_4(a_2(t)) \\
s_5(t) &= S_5(a_1(t)) \\
s_7(t) &= S_7(a_3(t)) \\
s_8(t) &= S_8(a_4(t)) \\
s_9(t) &= S_9(a_4(t)) \\
s_{10}(t) &= S_{10}(a_4(t))
\end{aligned}
$$

Given these equations we can calculate the state of the whole system at any future time from the initial state.

The statement we now wish to formalize is,

> Automata 1 can put Automata 3 into state 7 at time 10, but it won't

Definition 1 *Is there a set of outputs from automata 1 which would put Automata 3 into state 7 at time 1*

This is the correct notion if we consider automata 1 as a computer program that knows about the other automata. A possible task this program could be given is to achieve a certain state at a certain time by giving a sequence of outputs. It is worthwhile at this point to separate the concept *can do* and *knows how to*, the second is a much stronger notion, which we shall touch on later, the former deals with possibilities and does not concern itself with the difficulties of working with knowledge.

There are two exogenous inputs, 1 and 6. A notion of ability that disregards these inputs, as the previous notion does, seems less interesting, as it is very limited. A more realistic notion is the following:

Definition 2 *A1 can put A3 into state 7 at time 10, if there is a set of outputs depending on inputs 1 and 6, and its other inputs that always force this behaviour*

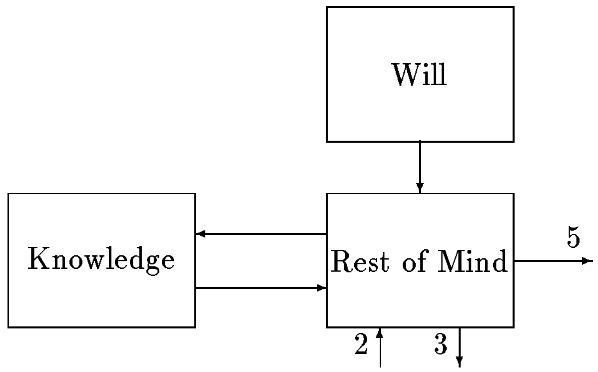
This is *can* from the viewpoint of an omniscient observer. It is a worthwhile notion of *can*, as in the statement, "A1 *can* do it, regardless of what happens, if only he knew what to do".

A weaker form of this is where we do not allow the output to depend on the exogenous inputs 2 and 6.

Definition 3 *A1 can put A3 into state 7 at time 10, if there is an automata that if it replaced A1, would do this.*

This definition of *can* is stronger than the previous one, as it is from the viewpoint of A1, not a omniscient observer. This seems very close to the idea of A1 as a yet unwritten program which we are analyzing to see what it could do.

A still more advanced notion is to take into account A1's knowledge of the world. A1 is an automata and there fore there is no obvious way to divide its internal structure into different parts, such as knowledge, control, or others. For the purposes of this analysis therefore we will split A1 into three sections.



A1 is know made of three subautomata that are connected as in the above figure. The automata are labeled, **knowledge**, **will**, and the **rest of the mind**. We consider the **rest of the mind** to be fixed. The definition of *can* for this model is as follows.

Definition 4 *A1 can put A3 into state 7 at time 10 if there is a signal that the* **will** *automata can send out, (which we will refer to as 'do it') that achieves the goal when we replace the* **knowledge** *automata by a certain other automata.*

Changing the **knowledge** automata is equivalent to changing A1's knowledge. The **will** automata sending out a certain signal can be viewed as the 'will' of A1 deciding to do the task. We have now defined *can* in the following way. A1 *can* do it, if there is a set of knowledge that would allow it to do it when its **will** decided to do it. We can ask does A1 *know how to*, by asking if the **will** sends out 'do it', will it do it.

Now that the previous definition of *can* has been fixed, the question arises should robots be programmed in this way. This seems to be a useful way of programming as it allows classes of tasks to be referred to, and has more "epistemological adequacy"[1] It also resolves the problem of free will for robots which was the starting point for this analysis.

[1]See [11] for a discussion of this.

3.2 Justifying the division of Knowledge and Control

A criticism that can be made of the analysis of automata that was carried out in the previous section is that it is completely arbitrary. A behaviourist might argue that this analysis is entirely presumptious. All that has been defined is a set of input output relationships, and thus there is no justification for splitting up the automata. Secondly, there might be many other decompositions in entirely different ways.

The division that was carried out above is not arbitrary, however. This can be shown in the following way. If the number of states and number of signals on each line are counted, it will be found that in some decompositions the number of signals is small with respect to the number of states. These decompositions have an objective character. A random decomposition will have a number of signals of the same order of magnitude as the number of internal states.

The reason for this is an analogous argument to Claude Shannon's [12] Ph.D. thesis. He was working on the realization of boolean networks with relays. In general to realize a given network requires a large number of contacts. Rather than prove that any given class of networks required a certain number of contacts he used the following counting argument. There are a certain number of ways of connecting n relays with m contacts each. The number of boolean networks that are of this size is much larger than this number, and hence there must be a large number of boolean expressions of this size not representable.

Similarly when we consider the analysis of A1 into parts, the number of three automatas with a bound on the number of signals that behave like it, are very few in number. Therefore the existence of a breakdown is significant. [2]

This can be seen by the following analogy. Imagine two Martians came down and saw a classroom. The first Martian decides to separate the room into separate entities, so that each person is a separate entity. The second decides to view the room as only two entities, all the heads

[2]That it is highly unlikely to be represented in a very different way is not proven by this, but that there os a very small number of different ways in which it is representable is.

grouped together, as one entity, and all the bodies grouped together as the other. The first Martian could claim that in his decomposition the amount of interaction is much smaller, and therefore there is an objective reason for preferring his analysis of considering people, rather than heads and bodies.

3.3 Reservations, Disclaimers, and Future Work

The previous section is not essential for the analysis of *can* that was carried out, as *can* was defined relative to a specific analysis. This sub-argument is just a justification for choosing this decomposition.

The division of a robot into will, knowledge, and rest of the mind, is probably eventually unarguable, but there are objective ways of deciding whether or not it is correct. There os some evidence that it is in the right direction. A lot of problems regarding ability can be restated into the form "If I decide, I will, because I know ... ". Here the question of decision, ability, and knowledge are separated. When robots are designed this separation may prove useful.

In the section several different forms of *can* are studied. That there are many different sorts is clear. Consider the sentence fragment, "Well, I could but I can't because it would ... ". Here *can* and *can* with some restraint are used. In general there can be many different kinds of restraint, fiscal, moral, or those based on conflicting choices. Further investigation into this needs to be carried out.

Chapter 4

Lecture 4:

4.1 Situation Calculus

4.1.1 Introduction

The situation calculus is a formalism for reasoning about actions and events in the world. The paper that introduced situations [11] defined them as follows, "A situation s is the complete state of the universe at an instant of time". The basic mechanism used in the calculus to define a new situation is the *result* function,

$$s' = result(e, s)$$

In this formula s is a situation, and e is an event, and s' is a new situation that results when e occurs. If an event occurs in a situation, a new situation results. The most common type of event is an action. The differences between events and actions has not yet been fully clarified.

4.1.2 Rich and Poor Objects

Situations are rich objects. A rich object is one that is not, and cannot be completely defined. Only partial knowledge can be ever known about a rich object. Consider the concept of driving home on a particular day. This can be looked at two distinct levels of detail. The first is the level of complete detail, which will record how long the car stops at each traffic light, and every other conceivable detail. In this regard, driving home is a rich entity. The other, is the level at which planning about such a trip is made. When an action like this is planned the planner does not decide how long he will press the brake at each stop sign. From his point of view driving home is a poor entity, in that his plan can be completely described.

At one time it was thought that the difference between rich and poor entities was that rich entities were in the past, and poor entities were in the future. It now seems that this is too simplistic. Though correct from the point of view of a planner at most times, it fails when a theory is used to reason about the past from evidence about the present.

It is worthwhile to contrast rich objects with the the notion of state of a finite system. In the formalised Missionaries and Cannibals problem there is a fixed number of possible states. However if the real world situation is considered, it is a rich object with all sorts of details ranging from the cannibal's names to the colour of the boat. When analysing this problem, the solution space is reduced to thirty two positions and situations are mapped onto states. This is a many to one mapping. Every possible situation is mapped onto one of the 32 states. If the extended problem with oars included is considered, the same situations are now mapped onto the extended system of 64 states. Reality is characterised by situations, and to analyse reality, a mapping from situations to states is made, that retains the important relationships for the particular problem.

If a theory of reality is to be made, different models at different levels of detail, should be looked at. It might turn out that the mathematical concept of inverse limit as used in topology might be useful. Reality might then be defined as the inverse limit of the models of reality.

4.1.3 Fluents

A function or predicate of a situation is called a *fluent*. This terminology is taken from Newton who considered the various properties of objects to be fluents. The height of an object, its velocity, or its acceleration were the common fluents he was interested in. He termed the rates of change of fluents, fluxions, and used this terminology in his development of the differential calculus. Most representations that have used the situation calculus so far have been discrete, and therefore fluxions have not been taken up by the AI community.

4.2 Issues of Syntax

The way that fluents are written raises various issues. The first, and most obvious way to write the fluent that the block x is on the block y at situation s is,

$$on(x, y, s)$$

The reasons that other notations are used is to *reify*[1] objects so that they can be quantified over. This is following the ideas of W. V. O. Quine [10], his theory being that in a theory the things that are real, are exactly those over which bound variables range. The following examples are in order of increasing reification.

$$\forall x\ y\ s.movable(x, s) \supset on(x, y, move(x, y, s)) \tag{4.1}$$
$$\forall x\ y\ s.movable(x, s) \supset on(x, y, result(move(x, y), s)) \tag{4.2}$$
$$\forall x\ y\ s.holds(movable(x), s) \supset holds(on(x, y), result(move(x, y), s)) \tag{4.3}$$
$$\forall x\ l\ s.holds(movable(x), s) \supset value(loc\ x, result(move(x, l), s)) \tag{4.4}$$
$$\forall x\ l\ s.holds(movable(x), s) \supset holds(at(x, l), result(move(x, l), s)) \tag{4.5}$$

[2]

In the second equation the action has been reified. This allows classes of actions to be referred to. The third equation allows classes of qualities to be predicated. The fourth makes the location of x a function of a situation, before this it was impossible to refer to where x was. The fifth makes *at* a more general propositional fluent, so that x can now be at more than one place. An example is that at the moment I am both at my desk, and at home.

[1]Reify comes from the Latin *res* meaning thing, thus reify means " to make a thing out of"

[2]The convention will be used in this text that functions and predicates of one parameter will have no parentheses and will associate to the right.

4.2.1 Preconditions for Actions

The above examples give very weak preconditions for actions to succeed. A more realistic formalisation might be,

$$holds(clear\ top\ x, s) \wedge holds(clear\ l, s) \wedge \neg heavy\ x \supset$$

$$holds(at(x,l), result(move(x,l),s))$$

The advantage of reifying can be seen when reformulate this as,

$$\forall y.value(loc\, y, s) \neq top\ x \wedge \forall z.value(loc\ z, s) \neq l \wedge \neg heavy\ x \supset$$

$$value(loc\ x, result(move(x,l),s)) = l$$

Here we have introduced *top* x as a location, and have defined clear to be equivalent to having nothing whose location is that position.

4.2.2 Irrevocable Commitments

In the previous equation, *heavy* x does not have a situation argument. This means that *heavy* does not depend on the situation. Therefore a commitment has been made that the weight of that item will not change. A precondition of the form $weight(x, s) \leq 10$ is more general. This would allow actions that could change the weight of an object. Without a different formalisation a change in the weight cannot be expressed.

When contexts are discussed later it will be seen that the context mechanism allows a way out of these seemingly irrevocable commitments.

4.3 The Frame Problem

4.3.1 A definition of the Problem

The frame problem is the following, there is an axiom that states, what happens to x when an action is performed on it. If another property is added to the ontology, the axioms which described what effect the action had on various fluents , do not give us any information on what happens to the new fluent. The usual intent is that the new fluent remains unchanged by the action, but this is not always the case.

Consider the following example. The axioms describing what happens to blocks when they are moved is,

$$holds(at(x, l), result(move(x, l), s)$$

To express that the colour of the block does not change the axiom,

$$holds(colour(x, c), s) \supset holds(colour(x, c), result(move(x, l), s))$$

To express all the fluents that do not change, an axiom would be needed for each action fluent pair. This seems extravagant. Some ways have been introduced to reduce the number of axioms needed. The use of frames is one method, another is non-monotonic reasoning. At this time it is not clear which is preferable.

4.3.2 The use of Frames and State Vectors

A way of approaching the Frame Problem is to using frames, these can be viewed as a kind of state vector. The simplest idea of a vector is the one used in the logical formalisation of programming languages[3]. To work with state vectors two operations are used, which will be referred to as a and c. These are referred to as assignment and contents operators. [4] $c(x, \xi)$ returns the contents of the variable x in the state vector ξ. $a(x, v, \xi)$ returns the new state vector resulting from assigning the value v to location (variable) x in state ξ.

a and c obey certain axioms the most basic being.

$$c(x, a(x, v, \xi)) = v$$

$$y \neq x \supset c(y, a(x, v, \xi)) = c(y, \xi)$$

The second axiom depends on the fact that the assignment operator, does not affect the value v. As a value, v cannot be affected by another action.

