How to be Artificially Intelligent – Computational Logic and Human Thinking (draft)

Robert Kowalski
Department of Computing
Imperial College London
1 June 2009

The goal of this book

This book aims to show that Computational Logic, originally developed for Artificial Intelligence, can be used ordinary people to improve their own intelligence without the aid of computers.

Computational Logic has been used in Artificial Intelligence over the past 50 years or so, in the attempt to develop computers that display human intelligence. The results of this attempt may have been mixed; and, as some critics would argue, the goal itself may have been questionable. But despite these reservations, arguably the most important result of these efforts has gone largely unnoticed: Computational Logic can be used, not only for Artificial Intelligence, but also for the original purpose of logic, to improve human thinking, and thereby to improve human behaviour.

Computational Logic is based on Symbolic Logic. But Symbolic Logic has become a branch of Mathematics and has largely lost touch with its roots in human reasoning. Computational Logic as used in Artificial Intelligence also employs mathematical notation, which facilitates its computer implementation but obscures its relevance to human thinking. For this reason and to reach a wide audience, I have written the main part of the book informally, without mathematical or symbolic notation. Towards the end of the book, I intend to discuss how to represent the informal logical language in symbolic form.

Because human thinking is also the subject of study in many other fields, in addition to Logic and Artificial Intelligence, I have drawn upon related studies in Cognitive Psychology, Linguistics, Philosophy, Law, Management Science and English Composition.

The relationship between Logic and Thinking

Traditional Logic, Symbolic Logic and Computational Logic are all concerned with formalising the laws of thought. Along with related fields such as Law and Management Science, they focus on *normative* theories, which prescribe how people ought to think. Cognitive Psychology is also concerned with thinking, but it focuses almost exclusively on *descriptive* theories, which study how people actually think in practice. For the most part, the two kinds of theories have been developed in isolation, and bear little relationship with one another.

However, in recent years, cognitive psychologists have developed *Dual Process* theories, which can be understood as combining descriptive and normative theories. Viewed from the perspective of Dual Process theories, traditional descriptive theories focus on *intuitive thinking*, which is associative, automatic, parallel and subconscious. Traditional normative theories, on the other hand, focus on *deliberative thinking*, which is rule-based, effortful, serial and conscious.

Dual Process theories study the relationship between intuitive and deliberative thinking. Some of these theories argue that intuitive thinking "quickly proposes intuitive answers to judgement problems as they arise", whereas deliberative thinking "monitors the quality of

these proposals, which it may endorse, correct, or override". [1]In this book, I will argue that Computational Logic can serve as a dual process theory, in which intuitive and deliberative thinking are combined.

But Logic views thinking in linguistic terms, as representing information in the form of sentences, and manipulating sentences to infer new sentences representing additional information. Therefore, Logic is concerned both with thinking and with language. I will argue, accordingly, that not only can Computational Logic help us to improve the way we think, but it can also help us to improve the way we communicate with other people in natural languages, like English. Both arguments are linked to the thesis that Computational Logic can be viewed as an approximation to the language of human thought.

Computational Logic and the Language of Thought

As used in Artificial Intelligence, Computational Logic functions first and foremost as an intelligent agent's *language of thought*. It includes a *syntax*, which determines the *form* of the agent's thoughts, a *semantics*, which determines the *contents* of those thoughts, and an *inference engine*, which derives new thoughts as logical consequences of existing thoughts. In this role, Computational Logic functions as a *private language*, representing the agent's goals and beliefs, and helping the agent to regulate its behaviour. Other private languages can serve the same purpose, but I will argue that Computational Logic has significant advantages over the alternatives.

In *multi-agent systems* in Artificial Intelligence, the private language of an individual agent also serves the secondary function of representing the meanings of its communications with other agents. These communications are expressed in a shared *public language*, which may differ from the private languages of individual agents. The task of a communicating agent in such a situation is to translate thoughts from its private language into the public language, in such a way that the receiving agent can readily translate those public communications into appropriate thoughts in its own private language.

It would be easy if all agents shared the same private language, and if that private language were identical to the public language of the community of agents. This can arranged by design in an artificial multi-agent system, but at best it can only be approximated in a society of human agents.

The distinction between private and public languages, which is so clear cut in Artificial Intelligence, has been proposed in the Philosophy of Language to explain the relationship between human thinking and communication. Many of these proposals, which for simplicity can be lumped together as Language of Thought (LOT) proposals, maintain that much human thinking can be understood as taking place in a language of thought. The most famous proposal along these lines is Fodor's hypothesis[2] that the LOT is a universal language, which is independent of the Babel of public languages. Other proposals argue that a person's LOT is specific to the public language of the person's social community.

No matter where they stand on the issue of the universality of the LOT, most proposals seem to agree that the LOT has some kind of logical form. However, for the most part these proposals are remarkably shy about revealing the details of that logical form.

In this book, I argue that Computational Logic has many of the characteristics needed for a universal, logical LOT. I will draw the main support for my argument from advances in the use of Computational Logic in Artificial Intelligence. But I will also draw support from the

---

[1] Daniel Kahneman and Shane Frederick. Representativeness revisited: Attribute substitution in intuitive judgment. In T. Gilovich, D. Griffin and D. Kahneman (Eds) *Heuristics of Intuitive Judgment: Extensions and Application.* New York: Cambridge University Press (2002)

[2] Jerry Fodor. *The Language of Thought*, Harvard University Press, 1975.

relationship between Computational Logic and normative theories of human communication.

Computational Logic and Human Communication

Much of the time, when we speak or write, we simply express ourselves in public, without making a conscious effort to communicate effectively. But when it really matters that we are understood - like when I am writing this book - we try to be as clear, coherent and convincing as possible. The difference is like the difference between descriptive and normative theories of thinking; and, as in the case of the two kinds of thinking, the two kinds of communication are studied in different academic disciplines. Whereas linguistics is concerned with developing descriptive theories about how people use language in practice, rhetoric and allied disciplines such as English composition are concerned with *normative* theories about how people can use language to communicate effectively.

In this book we want to develop a normative theory of intelligent thinking and behaviour. But we need to pay attention to descriptive theories, because descriptive theories help us to understand where we are starting from, whereas normative theories show us where we need to go.

The descriptive theory of communication that comes closest to a normative theory is probably the Relevance theory of Sperber and Wilson[3]. It is based on a more general theory of cognition, which loosely speaking hypothesizes that, given competing inputs from their environment, people direct their attention to those inputs that provide them with the most useful information for the least processing cost. Applied to communication, the theory hypothesizes that, given a potentially ambiguous communication as input, readers or listeners translate the input into a logical form that maximises the amount of information it contains and minimises the computational effort needed to derive it.

Relevance theory is compatible with the hypothesis that Computational Logic, or something like it, is the logic of the language of thought. Like Computational Logic, Relevance theory also has both logical and computational components. Moreover, it provides a link with such normative theories of communication as Joseph Williams' guides to English writing style[4].

One way to interpret Williams' guidance is to understand it as including the advice that writers should express themselves in a form that is as close as possible to the logical form of the thoughts they want to communicate. In other words, they should say what they mean, and they should say it in a way that makes it as easy as possible for readers to extract that meaning. Or to put it still differently, the public expression of our private thoughts should be as close as possible to the structure of those thoughts.

If our private languages and public language were all the same, we could literally just say what we think. But even that wouldn't be good enough; because we would still need to organise our thoughts coherently, so that one thought is logically connected to another, and so that our readers or listeners can relate our thoughts to thoughts of their own.

Williams' guidance for achieving coherence includes the advice of placing old, familiar ideas at the beginnings of sentences and placing new ideas at their ends. Here is an example of his advice, which uses an informal version of the syntax of Computational Logic, and which incidentally shows how Computational Logic can be used to represent an agent's goals and beliefs to guide its behaviour:

> *You want to have a comfortable income.*
> *You can have a comfortable income if you have a good job.*

[3] Dan Sperber, Deidre Wilson. Relevance. 1986 - Blackwell Oxford.
[4] Joseph Williams. Style: Toward Clarity and Grace. Chicago: University of Chicago Press (1990, 1995).

You can have a good job if you study hard.
So you should study hard.

It may not be poetry, but at least it's clear, coherent and to the point. It's also pretty close to what most people think.

What is Computational Logic?

Computational Logic combines the use of logic to represent information with the use of inference to derive consequences and to perform computations. It includes the classical use of inference to derive consequences from input observations; for example, to derive the conclusion that *Mary can have a good job* from the observation that *Mary studies hard* using the belief that in general *a person can have a good job if the person studies hard*.

It also performs computation by using goal-directed inference, to reduce goals to sub-goals; for example to "compute" the sub-goal that *Bob should try to have a good job* from the goal *Bob wants to have a comfortable income* using the belief that in general *A person can have a comfortable income if the person has a good job*. Goal-directed inference gives Computational Logic the power of a high-level computer programming language. Indeed, the programming language, Prolog, which stands for Programming in Logic, exploits this form of computation mainly for applications in Artificial Intelligence.

Computational Logic, as we investigate it in this book, also includes the use of inference to help an agent choose between alternative courses of action. For example, having used goal-directed reasoning to derive two alternative ways, *Bob studies hard* and *Bob robs a bank,* of achieving the higher-level goal *Bob has a comfortable income*, Computational Logic can be used to infer the possible consequences of the alternatives before deciding what to do. In particular, if it derives the consequence that *Bob can go to jail* if Bob chooses the second alternative, *Bob robs a bank*, then it will encourage Bob to choose the first alternative, *Bob studies hard,* instead.

What is Artificial Intelligence?

Artificial Intelligence (AI) is the attempt to program computers to behave intelligently, as judged by human standards. Applications of AI include such problem areas as planning, natural language understanding, and robotics. The tools of AI include such techniques as search, symbolic logic, artificial neural networks and reasoning with uncertainty. Many of these tools have contributed to the development of the Computational Logic we investigate in this book. However, instead of concerning ourselves with Artificial Intelligence applications, we will see how the tools of Computational Logic can also be used by ordinary people to think and behave more intelligently.

I gave a short course based on this book at the The International Center for Computational Logic (ICCL) 2008 summer school on Computational Logic and Cognitive Science. A copy of the slides that accompanied the course can be found at:
http://www.computational-logic.org/content/events/iccl-ss-2008/lectures.php?id=24

My colleague, Jacinto Davila, has also used a draft of this book for a course at Universidad de Los Andes, Venezuela. Here is a link to his Spanish translation of an earlier draft:
http://webdelprofesor.ula.ve/ingenieria/jacinto/kowalski/logica-de-agentes.html

I would be very grateful for any comments on the book. Please send them to me at rak@doc.ic.ac.uk.

Contents

Because this draft is incomplete, I have included as appendices four papers, which I intend to integrate into the final version of the book:

> Kowalski, R. "The Logical Way to be Artificially Intelligent". Proceedings of CLIMA VI (eds. F. Toni and P. Torroni) Springer Verlag, LNAI, 2006, pp. 1-22.
>
> Kowalski, R. "Computational Logic in an Object-Oriented World" In Reasoning, Action and Interaction in AI Theories and Systems – Essays Dedicated to Luigia Carlucci Aiello (eds. O. Stock, M. Schaerf) Springer Verlag, Lecture Notes in Comp;uter Science, Vol 4155, LNAI, 2006.
>
> Kowalski, R. "Reasoning with Conditionals in Artificial Intelligence" To appear in The *Psychology of Conditionals* (ed. M. Oaksford) OUP, 2009.
>
> Kowalski, R., "Legislation as Logic Programs", in *Logic Programming in Action* (eds. G. Comyn, N. E. Fuchs, M. J. Ratcliffe), Springer-Verlag, 1992, pp.203-230.

The first paper is a summary of the book, written for a more academic audience. The second is a later version of chapters 9 and 10. The third discusses some of the relationships between computational logic and the psychology of human reasoning. The fourth discusses the relationships between computational logic and legal reasoning. Importantly, the papers contain references, which are missing from the main text. Unfortunately, there is considerable overlap between some of the papers and the book, for which I apologise. I promise to eliminate them in the final version.

Chapter 1. What to do in an emergency

If some form of Computational Logic is the language of human thought, then the best place to find out about it would seem to be inside our heads. But if we simply look at the structure and activity of our brains, it would be like looking at the hardware of a computer when we want to learn about its software. Or it would be like trying to do sociology by studying the movement of atomic particles instead of studying the interactions between human beings. Better, it might seem, just to use common sense and rely on introspection.

But introspection is notoriously unreliable. Wishful thinking can trick us into seeing what we want to see, instead of seeing what is actually there. The behavioural psychologists of the first half of the 20th century were so suspicious of introspection that they banned it altogether.

Artificial Intelligence offers us an alternative approach, to construct computer programs whose input-output behaviour simulates the externally visible manifestations of human mental processes. To the extent that we succeed in the simulation, we can then regard the structure of those computer programs as analogous to the structure of human mind, and we can regard the activity of those programs as analogous to the activity of human thinking.

But different programs with different structures and different modes of operation can display similar behaviour. As we will see later, many of the differences between different programs displaying the same behaviour can be understood as differences between levels. Some programs are closer to the lower level of the hardware, and consequently are more efficient; others are closer to the higher level of the application domain, and consequently are easier to understand. We will explore some of the relationships between the different levels later in the book, when we explore levels of consciousness and the meaning of life. But in the meanwhile, we can get an inkling of what is to come by first looking closer to home.

If human thoughts have the structure of language, then we should be able to get an idea of their structure by looking at natural languages such as English. Better than that, we can look at English communication in situations where we do our best to express ourselves as clearly, coherently and effectively as possible. Moreover, we can be guided in this by the advice we find in books on English writing style.

For the purpose of revealing the language of thought, the most important advice is undoubtedly the recommendation that we express ourselves as clearly as possible - making it as easy as we can for the person we are addressing to translate our communications into thoughts of her own. Everything else being equal, the form of our communications should be as close as possible to the form of the thoughts that they aim to convey.

What better place to look than at communications designed to guide people how to behave in emergencies, in situations where it can be a matter of life or death that the recipient understands the communication with as little effort as possible.

Imagine, for example, that you are travelling on the London underground and you hear a suspicious ticking in the rucksack on the back of the person standing next to you. Fortunately, you see a notice explaining exactly what to do:

Emergencies

Press the alarm signal button
to alert the driver.

The driver will stop
if any part of the train is in a station.

If not, the train will continue to the next station,
where help can more easily be given.

There is a 50 pound penalty
for improper use.

The public notice is designed to be as clear as possible, so that you can translate its English sentences into thoughts of your own with as little effort as possible. The closer the form of the English sentences to the form in which you structure your thoughts, the more readily you will be able to understand the sentences and to make use of the thoughts that they communicate.

The thoughts that the management of the underground wants you to have are designed to make you behave effectively in an emergency, as well as to prevent from you behaving recklessly when there isn't an emergency. They are designed, therefore, not only to be clear, but to be to the point – to tell you what to do if there is an emergency and what not to do if there isn't an emergency. But they are also intended to be coherent, so that you can easily relate the new thoughts that new sentences communicate to existing thoughts you already have in your head. These existing thoughts include both thoughts that were already there before you started reading and thoughts that might have been conveyed by earlier sentences in the text you are reading.

The London Underground Emergency Notice as a program

The purpose of the emergency notice is to regulate the behaviour of passengers on the London underground. It does so much in the same way that a computer program controls the behaviour of a computer. In general, much of our human communication can be understood in such computational terms, as one human attempting to program another, to elicit a desired response.

At first sight, this computational view of human communication may give offence, because it may suggest that people should be treated as though they were merely machines. That is far from my intention. I intend only to propose that thinking of people as computing agents can sometimes help us to communicate with them in more effective and more efficient terms. Our communications will be more *effective*, because they will better accomplish our intentions; and they will be more *efficient*, both because they will be easier for other people to understand, and because the information they convey will be easier for other people to use for their own purposes.

Understanding a communication is like the process that a computer performs when it *translates* (or *compiles*) a program written in an external *source language* into an internal *target language* that the computer already understands. When a computer compiles the source program, it needs both to translate individual sentences of the program into the target language and to place those sentences into a coherent internal structure expressed as a target program. Compiling a program is efficient when it can be done with as little processing as necessary. Analogously, understanding an English communication is *efficient* when compiling it from its English form into a mental representation can be done with as little effort as possible.

Using the information in a communication is like *executing* a target program, after it has been compiled. When a computer executes a program, it follows the instructions mechanically in a systematic manner. When a person uses the information in a communication, the person combines that information with other information that the person already has and uses the information to solve problems. I will argue that people perform much of this process of using information systematically, automatically and unconsciously. Like a computer program, the information that people use to solve problems is *efficient* if it helps them to solve problems with as little effort as possible.

The computational nature of the emergency notice is most obvious in the first sentence:

Press the alarm signal button
to alert the driver.

This has the form of a *goal-reduction procedure*:

Reduce the goal of alerting the driver
to the sub-goal of pressing the alarm signal button.

Goal-reduction procedures are a common form of human knowledge representation. They structure our knowledge in a way that facilitates achieving goals and solving problems. Here the thought communicated by the sentence is that the goal of alerting the driver can be reduced to the sub-goal of pressing the alarm signal button.

To understand and make use of the goal-reduction procedure, you need to assimilate it into your existing goals and beliefs. For example, you might already know that there could be other ways of altering the driver, such as shouting out loud. You probably know that alerting the driver is one way of getting help, and that there are other ways of getting help, such as enlisting the assistance of your follow passengers. You probably recognize that if there is an emergency then you need to deal with it appropriately, and that getting help is one such way, but that other ways, such as running away or confronting the emergency head on yourself, might also be worth considering.

Goal-reduction procedures are also a common form of computer knowledge representation, especially in Artificial Intelligence. Liberally understood, they can serve as the sole construct for writing any computer program. However, almost all computer languages also use lower-level programming constructs. Most of these constructs bear little resemblance to human ways of thinking.

But there is one other construct that is even higher-level than goal-reduction, and which may be even closer to the way humans structure their thoughts. This construct is exemplified by the logical form of the conditional sentences found in the second and third sentences of the emergency notice.

The Logic of the Second and Third Sentences of the Emergency Notice

Many linguists and philosophers subscribe to some form of Language of Thought Hypothesis (LOTH), the hypothesis that many of our thoughts have a structure that is similar to the structure of natural languages such as English. Most of those who subscribe to LOTH also seem to believe that the language of thought has a logical form. In this book, I will explore the more specific hypothesis that the language of thought has the logical form of conditional sentences. This hypothesis is supported in the emergency notice by the English form of the second and third sentences.

The second and third sentences of the emergency notice have the logical form of *conditionals*, which are also called *implications*. Conditionals are sentences of the form:

if condtions then conclusion
or equivalently *conclusion if conditions.*

The second sentence is written with its conclusion first; and the third sentence is written the other way around, with its implicit conditions first.

I have argued that the notice is designed to be as easy to understand as possible, and that as a consequence its external form should be a good indication of the internal form of its intended meaning. In particular, the external form of the second and third sentences suggests that their intended meaning has the logical form of conditionals.

However, whatever the form of the LOT, one thing is certain: Its sentences are *unambiguous*, in that they mean what they say. In contrast, English sentences are often *ambiguous*, because they can have several different meanings. For example, the English sentence *the first passenger attacked the second passenger with a rucksack* has two possible meanings. Either *the first passenger carried out the attack with a rucksack* or *the second passenger had a rucksack, and the first passenger attacked the second passenger in some indeterminate way.* The difference between the two meanings could make a big difference in a court of law.

Ambiguity is the enemy of clarity. It creates confusion, because the reader does not immediately know which of the several possible interpretations of the communication is intended; and it creates extra effort, because the reader has to explore different interpretations, to find an interpretation that makes the most sense in the context of the reader's background goals and beliefs.

You might be surprised, therefore, to discover that the second and third sentences of the notice are more ambiguous than they appear at first sight. In particular, the second sentence does not explicitly state what the driver will actually stop doing. It is unlikely, for example, that:

The driver will stop causing the emergency
if any part of the train is in a station.

Instead, it is more likely that:

The driver will stop the train in a station
if any part of the train is in the station.

But even this interpretation does not fully capture the sentence's intended meaning. Understood in the context of the first sentence, the second sentence has an additional implicit condition, namely that the driver has been alerted to an emergency. Therefore, the intended meaning of the second sentence is actually:

The driver will stop the train in a station
if the driver is alerted to an emergency
and any part of the train is in the station.

Without the additional condition, the sentence literally means that the driver will stop the train whenever the train is in a station, whether or not there is an emergency. If that were the case, the train would never leave a station once it arrived. To understand the sentence, the reader of the notice needs both general background knowledge about the way train drivers normally behave and specific knowledge about the context of the earlier sentences in the notice.

In the spirit of our interpretation of the second sentence, it should now be clear that the intended meaning of the third sentence is:

The driver will stop the train at the next station
and help can be given there better than between stations
if the driver is alerted to an emergency
and not any part of the train is in a station.

In natural language, it is common to leave out some conditions, such as *any part of the train is in the station*, that are present in the context. In more formal logic, however, the context needs to be spelled out explicitly. In other words, sentences in formal logic need to stand on their own two feet, without relying for support on the context around them.

The web of belief

Because individual sentences expressed in purely logical form do not rely on their contexts, collections of sentences in logical form can be written in any order. In theory, therefore, if this book were written in purely logical form, I could write it - and you could read it - forwards, backwards, or in any other order, and it would still have the same meaning. In fact, you could take any text written as a sequence of sentences in logical form, write the individual sentences on little pieces of paper, throw them up in the air like a pack of cards, and pick them up in any order. The resulting sequence of sentences would have the same meaning as the text you started with.

In contrast, much of the work in writing a book like this is in trying to find an order for presenting the ideas, so they are as clear, coherent, and convincing as possible. No matter whether or not I spell out in detail all of the contexts of individual sentences, I need to present those sentences in a coherent order, which relates consecutive sentences both to ideas you already had before you started reading and to ideas you obtained from reading earlier sentences.

One way to achieve coherence is to follow Williams's advice of placing old, familiar ideas at the beginnings of sentences and new ideas at their ends. Sometimes, as a limiting case, if an "old" idea is particularly salient, because it has just been introduced at the end of the previous sentence, then the old part of the next sentence can be taken for granted and simply left out. This is what happens in the

emergency notice, both in the transition from the first sentence to the second sentence, where the condition *the driver is alerted to an emergency* has been left out, and in the transition from the second sentence to the third sentence, where *any part of the train is in a station* has been left out.

If the language of thought is a logic of conditional forms, then the simplest way to achieve coherence is by linking the beginnings and ends of consecutive sentences by means of the conclusions and conditions of the thoughts they express, using such obvious patterns as:

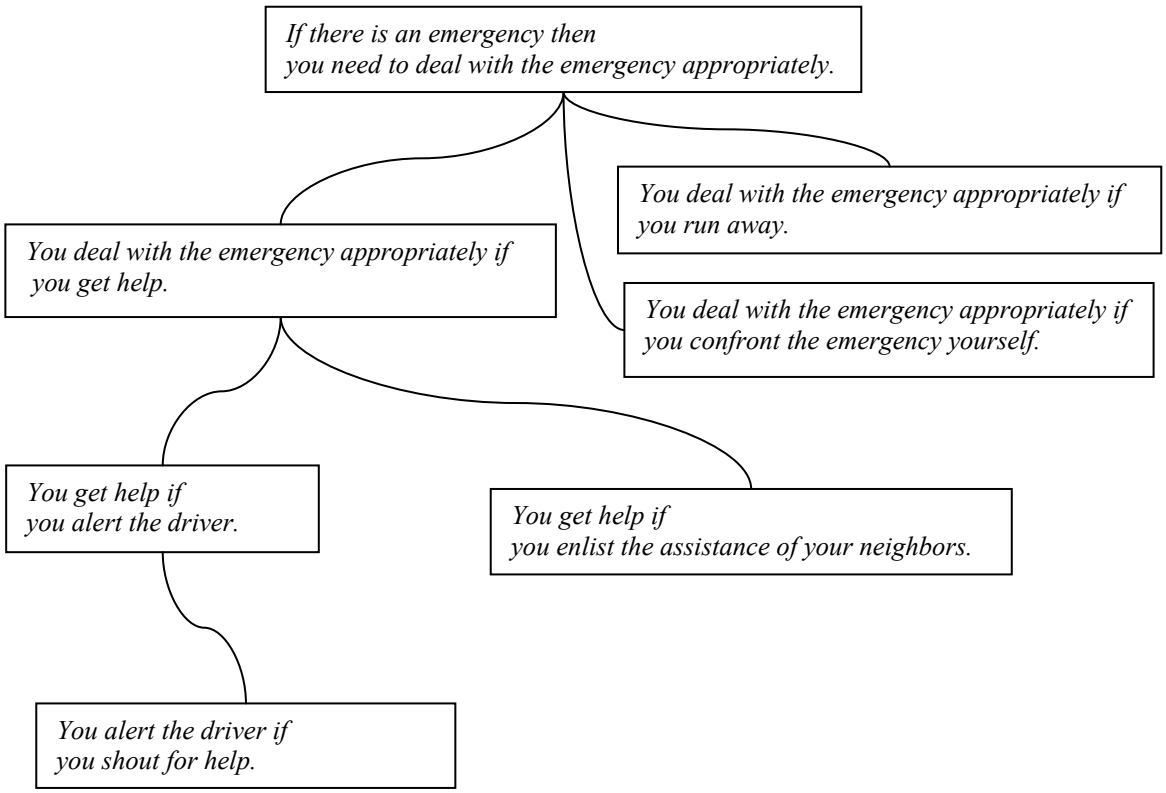
If condition A then conclusion B.
If condition B then conclusion C.

and

conclusion C if condition B.
conclusion B if condition A.

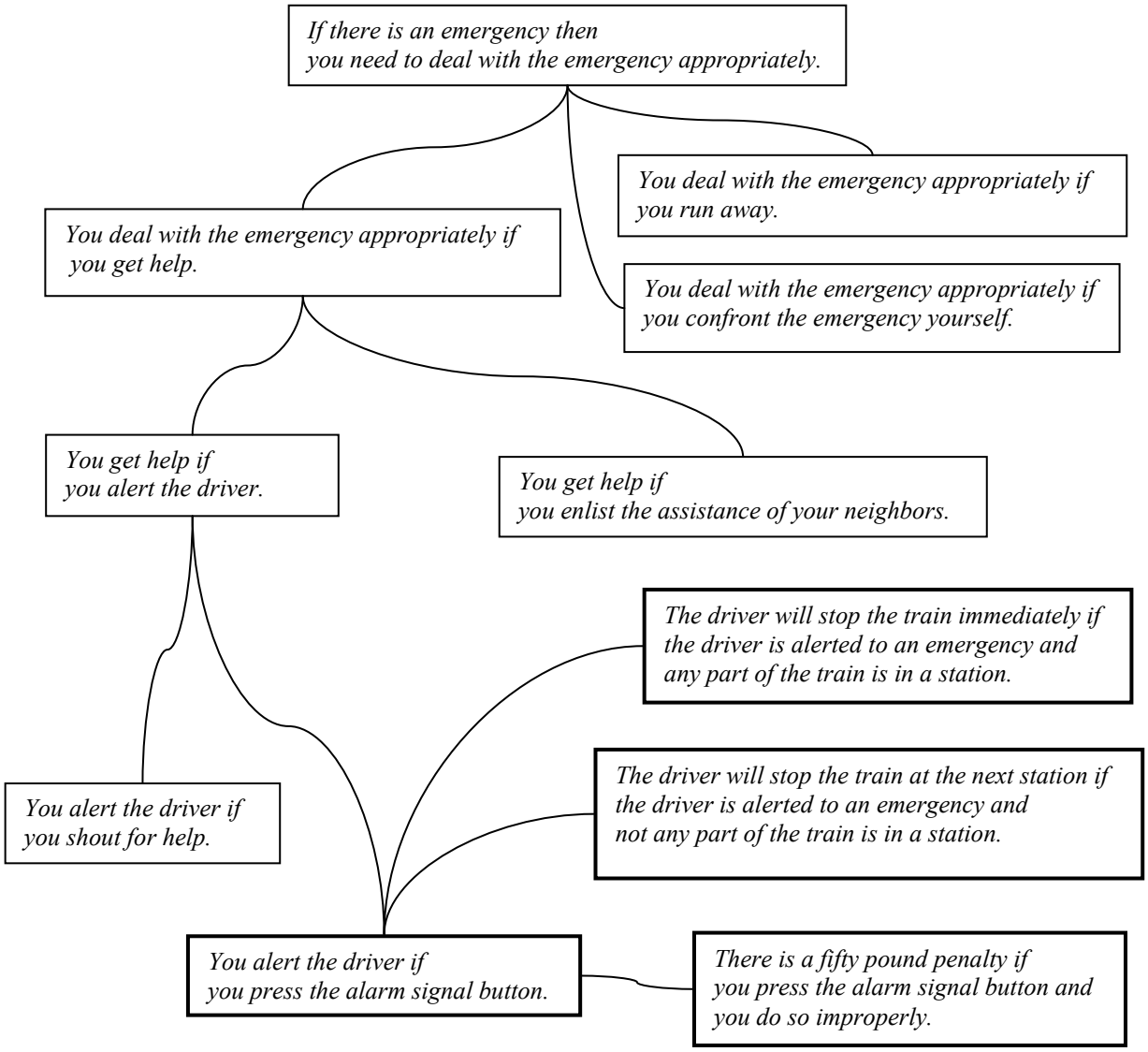
The need for coherence in human communication suggests that the language of thought is not an unstructured collection of sentences, after all. Rather, it is a linked structure in which sentences are connected by means of their conclusions and conditions.

Connection graphs[5], which link conclusions and conditions of sentences in logical form, have been developed in Artificial Intelligence to improve the efficiency of automated deduction. The links in connection graphs pre-compute much of the thinking that might be needed later when the need arises. Here are two connection graphs, representing some of a person's goals and beliefs before and after reading the emergency notice:



Part of a connection graph of a person's goals and beliefs ***before*** reading the emergency notice.

so Chapter 8 in Robert Kowalski, 1979: *Logic for Problem Solving*, North Holland Elsevier. Online at http://www.doc.ic.ac.uk/~rak/.



Part of a connection graph of the person's goals and beliefs ***after*** reading the emergency notice, assuming the person believes everything written in the notice. New beliefs are enclosed in bold boxes.

Connection graphs are similar to the structure that the philosopher W. V. Quine[6] called the *web of belief*. He argued that scientific theories, and human beliefs more generally, form a web of inter-related beliefs, which interact with the world of experience by means of their observational consequences. Scientific theories stand and fall together as a whole, because any belief, no matter how theoretical, might be involved in the derivation of an empirically testable observational consequence. Should an observational consequence of a theory be refuted by experience, consistency can be restored by revising any belief in the web of beliefs involved in the derivation of the faulty consequence.

Connection graphs are a concrete realization of the web of belief, showing how goals and beliefs are connected by links between their conditions and conclusions. Although in theory it might be possible to find a chain of connections between any two beliefs, in practice connections seem to cluster in such

[6] Willard V. O. Quine, 1963: Two dogmas of empiricism. In *From a logical point of view*. Harper & Row. 20-46.

relatively self-contained domains as those concerned with linguistic, logic-mathematical, musical, spatial, bodily kinaesthetic, interpersonal, and intrapersonal knowledge, similar to modules in a computer program and to the different kinds of intelligence in Howard Gardner's Theory of Multiple Intelligences.

There is much more to be said about connection graphs and about how they relate to modularity, multiple intelligences and theories of mind, but this will have to wait until later. In the meanwhile, we have a more pressing concern: How does the connection graph view of the mind, as a web of conditionals, relate to goal-reduction procedures? The simple answer is that goal-reduction procedures are one way of using the connections.

The first sentence of the Emergency Notice as part of a Logic Program

The first sentence of the Notice, written in the form of a goal-reduction procedure, hides its underlying logical form. In general, *goal-reduction procedures* of the form:

Reduce goal to sub-goals

hide *conditionals* of the form:

Goal if sub-goals.

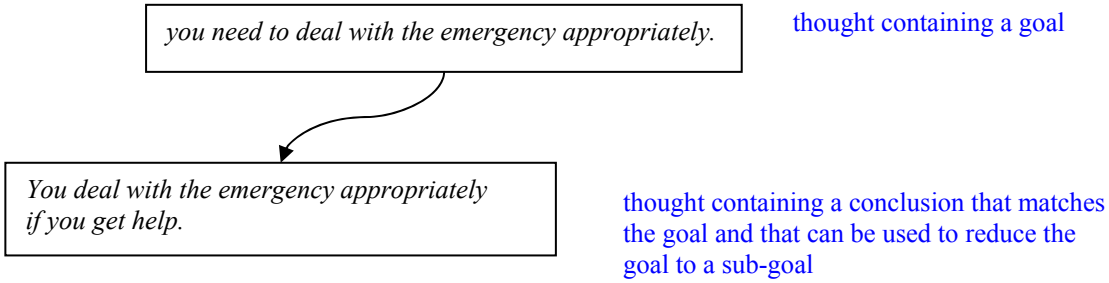
The goal-reduction behaviour of procedures can be obtained from conditionals by *backward reasoning*:

To conclude that the goal can be solved,
show that the sub-goals can be solved.

Thus, the first sentence of the Emergency Notice has the hidden logical form:

You alert the driver,
if you press the alarm signal button.

Viewed in connection graph terms, backward reasoning is one way in which a thinking agent can use links between conditionals to direct its attention from one thought to another. It directs its attention from a thought containing a condition that represents a goal to another thought, containing a conclusion that matches the goal. For example:



The use of backward reasoning to turn conditionals into goal-reduction procedures is the basis of *logic programming*. Logic programming became popular in Computing for a while in the 1980s, and then somewhat fell out of favour. One of the goals of this book is to show that logic programming can be useful, not only for Computing, but also for Human Reasoning.

Backward reasoning contrasts with *forward reasoning*, which is probably more familiar to most people. Given a conditional of the form:

If conditions then conclusion.

and a collection of statements that match the conditions, forward reasoning derives the conclusion as a logical consequence of the conditions. For example, given the statements:

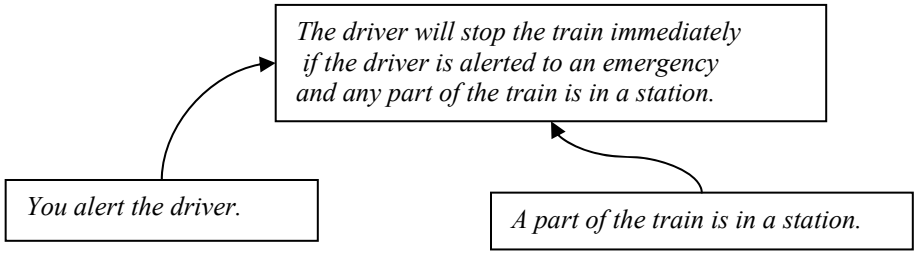
You alert the driver.
A part of the train is in a station.

forward reasoning uses the conditional:

The driver will stop the train immediately
if the driver is alerted to an emergency
and any part of the train is in a station.

to derive the conclusion that *the driver will stop the train immediately.*

Viewed in connection graph terms, forward reasoning directs attention from the conclusions of thoughts to a thought whose conditions are linked to those conclusions. For example:



Whether and when to use backward or forward reasoning is one of the main topics of this book.[7]

The fourth sentence of the Emergency Notice as a constraint

In natural language, the logical form of conditionals is often hidden below the surface, sometimes appearing on the surface in procedural form, at other times appearing in declarative form. For example, the last sentence of the Notice is a declarative sentence, which hides its underlying conditional form:

You may get a 50 pound penalty
if you press the alarm signal button improperly.

Backwards reasoning turns this conditional into a goal-reduction procedure:

To get a 50 pound penalty,
press the alarm signal button improperly.

It is very unlikely that a passenger would want to get a 50 pound penalty, and very unlikely, therefore, the passenger would want to use the conditional as a goal-reduction procedure. It is more likely that the passenger would use it to reason forward instead, to conclude that using the alarm signal button improperly would have an undesirable consequence. Thus, the sentence acts a constraint on action rather than as a motivator of action. This explains why the sentence is written declaratively and not procedurally.

[7] It is normal in formal logic to write conditionals in the forward direction: *If conditions then conclusion*. This is why reasoning from conditions to conclusions is called *forward reasoning*. In fact, conditionals can be used to reason both forwards and backwards, no matter how they are written. However, we often write them one way rather than the other when we have one preferred direction of use in mind.

In fact, only the first sentence of the Emergency Notice is written in procedural form, and only this first sentence of the Notice functions as a normal program, to evoke the behaviour that is desired of passengers on the underground. The fourth sentence functions as a constraint, to prevent undesired behaviour.

The second and third sentences, on the other hand, describe part of a program to be executed by a different agent, namely by the driver of the train. These two sentences are written declaratively and not procedurally precisely because they are to be executed by a different agent, and not by the agent observing the emergency. However, passengers can use the sentences to reason forwards to derive the likely consequences of pressing the alarm signal button.

Programs with purpose

It is implicit that the purpose[8] (or goal) of the Notice is to explain how you can get help from the driver in an emergency. That is why the third sentence includes a phrase that explains why the driver does not stop the train immediately when it is not in a station, but waits to stop until the next station:

> *where help can more easily be given.*

The Notice makes sense because the first sentence, in particular, coheres with the goals and beliefs that you probably had already before you started reading the Notice. For example, with such sentences as:

> *If there is an emergency then*
> *you need to deal with the emergency appropriately.*
>
> *You deal with the emergency appropriately if*
> *you get help.*
>
> *You get help if you alert the driver.*

Although I have deliberately written the second and third sentences here conclusion-first, because it is natural to use them conclusion-first, as a procedures for dealing with emergencies, I have written the first sentence condition-first, because it is natural to use it condition-first, to respond to emergencies.

Like the other sentences, the first sentence also has the form of a conditional. But its conclusion is effectively imperative (*deal with the emergency appropriately*) rather than declarative (*you will deal with the emergency appropriately*). We will see later that the first sentence can be written declaratively if it is regarded as a goal rather than as a belief.

The difference between goals and beliefs is that beliefs describe an agent's past observations and future expectations, whereas goals describe future states of the world that the agent actively seeks to achieve. The distinction between goals and beliefs has been somewhat neglected in symbolic logic, but is an important feature of the Computational Logic developed in Artificial Intelligence.

Where do we go from here?

This chapter has been intended to give an overall impression of the book as a whole. It shows how the meanings of English sentences can be viewed in both computational and logical terms; and it shows how the two views are combined in logic programming, which is a special case of computational logic, which is the topic of this book.

Traditional logic, on which computational logic is based, has fallen out a fashion in recent years, challenged by rival models of thinking, such as condition-action rules, connectionist neural networks, and Bayesian networks. One of the sub-tasks of this book is to argue that many of the features of these rivals are compatible with Computational Logic.

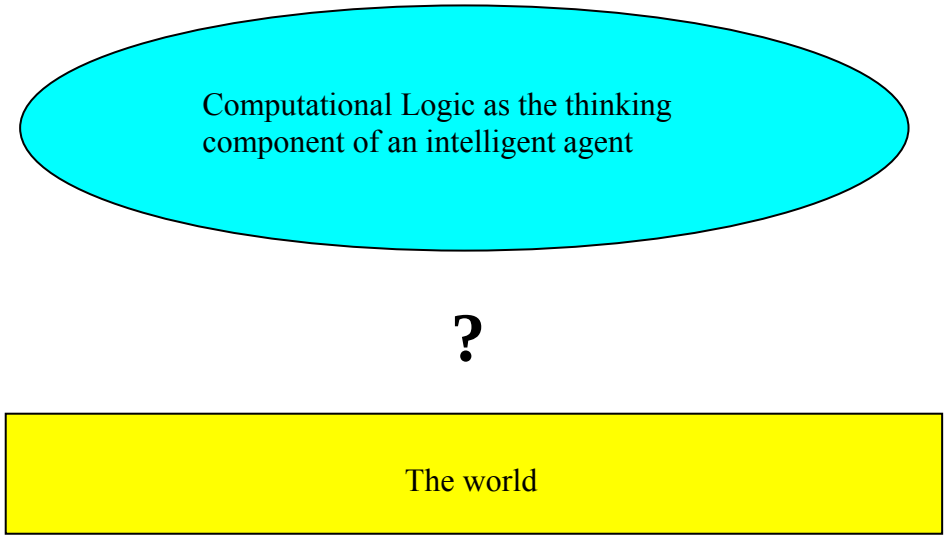
---

[8] The terms "goal" and "purpose" are interchangeable. Other terms that sometimes have the same meaning are "motivation", "reason", "interest", "desire", "objective" "mission", "target" etc.

Part of the problem with traditional logic is its lack of concern with a number of issues that are important in human thinking and behaviour. These issues include the need:

- to distinguish between goals and beliefs
- to be open to changes in the world
- to combine thinking about actions with actually performing them
- to combine the use of logic for thinking about actions with the use of probability and utility for deciding what actions to perform
- to reason by default and with rules and exceptions
- to combine arguments for and against a given conclusion.

We will see how Computational Logic addresses these issues in the following chapters. For the moment, we can roughly picture the problem we face like this:



Chapter 1a. The Case Against Logic

Suppose that in the interest of improving security, the management of the underground introduce a security check, during which the security officers stick a label with a letter from the alphabet to the front of every passenger entering the underground. Suppose that the security officers are supposed to implement the following conditional:

> *If a passenger is carrying a rucksack on his or her back,*
> *then the passenger is wearing a label with the letter A on his or her front.*

Imagine that you have the task of checking whether the security officers have properly implemented the conditional. Which of the following four passengers do you need to check? You can see only Bob and John's back and only Mary and Susan's front:

> *Bob, who is carrying a rucksack on his back.*
> *Mary, who has the label A stuck to her front.*
> *John, who is not carrying a rucksack on his back.*
> *Susan, who has the label B stuck to her front.*

Unfortunately, I haven't tried this test before. So I'm not entirely sure what to expect. But if you are like most ordinary people, and if the task I have asked you to perform is sufficiently similar to other experiments that many psychologists have devised, then it is likely that your performance will not be very logical.

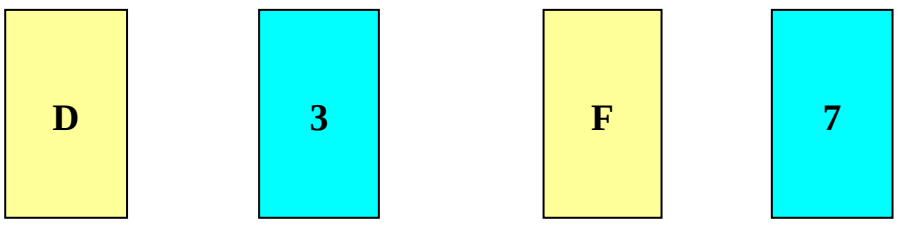
You will almost certainly check Bob, to make sure that he has the label A stuck to his front, which is the obviously correct thing to do; and you will probably not check John, which is also correct. However, if you are like most ordinary people, you will probably check Mary, even though there is no need to do so. But worst of all, you will probably fail to check Susan, even though she might be a terrorist carrying a rucksack on her back.

You might think that the psychologists who devise these experiments would be disappointed in the evidence that most people are apparently not very logical. But many of them seem to be absolutely delighted.

The Wason selection task

The first and most famous of these experiments was performed by Peter Wason[9] around 1968. It is cited by Stephen Pinker in his award winning book[10], *How the Mind Works*, as evidence that logic seems to have little to do with the way the mind really works.

In Wason's experiment, there are four cards, with letters on one side and numbers on the other. The cards are lying on a table with only one side of each card showing:



[9] Peter Wason, (1968) 'Reasoning about a rule', The Quarterly Journal of Experimental Psychology, 20:3, 273 - 281

[10] Stephen Pinker, 1997: *How the Mind Works.* New York: Norton.

The task is to select those and only those cards that need to be turned over, to determine whether the following conditional is true:

If there is a D on one side,
then there is a 3 on the other side.

Variations of this experiment have been performed numerous times, mainly with College students. The surprising result is that only about 10% of the subjects give the logically correct answer.

Almost everyone recognizes, correctly, that the card showing D needs to be turned over, to make sure there is a 3 on the other side. This is a logically correct application of the inference rule *modus ponens*, which is a variety of forward reasoning. Most people also recognise, correctly, that the card showing F does not need to be turned over.

But many subjects also think, incorrectly, that it is necessary to turn over the card showing 3, to make sure there is a D on the other side. This is logically incorrect, because the implication does not claim that *conversely*:

If there is a 3 on one side,
then there is a D on the other side.

The two implications are the *converse* of one another, in the same way that the two implications:

If it is raining, then there are clouds in the sky.
If there are clouds in the sky, then it is raining.

are also mutually converse. In fact, (in case it's not obvious) the first implication is true and the second implication is false.

However, more disturbingly, only a small percentage of subjects realise that it is necessary to turn over the card showing 7, to make sure that D is not on the other side. It is necessary to turn over the 7, because the original implication is logically equivalent to its *contrapositive*:

If the number on one side is not 3 (e.g. 7),
then the letter on the other side is not D.

Similarly, the second sentence in the pair of sentences:

If it is raining, then there are clouds in the sky.
If there are no clouds in the sky, then it is not raining.

is the contrapositive of the first sentence, and the two sentences are also logically equivalent.

The obvious conclusion, which many psychologists draw, is that people are not logical, and that logic has relatively little to do with real human reasoning.

A variant of the selection task with meaningful content

Psychologists have shown that people perform far better when the selection task experiment is performed with a problem that is formally equivalent to the card version of the task but that has meaningful content. The classic experiment of this kind considers the situation in which people are drinking in a bar, and the subject is asked to check whether the following conditional holds:

If a person is drinking alcohol in a bar,
then the person is over eighteen.

Again there are four cases to consider, but this time instead of four cards there are four people. We can see what two of them are drinking, but cannot see how old they are; and we can see how old two of them are, but not what they are drinking:

Bob, who is drinking beer.
Mary, who is a respected member of the senior citizens' community.
John, who is drinking cola.
Susan, who is an innocent looking child of primary school age.

In contrast with the card version of the selection task, most people solve the bar version correctly, realising that it is necessary to check Bob to make sure that he is over eighteen, and to check Susan to make sure that she is not drinking alcohol, but that it is not necessary to check Mary and John.

Cognitive psychologists have proposed a bewildering number of theories to explain why people are so much better at solving such meaningful versions of the selection task compared with the original card version. The most generally cited of these theories, due to Leda Cosmides[11], is that humans have evolved a specialized algorithm for detecting cheaters in social contracts. The algorithm has the general form:

If you accept a benefit,
then you must meet its requirement.

In the bar version of the selection task, the "benefit" is "drinking beer" and the "requirement" is "being over eighteen".

Cosmides and her co-workers also argue that humans have evolved other specialized algorithms for dealing with other kinds of problems, for example an algorithm for avoiding hazards:

If you engage in a hazardous activity,
then you should take the appropriate precaution.

Cheng and Holyoak[12], at about the same time, put forward a related theory, that people typically reason about realistic situations using pragmatic reasoning schemes, involving such notions as permission, obligation and causation. Chief among these are schemes involving *deontic* notions concerned with permission, obligation and prohibition. In English these notions are typically signalled by the use of such words as "can", "should", "need" and "must". But these explicit linguistic signals may be omitted if the context makes it obvious that an obligation or prohibition is involved, as in the way I have formulated of the bar version of the selection task above.

Both Cosmides and Cheng and Holyoak argue that people do not have an in-built, general-purpose capability for abstract logical reasoning, but that they have instead specialised procedures for dealing with classes of practical problems that arise naturally in the world around them.

Computational logic and the selection task

I will argue in this book that Computational Logic is compatible with many of the proposals that have been put forward to explain the way people reason in general, and the way people reason with variations of the selection task in particular. I will sketch my argument regarding the selection task in this chapter. However, the argument appeals to characteristics of Computational Logic that will be investigated in greater detail later in the book. The remainder of this chapter is not necessary for the rest of the book, and the reader who finds it too hard going or not entirely convincing can safely skip it for now.

In the case of the selection task, there are two main problems: how to explain illogical performance on "abstract" or meaningless versions of the task, and how to explain "logical" performance on

[11] Leda Cosmides. 1985 Deduction or Darwinian algorithms : an explanation of the "elusive" content effect on the Wason selection task. Ph.D. thesis. Harvard University.
Leda Cosmides. 1989. The logic of social exchange: has natural selection shaped how humans reason? Studies with the Wason selection task. Cognition **31**, 187 -276.
[12] Patricia W. Cheng and Keith J. Holyoak, (1985). Pragmatic reasoning schemas. *Cognitive Psychology*, **17**, 391-416.

meaningful versions. I put "logical" here in quotes, because many psychologists would deny that "logical" performance has anything to do with logic.

Meaningless versus meaningful problems. Although Computational Logic, like any other tool in Artificial Intelligence, is often applied to artificial, meaningless, abstract puzzles, its ultimate aim is to help an agent solve useful problems.

If Computational Logic is to serve a similar function in Human Intelligence, then its purpose must similarly be to help people deal more effectively with meaningful situations that are relevant to their own goals and experiences. Many of the experiments with the selection task show that people perform better with those variations, like the bar version of the task, which have meaningful content, than they do with those variations, like the card version, which are meaningless.

Like any other human biological or cultural characteristic that has evolved over time, a human form of Computational Logic might have evolved in response to evolutionary pressures and encouragements. If evolution can be invoked to explain the emergence of specialised algorithms, it can just as easily be invoked to explain the emergence of general-purpose logical methods.

Confusion about special-purpose versus general-purpose methods. Many of the theories invoked to explain human performance on the selection task, like those of Cosmides and Cheng and Holyoak, reject the notion that people use any form of logic in reasoning, and propose that people use heuristics, schemes and algorithms instead. There is a similar tradition in Artificial Intelligence of distinguishing between so-called *weak*, general-purpose problem-solving methods like logic and search, and *strong*, special-purpose methods like heuristics, schemes and algorithms. This distinction is commonly invoked to argue that logic is too general and too weak to be useful for problem-solving, and that heuristics, schemes and algorithms are stronger and more useful.

However, this way of formulating the distinction between *weak* and *strong* methods fails to recognise the relationship between them and creates both confusion and a false sense of conflict. The relationship is hidden inside the formula:

algorithm = knowledge + reasoning.

which emphasises the more fundamental distinction, not between *weak* and *strong*, but between *knowledge* (or better *goals* and *beliefs*) and *reasoning* (or *problem-solving*).

The contrast between *weak* and *strong* methods is a false contrast between different categories. The term *strong method* applies to algorithms, whereas the term *weak method* applies both to general-purpose problem-solving methods, as well as to general-purpose knowledge. The contrast between *weak* and *strong* methods fails to recognise that both knowledge and reasoning can be weak or strong.

Knowledge is weak when it is able to solve any problem in a given domain in general, but when it is not especially useful for solving any problem in particular. In contrast, strong knowledge is knowledge that is well-tuned to problems and situations that arise commonly in practice.

Reasoning is weak when it is general-purpose, like backward and forward reasoning, and when it does not involve very much sophistication. Strong reasoning methods are also general-purpose, but they involve more sophisticated methods involving such techniques as "intelligent backtracking", "heuristic search" and "constraint processing"[13].

Algorithms, which combine knowledge and reasoning, therefore, can be *weak + weak*, *weak + strong*, *strong + weak*, or *strong + strong*. Strong heuristics, schemes and algorithms can typically be decomposed into *strong + weak*, or *strong + strong* combinations of knowledge and reasoning methods. In these combinations, it is less important whether reasoning is weak or strong, and more

[13] Both weak and strong logic-based problem-solving methods are described in: Robert Kowalski. *Logic for Problem Solving*, Elsevier North Holland, 1979. A simple example, given on page 93, is the contrast between the weak method of solving sub-goals of a problem in a fixed order, and the strong method, which procrastinates attempting to solve explosive subgoals, which have many candidate solutions.

important that knowledge is strongly oriented towards frequently occurring problems whose solution has high utility.

The relationship between knowledge and reasoning has been studied and exploited extensively in logic programming[14], where the relationship can be expressed by the formula:

algorithm = beliefs expressed as conditionals + backward reasoning.

In this book, we will see how Computational Logic extends logic programming, mainly by employing other kinds of reasoning, including forward reasoning, and by employing other kinds of knowledge, including conditional goals. The important point, to draw attention to here, is that the use of *strong*, special-purpose heuristics, schemes and algorithms is entirely compatible with the use of general-purpose, logical reasoning, provided logical reasoning is combined with pragmatically useful goals and beliefs.

Reasoning with goals and beliefs. So far in the first Chapter of this book, we focused on the use of logic to represent *beliefs,* which aim to *describe* the way things are. However, as we will see later, arguably a more fundamental use of logic is to represent *goals*, which *prescribe* the way things ought to be. In Computational Logic, an agent uses its goals and beliefs in different ways. In dealing with observations, it typically uses its goals to generate actions to make its goals true in the light of its observations, and it uses its beliefs to help it to achieve its goals, assuming that its beliefs are already true. These two uses of logic require different kinds of reasoning. Beliefs require mainly just backward and forward reasoning, whereas goals require reasoning that is more like that associated with classical logic.

Despite the declarative surface form in which I have formulated the bar version of the selection task above, the content of the task makes it clear that it should be interpreted prescriptively, as a conditional form of goal:

If a person is drinking alcohol in a bar,
then the person ***must be*** *over eighteen.*

This is because the sentence is obviously intended to prescribe the way people should behave rather than to describe how they actually behave.

Reasoning with conditionals interpreted as beliefs. In contrast, it is not natural to interpret the card version of the selection task as a goal, because there is no obvious agent to associate with the goal of making the conditional true. It is more natural to interpret it as a belief. We will see later in the book that, in Computational Logic, beliefs are used mainly to reason backwards or forwards, and they inhibit reasoning with their contrapositives. This might explain why people have trouble reasoning with contrapositives when conditionals are interpreted as beliefs.

In logic programming, it is also natural to assume that *all* the beliefs having the same *conclusion*:

conclusion if conditions$_1$ …… *conclusion if conditions*$_n$

specify the *only conditions* under which the *conclusion* holds. In other words, the conditionals are interpreted as the *biconditional*:

conclusion if and only if conditions$_1$ **or** …… **or** *conditions*$_n$

This interpretation of conditionals as biconditionals is sometimes called the *closed world assumption* and is used to justify so-called *negation as failure*, one of the main ways of both formalising and implementing *default reasoning*. Default reasoning is an important feature of human reasoning and has been a major focus of attention in Artificial Intelligence.

---

[14] Robert Kowalski, "Algorithm = Logic + Control", in *CACM,* Vol. 22, No. 7, 1979, pp. 424-436. Reprinted in *Programming Languages: A Grand Tour,* Third Edition, (ed. E. Horwitz), Computer Science Press, Maryland, 1986, pp. 480-492.

We will return to the topic of default reasoning later. But in the meanwhile, we note that the closed world assumption might also explain why, when a single conditional:

conclusion if conditions

is interpreted as a belief, people often assume that the converse of the conditional holds:

conclusion if and only if conditions.

Therefore, taking them together, the inhibition of reasoning with contrapositives and the closed world assumption, these two features of logic programming might explain the two apparent mistakes that people make with card variants of the selection task. In both cases the mistake is not that people are not logical, but rather that our judgement of their performance fails to distinguish between the logic of beliefs and the logic of goals. A similar argument has also been made by Stenning and van Lambalgen[15].

Reasoning with conditionals interpreted as goals. In Computational Logic, goals have several uses. As we will see in subsequent chapters, their primary use is to represent condition-action rules and more generally to represent condition-goal rules, such as:

If there is an emergency then
*you **need** to deal with the emergency appropriately.*

But goals also include prohibitions and constraints, as illustrated by the following injunction not to misuse the alarm signal button:

Do not use the alarm signal button improperly.

In Computational Logic, it is often useful to represent denials *it is not the case that conditions* in conditional form *if conditions then false.* In particular, it is useful to represent prohibitions as conditional goals with conclusion *false.* In this example, the injunction not to misuse the alarm signal button can be represented as the conditional goal:

If you misuse the alarm signal button then false.

This representation of prohibitions as conditional goals with conclusion *false* has the advantage that the same pattern of reasoning that applies to other kinds of conditional goals also applies to prohibitions. In general, given a conditional goal of the form:

If conditions, then conclusions.

when one of the conditions is triggered by some real or hypothetical observation,
determine whether the other conditions also hold,
and if they do, derive the conclusions as a goal to be solved.

This form of reasoning is well suited to condition-action rules and condition-goal rules, to derive appropriate actions and goals, in response to changes in the environment. It is also tailor-made to monitor observations for violations of prohibitions, when they are represented in conditional form:

If conditions then false.

when one of the conditions is triggered by some real or hypothetical observation,
determine whether the other conditions also hold,
and if they do, conclude that the prohibition has been violated.

Applied to the bar version of the selection task represented in the form:

[15] Stenning and van Lambalgen 2008.

If a person is drinking alcohol in a bar,
and the person is eighteen or under
then false.

this general pattern of reasoning monitors all and only those observations that match one of its two conditions, namely observations of people drinking alcohol and of people who are eighteen years old or younger. This is consistent both with the dictates of classical logic and with the results of psychological experiments. Moreover, it is supported by the arguments and experiments of Sperber et al[16] showing that the more natural it is for people to interpret the natural language statement of the conditional:

If conditions then conclusion.

as a denial:

It is not the case that conditions and not-conclusion

the more readily they solve the selection task in conformance with the rules of classical logic. Here I have written what should have been a negative condition *not conclusion* as a condition *not-conclusion* intended to be understood positively.

Sperber et al show that logically correct reasoning with the selection task depends in large part upon the extent to which the negation of the conclusion of the conditional has a natural positive counterpart. In the bar version of the selection task, the negation *it is not the case that the person is over eighteen* has the natural positive counterpart *the person is eighteen or under*. Similarly, the negation of such notions as *tall*, *good*, and *young* have natural positive counterparts such as *short*, *bad*, and *old*. In contrast, there is no natural positive counterpart to such negative notions as *a number is not 3*, *a person is not a sailor* or *a book is not 201 pages long*.

The reason why it is important whether or not a negative notion has a natural positive counterpart is that observations are positive and not negative. We directly observe, for example, the positive fact that a person is a child and therefore is under eighteen years old. Similarly, we directly observe the positive fact that a card has the number 7 showing on one side, but we need to conclude that it does not have the number 3 showing.

Indeed, even if it were possible to phrase the card version of the selection task so that it was natural to interpret the conditional as a denial, it would still be hard to conclude that 3 is not showing if all we observe is the positive fact that 7 is showing. This is because, given the observation, we could equally well conclude that the number 1 is not showing, the number 2 is not showing, …. etc. It is only because we are asked to test a rule referring to the number 3, that we are able to make the mental leap from "7" to "not 3". The difficulty of making this leap may be yet another reason why so many people fail to choose the card showing the number 7 in the card version of the selection task.

In general, the harder a human or artificial agent needs to work to derive a conclusion, the less likely it is that the agent will be able to so. This may help to explain, not only some of the problems that people have with certain variants of the selection task, but also some of the differences between the way people would think if they had unlimited time and energy, and the way they actually think with their limited resources in practice. In particular, it may help to explain how people may fail to apply logic themselves, yet still recognize a logically correct solution when they see it.

Natural language versus formal problem solving. This analysis of the selection task supports the view that the solution of any problem communicated in natural language is a two-stage process. In the first stage, the recipient of the communication attempts to interpret the natural language description of the task to determine its intended meaning and to assimilate that meaning into the recipient's own goals and beliefs. In the second stage, the recipient attempts to solve the resulting logical form of the interpretation of the problem. A failure to solve the problem in the eyes of the communicator may be due, therefore, either to the recipient's failure to interpret the problem as the comunicator intended, or

[16] Sperber, D., Cara, F., & Girotto, V. (1995). Relevance theory explains the selection task. *Cognition, 52*, 3-39.

to the recipient's failure to solve the correctly interpreted problem. As we have seen, both of these kinds of failure can arise in the selection task.

Sperber et al argue, there is so much variation possible in the first stage of the selection task, that it is impossible to form any judgement about the correctness of the reasoning processes involved in the second stage. This view is also supported by the results of experiments performed by Amit Almor and Steven Sloman[17], who showed that, when subjects are asked to recall the problem after they have solved the task, they report a problem statement that is consistent with their solution instead of with the original natural language problem statement.

Does Computational Logic explain human performance on the selection task? If the selection task were presented to an artificially intelligent agent using Computational Logic to guide its thoughts and behaviour, the agent would need first to decide whether to interpret the conditional as a goal or as a belief. It would use both the syntax of the natural language statement of the problem and its own background goals and beliefs, to guide it in the interpretation process.

If the agent interprets the conditional as a belief, then it would reason with the belief using backward or forward reasoning, but not with contrapositive reasoning. If, in addition, the agent interprets the conditional as the only conditional relevant to establishing its conclusion, then it would interpret it as a biconditional. This interpretation of the conditional as a biconditional and the inhibition from using it for contrapositive reasoning would mirror the way people reason with conditionals of the kind involved in the card version of the selection task.

However, if the agent interprets the conditional as a goal, then the biconditional interpretation would not apply. Moreover, if the conditional has a natural interpretation as a prohibition and if the negation of the conclusion has a natural positive counterpart, then the agent's internal representation of the conditional would facilitate monitoring both the condition of the original natural language conditional and the positive counterpart of the negation of the conclusion. This interpretation of the conditional would mirror the way people reason with conditionals of the kind involved in the bar version of the selection task.

To be conclusive, these arguments would need to be tested with many other examples and would need to be compared with many other theories. But even as the arguments currently stand, they support the thesis, not only that Computational Logic might explain human performance on the selection task, but that it might also approximate the logic of the language of human thought more generally.

Other problems with conditionals

In later versions of the book, I intend to explore other relationships between conditionals in natural language and conditionals in the language of thought. In the meanwhile, a discussion of some of these relationships can be found in the attached appendix "Reasoning with Conditionals in Artificial Intelligence". Here is a preview:

Truncation of conditionals. Several authors[18] have observed that conditionals in natural language often have a *truncated form*:

if conditions then conclusion

---

[17] Amit Almor and Steven Sloman. Reasoning versus text processing in the Wason selection task: A non-deontic perspective on perspective effects. *Memory & Cognition 2000, 28 (6), 1060-1070.*

[18] For example: Stenning, K. and van Lambalgen M., (2008) *Human reasoning and cognitive science.* MIT Press. Bonnefon, J.-F. and Politzer, G. (2009) Pragmatic Conditionals, Conditional Pragmatics, and the Pragmatic Component of Conditional Reasoning. To appear in The Psychology of Conditionals (ed. M. Oaksford) OUP. Kowalski, R. "Reasoning with Conditionals in Artificial Intelligence" to appear in The Psychology of Conditionals (ed. M. Oaksford) OUP, to appear 2009.

which, to be interpreted correctly, needs additional conditions to be made explicit:

if conditions ***and other conditions*** *then conclusion.*

For example, each of the following conditionals has at least one missing condition:

If any part of the train is in a station, then the driver will stop the train in the station.
If an animal is a bird then the animal can fly.
If she has an essay to write, then she will study late in the library.

We encountered the first conditional in the first chapter, where we saw that it is missing the condition *the driver is alerted to an emergency*, which is implicit from the context of the previous sentence.

The second conditional is a conditional representation of the most famous of all examples in Artificial Intelligence, the sentence *all birds fly*. Here the missing conditional is something like:

the animal is a typical bird
or *the animal is not an abnormal bird*, etc.

The third conditional is part of an experiment used by the psychologist Ruth Byrne[19] to support the argument that people do not reason with inference rules (like forward and backward reasoning) but use mental models instead. I leave it to the reader to decide which additional condition(s) might be missing.

Missing conditions may give the impression that a person does not reason logically. We will see some examples of this shortly. But first, consider an even more difficult problem.

Reversal of conditionals. What do you conclude from the following two premises?

If an object looks red, then it is red.
This apple looks red.

Few people would criticise you if you concluded *this apple is red*. A straight-forward application of modus ponens or forward reasoning. Indeed, this is a good example of what formal logic is all about: concentrating on the form of sentences, ignoring their content.

But suppose I now add:

If an object is illuminated by a red light, then it looks red.

Psychological experiments with similar examples suggest that you would probably suppress the application of modus ponens, withdrawing your previous conclusion *this apple is red*. The philosopher John Pollock[20] uses this example to explain such withdrawing of conclusions as a sophisticated form of argumentation, in which the second conditional supports an argument that defeats the argument supported by the first conditional.

However, there is another way of looking at the problem: Solving the problem is a two-stage process. In the first stage, the natural language statement of the problem needs to be translated into a logical form, using its surface syntax, the context of earlier and later sentences, and the content of any relevant background goals and beliefs. In the second stage, the resulting logical form needs to be processed using appropriate content-independent, formal reasoning methods.

In this example, the third sentence adds additional content that is relevant to the interpretation of the first sentence. In particular, the content of the third sentence suggests that the first sentence is expressed the wrong way around. Understood in every-day, common-sense terms, it is an object's being red or being illuminated by a red light that causes it to look red.

---

[19] Ruth M. J. Byrne, Suppressing valid inferences with conditionals. *Cognition* **31** (1989), 61–83.
[20]

The natural way to logically represent the relationship between *cause and effect* is in the form: *if causes then effects*. The third sentence represents causality in this way, but the first sentence represents it in the opposite, converse way. The natural way to represent the relationship between being red and looking red is therefore:

an object looks red if it is red.
an object looks red if is illuminated by a red light.

Given only the first of these two pairs of beliefs, it is natural to interpret the first conditional as the biconditional:

an object looks red if and only if it is red.

Together with the additional premise *this apple looks red*, the biconditional implies *this apple is red*.

However, given both pairs of beliefs, it is natural to interpret them together as the revised biconditional:

an object looks red if and only if
it is red ***or*** *it is illuminated by a red light.*

Now, together with the premise *this apple looks red*, the revised biconditional implies *this apple is red or this apple is illuminated by a red light*, withdrawing the previous conclusion.

Now for an example that combines truncation of conditionals with the reversal of conditionals.

Problems with rain. Which of the following two conditionals makes more sense?

If it is raining, then there are clouds in the sky.
If there are clouds in the sky, then it is raining.

Obviously, the first conditional is true and the second is false. So you can be excused for thinking that there isn't much of a contest between them – after all, true sentences make more sense than false sentences. But consider the possibility that the intended meaning of the second sentence is actually:

If there are clouds in the sky and ***the conditions in the clouds are right for rain****,*
then it is raining.

In other words, the surface structure of the second sentence is a truncated form of its real meaning. With the added, previously missing, extra condition, the second sentence is now also true. Moreover, compared with the first sentence, it is expressed in the more natural *if causes then effects* form.

In fact, in logic programming the first sentence is a logical consequence of the revised second sentence. This is because, given what most people believe about clouds and rain, the second sentence expresses the only conditions under which the conclusion holds. So, in logic programming, the second sentence is interpreted as a biconditional:

it is raining if and only if
there are clouds in the sky and the conditions in the clouds are right for rain.

The first sentence is now just part of the other half of the biconditional. So, arguably, the revised second sentence, with its intended meaning completely filled in, is more meaningful (literally "full of meaning") than the first sentence, because the revised second sentence now logically implies the first.

The Byrne suppression task. Now here is the original suppression task, as presented by Ruth Byrne: Suppose I tell you:

If she has an essay to write, then she will study late in the library.
She has an essay to write.

According to the experimental day, approximately 90% of you will draw the formally correct conclusion that *she will study late in the library.* If I now say:

If the library is open, then she will study late in the library.

then about half of you will probably withdraw your previous conclusion, suppressing your previous application of modus ponens.

How would you explain these results? I don't want to spoil it for you by giving my own explanation. You probably know by now what I am likely to say.

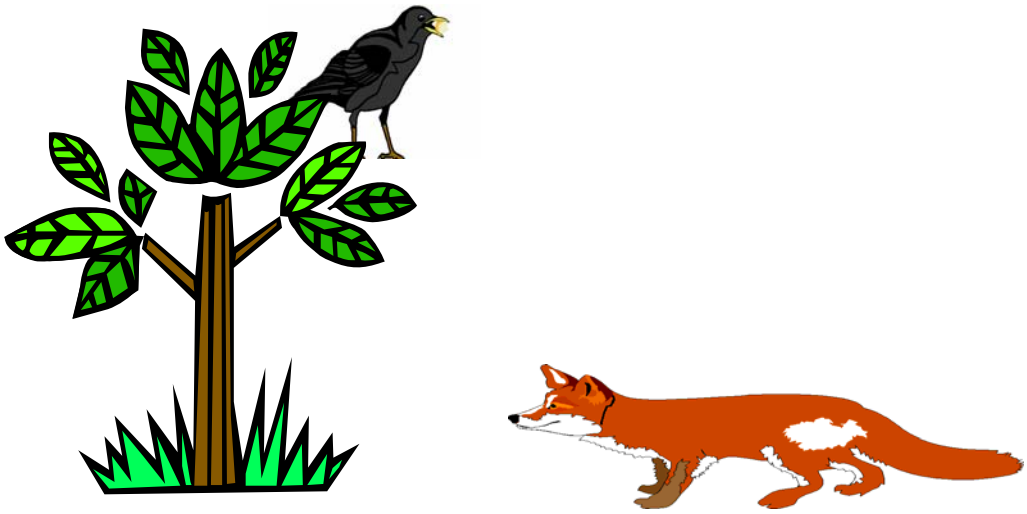
Chapter 2 The Fox and the Crow

If the previous chapter or two were a little heavy going, the next few chapters are going to try to make up for it. In this chapter we retell the story of a proactive fox, who outwits a reactive crow, taking the fox's point of view of what it means to be proactive. In the next chapter, we focus on reactive thinking; and in the chapter after that on putting the two kinds of thinking together.

The fox and the crow are a metaphor for different kinds of people. Some people are **proactive**, like the fox in the story. They like to plan ahead, foresee obstacles, and lead an orderly life. Other people are **reactive**, like the crow. They like to be open to what is happening around them, take advantage of new opportunities, and to be spontaneous. Most people are both proactive and reactive, at different times and to varying degrees.

The fox and the crow

Probably everyone knows the ancient Greek fable, attributed to Aesop, about the fox and the crow. It starts, harmlessly enough, with the crow sitting in a tree with some cheese in its beak, when along comes the fox, who wants to have the cheese.



In this version of the story, we consider the fox's point of view. To model her proactive way of thinking, we represent her goals and beliefs in logical form:

Goal *I have the cheese.*

Beliefs *The crow has the cheese.*

An animal has an object
if the animal is near the object
and the animal picks up the object.

I am near the cheese
if the crow has the cheese
and the crow sings.

The crow sings if I praise the crow.

As you can see, the fox is not only a logician, but also an amateur physicist. In particular, her belief about being near the cheese if the crow sings combines in a single statement her knowledge about her location with her knowledge of the laws of gravity. Reasoning informally:

> The fox knows that if the crow sings, then the crow will open its beak and the cheese will fall to the ground under the tree. The fox also knows that, because the fox is under the tree, the fox will then be near the cheese.
> Therefore, the fox knows that she will be near the cheese if the crow sings.

The fox is also an amateur behavioural psychologist. Being a behaviourist, she is interested only in the crow's external, input-output behaviour, and not in any internal methods that the crow might use to generate that behaviour. In particular, although the fox represents her own beliefs about the crow in logical terms, she does not assume that the crow also uses logic to represent any beliefs about the world. As far the fox is concerned, the crow's behaviour might be generated by means of condition-action rules without logical form. Or his behaviour might be "hardwired" directly into his body, without even entering into his mind.

Like the fox's belief about being near the cheese if the crow sings, the fox's belief about the crow's behaviour might be derived from other, separate beliefs – perhaps from more general beliefs about the way some naive, reactive agents respond to being praised, without thinking about the possible consequences of their actions.

The fox also has ordinary common sense. It knows that an animal will have an object if it is near the object and picks it up. It knows this as a general law, which applies universally to any animal and to any object (although it doesn't seem to know that the law also applies to robots, unless it views robots as a species of animal). It also knows enough logic to be able to instantiate the general law and to apply it to the special case where the fox is the animal and the cheese is the object.

The fox's beliefs as a Logic Program

The fox's beliefs have not only logical form, but they also have the more specialised form of a logic program. As we have already seen, a logic program is a collection of implications of the form:

Conclusion if Conditions.

Both the conclusion and the conditions are written in declarative form.

The implications are written backwards, conclusion first, to indicate that they can be used to reason backwards, from conclusions to conditions. As a consequence of backward reasoning, each such implication behaves as a goal-reduction procedure[21]:

To derive the Conclusion, derive the Conditions.

Even "facts", which record observations, like the belief that the crow has the cheese, can be viewed as implications that have a conclusion, but no conditions:

Conclusion if nothing.

Such facts also behave as procedures:

To derive the Conclusion, do nothing.

Therefore, the fox's beliefs can be used as a collection of procedures:

[21] Grammatically speaking, the goal of the procedure is expressed in the subjunctive mood and the sub-goals are expressed in the imperative mood. Implications in logic, on the other hand, are expressed purely in the declarative mood.

To have an object,
be near the object
and pick up the object.

To be near the cheese,
check that the crow has the cheese
and make the crow sing.

To make the crow sing,
praise the crow.

To check that the crow has the cheese,
do nothing.

These procedures can be applied, one after the other, to reduce the top-level goal:

I have the cheese.

to the two action sub-goals:

I praise the crow and I pick up the cheese.

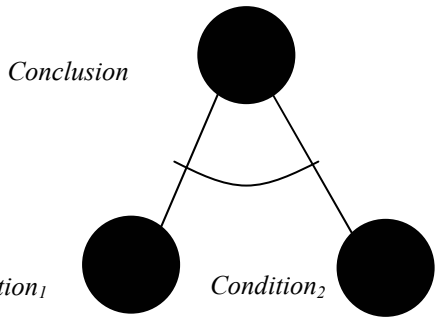
Together, these two actions constitute a plan for achieving the original goal.

Goal-reduction graphs

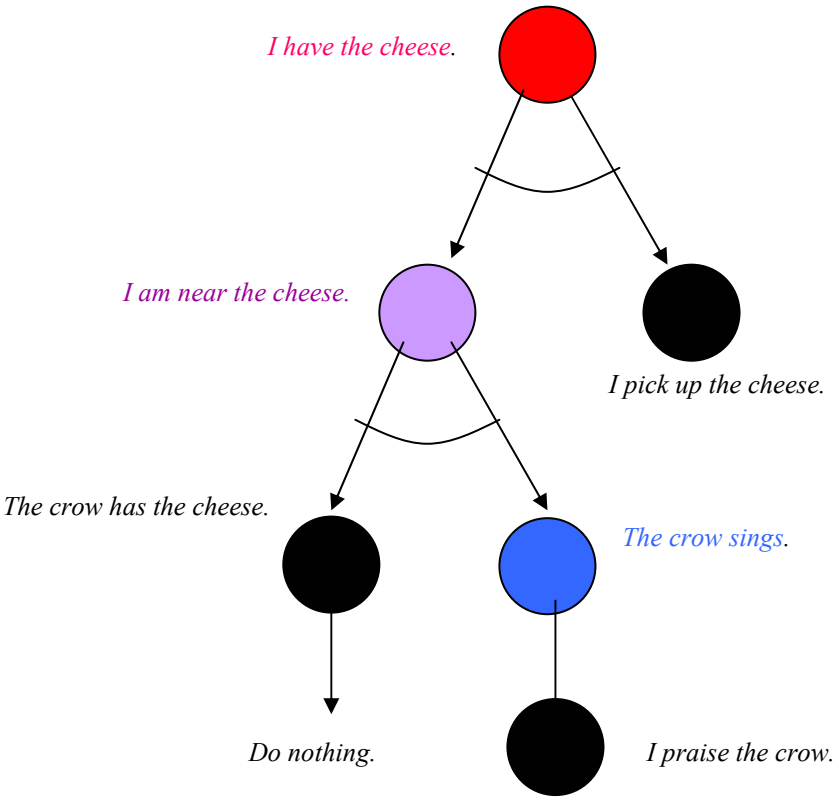
The fox's reduction of her original goal to the two action sub-goals can be visualized as a graph, in which implications of the form:

Conclusion if $Condition_1$ *and* $Condition_2$

are represented by sub-graphs of the form:



The graph has the form of an upside-down tree with the top-level goal at the top of the upside-down tree:



For the fox to solve the top-level goal, it suffices for her to grow the tree, starting from the top down, reducing goals to sub-goals, terminating when no further reduction is possible. If all the sub-goals at the "leaves" of the tree are irreducible action sub-goals, then these actions constitute a plan for solving the top-level goal.

Backward reasoning

The operation of reducing a goal to sub-goals can also be viewed in logical terms, as reasoning backwards with an implication, matching the goal with the conclusion of the implication and deriving the conditions of the implication as sub-goals.

For example, the top-level goal:

I have the cheese.

matches the conclusion of the general implication:

An animal has an object
if the animal is near the object
and the animal picks up the object.

Backward reasoning derives the two sub-goals:

I am near the cheese and I pick up the cheese.

by substituting the specific terms "I" and "the cheese" for the more general terms "the animal" and "the object" respectively.

The second of these two sub-goals is an action, which matches the conclusion of no implication. It can be solved only by performing it successfully. However, the first sub-goal can be reduced to other sub-goals by three further steps of backwards reasoning.

The final result of this chain of backward reasoning is a logical proof that the fox has the cheese if she praises the crow and picks up the cheese. The proof has the same tree-like structure as the goal-reduction graph we saw before.

In traditional logic, it is more common to present proofs in the forward direction. In this case, a traditional proof would look more like this:

I praise the crow.
Therefore *The crow sings.*

The crow has the cheese.
Therefore *I am near the cheese.*

I pick up the cheese.
Therefore *I have the cheese.*

The end of story?

For a Logic Extremist, this would be the end of the story. For the Extremist, there is no difference between the fox's world and the fox's beliefs about the world, and no difference between the fox's plan for getting the cheese and the fox's actually having it.

However, Common Sense tells us that there is more to life than just thinking – and perhaps even more to thinking than just logic alone. In addition to thinking, an intelligent agent needs to observe changes in the world and to perform actions to change the world in return. And there might be other ways of thinking - ways that do not use Logic and perhaps even ways that do not use any mental representation of the world at all.

We will come back to the story and address these issues in the next few chapters. But first:

What is the moral of the story?

Presumably Aesop's fable had a purpose – a lesson that it is not safe to take another agent's words and actions at face value, without trying to understand the agent's underlying goals and intentions. Or, even more simply, that before you do something you should think about its possible consequences.

The crow in our story responds to the fox's praise spontaneously - without thinking, you might say. A more intelligent crow would monitor his intended actions, before they are performed, to determine whether they might have any unintended and undesirable consequences.

If only the crow knew what the fox knows, the crow would be able to reason as follows:

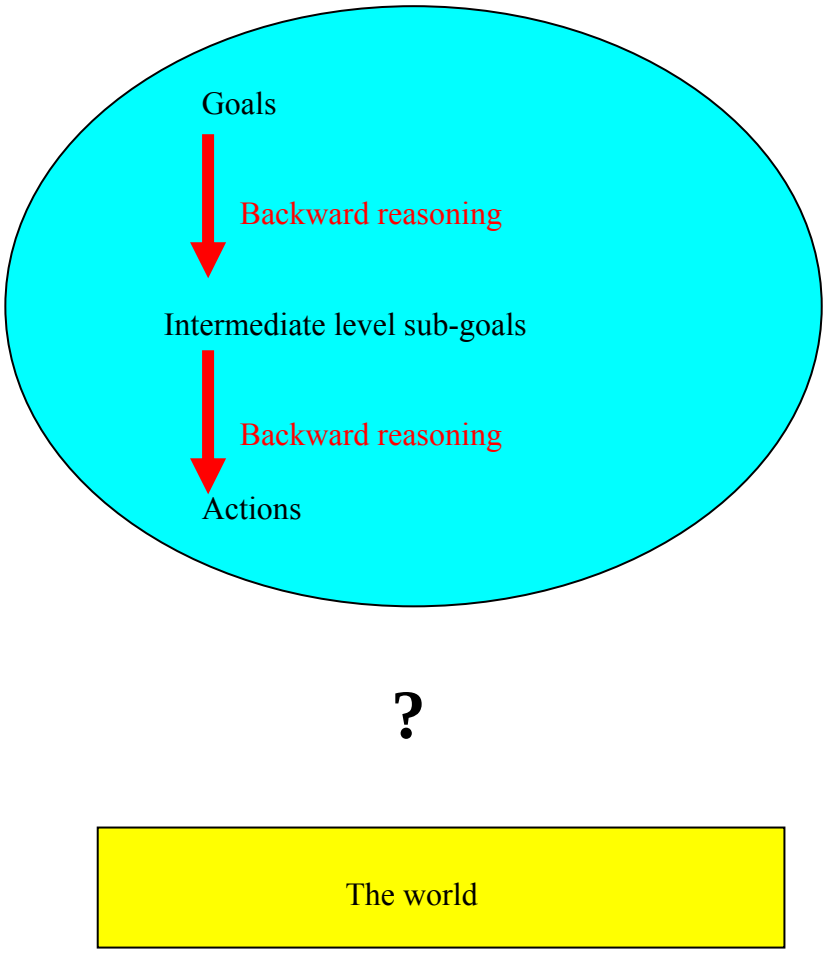
I want to sing.
But if I sing, then the fox will be near the cheese.
If the fox is near the cheese and picks up the cheese,
then the fox will have the cheese.
Perhaps the fox wants to have the cheese and therefore will pick it up.
But then I will not have the cheese.
Since I want to have the cheese, I will not sing.

Notice that this line of reasoning uses some of the same beliefs as those used by the fox, but it uses them forwards rather than backwards. We will investigate this dual use of beliefs for both backward and forward reasoning in the chapter after next.

In the meanwhile, we note that, although using logic might not always be the most natural way of thinking, it can sometimes help us (and the crow) to think and behave more effectively.

Summary

The view of computational logic in the mind of an intelligent agent, as seen so far in this chapter, looks something like this:



This picture will be elaborated considerably in the following chapters.

Chapter 3. The Louse and the Mars Explorer

Logical Extremism, which views life as all thought and no action, has given Logic a bad name. It has overshadowed its near relation, Logical Moderation, which recognises that Logic is only one way of thinking, and that thinking isn't everything.

The antithesis of Logical Extremism is Extreme Behaviourism, which denies any Life of the Mind and views Life instead entirely in behavioural terms. Extreme Behaviourism, in turn, is easily confused with the condition-action rule model of thinking.