[3]The following notation was introduced by [7]

[4]These operators have been used since their introduction by other writers who have adopted the opposite convention for a and c — c standing for change, and a for access.

Two more axioms that are normally used in program analysis are,

$$a(x, v_1, a(x, v_2, \xi)) = a(x, v_1, \xi)$$

$$y \neq x \supset a(x, v_1, a(y, v_2, \xi)) = a(y, v_2, a(x, v_1, \xi))$$

It is not clear whether these axioms will play any useful role in the use of these functions in the situation calculus. In the mathematical theory of computation, these have been used extensively in proving compilers correct, mostly being used as simplifications.

These two operators can be used in situation calculus as follows. In situation calculus, only the values of different fluents are of interest. The situation s is treated as a state vector ξ. The effect of a move action is then defined to be,

$$\ldots, \supset result(move(x, l), s) = a(loc\ x, l, s)$$

with the following restrictions.

$$(x \neq y \supset loc\ x \neq loc\ y \wedge colour\ x \neq colour\ y) \wedge colour\ x \neq loc\ y$$

It might be thought that this adds the restriction that all blocks must be different colours, this is not so, as it is the concept of the colour of the blocks that are different, not the value that concept is assigned.

Using these operators the axioms for each kind of action can be written more compactly. The effect of an action has been abstracted to assignment of a value to a fluent, affecting no other fluents. This effect, of achieving a kind of orthogonality of actions and fluents is why the problem is so named. In geometry, co-ordinate frames are defined over space. If 3d space is considered, various co-ordinate systems can be defined. A mapping that in one particular frame changes only the x value, may in other frames change all of the co-ordinate values. In the previous section the choice of a set of fluents for a particular action that had the property of changing only one fluent, allowed the use of an assignment to describe the action. However, while in geometry the use of different co-ordinates schemes has proved very useful, in AI applications it has not been used overly. Mixtures of fluents like *loc* x and *loc* y, or even *loc* x and *colour* y have not found a use. Perhaps certain frames are more objective in AI than in geometry.

4.4 Limitations of Situation Calculus

4.4.1 Main Criticisms

The main criticisms of the situation calculus are that it can only consider single discrete actions that have a well defined result. Because of this concurrent actions are difficult to represent. Continuous actions and actions that return the world to its previous state are also difficult, if not impossible to represent. Other questions have been asked about the ability of situation calculus to handle natural death and multiple agents, many of these criticisms are discussed and overcome in [8].

4.4.2 Continuous Actions

Nothing in the definition of situation calculus corresponds to a restriction that actions must be discrete. In the first paper on situation calculus [11] a continuous action problem, the law of falling bodies was considered. However, despite this, there is a tendency to consider situation calculus as a formalism that only handles discrete actions.

The equation that describes falling bodies, is well known from high school physics, as

$$h = x_0 + v_0 t - 1/2gt^2$$

Unfortunately this is not in the form that the average person could understand. A more explained version, with less implicit assumptions is

$$\begin{aligned}
&\mathit{body}\ x \wedge height(x,s) = h_0 \wedge falling(x,s)\\
&\quad \wedge up_velocity(x,s) = v_0 \wedge time(s) = \tau\\
&\quad \wedge h = h_0 + v_0\ t - 1/2g\ t^2\\
&\quad \wedge h > 0\\
&\quad \supset \exists s'.future(s',s) \wedge height(x,s') = h\\
&\quad \wedge time(s') = \tau + t \wedge up_velocity(x,s') = v_0 - g\ t
\end{aligned}$$

A formula like this if it is to be use to a computer needs a lot of common-sense information around it. h_0 is the height of the object, v_0 is the initial velocity, g is the gravitational constant. The form given with these made explicit is closer to what a robot that was designed to handle bodies might need in its database. Other things could be added, such as,

$$\forall s''.future(s'', s) \wedge future(s', s'') \supset falling(x, s')$$

which states that the body was falling for all the times between the start and the final time.

Notice that time has become a function of situations rather than vice-versa. Also the concept of *future* has been introduced. This fluent is true if its first argument is in the future of its second.

This axiomatisation can itself be criticised for not being general enough. This may not be a problem that can be overcome. It seems that there always is a more general description of any scene. Consider the writing of Gallileo, nowhere does he explicitly state that what he speaks of is only true on Earth. The current mechanism for treating this is contexts, where we would write,

$$ist(C_{physics}, h = x_0 + v_0 t - 1/2gt^2) \equiv ist(C_{robot}, \ldots)$$

where $C_{physics}$ is the context introduced by a physics text, and C_{robot} is the context of the robot that was spoken of earlier. The equation from the robot's point of view has been left out for clarity.

4.5 Strategies

4.5.1 Programs and Situation Calculus

There is a major need for a formalism that can capture projected strategies. Much work has been done in proving that a sequence of actions achieves a goal. The most studied problem has been the **Yale Shooting Problem**. In this problem the following axioms are given,

$$alive(Fred, S_0) \wedge loaded(gun, S_0)$$

$$loaded(gun, s) \supset \neg alive(Fred, result(shot(Fred, s))$$

Another condition is added that things do not change unless they must. This can be added in a variety of ways as will be seen later. The query

$$alive(Fred, result(shot(Fred), result(wait, S_0)))$$

is then asked. This is to be false, Fred should be dead. The main reason it is of interest is that many non-monotonic formalisms have difficulty with it.

A more normal way to give a strategy is to give a sequence of actions. If the strategy that is to be described is,

> walk north down Main Street for five blocks or until you reach Chesnut Street.

A program can be written to describe it,

```
        n := 5;
a:      if value(street, s) = Chesnut ∨ n = 0
              then goto b;
        s := result(block − north, s);
        n := n + 1;
        goto  a;
b:
```

This separates physical variables from intellectual variables. The $s :=$ ... command is not a calculation it is an action. The command $n := n + 1$ is calculational. That is, n is a mental state, it is something that is used by the planner, that does not directly correspond to an action. The assignment of s is a physical action. Many strategies can be written in this form, with the division between mental and physical entities made explicit.

If it required that this strategy be proved successful, the methods of program proving[5] developed in the theory of programming can be successfully used.

There is a large difference between the treatment of situations s in the program paradigm, and the situation calculus paradigm. Relating this strategy based formalism to pure situation calculus may prove difficult.

[5] See work by McCarthy [7]

Chapter 5

Lecture 5:

5.1 Planning and Situations

5.1.1 Blocksworld

The problem of planning has often been examined in the domain of Blocksworld. In this domain the only items are blocks, and a table, and blocks can be either moved or painted. Blocks have two properties, a colour and a location. A typical blocksworld planning problem might be to find a sequence of actions that turns figure 1, into figure 2.

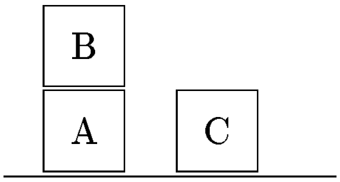


Fig.1

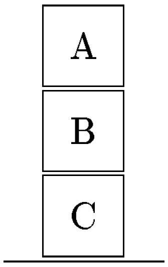


Fig.2

The plan can then be represented by the psuedo-axiom,

$$holds(loc\,A \simeq top\,B\ \underline{and}\ loc\,B \simeq top\,C\ \underline{and}\ loc\,C \simeq table,$$

$$result(move(A, top\,B), result(move(B, top\,C), result(move(C, table), S_0))))$$

It should be intuitively clear what is meant. If we want to talk about the result of a sequence of 3 actions we use the composition of functions[1]. In computer science terms we would make the following more compact representation.

$$result(move(C, table); move(B, top\ C); move(A, top\ C))$$

If a notation like this is to be used, this notation will have to be defined to be equivalent to the previous representation. The previous notation itself is only a shorthand for,

$$value(loc\ A, result(move(A, top\ B), result(move(B, top\ C), result(move(C, table), S_0)))) = top\ B$$
$$value(loc\ B, result(move(A, top\ B), result(move(B, top\ C), result(move(C, table), S_0)))) = top\ C$$
$$value(loc\ C, result(move(A, top\ B), result(move(B, top\ C), result(move(C, table), S_0)))) = table$$

This notation can be reduced by using lambda abstraction to give,

$$(\lambda s.value(loc\ A, s) = top\ B \wedge value(loc\ B, s) = top\ C \wedge value(loc\ C, s) = table)$$
$$(result(move(B, top\ C), result(move(C, table), S_0)))$$

It should be noted that what happens if the preconditions of an action are not met has not yet been defined. This can be defined explicitly, or some kind of nonmonotonic formalism can be used. The types of nonmonotonic formalisms that can do this will be discussed later.

[1] A more convenient notation has not been developed mainly because of the number of simple problems with a small number of actions. As more longer planning problems are considered a more compact representation will doubtlessly be developed.

5.1.2 Heuristics

The above problem can be solved by many planners that have been developed [5] [6]. All of the approaches used so far are computationally tedious. In the planning literature there are many different methods of approaching this but a method that has not been implemented is the use of declarative heuristics.

Suppose there is a collection of towers of blocks on a table, and the goal is to develop a plan that can re-arrange them into some other collection of towers. Heuristics similar to those used by humans can be written. This is in stark contrast to the heuristics that have been used so far by planner writers. The following heuristics are more human like in character,

> If a block is on the table and is on the table in the goal position it is in final position. If a block is on another block that is in final position, and is in the position it is in the goal state, it is on final position. A block is only in final position if the previous statements imply it. If a block is in final position do not move it.
>
> If any block can be moved to final position this should be done before any other type of move.
>
> If there is no block that can be moved to final position, and there is a block that is above a block it ought to be under, put it on the table.

This planning method is close to optimal.

These heuristics are not hard to build into computer programs, but as has been stressed before, it is difficult to give them as advice. As before, an **Advice Taker** that can take this advice needs to be developed.

5.1.3 Larger Ontologies

A *traffic light* can be defined to be a red block on a green block, on a yellow block, on top of two or more blocks or other colours. If the actions that exist include painting and moving blocks, and there exists enough blocks, a plan to build a *traffic light* can be developed.

In the formalism that has been developed so far a specific situation can be defined.

$$\exists a\ b\ c\ d\ e\ f\ .colour\ a = red \wedge colour\ b = yellow \wedge \ldots$$

$$on(a, b, s) \wedge on(b, c, s) \wedge \ldots$$

What are language does not allow us to do is to make a definition equivalent to our informal one. In the ontology that has been selected so far, there exists only, situation, blocks, actions, colours, and locations. We do not have a object sort called towers. Remember that the only *real* things that exists are those things that bound variables can range over[2].

There are many different ways in which towers can be added to our ontology. It is easy to add concrete towers, that is towers that are defined to be a certain set of blocks in a particular relationship. If we try to define a more abstract kind of tower we can define it as follows,

> The null tower is a tower, it is named by the (empty) location that it is at.
> Putting a block on a tower makes a tower.

This however locates towers. This is undesirable as reasoning about moving towers may later prove useful. A more abstract tower might be a sequence of colours, red/yellow/green/black/black/black. With this the action of building a *traffic* light can now be discussed.

5.2 Reasons for Formalisations

The reasons for the discussion of the problems of formalising in the previous discussion, is not driven by a need for robots that can build *traffic lights* in this restricted domain. The point of the section is the difficulties of formalisation in general. There should exist a system of

[2] If we were using a logic with sorts there would be a different sort for each type of object. Observing type restrictions does not however make a bad search program into a good one. A bad search program is one that runs in exponential time. Type restrictions will only reduce the level of exponential increase, and will not give a (good) polynomial time program

axiomatisation that would allows us to express ideas of this sort in a formal way.

Tests of this sort, building objects with bricks, are occasionally given to young children as a way of testing such things as I.Q. A five year old might be asked to construct a traffic light from a collection of bricks. Indeed most five year olds would have little difficulty with this problem if it were explained to them. Children of even this young age have the ability to consider abstract ideas such as *traffic lights* in this domain.

In asking a child to do this it is assumed that he has an abstract notion of what kind of a thing a *traffic light* is, and what properties are important, and which are not. Its location is not a defining characteristic, but the sequence of colours is. These abstract notions are difficult to inculcate into computers.

Chapter 6

Lecture 6:

6.1 Circumscription

6.1.1 Introduction

Circumscription was introduced as a formalism for nonmonotonic reasoning in 1977 in the paper[1]. It has since been extended and developed in [2] and [4]. Nonmonotonic reasoning, is reasoning with the property that a proof in a subset of axioms, may not carry forward to a larger set of axioms.