Behaviourism

If you were analysing the behaviour of a thermostat, which regulates the temperature of a room by turning the heat on when it is too cold and turning it off when it is too hot, you might ***describe*** the thermostat's input-output behaviour in condition-action terms:

> If the current temperature is T degrees
> and the target temperature is T' degrees
> and T < T' - 2°
> then the thermostat turns on the heat.
>
> If the current temperature is T degrees
> and the target temperature is T' degrees
> and T > T' + 2°
> then the thermostat turns off the heat.

But you wouldn't attribute the thermostat's behaviour to a mind that manipulates such descriptions to ***generate*** its behaviour.

In the same way that you would view the thermostat's external behaviour without committing yourself to a view of its internal operation, the behaviourist views agents in general.

Thus, in the story of the fox and the crow, a behaviourist, unable to examine the fox's internal, mental state, would view the behaviour of the fox in the same way that we view the behaviour of the thermostat:

If the fox sees that the crow has cheese, then the fox praises the crow.
If the fox is near the cheese, then the fox picks up the cheese.

The behaviourist's description of the fox begins and ends with the fox's externally observable behaviour. The behaviourist justifies her refusal to attribute any internal, mental activity to the fox, by the fact that it is impossible to verify such attributions by the scientific method of observation and experimentation.

According to the behaviourist, the fox is a purely reactive agent, simply responding to changes in the world around her. If, in the course of reacting to these changes, the fox gets the cheese, then this result is merely an indirect, emergent effect, rather than one that the fox deliberately brings about by proactive, goal-oriented reasoning.

The behaviourist also sees no reason to distinguish between the behaviour of a thermostat and the behaviour of a human. The behaviourist might use an implication:

> If a passenger observes an emergency on the underground,
> then the passenger presses the alarm signal button.

to ***describe*** the behaviour of a passenger on the underground. But the use of such an implication says nothing about how the passenger actually ***generates*** that behaviour. As far as the behavourist is concerned, pressing the alarm signal button whenever there is an emergency might be only an instinctive reaction, of whose purpose the passenger is entirely unaware.

Behaviourism is indirectly supported by Darwinism, which holds that organisms evolve by adapting to their environment, rather than by a goal-oriented process of self-improvement.

Behaviourism also shares with condition-action rules a focus on modelling behaviour as reactions to changes in the environment. However, whereas behaviourism restricts its attention to ***descriptions*** of behaviour, condition-action rules are used in production systems to ***generate*** behaviour.

Production Systems

Few psychologists subscribe today even to moderate versions of behaviourism. Most adhere instead to the cognitive science view that intelligent agents engage in some form of thinking that can usefully be understood as the application of computational procedures to mental representations of the world.

Paul Thagard states in his book, *Mind: Introduction to Cognitive Science,* that, among the various models of thinking investigated in cognitive science, production systems have "the most psychological applications" (page 51). Steven Pinker in *How the Mind Works* also uses production systems as his main example of a computational model of the mind (page 69).

> A ***production system*** is a collection of condition-action rules incorporated in the thinking component of an agent's observation-thought-decision-action cycle.
>
> ***Condition-action rules*** (also called ***production rules***) are similar to the behaviourist's descriptions of behaviour. However, because they are used by an agent ***internally*** to ***generate*** its behaviour, their conclusions are often expressed in the ***imperative***, rather than in the declarative mood:
>
> If **conditions** then **do actions**.

Production systems were invented in the 1930's by the logician, Emil Post, but were proposed as a computational model of human intelligence by the Artificial Intelligence researcher, Alan Newell.

The Production System Cycle

Production systems embed condition-action rules in an observation-thought-decision-action agent cycle:

> **To cycle**,
> **observe** the world,
> **think**,
> **decide** what actions to perform,
> **act**,
> **cycle** again.

Thinking is a form of ***forward reasoning***, initiated by an observation matching one of the conditions of a condition-action rule. In such a case, the observation is said to **trigger** the condition-action rule. As in logic, the remaining conditions of the rule are verified and the conclusion is derived.

In logic, the conclusion is an inescapable consequence of the conditions. However, in production systems, it is only a ***recommendation*** to perform the actions that are the conclusion. If only one rule is triggered by the observations, then the recommendation is, in effect, an unequivocal ***command***. However, if more than one is triggered, then the agent needs to choose between the different recommendations, to decide which actions to perform. This ***decision*** is called ***conflict resolution***, because the different recommendations may conflict with one another.

For example:

> If someone attacks me, then attack them back.
> If someone attacks me, then get help.
> If someone attacks me, then try to escape.

Deciding what to do, when there is a conflict between different recommendations, can be harder than generating the recommendations in the first place. We will come back to this decision problem later.

Production Systems without any representation of the world

In the simplest case, an agent's mental state might consist entirely of production rules alone, without any mental representation of the world. In such a case, the conditions of a rule are verified simply by matching them against the agent's current observations. In this case, it can be said (and has been said) that the world serves as its own representation: If you want to find out about the world, don't think about it, just look and see!

Observing the current state of the world is a lot easier than trying to predict it from past observations and from assumptions about the persistence of past states of affairs. And it is a lot more reliable, because persistence assumptions can easily go wrong, especially when there are other agents around, changing the world to suit themselves. It's too early to consider this issue further in this chapter, but it is an issue we will return to later when we look more closely at what's involved in reasoning about the persistence of states over time.

What it's like to be a louse

To see what a production system without any representation of the world might be like, imagine that you are a wood louse and that your entire life's behaviour can be summed up in the following three rules:

> If it's clear ahead, then move forward.
> If there's an obstacle ahead, then turn right.
> If I am tired, then stop.

Because you are such a low form of life, you can sense only the fragment of the world that is directly in front of you. You can also sense when you are tired. Thus, your body is a part of the world, external to your mind. Like other external objects, your body generates observations, such as being tired or being hungry, which have to be attended to by your mind.

It doesn't matter where the rules come from, whether they evolved through natural selection, or whether they were presented to you by some Grand Designer. The important thing is, now that you have them, they govern and regulate your life.

Suppose, for the purpose of illustration, that you experience the following stream of observations:

> Clear ahead.
> Clear ahead.
> Obstacle ahead.
> Clear ahead and tired.

Matching the observations, in sequence, against the conditions of your rules results in the following interleaved sequence of observations and actions:

> Observe: Clear ahead.
> Do: Move forward.
> Observe: Clear ahead.
> Do: Move forward.
> Observe: Obstacle ahead.
> Do: Turn right.
> Observe: Clear ahead and tired.

At this point, your current observations trigger two different rules, and their corresponding actions conflict. You can't move forward and stop at the same time. Some method of conflict resolution is needed, to decide what to do.

Many different conflict resolution strategies are possible. But, in this as in many other cases, the conflict can be resolved simply[22] by assign different priorities to the different rules, and selecting the action generated by the rule with the highest priority. It is obvious that the third rule should have higher priority than the second. So the appropriate action is:

> Do: Stop.

Once a louse has learned its rules, its internal state is fixed. Observations come and go and the louse performs the associated actions without needing to record or remember them. The price for this simplicity is that a louse lives only in the here and now and has no idea of the great wide world around it. But, for a louse, this is probably a small price to pay for being able to enjoy the simple life.

Production Systems with Memory

Although the simple life has its attractions, most people prefer more excitement. For this, you need at least a production system with an internal memory. The memory can be used to store a historical record of current and past observations.

[22] An even simpler approach is to avoid conflict resolution altogether, by changing the rules, adding an extra condition "and you are not tired" to the first and second rules. A more complicated approach is to use Decision Theory, to compare the different options and to select the option that has the highest expected benefit. But, no matter how it is done in this case, the result is likely to be the same – better to rest when you are tired than to forge ahead no matter what.

Typically, an individual observation has the form of an **atomic sentence**[23], so called because it contains no proper subpart that is also a sentence. Thus, the logical form of an observation contains none of the logical connectives, "and", "or", "if" and "not", which turn simpler sentences into more complex ones. An atomic sentence is also called an **atom**.

In a production system with memory, a rule is triggered by a current observation that matches one of the conditions of the rule. Any remaining conditions are then verified by testing them against records of current, past or future observations, and the actions of the rule are derived as candidates for execution.

What it's like to be a Mars Explorer

To imagine what a production system with memory might be like, suppose that your life as a louse is terminated and you are reincarnated as a robot sent on a mission to look for life on Mars.

Fortunately, your former life as a louse gives you a good idea how to get started. Moreover, because you are a robot, you never get tired and never have to rest. However, there are two new problems you have to deal with: How do you recognise life when you see it, and how do you avoid going around in circles.

For the first problem, your designers have equipped you with a life recognition module, which allows you to recognise signs of life, and with a transmitter to inform mission control of any discoveries. For the second problem, you need a memory to recognise when you have been to a place before, so that you can avoid going to the same place again.

A production system with memory, which is a refinement of the production system of a louse, might look something like this:

> If the place ahead is clear
> and I haven't gone to the place before,
> then go to the place.
>
> If the place ahead is clear
> and I have gone to the place before,
> then turn right.
>
> If there's an obstacle ahead
> and it doesn't show signs of life,
> then turn right.
>
> If there's an obstacle ahead
> and it shows signs of life,
> then report it to mission control
> and turn right.

To recognise whether you have been to a place before, you need to make a map of the terrain. You can do this, for example, by dividing the terrain into little squares and naming each square by a co-ordinate, (E, N), where E is the distance of the centre of the square East of the origin, N is its distance North of the origin, and the origin (0, 0) is the square where you start.

---

[23] This assumes that an agent's experience of the world can be expressed in linguistic terms. This is certainly not true of ordinary, natural language, but might, by some stretch of the imagination, apply to the "Language of Thought". More about this later.

For this to work, each square should be the same size as the step you take when you move one step forward. Assuming that you know the co-ordinates of your current location, you can then use simple arithmetic to compute the co-ordinates of the square ahead of you and the square to the right of you, and therefore the co-ordinates of your next location.

Every time you go to a square, you record your observation of the square together with its co-ordinates. Then, to find out whether you have gone to a place before, you just consult your memory of past observations.

Suppose for example, that you are at the origin, pointed in an Easterly direction. Suppose also that the following atomic sentences describe part of the world around you:

Life at (2, 1)
Clear at (1, 0)
Clear at (2, 0)
Obstacle at (3, 0)
Obstacle at (2, -1)
Obstacle at (2, 1).

Although there is life in your vicinity, you can't see it yet. So, when you start, the only thing you know about the world is that it is clear at (1, 0).

Assume also that, although it is your mission to look for life, you are the only thing that moves. So this description of the world applies to all states of the world you will encounter (assuming that, when you occupy a place, it is still considered clear).

With these assumptions, you have no choice. Your behaviour is completely predetermined:

Observe: Clear at (1, 0)
Do: Go to (1, 0)
Record: Gone to (1, 0)

Observe: Clear at (2, 0)
Do: Go to (2, 0)
Record: Gone to (2, 0)

Observe: Obstacle at (3, 0)
Do: Turn right

Observe: Obstacle at (2, -1)
Do: Turn right

Observe: Clear at (1, 0)
Remember: Gone to (1, 0)
Do: Turn right

Observe: Obstacle at (2, 1) and Life at (2, 1)
Do: Report life at (2, 1) to mission control
Do: Turn right.[24]

Notice that reporting your discovery of life to mission control is just another action, like moving forward or turning right. You have no idea that, for your designers, this is the ultimate goal of your existence.

[24] I leave it to the reader to work out what happens next, and I apologise in advance.

Your designers have endowed you with a production system that achieves the goal of discovering life as an emergent property. Perhaps, for them, this goal is but a sub-goal of some higher-level goal, such as satisfying their scientific curiosity. But for you, none of these goals or sub-goals is apparent.

The use of production systems to simulate goal-reduction

Production systems have been used, not only to construct computational models of intelligent agents, but also to build computer applications, most often in the form of expert systems. Many of these applications use condition-action rules to simulate goal-reduction explicitly, instead of relying on emergent properties to achieve higher-level goals implicitly.

For example, the fox's reduction of the goal of having cheese to the sub-goals of being near the cheese and picking it up can be simulated by the condition-action rule[25]:

> If I want to have an object
> then add to my beliefs that I want to be near the object
> and pick up the object.

Here a goal ***Goal*** is represented in the system's memory as a pseudo-belief of the form:

> I want ***Goal.***

The reduction of ***Goal*** to ***Sub-goals*** is simulated by a condition-action rule with a condition ***Goal*** and actions that are either genuine actions performed externally or pseudo-actions of the form:

> add to my beliefs that I want **Sub-goal**

performed internally on the system's memory.

The main problem with the simulation approach is that it looses the connection between goal-reduction and the belief that justifies it, in this case with the belief:

> An animal has an object
> if the animal is near the object
> and the animal picks up the object.

As we have already seen, the connection is that goal-reduction is the same as reasoning backwards with the belief, which is the main idea of logic programming.

Thagard (page 45) gives a similar example of a condition-action rule, but uses it to illustrate his claim that "unlike logic, rule-based systems can also easily represent strategic information about what to do":

> If you want to go home and you have the bus fare,
> then you can catch a bus.

Forward reasoning with the rule reduces the goal (going home) to a sub-goal (catching a bus), and simulates backward reasoning with the belief[26]:

---

[25] The rule can be paraphrased more naturally, although somewhat less precisely, in ordinary English: If I want to have an object, then I should be near the object and pick it up.

You go home if you have the bus fare and you catch a bus.

Thus Thagard's argument against logic can be viewed instead as an argument for logic programming, because it can "easily represent strategic information about what to do".

Indeed, it seems that Thagard's other arguments for production rules can also be understood as arguments for logic instead. This is because he confuses production rules:

If conditions then do actions.

with logical implications:

If conditions then conclusions.

An unfortunate confusion

This confusion is perhaps most apparent when Thagard writes (page 47) that "rules can be used to reason either *forward* or *backward*." But this ability to be used forward or backward is not a true property of production rules, but rather a characteristic of logical implications.

Because conditions in production rules come first and actions happen later, true production rules can be used only in the forward direction, when the conditions hold to derive candidate actions. But because conclusions in logic are always expressed in the declarative mood, logical implications can be used to reason either forward or backward.

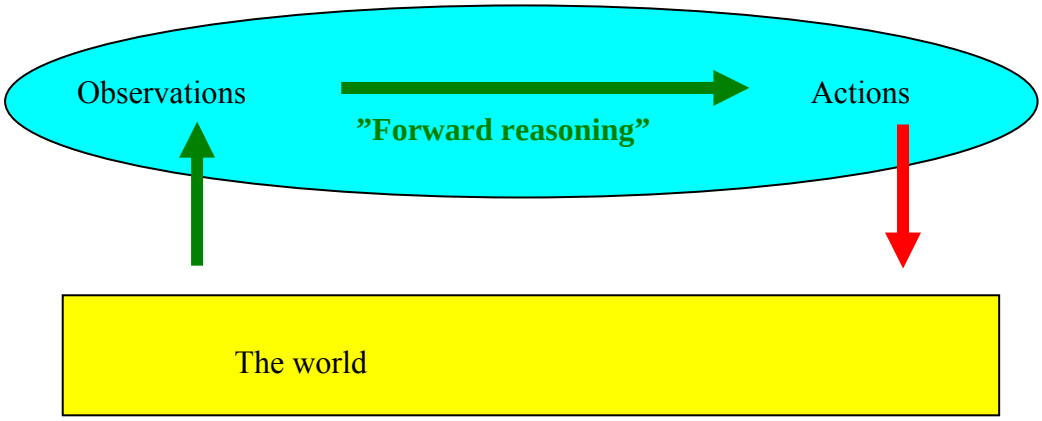
Thus Thagard mistakenly attributes to production rules a property that they don't have, but that logical implications do, and then he uses this mistaken attribution to argue that "rules" are better than logic.

To be fair to Thagard, he is only reporting a generally held confusion. In this case, the rule that he uses as an example simulates **goal-reduction**, which is a special case of **backwards reasoning with a belief** expressed in logical form. However, in the next chapter, we will see that true **production rules** are a special case of **forward reasoning with goals** expressed in logical form. We will also see how goal-reduction and production rules can be combined in a more general framework, which uses logic for both beliefs and goals.

[26] In this form it is perhaps more obvious that the procedure will work only if the bus you catch is going past your home.

Summary

The use of production systems to generate the behaviour of an intelligent agent, as seen in this chapter, can be pictured like this:



Chapter 4 A Logical Framework for Combining Goal-reduction, Condition-action Rules and More

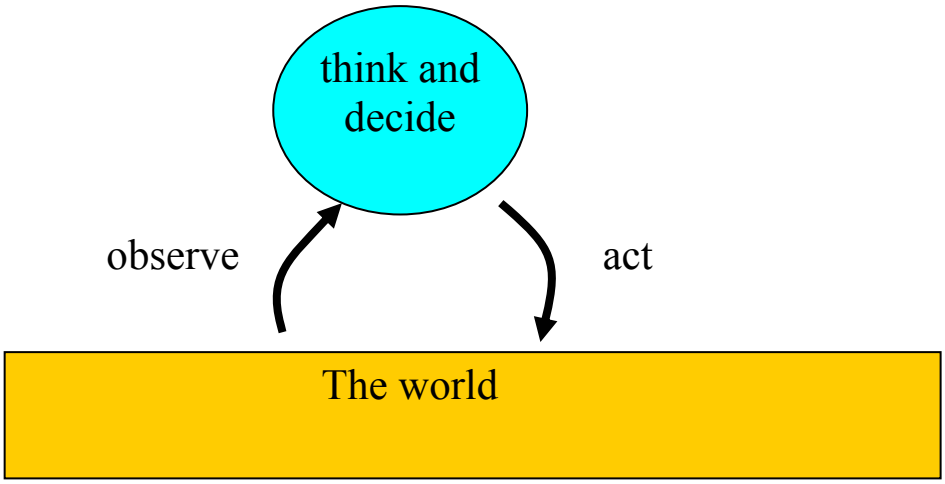
What do the passenger on the London underground, the fox, the wood louse, the Mars explorer and even the heating thermostat have in common? It certainly isn't the way they dress, the company they keep, or their table manners. It's that they all are engaged in a constant struggle with the changing world - a struggle that sometimes threatens their existence, and at other times provides them with opportunities to thrive and prosper.

For logic to be relevant in such a world, it needs first to overcome one main problem – that traditional logic, as normally understood, is closed to changes in the world. To be relevant, logic needs to be put in its proper place, in the thinking component of the agent cycle:

To cycle,
observe the world,
think,
decide what actions to perform,
act,
cycle again.

At this top-most level, logic shares the same agent cycle as production systems. The cycle is general enough to accommodate different kinds of thinking, including both condition-action rules and logical reasoning. Embedded in the agent cycle, logic is open both to inputs that the agent receives from the world and to outputs that the agent can use to change the world in return.

We can picture this relationship between the mind of an agent and the world like this:



In this picture, thinking is a form of symbol processing in which symbols in the mind represent objects and relationships among objects in the world. The world, on the other hand, is a semantic structure, which gives meaning to the agent's thoughts.

The logic of goals and beliefs

Another shortcoming of traditional logic is that it does not pay adequate attention to the distinction between an agent's goals and its beliefs. But for an intelligent agent, the distinction is fundamental.

An agent's goals represent its view of the world as it would like it to be. They include actions that the agent can perform immediately. They also include achievement goals, to attain some desired future state, maintenance goals, to maintain the agent in some desired relationship with the changing state of the world, and constraints, to prevent undesirable states. I will argue that condition-action rules can be understood as maintenance goals, and that goals in general can be expressed naturally in logical form.

An agent's beliefs, on the other hand, represent its view of the world as it really is, whether the agent likes it or not. They include atomic sentences that record the agent's observations. They also include its beliefs about the laws of nature, as well as definitions and taxonomies that the agent uses to classify and organise its experience. I will argue that many, perhaps all beliefs can be expressed naturally in logic programming form.

The story of the fox and crow revisited

The story in Chapter 2 begins when the fox has the goal of having the crow's cheese. But where did the goal come from?

Perhaps, like a spoiled child, whenever the fox sees that someone else has something, then she wants to have it as well. Or perhaps she's simply looking for her next meal. In either case, the fox's goal of having the cheese comes from a higher-level goal –

either the goal of possessing anything she sees anyone else have, or the goal of having food to eat whenever she becomes hungry.

Thus, the fox's goal of having the cheese can be viewed as a goal of ***achieving*** some future state of the world, in response to ***observing*** a change in the world that triggers a higher-level goal of ***maintaining*** some relationship with the world around her.

Let's give the fox the benefit of the doubt and assume that she wants to have the cheese simply because she is hungry, and not because she has a personality defect. This can be represented, by the higher-level goal:

If I become hungry, then I need to have food and eat the food.

This goal can be paraphrased in many different ways. For example, in the imperative:

If I become hungry, then have food and eat the food.

This imperative formulation resembles a condition-action rule, except the conclusion contains a goal "have food" that is not a simple action.

Alternatively, the goal can be stated in the declarative mood, but with an indication that it is a goal, rather than a belief:

Goal: If I become hungry, then I have food and I eat the food.

The advantage of expressing the goal as a declarative statement is that it has a logical form, which can take part in logical reasoning. The fact that it is a goal rather than a belief means that the agent needs to make it become true.

To ***show*** that an implication:

If conditions then conclusion.

is true, it is ***necessary*** to show that whenever the conditions are true, then the conclusion is true. Whether it is a goal or a belief is irrelevant.

However, to ***make*** an implication become true, because it is a goal, not only is it ***necessary*** to make the conclusion come true when the conditions become true, but it is ***sufficient***. In theory, it would be possible to make the implication true by making both the conditions and the conclusion true. However, making the conditions true creates unnecessary work. In practice, therefore, to minimise its work, an agent does not attempt to make conditions of implicational goals become true unnecessarily. Either the world makes them true, whether it likes it or not, or the agent makes them true for some other purpose[27].

In an open world, which is always changing, it may not be possible to make an implication true once and for all. It may be necessary, instead, to ***maintain*** its truth

[27] It is also possible to make an implication become true by making its conditions become false. In the case of the fox, she could also satisfy her goal by trying to prevent herself from becoming hungry.

indefinitely, dealing with individual instances of the conditions as they arise, on separate occasions. On each such occasion, it is necessary to make that instance of the implication true. The entire process consists of the following three steps:

- forward reasoning, matching a new observation with some condition of the implication, generating an instance of the implication,
- forward and/or backward reasoning, verifying any remaining conditions of this instance of the implication,
- deriving the new goal of making the conclusion true.

The new goal is a sub-goal of the higher-level ***maintenance goal***. Typically it is an ***achievement goal***, to attain some future state of affairs

Let's see how this derivation of an achievement goal from a maintenance goal works in this version of the story of the fox and the crow:

Goal: If I become hungry, then I have food and I eat it.

Beliefs: The crow has the cheese.

An animal has an object
if the animal is near the object
and the animal picks up the object.

I am near the cheese
if the crow has the cheese
and the crow sings.

The crow sings if I praise the crow.

Cheese is a kind of food,
Food is a kind of object.

To make the story work, the fox needs to have the taxonomic knowledge that cheese is a kind of food and that food is a kind of object. This knowledge can be represented in a number of different ways, and there are even specialised logics for this purpose. However, for the simple purpose of retelling the story, these details are unimportant. It suffices to recognise that this knowledge is needed simply to assist with the reasoning, as we will see below.

To see how the derivation of an achievement goal from a maintenance goal works in this example, suppose the fox has the goal and beliefs presented above and suddenly her body tells her that she has just become hungry:

Observe: I become hungry.

Since her body is a part of the world, she becomes aware of her hunger by means of an observation. The observation triggers the top-level goal, and forward reasoning derives the sub-goals:

Sub-goals: I have food and I eat the food.

Thus, the real achievement goal is not specifically to have the crow's cheese, but more generally to have any kind of food. And this achievement goal is only half the story. The other half of the story is that the fox also needs to eat the food. Having the food without eating it is useless.

The derivation of an achievement sub-goal from a higher-level maintenance goal generalises the derivation of actions from a condition-action rule, because the achievement sub-goal can include atomic actions. However, in the general case, the achievement sub-goal might contain sub-goals that need to be reduced to still lower-level sub-goals. Maintenance goals generalise condition-action rules also because the derived achievement sub-goals might need to be achieved at some future time and not simply at the next time moment.

The time factor

Our new version of the story of the fox and crow is still an oversimplification, because it does not deal adequately with the issue of time. It does not distinguish between different occurrences of becoming hungry at different times. Nor does it indicate how much time can elapse between becoming hungry and eating. For that matter, it does not even indicate that the fox needs to have the food before she can eat it.

This oversight has been deliberate, because common-sense reasoning about time is much more complicated than it seems. We will come back to this problem later. In the meanwhile we will continue to use our simplified representation of the story without time, both to simplify the example and to focus on how the fox interleaves its reasoning with its observations and actions in the context of the agent cycle.

Embedding the story in the agent cycle

Let's retell the story, starting this time at the point where the fox becomes hungry. To make the story more interesting, we assume that the fox has only enough time to perform one step of reasoning in a single cycle. We also assume that the fox observes whether her actions succeed or fail.

The first cycle.
Observation: I become hungry.
Forward reasoning, sub-goals: I have food and I eat the food.
No candidate action.
This is the classic case of an observation triggering a maintenance goal and deriving an achievement goal.

The second cycle.
No observation.
Backward reasoning, new sub-goals: I am near food and I pick up the food and I eat the food.
No candidate action.

The only thinking that the fox can do in this cycle is to reason backwards, to reduce the sub-goal of having food to the sub-goal of being near the food and picking it up. This reasoning involves the taxonomic reasoning of matching "food" with "object".

The third cycle.
Observation: The crow has cheese.
Forward reasoning, new belief: I am near the cheese if the crow sings.
No candidate action.
In this cycle, the fox has the choice of continuing to reason backwards from its current sub-goals or of reasoning forwards from its new observation. Normally, it is a good idea to give priority to reasoning with new observations, just in case there is an emergency that needs to be dealt with immediately or an opportunity that shouldn't be missed.

The observation matches one of the conditions of her belief:

I am near the cheese if the crow has the cheese and the crow sings.

Because the belief is expressed in logical form, it can be used to reason both forward and backward. Using it to reason forward, as in this case, it gives rise to a new belief.

The fourth cycle.
No observation.
Backward reasoning, new sub-goals: the crow sings and I pick up the cheese and I eat the cheese.
No candidate action.
The fox uses the new belief backwards, to reduce the goal of being near food to the sub-goal of making the crow sing. To do so, she matches "food" with "the cheese". This has a side effect of finding out what the food is going to be if the new sub-goal succeeds.

The fifth cycle.
No observation.
Backward reasoning, new sub-goals: I praise the crow and I pick up the cheese and I eat the cheese.
Action: I praise the crow.
The fox reduces the goal of making the crow sing to the sub-goal of praising the crow. She attempts to solve this new sub-goal by successfully performing it as an action. There are no other actions she can perform at this time, so there is no conflict resolution to perform.

The sixth cycle.
Observe: I praise the crow.
Forward reasoning, new sub-goals: I pick up the cheese and I eat the cheese.
Action: I pick up the cheese.
The fox observes the result of the action she performed in the previous cycle. The observation, which confirms the success of her action, solves the first of the three action sub-goals, leaving the remaining two sub-goals. The next of these two sub-goals is also an action, and there are no other candidate actions that she can perform at this time.

The seventh cycle.

Observe: I do not pick up the cheese.

No thinking.

Action: I pick up the cheese.

The fox observes the result of her action. However, this time, we assume that it fails, either because the crow has not yet started singing, because the cheese has not yet reached the ground, or because she is physically inept[28]. We assume that the fox can try the same action again[29].

The eighth cycle.

Observe: I pick up the cheese.

Forward reasoning, new sub-goals: I eat the cheese.

Action: I eat the cheese.

The fox observes that this time the action was successful. The observation solves the associated action sub-goal, leaving only the last action sub-goal, which the fox decides to perform in this cycle:

The ninth cycle.

Observe: I eat the cheese.

The observation solves the last of the action sub-goals. However, the maintenance goal still remains, to be triggered on other, future occasions.

The general pattern of reasoning in this example, spread out over several cycles and possibly interleaved with other observations and actions, is this:

Observation:	An event happens.
Forward reasoning:	The event matches the condition of a maintenance goal or of a belief.
Achievement goal:	Eventually the conclusion of some maintenance goal is derived as an achievement goal.
Backward reasoning:	Beliefs are used to reduce the achievement goal to actions.
Actions:	Action sub-goals are selected for execution.
Observation:	The agent observes whether the actions succeed or fail.

This pattern is not exceptional. A similar pattern arises in the London underground example.

---

[28] This complication draws attention to a shortcoming of our formulation of the agent cycle: There is no explicit check in the cycle to determine whether or not lower-level actions actually achieve their higher-level goals. The cycle checks whether or not atomic actions succeed, but it does not check whether their success leads to the success of the higher-level goals they were meant to achieve. If the beliefs that were used to reduce these goals to sub-goals were guaranteed to be true, then such checks would be unnecessary. Thus, one way to rectify the shortcoming is to add to the cycle a general learning, truth-testing component. We need to come back to this later.

[29] This assumes that there is an associated time by which the action needs to be performed and that that time is not yet used up. We will see how this works in greater detail later, when we look more closely at the representation of time.

The underground revisited

Remember our restatement of the first sentence of the Emergency Notice:

> If there is an emergency then get help.
> You get help if you alert the driver.
> You alert the driver if you press the alarm signal button.

Here the first sentence expresses a maintenance goal. We can rewrite it as a declarative sentence, indicating that it is a goal rather than a belief. To better capture the intended meanings of the second and third sentences, we replace the second person "you" by the third person:

Goal: If there is an emergency then I get help.

Beliefs: A person gets help if the person alerts the driver.
A person alerts the driver if the person presses the alarm signal button.

To recognise when there is an emergency, we need additional beliefs. For example:

Beliefs: There is an emergency if there is a fire.
There is an emergency if one person attacks another.
There is an emergency if someone becomes seriously ill.
There is an emergency if there is an accident.

We could continue this reduction of abstract, higher-level concepts to more concrete, lower-level ones, down to any level of detail. For example, in the case of recognising fire, we might add:

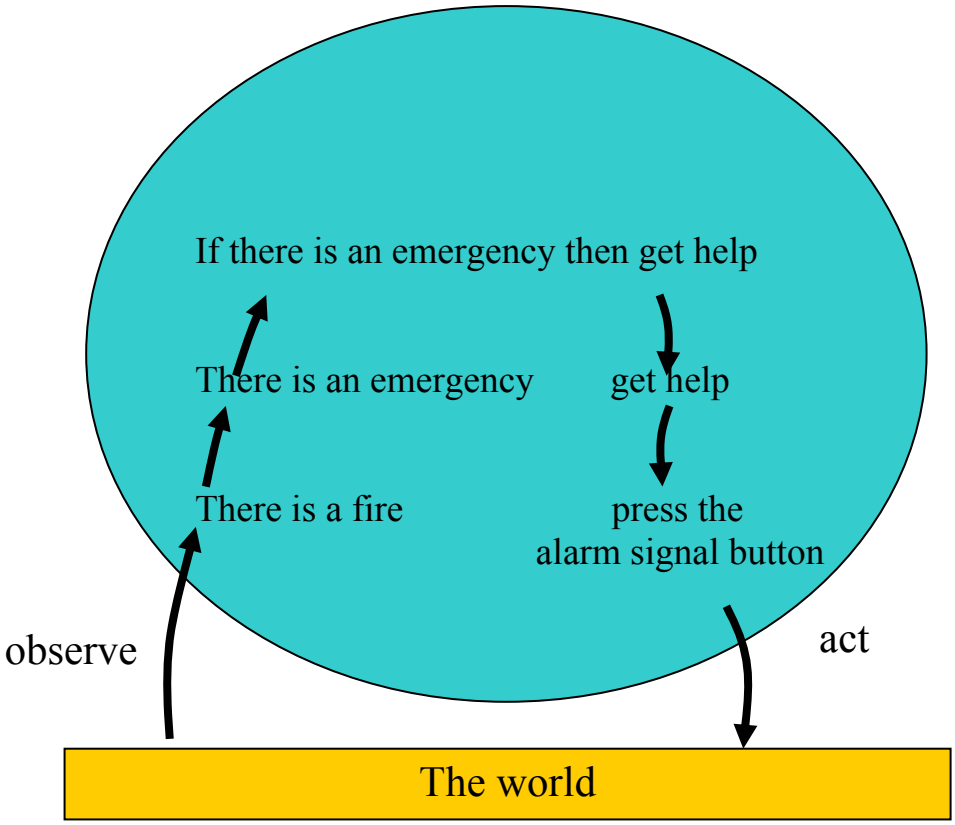
Additional beliefs: There is a fire if there are flames.
There is a fire if there is smoke.

However, we would soon find it increasingly difficult to define the lower-level concepts in linguistic terms. Eventually, there must come a point at which there is a lowest level, which is irreducible and which is the level at which the agent's sensory system transforms the sensations it receives from the environment into observations that can be represented in conceptual terms.

Suppose in this example that the concept of recognising there is smoke is such a lowest level of concept directly observed in the environment. Suppose, moreover, that you are traveling on the underground and you observe smoke:

Observation:	There is smoke.
Forward reasoning, new belief:	There is a fire.
Forward reasoning, new belief:	There is an emergency.
Forward reasoning, achievement goal:	I get help!
Backward reasoning, sub-goal:	I alert the driver!
Backward reasoning, action:	I press the alarm signal button!

We can picture this combination of forward and backward reasoning like this:



In three steps of forward reasoning, you derive the achievement goal of getting help. In the first step you recognise that there is a fire, in the second you recognise that there is an emergency, and in the third you use the maintenance goal to derive the achievement goal of getting help. Then in two steps of backward reasoning, you reduce the goal of getting help to the action sub-goal of pressing the alarm signal button.

The action of pressing the alarm signal button, like the observation of an emergency, can be reduced to lower-level terms - for example, by first moving your finger to the button and then pushing the button with your finger. Moving your finger to the button can also be reduced, in turn, to still lower-level sub-goals, like first moving your arm to the vicinity of the button and then moving your finger to the button. But eventually, there has to be a point where your body takes over from your mind and performs the actions directly on its own.

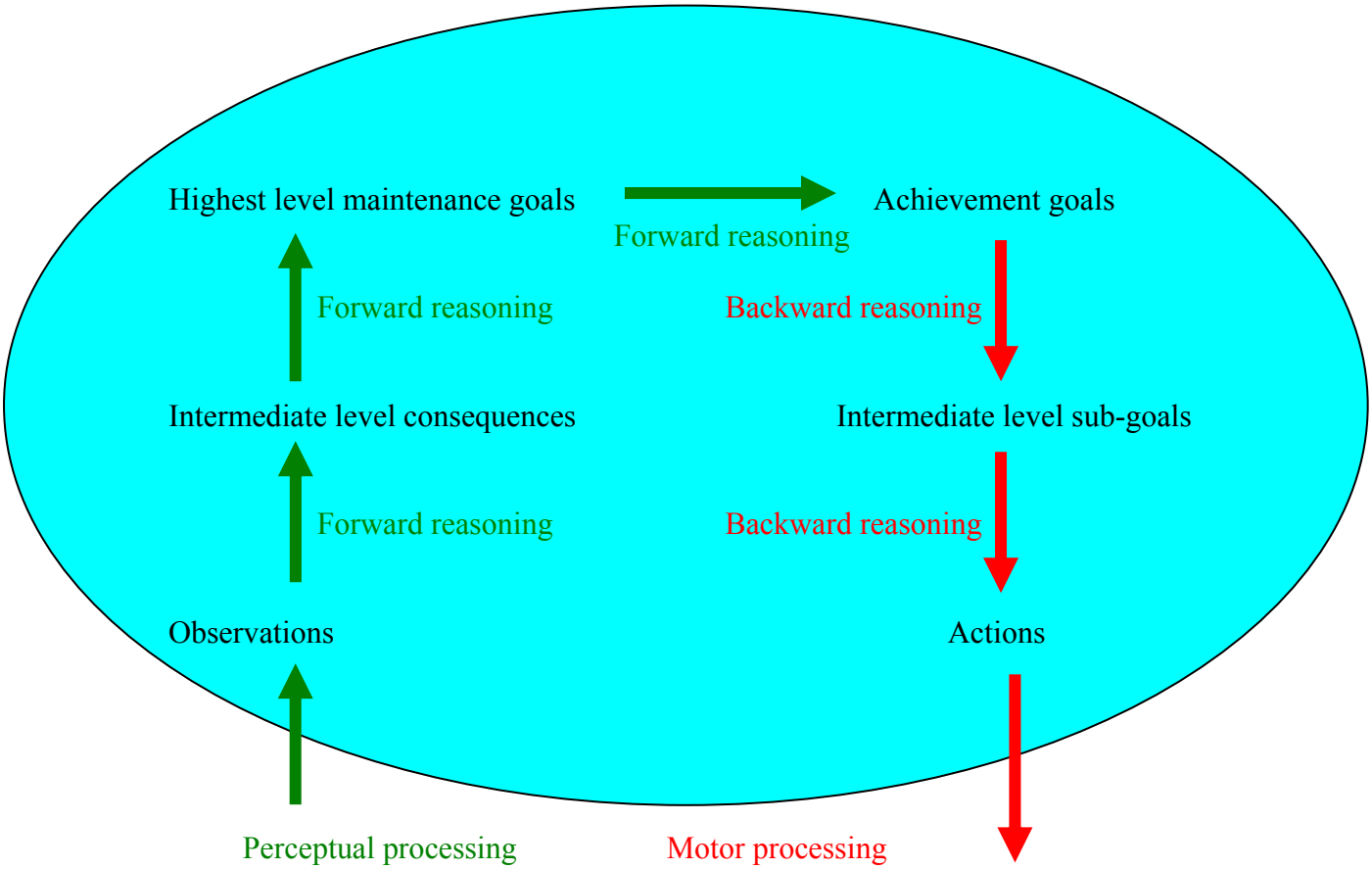
All of this thinking takes time, during which you may have to record other observations and perform other actions. Scheduling these so that everything is dealt with in a timely manner is a task for the decision making component of the agent cycle. We have kept the examples in this chapter deliberately simple, so that no such decisions need to be made. However, we will address some of the issues involved in making decisions in the next chapter.

Summary

The examples in this chapter illustrate how logic can be used in the context of an agent's observation-thought-decision-action cycle. Placed in this context, logic is used for the higher levels of thought - both to reason forwards from observations, triggering maintenance goals and deriving achievement goals, and to reason backwards to reduce achievement goals to actions.

Below the logical level, perceptual processes transform raw sensation into observations and motor processes transform conceptual representations of actions into raw physical activity.

This can be pictured in the following form:



We have seen that forward reasoning with implicational goals generalises condition–action rules, whereas backward reasoning with implicational beliefs generalises goal-reduction procedures. In the next chapter, we will see how backward reasoning can be used to search for solutions and how forward reasoning can be used to infer consequences of solutions. We will also see how this combination of search plus inference helps to inform the next, decision-making stage in the cycle, so that different candidate solutions can be compared, and so that better informed decisions can be made.

Chapter 5 Thinking = Inference + Search + Inference

Jonathan Baron (1994) in his textbook, "Thinking and Deciding" writes on page 4:

"Thinking about actions, beliefs and personal goals can all be described in terms of a common framework, which asserts that thinking consists of *search* and *inference*. We search for certain objects and then make inferences from and about the objects we have found."

Although Baron sees the role of logic as limited to inference, we see logic involved in both search and inference. The objects that we search for are solutions of goals, and we do so by using backward reasoning to reduce goals to sub-goals. The inferences we make derive consequences of candidate solutions by using forward reasoning. Like Baron, we distinguish between ***thinking***, which generates candidate solutions and derives their consequences, and **deciding**, which evaluates solutions and chooses between them.

It seems to be a common view that logic has nothing to do with search. Indeed Paul Thagard in his introduction to cognitive science (page 45) states: "In logic-based systems the fundamental operation of thinking is logical deduction, but from the perspective of rule-based systems the fundamental operation of thinking is search."

To see how logic is concerned with search, consider the problem of

Going from A to B

We are all familiar with searching for objects in physical space, and with searching how to go from one place to another:

To go from A to B,
if A is directly connected to B then
go from A to B directly.

To go from A to B,
if C is between A and B then
go from A to C and
go from C to B.

More generally and expressed as beliefs in logical terms:

An agent goes from A to B,
if A is directly connected to B and
the agent goes from A to B directly.

An agent goes from A to B,
if C is between A and B and
the agent goes from A to C and
the agent goes from C to B.

The procedures and beliefs apply not only to physical places but also to conceptual places, like "rags" and "riches".

The goal-reduction procedures are a special case of the beliefs. They are the special case in which the beliefs are used to reason backward and "the agent" is the agent who uses them to reduce goals to sub-goals. However, unlike the procedures, the beliefs can also be used to reason forward, for example to draw consequences from observations about another agent's behaviour.

There can be many ways of choosing a place C, between A and B. For example, you can go from rags to riches either by getting a paid job or by robbing a bank. Similarly, there can be many ways of going from A to C and of going from C to B. For example, you can get a paid job either by going to work directly after finishing school or by getting higher qualifications first and then going to work after you graduate.

Some of the choices for the intermediate place C might not succeed in solving the other sub-goals of going from A to C or of going from C to B. For example, although you might be able to get a paid job directly after leaving school, you might not then be able to go on from that job to become rich.

In the general case, therefore, to solve the goal of going from A to B, you need to search for a solution. Instead of searching in material space, you can save some of the work by doing some of the searching in your mind.

You can use the beliefs, for example if you're planning your summer holiday, to search for a plan for getting from A to B, long before you actually start your holiday. You can mentally explore alternative plans, and even search for a plan that optimises the outcome, seeking to minimise its costs and maximise its benefits. Moreover, you can interleave your planning with other things, suspending it when you have other more pressing commitments to attend to, and resuming it when you have nothing more important to do.

How to get to the French Riviera

If you still need convincing, consider the goal:

Goal: I go from Wimbledon to the French Riviera.

Suppose that I have the following beliefs:

Beliefs: Nice is between Wimbledon and the French Riviera.
Paris is between Wimbledon and the French Riviera.
Heathrow is between Wimbledon and the French Riviera.
Gatwick is between Wimbledon and Nice.
Clapham Junction is between Wimbledon and Gatwick.

Wimbledon is directly connected to Clapham Junction.
Clapham Junction is directly connected to Gatwick.
Gatwick is directly connected to Nice.
Nice is directly connected to the French Riviera.

etc.

I might have this information already stored in my memory directly as atomic facts, or I might be able to derive it from other sources.

Reasoning backwards, I have two alternative ways of trying to solve my goal. I can generate either the sub-goals:

> Wimbledon is directly connected to the French Riviera and
> I go from Wimbledon to the French Riviera directly.

or the sub-goals:

> C is between Wimbledon to the French Riviera and
> I go from Wimbledon to C and
> I go from C to the French Riviera.

Which of these I generate first, or whether I generate both simultaneously, depends on my search strategy. Suppose I generate the first one first. Then I have to decide which sub-goal to work on first, the sub-goal Wimbledon is directly connected to the French Riviera or the sub-goal I go from Wimbledon to the French Riviera directly.

Suppose I regard the second sub-goal is an atomic action, because I am not interested in going beyond this level of detail at this time. Then I have to work on the first sub-goal Wimbledon is directly connected to the French Riviera. Given only the information I have listed above, this sub-goal has no solution. I must, therefore, abandon or suspend this line of search or else perform an external action to try to find additional information in case there is a connection I don't know about. Suppose I decide to suspend this line of search.

I am left with the other alternative way of trying to solve my top-level goal:

> C is between Wimbledon to the French Riviera and
> I go from Wimbledon to C and
> I go from C to the French Riviera.

Suppose I decide to work on the first of the three sub-goals. (There is no point of working on either of the other two sub-goals before I have picked an intermediate place C.) Given only the limited information I have listed above, there are three ways to solve this sub-goal. I have to decide which way to try first.

And so the search progresses, choosing an alternative, choosing a sub-goal in the alternative, and deciding whether to perform external actions to get more information, until I find one or more solutions. In this case, if I decide not to perform any external, information-gathering actions, the only solution is the plan:

> I go from Wimbledon to Clapham Junction directly.
> I go from Clapham Junction to Gatwick directly.
> I go from Gatwick to Nice directly.
> I go from Nice to the French Riviera directly.

I can derive this plan, searching among alternatives in my mind, either forward from Wimbledon or backward from the French Riviera, depending on which sub-goals I work on first. However, in either case, I search by reasoning backwards from goals to sub-goals.

Of course, if I had additional information, I might be able to find additional solutions for my initial goal. I would then need to choose between the alternatives, to decide which solution to implement. Because the ***purpose of thinking*** in this case is ultimately to help in ***deciding*** what to do, I could use the same criteria that I use to decide between solutions - for example the criterion of most benefit for least cost – as a search strategy, to explore more promising before less promising avenues of thought.

Logical Reasoning = Search + Inference

There is another, more interesting sense in which logic combines search and inference – the sense in which Jonathan Baron characterises thinking in general: "We search for certain objects and then we make inferences from and about the objects we have found."

In our logic-based framework, we search for solutions by reasoning backwards from goals. However, to help in deciding between alternative solutions, we can explore the space of forward inferences, to find any additional, desirable or undesirable consequences of the solutions.

To illustrate this sense in which thinking combines search and inference, Baron gives the example of a student trying to decide what course to take as an elective. First she considers a course on modern history, which sounds interesting, but involves too much work. Then she thinks of another modern history course, which is also interesting, but which might not be as much work. So she tries to find someone who has taken the course before to find out how much work is actually involved.

Baron's example is similar to our example of planning to take a holiday on the French Riviera, but it illustrates the importance of information-gathering actions. It shows that you can't expect to have all the information you need to solve a problem already in your internal memory. You might need to consult external sources as well.

However, the main purpose of the example is to illustrate the use of inference to derive consequences of candidate solutions, to help in deciding what to do. In Baron's example, the inference is simple: If the student takes the first course, then it will involve a lot of work.

Deciding what to do, based on these inferences, is more complex. It involves comparing different candidate courses for their advantages and disadvantages. Since no single course is likely to outrank all other courses on all the relevant criteria, hard choices will probably need to be made, perhaps sacrificing some advantages of a course in order to avoid some of its disadvantages. To make matters even more complicated, the student will need to base her estimates of the costs and benefits of the different alternatives on uncertain, perhaps probabilistic information.

Uncertainty

Uncertainty about future circumstances beyond our control is a feature of most real-life problem-solving situations. Consider, once more, the problem of going from rags to riches, and suppose that you are thinking about robbing a bank in order to get rich. Robbing a bank isn't an easy option. You would need to think hard, to construct a plan that would be likely to succeed. You would need to pick a bank, consider whether to go it alone or to organise a gang, decide whether to do it in broad daylight or after dark, and plan your get-away.

But before constructing a plan in all its detail, you could mentally explore the likely consequences of robbing a bank, to see if there are any other desirable or undesirable possible outcomes. Apart from any moral considerations, if you rob a bank, get caught, and are convicted, then you will end up in jail; and presumably you don't want to go to jail.

You can control whether or not you try to rob a bank. But you can't control whether you will be caught or be convicted. Not only are these possibilities beyond your control, but you can not even predict their occurrence with any certainty. At best, you can only try to estimate their probability.

If you judge that the chances of getting caught and being convicted are high, then you should decide not to rob a bank, because you don't want to go to jail. There is no need for you to consider the alternative ways of robbing a bank, because all of the alternatives lead to the same unwanted conclusion.

The problem of thinking about robbing a bank not only shows the value of inferring consequences of alternative solutions, but it also shows the need to judge the probability of circumstances outside our control.

Combining judgements of probability with assessments of the utility of different outcomes belongs to the domain of ***Decision Theory***. We will come back to these Decision Theoretic matters later. In the meanwhile, it suffices to note that in many cases, ***thinking***, which combines searching for options with inferring their consequences, can often be a lot easier than ***deciding*** what to option to choose.

Thinking without Search

The characterisation of thinking as search plus inference is a big advance over some other theories, in which thinking is viewed as little more than just search. However, it fails to account for the kind of thinking that is needed to deal with changes in the environment – especially when those changes necessitate a rapid response, and there isn't enough time to search for a solution.

Thinking by searching and inferring consequences is appropriate in many situations, like when you are planning your summer holidays, choosing an elective course or planning to rob a bank, when you have plenty of time to search. However, there are other situations, like when you are in an emergency, when you don't have time to consider all the alternatives and when you don't even have time to finish thinking before you need to act.

Suppose, for example, that you are a Mars Explorer Mark II, equipped with the full capabilities of logical reasoning. You are physically searching for life on Mars, when a Mars Annihilator leaps into your path from over the horizon. Fortunately, you have been forewarned about such emergencies and are equipped with the appropriate maintenance goal:

Goal: If Mars Annihilator in sight,
then go from where I am back to the space ship.

Observation: Mars Annihilator in sight.
Forward reasoning, achievement goal:
go from where I am back to the space ship.

In theory, you could sit down and mentally explore the different ways of getting back to the safety of the space ship, in the same way you would if you were planning your holidays on the French Riviera. But then in practice, you would probably be finished before you got started.

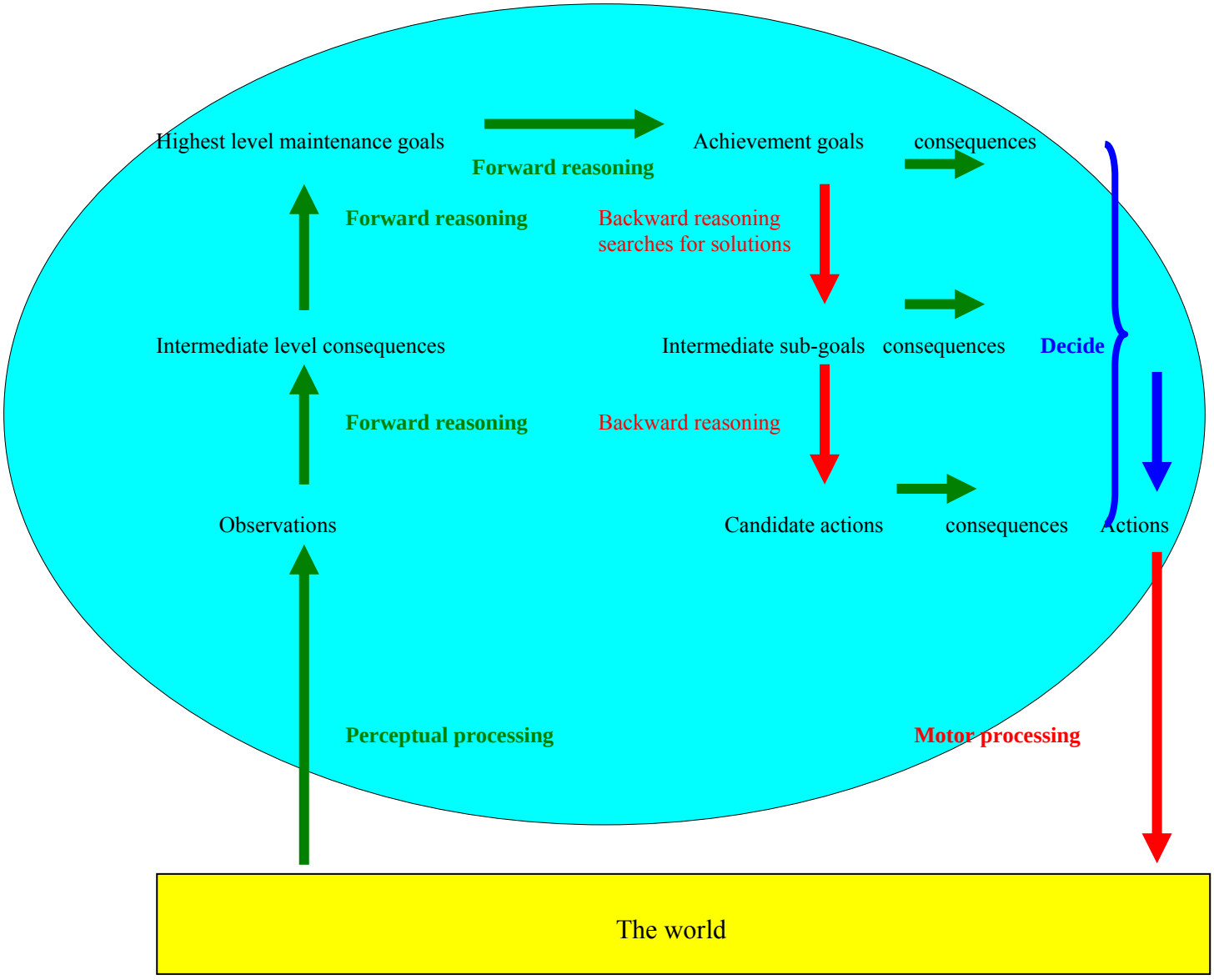
What you need to do instead is to think on your feet, using the same knowledge that you use when planning your summer holidays, but without searching the complete mental space of alternatives. You have to choose a place C directly connected to your current location and in the direction of your space ship and go to C directly, before you start thinking about what you are going to do after that. When you get to C, you need to choose another place C' directly connected to your new location and also in the direction of the space ship and go there directly. you continue in this way, thinking about where to go next and going there, until you reach the space ship if you are lucky or are caught by the Mars Annihilator if you are not.

Thinking about time

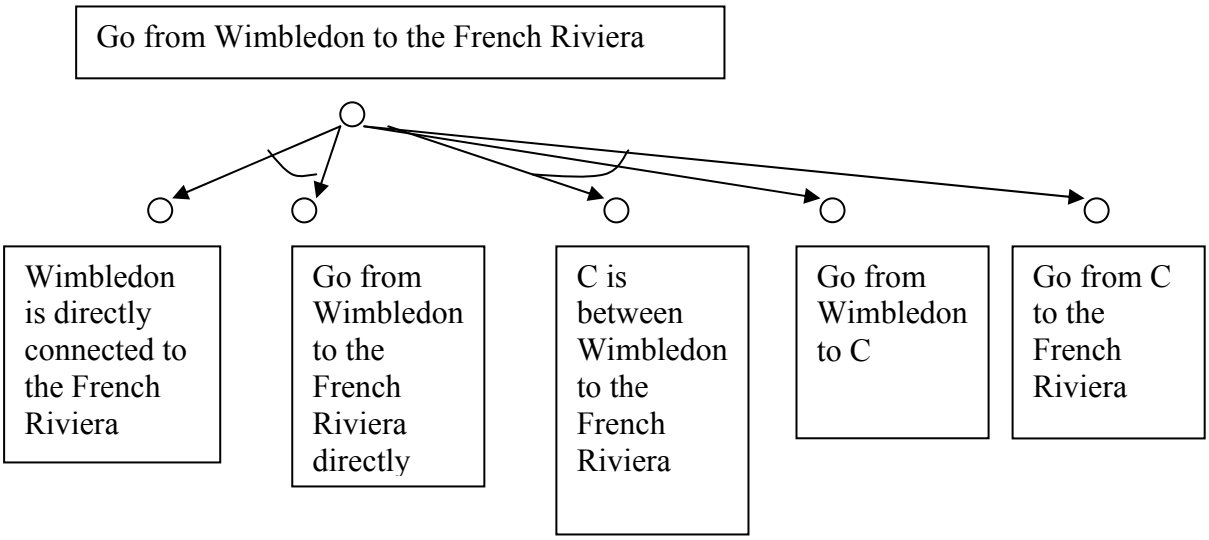
In the general case, to get the right balance between thinking and acting, an agent needs to think about time – both to think about the time when actions need to be taken and to think about how much time is available for thinking before needing to act. The topic of thinking about time is coming up soon.

Thinking = Inference + Search + Inference

We now have a more complete view of the role of logic in the observation-thought-decision-action cycle:



We can view the top level of the search space as a goal-reduction tree:



Chapter 6 The Meaning of Life

It's bad enough to be a Mars explorer and not to know that your purpose in life is to find life on Mars. But it's a lot worse to be a wood louse and have nothing more important to do with your life than to just follow the meaningless rules:

Goals: If it's clear ahead, then I move forward.
If there's an obstacle ahead, then I turn right.
If I am tired, then I stop.

In fact, it's even worse than meaningless.

Without food the louse will die, and without children the louse's genes will disappear. What is the point of just wandering around if the louse doesn't bother to eat and make babies?

Part of the problem is that the louse's body isn't giving it the right signals - not making it hungry when it is running out of energy, and not making it desire a mate when it should be having children. Its body also needs to be able to recognise food and eat, and to recognise potential mates and propagate.

So where does the louse go from here? If it got here by natural evolution, then it has nowhere to go and is on the road to extinction.

But if it owes its life to some Grand Designer, then it can plead with her to start all over again, this time working from the top-down. The Grand Designer would need to rethink the louse's top-level goals, decide how to reduce them to sub-goals, and derive a new, more effective specification of its input-output behaviour.

Suppose the Grand Designer identifies these as the louse's top-level goals:

Top-level goals: The louse stays alive for as long as possible and
the louse has as many children as possible.

Of course, a critic might well ask: What purpose do these goals serve, and why these goals and not others? Perhaps staying alive is just a sub-goal of having children. And perhaps having children is just one way of promoting the survival of one's genes. But eventually the critic would have to stop. Otherwise he could continue asking such questions forever.

To reduce the louse's top-level goals to sub-goals, the designer needs to use her beliefs about the world, including her beliefs about the louse's bodily capabilities. Moreover, she can build upon her earlier design, in which the louse moved around aimlessly, and give its movements a purpose. She could use such beliefs as:

Beliefs:

The louse stays alive for as long as possible,
if whenever it is hungry then it looks for food and when there is food ahead it eats it, and
whenever it is tired then it rests, and
whenever it is threatened with attack then it defends itself.

The louse has as many children as possible,
if whenever it desires a mate then it looks for a mate and

when there is a mate ahead it tries to make babies.

The louse looks for an object,
if whenever it is clear ahead then it moves forward,
and whenever there is an obstacle ahead and it isn't the object then it turns right
and when the object is ahead then it stops.

The louse defends itself
if it runs away.

Food is an object.
A mate is an object.

If the louse were as intelligent as the designer, then the designer could just hand these beliefs and the top-level goal directly over to the louse itself. The louse could then reason forwards and backwards, as the need arises, and would be confident of achieving its goals, provided the designer's beliefs are actually true.

But the louse possesses neither the designer's intellect, nor her gorgeous body and higher education. The designer, therefore, not only has to identify the louse's requirements, but she has to derive an input-output specification, which can be implemented in the louse, using its limited physical and mental capabilities.

One way for the designer to do her job is to do the necessary reasoning for the louse in advance. She can begin by reasoning backwards from the louse's top-level goals, to generate the next, lower level of sub-goals:

Sub-goals:

whenever the louse is hungry then it looks for food and when there is food ahead it eats it, and
whenever the louse is tired then it rests, and
whenever the louse is threatened with attack then it defends itself and
whenever the louse desires a mate then it looks for a mate and when there is a mate ahead
it tries to make babies.

The English words "whenever" and "when" are different ways of saying "if", but they carry an additional, temporal dimension[30]. It would be a distraction to deal with such temporal issues here. For that reason, it is useful to reformulate the sub-goals in more conventional logical terms. At the same time, we can take advantage of the reformulation to eliminate an ambiguity associated with the scope of the words "and when":

Sub-goals:

If the louse is hungry then it looks for food, and
If the louse is hungry and there is food ahead then it eats it, and
If the louse is tired then it rests, and
If the louse is threatened with attack then it defends itself, and
If the louse desires a mate then it looks for a mate, and
If the louse desires a mate and there is a mate ahead then it tries to make babies.

Unfortunately, the designer's work is not yet done. Some of the conclusions of the sub-goals include other goals (like looking for food, defending itself, and looking for a mate) that need to be reduced to

[30] It is interesting that both the temporal and logical interpretations of the ambiguous English word "then" are meaningful here.

still lower-level sub-goals[31]. Fortunately, for the designer, this is easy work. It takes just a little further backward reasoning and some logical simplification[32], to derive a specification that a behaviourist would be proud of:

New Goals:

If the louse is hungry and it is clear ahead
then the louse moves forward.

If the louse is hungry and there is an obstacle ahead and it isn't food
then the louse turns right.

If the louse is hungry and there is food ahead
then the louse stops and it eats the food.

If the louse is tired
then the louse rests.

If the louse is threatened with attack
then the louse runs away.

If the louse desires a mate and it is clear ahead
then the louse moves forward.

If the louse desires a mate and there is an obstacle ahead and it isn't a mate
then the louse turns right.

If the louse desires a mate and there is an obstacle ahead and it is a mate
then the louse stops and it tries to make babies.

The new goals specify the louse's input-output behaviour and can be implemented directly as a production system without memory. However, the new goals are potentially inconsistent. If the louse desires a mate and is hungry at the same time, then it may find itself in a situation, for example, where it has to both stop and eat and also turn right and look for a mate simultaneously. To avoid such inconsistencies, the louse would need to perform conflict resolution.

But if it's too much to expect the louse to reason logically, it's probably also too much to expect the louse to perform conflict resolution. And it's certainly far too much to expect it to apply Decision Theory to weigh the relative advantages of satisfying its hunger compared with those of satisfying its longing for a mate. The simplest solution is for the designer to make these decisions for the louse, and to
build them into the specification:

If the louse is hungry and is not threatened with attack and
it is clear ahead
then the louse moves forward.

If the louse is hungry and is not threatened with attack and
there is an obstacle ahead and it isn't food and it doesn't desire a mate
then the louse turns right.

---

[31] For simplicity, we assume that running away, resting and trying to make babies are all actions that the louse can execute directly without reducing them to lower-level sub-goals.
[32] The necessary simplification is to replace sentences of the form: "if A, then if B then C" by sentences "if A and B then C".

If the louse is hungry and is not threatened with attack and
there is food ahead
then the louse stops and eats the food.

If the louse is tired and is not threatened with attack and
is not hungry and does not desire a mate
then the louse rests.

If the louse is threatened with attack
then the louse runs away.

If the louse desires a mate and is not threatened with attack and
it is clear ahead
then the louse moves forward.

If the louse desires a mate and is not threatened with attack and
is not hungry and there is an obstacle ahead and it isn't a mate
then the louse turns right.

If the louse desires a mate and is not threatened with attack and
there is a mate ahead
then the louse stops and tries to make babies.

If the louse desires a mate and is hungry and
is not threatened with attack and
there is an obstacle ahead and it isn't a mate and it isn't food
then the louse turns right.

The new specification is a collection of input-output associations that give highest priority to reacting to an attack, lowest priority to resting when tired, and equal priority to mating and eating. Now the only situation in which a conflict can arise is if there is a mate and food ahead at the same time. Well, you can't always worry about everything.

The mind body problem

In general, a designer's job ends when she has constructed a declarative description of her object's input-output behaviour. How that behaviour is implemented inside the object is not her concern.

In computer science, this decoupling of an object's design from its implementation is called ***encapsulation***. The implementation is encapsulated inside the object. Objects can interact with other objects, taking only their input-output behaviour into account.

The notion of encapsulation partially vindicates the behaviourist's point of view. Not only is it impossible in many cases to determine what goes on inside another object, but for many purposes it is also unnecessary and even undesirable.

Our louse is no exception. It would be easy, given the input-output specification, to implement the louse's behaviour using a primitive production system without memory and without conflict resolution. But does the louse need to have a mind at all - to represent concepts such as hunger and food and to derive symbolic representations of its actions? Does

the louse really need to carry around all this mental baggage, when the necessary, instinctive behaviour can be hardwired, as a collection of input-output associations, directly into the louse's body instead[33]?

Similarly, a designer might specify the design of a thermostat in symbolic terms:

Goals:

If current temperature is T degrees and target temperature is T' degrees and T < T' - 2°
then the thermostat turns on the heat.

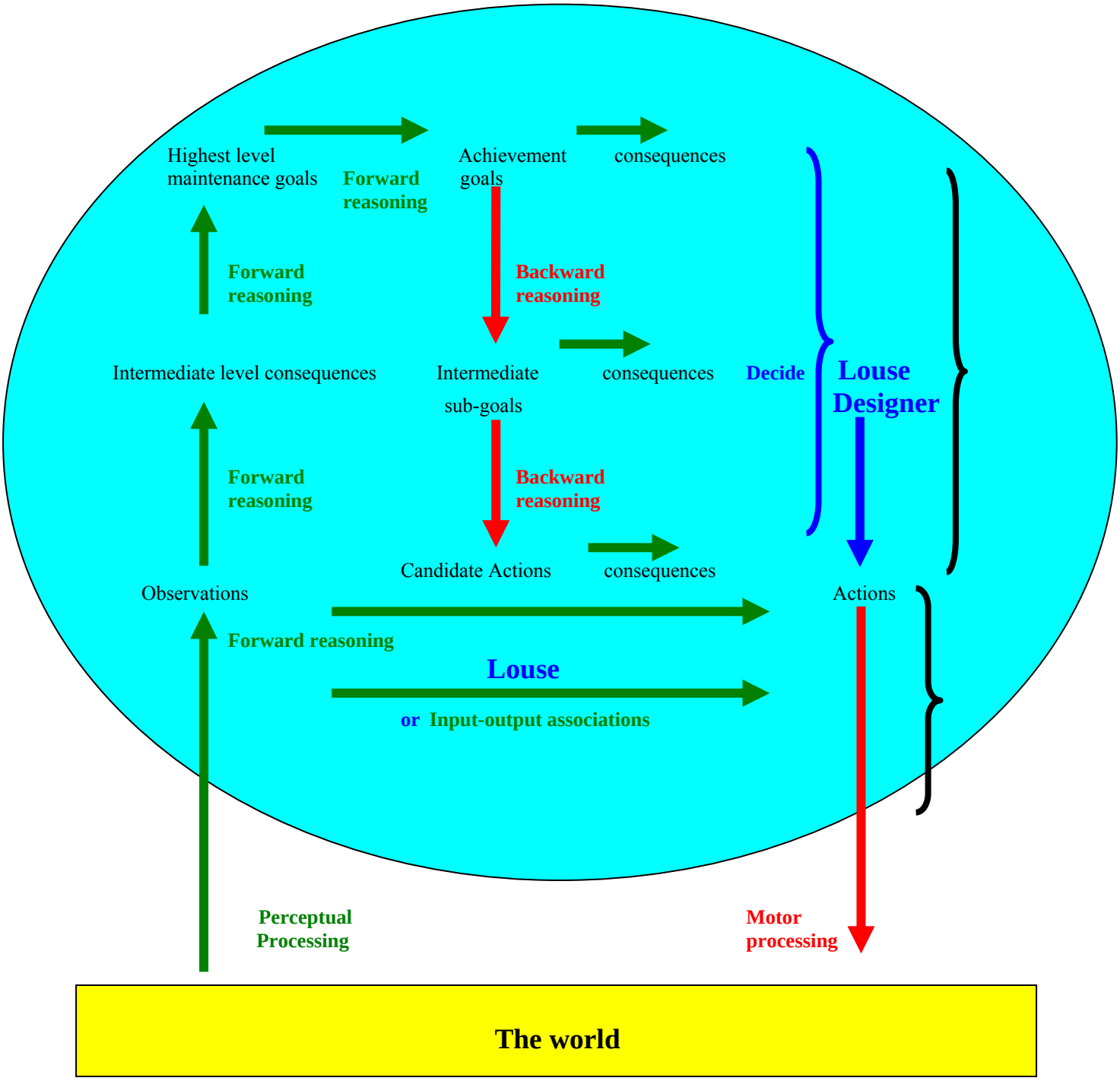
If current temperature is T degrees and target temperature is T' degrees and T > T' + 2°
then the thermostat turns off the heat.

But it doesn't follow that the thermostat needs to manipulate symbolic expressions to generate its behaviour. Most people would be perfectly happy if the design were implemented with a simple mechanical or electronic device.

In the same way that a thermostat's behaviour can be viewed externally in logical, symbolic terms, without implying that the thermostat itself manipulates symbolic expressions, our louse's behaviour can also be implemented as a collection of instinctive input-output associations in a body without a mind.

These possibilities can be pictured like this:

[33] This argument has been made, among others, by Rodney Brooks at MIT, who has implemented several generations of mindless, louse-like robots, which display impressively intelligent behaviour.


Highest level maintenance goals
Forward reasoning
Achievement goals
consequences
Forward reasoning
Backward reasoning
Intermediate level consequences
Intermediate sub-goals
consequences
Decide
Louse Designer
Forward reasoning
Backward reasoning
Candidate Actions
consequences
Observations
Actions
Forward reasoning
Louse
or Input-output associations
Perceptual Processing
Motor processing
The world

People are different

Although much of our human behaviour is instinctive and even mindless, we can sometimes step back from what we are doing, consciously reflect upon our goals, and to some extent control our behaviour to better achieve those goals. It is as though we could be both a louse and a louse designer at the same time.

That, in effect, is the ultimate goal of this book – to investigate how we can use logic to monitor our behaviour and how we can accomplish our goals more effectively as a result. For this purpose, we need to investigate the relationship between instinctive and logical thought. We have seen an example of this relationship in this chapter. But in this example the instinctive and logical levels were present in different individuals. In the next chapter we will see how they can be present in the same individual.

Chapter 7 Levels of Consciousness

There is a sense of consciousness that can be understood in both computational and logical terms. It is the sense in which an agent is ***conscious*** when it is ***aware*** of what it is doing and why it is doing it. Whether or not the agent is conscious in this sense, the external manifestation of its behaviour is the same. However, if the agent is conscious of what it is doing, then its behaviour is deliberate and controlled. If it is not conscious, then its behaviour is automatic and instinctive.

The computational interpretation of consciousness as awareness is that, when an agent is conscious, its behaviour is generated by a high level program, which manipulates symbols that have meaningful interpretations in the environment. However, when an agent is not conscious, then its behaviour is generated by a lower level program or physical device, whose structure is ultimately determined by the physical characteristics of the agent's body.

The logical interpretation is that, when an agent is conscious, its behaviour is generated by reasoning with goals and beliefs. When the agent is not conscious, then its behaviour is determined by lower-level input-output associations. These associations can be represented at different levels in turn, including both a logical, symbolic level and the lower, physical level of the agent's body.

These two interpretations coincide in computational logic.

Consciousness on the underground

Remember our last version of the London underground example:

Goal: If there is an emergency then I get help.

Beliefs: A person gets help if the person alerts the driver.
A person alerts the driver if the person presses the alarm signal button.
There is an emergency if there is a fire.
There is an emergency if one person attacks another.
There is an emergency if someone becomes seriously ill.
There is an emergency if there is an accident.
There is a fire if there are flames.
There is a fire if there is smoke.

A passenger can use the goal and the beliefs explicitly, reasoning forward from observations to recognise when there is an emergency and to derive the goal of getting help, and then reasoning backward, to get help by pressing the alarm signal button.

However, the passenger can generate the same behaviour using lower-level input-output associations or condition-action rules, which can also be represented as goals in logical form:

Goals: If there are flames then I press the alarm signal button.
If there is smoke then I press the alarm signal button.
If one person attacks another then I press the alarm signal button.

If someone becomes seriously ill then I press the alarm signal button.
If there is an accident then I press the alarm signal button.

The new goals are more efficient than the original goal and belief. They need only one step of forward reasoning to associate the appropriate action as output with the relevant observation as input. In this respect, they are like a program written in a lower-level language, which is more efficient than a program with the same functionality written in a higher-level language.

In fact, in this case, the two programs are written in the same logical language. However, the original program is written at a higher-level, which requires, not only greater computational resources, but also more sophisticated reasoning power.

In Computing, different levels of language and different levels of representation in the same language have different advantages and are complementary. Lower-level representations are more efficient. But higher-level representations are more flexible, easier to develop, and easier to change.

In this example, the lower-level representation lacks the awareness, which is explicit in the higher-level representation, of the goal of getting help, which is the purpose of pressing the alarm signal button. If something goes wrong with the lower-level representation, for example if the button doesn't work or the driver doesn't get help, then the passenger might not realise there is a problem. Also, if the environment changes, and newer and better ways of dealing with emergencies are developed, then it would be harder to modify the lower level representation to adapt to the change.

In Computing, typically, the higher-level representation is developed first, sometimes not even as a program but as an analysis of the program requirements.[34] This higher-level representation is then ***transformed***, either manually or by means of another program called a ***compiler***, into a lower-level, more efficiently executable representation.

The reverse process is also possible. Low-level programs can sometimes be ***decompiled*** into equivalent higher level programs. This is useful if the low-level program needs to be changed, perhaps because the environment has changed or because the program has developed a fault. The higher-level representation can then be modified and ***recompiled*** into a new, improved, lower-level form.

However, this reverse process is not always possible. Legacy systems, developed directly in low-level languages and modified over a period of many years, may not have enough structure to identify their goals unambiguously and to decompile them into high-level form. But even then it may be possible to decompile them partially and approximate them with higher-level programs. This process of rational reconstruction can help to improve the maintenance of the legacy system, even when wholesale reimplementation is not possible.

[34] As in the wood louse designer example.

Compiling by reasoning in advance

Some of the work of compiling a high-level program into a lower-level form can be done by performed in advance some of the computation that would otherwise be performed only when it is needed.

In the underground example, instead of waiting for an emergency to happen, a compiler can anticipate the need to reduce the conclusion of the top-level maintenance goal:

Goal: If there is an emergency then I get help.

by performing the necessary backward reasoning in advance. In two such inference steps, the goal can be transformed into a declarative version of a condition-action rule:

Goal: If there is an emergency then I press the alarm signal button.

Similarly, the compiler can anticipate the need to recognise there is an emergency by performing the necessary forward reasoning in advance. In two inference steps, corresponding to the two ways of recognising there is a fire, the compiler can derive the new beliefs:

Beliefs: There is an emergency if there are flames.
There is an emergency if there is smoke.

In five further inference steps, corresponding to the five kinds of emergency, the compiler can obtain the simple input-output associations we saw before:

Goals: If there are flames then I press the alarm signal button.
If there is smoke then I press the alarm signal button.
If one person attacks another then I press the alarm signal button.
If someone becomes seriously ill then I press the alarm signal button.
If there is an accident then I press the alarm signal button.

This representation is as low as a representation can go, while still remaining in logical form. However, it is possible to go lower, if these associations are implemented by direct physical connections between the relevant parts of the human sensory and motor systems. This is like implementing software in hardware.

Combining deliberative and intuitive thinking

In Cognitive Psychology, a similar distinction has been made between ***deliberative*** and ***intuitive*** thinking. Deliberative thinking, which is self-aware and controlled, includes logical thinking. Intuitive thinking, which is opaque and automatic, extends perceptual processing to subconscious levels of thought. Cognitive Psychologists have proposed dual process models of human thinking in which the two kinds of thinking interact.

The simplest interaction occurs when deliberative thinking migrates to the intuitive level over the course of time, as for example when a person learns to use a keyboard,

play a musical instrument or drive a car. This migration from deliberative to intuitive thinking is like compiling a high level program into a lower-level program – not by reasoning in advance, but by reasoning after the fact. After a combination of high-level, general-purpose procedures has been used many times over, the combination is collapsed into a lower-level shortcut. The shortcut is a specialised procedure, which achieves the same result as the more general procedures, only more efficiently.

It is sometimes possible to go in the opposite direction, reflecting upon subconscious knowledge and representing it in conscious, explicit terms – for example when a linguist tries to construct a formal grammar for a language. Whereas the native speaker of the language knows the grammar tacitly and subconsciously, the linguist formulates an explicit grammar for the language. Teachers can teach the resulting explicit grammar to students. With sufficient practice, the students may eventually compile the explicit grammar into their own subconscious, and learn to speak the language more efficiently.

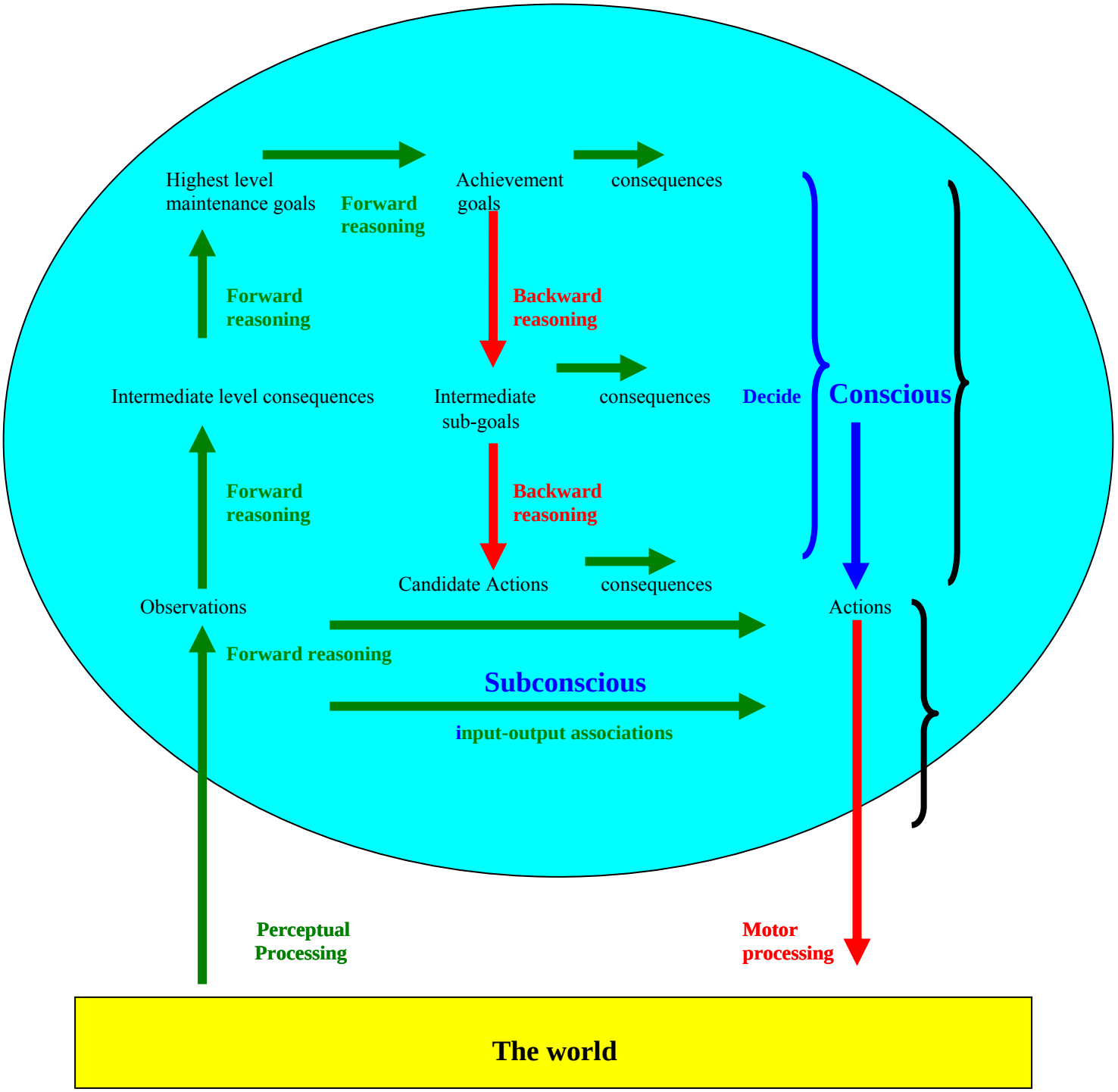
In the same way that many low-level programs can not be completely decompiled, but can only be approximated by higher-level programs, it seems likely that much of our subconscious thinking can only be approximated by conscious thought. The formal grammars of the linguist are an example.

In human thinking, the two levels of thought can operate in tandem. In the Kahneman-Tversky dual process model, the intuitive, subconscious level "quickly proposes intuitive answers to judgement problems as they arise", while the deliberative, conscious level "monitors the quality of these proposals, which it may endorse, correct, or override"[35]. The use of formal grammar to monitor and correct the instinctive use of natural language is a familiar example.

Logic as the higher-level language of thought

The general view of logic that emerges from these considerations is that logic is the higher level language of thought, at which thoughts are deliberate, controlled and conscious. At this higher level, logical reasoning reduces goals to sub-goals, derives consequences of observations and infers consequences of candidate actions. This reasoning can be performed only when it is needed, or it can be performed in advance. When it is performed in advance, it transforms a higher-level representation of goals and beliefs into a more efficient, lower-level representation. At this lower-level of representation, behaviour is instinctive, automatic and subconscious. These relationships can be pictured in this way:

[35] "Representativeness revisited: Attributive substitution in intuitive judgement", Daniel Kahneman and Shane Frederick. In *Heuristics of Intuitive Judgement: Extensions and Applications*, Cambridge University Press, 2002.


Highest level maintenance goals
Forward reasoning
Achievement goals
consequences
Forward reasoning
Backward reasoning
Intermediate level consequences
Intermediate sub-goals
consequences
Decide
Conscious
Forward reasoning
Backward reasoning
Candidate Actions
consequences
Observations
Actions
Forward reasoning
Subconscious
input-output associations
Perceptual Processing
Motor processing
The world

Chapter 8 The Prisoner's Dilemma

Suppose, in your desperation to get rich as quickly as possible, you consider the various alternatives, infer their likely consequences and decide that the best alternative is to rob the local bank. You recruit your best friend, Keith, well known for his meticulous attention to detail, to help you plan and carry out the crime. Thanks to your joint efforts, you succeed in breaking into the bank in the middle of the night, opening the safe, and making your get-away with a cool million pounds (approximately 1.65 million dollars at the time of writing) in the boot (trunk) of your car.

Unfortunately, years of poverty and neglect have left your car in a state of general disrepair, and you are stopped by the police for driving at night with only one headlight. In the course of a routine investigation, they discover the suitcase with the cool million pounds in the boot. You plead ignorance of any wrong doing, but they arrest you both anyway on the suspicion of robbery.

Without witnesses and without a confession, the police can convict you and your friend only of the lesser offence of possessing stolen property, which carries a penalty of one year in jail. However, if one of you turns witness against the other, and the other does not, then the first will be released free of charge, and the second will take all of the blame and be sentenced to six years in jail. If both of you turn witness, then the two of you will share the blame and each be sentenced to three years in jail.

This is an example of the classical Prisoner's Dilemma. In Game Theory, the problem of deciding between alternative actions is often represented as a table, in which the rows and columns represent the actions of the players and the entries represent the resulting outcomes. In this case, the table looks like this:

	You turn witness.	You do not turn witness.
Keith turns witness.	You get 3 years in jail. Keith gets 3 years in jail.	You get 6 years in jail. Keith gets 0 years in jail.
Keith does not turn witness.	You get 0 years in jail. Keith gets 6 years in jail.	You get 1 year in jail. Keith gets 1 year in jail.

If the two prisoners are able to consult with one another, then they will soon realise that the best option for both of them is not to turn witness against the other. To prevent this, the police separate them before they have a chance to consult. Thus each prisoner has to decide what to do without knowing what the other prisoner will do.