6.1.2 Reasons for Nonmonotonicity

As it has been stated already the Missionaries and Cannibals problem 2.2.4 is informal in nature. In solving the problem, given in this informal form, nonmonotonic reasoning is done. This reasoning is in the changing from the informal description to an Amarel model [3]. Amarel models were introduced by Saul Amarel, when he studied various different representation schemes. Two examples of these schemes are representing the problem as set of triplets, containing the number of cannibals and missionaries and boats on the first side of the river. This gives a search space of 32 states[1]. A second example os where

[1]However due to restrictions in the problem, such as the Missionaries outnumbering the Cannibals only 18 of these states are reachable.

each of the Missionaries and Cannibals is given a name. The search space with these names added now has 128 states.

There are of course more elaborate representations, which include the ability to represent how far across the river the boat is, and other non-important facts. The model with just the triple is the smallest model that allows the problem to be solved and can thus be considered the best.

This model is reached by using a bit of creativity. It is noticed that the solution must be invariant over permutations among the Missionaries and Cannibals. Therefore only their total number matters. This allows us to dispense with their names. It is not obvious when a counting argument like this can be used. In problems of this sort it is always easy to assign names, and treat each individual separately, when the reverse can be done, depends much more on the structure of the problem and the actions allowed.

The first step in solving this problem is to translate the statement of the problem into a more formal language. Pre-existing fluents that have been used for describing situations can be used, e.g. locations, situations. This language must be connected to the language in which the problem solver has its common sense facts formalised in.

There are many different facts that need to be formalised. It must be known that a river has two banks, and that these are places. However the banks of the Mississippi are several thousands of miles long. It is not obvious that if two things are on the same bank of the river that they are close together. It must be known that in problems like this bank of a river indicates a small place. This is a nonmonotonic assumption, both Cedar Falls, and Kansas City are on the same bank of the Mississippi, but we would not want a problem solver to consider that they both were in the same small place.

The common sense notion that the boat is to be used for crossing the river must also be expressed. This includes expressing facts such as that the boat moves with its contents to the other side of the river. In the representation scheme, or Amarel model that is chosen there will probably not be a representation of getting in and out of the boat, this must also be known. Another commonsense facts like that the boat will not go anywhere by itself.

Quite apart from getting the necessary information to develop this much of the Amarel model, there is also the other nonmonotonic part. This involves knowing that, unless stated otherwise there are no bridges, and that the boat is usable if all the stated conditions are met. Apart from the restrictions explicitly stated there are no others. If all the restrictions were dropped, there would not be a puzzle, however a simple answer that they all get in the boat and cross might be acceptable if this were part of a larger puzzle.

The nonmonotonicity of restrictions can also be seen from the previous treatment of the elaborated puzzle with the facts about oars added. The reasoning that brings us from the statement to the Amarel model is nonmonotonic as additional information can cause our model to change.

6.2 The Circumscription Method

6.2.1 Motivations

The original example of nonmonotonic reasoning was about the lack of a bridge. The idea of minimising objects was therefore the first attempted solution. All of the objects in the problem were minimised. However it worked out that it was better to minimise predicates. This can achieve the minimisation of objects as the predicate *present* is minimised in the Missionaries and Cannibals, or *ontable* is minimised in Blocksworld[27]. Therefore the notion that in a a problem there are as few objects as possible, compatible with commonsense, was arrived at.

The reservation, compatible with commonsense, is important, as if complete minimisation is used, there would be no water in the river, etc.

6.2.2 Notation

We introduce a concept of partial order on predicates. A predicate will be assumed to have arity one, for convenience.

$$p' \leq p \equiv_{def} \forall x . p'(x) \supset p(x)$$

That is whenever p' is true, p is true. What is being compared here is sets of objects. The notation of set theory would write this as

$$\{x|p'(x)\} \subset \{x|P(x)\}$$

The partial ordering is that of set inclusion. Set theory is more elaborate than second order logic, so for simplicity second order logic notation will be used.

The following relational operators are also defined,

$$p' < p \equiv_{def} (p' \leq p) \wedge (p' \neq p)$$
$$p' = p \equiv_{def} \forall x.p(x) \equiv p'(x)$$

With these relations two ways of defining the minimum suggest themselves.

$$A(p) \wedge \neg\exists p'(A(p') \wedge p' < p)$$

and,

$$A(p) \wedge \forall p'(A(p') \supset p \leq p')$$

The second is an absolute minimum, however it turns out that the concept of a relative minimum defined in the first is more useful.

A very simple example is the following: the domain is over { 0,1 }, and A is defined to be,

$$A(p) \equiv p(0) \vee p(1)$$

The four possibilities for p are,

$$\begin{array}{ll} p_1 & \{\ \} \\ p_2 & \{0\} \\ p_3 & \{1\} \\ p_4 & \{0,1\} \end{array}$$

Only p_2, p_3 and p_4 satisfy $A(p)$. Thus the relative minima are p_2 and p_3, there is no absolute minima.

6.2.3 Identities

The circumscription formula is defined to be,

$$A(p) \wedge \forall p'(A(p') \supset \neg(p' < p))$$

That is p satisfies a and all other predicates that are less than p do not satisfy A. The above formula can be rewritten to,

$$\begin{array}{lcl}
A(p) & \wedge & \forall p'(A(p') \supset \neg(p' < p)) \\
A(p) & \wedge & \forall p'(A(p') \supset \neg(p' \leq p \wedge p' \neq p)) \\
A(p) & \wedge & \forall p'(A(p') \supset \neg(p' \leq p) \wedge p' = p) \\
A(p) & \wedge & \forall p'(A(p') \supset [\neg\forall(p'(x) \supset p(x)) \vee \forall x(p(x) \equiv p'(x))]) \\
A(p) & \wedge & \forall p'(\neg A(p') \vee [\neg\forall x.(p'(x) \supset p(x)) \vee \forall x.(p(x) \equiv p'(x))]) \\
A(p) & \wedge & \forall p'.(A(p') \wedge \forall x.(p'(x) \supset p(x)) \supset \forall x(p(x) \equiv p'(x)))
\end{array}$$

This is the form that is given in [2], and is a useful form.

6.2.4 Conjunctions and Disjunctions

Consider the formula,

$$isblock\ A \wedge isblock\ B \wedge isblock\ C$$

If the predicate *isblock* is circumscribed with this formula, the circumscription formula gives,

$$isblock\ A \wedge isblock\ B \wedge isblock\ C \wedge [\forall p'.(p'\ A \wedge p'\ B \wedge p'\ C \wedge$$

$$\forall x.(p'(x) \supset isblock\ x) \supset \forall x(isblock\ x \equiv p'\ x))$$

If we substitute for p' ,

$$p' = \lambda x.(x = A \vee x = B \vee x = C)$$

we get,

$$\lambda x.(x = A \vee x = B \vee x = C)(A) \wedge \lambda x.(x = A \vee x = B \vee x = C)(B) \wedge$$

$$\lambda x.(x = A \vee x = B \vee x = C)(C) \wedge$$

$$\forall x.(\lambda x.x = A \vee x = B \vee x = C)(x) \supset isblock\ x$$
$$\supset \forall x.isblock\ x \equiv (x = A \vee x = B \vee x = C)$$

The right hand side reduces to true, giving,

$$\forall x.isblock\ x \equiv (x = A \vee x = B \vee x = C)$$

This can be seen to be a nonmonotonic conclusion by adding *isblock* D. This gives exactly the blocks A, B, C, and D. Thus the previous formula, which asserts there are only three blocks is no longer derivable. If this is viewed epistemologically, then is we are told that A, B and C exist we assume that is all that exists.

The previous example was circumscribing a conjunction. If we consider the disjunction,

$$isblock\ A \vee isblock\ B \equiv_{def} A(isblock)$$

then, if *isblock* is minimised, the answer that is needed is that one or other of A and B are blocks, but not both[2]. As before the circumscription formula is,

$$isblock(A) \vee isblock(B) \wedge [\forall p'.(p'(A) \vee p'(B) \wedge p'(C) \wedge$$
$$\forall x.(p'(x) \supset isblock(x)) \supset \forall x(isblock(x) \equiv p'(x)))$$

Here two substitutions are done for p',

$$p' = \lambda(x = A)$$
$$p' = \lambda(x = B)$$

These two substitution reduce to,

$$\forall x.(x = A \supset isblock\ x) \supset \forall x.(isblockx \equiv x = A)$$
$$\forall x.(x = B \supset isblock\ x) \supset \forall x.(isblockx \equiv x = B)$$

Now since the disjunction of the premisses was asserted originally, the disjunction of the conclusion can be derived, giving,

$$\forall x(isblock\ x \equiv x = A) \vee \forall x(isblock\ x \equiv x = B)$$

[2]Equality is not a problem in this example, as if A is a block and if $A = B$, then B is a block.

6.2.5 Circumscription and Linguistic Conventions

In puzzles there ar various types of linguistic conventions. Suppose someone is hired to build a bird cage. Directions are given about how high it is to be and other matters. When it is delivered the buyer complains that it has a roof, but it shouldn't have, because his bird is a penguin. A judge in this case will usually side with the maker. However if the reverse happens and someone builds a bird cage without a roof, and the buyer complains, the judge will side with the buyer. It is a convention of English that if one describes a bird one need not mention that it can fly. If no mention of flying is made, it is to be assumed that it can fly. If the bird cannot fly and this is relevant then this must be stated.

There are some people who believe that the reason this communication conventions was made in the direction it was, is because more birds can fly than cannot. This is not the same as a causal link.

Another communication convention is that if someone states that their son is less than 30 years old, they would be thought misleading if their son was actually 6. If someone was told that he was under 30, they would conclude that he was in his 20's. It is necessary in normal language to draw such conclusions, otherwise normal language would become legalise, full of codiciles and reservations.

Nonmonotonic reasoning has a variety of applications, perhaps the most straightforward is formalising communication conventions. A good example of this is the word 'but'. From the point of view of logical analysis, the word 'but' and the word 'and' are the same. Thus logically, the sentence

I am going home, but I am coming back.

means exactly the same thing as,

I am going home, and I am coming back.

However it blocks the hearer from making nonmonotonic inferences. When a speaker says " A but B" they mean that there is some inference that one might make from A that should not be made. Changing 'but' to 'and' does not change the ostensible meaning of the sentences, but because these signals to our nonmonotonic reasoning are removed, it makes it unreadable.

6.3 Induction and Missing Minima

The natural number are often defined in the following way,

> Zero is a natural number, the successor of any natural number is a natural number, and only those things which are natural number under the above definitions are natural numbers.

This seems very like a circumscription formula. It can be written as follows.

$$Isnatnum\ 0 \wedge \forall x.(Isnatnum\ x \supset Isnatnum\ succ\ x)$$

$$\forall \Theta.\Theta(0) \wedge (\forall x.\Theta(x) \supset \Theta(succ\ x))$$

$$\forall x.(\Theta(x) \supset Isnatnum\ x) \supset \forall x.(Isnatnum\ x \supset \Theta x)$$

This looks very like the axiom schema for natural numbers except it has an extra term. If we add

$$\Theta(x) \equiv \Psi(x) \wedge Isnatnum\ x$$

as we are only interested in the natural numbers, we get

$$\Psi(0) \wedge \forall x.(\Psi(x) \supset \Psi(succ\ x)) \supset \forall x.(Isnatnum\ x \supset \Psi(x))$$

This is almost the axiom schema for induction on natural numbers. This states that if it is true for 0 and if it is true for x it is true for the successor of x, then it is true for all the natural numbers.

The natural numbers are what you get if you start with zero, and use the successor relation.

However if we take the following slightly different axiomatisation.

$$\exists y.Isnatnum\ y \wedge \forall x.(Isnatnum\ x \supset isnatnum\ succ\ x)$$

$$\forall x.succ\ x \neq y \wedge \forall x\ z.succ\ x = succ\ z \supset x = z$$

If this is circumscribed, there is no minimal model. This was first discovered by Martin Davis in 1980. The circumscription formulas here gives a contradiction. This shows that a minimal model may not always exists for a given set of sentences.

6.3.1 On and Above

Above is the transitive closure of on. In general transitive closure is not expressible in first order logic. Using circumscription we can define above as the circumscription of the following sentences minimising the predicate *above*, while varying *on*.

$$\forall x\ y.on(x,y) \supset above(x,y)$$

$$\forall x\ y\ z.above(x,y) \wedge above(y,z) \supset above(x,z)$$

Instead of the second axiom we could use

$$\forall x\ y\ z.on(x,y) \wedge above(y,z) \supset above(x,z)$$

or

$$\forall x\ y\ z.above(x,y) \wedge on(y,z) \supset above(x,z)$$

They both give the same models but some theorem provers treat them differently.

6.4 Varying Other Predicates

One of the early criticisms of circumscription was that it never introduced a new positive occurence of a predicate. This and other problems were overcome by allowing other predicates to vary while minimising. This is written as,

$$A(p,z) \wedge \forall p'\ z'.A(p',z') \wedge (\forall x.(p'(x) \supset p(x))$$

This is written as $Circum(A;p;z)$ This more elaborate form of circumscription has much the same properties as the first.

6.5 Abnormality

The idea of abnormality has often been closely associated with non-monotonic reasoning. The idea is that, in general birds fly, and any that don't are abnormal in some regard. We give this abnormality a name ($aspect2$) and thus get a formula like,

$$\forall x.\neg ab\, aspect1 \;\supset \neg fly\ x$$

In general things don't fly.

$$\forall x.bird\ x \supset ab\ aspect1x$$

Birds are abnormal in that the previous statement does not hold. This does not mean that birds do fly. It merely blocks the line of reasoning that would allow us to conclude they would not fly.

$$\forall x.bird\ x \wedge \neg ab\ aspect2\ x \supset flies\ x$$

Birds, if they are not abnormal in this respect, fly.

$$\forall x.bird\ x \wedge \neg ab\ aspect2\ x \supset feathered\ x$$

Birds, if they are abnormal in another aspect, have feathers.

$$\forall x.penguin\ x \supset ab\ aspect2\ x$$

Penguins are abnormal in that the rule that in general birds fly can not be used.

$$\forall x.penguin\ x \wedge \neg abaspect4x \supset \neg flies\ x$$

In general penguins fly, unless they are abnormal in this aspect.

$$\forall x.penguin\ x \supset bird\ x$$

Penguins are birds. This is a monotonic rule. We also need unique names for aspects. That is that all aspects are different.

$$\forall x\ y.aspect1\ x \neq aspect2\ y$$

$$\forall xy.x \neq y \supset aspect1\ x \neq aspect1\ y$$

If we are given $bird(Tweety)$, and A is the conjunction of all the previous axioms, then,

$$Circum(A; ab; flies)$$

is a function of two predicates ab and $flies$, and we get the result that the things that cannot fly are exactly the birds excepting penguins.

Chapter 7

Lecture 7:

7.1 EKL

EKL is an interactive theorem checker. It can be programmed, as its commands are lisp functions. In principle it could be driven by a lisp program. No-one however has done this very much, perhaps no-one at all. This may be due to a lack of inspiration rather than a lack of need for this type of venture.

EKL is not quite like instrument flying, but if you have not used it for three hours in the last three months you probably need lessons again.

Its language is the functional calculus. That is, what we normally call predicate calculus in AI. EKL's name is slightly misleading, as its stands for first order logic in Finnish.

EKL announces itself by putting up a prompt,

```
:proof?
```

All of the commands it is given are given in lisp form, in parentheses, so the first line often is `(proof xxxx)`. EKL proofs can be typed directly into EKL or a file may be loaded. EKL is not a system that you once learn and use there after. A finger in the manual can be very useful to almost all users.

The following is an EKL proof of the existence of a unique upper bound. See EKL Manual.

We give EKL some axioms, normally we would have the axioms in a file. There are two ways to do his. One is to save it in EKL's internal file format, using `save-proof`, the other is to keep a file of the lisp commands.

An axiom command tells EKL that the following is an axiom and that it is true. EKL should therefore not complain unless it is ungrammatical.

7.1.1 Definitions

A definition allows you to define new terms in terms of old terms. Definitions can be used in place of axioms. EKL will check the new term has not been defined before and that it is a term that EKL can prove exists. Terms must be defined in terms of formulas that do not contain the terms themselves. This is contrary to the computer science notion of inductive definitions. If a theorem that does not contain any of the defined terms can be derived, there is a theorem of mathematical logic that says that it could have been proved without the introduction of the term.

7.1.2 Natural Deduction

EKL is a natural deduction style theorem prover. If it is desired to prove $p \supset q$, the standard approach is to assume p, then from this and the other axioms, derive q. The assumption is then discharged to give $p \supset q$. This is then introduced as a full theorem with no assumptions. In Hilbert style deduction, preconditions are always carried around, making formulas much bigger. Natural deduction earns this sobriquet because it was seen as more natural.

To carry out natural deduction style proofs two functions are needed, one that creats an assumption and one that discharges it. In EKL these are `assume` and `ci`. If a sentence depends on an assumptions the assumption's number is listed below the sentence in the dependency list.

7.1.3 Other Functions

Another function that is implemented in EKL is `trw` which given a sentence rewrites it using the standard rewriters and any other that are specified. It tries to simplify the formula, and returns te sentence that the simplified formula is true if and only if the original is true. Thus `trw` always gives a result.

7.2 Unintended Models

In the proof above (see EKL Manual), the intended interpretation is sets of real numbers. However, the axioms do not imply that this is what is meant. Any domain that obeyed the axioms could be the model. What the axioms actually tell EKL is that there is an object S, and there is a relation between this s and some objects called x's. Among the x's there is a relation, that is transitive. There is one particular x that the relation holds between it and all the other x's but the relation does not holds between any of the other x's and it.

The above might seem like pedantry, but it is important to keep in mind that as far as logic is concerned it does not matter what you meant, it only matters what you said. There may be many other interpretations that you did not think of, but if what you wanted to prove does not hold in every interpretation, then it will not be provable in logic.

It is quite common for people to bet caught by not writing enough axioms to specify the domain. Then, they wonder why they cannot prove what seems obvious. There is a very important theorem, established in Gödel's Ph.D. thesis,which says that if the sentence is true in all models then there is a proof of it. [1]

[1] He went on to show that there are always unintended models of arithmetic in any finite axiomatisation, and thus there are always some theorems os arithmetic that cannot be proven, as they holds in one model of the axioms but not in another. This is called Gödel's Incompleteness theorem.

7.3 Types, Sorts, and Variable Number of Arguments

EKL is a typed logic. It has a hierarchy of types, so it can handle many ordered logics. In Computer Science types usually refer to what logicians call sorts. The logicians types are also useful in Computer Science but the is no accepted name for them.

In EKL there is a ground type, which a term has if it refers to what is intuititively an object. Derived types can be made up from this ground type. The most important type is the set of functions. An example is the set of functions that maps ground onto ground. Cartesian product also allow new types. Besides ground which is the conventional symbol for terms that are not functions we have a type truthval which has two members — True and False.

Lisp has the embarrassing property of having functions with variable number of arguments. EKL has features to help this. The features are not fully adequate, as although axioms about these functions can be given, theorems about these many argument functions cannot be proved.

In EKL there is a concept of sorts, which corresponds to the idea of types in Computer Science. Variables have sorts. If we are axiomatising arithmetic we need numbers, but if we also are reasoning about fruit and vegetables we want to separate the facts about each. In standard unsorted logic we could write;

$$\forall x\ y. number(x) \wedge number(y) \supset x + y = y + x$$

If we use a sorted logic, we can give x and y sort *number*, and just write;

$$\forall x\ y. x + y = y + x$$

This is stored internally as above. New sorts that are the union or intersection of other sorts can be defined. EKL automatically deals with these sorts and keeps the theorems valid. Sort universal is the default sort.

Chapter 8

Lecture: 8

8.1 Problems in Reasoning about Change

There are three major problems that have arisen in formalising reasoning about change. We will examine these from the standpoint of the situation calculus. They are,

The Frame Problem

The Qualification Problem

The Ramification Problem

The ramification problem is that it is unreasonable to explicitly record all of the consequences of action. For example if we move a bookcase, all of the books inside move along with it. For any action there are essentially an infinite number of possible consequences that might occur, depending upon the details of the situation in which the action occurs.

There are four major approaches to dealing with the ramification problem. The first is the monotonic approach given in McCarthy and Hayes which involves *action axioms* and *frame axioms* indicating things that do not change. This involves giving axioms for each fluent in the domain and indicating whether or not it will change when an action is done. This is the type of formalisation given in exercise 1b part 4.

4. Derive that Fred is not alive and the gun is unloaded after a particular plan. Use the following axioms.

```
1.  (AXIOM "ALL S G.HOLDS(LOADED(G),RESULT(LOAD(G),S))")
```

```
   2.  (AXIOM "HOLDS(ALIVE(FRED),SO)")

   3.  (AXIOM "ALL P S.NOT HOLDS(P,S) IFF HOLDS(NEG(P),S)")

   4.  (AXIOM "ALL G F.NOT SHOOT(F)=LOAD(G)&NOT SHOOT(F)=WAIT&NOT
WAIT=LOAD(G)")

   5.  (AXIOM "ALL G F.NOT ALIVE(F)=LOADED(G)")

   6.  (AXIOM
"ALL A S P.HOLDS(P,S) IMPL
HOLDS(NEG(P),RESULT(A,S)) IFF
(ALL G.P=NEG(LOADED(G))&A=LOAD(G)) OR
(ALL G F.A=SHOOT(F)&(P=LOADED(G) OR P=ALIVE(FRED)))")

   7.  (AXIOM "ALL X Y.NOT ALIVE(X)=NEG(LOADED(Y))")

   8.  (AXIOM
"ALL A S P.HOLDS(P,S) IMPL
HOLDS(P,RESULT(A,S)) IFF
NOT (ALL G.P=NEG(LOADED(G))&A=LOAD(G)) OR
(ALL G F.A=SHOOT(F)&(P=LOADED(G) OR P=ALIVE(FRED)))")

   9.  (AXIOM
"ALL S G F.HOLDS(LOADED(G),S) IMPL
HOLDS(NEG(ALIVE(F)),RESULT(SHOOT(G),S))&
HOLDS(NEG(LOADED(G)),RESULT(SHOOT(G),S))")
```

This has two major problems a epistemic one and a computational one. The epistemic difficulty is that we must provide explicit frame axioms for every action and every relation of interest stating under what circumstances a change occurs. In general if there are a possible actions and r possible relations we have on the order of $a * r$ frame actions. These actions may be very complex.

The second problem is that it is computationally intractable if there are many facts in the database. Every fact in the database world model must be examined and proved that it changes/holds. This can be seen in the exercise 3 in assignment 1b. 3. Given the following axioms derive a plan that will put A on top of B and B on top of C, and C in location 1.

```
   1.  (AXIOM "ALL X.BLOCK(X) IFF X=A OR X=B OR X=C")

   2.  (AXIOM "NOT A=B&NOT A=C&NOT B=C")

   3.  (AXIOM "ALL X Y S.COLOUR(X,Y,RESULT(PAINT(X,Y,S)))")

   4.  (DEFINE CLEAR
"ALL X S.CLEAR(X,S) IFF (ALL Y.BLOCK(Y) IMPL NOT AT(Y,TOP(X),S))"
NIL)

   5.  (AXIOM "ALL X Y P S.NOT X=Y IMPL NOT (AT(X,P,S)&AT(Y,P,S))")

   6.  (ASSUME
"AT(A,LOC1,S0)&AT(B,LOC2,S0)&AT(C,LOC3,S0)&COLOUR(A,BLACK,S0)&
COLOUR(B,BLACK,S0)&COLOUR(C,BLACK,S0)")
Deps:  (6)

   7.  (AXIOM
"ALL X Y S.CLEAR(X,S)&CLEAR(Y,S)&BLOCK(Y)&BLOCK(X) IMPL
AT(X,TOP(Y),RESULT(MOVE(X,TOP(Y),S)))")

   8.  (AXIOM
"ALL X Y C1 C2 S.NOT X=Y&COLOUR(X,C1,S) IMPL
COLOUR(X,C1,RESULT(PAINT(Y,C2,S)))")

9.  (AXIOM
"ALL X Y.NOT X=Y IMPL
NOT TOP(X)=TOP(Y)&NOT TOP(X)=LOC1&NOT TOP(X)=LOC2&NOT TOP(X)=LOC3&
NOT LOC1=LOC2&NOT LOC2=LOC3&NOT LOC3=LOC1")

   10.  (ASSUME "CLEAR(A,S0)&CLEAR(B,S0)&CLEAR(C,S0)")
Deps:  (10)
```

```
   11.  (AXIOM
"ALL X Y L1 L2 S.NOT X=Y&AT(X,L1,S) IMPL AT(X,L1,RESULT(MOVE(Y,L2,S)))")

   12.  (AXIOM
"ALL Z X S Y.CLEAR(X,S)&NOT X=Y IMPL CLEAR(X,RESULT(MOVE(Z,TOP(Y),S)))")
```

The second method is the nonmonotonic method. This involves writing a single persistence axiom which says that relations remain unchanged if it is possible for them to remain so. This does not suffer from the epistemic difficulties, but suffers seriously from computational difficulties. These are due to the inherent problem of nonmonotonic formalisms and the necessity of looking at every fact in the database as before.

There also is the problem of multiple extensions which we will come to later.

A third method if the STRIPS approach [21]. This relies on the fact that the world does not change much from one situation to the next. STRIPS keeps a single model of the world and updates it as a result of actions. The STRIPS approach describes actions in terms of a list of preconditions, an add list and a delete list. The intention is that an action can be carries out if its preconditions hold. The result of an action is adding the facts that are on the add list and removing the facts that are on the delete list. This suffers from two problems, firstly it can not deal with inferred consequences, and secondly it suffers from unclear semantics. We shall address the first problem now.

Consider the following domain, due to Ginsberg[22]:

move(tv,bottom-shelf,floor) :precondition; on(tv,bottom-shelf)

add: on(tv,floor)

delete: on(tv, bottom-shelf)

This works but we cannot have a definition of move that will in general tell us if the room is stuffy after a move operation, for the constraints in this problem see the paper by Ginsberg..