The Logic of the Prisoner's Dilemma

The Prisoner's Dilemma has a natural representation in terms of the prisoners' goals and beliefs:

Goal: If I am arrested, then I turn witness or I do not turn witness.

Beliefs: I am arrested.

A prisoner gets 0 years in jail
if the prisoner turns witness
and the other prisoner does not turn witness.

A prisoner gets 6 years in jail
if the prisoner does not turn witness
and the other prisoner turns witness.

A prisoner gets 3 years in jail
if the prisoner turns state witness
and the other prisoner turns witness.

A prisoner gets 1 year in jail
if the prisoner does not turn witness
and the other prisoner does not turn witness.

This assumes, of course, that the prisoners believe what they are told by the police. It also assumes that both prisoners know that the same deal has been offered to the other prisoner. However, the analysis at the end of this chapter can easily be modified to deal with other cases.

The Logic of Games

In general, any two-person game represented as a table can also be represented as goals and beliefs. For example, the table:

	First player does action A.	First player does action B.
Second player does action C.	First player gets outcome AC. Second player gets outcome CA.	First player gets outcome BC. Second player gets outcome CB.
Second player does action D.	First player gets outcome AD. Second player gets outcome DA.	First player gets outcome BD. Second player gets outcome DB.

can be represented by goals and beliefs, which in the case of the first player are:

Goal: First player does action A or First player does action B.

Beliefs: First player gets outcome AC
if First player does action A
and Second player does action C.

First player gets outcome BC
if First player does action B
and Second player does action C.

First player gets outcome AD
if First player does action A
and Second player does action D.

First player gets outcome BD
if First player does action B
and Second player does action D.

Depending on the circumstances, a player may or may not know the outcomes for the other player.

Should you carry an umbrella?

Before discussing how to solve the prisoner's dilemma, it is useful to compare it with the seemingly unrelated problem of deciding whether or not to take an umbrella when you leave home in the morning.

We can represent the umbrella problem as a game against nature:

	I take an umbrella.	I do not take an umbrella.
It will rain.	I stay dry. I carry the umbrella.	I get wet.
It will not rain.	I stay dry. I carry the umbrella.	I stay dry.

We can represent the agent's side of the game in terms of the agent's goals and beliefs[36]:

Goal: If I go outside, then I take an umbrella or I do not take an umbrella.

Beliefs: I go outside.

I carry an umbrella
if I take the umbrella.

I stay dry
if I take the umbrella.

I stay dry
if it will not rain.

[36] Notice that the representation in terms of beliefs is more informative than the game representation, because it indicates more precisely than the game representation the conditions upon which the outcome of an action depends. For example, the representation in terms of beliefs indicates that staying dry depends only on taking an umbrella and not on whether or not it rains:

I get wet
if I do not take an umbrella
and it will rain.

You can control whether or not you take an umbrella, but you can not control whether or not it will rain. At best, you can only try to estimate the probability of rain.

This should sound familiar. In chapter 5, when considering whether or not to rob a bank, I wrote:

> "You can control whether or not you try to rob a bank. But you can't control whether you will be caught or be convicted. Not only are these possibilities beyond your control, but you can not even predict their occurrence with any certainty. At best, you can only try to estimate their probability."

It's the same old story. To decide between different actions, you should infer their consequences, judge the utility and probability of those consequences, and choose the action with highest overall expected utility.

Suppose you judge that the benefit of staying dry, if it rains, is significantly greater than the cost in inconvenience of taking an umbrella, whether or not it rains.[37] Then you should decide to take the umbrella, if you estimate that the probability of rain is relatively high. But, you should decide not to take the umbrella, if you estimate that the probability of rain is relatively low.

Applying Decision Theory to Taking an Umbrella

This kind of "thinking"[38], which combines judgements of utility with estimates of probability, is formalised in the field of Decision Theory. According to the norms of Decision Theory, you should weight the utility of each alternative outcome of an action by its probability, and then sum all of the alternative weighted utilities to measure the overall expected utility of the action. You should then choose the action with highest expected utility[39].

In the case of deciding whether or not to take an umbrella, suppose you judge that

The benefit of staying dry is D.
The cost of carrying an umbrella is C.
The cost of getting wet is W.
The probability that it will rain is P,
and therefore that it will not rain is $(1 - P)$.

---

[37] In general, assuming we can quantify benefits and costs in the same units, then utility = benefits – costs.

[38] According to Baron's "Thinking and Deciding", this is not "thinking" at all, but "deciding" between different options. It is an interesting question to what extent "deciding" might also involve "thinking" at a different, perhaps meta-level. More about this later.

[39] In mathematical terms, if an action has n alternative outcomes with ***utilities*** $u_1, u_2, ..., u_n$ with respective ***probabilities*** $p_1, p_2, ..., p_n$ then ***the expected utility of the action*** is $p_1.u_1 + p_2.u_2 + ... + p_n.u_n$.

Then the expected utility of taking the umbrella is
the benefit of staying dry
minus the cost of carrying the umbrella
$= D - C$.

The expected utility of not taking the umbrella is
the benefit of staying dry if it doesn't rain
minus the cost of getting wet if it does rain
$= (1 - P) \cdot D - P \cdot W$.

So, for example, if the benefit of staying dry is worth 1 candy bar, the cost of carrying an umbrella is worth 2 candy bars, and the cost of getting wet is worth 9 candy bars, then

$D = 1$
$C = 2$
$W = 9$.

The expected utility of taking the umbrella $= -1$.
The expected utility of not taking the umbrella $= (1 - 10P)$.

Therefore, if the probability of rain is greater than .2, then you should take an umbrella; and if it is less than .2, then you shouldn't take an umbrella. If it is exactly .2, then it makes no difference, measured in candy bars, whether you take an umbrella or not.

The use of Decision Theory is ***normative***, in the sense that its estimations and computations are an ideal, which we only approximate in reality. In Real Life, we tend to compile routine decisions into simpler rules, represented by means of goals and beliefs. For example:

Goals: If I go outside
and it looks likely to rain,
then I take an umbrella.

If I go outside
and it looks unlikely to rain,
then I do not take an umbrella.

Beliefs: It looks likely to rain if there are dark clouds in the sky.
It looks likely to rain if it is forecast to rain.

It looks unlikely to rain if there are no clouds in the sky.
It looks unlikely to rain if it is forecast not to rain.

Solving the Prisoner's Dilemma

Just as in the case of deciding whether to take an umbrella when you go outside, you can control your own actions in the Prisoner's Dilemma, but you can not control the

world around you. In this case you can not control the actions of the other prisoner. However, you can try to predict them as best as possible.

Suppose you take a Decision Theoretic approach and judge:

The utility of your getting N years in jail is $-N$.
The probability that Keith turns witness is P,
and therefore that Keith does not turn witness is $(1 - P)$.

Then the expected utility of your turning witness
is 3 if Keith turns witness,
and 0 if he does not
$= -3 \cdot P + 0 \cdot (1 - P)$
$= -3 \cdot P$.

The expected utility of your not turning witness
is –6 if Keith turns witness,
and –1 if he does not
$= -6 \cdot P - 1 \cdot (1 - P)$
$= -1 - 5 \cdot P$.

But $-3 \cdot P > -1 - 5 \cdot P$, for all values of P. Therefore, no matter what the probability P that Keith turns witness, you are always better off by turning witness yourself.

Unfortunately, if Keith has the same beliefs, goals and utilities as you, then he will similarly decide to turn witness against you, and in which case both of you will get a certain 3 years in jail. You would have been better off if you forgot about Decision Theory, took a chance, and both of you refused to turn witness against the other, in which you would have both gotten only 1 year in jail.

But there is a different moral that you could draw from the story – that the fault lies, not with Decision Theory, but with your own selfish judgement of utility.

Suppose, instead of carrying only about yourself, you care about both yourself and Keith equally, and you judge:

The utility of your getting N years in jail and of
Keith getting M years in jail is $-(N + M)$.

Then the expected utility of your turning witness
is –6 if Keith turns witness, and
is –6 if he does not
$= -6 \cdot P - 6 \cdot (1 - P)$
$= -6$.

The expected utility of your not turning witness
is –6 if Keith turns witness, and
is –2 if he does not
$= -6 \cdot P - 2 \cdot (1 - P)$
$= -2 - 4 \cdot P$.

But $-6 \geq -2 - 4 \cdot P$, for all values of P. Therefore, no matter what the probability P that Keith turns witness, there is never any advantage in you turning witness yourself.

Now, if Keith has the same beliefs, goals and utilities as you, then he will similarly decide not to turn witness against you, in which case both of you will get a certain one year in jail.

But carrying equally about both yourself and Keith is probably unrealistic. To be more realistic, suppose instead that you care about Keith only half as much as you do about yourself:

The utility of your getting N years in jail and of
Keith getting M years in jail is $-(N + 1/2 \cdot M)$.

Then the expected utility of your turning witness
is –4.5 if Keith turns witness, and
is –3 if he does not
$= -4.5 \cdot P - 3 \cdot (1 - P)$
$= -3 - 1.5 \cdot P$.

The expected utility of your not turning witness
is –6 if Keith turns witness, and
is –1.5 if he does not
$= -6 \cdot P - 1.5 \cdot (1 - P)$
$= -1.5 - 4.5 \cdot P$.

But $= -3 - 1.5 \cdot P = -1.5 - 4.5 \cdot P$ when $P = .5$. Therefore, if you judge the probability of Keith turning witness is less than .5, then you should not turn witness. But if you judge that the probability is greater than .5, then you should turn witness – tit for tat.

Just as in the case of deciding whether to take an umbrella when you go outside, these calculations are a normative ideal, which we tend only to approximate in practice. In Real Life, we tend to compile our decisions into rules of behaviour, represented by goals and beliefs. For example:

Goals: If I am offered a deal
and the deal benefits me
and the deal harms another person more than it benefits me
and the person is my friend
then I reject the deal.

If I am offered a deal
and the deal benefits me
and the deal harms another person
and the person is not my friend
then I accept the deal.

These rules are not very subtle, but it should be clear that they can be refined, both to deal with other cases and to distinguish more subtly other characteristics of the deal under consideration.

Conclusions

There are three conclusions. The first concerns the Prisoner's Dilemma itself – that it pays to co-operate with other agents, and not to try only to optimise our own narrow self-interests. This conclusion is, of course, well known in the literature about the Prisoner's Dilemma. What may be less well known is the extent to which the benefits of co-operation can often be obtained simply by incorporating concern for the well-being of others in the utility function.

The second conclusion is much more general – that to decide between different courses of action we need, not only to judge the costs and benefits of our actions, but also to estimate the probability of circumstances outside control. We have seen this before, but it needs to be emphasised again, not only because it is so important, but also because it has been largely ignored in traditional logic. The approach taken in the chapter shows one way in which logic and probability can usefully be combined.

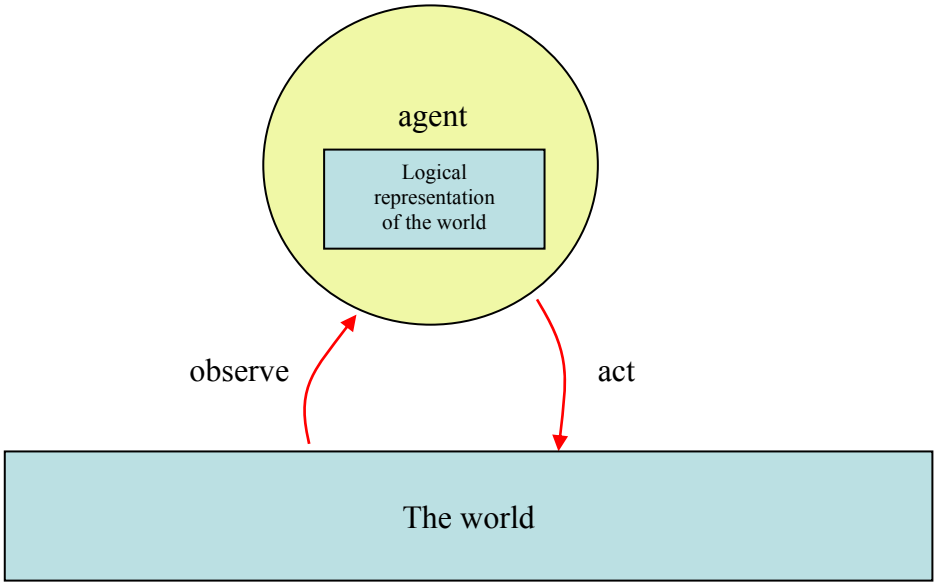
The third conclusion is more subtle. It is that the computations of Decision Theory are a normative ideal, which we often approximate in Real Life by using simpler rules represented by goals and beliefs. This relationship between "higher-level" Decision Theory and "lower-level" decision rules is like the relationship between higher-level logical representations and lower-level input-output associations.

As we have seen, we can compile logical representations of goals and beliefs into input-output associations, and sometimes decompile associations into logical representations. Moreover, it seems that in human thinking, the two levels of thought can operate in tandem. The input-output associations efficiently propose candidate outputs in response to inputs, while reasoning about goals and beliefs monitors the quality of those responses.

There seems to be a similar relationship between Decision Theory and decision rules. Rules can be executed efficiently, but Decision Theory gives better quality results. As in the case of higher and lower level representations, Decision Theory can be used to monitor the application of rules and propose modifications of the rules when they need to be changed, either because they are faulty or because the environment itself has changed. In his book, *Thinking and Deciding*, Baron discusses similar relationships between normative, prescriptive and descriptive approaches to decision making in detail.

Chapter 9 The Changing World

I have argued that the purpose of logic is to help an agent survive and prosper in the world. Logic serves this task by providing the agent with a means for constructing symbolic representations of the world and for processing those representations to reason about the world. We have pictured this relationship between logic and the world like this:



However, we have ignored any considerations about the way in which logical representations are related to the structure of the world. There are two parts to these considerations: what is the relationship between logic and static states of the world, and what is the relationship between logic and change. We shall consider these two issues now.

World structures

The relationship between logic and the world can be seen from two points of view. Seen from the perspective of the world, sentences of logic represent certain features of the world. Seen from the perspective of logic, the world gives meaning to sentences. This second viewpoint is also called "***semantics***".

Although a real agent needs to worry only about the real world, it is convenient to consider other possible worlds, including artificial and imaginary worlds, like the world in the story of the fox and the crow. Both kinds of world, both real and possible, can be understood in similar terms, as world structures. A ***world structure*** is just a collection of ***individuals*** and ***relationships*** among them. Relationships are also called "***facts***".
In traditional logic, "world structures" are usually called "interpretations", "models", or sometimes "possible worlds". For simplicity, properties of individuals are also regarded as relationships.

Traditional logic has a very simple semantics, in terms of whether or not ***sentences*** are ***true*** or ***false*** in a world structure. Sentences that are true are normally more useful to an agent than sentences that are false.

A world structure generally corresponds to a single, static state of the world. For example:

> In the story of the fox and the crow, the fox, crow, cheese, tree, ground under the tree, and airspace between the crow and the ground can be regarded as individuals; and someone having something can be regarded as a relationship between two individuals. The sentence "The crow has the cheese." is true in the world structure at the beginning of the story and false in the world structure at the end of the story.

An ***atomic sentence is true*** in a world structure if the relationship it expresses holds in the world structure, and otherwise it is false.

The simplest way to represent a world structure in logical terms is to represent it by the set of all atomic sentences that are true in the structure – in this example we might represent the world structure at the beginning of the story by the atomic sentences:

The crow has the cheese.
The crow is in the tree.
The tree is above the air.
The air is above the ground.
The tree is above the ground.
The fox is on the ground.

The difference between such atomic sentences and the world structure they represent is that in a world structure the individuals and the relationships between them have a kind of external existence that is independent of language. Atomic sentences, on the other hand, are merely symbolic expressions that stand for such external relationships. In particular, words and phrases like "the crow", "the cheese", "the tree", etc. are ***names*** of individuals and "has", "is in", etc. are ***names*** of relations between individuals.

The attraction of logic as a way of representing the world lies mainly its ability to represent regularities in world structures by means of non-atomic sentences. For instance, in the example above:

One object is above another object
if the first object is above a third object
and the third object is above the second.

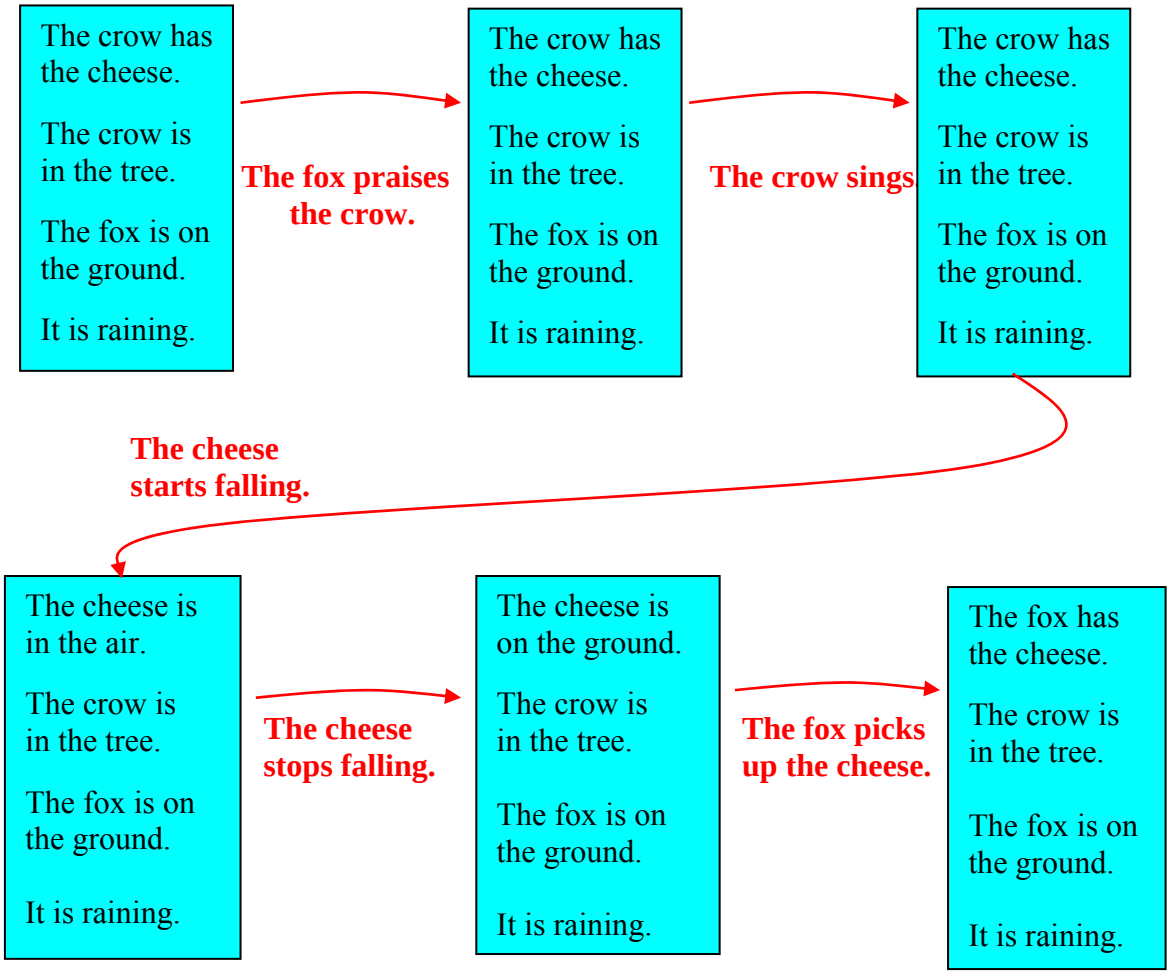
The truth value of non-atomic sentences is defined in terms of the truth values of simpler sentences - for example, by means of such "meta-sentences" as:

A sentence of the form "conclusion if conditions" is true
if "conditions" is true and "conclusion" is true
or "conditions" is not true.

A sentence of the form “everything has property P” is true
if for every thing T in the world, “T has property P” is true.

Dynamic world structures

World structures in traditional logic are ***static***, in the sense that they represent a single, static state of the world. One natural way to understand change is to view actions and other events as causing a change of state from one static world structure to another. For example:



This view of change is the basis of the semantics of modal logic. In ***modal logic***, sentences are given a truth value relative to a static world structure embedded in a collection of world structures linked by state-transforming events. Syntactic expressions such as “in the past”, “in the future”, “after”, “since” and “until” are treated as ***modal operators***, which are logical connectives, like “and”, “or”, “if”, “not” and “all”. The truth value of sentences containing such modal operators is defined in terms of the truth values of simpler sentences - for example, by means of such meta-sentences as:

A sentence of the form “in the future P” is true at a world structure S
if there is a world structure S’

that can be reached from S by a sequence of state-transforming events
and the sentence "P" is true at S'.

For example, in modal logic, it is possible to express the sentence

"In the future the crow has the cheese."

This sentence is true in the world structure at the beginning of the story and false in the world structure at the end of the story (assuming that the world ends after the fox picks up the cheese).

One objection to the modal logic approach is that its semantics defined in terms of the truth of sentences at a world structure in a collection of world structures linked by state-transforming events is too complicated. One alternative, which addresses this objection, is to simplify the semantics and increase the expressive power of the logical language by treating states of the world as individuals. To treat something as an individual, as though it exists, is to ***reify*** it; and the process itself is called ***reification***.

The advantage of reification is that it makes talking about things a lot easier. The disadvantage is that it makes some people very upset. It's alright to talk about material objects, like the fox, the crow and the cheese, as individuals. But it's something else to talk about states of the world and other similarly abstract objects as though they too were ordinary individuals.

The situation calculus

The ***situation calculus***[40], developed by McCarthy and Hayes in Artificial Intelligence, shares with modal logic the same view of change as transforming one state of the world into another, but it reifies states as individuals. As a consequence, world structures are dynamic, because they include state transitions as relationships between states.

For example, in the situation calculus, in the story of the fox and the crow, there is only one world structure and it contains, in addition to ordinary individuals, individuals that are global states. It is possible to express such sentences as:

"The crow has the cheese in the state at the beginning of the story."

"The crow has the cheese in the state after the fox picks up the cheese,
after the cheese stops falling,
after the cheese starts falling,
after the crow sings,
after the fox praises the crow,
after the state at the beginning of the story."

The first of these two sentences is true and the second is false in the situation calculus world structure.

[40] A ***situation*** is just another name for global state.

Reifying states of the world as individuals makes it possible to represent and reason about the effect of actions on states of affairs. If we also reify "facts", then this representation can be formulated as two situation calculus axioms:

> A fact holds in the state of the world after an action,
> if the fact is initiated by the action
> and the action is possible in the state before the action.
>
> A fact holds in a state of the world after an action,
> if the fact held in the state of the world before the action
> and the action is possible in the state before the action
> and the fact is not terminated by the action.

Our original version of the story of the fox and the crow can be reformulated in situation calculus terms (simplifying the first axiom by particularising it to this special case):

> An animal has an object
> in the state of the world after the animal picks up the object
> if the animal is near the object
> in the state of the world before the animal picks up the object
>
> I am near the cheese
> in the state of the world after the crow sings
> if the crow has the cheese
> in the state of the world before the crow sings
>
> The crow sings
> in the state of the world after I praise the crow.

In theory, an agent, such as the fox, could include such axioms among its beliefs, to plan its actions, infer their consequences, and infer the consequences of other agents' actions. In practice, however, the use of the second axiom (called "***the frame axiom***"), to reason about facts that are not affected by actions, is computationally explosive. This problem, called "***the frame problem***", is often taken to be an inherent problem with the use of logic to reason about change.

Arguably, it is not logic that is the source of the problem, but the situation calculus view of change, which it shares with modal logic and which is too global. Every action, no matter how isolated, is regarded as changing the entire state of the world. Even worse than that, to reason about the state of the world after a sequence of actions, it is necessary to know all the other actions that take place throughout the entire world in the meanwhile.

Thus to reason about the state of the world after the fox praises the crow, the crow sings, the cheese falls and the fox picks up the cheese, it is necessary to know and reason about everything that has happened everywhere else in the world between the beginning and the end of the story. This kind of thinking is not so difficult in the

imaginary world of the fox and the crow, but it is clearly impossible for a real agent living in the real world.

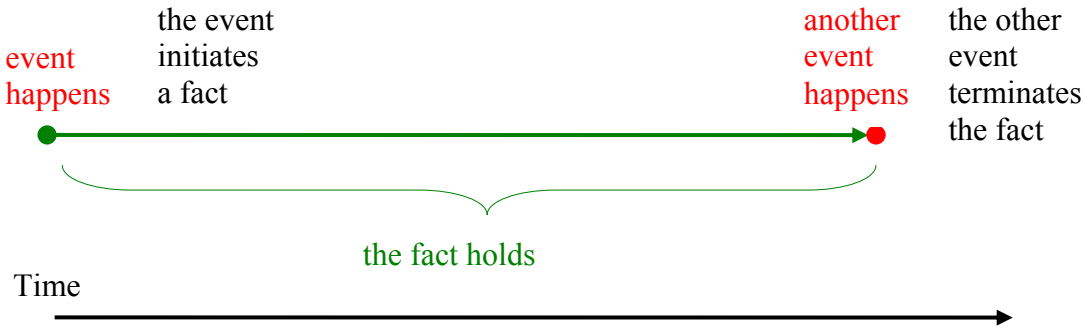
An event-oriented approach to change

One alternative is to abandon the view that actions transform global states of the world and replace it with the view that actions and other events can occur simultaneously in different parts of the world, independently and without affecting other parts of the world. In this alternative approach, the focus is on the occurrence of events and on the effect of events on local states of affairs.

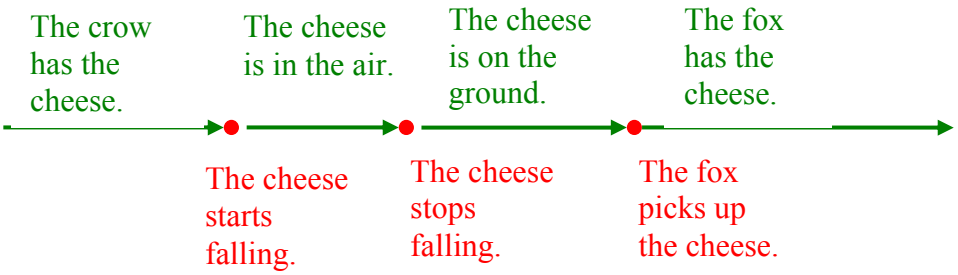
Events include both ordinary actions, which are performed by agents, and other events, like the cheese landing on the ground, which can be understood metaphorically as actions that are performed by inanimate objects.

For simplicity, we can assume that events occur instantaneously. For this purpose, an event that has duration can be decomposed into an instantaneous event that starts it, followed by a state of continuous change, followed by an instantaneous event that ends it. Thus the cheese falling to the ground can be decomposed into the instantaneous event in which the cheese starts to fall, which initiates the state during which the cheese is actually falling, which is terminated by the instantaneous event in which the cheese lands.

Events initiate and terminate relationships among individuals. These relationships, together with the periods for which they hold, can be regarded as ***local states*** of affairs. We can picture such a local state and the events that initiate and terminate it like this:



In the story of the fox and the crow, we can picture the effect of events on the state of the cheese like this:



A simplified calculus of events

Although we can represent world structures by atomic sentences, logic allows us to represent them more compactly by means of non-atomic sentences. In particular, we can derive information about local states of affairs from information about the occurrence of events, by means of the following event calculus axiom[41]:

A fact holds at a point in time,
if an event happened earlier
and the event initiated the fact
and there is no other event
that happened after the initiating event and before the time point and
that terminated the fact.

Because this axiom uses information about the occurrence of events to "calculate" information about local states of affairs, we call it the "***event calculus***".

The event calculus can be used, like the situation calculus, by an agent to plan its actions, infer their consequences, and infer the consequences of other agents' actions. Because it requires only localised knowledge of the affect of events, it is potentially more practical than the situation calculus.

To apply the event calculus in practice, it needs to be augmented with other axioms that define initiation, termination and temporal order. In the case of the changing location of the cheese in the story of the fox and the crow, we need information about the events that affect that location – for example:

The cheese falls at time 3.
The cheese lands at time 5.
The fox picks up the cheese at time 8.

We need to know what local states of affairs such events initiate and terminate:

The falling of an object initiates the fact that the object is in the air.
The landing of an object initiates the fact that the object is on the ground.

[41] It is convenient to adopt the convention that a fact holds after the event that initiates it, but at the time of the event that terminates it.

The picking up of an object by an agent initiates
the fact the agent has the object.

We also need some explanation for the fact that the crow has the cheese at the beginning of the story. This can be given, for example, by assuming an additional event, such as:

The crow picks up the cheese at time 0.

Finally, we need to be able to determine temporal relationships between time points and events. Because, in this example, we conveniently used numbers to name time points, we can do this with simple arithmetic. However, because the event calculus axiom is vague about time, we can use any system for measuring time, as long as we can then determine when one event occurs before another and when a time point is after an event. Whether we use numbers, dates and/or clock time is not important.

Keeping Track of Time

What is important is to keep track of time, to make sure that you do what you need to do before it is too late. So, if you are hungry, then you need to get food and eat it before you collapse from lack of strength. If a car is rushing towards you, then you need to run out of the way before you get run over. If you have a 9:00 appointment at work, then you need to get out of bed, wash, eat, dress, journey to work, and arrive before 9:00.

To get everything done in time, you need some kind of internal clock, both to timestamp externally observed events and to compare the current time with the deadlines of any internally derived future actions. This creates yet more work for the agent cycle:

To cycle,
observe the world, record any observations,
together with the time of their observation,
think,
decide what actions to perform, choosing only actions
that have not exceeded their deadline,
act,
cycle again.

Consider, for example, the problem of the fox when she becomes hungry. When her body signals that she is hungry, she needs to estimate how long she can go without eating and derive the goals of getting food and eating it before it is too late. One way for her to do this is to use a maintenance goal, with an explicit representation of time:

If I am hungry at time T_{hungry}
and I will collapse at a later time $T_{collapse}$ if I don't eat
then I have food at a time T_{food}
and I eat the food at the time T_{food}
and T_{food} is after T_{hungry} but before $T_{collapse}$.

She also needs to be able to deal with any attack from the local hunt:

> If the hunters attack me at time T_{attack}
> and they will catch me at a later time T_{catch} if I don't run away
> then I run away from the hunters at a time T_{run}
> and T_{run} is after T_{attack} but before T_{catch}.

Suppose, the fox is both hungry and under attack by the hunt at the same time. Then the fox needs to do a quick mental calculation, to estimate both how much time she has to find food and how much time she has to run away. She needs to judge the probability and utilities of the two different actions, and schedule them to maximise their overall expected utility. If the fox has done her calculations well and is lucky with the way subsequent events unfold, then she will have enough time both to satisfy her hunger and to escape from attack. If not, then either she will die of starvation or she will die from the hunt.

A critic might object, however, that this kind of reasoning is an unrealistic, normative ideal, which is better suited to a robot than to an intelligent biological being. An ordinary person, in particular, would simply give higher priority to escaping from attack than to satisfying its hunger. A person's maintenance goals would be "rules of thumb" that might look more like this:

> If I am hungry at time T_{hungry}
> then I have food at a time T_{food}
> and I eat the food at the time T_{food}
> and T_{food} is as soon as possible after T_{hungry}.

> If someone attacks me at time T_{attack}
> then I run away from the attackers at a time T_{run}
> and T_{run} is immediately after T_{attack} .

Thus assuming that you are a person who is hungry and attacked at the same time, say time 1 arbitrarily, your goals would look like this:

> I have food at a time T_{food}
> I eat the food at the time T_{food}
> I run away from the hunters at a time T_{run}
> and T_{run} is immediately after time 1 .
> and T_{food} is as soon as possible after 1 .

It would then be an easy matter for you to determine not only that T_{run} should be before T_{food} but that T_{run} should be the next moment in time.

It would be the same if you were attacked after you became hungry, but before you have succeeded in obtaining food. You would run away immediately, and resume looking for food only after (and if) you have escaped from attack.

Rules of thumb give a quick and easy result, which is not always optimal. If you were running away from attack and you noticed a piece of cheese on the ground, a

normative calculation might determine that you have enough time both to pick up the cheese and to continue running and escape from attack. But rules of thumb, which are designed to deal with the most commonly occurring cases, are less likely to recognise this possibility.

The relationship between normative calculation and rules of thumb is the same as the relationship between deliberate and intuitive thinking that we discussed in the chapter about levels of consciousness.

Conclusion

The world is a difficult place, which doesn't stand still. By the time you've thought about one problem, it throws up another one that is even more urgent. It trips you up and it keeps you on your toes.

We do our best to get on top of it, by forming mental representations of the world. But it doesn't make it easy. It wipes out its past and conceals its future, revealing only the way it is here and now.

In the struggle to survive and prosper, we use our memory of past observations, to generate hypothetical beliefs, to explain the past and predict the future. We compare these predictions with reality and revise our beliefs if necessary. This process of hypothesis formation and belief revision takes place in addition to the agent cycle, as times when the world slows down long enough to take stock. It is one of the many issues that we have yet to discuss in greater detail later in the book.

Chapter 10 Logic and Objects

What is the difference between the fox and the crow, on the one hand, and the cheese, on the other? Of course, the fox and crow are animate, and the cheese is inanimate. Animate things include agents, which observe changes in the world and perform their own changes on the world. Inanimate things are entirely passive.

But if you were an Extreme Behaviourist, you might think differently. You might think that the fox, the crow, and the cheese are all simply ***objects***, distinguishable from one another only by their different input-output behaviours:

> If the fox sees the crow and the crow has food in its mouth,
> then the fox praises the crow.
>
> If the crow is praised by the fox,
> then the crow sings.
>
> If the cheese is in the crow's beak and the crow sings,
> then the cheese falls to the ground.
>
> If the cheese is next to the fox,
> then the fox picks up the cheese.

Extreme Behaviourism was all the rage in Psychology in the mid-20th century. It has been the rage in Computing for approximately the past twenty years, in the form of ***Object-Orientation.***

It's easy to make fun of yesterday's Extreme Behaviourists. But it's not so easy to dismiss today's Object-Orientated Computer Scientists and Software Engineers. Object-Orientation today dominates every aspect of Computing: from modelling the system environment, through specifying the system requirements, to designing and implementing the software and hardware.

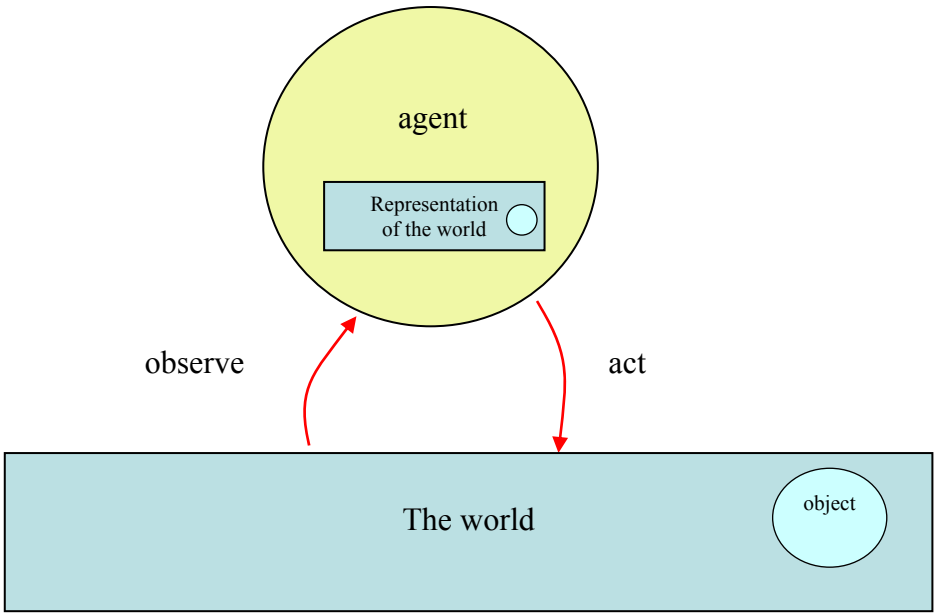
For a while in the 1980s, it looked as though Computational Logic might come to occupy the central role in Computing that Object-Orientation occupies today. Unfortunately for Logic, it was Object-Orientation that won the day.

If we can understand what makes OO so attractive in Computing today, then we might be able to apply the lessons we learn, to improve the suitability of Computational Logic, not only for Computing, but for Human Reasoning as well.

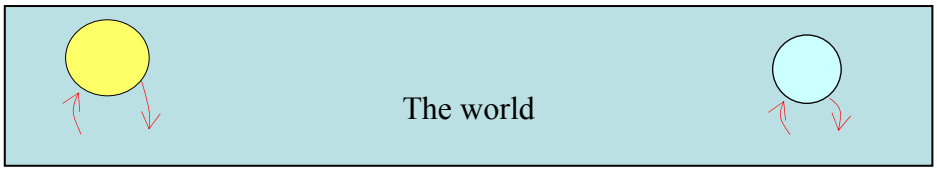
Objects as individuals in the changing world

In the object-oriented way of looking at things, the world consists of encapsulated objects, which interact with one another through their externally manifest input-output behaviour. There is no difference in this respect between an intelligent agent and an inanimate object. An agent is just another object embedded in the world.

Object-Orientation turns our earlier picture of the relationship between an agent and the world:



outside in:



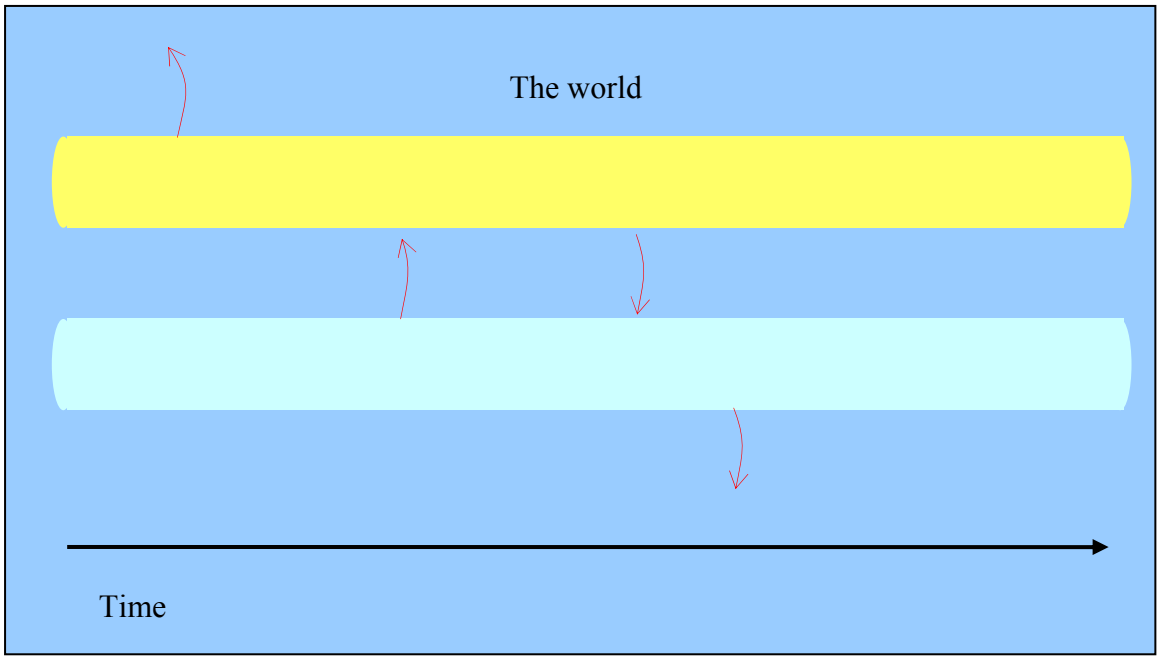
In this picture, objects correspond to individuals in the logic-oriented way of looking at the world. They are semantic rather than linguistic entities. However, whereas in traditional logic static relationships among individuals are the primary concern and dynamic behaviour is only secondary, in object-orientation it is the other way around - behaviour is primary and static relationships are secondary and relatively unimportant.

With logic embedded in the thinking component of an intelligent agent, change is dealt with at both the semantic and linguistic levels. At the semantic level it is dealt with by the dynamics of the agent's observation-thought-action cycle. At the linguistic level it is represented in the agent's mental language of goals and beliefs, using for example modal logic, the situation calculus or the event calculus.