The second problem with STRIPS is that its database contain only a very special set of sentences. Analysis by Lifschitz [23] shows that only universal formulas that always hold and ground terms (and their negations) can be in the database. Further more only a subset of possible ground terms may be in the database for binary relations. Lifschitz terms this set the *essential* set of formulas.

This is an increasingly restrictive assumption as the domain gets more complex.

This problem led to a fourth solution, expressed in Ginsberg's Possible World Approach, where STRIPS like add and delete lists are used with domain constraints, and the closest model subject to the domain constraints, to the answer STRIPS would give is chosen. This is computationally good, but has the problem that it cannot chose between different models, which do not have a subset relation between them. In essence it fails a simpler problem than the Yale Shooting Problem. It also lacks a clear epistemic basis.

A computationally feasible way of dealing with the ramification problem has not yet been found.

8.2 The Frame Problem

As in the ramification problem, there is the monotonic approach. This fails for the reasons given before.

Computationally bad

Epistemologically too many frame axioms needed.

The nonmonotonic approaches can be divided into three basic type. The simple theory, the chronological minimisation method, and the causal minimisation methods.

The simple method is as before to minimise change it fails to solve the Yale Shooting Problem.

8.3 Yale Shooting Problem

The Yale shooting problem [28] is the occurence of two or more models when only one should occur.

This has been addressed in several ways. The early methods by Shoham[25] and Lifschitz[24] approached it as a pure prediction problem. In prediction problems, problems where an initial state is given and conclusions are to be drawn, choosing the model where abnormality happens as late as possible is the correct answer. Choosing one model over another can be done in different ways. Both Lifschitz and Shoham used modification of previous non-monotonic logics to address the problem. Shoham introduced an ordering on models which chose the most chronologically ignorant. This is done in terms of possible worlds and Kripke semantics.

Lifschitz formalisation is a modification of circumscription. It defines a new type of circumscription which defines

$$Circum_P(A) \equiv A(P) \wedge \forall x \neg(p\ x \wedge A(\lambda y(p\ y \wedge x \neq y)))$$

This says that P is the minimised predicate if there is no x such that $P(x)$ is true that could be made false without violating $A(P)$. In essence it does minimisation one point at a time. It can in its simplest form get stuck at local minima, for instance there are two models of the formula $ab(x) \wedge ab(y)$ if $x \neq y$. The model where both are true and the model where both are false. However this can be overcome by various tricks as:

$$Circum(A \wedge \forall x(P\ x \supset Q\ x); Q; P, Z)$$

is equivalent to

$$Circum(A; P; Z) \wedge \forall\ x(P\ x \equiv Q\ x)$$

Because we have a formalism that explicitly refers to minimising at points we can express the fact we wish to minimise at one point rather than another. This allows us to chose to minimise at earlier points in time preferentially. This is done by adding a term into the second part. When the term is true the point is minimised.

This allows great flexibility in terms of circumscription strategy.

8.3.1 Causal Minimisation

The previous chronological ignorance methods succeeded in solving the predication problem, but fail in more general problem where explanation is needed. In the light of this Lifschitz and Haugh separately developed causal minimisation which was described in earlier lectures. It should be noticed that this approach only partially addresses the qualification problem, and does not address the ramification problem. McDermott argues that this approach is invalidated by its unintuitive nature – one of the principle arguments in favour of the nonmonotonic a solution to the frame problem is that it makes formal sense of an intuitively satisfying description of nature.

All of the above methods share the problem that in general they are computationally intractable.

8.4 Criticisms of Situation Calculus

Several criticisms of the situation calculus have been made. It has been claimed that its primitive ontology does not allow the representation or solution of many types of common reasoning. Examples of this are:

1. Concurrency
2. Actions which return the original state.
3. Incomplete information, partial ordering of actions.
4. Reasoning about duration and time.
5. Delayed effects.
6. Natural death
7. Divisible actions
8. Continuous processes
9. Overlapping actions.

It can be shown[8] that all the problems except for 1 2 and 9 which deal with concurrency can easily be dealt with. Concurrency is still a problem but there are some ways for dealing with it.

2. Actions which return the original state, examples are running around a track or waiting, almost always change something, if nothing else time.

3. Incomplete information can be dealt with by quantification, partial ordering by disjunctions.

4. Reasoning about time and duration can be dealt with by introducing a fluent *Time* and a function *duration* of an action.

5. Delayed effects can be dealt with using these time operators.

6. Natural death is in effect the same problem as 5.

7. Divisible action can be accommodated by introducing functions that decompose and join other actions. $a;b$ joins actions a and b, $head(a,t)$ gives the action of doing a for t seconds, and $tail$ its remainder. This allows us to divide actions.

8. Continuous process can be dealt with by quantifying over all divisions of actions, so that concepts of speed, and constant motion.

8.4.1 Concurrent Actions

The problems with concurrent actions are that merely adding causes together is unsatisfactory, as the actions may not start simultaneously so that the fluents that one action changes are undefined. One action may block another. Two actions done separately often have different effects than done simultaneously. The problem of needing to keep all sets of concurrent actions synchronised has also been raised. Cancellation, and conflicting subactions have been axiomatised, but the problem of one action succeeding when the other fails has not been clarified. Unknown actions done concurrently have also been addressed, and seem problematic. There are two basic methods. One considers an set of concurrent actions a which is minimised, the other is driven by minimising change in the world. These may turn out to be equivalent.

Concurrent actions pose difficulties for most solutions of the frame problem. In particular STRIPS or possible worlds approaches have great difficulty.

Chapter 9

Lecture: 9

9.1 Applications of Nonmonotonicity

The goal of this research is to make a straightforward system in which any collection of facts can be formalised. It should also follow in the system that some actions will achieve their goal, some steps will not achieve their goal, and the outcome of some, is undetermined.

The Yale Shooting Problem(YSP) is now a classical problem of non-monotonic formalisms. In the paper [16] the notion of a simple abnormality theory was introduced. A simple abnormality theory has the following characteristics, There is a simple predicate *ab*, which is minimised, and all other predicates are varied.

If this worked, it would be very advantageous. It would mean that all that was needed was to minimise one predicate, and that would capture all the non-monotonicity that we wanted. Simple abnormality theories are quite close to logic programs, and they can be converted to logic programs under quite general conditions. This make them attractive as they are thus easily computable.

The Yale Shooting Problem is one of the symptoms of the failure of simple abnormality theories. Though research has not finished analysing the usefulness of this type of theory it seems unlikely that it will achieve the necessary results. We would like to be able to write as axioms the facts involved, and the preference policies. Unfortunately some additional conditions need to be imposed. If you have a specific problem in mind, such as the YSP, there are ways to solve it, but in the general case the problems of this approach have not been overcome.

Some approaches such as chronological minimisation sometimes express our intuition. In particular chronological minimisation seems to work for the prediction problem. However for explaining facts it seems to work less intuitively. For instance if we had a loaded gun and waited and waited and shot, and also the fact that fred was alive eventually, chronological minimisation would give the result that the gun became unloaded at the last possible time. This does not agree with our intuitions.

Sometimes we wish to use actions to reason backwards in time. We can consider the following example. The example comes from geology. We observe an intrusion and a dyke. If we see the following;

picture missing

we conclude that an earthquake happened first. If we saw the following;

picture missing

We would conclude that the intrusion happened first. We would like our general formalisation to capture this form of past reasoning as well as purely projective reasoning.

9.2 Deriving Default Rules

The next skeleton in the closet is that nonmonotonic formalisms in general cannot derive new instances of their default rules. This affects circumscription, but not nearly as badly as it affects the other nonmonotonic formalisms. In this way simple abnormality theory is a win because syntactically its formulas are the same. Perhaps a way could be developed to infer default rules. In default logic[29], the rules are not logical sentences, and thus cannot be inferred. When we get round to inferring normalities from the computer's experience we will need this quality.

9.3 Attempts to Circumscribe Equality

Suppose the Missionaries and Cannibals problem is presented. We are told that there are three missionaries and three cannibals. How do we know that some of the missionaries are not cannibals. If we are told that there are three blocks on the table, A B and C we assume that they are different. If we assert $ontable\ A \wedge ontable\ B \wedge ontable\ C$ we want to assume that they are different. It is difficult to express this as a nonmonotonic convention. An example of the problems associated with this is the following; a character could be introduced in a detective story under one name, we also might have lots of reasoning about the identity of the murderer. We want to be able to relate the two names.

We could try to address this problem by circumscribing equality;

$$A(=; Z) \wedge \forall e'(A(e', z) \supset \neg(e' <=)$$

$e' < e$ implies that there are some things which do not satisfy e' but which are equal, i.e.:

$$\neg e'(x, y) \wedge x = y$$

But at the same time we want e' to satisfy the definition of equality

$$\forall x.e'(x, x)$$

$$\forall x\ y.e'(x, y) \supset e'(y, x)$$

$$\forall x\ y\ z.e'(x, y) \wedge e'(y, z) \supset e'(x, z)$$

$$e'(x, y) \supset (P(x) \equiv P(y))$$

It can be seen from the above that our attempts to circumscribe equality has failed.

However if we introduce names as objects, and what they denote by a function we can overcome this. We have A, B and C as names with uniqueness,

$$A \neq B \wedge B \neq C \wedge A \neq C$$

$denote(x)$ is a function, which returns the object that a name denotes. Now if we have

$$isblock\ denote\ A)\ \wedge isblock\ denote\ B \wedge isblock\ denote\ C$$

we have put a distance between the symbol and the block itself. We now introduce a predicate $eqdenote(x, y)$. We can circumscribe *eqdenote* without the earlier problem.

We get out of the difficulty at the cost of introducing a new piece of machinery. All the constants get doubled. To assume that things are not equal one has to double size of the domain, as names have to be introduced as separate objects. This is described in [16], this paper recommends that everything be reformulated in this way. It should be noticed that Prolog takes uniques names as an axiom. Pat Landon called this syntactic sugar. Perhaps equality should be written with a slight twiddle to show that we are actually using *eqdenote*.

Chapter 10

Lecture: 10

10.1 Auto-epestemic Logic

10.1.1 Introduction

The main introduction of non-monotonic logic was in a special issue of the J.A.C.M. This introduced non-monotonic logic I, circumsciprion and default logic. Auto-epestemic logic (AE) was introduced in I.J.C.A.I. in '83, and in A.I.J. in '85. AE, circumscription and default logic are the three surviving non-monotonic logics.

The easiest way to explain auto-epestemic logic is from a historical perspective. In McDermott's and Doyle's Non-monotonic logic I, they pointed out some strange and undesirable properties their logic had. McDermott tried to correct these in non-monotonic logic II.

AE was developed in response to this. The original NMI was an extension to propositional logic by adding a modal operator. The modal operator was called M and was read informally as 'is consistent'. Therefore we have rules of the sort

$$\forall x. bird\ x. \wedge M\ flies\ x \supset flies\ x$$

There was a single non-monotonic rule of inference. That was that $M\ p$ is a theorem if $\neg p$ was not a theorem.

In exploring the consequences of this approach some problems arose. The main problem was that it was much weaker than intended. It was possible to have $M\ p$ and $\neg p$ in the same system, so that M was a weaker notion than consistency.

McDermott tried to base non-monotonicity more fully on modal lgics. He tried to use the well studied systems T, $S4$, and $S5$, looking for intuititve semantics. He decided that $S5$ had the correct semantics, but found out that non-monotonic $S5$ was equivalent to monotonic $S5$.

About 1983 Bob Moore started studying the logics, looking for what exactly the formulas said. McDermott considered the formula above to mean that most birds fly, while Moore felt it should be all birds can fly except those that are believed not to fly.

Auto-epestemic logic ia a reconstruction of non-monotonic logic as a logic of reasoning about one's own beliefs. We have a modal operator M and its dual L. We can imagine $L\ p$ as p is believed. As usual $M\ p$ is defined to be $\neg L \neg p$. The semantics for $M\ p$ is not p is not believed. This is close to p is consistent.

10.2 The Semantics of AE

We say that an AE thoery is intended to be the model of the beliefs of a rational agent that can introspect on its beliefs. We have $L\ p$ is true relative to a theory T iff $p \in T$.

We take the theory initially to be a unstructured set of formulas. An auto-epestemic interpretation of a theory T is an interpretation of T in which for all p, $L\ p$ is true iff $p \in T$. Thus it is an ordinary interpreation with a constraint. An AE model is just a model with this constraint.

If we start with ordinary semantics for propositional logic, and the operator L, and we generate all interpretations, we can then remove all the interpreations that do not satisy the constraints. This gives us the interpretations we want.

10.2.1 Soundness and Completeness

When one develops a logic the most important things are completeness and soundness. We say that T is semantically complete if it contains every formula that is true in every auto-epestemic model of T. We can view this as, given everything that you have got, then there is nothing that must be true that has been left out. If you look at all models and find a formula that's true in all models, then it is a consequence of the theory and should eb added.