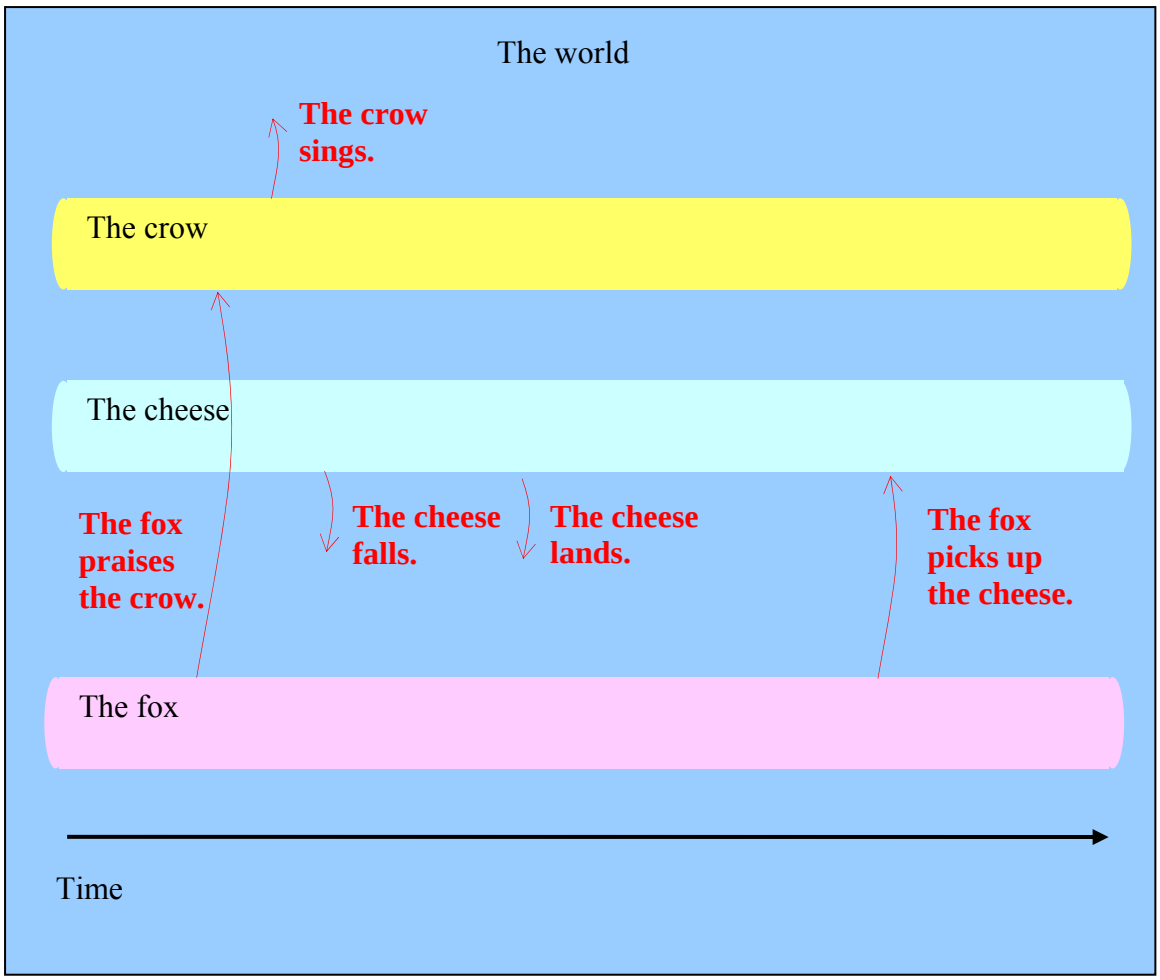
With objects, change is dealt with primarily at the semantic level. Objects, both animate and inanimate, interact with one another by sending and receiving messages. Receiving a message corresponds to an agent's observing the current state of the world. Sending a message corresponds to an agent's performing an action on the world.

With objects, changes can occur simultaneously in different parts of the world, independently and concurrently, without affecting other parts of the world. The event calculus has a similar view of change. But in the event calculus this view is linguistic - talking about change, without actually doing it. In object-oriented systems, these changes actually take place.

Taking account of the dynamic nature of objects, our object-oriented picture of the world should look more like this:



For example, in the object-oriented version of the story of the fox, crow and cheese, the various actions and events are messages between objects. The obvious candidates for sending and receiving these messages are the fox, crow and the cheese:



Advocates of object-orientation regard it as a natural way both to view the natural world and to construct artificial worlds. Systems engineers, in particular, construct complex systems by combining objects, to achieve some overall goal. Typically, the component objects, which make up the system, are themselves sub-systems, like thermostats, printers, computers, cars and airplanes, which are composed in turn of other, simpler objects.

Encapsulation

An object consists of a ***local state***, which is a collection of current ***values*** of ***attributes***, and a collection of ***methods***, which the object uses to respond to messages or to compute values of attributes. Both of these are ***encapsulated*** within the object, hidden from other objects.

Encapsulation of an object's methods is an inherent property of the natural world, because no object can tell for sure what goes on inside another object. In theory, if you could get inside another object, you might discover that it is just like you. Every object - bear, tree, river, mountain or stone - might have a spirit, which is its internal

mental state. Contrariwise, you might discover that no other object, other than yourself, has any kind of internal state whatsoever.

Encapsulation of methods serves a different function for artificial worlds. It reduces the complexity of constructing complex systems from component objects, because the engineer needs to take into account only the external behaviour of the components. Furthermore, should one of the components of a functioning system become defective or obsolete, it can be replaced by a new component that has the same external behaviour, without affecting the overall behaviour of the system.

Object-orientation does not distinguish between an object's external and internal state. All values of the attributes of an object are encapsulated inside the object. The only way to find the value of an attribute is by sending a message to the object asking for the value and by receiving a message from the object telling what the value is. I will argue later that this is too extreme.

However, in another respect, object-orientation is less extreme than extreme behaviourism. When an engineer constructs an object-oriented system, the engineer needs to get inside the system, both to combine pre-existing component objects and to create new objects from scratch.

Methods

To create an object, the engineer needs to initialise the values of its attributes and implement its methods. The values of attributes in this initial state can be given explicitly at the time of creation or can be derived by means of methods.

The common OO languages used for implementing methods are typically procedural languages with a syntax inherited from pre-OO programming languages. Unlike logic, these languages do not have a declarative semantics, because they make no attempt to represent the world around them.

However, even when methods are implemented in procedural programming languages, it is natural to express their specifications in logical form. These specifications have the form of condition-action rules in declarative mood:

> If object receives message of form R
> then object sends message of form S.

For example:

> If the fox receives a message that the crow has cheese in its mouth,
> then the fox sends a message of praise to the crow.
>
> If the crow receives a message of praise from the fox,
> then the crow sends a message of song.
>
> If the cheese receives a message of song from the crow
> then the cheese sends a message of falling to the ground.

If the cheese sends a message that it is next to the fox,
then the fox sends a message that she picks up the cheese.

The methods by means of which these specifications are implemented inside an object can be programmed in different ways. They can also be implemented, as we will discuss later and as should be obvious to the reader already, by means of goals and beliefs expressed in logical form.

Classes

Object-orientation prides itself on making it easy for the engineer to create new objects by instantiating more general classes of objects, inheriting methods associated with those classes.

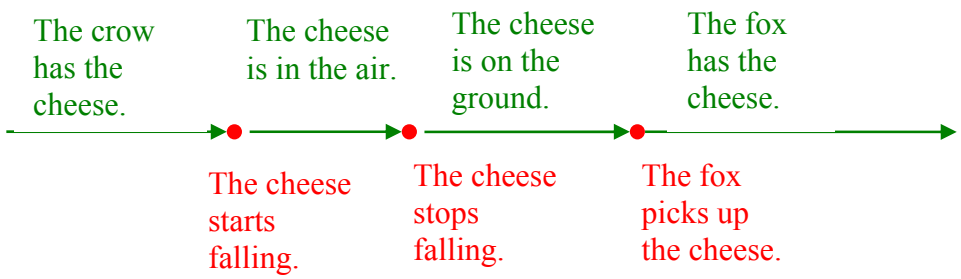
For example, an engineer might create a new fox by creating an instance of the general class of all foxes. The class of foxes as a whole might have general methods for dealing with such messages as the sight of another animal having food and the appearance of food within its grasp. It might also have typical values for such attributes as the colour of its fur and the shape if its tail. The new fox could then inherit these methods and values of attributes with little or no modification, possibly with the addition of certain special methods and attributes unique to itself.

Classes are organised in taxonomic hierarchies. So for example, the class of all foxes might inherit most of its methods and attributes from the class of all animals. The class of all animals might inherit them, in turn, from the class of all animate beings; the class of all animate beings might inherit them from the class of all material objects; and the class of all material objects might inherit them from the class of all things.

Change of state

Objects and classes have different kinds of existence. Objects are concrete individuals, which typically exist in time and undergo change of state. Classes are abstract individuals, which are timeless.

For example, in the story of the fox and the crow, the cheese changes state in response to local events that take place around it:



These changes of state are generated by methods inherited from the general class of all cheeses, some of which, like the propensity to be in the air after starting to fall, are inherited in turn from the class of all material things. However, the classes themselves do not change state in the same way.

OO systems are semantic rather than linguistic entities. As a consequence, objects need not have any memory of the past. When an object changes state, the change generally wipes out the previous state and replaces it with the new one.

In contrast, logical representations of change describe changes of state without actually performing them. Therefore, in addition to recording the current state of an object, they can record previous states, as well as predict and explore future ones.

Reconciling logic and objects

There is an obvious way to reconcile logic and objects: simply by using logic to implement the methods associated with objects and classes of objects. A typical implementation of this logical kind might consist of maintenance goals and beliefs, which eventually reduce goals to action sub-goals, including messages to other objects. For example:

Goal: If I receive message of form R
then solve goal G.

Beliefs: G if conditions.
…
G' if other conditions and I send message of form S.
etc.

Creating a new object by instantiating a class then becomes a simple matter of logical instantiation, substituting a concrete term (such as " I ") for an abstract term (such as "any fox"), possibly adding specialised methods and assigning explicit values for some of the object's attributes.

Logic, used to implement methods in this way, can enhance the power of object-oriented systems. Some, intelligent objects can use logic not only to react to incoming messages and to reduce goals to sub-goals, but also to represent the world, generate alternative courses of action and derive their likely consequences. Other, less intelligent objects can also use logic as their higher-level language of implementation. For purposes of efficiency, these higher-level implementations can be compiled into lower-levels, including hardware. Conversely, lower level implementations can sometimes be decompiled into logical form, to improve and reason about the behaviour of individual objects and collections of objects.

Combining logic and object-orientation can also enhance our logic-based agent model. Most importantly, it provides a framework for extending single logic-based agents to multi-agent systems, by embedding agents in a shared semantic structure. Different objects in this structure can have different degrees of intelligence and different degrees of activity.

The OO semantic structure is one in which objects can change state concurrently in different parts of the world, independently and without affecting other objects. These changes can take place destructively, without leaving any record of the past. However, in addition to such real changes taking place externally in the semantic structure, an intelligent agent can use logic internally to represent changes, without actually performing them. These internal representations of change are not destructive in the same way that external changes destroy earlier states of the world.

To achieve these benefits, we do not need to abandon the distinction made in logic between world structures and their linguistic representations - nor the view that world structures consist of individuals, possibly including events, and of relationships among individuals. On the contrary, we may need to abandon the extreme version of OO, which insists that all attributes of individuals are encapsulated and that all interactions are messages.

Consider, for example, yet again the story of the fox and the crow, and in particular the fox's observation that the crow has the cheese. In the strict OO version of the story, the observation is a message sent to the fox by another object. But what object is it? The obvious candidates are the crow and the cheese. But clearly the crow object has no goal to achieve in sending such a message to the fox, and the cheese object has no real goals of any kind. Even if one of them were responsible for sending the message, why send it only to the fox? Why not send it to all the other creatures in the area as well?

In our earlier version of the story and in our logic-based agent model more generally, the fox and the crow, but not the cheese, are both agents, and they interact with one another, not by sending messages, but by observing the world and by performing actions on the world. The actions they perform might be observed by other agents in the area.

It seems to me that object-orientation has little new to contribute to our understanding of the story, except to remind us that

- it is important to distinguish between the world and any representations that agents may have of the world;
- that it is useful to understand the world in terms of hierarchies; and
- that the internal methods agents use to interact with the world are encapsulated, so that they can not be observed directly by other agents.

OO has another virtue, namely that it shows us how to construct artificial worlds in which separate objects can change state concurrently and independently. But this is not relevant in the story of the fox and the crow, where we are only interested in telling a story, and not in constructing a world.

Object-orientation in moderation

However, comparing logic with OO, it seems that OO goes too far - encapsulating all relationships among individuals inside objects. Thus, the fact

that the crow has the cheese at the beginning of the story is encapsulated inside one or both of the crow and the cheese objects, accessible to other objects only by sending and receiving messages.

The alternative is to embrace a less extreme form of object-orientation, in which only those individuals that actively change state are regarded as objects, and only the methods they use to interact with the world are encapsulated. Relationships among objects and other kinds of individuals are not encapsulated, but are manifest in the external world.

Objects, in this more moderate form of OO, interact with the world, not only by sending and receiving messages, but more generally by observing relationships in the world and by performing actions that change relationships in the world. Messages can be viewed as speech acts, in which one object performs a special kind of action whose effects are intended for observation by another object.

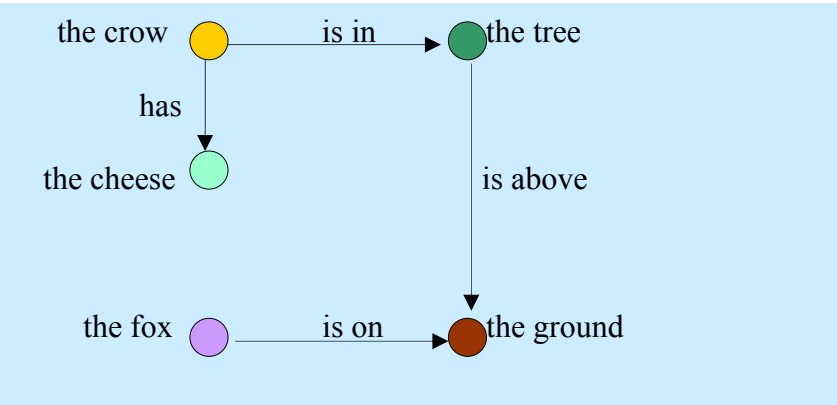
Other individuals, like classes, events, numbers and time points, can partake in relationships without being coerced into encapsulated objects.

This moderate form of object-orientation is compatible with our logic-based agent model. It liberalises our notion of agent, to include more primitive kinds of encapsulated object. It enriches our notion of the agent's environment, to include, not only simple individuals and relationships, but individuals that are objects, which can change state concurrently using encapsulated methods, which are hidden from the external world.

Semantic networks as a variant of object-orientation

There are several other Computing paradigms that subscribe to similarly moderate variants of object-orientation. These include semantic networks and frames, developed as knowledge representation formalisms in Artificial Intelligence, and the entity-relationship model of databases.

Semantic networks represent the world as a web of relationships among individuals. For example, the network representing the initial state of the story of the fox and the crow might like this:

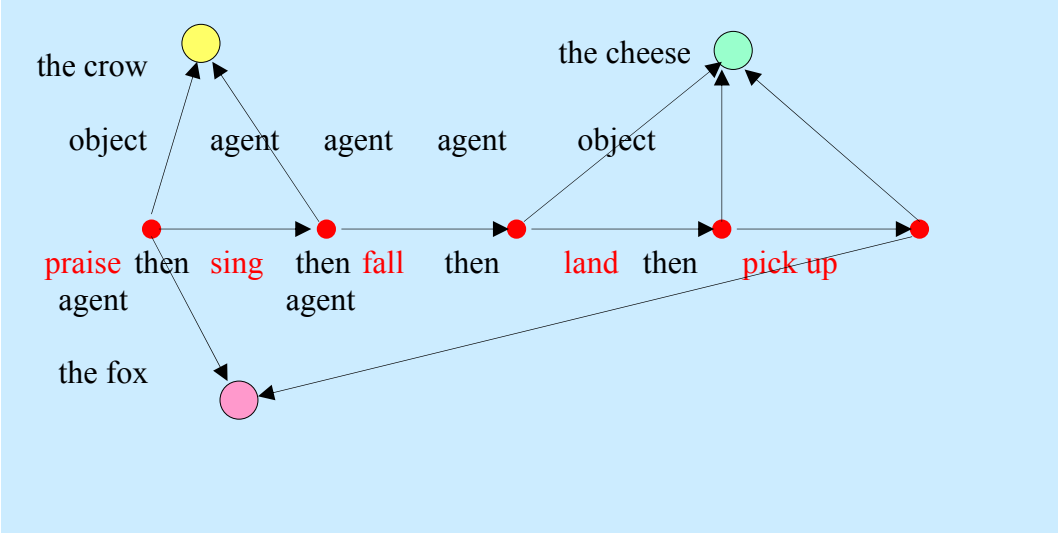


Here nodes represent individuals, and arcs represent binary relationships, between pairs of individuals. The representation can be extended to non-binary relationships.

Semantic network representations are object-oriented, in the sense that all the facts about an individual are located in a single place, namely surrounding the node that represents the individual. These facts are represented by the arcs connected to that node and by the other nodes to which those arcs are also connected.

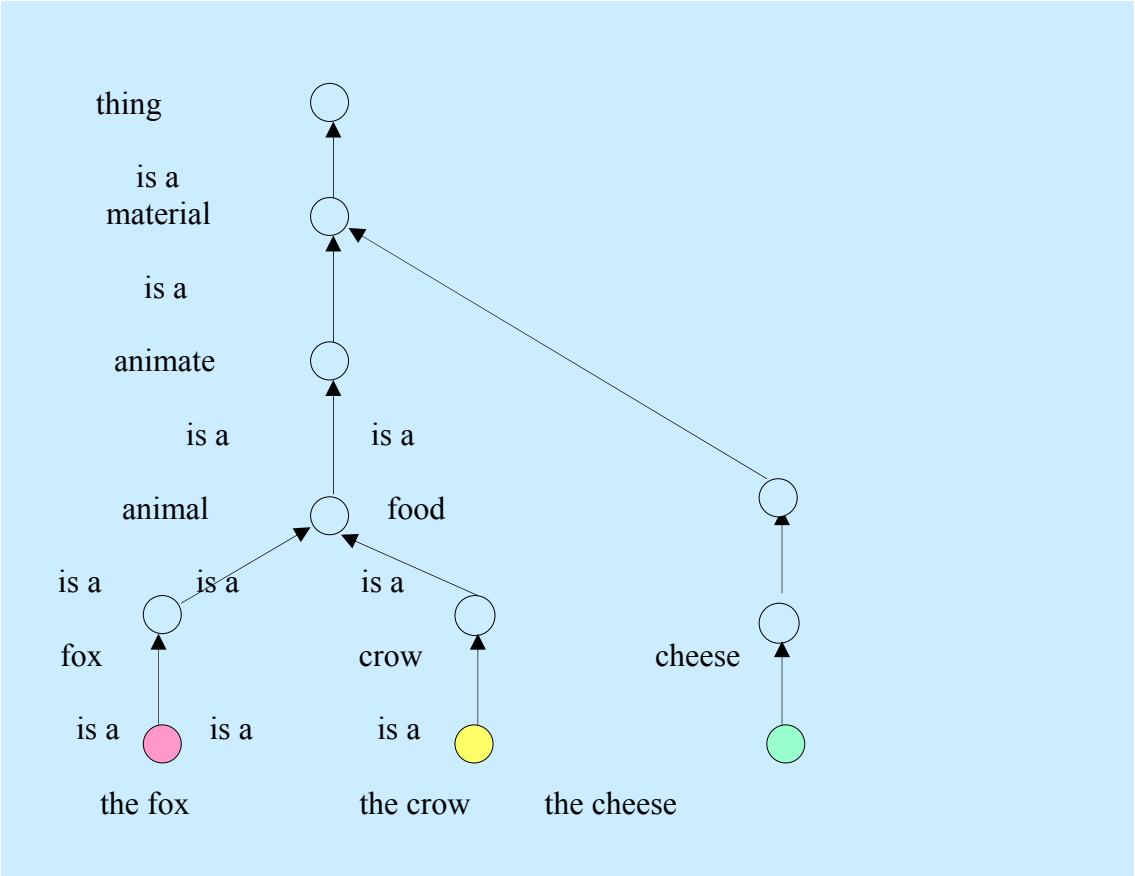
However, in contrast with orthodox object-orientation, relationships are represented only once, but are connected to all the individuals that participate in the relationship. Moreover, they are visible to the outside world, and not encapsulated inside objects.

Semantic networks have also been used to represent dynamic information, by reifying events. For example:



Here the semantic network terms “object” and “agent” are only loosely associated with our notions of object and agent. Here the term “agent” is analogous to the subject of an English sentence, and the term “object” is analogous to the object of an English sentence.

Semantic networks have also been used to represent hierarchies of classes. For example:



Despite their name, semantic networks are not semantic structures in the same sense as OO structures, but they are like semantic structures as represented by sets of atomic sentences in logic. In fact, semantic network connections of the form:

one thing is related to another thing

are simply graphical representations of atomic sentences. Like sentences of logic, when they are used to represent dynamic information by reifying events, they represent change without performing it.

On the other hand, semantic networks are like semantic structures in the sense that they represent only atomic facts, rather than more general regularities. Although several extensions of semantic networks have been developed to represent regularities, these extensions are not convincing.[42]

Nonetheless, semantic networks show that it is possible to use objects to structure atomic representations of the world and not only to structure the world itself. They also show that it is possible to subscribe to moderate forms of OO, in which objects structure our understanding of the world without hiding everything inside themselves. Frames and entity-relationship databases are similar in this respect.

Object-oriented structuring of sentences

Semantic networks are one way objects can be used to structure the representation of the world. Objects can also be used to structure more general linguistic representations.

We noted, already in Chapter 1, that sentences in logic can be written in any order. However, some sequences of sentences can be much easier to understand than other sequences. Grouping sentences into sets of sentences about objects is one way to make sentences easier to understand.

For example, we can group the atomic sentences describing the beginning of the story of the fox and the crow into the sets of sentences:

The crow:	The crow has the cheese. The crow is in the tree.
The tree:	The tree is above the ground.

[42] I have contributed to one such extension myself, in Deliyanni, A. and Kowalski, R., "Logic and Semantic Networks", in CACM, Vol. 22, No. 3, 1979, pp. 184-192.

The fox: The fox is on the ground.

Of course, we can also group the same sentences by means of other objects:

The cheese: The crow has the cheese.

The tree: The crow is in the tree.

The ground: The tree is above the ground.
The fox is on the ground.

To find a good organisation, it is necessary to decide which objects are the most important.

Natural languages, like English, take object-orientation a step further, by employing grammatical forms in which the beginning of a sentence indicates its ***topic*** and the following part of the sentence expresses a ***comment*** about the topic. This form often coincides with, but is not limited to, the grammatical structuring of sentences into ***subjects*** and ***predicates***.

The two forms of object-orientation – grouping sets of sentences by object and structuring individual sentences by object – are often combined in practice. Consider, for example, the pair of English sentences[43]:

> The prime minister stepped off the plane.
> Journalists immediately surrounded her.

Both sentences are formulated in the active voice, which conforms to the guidelines for good practice advocated in most manuals of English style.

The two sentences refer to three objects, the prime minister (referred to as "her" in the second sentence), journalists and the plane. The prime minister is the only object in common between the two sentences. So, the prime minister is the object that groups the two sentences together. However, the topic changes from the prime minister in the first sentence to the journalists in the second.

Now consider the following logically equivalent pair of sentences:

> The prime minister stepped off the plane.
> She was immediately surrounded by journalists.

Here, not only do the two sentences refer to a common object, but they also have the same topic. However, the second sentence is now expressed in the passive voice. Despite this fact and despite its going against a naïve interpretation of the guidelines of good writing style, most people find this second pair sentences easier to understand. This seems to suggest that people have a strong preference for organising

[43] This example is from "Discourse Analysis" by Gillian Brown and George Yule, Cambridge University Press, 1983, page 130.

their thoughts in object-oriented form, which is stronger than their preference for the active over the passive voice.

Object-orientation is not the only way of structuring and ordering sentences. In both of the two pairs of sentences above, the sentences are ordered by the temporal sequence of events.

Now consider the following sequence of sentences:

> The fox praised the crow.
> The crow sang a song.
> The cheese fell to the ground.
> The fox picked up the cheese.

Here the sentences are grouped by temporal proximity and ordered by temporal sequence. Individual sentences are structured, not by object, but by agent, as reflected in the use of the active voice.

It's important to keep things in perspective. Object-orientation isn't everything. In the case of English, it has nothing to do with the content of individual sentences, but it is one way to group and structure sentences. Compared with logic, OO is like the index of a book, and logic is like the language in which the book is written.

Conclusions

Compared with logic, the main attraction of OO is that it shows us how to construct artificial worlds in which individual objects can change state concurrently and independently from other objects. It facilitates constructing new objects from existing ones by encapsulating methods, to reduce complexity and to improve maintainability.

However, compared both with logic and with more moderate variants of OO, such as semantic networks, extreme OO takes encapsulation too far. Instead of hiding attributes and relationships inside objects, it would be more natural to make them visible to the outside world. Instead of treating all individuals as encapsulated objects, objects should be restricted to individuals that interact with the world.

Extreme OO also takes the message passing metaphor too far. Instead of forcing all interactions between objects and the world to be messages, it would be more natural for agents and other objects to interact with the world by making observations and performing actions.

Computational logic can be reconciled with both extreme and moderate forms of OO, by using it as a higher-level language for implementing methods and for defining classification hierarchies. Although we have not discussed the possibility in this chapter, it can also be used by a system designer to show that a collection of objects achieves some over-all goal.

Semantic networks and natural languages such as English show that OO is not restricted to modelling semantic structures, but can also be used to structure representations of the world. The example of English, in particular, shows that, in the

area of linguistic representations, logic and OO have different concerns. Logic is concerned with representing the world, whereas OO is only concerned with one way of structuring representations. As we have seen, OO needs to be combined with other ways of structuring representations, in order to be useful. It would be interesting, for example, to see how OO might be used to structure the collection of sentences that make up this chapter.

Appendix 1

The Logical Way to Be Artificially Intelligent

Robert Kowalski
Imperial College London
rak@doc.ic.ac.uk
http://www.doc.ic.ac.uk/~rak/

Abstract. Abductive logic programming (ALP) can be used to model reactive, proactive and pre-active thinking in intelligent agents. Reactive thinking assimilates observations of changes in the environment, whereas proactive thinking reduces goals to sub-goals and ultimately to candidate actions. Pre-active thinking generates logical consequences of candidate actions, to help in deciding between the alternatives. These different ways of thinking are compatible with any way of deciding between alternatives, including the use of both decision theory and heuristics.

The different forms of thinking can be performed as they are needed, or they can be performed in advance, transforming high-level goals and beliefs into lower-level condition-action rule form, which can be implemented in neural networks. Moreover, the higher-level and lower-level representations can operate in tandem, as they do in dual-process models of thinking. In dual process models, intuitive processes form judgements rapidly, sub-consciously and in parallel, while deliberative processes form and monitor judgements slowly, consciously and serially.

ALP used in this way can not only provide a framework for constructing artificial agents, but can also be used as a cognitive model of human agents. As a cognitive model, it combines both a descriptive model of how humans actually think with a normative model of humans can think more effectively.

1 Introduction

Symbolic logic is one of the main techniques used in Artificial Intelligence, to develop computer programs that display human intelligence. However, attempts to use symbolic logic for this purpose have identified a number of shortcomings of traditional logic and have necessitated the development of various improvements and extensions. This paper - and the draft book [6] on which it is based - aims to show that many of these developments can also be used for the original purpose of logic – to improve the quality of human thinking.

I have written the book informally, both to reach a wider audience and to demonstrate that the enhanced logic is in fact relevant and congenial for human thinking. However, in this paper, I will draw attention to some of the more technical issues, for the consideration of a more academic audience.

The logic used in the book is based on an extension of abductive logic programming (ALP) to logic-based agents [7]. In ALP agents, *beliefs* are represented by logic programs and *goals* are represented by integrity constraints. The agent's *observations* and *actions* are represented by abducible predicates. Beliefs and goals have both a declarative interpretation in logical terms, as well as a procedural interpretation in computational terms.

ALP agents are both *reactive* to changes they observe in the environment and *proactive* in planning ahead and reducing goals to sub-goals. In this paper I show that ALP agents can also be *pre-active* in thinking about the possible consequences of actions before deciding what to do.

In conventional ALP, the logical consequences of abductive hypotheses are checked to determine whether they violate any integrity constraints. However, in ALP agents, where abductive hypotheses include alternative, candidate actions, the pre-actively generated consequences of candidate actions are used to decide between the alternatives. This decision can be made in different ways. One way is to use conventional Decision Theory, judging the utilities and probabilities of the consequences of the alternative candidates and choosing an action that maximizes

expected utility. However, other ways of deciding between actions are also compatible with ALP, including ways that compile decision making into heuristics.

The combination of reactive, proactive and pre-active thinking is obtained in ALP agents by combining forward and backward reasoning. This reasoning can be performed whenever the need arises, or it can be performed once and for all by reasoning in advance. Reasoning in advance transforms and compiles higher-level goals and beliefs into lower-level goals, which are similar to condition-action rules, which implement stimulus-response associations compiled into neural networks.

In modern computing, it is common to develop programs in a high-level representation and then to transform or compile them into a lower-level representation for the sake of efficiency. If it later becomes necessary to correct or enhance the resulting lower-level program, this is generally done by first modifying the higher-level representation and then recompiling it into a new lower-level form.

However, many existing computer systems are legacy systems developed before the existence of higher-level programming languages. It is often possible to decompile these lower-level programs into higher-level form, although, because of the undisciplined nature of lower-level languages, sometimes the relationship is only approximate.

The relationship between higher-level and lower-level computer programs is analogous to the relationship between higher-level and lower-level representations in ALP agents. It is also similar to the relationship between deliberative and intuitive thinking in the *dual process model* of human thinking [10]. In the dual process model, one system, which is older in evolutionary terms, is responsible for *intuitive thinking*. It is associative, automatic, unconscious, parallel, and fast. The other system, which is distinctively human, is responsible for *deliberative thinking*. It is rule-based, controlled, conscious, serial, and slow.

In computing, high-level and low level representations normally operate separately, but can be compiled or decompiled from one into the other. In the dual process model, however, intuitive and deliberative thinking can operate in tandem, as when the intuitive, subconscious level "quickly proposes intuitive answers to judgement problems as they arise", while the deliberative, conscious level "monitors the quality of these proposals, which it may endorse, correct, or override" [3]. This interaction between intuitive and deliberative thinking can be mimicked in part by the use of pre-active thinking in ALP agents, to monitor and evaluate candidate actions generated by reactive thinking. In ALP agents both the deliberative level and the intuitive level are represented in logical form.

These topics are expanded upon in the remainder of the paper. Section 2 outlines the basic features of the ALP agent model, including reactive, proactive, and pre-active thinking. Section 3 investigates the relationship between thinking and deciding. Section 4 discusses the transformation of high-level representations into lower-level, more efficient form, and the way in which high-level and lower-level representations interact. Section 5 shows how low-level feed-forward neural networks can be represented in logical form and can be simulated by forward reasoning. Section 6 discusses some of the implications of this for the notion that logic can serve as a wide-spectrum language of thought. Section 7 addresses some of the arguments against logic as a model of human thinking, and section 8 is the conclusion.

2 The Basic ALP Agent Model

2.1 Putting Logic in its Place in the Agent Cycle

The logic used in the book is based on an extension of abductive logic programming (ALP) to logic-based agents [7]. The most important feature of the extension is that it embodies logic in the thinking component of an agent's observe-think-decide-act cycle:

> To cycle,
> observe the world,
> *think*,
> decide what actions to perform,
> act,
> cycle again.

The agent cycle can be viewed as a generalisation of production systems, in which thinking is performed by using condition-action rules of the form:

If conditions then candidate actions.

Condition-action rules look a lot like logical implications, but they do not have the declarative semantics of implications. Nonetheless, as we will later see, in ALP agents, condition-action rules are represented by goals expressed in logical form.

This view of logic in the mind of an agent embodied in the world is pictured in figure 1. In this picture, the agent uses logic to represent its goals and beliefs, and to help control its interactions with the world. It transforms its experience into observations in logical form and uses its goals and beliefs to generate candidate actions, to satisfy its goals and to maintain itself in a satisfactory relationship with the changing world.

The agent's body, in addition to being a part of the world, transforms both raw experience into observations and the will to act into physical changes in the world. This is analogous to the way in which hardware and software are related in a computer. The hardware transforms stimuli from the environment into inputs and transforms outputs into physical changes in the environment. The internal processing of inputs into outputs is performed by the hardware, but is controlled conceptually by software. In this analogy, the brain and body of an agent are to the mind as hardware is to software.

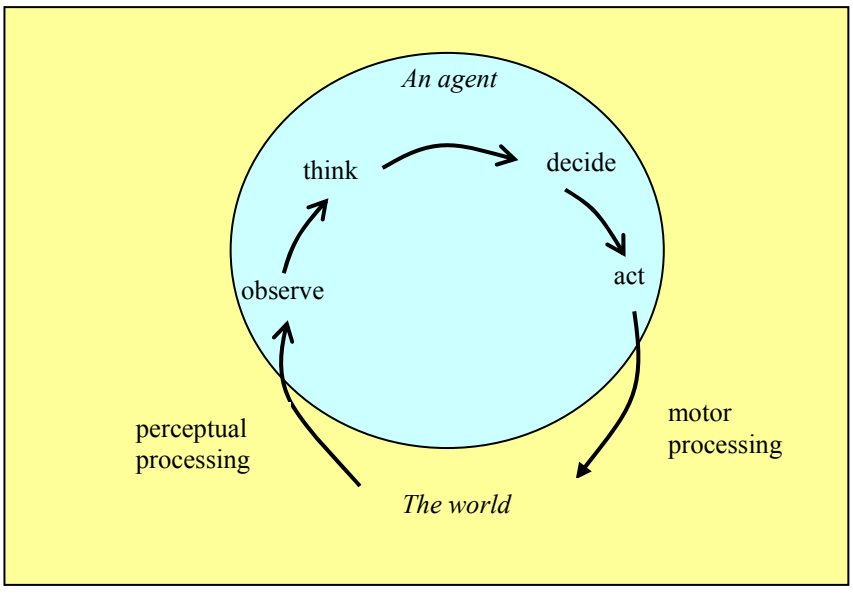


Fig. 1 The agent cycle.

In general, the result of thinking is a set of candidate actions, which are the input to the decision-making component of the agent cycle. In the same way that Logic is only one way of thinking, there are many ways of deciding what to do. Decision theory, which combines judgements about the utility of the outcomes of actions with judgements about their probability, is one such way of deciding. As we will see in an example later, it is also possible to compile decision-making directly into lower-level goals and beliefs. In production systems, decision making is called "conflict resolution".

An agent's ultimate goal is to maintain itself in a satisfactory relationship with the surrounding world, and thinking and deciding are only one way of achieving that goal. An agent can also act to satisfy its goals instinctively, by means of *stimulus-response associations*, in a way that might be characterised as acting without thinking. Instinctive behaviour can be hardwired into an agent's body, without entering into its mind. Or it might be learned as the result of repeated performance and feedback. Instinctive behaviour is a near relation of intuitive thinking in the dual process model.

The agent cycle, as described above, concerns the real time behaviour of an agent, and does not address the processes involved in learning new behaviours and updating old ones. Suffice it to say that learning, belief revision and goal revision are essential activities for a real agent interacting with the real world. Because of the logical nature of ALP agents, such techniques as inductive logic programming are especially suitable to model such activities. They are, however, beyond the scope of this paper.

2.2 ALP Combines Forward and Backward Reasoning

Abductive logic programming [4] comes in many forms and variations, both in terms of its semantics and in terms of its proof procedures. However, in all of these forms, abductive logic programs have two components: ordinary logic programs and integrity constraints. They also have two, corresponding kinds of predicates – *ordinary predicates* that are defined by logic programs and *abducible predicates* that are, directly or indirectly, constrained by integrity constraints.

In ALP agents, logic programs are used to represent an agent's *beliefs*, and integrity constraints to represent its *goals*. The abducible predicates are used to represent the agent's *observations* and *actions*. The integrity constraints are *active*, in the sense that they can generate representations of actions that the agent can perform, in order to maintain integrity.

Consider, for example, the goal of getting help in an emergency on the London underground.

Goal If there is an emergency then I get help.

Beliefs There is an emergency if there is a fire.
There is an emergency if one person attacks another.
There is an emergency if someone becomes seriously ill.
There is an emergency if there is an accident.

There is a fire if there are flames.[44]
There is a fire if there is smoke.

A person gets help
if the person alerts the driver.

A person alerts the driver
if the person presses the alarm signal button.

Here, for simplicity, the abducible predicates associated with observations are the predicates "there are flames", "there is smoke", "one person attacks another", "someone becomes seriously ill", and "there is an accident". The only abducible predicate associated with candidate actions is "the person presses the alarm signal button". All of these abducible predicates are indirectly constrained by the goal of getting help whenever there is an emergency. All the other predicates, including the

[44] These two rules, relating fire, flames and smoke are the converse of the causal rules, which state that if there is a fire then there are flames and smoke. The causal rules are a higher-level representation, whereas the rules used here are a lower-level, more efficient representation. The higher-level, causal representation would need abduction to explain that an observation of smoke or flames can be caused by fire. In fact, the term "abduction" normally refers to such generation of hypotheses to explain observations. The lower-level representation used here replaces abduction by deduction.

higher-level actions of getting help and alerting the driver are ordinary predicates, defined by the agent's beliefs.

The goal itself is a *maintenance goal*, which an agent can use to derive actions to maintain itself in a desired relationship with the changes that it observes in its environment. Maintenance goals can be viewed as a generalization of condition-action rules.

Maintenance goals are triggered as a consequence of observations, similarly to the way in which integrity constraints in a database are triggered as the result of updates. An agent reasons forwards from its beliefs, to derive consequences of its observations. Suppose, for example, that I am travelling as a passenger on the underground and that my body experiences a combination of sensations that my mind interprets as an observation of smoke. The observation triggers my beliefs, which I use to reason forward in two steps, to recognize that there is an emergency.

The conclusion that there is an emergency triggers the maintenance goal, which I then use to reason forward one more step, to derive the achievement goal of getting help. The achievement goal triggers other beliefs, which I use to reason backwards in two steps, to reduce the achievement goal to the action sub-goal of pressing the alarm signal button. Since there are no other candidate actions in this simple example, I decide to press the alarm signal button, which my body then transforms into a combination of motor activities that is intended to accomplish the desired action.

The fact that pure logic programs are declarative means that they can be used to reason in many different ways. In the procedural interpretation, they are used only to reason backwards, as procedures that reduce goals to sub-goals. However, in ALP they are used to reason both backwards and forwards.

This combination of forward and backward reasoning, together with the interface between the agent's mind and the world, is pictured in figure 2. Arguably, this treatment of maintenance goals as integrity constraints generalizes condition-action rules in production systems. Condition-action rules are the special case where no forward reasoning is needed to trigger the maintenance goal and no backward reasoning is needed to reduce the achievement goal to actions. Thus maintenance goals include condition-action rules as a special case, but in general are much higher-level.

Vickers [12], in particular, championed the idea that human activity and organizations should be viewed as maintaining relationships with the changing environment. He characterized Simon's view of management and problem solving as the narrower view of only solving achievement goals. Vickers view-point has been taken up by in recent years by the soft systems school of management [2].

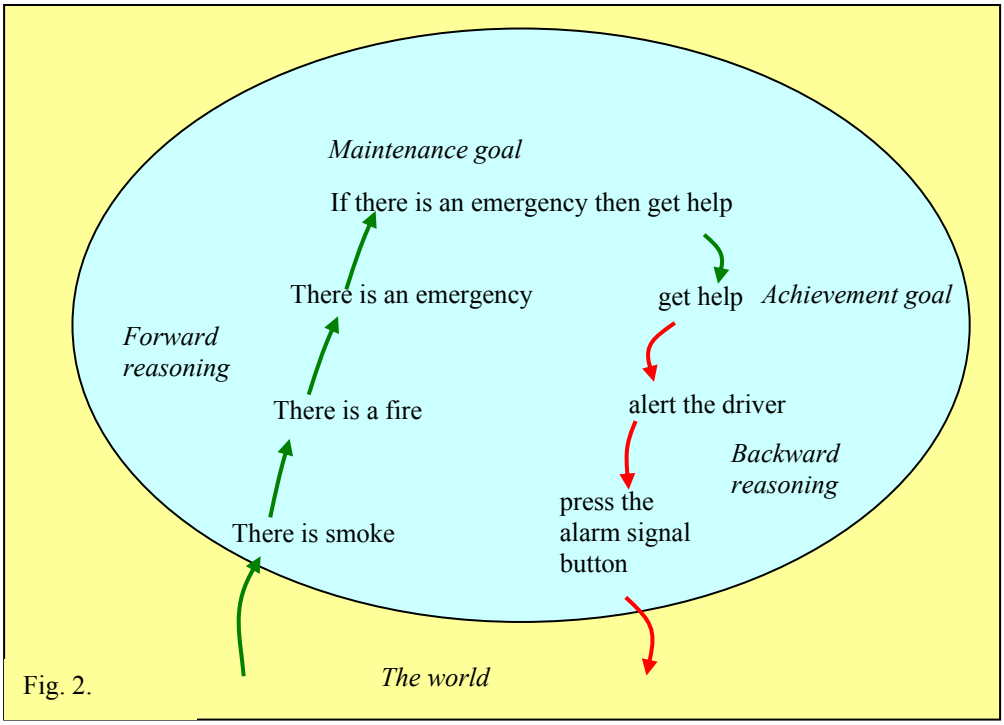


Fig. 2.

2.3 ALP Combines Reactive and Proactive Thinking

The combination of forward and backward reasoning enables ALP agents to be both reactive and proactive. They are reactive when they use forward reasoning to respond to changes in the environment, and they are proactive when they use backward reasoning to achieve goals by reducing them to sub-goals. In everyday life, human agents are both reactive and proactive to varying degrees.

Consider, as another example, a simplified ALP version of Aesop's fable of the fox and the crow. Suppose the fox has the following achievement goal and beliefs:

Goal I have the cheese.

Beliefs The crow has the cheese.

An animal has an object
if the animal is near the object
and the animal picks up the object.