T is sound with respect to A if and only if every auto-epestemic interpretation of T that is a model of A is a model of T. We can consider A to consist of premisses, they are guarunteed to be true. The content of the rest of T is open. We thus only need to look at the auto-epestemic parts. If we take every AE interpretation that is a model of A, and we have the property that everything in T is modelled by this then we have achieved our goal.

Soundness and completeness are semantic properties. They are not defined with a notion of derivation. Given a notion of truth, we can ask what syntactic properties can we use to syntacticallly derive just the true premises. In almost all non-monotonic logics derivation is difficult. If you wish to draw a conclusion subject to a constraint, you do not use an iterative process, you chose different values until the constraint is satisfied.

10.3 Stability

An AE theory is stable if :

1: Closed under ordinary logic.

$$p_1, \ldots, p_n \in T \;\; and \;\; p_1, \ldots, p_n \vdash Q \; then \; Q \in T$$

2: If $p \in T$ then $L\ p \in T$. If you believe p then you should believe that you believe it.

3: If $p \notin T$ then $\neg L\ p \in T$. The non-monotonic property.

Theorem: An auto-epestemic theory is semantically complete iff it is stable.

We gave a semantic definition of completeness and a syntactic definition of stability, and this theorem relates them. The syntactic condition that relates completeness is stability, for soundness we have groundedness.

An AE theory is grounded in a set of premises A iff for all formulas $Q \in T$

$$A \cup \{L\ p | p \in T\} \cup \{\neg L\ p | p \notin T\} \vdash Q$$

If a theory satisfies stability then the theory is sound and complete if it is grounded. What we want to do is combine stability and groundedness. The above formula does this. It should be noted that stable expansions are not defiend by an iterative process. We are not guarunteed a unique expansion, nor for that matter any expansion at all. We have examples of each abnormal case.

1: Not always unique:

$$\{\neg L\ p \supset Q,\ \neg L\ Q \supset P\}$$

We have symmetry between the P and Q. Thus we have the exapansion containing P and the expansion comtaining Q.

2: Do not always exist:

$$\{\neg L\ P \supset P\}$$

Here we have no expansion.

There is no stable set of beliefs that can be derived from just this premiss. This relates to the puzzle of the unexpected execution. This tells of a prisoner who was told that they would be killed in a week but would not know the day. The puzzle is explained by this case of paradox, that is a formula that is true but cannot be used to ground its own beliefs[30].

10.4 Relation to McDermott and Doyle Logic

We have not yet addressed, the question of how this relates to McDermott's and Doyle's logic. They defined the non-monotonic fix-point T of a theory A to be

$$T = \{Q | A \cup \{\neg L\ p | p \notin T\} \vdash Q\}$$

This contrasts with the definition of AE in that it lacks the term, $L\ p \in T \equiv p \in T$. This explains the problems they observed in NMI. In their logic is is not inconsistent to have $M\ p$ and $\neg p$. If that missing constraint is added the undesirable feature disappears.

NMI also fails to enforce $p \vdash L\ p$, otherwise $\neg p$ would be inconsistent with $M\ p$

McDermott realised that NMI was too weak. He tried to strengthen it by adding more modal rules. He thought that more modality would work. He changed the fix-point to

$$T = \{Q | A \cup \{\neg L\ p | p \notin T\} \vdash_{modal} Q\}$$

Where $\vdash_{modal}$ is logical inference with $p \vdash L\ p$ as an inference rule. The difference are rather subtle, but having $p \vdash L\ p$ as an inference rule rather as a set is difficult.

McDermott bases his modal theory on the the axioms of the standard modal logics K, $S4$, and $S5$. These axioms were to represent beliefs about auto-epestemic reasoning. This is having beliefs about your own reasoning.

$$K:\ L(p \supset q) \supset (L\ p \supset L\ q)$$

Belief is closed under logical consequence, and you believe you work that way. That is not only is it true, but you believe it is true.

$$S4:\ L\ p \supset L\ L\ p$$

$$S5:\ \neg L\ p \supset L \neg L\ p$$

S4 is a statement of the second condition in the fix point of AE as T is of the first. Since all these axioms formalise stability conditions it should be harmless to add them. McDermott also did somthing else, he added the axiom schema, $L\ p \supset p$. None of the constraints we have had said anything about beliefs being true. Howver this is even stronger than that. It says that if you believe some-thing then it becomes true. There is really no reason to allow this.

Non-monotonic S5 will licence any belief what so ever. If you have this premiss, then any of you beliefs are necessarily true. This gets rid of the non-monotonicity. This allows completely circular justifications. There is thus no reason for this axiom.

First, we can demonstrate that having this axiom justifies belief in anything. Suppose:

$$P \in T \quad \wedge \neg L\ P \in T$$

Then

$$\neg L \neg L\ P \in T$$

$$\neg L \neg L\ P \vdash P$$

From the fact that I believe p I can derive that I believe it. From non-monotonic S5 you can derive anything.

Another issue is the notion of theorem in NMII. McDermott noticed that there could be more or less than one fix-point. They defined theoremhood as the intersection of all fix-points. Because of the self-re-enforcing there will be no formulas that are absent from every fix-point. Therefore there are no formulas of the form $M\ p$ that are present in every such fix-point. There are no theorems of the form $M\ p$ in any NM logic based on S5.

They key differences between AE and NMII are the extra clause in the fix-point, and the addition of p , $L\ p$. Looking at this in another way, we create a logic based on NMII but using weak S5, i.e. S4.5. We calll this K45. This buys us nothing, as if A is any set of premises, and p is any axioms of T, $S4$, $S5$, other than $L\ p \supset p$, then $A \cup \{p\}$ has exactly the same stable expansions as A.

McDermott mis-diagnosed the problem with NMII. He thought the problem was connecting provability with truth. The real problem was that provability (or belief) was only partially defined. Relating truth and provability was mistaken, the correct thing was to complete the definition of provability and keep no connection with truth.

10.4.1 Negation as Failure

AE is known to be decidable. There are some connections with AE and negation as failure, but adding these premises as theorems probably would not help to make this clearer. If you add as axioms sentences that are derivable as theorems it usually makes things worse from a computational view point as you have a bigger space to get lost in.

10.4.2 Extensions to Predicate Logic

There really isn't any good extension of AE to predicate logic We can add free variables, but there remain problems about the meaning of quantifying into a modal context. $\exists x.L(P(x))$ The problem has been addressed by Quine, Kaplan, and Kripke. It is a thorny issue. The classic example is given in Quine's "Quantifiers and Propositional Attitudes".

> Ralph believes the man in the raincoat is a spy. The man in the raincoat is Ortcutt, a fine upstanding citizen, so Ralph does not believe that Ortcutt is a spy.

Therefore there is a difficulty about what $\exists$ means.

10.4.3 Applying AE to Common Sense?

Anyone who uses a logic programming tool is using AE, but in the intended sense of the title, very few actually realise it. Negation as failure is the right implementation of introspection. Normally it is only used for Horne theories. There is a need for AE with negation as failure implemented on top of a full first order system.

Chapter 11

Lecture: 11

11.1 Beliefs

What we want is to be able to reason about is our own and other peoples knowledge. The conventional thing to do (Hintikka 1962) [17] is to use modal logic.

Modal logic was invented by Lewis [18] in 1920's. Hw was interested in the paradoxes of material implication, especially those of the form

$$p \supset q, when\ p\ is\ false$$

This is always true if we only consider the truth values of p and q. This is not always the case. Consider the sentence

> Rome was not built in a day implies that Bush was elected President in 1988.

As there is no connection between the clauses, we would like this to be false. However normal logic says that this is true. To avoid these unintuititive results Lewis introduced the "Calculi of Strict Implication".

$$p \succ q$$

which was to mean p strictly implies q. This depends not only on the truth values of p and a but on other things.

Lewis later simplified the notion by introducing a simple one place connective with conventional implication.

$$\Box(p \supset q)$$

The symbol $\Box$ is read as necessarily. Strict implication now corresponds to a necessary implication. Whether or not a proposition is necessary does not depend on the truth value of it. Another connective is introduced, defined in terms of the $\Box$. $\Diamond$ is defined to be:

$$\Diamond p \equiv \neg\Box\neg p$$

$\Diamond$ is read as possible.

For some reasons logicians use L and K and M and N, for these symbols. Here we will use N for necessarily. People quickly got into disputes about the axioms characterising these symbols. The most commonly agreed axiom is probably T,

$$N\ p \supset p$$

if something is necessary then it is true.

Another axiom is K

$$[N\ p \wedge N\ (p \supset q)] \supset N\ q$$

that is modus ponens preserves necessarily. Also, we want to have the fact that all the tautologies of propositional logic are necessary. If p can be proved from the axioms of propositional logic, we can infer, $N\ p$. This inference rule is called necessitation. If you can prove p from nothing at all, then you can assume $N\ p$.. You cannot write down any contingent axioms in the proof if you wish to use this rule.

We also can have 4

$$N\ p \supset N\ N\ p$$

this says that that which is necessary is necessarily necessary. These three axioms with the rule of necessiatation you have $S4$. If we add the rule 5 that

$$\neg N\ p \supset N\ \neg N\ p$$

the you have the system $S5$.

In the 1930's people argued about which was the correct notion of necessarily. These disputes remained unresolved, and people tend to study these logics now, without arguing which is the one true modal logic.

11.1.1 Kripke Semantics

In the 1950's Saul Kripke came up with a semantics for modal logic. He imagined that there were a set of possible worlds and a set of base letters p_1 to p_n. In each world there were a set of truth values for the p's. An accessibility relation on possible worlds, was defined. A proposition was considered necessary if it was true in all accessible worlds. This gives a system weaker than the system K, T. If we add that each vertex is accesible to itself we have T.

This seemed very attractive as it gave meaning in terms of structures. The interpretation of accessibility is that if you are in a world then any accessible world is possible. You do not know which of the worlds that are accessible is the real world, but you know that one of the accessible worlds is the real world.

If we add 4, that corresponds to the transitive closure of the accessibility relation. 5 implies that accessibility is a transitive relation.

11.2 Knowledge

Knowledge is like necessity but with a parameter for the person added to the accessibility relation. We will write knowledge ass a binary operator.

$$K_n\ p \quad or \quad s\ *\ p$$

We need the parameter as we can have several different agents. This can be seen in the puzzle of the three wise men.

> A King wished to find out which of his three wise men was the wisest. To do this he set them the following puzzle. He put a white spot on each of their foreheads and told them that he had put either a black or a white spot on their foreheads. He told them that at least one spot was

> white. They could see the colour of the other wise men's spots but not their own. Eventually, the wisest wise man said, "My spot is white".

The reasoning the wisest wise man did is as following; suppose my spot were black, then the second wise man would say, suppose my spot were black, then the third would know that, as he saw two black spots his spot must be white. But the second wise man knows that the third has not reasoned this out, so he must know that the third does not see two black spots. Therefore if I had a black spot he would know that he did not. As he has not spoken, he must not know I have a black spot, therefore my spot must be white.

The trouble with the problem in its initial form is that it deals with the speed of reasoning. To simplify this we will ask them all the question, "what color is your spot", at the same time. The first two times we ask the question they will say "I don't know", but the last time they will say, "My spot is white". The reasoning is as above.

11.2.1 A Politically Incorrect Puzzle

Another puzzle concerns the Caliph of Bagdad. He issues a statement that there is too much infidelity in Bagdad. He says that he wishes every man whose wife is unfaithful to kill her at dawn in the marketplace. It so happens that everyone in Bagdad gossips so everyone knows whether another man's wife is unfaithful, but of course no-one would tell the man himself that his wife was unfaithful. No-one is killed on the first thirty nine days, but on the fourtieth, the fourty husbands in Bagdad whose wives were unfaithful, execute the Caliph's order.

Their reasoning is as follows, suppose my wife were faithful. Then as I know there are 39 unfaithful wives, there must be some man who only knows of 38 wives, but he would have carried out this reasoning yesterday, saying, suppose my wife were fateful, ... Thus there must be fourty unfaithful wives, thus my wife must be unfaithful.

The trouble with this reasoning is the "...". Not only are the arabs smart, they can do mathematical induction. In principle the same method would work if there were 10,000 wives involved it would just take longer.

11.2.2 Mr. S and Mr. P

Another puzzle involves a King, who only has two wise men. We will call them Mr. S and Mr. P. He chooses two number between 2 and 99 and he tells Mr. S the sum, and Mr. P the product. The following exchange takes place.

Mr. P: I don't know the numbers.
Mr. S: I knew you didn't know. I don't know either.
Mr. P: Now I know.
Mr. S: So do I.

We can assume that the wise men told the truth, as Kings have a way of dealing with unethical wise men. Assuming this, what were the numbers. We will formalise this puzzle, but first we need some logical tools.