I am near the cheese
if the crow has the cheese
and the crow sings.

The crow sings if I praise the crow.

The fox can use its beliefs as a logic program, to reason backwards, to reduce its goal to the actions of praising the crow and picking up the cheese.[45] The fox's reduction of its goal to sub-goals is pictured in figure 3.

In keeping with the view that the primary goals of an agent are all maintenance goals, the fox's achievement goal almost certainly derives from a maintenance goal, such as this:

[45] The story is simplified partly because the element of time has been ignored. Obviously, the fox needs to praise the crow before picking up the cheese.

If I become hungry, then I have food and I eat it.

Here the condition of being hungry is triggered by an observation of being hungry, which the fox receives from its body. Notice that the achievement goal of having the food is only half of the story. To satisfy the maintenance goal, the fox also needs to eat the food.

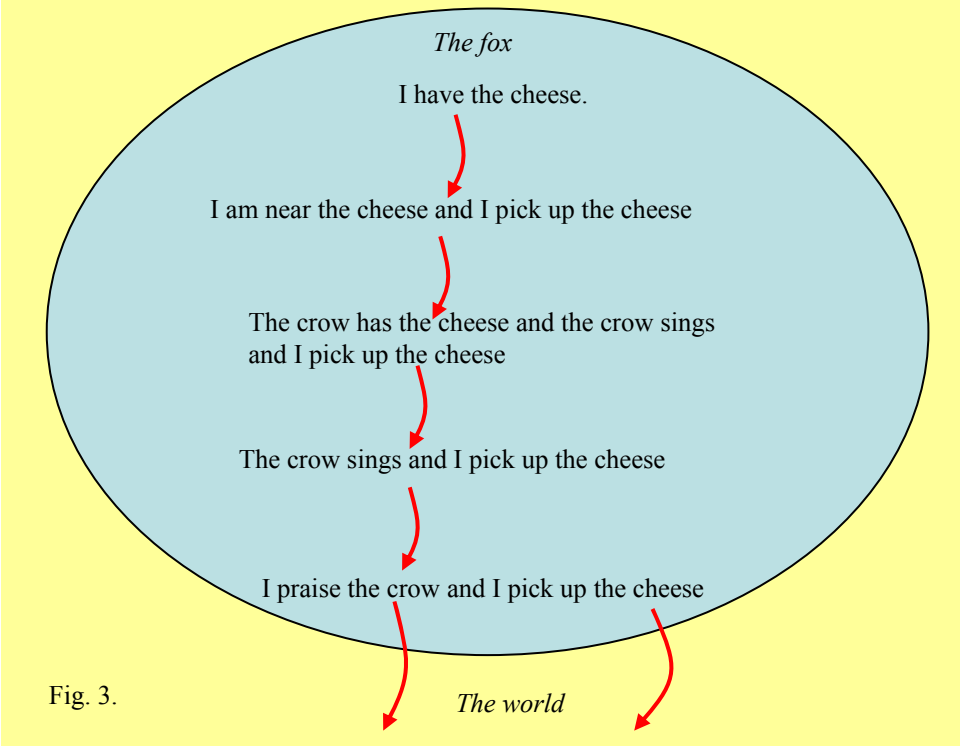


Fig. 3.

In Aesop's fable, the fox's belief about the behaviour of the crow is true. The crow is a purely reactive agent, which responds to praise as the fox believes. The reactivity of the crow can be viewed as reasoning forwards in one step from an observation to derive an achievement goal, which is an action, from a maintenance goal. This is pictured in figure 4.

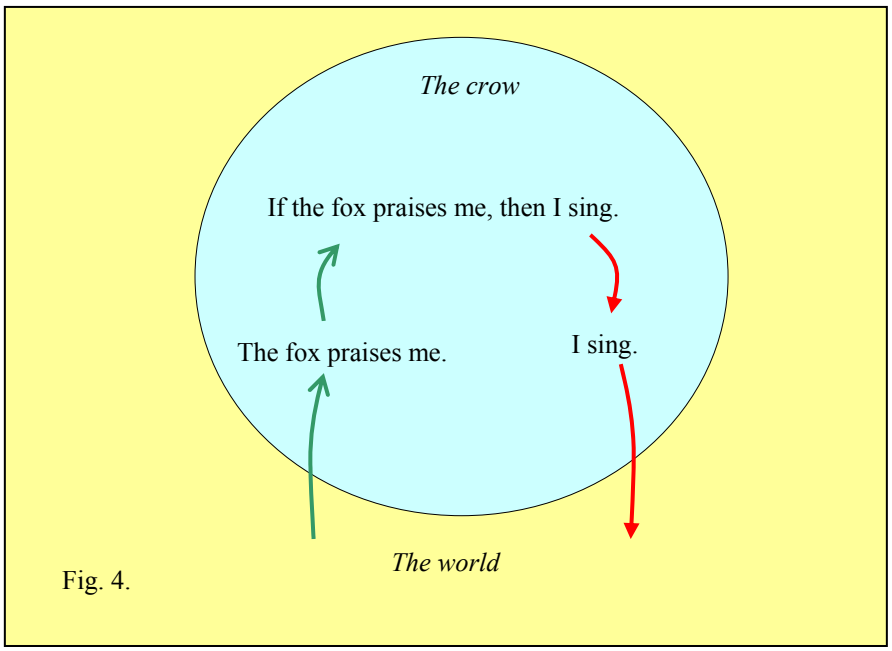


Fig. 4.

This view of the crow's behaviour is a logical abstraction of behaviour that might be hardwired into the crow as a system of lower-level stimulus-response associations. The relationship between such a logical abstraction and the stimulus-response associations is, arguably, like the relationship between software and hardware.

Notice the difference between the sentence

If the fox praises me, then I sing.

which is a goal for the crow, and the sentence

The crow sings if I praise the crow.

which is a belief for the fox. Both sentences are implications. However, for the crow, the implication is used as a goal, to generate its behaviour. But for the fox, the implication is used as a belief, to describe the crow's behaviour and to reduce goals to sub-goals.

The difference between the two sentences has nothing to do with the order in which the conclusion and condition of the implication is written, because there is no semantic difference between writing an implication forwards in the form

If conditions then conclusion.

and writing it backwards in the form

Conclusion if conditions.

Semantically both implications have the same declarative meaning. (In the same way that both inequalities $1 < 2$ and $2 > 1$ have the same meaning.)

However, no matter how implications are written, there is an important distinction between them depending upon whether they are used as goals or as beliefs. When they are used as beliefs, they represent the world as it actually is. When they are used as goals, they represent the world as the agent would like it to be. When a goal is an implication, the agent performs actions to make the implication true. It only needs to perform these actions to make the conclusion of the implication true when the world makes the conditions of the implication true. It need not worry about performing actions when the world makes the conditions of the implication false. The analogous distinction in deductive databases between implications used as integrity constraints and implications used as rules was first investigated by Nicolas and Gallaire [ ].

2.4 ALP Includes Pre-active Thinking

Aesop's fable shows how a proactive fox outwits a reactive crow. But there is an even more important moral to the story - namely that an intelligent agent should think before it acts. Thinking before acting is more than just proactive thinking. It is thinking about the possible consequences of candidate actions - *pre-actively* – before deciding what to do. Pre-active thinking is obtained in ALP by reasoning forward from candidate actions, whether derived proactively or reactively, and whether generated by symbolic reasoning or by instinctive stimulus-response associations.

Suppose, for example, that the crow not only has the maintenance goal of singing whenever it is praised, but also has the achievement goal (for whatever reason) of having the cheese. If the crow also has the same beliefs as the fox, then the crow would be able to reason forward, pre-actively, to deduce the possible consequences of singing:

I want to sing.

But if I sing,
then the fox will be near the cheese.

Perhaps the fox will pick up the cheese.
Then the fox will have the cheese,
and I will not have the cheese.

Since I want to have the cheese,
I will not sing.

Notice that the crow can not consistently achieve the goal of having the cheese and also maintain the goal of singing whenever it is praised. In real life, an agent needs to weigh its goals, trading one goal off against another.[46]

Notice too that the outcome of an agent's actions typically depends also on external events, over which the agent may have little or no control. In the story of the fox and crow, the outcome of the crow's singing depends on whether or not the fox decides to pick up the cheese.

4 Thinking Needs to be Combined with Deciding What to Do

In ALP, pre-active thinking simply checks whether candidate actions satisfy the integrity constraints. However, in real life, we also have to choose between actions, taking into consideration the relative utilities and probabilities of their possible consequences. In Decision Theory, the agent uses these considerations to choose an action that has maximum expected utility.

4. 1 Combining ALP with Decision Theory

Suppose, for example, that I have the following beliefs:

I get wet if it rains and I do not carry an umbrella.
I stay dry if I carry an umbrella.
I stay dry if it doesn't rain.

Assume also that I am about to leave home, and that as a sub-goal of leaving home I have to decide what to take with me, and in particular whether or not to take an umbrella. I can control whether to take an umbrella, but I can not control whether it will rain. At best I can only judge the probability of rain.

Reasoning forward from the assumption that I take an umbrella and then have to carry it, I can derive the certain outcome that I will stay dry. However, reasoning forward from the assumption that I do not carry an umbrella, I derive the uncertain outcome that I will get wet or I will stay dry, depending on whether or not it will rain.

In classical logic, that would be the end of the story. But, in Decision Theory, I can judge the likelihood that it is going to rain, judge the positive utility of staying dry compared with the negative utility of having to carry the umbrella, weigh the utilities by their associated probabilities, and then choose an action that has the maximum expected utility.

For the record, here is a simple, example calculation:

Utility of getting wet = – 8.
Utility of staying dry = 2.
Utility of carrying an umbrella = – 3
Utility of not carrying an umbrella = 0
Probability of raining = .1
Probability of not raining = .9

Assume I take an umbrella.
Then Probability of staying dry = 1

[46] Alternatively, if the crow wants to have the cheese in order to eat it, then the crow could satisfy both goals by first eating the cheese and then singing.

Expected utility = 2 – 3 = - 1

Assume I do not take an umbrella .
Then Probability of staying dry = .9
Probability of getting wet =.1
Expected utility = .9 ·2 - .1 ·8 = 1.8 - .8 = 1

Decide I do not take an umbrella!

Given the same utilities, the probability of rain would have to be greater than .3 before I would decide to take an umbrella.

Because thinking and deciding are separate components of the agent cycle, any way of thinking is compatible with any way of deciding. Thus the use of ALP for thinking can be combined with Decision Theory or any other way of deciding what to do. This combination of thinking and deciding in ALP agents is pictured in figure 5.

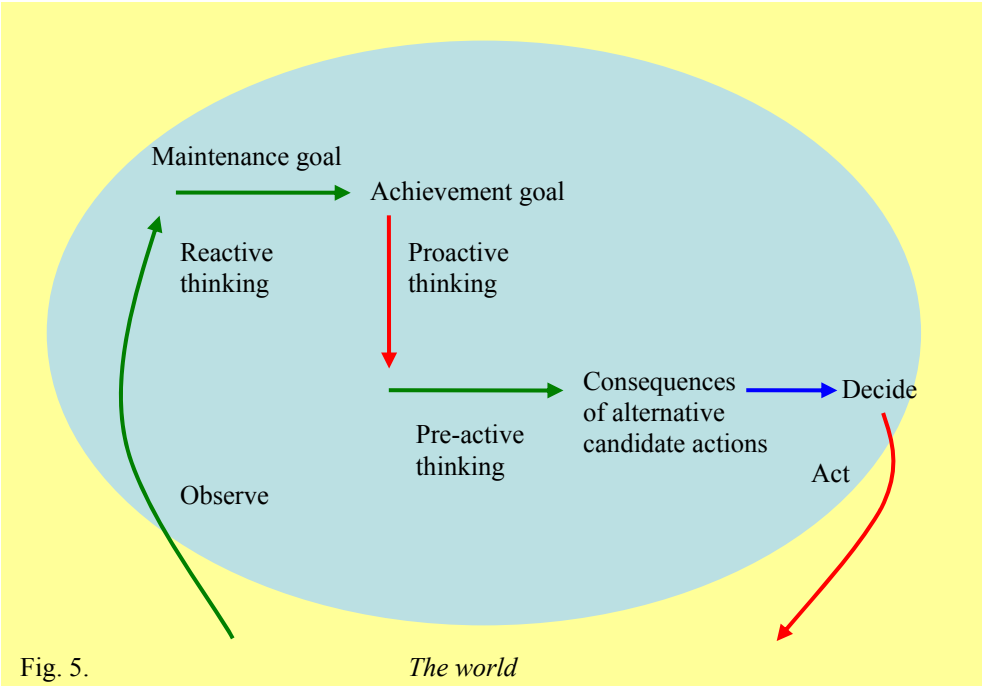


Fig. 5.

A combination of abductive logic programming and Decision Theory has been developed by David Poole in his Independent Choice Logic [8]. He shows how the logic can be used to represent Bayesian networks, influence diagrams, Markov decision processes and the strategic and extensive form of games.

Poole focuses on the semantics of ICL, whereas I focus on the logical and computational procedures an individual agent might use in practice. One consequence of this difference is that he views condition-action rules as *policies*, and represents them by ordinary logic programs, whereas I view them as goals, and represent them as integrity constraints.

4.2 Decision Making Can Often Be Compiled into the Thinking Component of the Agent Cycle

The problem with Decision Theory is that it requires an unrealistic amount of information about utilities and probabilities and too much computation. Nonetheless, Decision Theory represents a normative ideal against which other, more practical decision-making methods can be evaluated.

In the case of taking or not taking an umbrella, a more practical alternative might be to use maintenance goals or condition-action rules instead[47]:

If I leave home and it is raining then I take an umbrella.
If I leave home and there are dark clouds in the sky then I take an umbrella.
If I leave home and the weather forecast predicts rain then I take an umbrella.

The maintenance goals in this example compile decision-making into the thinking component of the agent cycle. In some cases, the compilation might be an exact implementation of the Decision Theoretic specification. In other cases, it might only be an approximation.

Other alternatives to Decision Theory include the use of priorities between different actions, goals or condition-action rules, and the use of default reasoning.

5 Combining Higher-Level and Lower-Level Thinking

5.1 Higher Levels of Thinking Can Be Compiled into Lower Levels

Abductive logic programs have a computational, as well as a logical, interpretation. Goals and beliefs expressed in logical form can be viewed as programs written in a high-level programming language. Programs written at this high, logical level are executed by backward and forward reasoning.

For the sake of efficiency, high-level programs are often compiled into lower-level programs and are executed using corresponding lower-level computational mechanisms. Usually the higher and lower-level programs are written in distinct programming languages. However, they can also be written in the same language.

Compiling a high level program into a more efficient, lower level program written in the same language is called *program transformation.* Program transformation typically gains efficiency by performing at compile time, once and for all, execution steps that would otherwise have to be performed repeatedly, at run time. In the case of abductive logic programs, higher-level programs can be transformed into lower-level, more efficient programs, by performing reasoning steps in advance, before they are needed.

This is easy to see in the London underground example. The original high-level ALP representation can be compiled/transformed into the equivalent, more efficient condition-action rule representation:

If there are flames then I press the alarm signal button.
If there is smoke then I press the alarm signal button.
If one person attacks another then I press the alarm signal button.
If someone becomes seriously ill then I press the alarm signal button.
If there is an accident then I press the alarm signal button.

This lower-level program is written in the same higher-level ALP language as the original representation, but it now consists of five maintenance goals, rather than one maintenance goal and eight beliefs. It is obtained by reasoning in advance, replacing the concept of "emergency" by all of the alternative types of emergency, replacing the concept of "fire" by the two different ways of recognizing a fire, and reducing "getting help" to the action of pressing the alarm signal button.

The two representations are computationally equivalent, in the sense that they give rise to the same externally observable behaviour. However, the lower-level program is more efficient. Not only does it require fewer steps to execute at run time, but it uses simpler reasoning techniques, consisting of forward reasoning alone, instead of the combination of forward and backward reasoning needed by the higher-level program.

[47] In this representation the decision not to take an umbrella is implicit. It holds if the decision to take an umbrella does not hold.

The two representations are not logically equivalent. The high-level representation logically implies the lower-level representation, but not vice versa. In particular, the higher-level representation has an explicit representation of the concepts of there being an emergency and of getting help, which are only implicit in the lower-level representation. Moreover, the higher-level representation also has an explicit representation of the purpose of the agent's behaviour, namely to get help whenever there is an emergency, which is only implicit as an *emergent goal* in the lower-level representation.

In computing, higher-level representations (including program specifications) are generally developed, before they are compiled/transformed into lower-level representations for the sake of efficiency. However, if anything then goes wrong with the lower-level representation, it is generally easier to debug and correct the higher-level representation and to recompile it into the lower-level form, than it is to change the lower-level representation itself.

For example, if something goes wrong with the condition-action rule formulation of the London underground rules - if the button doesn't work, or if the driver doesn't get help - then the rules will fail, but the passenger might not even recognise there is a problem. Or, if the environment changes – if new kinds of emergencies arise or if better ways of getting help are developed – then it is easier to extend the higher-level representation than it is to modify the lower-level rules.

In computing, it is common to iterate the compilation of programs into a number of increasingly lower-levels, and ultimately into hardware. Historically, however, lower-level languages were used before higher-level, more human-oriented languages were developed. Because legacy systems originally developed and implemented in such lower-level languages are difficult to maintain, it is common to re-implement them in modern higher-level languages. This can sometimes be done by an inverse process of *decompiling* lower-level programs into higher-level programs. However, because of the undisciplined nature of low-level programming languages, the attempt to decompile such programs may only be partially successful. In many cases it may only be possible to approximate the lower-level programs by higher-level ones, sometimes only guessing at their original purpose.

5.2 Combining Deliberative and Intuitive Thinking

The relationship between deliberative and intuitive thinking is analogous to the relationship between higher-level and lower-level program execution.

The simplest relationship is when, as the result of frequent repetition, deliberative thinking migrates to the intuitive level – when, for example, a person learns to use a keyboard, play a musical instrument, or drive a car. This is like compiling or transforming a high-level program into a lower-level program. After a particular combination of high-level, general-purpose procedures has been used many times over, the combination is compressed into a computationally equivalent, lower-level shortcut. The shortcut is a special-purpose procedure, which achieves the same result as the combination of more general procedures, but it does so more efficiently and with less awareness of its purpose.

Conversely, intuitive thinking and tacit knowledge can sometimes be made explicit – for example, when a linguist constructs a formal grammar for a natural language, a coach explains how to swing a golf club, or a knowledge engineer develops an expert system. This is like decompiling a low-level representation into a higher-level representation. In many cases it can be hard to distinguish whether the low-level representation is implemented in hardware or in software, and the resulting higher-level representation may only be approximate.

In computing, higher-level and lower-level programs can operate in tandem, as when the lower-level program is used on a routine basis, but the higher-level program is used to modify and recompile the lower-level program when it goes wrong or needs to be updated. In human thinking, however, intuitive and deliberative thinking are often coupled together more closely. Intuitive thinking generates candidate judgments and actions rapidly and unconsciously, while deliberative thinking consciously monitors the results. This close coupling of deliberative and intuitive thinking is like

the use of pre-active thinking in ALP agents to monitor candidate actions generated reactively by condition-action rules.

These relationships between different levels of thinking are pictured, somewhat imperfectly, in figure 6.

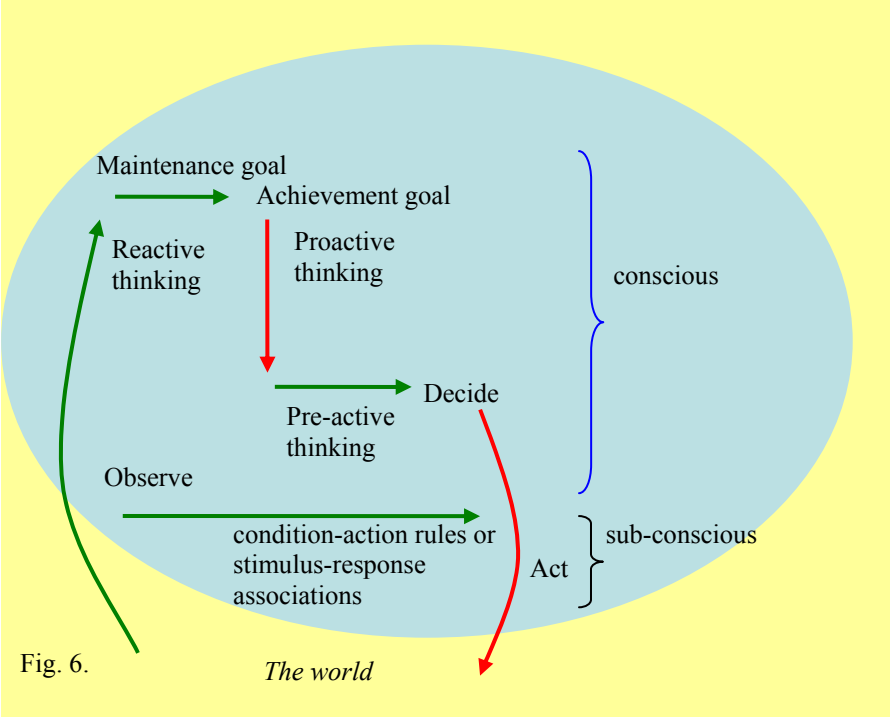


Fig. 6.

5 Neural Networks

It is a common view in Cognitive Science that intuitive thinking is best modelled by sub-symbolic neural networks [13], which employ distributed representations with hidden nodes that do not have a symbolic interpretation. However, in their text book, *Computational Intelligence: A Logical Approach*, Poole *et al* [9] show how to represent any feed-forward neural network as a logic program. Forward reasoning with the logic program simulates forward execution of the neural network.

Poole *et al* illustrate their representation with the example (figure 7) of a person's decision whether to read an article. The decision is based upon such factors as whether the author is known or unknown, the article starts a new thread or is a follow-up article, the article is short or long, and the person is at home or at work.

The weights on the arcs are obtained by training an initial version of the network with a training set of examples. In the logic program, "f" is a sigmoid function that coerces the real numbers into the range [0,1]. Similarly, the "strengths" of the inputs lie in the range [0,1], where 0 is associated with the Boolean value *false* and 1 with *true*.

It is generally held that neural networks are unlike logic, in that they can have hidden units that can not be interpreted as concepts or propositions. Indeed, Poole *et al* characterize their example as illustrating just that point. However, in my formulation of the logic program, to make it more readable, I have given the predicate symbols "meaningful" predicate names, interpreting the hidden units in the middle layer of the network as summarizing the arguments for and against reading the paper.

Example	Action	Author	Thread	Length	Where read
E1	skip	known	new	long	home
E2	reads	unknown	new	short	work
E3	skips	unknown	follow-up	long	work

Neural network

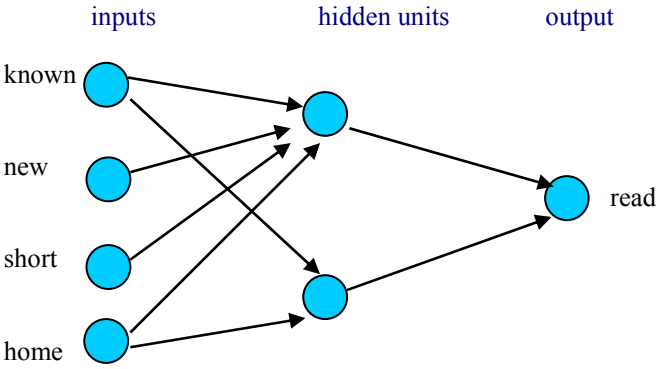


Fig. 7.

Logic program

I read with strength S_3
if there is an argument for reading with strength S_1
and there is an argument against reading with strength S_2
and $S_3 = f(-2.98 + 6.88\ S_1 - 2.1\ S_2)$

There is an argument for reading with strength S_1
if known with strength S_4
and new with strength S_5
and short with strength S_6
and home with strength S_7
and $S_1 = f(-5.25 + 1.98\ S_4 + 1.86\ S_5 + 4.71\ S_6 - .389\ S_7)$

There is an argument against reading with strength S_2
if known with strength S_4
and new with strength S_5
and short with strength S_6
and home with strength S_7
and $S_2 = f(.493 - 1.03\ S_4 - 1.06\ S_5 - .749\ S_6 + .126\ S_7)$

The logic program is an exact, logical representation of the neural network. However, it employs numerical values and functions, which can only be approximated by a natural language representation, such as this:

I read an article
if the argument for reading the article is strong
and the argument against reading the article is weak.

There is an argument for reading an article
if the article is short.

There is an argument against reading an article
if the thread is a follow-up and the author is unknown.

The terms “strong” and “weak” are explicitly vague, whereas the notions of “an article being short”, “a thread being new” and “an author being known” are implicitly vague. Taking this into account, the representation can be transformed into a simpler form, where all vagueness is implicit and where the arguments for and against reading an article are also implicit:

> I read an article
> if the article is short and the thread is new.
>
> I read an article
> if the article is short and the thread is a follow-up and the author is known.

Expressed in this way and treating the sentences as goals rather than as beliefs, the problem of deciding whether to read an article is similar to the problem of deciding whether to take an umbrella when leaving home. In both cases, the decision depends upon the interpretation of implicitly vague concepts, such as an article being short or there being dark clouds in the sky.

In both cases, moreover, the decision can also be made at a higher-level, by analysing the goals and other outcomes that the decision might be expected to achieve. In the case of reading an article, is the higher-level goal to gain information? Or is it simply to be entertained? If it is to gain information, how likely is it that the article will contain the information I am looking for? Is it worth the effort involved? Or would it be better to consult an expert instead?

In the case of taking an umbrella when I leave home, is the higher-level goal to keep my hair and clothing neat and tidy? Or is it to avoid catching a chill and coming down with a cold? In the first case, maybe it should just wear a hat and some suitable outdoor clothing. In the second case, if I am so fragile, then maybe I should stay home or travel by taxi.

6 Logic as wide-spectrum language of thought

The neural network example suggests that logic can represent a wide spectrum of levels of thought, ranging from subconscious thought at the neural network level to conscious thought in natural language. At the neural network level, logic programs can represent connections among vague concepts that hold with varying degrees of strength. Although these degrees might have precise values at the neurological level, they are not accessible to higher-levels of consciousness and can only be approximated in natural language.

A number of authors have investigated the relationship between neural networks and logic programs. One of the earliest of these investigations, by Holldobler and Kalinke [15], studied the problem of translating normal logic programs into neural networks. More recently, Stenning and van Lambalgen [16] have argued that the implementation of logic programs by neural networks shows that logic can model intuitive thinking in the dual process model. D’Avila Garcez, Broda and Gabbay [14], on the other hand, studied the problem of extracting higher-level, “meaningful” logic programs from neural networks. Taken together with the direct translation of neural networks into correspondingly low-level logic programs of Poole et al [9], these studies suggest that logic can model different levels of thought, from neural networks to natural language.

The relationship between logic and natural language is normally viewed from the linguistic perspective, by studying the problem of extracting logical meaning from natural language. But it can also be viewed from the knowledge representation perspective, by comparing the logical form of an agent’s thoughts with the communication of those thoughts to another agent in natural language.

Although logical representations are normally presented in symbolic, mathematical form, they can also be expressed in a stylized form of natural language, as in this paper. Both of these forms are unambiguous and context-independent. Thus, to the extent that some form of logic is adequate for knowledge representation, this provides

evidence that human agents might think in a mental language that is a logical form of natural language.

In contrast with the thoughts we have in our mind, our natural language communication of those thoughts is generally more ambiguous and context-sensitive than we intend. This suggests that our thoughts may be more logical than their natural language expression might suggest. Even natural language communications that seem to be in explicit logical form can be more ambiguous than they seem on the surface.

As Stenning and van Lambalgen [16] argue, natural language communications need to be interpreted to determine their intended logical form, even when those communications are already expressed in logical form. They argue that the gap between the surface logical structure of sentences and the deeper logical structure of their intended meanings helps to explain and refute certain psychological experiments that suggest that people are not logical. They show, moreover, that human performance in these experiments is compatible with the thesis that people apply logical reasoning to the intended meanings of sentences rather than to their surface form. In fact, in their main example, they show, not only that the intended meanings of sentences can be expressed in logical form, but that they have logic programming form, and that the minimal model semantics of logic programs gives a better analysis of human performance in these experiments than the classical semantics of traditional logic.

This difference between the surface structure of natural language and its underlying logical form is illustrated also by the second sentence of the London underground Emergency Notice:

> If there is an emergency then you press the alarm signal button.
> The driver will stop if any part of the train is in a station.

The second sentence has an explicitly logical form, due to its use of the logical connective "if" and the quantifier "any". However, taken literally, the sentence doesn't express what its authors probably had in mind:

> The driver will stop *the train* if *someone presses the alarm signal button*
> and any part of the train is in a station.

It is likely that most people interpret the second sentence of the Notice as it is intended, rather than as it is expressed. This suggests that people are more logical than many psychologists are inclined to believe.

7 Thinking = Logic + Control

The view that logic can serve as a wide-spectrum language of thought is in marked contrast with conventional views of logic in cognitive science. Paul Thagard [11], for example, in his introductory textbook, "Mind: Introduction to Cognitive Science" (page 45) writes:

> "In logic-based systems the fundamental operation of thinking is ***logical deduction***, but from the perspective of rule-based systems the fundamental operation of thinking is ***search***."

Here he uses the term "rule-based system" to refer to condition-action rule production systems. He then goes on to say that among the various models of thinking investigated in cognitive science, rule-based systems have "the most psychological applications" (page 51).

Jonathan Baron [1] in his textbook, "Thinking and Deciding" writes, page 4:

> "Thinking about actions, beliefs and personal goals can all be described in terms of a common framework, which asserts that thinking consists of ***search*** and ***inference***. We ***search*** for certain objects and then ***make inferences*** from and about the objects we have found."

Baron associates logic with making inferences, but not with performing search. He also distinguishes thinking from deciding, but restricts the application of logic to the pre-active, inference-making component of thinking.

Both Thagard and Baron fail to recognize that, to be used in practice, logic needs to be controlled. This could be put in the form of a pseudo-equation[48]:

Thinking = Logic + Control.

Here the term "Logic" refers to goals and beliefs expressed in logical form and "Control" refers to the manner in which the inference rules of logic are applied. Control includes the use of forward and backward reasoning. In the case of backwards reasoning, it includes strategies for selecting sub-goals, as well as strategies for searching for alternative ways of solving goals and sub-goals. It also includes the application of inference rules in sequence or in parallel.

Frawley [13] argues that the analysis of algorithms into logic plus control also applies to mental algorithms and helps to explain different kinds of language disorders. He argues that Specific Language Impairment, for example, can be understood as a defect of the logic component of mental algorithms for natural language, whereas Williams syndrome and Turner syndrome can be understood as defects of the control component.

In fairness to Thagard and Baron, it has to be acknowledged that they are simply reporting generally held views of logic, which do not take into account some of the more recent developments of logic in Artificial Intelligence. Moreover, both, in their different ways, draw attention to characteristics of thinking that are missing both from traditional logic and from the simple pro-active model of thinking associated with logic programming. Thagard draws attention to the importance of reactive thinking with condition-action rules, and Baron to the importance of pre-active thinking by inference after search.

8 Conclusions

There isn't space in this paper to discuss all of the arguments that have been made against logic. Instead, I have considered only some of the most important alternatives that have been advocated – production systems, decision theory, and neural networks, in particular.

In the case of production systems, I have argued that condition-action rules are subsumed by maintenance goals in logical form. They are the special case of maintenance goals in which no forward reasoning is necessary to process observations, and no backward reasoning is necessary to reduce goals to sub-goals.

In the case of decision theory, I have argued that forward reasoning can be used pre-actively to derive possible consequences of candidate actions, and can be combined with any way of deciding between the alternatives. One such possibility is to use decision theory directly to choose a candidate action having maximum expected utility. Another is to compile such decisions into heuristic maintenance goals that approximate the decision-theoretic normative ideal.

In the case of neural networks, I have considered how the low-level logic-programming representation of feed-forward networks, given by *Poole et al*, might be approximated by higher-level logical representations. I have also suggested that such lower-level and higher-level logical representations might interact in a manner similar to the way in which intuitive and deliberative thinking interact in the dual process model. The lower-level representation proposes intuitive answers to problems as they arise, and the higher-level representation monitors and modifies the proposals as time allows.

I have restricted my attention in this paper to the way in which logic can be used to help control the routine, real-time behaviour of an intelligent agent. Except for program transformation, in which a higher-level representation is compiled into a more efficient, lower-level form, I have not considered the wider issues of learning

[48] In the same sense that *Algorithm = Logic + Control* [5].

and of revising goals and beliefs. Fortunately, there has been much work in this area, including the work on inductive logic programming, which is relevant to this issue.

Again for lack of space, I have not been able to discuss a number of extensions of logic that have been developed in Artificial Intelligence and that are important for human thinking. Among the most important of these is the treatment of default reasoning and its interpretation in terms of argumentation. Also, I do not want to give the impression that all of the problems have been solved. In particular, the treatment of vague concepts and their approximations is an important issue that needs further attention.

Despite the limitations of this paper, I hope that it will suggest, not only that logic deserves greater attention in Cognitive Science, but that it can be applied more effectively by ordinary people in everyday life.

Acknowledgements

Many thanks to the anonymous reviewers and to Ken Satoh for their helpful comments on an earlier draft of this paper.

References

1. Baron, J.: Thinking and Deciding (second edition). Cambridge University Press(1994)

2. Checkland, P.: Soft Systems Methodology: a thirty year retrospective. John Wiley Chichester (1999)

3. Kahneman, D., Shane F.: Representativeness revisited: Attributive substitution in intuitive judgement. In: Heuristics of Intuitive Judgement: Extensions and Applications, Cambridge University Press (2002)

4. Kakas, T., Kowalski, R., Toni, F.: The Role of Logic Programming in Abduction. In: Gabbay, D., Hogger, C.J., Robinson, J.A. (eds.): Handbook of Logic in Artificial Intelligence and Programming 5. Oxford University Press (1998) 235-324

5. Kowalski, R.: Logic for Problem Solving. North Holland Elsevier (1979)

6. Kowalski, R.: How to be artificially intelligent. http://www.doc.ic.ac.uk/~rak/ (2002-2006)

7. Kowalski, R., Sadri, F.: From Logic Programming towards Multi-agent Systems. Annals of Mathematics and Artificial Intelligence. Vol. 25 (1999) 391-419

8. Poole, D.L.: The independent choice logic for modeling multiple agents under uncertainty. Artificial Intelligence. Vol. 94 (1997) 7-56

9. Poole, D.L., Mackworth, A.K., Goebel, R.: Computational intelligence: a logical approach. Oxford University Press (1998)

10. Smith, E.R., DeCoster, J.: Dual-Process Models in Social and Cognitive Psychology: Conceptual Integration and Links to Underlying Memory Systems. Personality and Social Psychology Review. Vol. 4 (2000) 108-131

11. Thagard, P.: Mind: Introduction to Cognitive Science. MIT Press (1996)

12. Vickers, G.: The Art of Judgement. Chapman and Hall, London (1965)

13. Frawley, W.: Control and Cross-Domain Mental Computation: Evidence from Language Breakdown. *Computational Intelligence,* 18(1), (2002) 1-28

14. d'Avila Garcez, A.S., Broda, K., Gabbay, D.M.: Symbolic knowledge extraction from trained neural networks: A sound approach. Artificial Intelligence 125 (2001) 155–207

15. Holldobler, S. Kalinke, Y. : Toward a new massively parallel computational model for logic programming. In *Proceedings of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*, (1994) 68-77

16. Stenning, K., van Lambalgen, M.: Semantic interpretation as computation in non-monotonic logic. Cognitive Science (2006)

17. Nicolas, J.M., Gallaire, H.: Database: Theory vs. interpretation. In Gallaire, H., Minker, J. (eds.): Logic and Databases. Plenum, New York (1978)

Appendix 2

Computational Logic in an Object-Oriented World

Robert Kowalski

Imperial College London
rak@doc.ic.ac.uk

Abstract Logic and object-orientation (OO) are competing ways of looking at the world. Both view the world in terms of individuals. But logic focuses on the relationships between individuals, and OO focuses on the use of hierarchical classes of individuals to structure information and procedures. In this paper, I investigate the similarities and differences between OO and abductive logic programming multi-agent systems (ALP systems) and argue that ALP systems can combine the advantages of logic with the main benefits of OO.

In ALP systems, relationships between individuals are contained in a shared semantic structure and agents interact both with one another and with the environment by performing observations and actions. In OO systems, on the other hand, relationships are associated with objects and are represented by attribute-value pairs. Interaction between objects is performed by sending and receiving messages.

I argue that logic can be reconciled with OO by combining the hierarchical, modular structuring of information and procedures by means of objects/agents, with a shared semantic structure, to store relationships among objects/individuals, accessed by observations and actions instead of by message passing.

Keywords: object-orientation, ALP systems, multi-agent systems, logic programming, Linda

1 Introduction

There was a time in the 1980s when it seemed that Computational Logic (CL) might become the dominant paradigm in Computing. By combining the declarative semantics of logic with the computational interpretation of its proof procedures, it could be applied to virtually all areas of Computing, including program specification, programming, databases, and knowledge representation in Artificial Intelligence.

But today it is Object-Orientation (OO), not Logic, that dominates every aspect of Computing – from modelling the system environment, through specifying system requirements, to designing and implementing the software and hardware. Like CL, OO owes much of its attraction, not only to its computational properties, but also to its way of thinking about the world. If these attractions have real substance, then they potentially undermine not only CL's place inside Computing, but also its way of modelling and reasoning about the world outside Computing.

The aim of this paper is to try to understand what makes OO so attractive and to determine whether these attractions can be reconciled with CL, both in Computing and in the wider world. I will argue that logic-based multi-agent systems can combine the advantages of CL with the main benefits of OO.

I will illustrate my argument by using abductive logic programming (ALP) multi-agent systems [1]. However, most of the argument applies to more general logic-based multi-agent systems, and even to heterogeneous systems that use different programming languages, provided their external interfaces can be viewed in logical terms.

ALP multi-agent systems (ALP systems, in short) are semantic structures, consisting of individuals and relationships, as in the conventional semantics of classical logic. However, in ALP systems, these structures can change state, in the same way that the real world changes state, destructively, without remembering its past. Some individuals in the structure are agents, which interact with the world, by

observing the world and by performing actions on the world. Other individuals passively undergo changes performed by agents, and still other individuals, like numbers, are immutable and timeless.

ALP agents, which are individuals in the ALP semantic structure, also have an internal, syntactic structure, consisting of goals and beliefs, which they use to interact with the world. Their beliefs are represented by logic programs, and their goals are represented by integrity constraints. Their observations and actions are represented by abducible (undefined) predicates.

I argue that such logic-based multi-agent systems share many of the attractions of OO systems. In particular, they share with objects the view that the world consists of individuals, some of which (objects or agents) interact with other individuals and change the state of the world. However, whereas in OO systems relationships among individuals are associated with objects and are represented as attribute-value pairs, in ALP systems relationships belong to the semantic structure of the world.

Both agents in ALP systems and objects in OO systems encapsulate their methods for interacting with the world, hiding their implementation details from other agents and objects. Both agents and objects can inherit their methods from more general classes of agents or objects. Whereas objects use methods implemented in conventional, imperative programming languages, ALP agents use methods implemented by means of goals and beliefs in logical form. The methods used by ALP agents have both a procedural behaviour, as well as a declarative semantics. In the declarative semantics, the goals and beliefs of an agent have a truth value in the semantic structure that is the ALP system as a whole. Normally, beliefs that are true and goals that can be made true are more useful to an agent than ones that are false[49].

Both ALP systems and OO systems share a local notion of change, in which changes can take place in different parts of the world locally, concurrently and independently. This local notion of change contrasts with the global notion that is prevalent in most logical treatments, including the possible world semantics of modal logic and the situation calculus. The global notion of change is useful for theoretical purposes, but the local notion is more useful both as a model of the real world and as a model for constructing artificial worlds.

ALP agent systems differ from OO systems in one other important respect: Whereas objects interact by sending and receiving messages, agents interact by observing and performing actions on the shared semantic structure. This semantic structure acts as a shared environment, similar to the blackboard in a blackboard system [8] and to the tuple-space in a Linda programming environment [2]. In the same way that Linda processes can be implemented in different and heterogeneous programming languages, the methods used by ALP agents can also be implemented in other programming languages, provided their externally observed behaviour can be viewed in logical terms.

In the remainder of the paper, I will first introduce ALP systems in greater detail and then distinguish between the semantic and syntactic views of OO systems. I will then compare OO systems and ALP systems by investigating how each kind of system can be simulated by the other. The directness of these simulations is the basis for the comparison of the two approaches. The simulations are informal and should be viewed more as illustrations than as outlines of formal theorems.

[49] A false belief can be more useful than a true belief, if the truth is too complicated to use in practice.