11.3 Logical Apparatus

We will use $s*p$ for s knows p. We will avoid using a necessarily operator by introducing an individual called any fool, who will represent as o. He will know knowledge that everyone knows. We need the following rules

$$s * p \supset p$$

If some-one knows something then it is true.

$$o * ((s * p) \supset p)$$

Any fool know that the above is true.

$$o * ((o * p) \supset o * s * p)$$

Any fool knows that if any fool knows some thing then everyone knows it.

$$o * (s * p \wedge s * (p \supset q) \supset s * q)$$

Any fool knows that anyone can do modus ponens.

Positive and negative introspection, 4 and 5 are also added, and any fool knows these are true.

$$o * (s * p \supset s * s * p)$$

$$o * (\neg s * p \supset s * (\neg s * p))$$

The last rule is undesirable in some applications.

We can now do the wise men problem. Our solution will be unfair however, as we will ask each man in turn. We need three individuals, s_1, s_2 and s_3. The propositions that they have a white spot will be p_1, p_2, and p_3. We have as an axiom,

$$p_1 \wedge p_2 \wedge p_3$$

As we already know the answer. The Kings announcement that there is at least one spot is captured by,

$$o * (p_1 \vee p_2 \vee p_3)$$

We now want another piece of syntax. We will use $s\$p$ to mean, s knows whether p. This is captured by,

$$s\$p \equiv s * p \vee s * \neg p$$

We know need to say that they each can see the others spot.

$$o * (s_1\$p_2 \wedge s_1\$p_3 \ldots s_3\$p_2)$$

We can formalise that the king asks the first wise man the colour of his spot as;

$$o * (\neg s_1\$p_1)$$

Unfortunately, this is too strong. This introduces inconsistency, as if s_1 knows that the others know that he doesn't know the colour of his spot then he can do the same reasoning. We avoid this by using;

$$s_3 * s_2 * (\neg s_1\$p_1)$$

and

$$s_3 * (\neg s_2\$p_2)$$

We can know prove $s_3 * p_3$

This formalisation is unsatisfactory in the following respects. We should be able to infer what they answer, and we should not need to give as axioms what s_1 and s_2 give as there answers. This formalism is not strong enough to express what they do not know. There is no way of saying that that is all the knowledge available. We do not yet have an answer for this problem.

Another problem is that the formalisation does allow direct knoweledge of chronology. This seems as if it could be solved by adding some kind of time parameter.

11.3.1 Negative Introspective

We can ask the question of whether George Bush knows whether Yeltsin os standing or sitting at the moment. An intelligent system should jump to the conclusion that he does not know. However this is difficult to justify logically, as Bush ,might know that Yeltsin always goes to bed early, and reason from the time that it is in Russia that he is in bed. We want a system that can make inferences about lack of knowledge, without getting tied up in proving things like the above do not apply.

11.4 Concepts and Propositions

> "it seems that hardly anyboby proposes to use different variables for propositions and truth-values, or different variables for individuals and individual concepts."
>
> Carnap, 1956, p 113 [26]

In this section we differentiate between concepts, and the things they refer to. We will use lower case to refer to things and upper case to refer to the concept of the thing. We will use $mike$ to refer to the person, and $Mike$, to refer to the concept of the person. We extend this notation to functions so that $telephone(mike)$ is the thing, and $Telephone(Mike)$ is the concept of Mike's telephone number . Relaxations of this notation are possible an will be pointed out when we break our own rules. In this notation there is no way to have concepts of concepts, later we will introduce them by doubling the first letter [1].

We wish to say that Pat know Mike's telephone number. We can express this as ;

$$know(pat, Telephone\ Mike)$$

That is that Pat knows the concept of Mike's telephone number. We need to avoid that fact that we do not want substitution for equals. If Mike's telephone number is 765-7867 we do not want to derive that Pat know's Mike's telephone number from the fact he knows that number. However we do want,

$$dials(pat, telephone\ mike) \equiv dials(pat, 7657876)$$

One of the characteristics of this language is that we have unrestricted substitution for equals. That is if Mike's telephone number is the same as Mary's then we have

$$dials(pat, telephoone\ mike) \equiv dials(pat, telephone\ mary)$$

But we do not have,

$$know(pat, Telephone\ Mike) \equiv know(pat, Telephone\ Mary)$$

This is because

$$Telephone\ Mike \neq Telephone\ Mary$$

They are different concepts, so we can not make the substitution.

11.4.1 Relating Concepts to Things

We do need to relate the concepts to the things that they represent. For this reason we have the function *denot*. *denot*, takes a concept, and maps it to the object the concept refers to.

$$denot(Mike) = mike$$

$$denot(Mary) = mary$$

[1]This is acceptable as long as we do not need the concept of an Aardvark

We also need to be able to map functions on concepts to functions on objects so;

$$denot(Telephone\ X) = telephone\ denot(X)$$

Using this we can reason as follows:

$$denot(Telephone\ Mike) = telephone\ denot(Mike)$$

$$denot(Telephone\ Mike) = telephone\ mike$$

$$denot(Telephone\ Mike) = telephone\ mary$$

but we cannot get Pat knows Mary's telephone number.

In the paper [19], *Know* is a concept so we have a function *true* which maps concepts of propositions onto their truth value.

Chapter 12

Lecture: 12

12.1 Mr. S and Mr. P again

In the problem, we saw that two numbers between 2 and 99 were chosen. As Mr. P said he didn't know what they were, Mr. S knew that they were not two primes. If they were both primes then Mr. S would have been able to deduce what the two numbers were from their product. Mr. S said that he already knew this. Therefore the sum must not be the sum of two primes. This means it could not be, even, by Goldbach's Conjecture, and also can not be an odd number with a prime two less than it.

This can be better seen by examining the problem, using the actual numbers that must have been given. The only numbers that work, are 4 and 13. When Mr. P says he doesn't know, he means that since he knows the sum is 52, he knows that the two possibilities are 4 and 13, or 2 and 26. When he is told that Mr. S knew that he didn't know, he is told that the sum is not the sum of two primes. 28 is the sum of 17 and 11, and thus the answer must be 14 and 3. Therefore, he now knows. The reasoning Mr. S needs to go through is similiar but more complicated.

SO far, no-one has succeeded in solving this problem using modal logic, because of the difficulty in axiomatising lack of knowledge. It has however been addressed using Kripke accessibility relation.

Chapter 13

Lecture: 13

13.1 Knowing That, and Knowing What

We will use slightly different notation than in the paper, [19]. We will write *knows* where in the paper $true(Know)$ was used. It is questionable as to whether this extra level was needed. When we say that someone knows that concept of a telephone number perhaps we mean that he know that concept of the concept of the telephone number.

We can define *exists* y the following axiom.

$$true\ Exists\ X \equiv \exists x.denotes(X, x)$$

or

$$exists\ X \equiv \exists x.denotes(X, x)$$

This is an area where there has been confusion and misguided attempts to straighten it out. Existence is not a predicate of individuals, but it is a predicate of concepts. Logic books stress the first, but ignore the second. This usage corresponds to the ordinary language usage. When people ask about the existence of God, they are asking whether the concept has a reference.

A classic example of this is;

$$true\ IshorsE\ Pegasus$$

The capitalisation of the E is to stress that the function takes a concept as its argument. Therefore we have;

$$ishorsE\ Pegasus \wedge \neg exists\ Pegasus$$

This gives the anticipated result, that Pegasus is a non-existant horse.

It is important to remember that these predicates are for our convenience. The only problem is adding too many axioms, as this might result in inconsistency. As long as the system remains consistent, and seems intuitive and useful, it is correct.

We, in general, have the rule;

$$denotes(A, a) \supset isX\ a \equiv true\ IsX\ A$$

13.2 Proofs of the Existence of God

The ontological proof for the existence of God goes as follows. The concept of God is by definition perfect. A concept is more perfect than another if it has a reference. Therefore as the concept of God is perfect it must have a reference. Thus God exists. We can attempt to formalise this.

$$perfecT\ X \supset exists\ X$$

$$perfecT\ X \supset omniscienT\ X$$

However we see that our original assumption was not,

$$perfecT\ God$$

but the more removed,

$$perfect\ GGod$$

that is the concept of the concept of God is perfect. Thus we must show that $perfect\ God$ that is that out concept of God is perfect, not that we can imagine a perfect concept of that concept.

13.3 Concepts and Objects

In general the relation between concepts and objects is many to one. We have seen examples of concepts which denote no objects. We also like to use a function *denot* which returns just one object.

Certain kinds of objects have standard concepts. Numbers have a standard concept, their representation in decimal. This will become clearer if the ideas of names were used. Standard names are strings. This has been extensively examined by philosophers.

We can have

$$\neg knew\_that(kepler, Number\ Planets)$$

which is the statement that Kepler did not know how many planets there were.

$$\neg knew\_that(Kepler, Composite\ Number\ Planets)$$

This states that Kepler did not know that the number of planets was composite, but we have

$$knew\_that(kepler, Concept1\ denot(Number\ Planets))$$

This says that Kepler knew that eight was composite. People usually will chose the first interpretation, but we can express both in our formalisation. $Concept1$ is a function which takes an object ant returns its standard concept.

13.4 Knowledge and True Belief

In the previous example if we considered the number of planets to include Earth, then Kepler would have thought the number of planets was six. He thus would have thought that the number of planets was composite. However, if he was told that science had discovered new planets then he would not know that the number of planets was composite.

If you believe something and it is true, philosophers will say that you do not necessarily know it. If you heard a shot, you might think the President was shot. You later find it was a back-fire. You believed the fact, and it was true, but you did not really know it.

We can have a concept of stable belief, that is belief that cannot be over-ruled by extra information.

Generally philosophers think they are done when they have proved that knowledge is not true belief. In AI a lot of problems are treated in a perfectly acceptable way by considering knowledge as true belief. We should look for sufficient conditions, for the use of the naive notion, not dismiss it out of hand.

If you follow the philosophers literature, the naive notions are abandoned, and ideas become more complex. Philosophers may to a certain extent be splitting hairs that do not need to be split. It may be possible to make this into a technical argument. If we can show that a concept is sensible in its common notions, then to declare it invalid for a reason that does not commonly occur is itself invalid.

This can be shown in the analysis of digital circuits. In linear circuts with capacitors and resistors, you have rules that allow you to calculate the responses of the circut to any applied voltages. This is no longer true for digital circuits. If you idealise the circut to and's and or's you replace the analysis by logic. However if you consider it at the level of the flip flop, you have;

If the clock is off it is unresponsive to all inputs. If the clock is on and 0/1 is applied to the inputs S/R, then Q goes to 0 and $\overline{Q}$ goes to 1. If you set S/R to 1/0 you get Q going to 1 and $\overline{Q}$ to zero. If you set S/R to 0/0, Q and $\overline{Q}$ remain in their previous states. However, if you set S/R to 1/1, the output is undefined.

You can ask is this an incomplete description of the flip-flop. It tells you what happens if you go by the rules. In practise, no particular behaviour is guaranteed for that input. It may vary between different hips, it may vary between different trials without the same chip. The conceptt of how digital circuits behave depends on them being used according to the specifications.

This is true of many concepts in common-sense. They make sense only when used in their domain of applicability, if they are used according to their design rules. Wanting, Knowing, Believing, are concepts which have this property. In general intensional properties work under a limited usage.

Another concept like this is welfare. Consider the following; there is this chicken in a chicken farm. It is going to become a broiler. If you step on its foot you will save it from being cooped up for 10 weeks. Is it in the chicken's welfare to step on its foot.

The chicken's welfare is not well defined. The chicken's welfare only makes sense if you limit the temporal scope of what you want to discover. Is it better for the welfare of an egg to hatch and grow up for 6 weeks to be slaughtered. It is clearly difficult to have a non-local notion. This however, does not excuse us from dealing with the simple cases of the notion. Like the flip-flop only part of it is defined. If you find a counter example it could be a mistake, but either it could be an unstated restriction on the domain of applicability of the concept.

13.5 Telephones Again

Knowing what can be defined in terms of knowing that. If we have telephone number as a predicate of objects, we can use the notion of standard concept.

$$knows\_what(pat, Telephone\ Mike) \equiv$$

$$\exists x.t\_number x \wedge know\_that(pat, Equal(Telephone\ Mike), Concept1\ x)$$

Pat knows Mike's telephone number if and only if there is a number x and Pat knows that the standard concept of x is equal to The concept of Mike's telephone number on denotation. Naturally we have,

$$denotes(Concept1\ x, x)$$

There are problems with the idea of a standard concept. With people there is not exactly a standard concept. If you go to a party and ask who is John Smith, and they say that guy over there, you are satisfied. On the other hand if you go to a party and ask who is that guy over there the answer John Smith is acceptable.

There are some functions appearance and name, that seem related to the idea of standard concept. In a logical sense the remembered appearance can be treated in the same way as a name. You come up with a description of an appearance, not an appearance. But in the same way, we give a descritpion of a name not the name itself. From a logical point of view, there seems little difference. A computer program, if it had this ability could readily transfer this information. Unfortunately, we cannot transfer our sensations, we can only transfers descriptions of them in language. Appearances are describable in words, it just happens that we are not good at it. In the 19^{th} century, before photographs, the vocabulary for description was more developed than it is today.

Chapter 14

Lecture: 14

14.1 Ascribing Mental Qualities

14.1.1 Counterfactuals

This lecture concerns the paper [20]. It has been claimed that one part of the paper is difficult to understand. This is the part on Cartesian counterfactuals. Counterfactuals is short for counterfactual conditional sentences, that is a sentence $p \supset q$ where p is false. In logics with material implication all such sentences are true. To get a useful notion therefore we must use something else.

It is quite useful to sometimes consider these sentences as false. Picking up a pen is intentional. We can say that, if I did not intend to have picked up this pen, then I would not have picked it up. This only creates a useful definition if it is somtimes ture and sometimes false.

The general philosophical approach has been to try to assign truth values to them. The Lewis/Stallmaker approach is the leading one. David Lewis wrote a book called "Counterfactuals". Their theory is this, they refer to the notion of a possible world. The counterfactual $p \supset q$ asserts that the statement $p \supset q$ is true in the nearest world where p is true, It does not carry with it any notion of what the possible wolds are. Thus it seems difficult to evaluate the truth of any particular counterfactual.

14.1.2 Cartesian Counterfactuals

Cartesian counterfactuals, can be viewed as a particular case of the Lewis approach with a metric for closeness. It is the simplest case. It is not claimed that this approach covers all cases of interest, but that it is the clearest case.

Cartesian counterfactuals exist with a sentence a theory and a world. The sentence is interpreted by the theory. That is, the sentence is determined by looking at the world through the lens of the theory. We imagine the world, or database as a set of cartesian co-ordinates. We can imagine in a particular case they would be a four tuple. We can then talk about a particular point say $(1, 2, 1, 2)$. If we are interested in the distance from the origin we have

$$s = \sqrt{x^2 + y^2 + z^2 + w^2}$$

Now if we ask the question, if y were 3, would s be 7. This means that x, z and w hold their particular values. Therefore we have

$$\sqrt{1 + 9 + 1 + 4} = \sqrt{15} \neq 7$$

So this is an example of an untrue counterfactual. As long as we have a cartesian product structure we can determine the value of a counterfactual relative to the present state. A true counterfactual would be "if y were 3 then s would be $\sqrt{15}$".

14.1.3 Example of the Skier

Consider the following example, in a larger domain. Suppose two ski instructors are watching a pupil, who falls. One says that if he had bent his knees he would not have fallen. The other says that he disagrees, and that had he put his weight on his downward ski, he would not have fallen. Suppose they had a video made. After examining it they agree, that he should have bent his knees.

What is the closest world in which he bent his knees. We could imagine a childhood accident that caused him always to have bent knees, or perhaps something hitting him on the knee. We should exclude these possibilities as the nearest world.

The cartesian notion is as follows: The ski instructors have a theory of sking. A skier is a stick figure with a mass distribution. There is a 2d slope, the skier has joints, and has he slides down the angle of his joints are changed. What will happen is a consequence of what his joints do. According to this theory, bending his knees is now a perfectly valid set of trajectories. What is meant is that he did each of the other parameters in the theory the same and changed just the bend of his knees. Their theory of sking would give a definite answer.

The truth of the counterfactual is related to two things, the theory of sking and the slope, motion of his joints, etc. This theory of sking has not reasons why there is movement, but we could imagine a more complicated theory that did. The important property is that there are a number of input parameters and we can relate the distance between possible worlds by the number of input parameters that are changed.

14.1.4 Usefulness of Counterfactuals

We have not yet addressed what the usefulness of counterfactuals is. It allows us to learn more than we otherwise could from examples. Suppose you pass a car on a hill. Someone says that if there had been a car coming the other way, you would have had a head on collision. From this you can learn, without having the collision, no to pass on hills so readily. Mentally we can vary the situation, and learn something from this. We would like computers to be able to do this kind of learning. We want them to learn more than just what actually happened.

A chess program, playing over a game, can reason counterfactually — "If Spassky had played this move, he would have beaten Fischer." The program can learn from this. It is important, if we are to use counterfactuals for learning that we can recognise that they are sometimes false. In the above example we can imagine the response that "If there was another car it would have been visible in time for me to avoid it". The counterfactual can be true or false here. This use of counterfactuals for learning is a different use than for defining intentional properties.

14.1.5 Meaningfulness of Counterfactuals

The most common form of counterfactual is a change in one of the components describing the real world. We are told that a single component of a multi-component thing is changed. The meaningfulness of the counterfactual depends on their being some structure to the propagation of the change, through the situation. If there is a function of the situation that gives these constraints, then we have a definition for a theory, and we can answer the counterfactuals as before. Without this property there is no way of deciding what else, if anything, changes.

If you have a Cartesian co-ordinate system, and an oblique system as well, then you will have two different answers to a counterfactual query. In this case the counterfactual has no definite meaning. Thus we can see that the meaning of a counterfactuals is related to the theory as well as the situation itself.

Consider the following historical example. If the South had won the civil war then slavery would have been abolished by 1890. This is not definite enough for us to be able to answer meaningfully. If we add how they won, let us presume the Northern troops were hit by a meteorite, then we may be able to use our historical theories to good effect.

A more difficult example is "If wishes were horses, beggars would ride". This seems to pose problems for anyone trying to choose a closest world. There are probably enough different theories to make positing any particular truth value to this impractical. However the proverb does have meaning in social situations. It means that one should not engage in idle speculation. A philosopher might say that a theory of counterfactuals is no good unless it ascribes meaning to all of the examples. Possibly a lot can be achieved by AI with a theory of counterfactuals which does not include the above proverb as a solved case. The interesting thing to do when faced by a simple theory is to come up with sufficient conditions not counter examples.

14.2 Ascribing Mental Qualities

The knock-down drag-out battle in this area of philosophy is ascribing beliefs to thermostats. Dennett is one of the philosophers who sides with AI in this respect. Dennett goes into great detail in the ascription of mental qualities in his book "The Intentional Stance". Dennett claims that with regard to an object you can take three stances.

14.2.1 The Physical Stance

The physical stance is the simplest where you consider the physical qualities of the object. In the case of thermostat you might study the bi-metallic strip, that changes with temperature, due to different co-efficients of thermal expansion on either side. The physical stance would also cover its connectivity to other things.

14.2.2 The Intentional Stance

The intentional stance is where we predict the behaviour of something by ascribing goals and belief to it, and judging that it will act in a way that it believes will achieve its goals. The reason that this is useful is that we may not have sufficient information in many situations to examine it from the physical stance. We may only know that it is a thermostat, and not what type of internal mechanism it has.

The following example was found by Michael Besson. In the instruction for a particular electric blanket, it says not to place the thermostat on a window sill or near a radiator, or the thermostat will think that the room is hotter/colder than it actually is. We can ask why did they not use a description using the physical stance, with bi-metallic strips. The manufacturer thought that the users would understand the first explanation more easily. Also the company might want to change the bi-metallic strip to an electronic control. Here the physical stance changes but the intentional does not.

14.2.3 The Design Stance

Dennett has a third stance, the design stance. This says that an object has a purpose, and is described in terms of this function. The canonical example is an alarm clock.

If you go to a hotel you usually set the alarm clock to wake you up. You do not care whether it is electric, pendulum, or mechanically controlled. What you do know is that there is a current time, a time when you want to get up, and a way of turning the alarm on and off. All alarm clocks, even those controlled by balance wheels or quartz crystals have these properties.

It is legitimate to apply this standard to a whole range of devices. This can be applied to all sorts of biological systems that have evolved an ability to function in a particular way. If you saw an animal you had never seen before with legs, you would be justified in surmising that it used them to get around. This is taking the design stance. You have not observed that the new animal uses them for that purpose but you assume that they have this function.

Another example is that if you observe a gully in a mountain you realise that the water must be able to get to the sea, down some channels.

14.2.4 When to Ascribe Beliefs

Dennett is not very interested in machines. He instead uses a lot of biology to confound his protagonists in philosophy. In his book "Brainstorms", he asks, in relation to whether a computer can feel pain, whether one can in general feel pain and not know it. In vertebrates, whether you feel pain can be very complicated. If you describe the pain in terms of various effects the above seems to be true. The book goes into a lot of detail about his and has been influential in the field of psychology.

One of the talking points in this field is whether Vervet monkeys deceive each other. There is a great need to be precise on exactly what observations need to be made to establish this. Primatologists tend to conclude that Vervet monkeys do deceive each other based on Dennett's analysis.

If we were interested in the question of whether a machine believed it was in state s, after enough study we could come up with a definition in terms of its structure answering this question. This is the wrong approach, and will not succeed. Everyone would agree that most machines will not have any beliefs in most states.

We can ascribe a belief to a pedestal in the main hall of Oxford University. It believes that it is located in the centre of the cultural world, and it has the goal that it wishes to remain in the centre of the cultural world. This explains its behaviour, it does nothing. The flaw here is that ascribing beliefs tells us nothing that we did not know before.

We need to give criteria for the useful ascription of belief to a machine. It is possible to assign a second order predicate, that is true if the belief predicate we are using is a useful predicate. We can imagine some conditions we would like it to satisfy. The set of beliefs should contain sufficient obvious consequences. It should be consistent. It should be able to get new beliefs as the state changes, that is there should be new beliefs about the environment that correspond to observation. We would also it to be able to get new beliefs from communication. If we call some of its beliefs, goals, it should act in a way that it believes will achieve its goals.

If we have found one such belief predicate that satisfies the above condition, we have done very well. However we can imagine being asked how we know that there is not another predicate. We can argue against this using argument from cryptography.

The argument claims that it could happen but is exceedingly unlikely. If we look at simple substitution ciphers we have only one or two that give two answers.

"Le prisonier est fort ansi il ne rien dit"

and

"Le prisonier est mort ansi il ne rien dit"

Naturally the ambiguity does not translate. The likliehood of this happening is a function of the space of meaningfull sentences and the length of the cipher. In any case the argument is that if you have one good ascription it is unlikely that you will find another.

Bibliography

[1] **Epistemological Problems in AI** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[2] **Circumscription, A form of Nonmonotonic Reasoning** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[3] **On Representations of Problems of Reasoning about Actions** *Saul Amarel* Readings in Artificail Intelligence Bonnie L. Weber and Nils J. Nilsson

[4] **Application of Circumscription to Common Sense Reasoning** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[5] **Hacker** *Sussman* 1972

[6] **Interplan** *Waldinger* 1973

[7] **Towards a Mathematical Theory of Computation** *McCarthy* 1963

[8] **What are the Limitations of the Situation Calculus?** *Vladimir Lifschitz and Michael Gelfond* Automated Reasoning, Essays in Honor of Woody Bledsoe, Kluwer Academic Publishers 1991

[9] **Some Philosophical Problems from the Standpoint of AI** *McCarthy and Hayes* 1969 in Formalizing Common Sense, 1990 Ablex publishing Corporation

[10] **Word and Object** *Willard V. O. Quine* 1960

[11] **Philosophical Problems from the Standpoint of AI**, *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[12] **Ph.D. thesis**, *Claude Shannon* 1940

[13] **Programs with Common Sense** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[14] **A Program to Play Chess End Games**, *Barbara J. Huberman* Stanford University. Department of Computer Science. CS 106 tech report. 1968

[15] **Map Colouring and the Kowalski Doctrine** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[16] **Applications of Circumscriptin to Formalising Common Sense Knowledge** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[17] **Modal Logic** *Hintikka K. J. J.*

[18] **Counterfactuals** *David Lewis* Cambridge Mass; Harvard University Press 1973

[19] **First Order Theories of Individual Concepts and Propositions***John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[20] **Ascribing Mental Qualities to Machines** *John McCarthy* in Formalizing Common Sense, 1990 Ablex publishing Corporation

[21] **Learning and Executing Generalised Robot Plans** *Fikes Hart and Nilsson.*A. I. J. vol 3 issue 4 1972

[22] **Reasoning about Actions: Possible Worlds Aprroach** *Matt Ginsberg* Readings in Non-monotonic Reasoning Ed. Ginsberg.

[23] **On the Semantics of Strips** *V. Lifcshitz*, Reasoning About Actions and Plans, 1986 workshop.

[24] **Pointwise Circumscription** *V. Lifschitz*, Readings in Nonmonotonic Reasoning, Ed. Ginsberg.

[25] **Chronological Ignorance** *Y. Shoham*, Readings in Nonmonotonic Reasoning, Ed. Ginsberg.

[26] **Meaning and Necessity** *R. Carnap*, Chicago IL. University of Chicago Press, 1956.

[27] **An introduction to Common Sense Reasoning** *V. Lifschitz* in handout.

[28] **Default Reasoning, Nonmonotonic Logic, and the Frame Problem***D. McDermott and S. Hanks* Readings in Nonmonotonic Reasoning, Ed. Ginsberg

[29] **A Logic for Default Reasoning***R. Reiter* Readings in Nonmonotonic Reasoning, Ed. Ginsberg

[30] **A Paradox Reagained** *Kaplan and Montague* Notre Dame Journal of Logic 1963