2 The Logical Way of Looking at the World

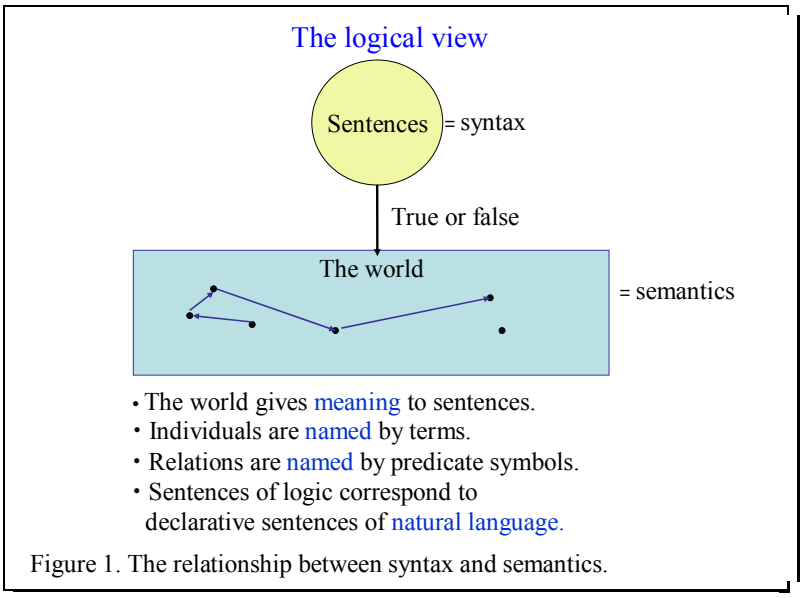


Figure 1. The relationship between syntax and semantics.

In logic there is a clear distinction between syntax and semantics. Syntax is concerned with the grammatical form of sentences and with the inference rules that derive conclusion sentences from assumption sentences. Semantics is concerned with the individuals and relationships that give sentences their meaning. The relationship between syntax and semantics is pictured roughly in figure 1.

The distinction between atomic sentences and the semantic relationships to which they refer is normally formalised by defining an interpretation function, which interprets constant symbols as naming individuals and predicate symbols as naming relations. However, it is often convenient to blur the distinction by restricting attention to *Herbrand interpretations*, in which the semantic structure is identified with the set of all atomic sentences that are true in the structure. However, the use of Herbrand interpretations can sometimes lead to confusion, as in the case where a set of atomic sentences can be considered both semantically as a Herbrand interpretation and syntactically as a set of sentences. Sometimes, to avoid confusion, atomic sentences understood as semantically as relationships are also called *facts*.

For notational convenience, we shall restrict our attention to Herbrand interpretations in the remainder of the paper. However, note that, even viewing Herbrand interpretations as syntactic representations, there is an important sense in which they differ from other syntactic representations. Other syntactic representations can employ quantifiers and logical connectives, which generalize, abstract and compress many atomic sentences into a smaller number of sentences, from which other sentences, including the atomic sentences, can be derived.

2.1 ALP Agents

Traditional logic is often accused by its critics of being too concerned with static states of affairs and of being closed to changes in the world. The first of these criticisms has been addressed in various ways, either by making the semantics more dynamic, as in the possible worlds semantics of modal logic, or by making the syntax more expressive by reifying situations or events, as in the situation or event calculus.

The second of these criticisms has been addressed by embedding logic in the thinking component of the observation-thought-decision-action cycle of an intelligent agent, for example as in ALP agents, pictured in figure 2.

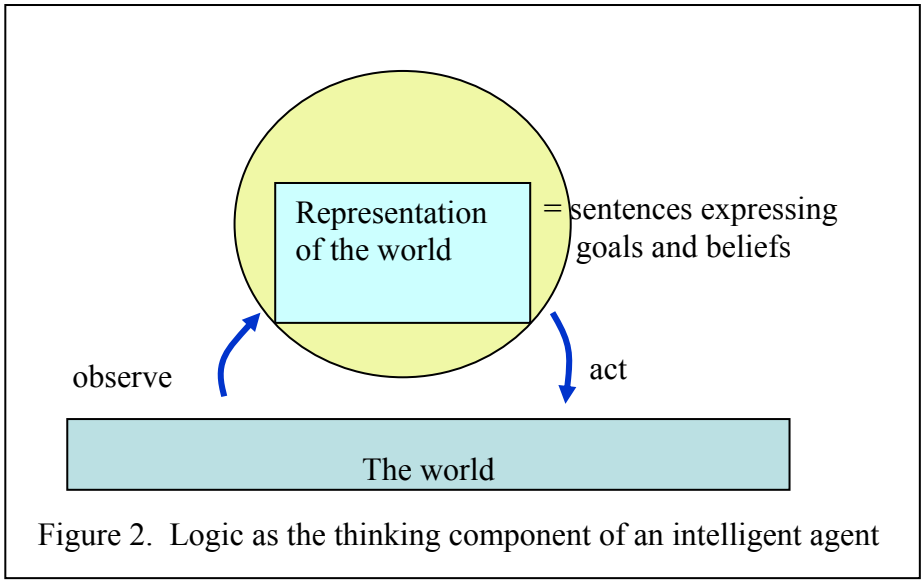


Figure 2. Logic as the thinking component of an intelligent agent

In ALP agents, beliefs are represented by logic programs and goals are represented by integrity constraints. Integrity constraints are used to represent a variety of kinds of goals, including maintenance goals, prohibitions, and condition-action rules. Abducible predicates, which are not defined by logic programs, but are restricted by the integrity constraints, are used to represent observations and actions.

ALP agents implement *reactive* behaviour, initiated by the agent's observations, using forward reasoning to trigger maintenance goals and to derive achievement goals. They also implement *proactive* behaviour, initiated by achievement goals, using backward reasoning to reduce goals to sub-goals and to derive action sub-goals. In addition to reactive and proactive thinking, ALP agents can also perform *pre-active* thinking [20], using forward reasoning to simulate candidate actions, to derive their likely consequences, to help in choosing between them.

2.2 An ALP Agent on the London Underground

Passengers on the London underground have a variety of goals – getting to work, getting back home, going out shopping or visiting the tourist attractions. In addition, most, law-biding passengers are also concerned about safety. This concern can be represented by goals in logical form, which might include the (simplified) goal:

If there is an emergency then I get help.

To recognize when there is an emergency and to find a way to get help, a passenger can use beliefs[50] in logic programming form:

I get help if I alert the driver.
I alert the driver if I press the alarm signal button.

There is an emergency if there is a fire.
There is an emergency if one person attacks another.
There is an emergency if someone becomes seriously ill.
There is an emergency if there is an accident.

The beliefs about getting help are declarative sentences, which may be true or false about the effect of actions on the state of the world. The beliefs about emergencies are also declarative sentences, but they are simply true by definition, because the concept of emergency is an abstraction without a direct interpretation in concrete experience.

[50] For simplicity, this representation of goal and beliefs ignores the element of time.

In ALP, beliefs can be used to reason forwards or backwards. Forward reasoning is useful for deriving consequences of observations and candidate actions. Backward reasoning is useful for reducing goals to sub-goals. A combination of forward and backward reasoning in the London underground example is illustrated in figure 3.

The mental activity of an ALP agent is encapsulated in the agent, hidden from an observer, who can see only the agent's input-output behaviour. In the case of the London underground passenger, this behaviour has the logical form:

If there is a fire, then the passenger presses the alarm signal button.

As far as the observer is concerned, this externally visible, logical form of the passenger's behaviour could be implemented in any other mental representation or programming language.

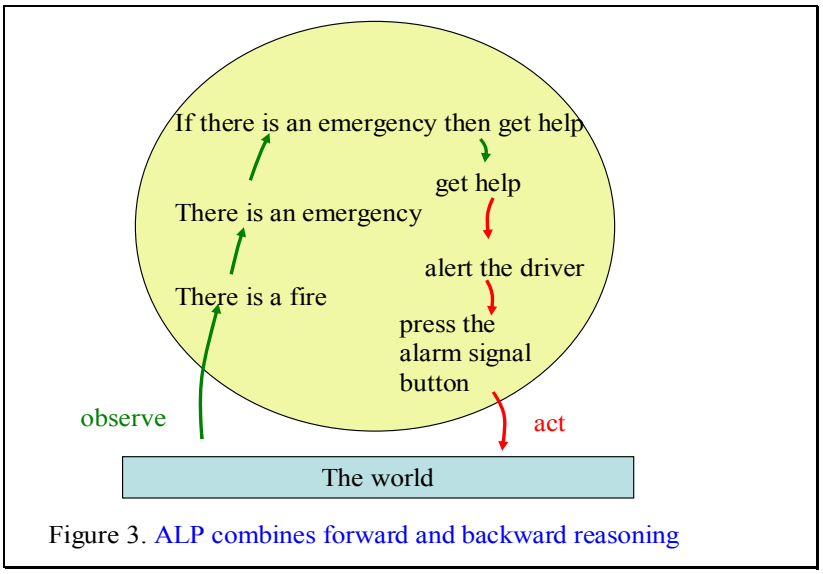


Figure 3. ALP combines forward and backward reasoning

2.3 ALP Agent Systems

Similarly to the way that agents interact with other individuals in real life, ALP agents interact with other individuals embedded in a shared environment. This environment is a semantic structure consisting of individuals and relationships. Some of the individuals in the environment are agents of change, while others undergo changes only passively. To a first approximation, we can think of an ALP environment as a relational database, which changes destructively as the result of agents' actions.

The environment that ALP agents share is a dynamic structure, in which relationships come and go as the result of actions, which occur locally, concurrently and independently of other actions. Because this environment is a semantic structure, relationships can appear and disappear destructively, without the environment having to remember the past. In the London underground example, the observations and actions of the passenger, train driver and fire department agents are illustrated in figure 4. Instead of standing apart and, as it were, above the world, as pictured in figures 1-3, the agents are embodied within it. Their actions change the shared environment by adding and deleting facts (or relationships):

The passenger's action of pressing the alarm signal button
deletes the fact that the alarm is off and
adds the fact that the alarm is on.

The driver's action of calling the fire department
adds the fact that the fire department has been called.

The fire department's action of putting out the fire
deletes the fact that there is a fire in the train.

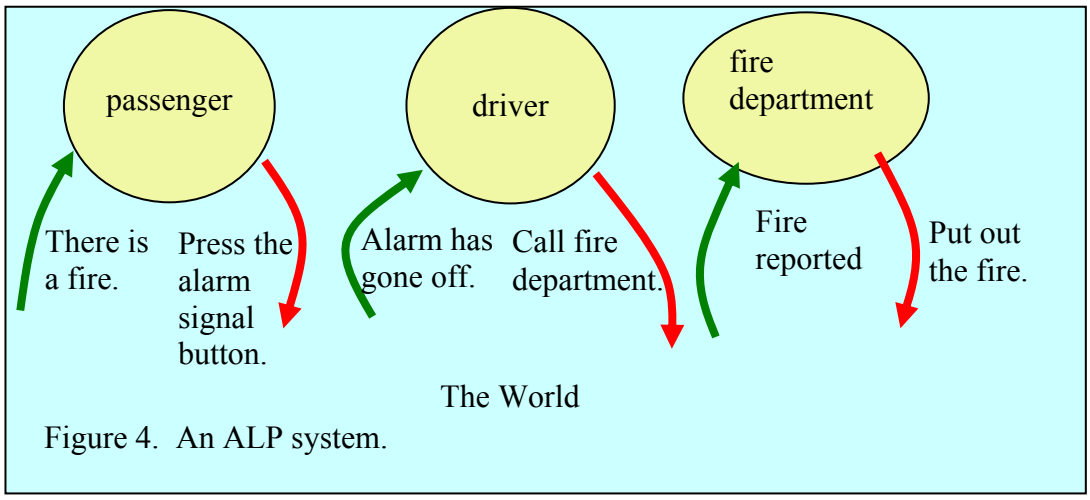


Figure 4. An ALP system.

The driver's action of calling the fire department can be viewed as sending the fire department a message, in the form of a fact that is stored in the shared environment. The fire department observes the message and, if it chooses, may delete it from the environment. Other agents may be able to observe the message, as long as it remains in the environment, provided they can access that part of the environment.

Notice that, just as in the case of a single agent, an observer can see only the agents' external behaviour. In this case also, that behaviour has a logical form:

If there is a fire, then the passenger presses the alarm signal button.
If the alarm has gone off, then the driver calls the fire department.
If a fire is reported, then the fire department puts out the fire.

These implications can be combined with sentences describing the effect of the agents' actions on the world:

If a person presses the alarm signal button, then the alarm goes off.
If a person calls the fire department, then a fire is reported.

to derive the input-output behaviour of the combined system as a whole:

If there is a fire, then the fire department puts out the fire.

3 Object-Oriented Systems

Despite the dominant position of OO in Computing, there seems to be no clear definition or consensus about its fundamental concepts. One recent attempt to do so [21] identifies inheritance, object, class, encapsulation, method, message passing, polymorphism, and abstraction, in that order, as its most frequently cited features. However, the relative importance of these concepts and their precise meaning differs significantly from one OO language to another. This makes comparison with logic very difficult and prone to error. Therefore, the claims and comparisons made in this paper need to be judged and qualified accordingly.

Nonetheless, viewed in terms of the concepts identified in [21], the argument of this paper can be simply stated as claiming that all of these concepts are either already a feature of ALP systems (and other, similar logic-based multi-agent systems) or can readily be incorporated in them, with the exception of *message-passing*.

OO shares with logic the view that the world consists of individuals, some of which (objects or agents) interact with other individuals and change the state of the world. In OO, objects interact with one another by sending and receiving messages, using encapsulated methods, which are hidden from

external observers, and which are acquired from more general classes of objects, organised in hierarchies.

Whereas ALP agents use goals and beliefs to regulate their behaviour, objects use methods that are typically implemented by means of imperative programming language constructs. An object-oriented system corresponding to the multi-agent system of figure 4 is pictured in figure 5.

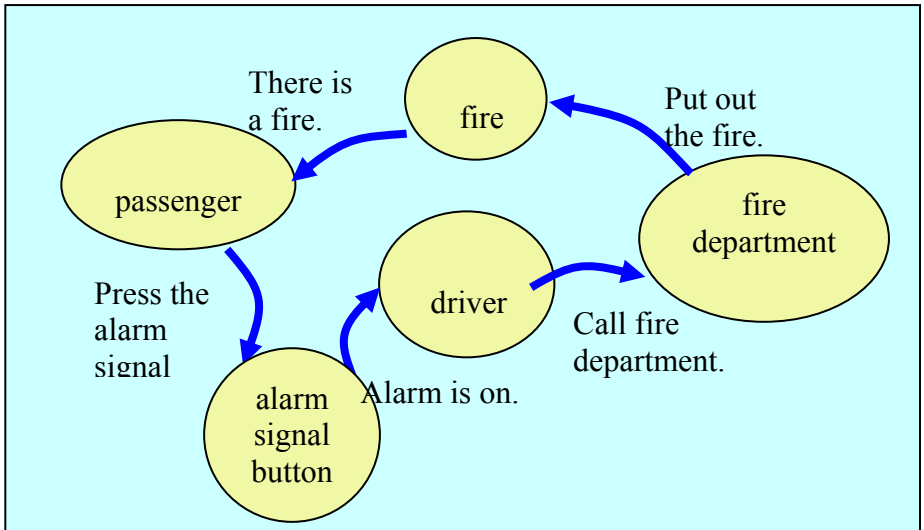


Figure 5. An object-oriented system

Both ALP systems and OO systems can be viewed as semantic structures, in which the world is composed of individuals that interact with one another and change state. However, there are important differences between them:

1. *The treatment of individuals.* In ALP systems, a distinction is made between agents, which are active, and other individuals, which are passive. In OO systems, both kinds of individuals are treated as objects.
2. *The treatment of attributes and relationships.* In the semantic structures of logic, individuals have externally observable attributes and relationships with other individuals. Attributes are treated technically as a special case of relationships.

   In OO systems, relationships between objects are treated as attributes of objects. Either one of the objects in a relationship has to be treated as its "owner", or the relationship needs to be represented redundantly among several "owners".
3. *The way of interacting with the world.* ALP agents interact with the world by observing the current state of the world and by performing actions to change it. A relationship between several individuals can be accessed in a single observation; and a single action can change the states of several relationships.

   Objects in OO systems, on the other hand, interact with one another directly by sending and receiving messages. But the concept of "message" is not defined. In many - perhaps most - cases, messages are used to request help from other objects in solving sub-goals. In other cases, messages are used to send information (including solutions of sub-goals) from one object to another. But in the general case, in many OO languages, messages can be used for any arbitrary purpose.

3.1 Object-oriented Systems as Syntactic Structures

The relationship between logic and objects can be viewed in both semantic and syntactic terms. However, it is the syntactic structuring of information and methods into encapsulated hierarchies of classes of objects that is perhaps the most important reason for the practical success of OO in Computing.

In ALP systems, information and methods are syntactically formulated by means of goals and beliefs in logical form. In OO systems, methods are typically implemented in an imperative language. In both cases, internal processing is encapsulated, hidden from other agents or objects, and performed by manipulating sentences in a formal language.

In logic, there is a well understood relationship between syntax and semantics, in which declarative sentences are either true or false. In ALP agents, declarative sentences representing an agent's goals and beliefs are similarly true or false in the semantic structure in which the agent is embedded.

In OO systems, where methods are implemented in imperative languages, there is no obvious relationship between the syntax of an object's methods and the semantic structure of the OO system as a whole. In part, this is because purely imperative languages do not have a simple truth-theoretic semantics; and, in part, because messages do not have a well defined intuitive interpretation.

3.2 Natural Language and Object-orientation

We can better understand the nature of OO and logical syntax by comparing them both with natural language. Comparing logic with natural language, an important difference is that sets of sentences in logic are unstructured and can be written in any order, without affecting their meaning. But in natural language, the way in which sentences are grouped together and the order in which they are presented affects both their meaning and their intelligibility.

In contrast with logic, but like natural language, OO is also concerned with the structuring of sentences. It structures sentences by associating them with the objects that the sentences are about. Natural languages, like English, employ a similar form of object-orientation by using grammatical structures in which the beginning of a sentence indicates a *topic* and the rest of the sentence is a *comment* about the topic. This kind of structure often coincides with, but is not limited to, the grammatical structuring of sentences into *subjects*[51] and *predicates*.

Consider, for example, the pair of English sentences [3, p. 130]:

The prime minister stepped off the plane.
Journalists immediately surrounded her.

Both sentences are formulated in the active voice, which conforms to the guidelines for good writing style advocated in most manuals of English.

The two sentences refer to three individuals/objects, the prime minister (referred to as "her" in the second sentence), journalists and the plane. The prime minister is the only object in common between the two sentences. So, the prime minister is the object that groups the two sentences together. However, the topic changes from the prime minister in the first sentence to the journalists in the second.

Now consider the following logically equivalent pair of sentences:

The prime minister stepped off the plane.
She was immediately surrounded by journalists.

Here the two sentences have the same topic, which is the individual/object they have in common. However, the second sentence is now expressed in the passive voice.

Despite the fact that using the passive voice goes against the standard guidelines of good writing style, most people find the second pair sentences easier to understand. This can be interpreted as suggesting that people have a strong preference for organising their thoughts in object-oriented form, which is even stronger than their preference for the active over the passive voice.

However, OO is not the only way of structuring sentences. Both linguists and proponents of good writing style have discovered a more general way, which includes OO as a special case. As Joseph Williams [4] argues:

Whenever possible, express at the beginning of a sentence ideas already stated, referred to, implied, safely assumed, familiar, predictable, less important, readily accessible.

Express at the end of a sentence the least predictable. The newest, the most important, the most significant information, the information you almost certainly want to emphasize.

This more general way of structuring sentences also includes the use of logical form to make sets of sentences easier to understand. For example:

[51] In this analogy between objects and topics, objects are more like the grammatical subjects of sentences than they are like the grammatical objects of sentences.

A if B.	or	D.
B if C.		If D then C.
C if D.		If C then B.
D.		If B then A.

3.3 Classes in Object-Oriented Systems Correspond to Sorts in Logic

Perhaps the most important practical feature of OO systems is the way in which objects acquire their methods from more general classes of objects. For example, an individual passenger on the underground can obtain its methods for dealing with fires from the more general class of all humans, and still other methods from the class of all animals.

Thus, classes can be organised in taxonomic hierarchies. Objects acquire their methods from the classes of which they are instances. Similarly sub-classes can inherit their methods from super-classes higher in the hierarchy, possibly adding methods of their own.

However, classes and class hierarchies are neither unique nor original features of OO systems. Classes correspond to types or sorts in many-sorted logics, and hierarchies of classes correspond to hierarchies of sorts in order-sorted logics.

Sorts and hierarchies of sorts can be (and have been) incorporated into logic programming in many different ways. Perhaps the simplest and most obvious way is by employing explicit sort predicates, such as Passenger(X) or Human(X), in the conditions of clauses, together with clauses defining sort hierarchies and instances, such as:

```
Human(X) if Passenger(X)
Passenger(john).
```

However, even unsorted logic programs already have a weak, implicit hierarchical sort structure in the structure of terms. A term f(X), where f is a function symbol, can be regarded as having sort f( ). The term f(g(X)), which has sort f(g( )) is a sub-sort of f( ), and the term f(g(a)), where a is a constant symbol is an instance of sort f(g()). Two terms that differ only in the names of variables, such as f(g(X)) and f(g(Y)) have the same sort. Simple variables, such as X and Y, have the universal sort. Both explicitly and implicitly sorted logic programs enjoy the benefits of inheritance[52].

Although sorts and inheritance are already features of many systems of logic, OO goes further by grouping sentences into classes. Sentences that are about several classes, such as the methods for humans dealing with fire, have to be associated either with only one of the classes or they have to be associated with several classes redundantly.[53]

3.4 Object-Oriented Logic Programming

We can better understand the relationship between OO systems and ALP systems if we see what is needed to transform one kind of system into the other. First, we will show how, under certain restrictions, logic programs can be transformed into OO systems. Later, we will show how to extend this transformation to ALP systems, and then we will show how to transform OO systems into ALP systems. In each of these cases, the OO system is an idealized system, corresponding to no specific OO language in particular. Thus the claims made about these transformations need to be qualified by this limitation.

---

[52] Inheritance can be inhibited by the use of abnormality predicates. For example, the clauses Fly(X) if Bird(X) and not Abnormal(X), Bird(X) if Penguin(X), Walk(X) if Penguin(X), Swim(X) if Penguin(X), Abnormal(X) if Penguin(X) prevent penguins from inheriting the property of flying from the more general class of all birds.

[53] The problem of choosing a class or class hierarchy to contain a given sentence is similar to the problem of choosing a folder to store a file. Search engines like Google make it possible to store information in one structure but to access it without reference to the place it is stored. It also makes it possible to store information in an unstructured way, without any penalty in accessing it. Email clients offer similar, but more limited facilities to order emails by such different attributes as sender, date, subject, size, etc., without needing to duplicate them.

OO grouping of sentences into classes can be applied to any language that has an explicit or implicit class structure, including sentences written in formal logic. However, as we have just observed, an arbitrary sentence can be about many different individuals or classes, making it hard to choose a single individual or class to associate with the sentence.

But it is easier to choose a class/sort for clauses in logic programs that define input-output predicates. For such programs, it can be natural to nominate one of the input arguments of the conclusion of a clause (or, more precisely, the sort of that argument) to serve as the "owner" of the clause. The different instances of the nominated input argument behave as sub-classes and objects, which use the clause as a method to reduce goals to sub-goals.

Whereas arbitrary messages in OO systems may not have a well-defined intuitive interpretation, messages in OO systems that implement input-output logic programs either send requests to solve goals and sub-goals or send back solutions. An object responds to a message requesting the solution of a goal by using a clause to reduce the goal to sub-goals. The object sends messages, in turn, requesting the solution of the sub-goals, to the objects that are the owners of the sub-goals. When an object solves a goal or sub-goal, it sends the solution back to the object that requested the solution.

More formally, let a logic program contain the clause:

$P_0(o_0 , t_{01} , \dots , t_{0m0})$ if $P_1(o_1 , t_{11} , \dots , t_{1m1})$ and … and $P_n(o_n , t_{n1} , \dots , t_{nmn})$

where, without loss of generality, the sort of the first argument of each predicate is selected as the owner of the clause. We also call that argument the "owner argument". Assume also that each such owner argument o_i is an input argument in the sense that at the time the predicate is invoked, as a goal or sub-goal for solution, the argument o_i is instantiated to some object (variable-free term).

Assume for simplicity that the sub-goals in the body of the clause are executed, Prolog-fashion, in the order in which they are written. Then the use of the clause to solve a goal of the form $P_0(o'_0 , t'_{01} , \dots , t'_{0m0})$, where o'_0 is a fully instantiated instance of o_0, is simulated by some sender object o sending the goal in a message to the receiver object o'_0 and by the receiver object:

0. matching the goal with the head of the clause,
   obtaining some most general unifying substitution θ_0

1. sending a message to object $o_1 \theta_0$ to solve the sub-goal
   $P_1(o_1 , t_{11} , \dots , t_{1m1}) \theta_0$
   receiving a message back from $o_1 \theta_0$ reporting that the sub-goal
   $P_1(o_1 , t_{11} , \dots , t_{1m1}) \theta_0$ has been solved with substitution θ_1
   ……

n. sending a message to object $o_n \theta_0\theta_1 \dots \theta_{n-1}$ to solve the goal
   $P_n(o_n , t_{n1} , \dots , t_{nmn}) \theta_0\theta_1 \dots \theta_{n-1}$
   receiving a message back from $o_n \theta_0\theta_1 \dots \theta_{n-1}$ that the goal
   $P_n(o_n , t_{n1} , \dots , t_{nmn}) \theta_0\theta_1 \dots \theta_{n-1}$ has been solved with substitution θ_n

n+1. sending a message back to the sender object o that the goal
   $P_0(o_0, t_{01} , \dots , t_{0m0}) \theta_0$ has been solved with substitution $\theta_0\theta_1 \dots \theta_{n-1} \theta_n$.

Notice that objects o_i and o_j need not be distinct. The special case of an object sending a message to itself can be short circuited by the object simply solving the sub-goal locally itself.

If n=0, then the clause represents a relationship between the owner object and other individuals. The relationship is represented only once, associated with the owner object. All such relationships, like all other clauses, are encapsulated and can be accessed by other objects only by sending and receiving messages.

For example, the atomic clauses:

Father(john, bill)
Father(john, jill)
Mother(mary, bill)
Mother(mary, jill)

would be encapsulated within the john and mary objects. The goal Father(john, X) sent to the object john would receive two messages X=bill and X=jill in return.

It would not be possible with this simple implementation to find out who are the parents of bill or jill. This problem can be solved by redundantly nominating more than one argument to serve as the owner of a clause.

Notice that methods can be public or private. Public methods are ones that are known by other objects, which those objects can invoke by sending messages. Private methods are ones that are used only internally.

3.5 Polymorphism

The mapping from input-output logic programs to OO systems illustrates polymorphism. In the context of OO systems, polymorphism is the property that the "same" message can be sent to and be dealt with by objects belonging to different classes; i.e. except for the class of the recipient, everything else about the message is the same:

$P(o , t_1 , \dots , t_m)$ and $P(o', t_1 , \dots , t_m)$.

Thus objects from different classes can respond to the same message using different methods.

Like classes and class hierarchies, polymorphism is neither a unique nor an original feature of OO systems. In the context of logic, polymorphism corresponds to the fact that the same predicate can apply to different sorts of individuals.

3.6 Aspects as Integrity Constraints in Abductive Logic Programming

The mapping from logic programs to OO systems highlights a number of features of logic programming that are not so easily addressed in OO systems. I have already mentioned the problem of appropriately representing relationships, as well as the problem about representing more general logic programs that do not have input-output form. However, a problem that has attracted much attention in software engineering, and which is addressed in ALP, is how to represent *cross-cutting concerns*, which are behaviours that span many parts of a program, but which can not naturally be encapsulated in a single class.

Integrity constraints in ALP have this same character. For example, the concern:

If a person enters a danger zone,
then the person is properly equipped to deal with the danger.

cuts across all parts of a program where there is a sub-goal in which a person needs to enter a danger zone (such as a fireman entering a fire). In ALP, this concern can be expressed as a single integrity constraint, but in normal LP it needs to scattered throughout the program, by adding to any clause that contains a condition of the form:

a person enters a danger zone

an additional condition:

the person is properly equipped to deal with the danger.

In software engineering the problem of dealing with such cross-cutting concerns is the focus of *aspect-oriented programming* (AOP) [5]. AOP seeks to encapsulate such concerns through the introduction of a programming construct called an *aspect.* An aspect alters the behavior of a program by applying additional behavior at a number of similar, but different points in the execution of the program.

Integrity constraints in ALP give a declarative interpretation to aspects. ALP provides the possibility of executing such integrity constraints as part of the process of pre-active thinking [20], to monitor actions before they are chosen for execution. This is like using integrity constraints to monitor updates in a database, except that the updates are candidates to be performed by the program itself.

It is also possible to transform logic programs with integrity constraints into ordinary logic programs without integrity constraints [6,7]. This is similar to the way in which aspects are implemented in AOP. However, whereas in AOP the programmer needs to specify the "join points"

where the aspects are to be applied, in logic programming the transformations of [6, 7] can be performed automatically by matching the conditions of integrity constraints with conditions of program clauses.

4 The Relationship between OO systems and ALP Systems

The mapping from input-output logic programs to OO systems can be extended to more general ALP agent systems, and a converse mapping is also possible. These mappings exploit the correspondence between agents and objects, in which both are viewed semantically as individuals, mutually embedded with other individuals in a common, dynamically changing world. Both agents and objects process their interactions with the world, by manipulating sentences in a formal language, encapsulated, and hidden from other individuals.

The biggest difference between logic and objects, and therefore between ALP agents and objects, is their different views of semantic structure. For logic, the world is a relational structure, consisting of individuals and relationships that change over time. Such changes can be modeled by using the possible world semantics of modal logic or by treating situations or events as individuals, but they can also be modeled by using destructive assignment.

With destructive assignment, the world exists only in its current state. Agents perform actions, which initiate new relationships (by adding them) and terminate old relationships (by deleting them), without the world remembering its past. The agents themselves and their relationships with other individuals are a part of this dynamically and destructively changing world.

ALP agents, as well as undergoing destructive changes, can also represent changes internally among their beliefs. Using such syntactic representations of change, they can represent, not only the current state of the world, but also past states and possible future states. We will return to this use of logic to represent change later in the paper.

Whereas logic distinguishes between changes that take place in the semantic structure of the world and changes that are represented syntactically in an agent's beliefs, objects do not. In OO systems, all changes of state are associated with objects. It makes it easy for objects to deal with changes of values of attributes, but more difficult for them to deal with changes of relationships.

The different ways in which logic and objects view the world are reflected in the different ways in which they interact with the world. In OO systems, because the state of the world is distributed among objects as attribute-value pairs, the only way an object can access the current state is by accessing the attribute-value pairs of objects. The only way an object can change the current state is by changing the attribute-value pairs of objects. In some OO languages these operations are carried out by sending and receiving messages. In other OO languages they are performed directly.

ALP agents, on the other hand, interact by observing and acting on the external world. These interactions typically involve observing and changing relationships among arbitrarily many individuals, not only attributes of individual objects. This way of interacting with the world is similar to the way that processes use the Linda tuple-space as a shared environment and to the way that experts use the blackboard in a blackboard expert system.

4.1 Transformation of ALP systems into OO systems

Agents. Our earlier transformation of input-output logic programs into OO systems implicitly treats the owners of clauses as agents. In this transformation, the owner of a clause/belief is selected from among the input arguments of the conclusion of the clause. However, in ALP systems, goals and beliefs are already associated with the agents that are their owners. In transforming ALP systems into OO systems, therefore, it is a simple matter just to treat agents as objects and to treat their goals and beliefs as the objects' methods. In some cases, this transformation of agents into objects coincides with the transformation of input-output logic programs into OO systems. However, in many other cases, it is more general.

The ALP semantic structure. An agent's beliefs include its beliefs about relationships between individuals, expressed as unconditional clauses. The individuals included in these relationships need not explicitly include the agent itself, as in the case of a passenger's belief that there is a fire in a train. These beliefs can be used as methods to respond to requests for information from other agents/objects.

In addition to these syntactic representations of relationships as beliefs, an ALP system as a whole is a semantic structure of relationships between individuals. This semantic structure can also be

transformed into objects and their associated attribute-values, similarly to the way in which we earlier associated input-output clauses with owner objects and classes.

However, to obtain the full effect of ALP systems, we need to let each of the individuals o_i in a semantic relationship $P(o_1 , \ldots, o_n)$ (expressed as an atomic fact) be an object, and to associate the relationship redundantly with each of the objects o_i as one of its attribute-values. The problem of representing such relationships redundantly has been recognized as one of the problems of OO, and representing relationships as aspects in AOP [9] has been suggested as one way of solving the problem.

Notice that an object corresponding to an agent can contain two representations of the same relationship, one as an attribute-value representation of a semantic relationship, and the other as a very simple method representing a belief. When the two representations give the same result, the belief is true. When they give different results, the belief is false.

Actions. Actions and observations need to be transformed into sending and receiving messages. An action performed by an agent A that initiates relationships

$P_1(o_{11} , \ldots, o_{1l1})$
……..
$P_n(o_{n1} , \ldots, o_{nln})$

and terminates relationships

$Q_1(p_{11} , \ldots, p_{1k1})$
……..
$Q_m(p_{m1} , \ldots, p_{mkn})$

is transformed into a message sent by object A to each object o_{ij} to add $P_i(o_{i1} , \ldots, o_{il1})$ to its attribute-values together with a message sent by A to each object p_{ij} to delete $Q_i(p_{i1} , \ldots, pi_{k1})$ from its attribute-values.

Observations. The transformation of observations into messages is more difficult. Part of the problem has to do with whether observations are active (intentional), as when an agent looks out the window to check the weather, or whether they are passive (unintentional), as when an agent is startled by a loud noise. The other part of the problem is to transform an observation of a relationship between several objects into a message sent by only one of the objects as the messenger of the relationship. To avoid excessive complications, we shall assume that this problem of selecting a single messenger can be done somehow, if necessary by restricting the types of observations that can be dealt with.

An *active observation* by agent A of a relationship $P(o_1 , o_2 , \ldots, o_n)$ is transformed into a message sent by object A to one of the objects o_j, say o_1 for simplicity, requesting the solution of a goal $P(o_1 , o_2' , \ldots, o_n')$ and receiving back a message from o_1 with a solution θ, where $P(o_1 , o_2 , \ldots, o_n) = P(o_1 , o_2' , \ldots, o_n') \theta$.[54]

A *passive observation* of the relationship can be modeled simply by some object o_j sending a message to A of the relationship $P(o_1 , o_2 , \ldots, o_n)$.

The objects that result from this transformation do not exhibit the benefits of structuring objects into taxonomic hierarchies. This can be remedied by organising ALP agent systems into class hierarchies, similar to sort hierarchies in order-sorted logics. The use of such hierarchies extracts common goals and beliefs of individual agents and associates them with more general classes of agents. Translating such extended ALP agent systems into OO systems is entirely straight-forward.

4.2 Transformation of OO systems into ALP systems

The semantic structure. Given the current state of an OO system, the corresponding semantic structure of the ALP system is the set of all current object-attribute-values, represented as binary relationships, attribute(object, value), or alternatively as ternary relationships, say as relationship(object, attribute, value).

[54] In most OO languages this can be done more directly, without explicitly sending and receiving messages. This more direct access to an object's attribute-values can be regarded as analogous to an agent's observations and actions.

Agents. We distinguish between *passive objects* that merely store the current values of their attributes and *active objects* that both store their current values and also use methods to interact with the world. Both kinds of objects are treated as individuals in the ALP semantic structure. But active objects are also treated as *agents*.

Messages. The treatment of messages is dealt with case by case:

Case 1. A passive object sends a message to another object. By the definition of passive object, this can only be a message informing the recipient of one of the passive object's attribute-values. The only kind of recipient that can make use of such a message is an active object. So, in this case, the message is an *observation* of the attribute-value by the recipient. The observation is *active* (from the recipient's viewpoint) if the message is a response to a previous message from the recipient requesting the sender's attribute-value. Otherwise it is *passive*.

Case 2. An active object sends a message to another object requesting one of the recipient's attribute-values. This is simply the first half of an *active observation* of that attribute-value. The second half of the observation is a message from the recipient sending a reply. If the recipient does not reply, then the observation fails.

Case 3. An active object sends a message to another object changing one of the recipient's attribute-values. The message is simply an *action* performed on the semantic structure.

Case 4. An active object sends any other kind of message to another active object. The message is a combination of an *action by the sender* and an *observation by the recipient*. The action, like all actions, is performed on the semantic structure. In this case the action adds a ternary relationship between the sender, the recipient and the content of the message to the semantic structure. The observation, like all observations, syntactically records the semantic relationship as a belief in the recipient's internal state. The observation then becomes available for internal processing, using forward and backward reasoning with goals and beliefs to achieve the effect of methods. The recipient may optionally perform an action that deletes the ternary relationship from the semantic structure.

Methods. Finally, we need to implement methods by means of goals and beliefs (or equivalently, for ALP agents, by integrity constraints and logic programs). Recall that, in our analysis, only active objects (or agents) employ methods, and these are used only to respond to messages that are transformed into observations. Other messages, which simply request or change the values of an object's attributes, are transformed into operations on the semantic structure.

We need a sufficiently high-level characterization of such methods, so they can be logically reconstructed. For this reason, we assume that methods can be specified in the following input-output form:

> If observation and (zero or more) conditions,
> then (zero or more) actions.

This is similar to event-condition-action rules in active databases [10] and can be implemented directly as integrity constraints in logical form. However, such specifications can also be implemented at a higher level, by means of more abstract integrity constraints (with more abstract conditions and higher-level conclusions), together with logic programs. At this higher level, an agent implements an active object's method by

- recording the receipt of the message as an observation;
- possibly using the record of the observation to derive additional beliefs;
- possibly using the record of the observation or the derived additional beliefs to trigger an integrity constraint of the form:

  if conditions, then conclusion

- verifying any remaining conditions of the integrity constraint and then,
- reducing the derived conclusion of the constraint, treated as a goal, to sub-goals, including actions.

Beliefs, in the form of logic programs, can be used to reason forwards from the observation and backwards both from any remaining conditions of the integrity constraint and from the conclusion of the integrity constraint. In addition to any actions the agent might need to perform as part of the specification of the method, the agent might also send requests to other agents for help in solving sub-goals, in the manner of the object-oriented logic programs of section 3.

All messages that do not request or change attribute-values are treated by the recipient uniformly as observations. If the message is a request to solve a goal, then the recipient records the request as an

observation and then determines whether or not to try to solve the goal, using an integrity constraint such as:

If an agent asks me to solve a goal,
and I am able and willing to solve the goal for the agent,
then I try to solve the goal and I inform the agent of the result.

The recipient might use other integrity constraints to deal with the case that the recipient is unable or unwilling to solve the goal.

Similarly, if the message is a communication of information, then the recipient records the communication as an observation and then determines whether or not to add the information to its beliefs. For example:

If an agent gives me information,
and I trust the agent,
and the information is consistent with my beliefs,
then I add the information to my beliefs.

The input-output specification of OO methods that we have assumed is quite general and hopefully covers most sensible kinds of methods. As we have seen, the specification has a direct implementation in terms of abductive logic programming. However, as we have also noted earlier, other implementations in other computer languages are also possible, as long as they respect the logical specification.

Classes. Methods associated with classes of objects and inherited by their instances can similarly be associated with sorts of ALP agents. This can be done in any one of the various ways mentioned earlier in the paper. As remarked then, this requires an extension of ALP agents, so that goals and beliefs can be associated with sorts of agents and acquired by individual agents. There is an interesting research issue here: whether sentences about several sorts of individuals can be represented only once, or whether they need to be associated, possibly redundantly, with owner classes/sorts.

5 Local versus Global Change

One of the attractions of object-orientation is that it views change in local, rather than global terms. In OO the state of the world is distributed among objects as the current values of their attributes. Change of state is localized to objects and can take place in different objects both concurrently and independently.

Traditional logic, in contrast, typically views change in global terms, as in the possible-worlds semantics of modal logic and the situation calculus. Modal logic, for example, deals with change by extending the static semantics of classical model theory to include multiple (possible) worlds related by a temporal, next-state accessibility relation. Semantically, a change of state due to one or more concurrent actions or events, is viewed as transforming one global possible world into another global possible world. Syntactically, change is represented by using modal operators, including operators that deal with actions and events as parameters, as in dynamic modal logic.

The situation calculus [11] similarly views change as transforming one global possible world (or situation) into another. Semantically, it does so by reifying situations, turning situations into individuals and turning the next-state accessibility relation into a normal relation between individuals. Syntactically, it represents change by using variable-free terms to name concrete situations and function symbols to transform situations into successor situations.

It was, in part, dissatisfaction with the global nature of the possible-worlds semantics that led Barwise and Perry to develop the situation semantics [12]. Their situations (which are different from situations in the situation calculus) are semantic structures, like possible-world semantic structures, but are partial, rather than global.

It was a similar dissatisfaction with the global nature of the situation calculus that led us to develop the event calculus [13]. Like situations in the situation calculus, events, including actions are reified and represented syntactically. The effect of actions/events on the state of relationships is represented by an *axiom of persistence* in logic programming form:

A relationship holds at a time T_2
if an event happens at a time T_1 before T_2

and the event initiates the relationship
and there is no other event
that happens at a time after T_1 and before T_2 and
that terminates the relationship.

Like change of state in OO, change in the event calculus is also localised, but to relationships rather than to objects. Also as in OO, changes of state can take place concurrently and independently in different and unrelated parts of the world. Each agent can use its own local clock, time-stamping observations as they occur and determining when to perform actions, by comparing the time that actions need to be performed with the current time on its local clock.

The event calculus is a syntactic representation, which an agent can use to reason about change. It can be used to represent, not only current relationships, but also past and future relationships, both explicitly by atomic facts and implicitly as a consequence of the axiom of persistence. However, the event calculus does not force an agent to derive current relationships using the persistence axiom, if the agent can observe those relationships directly, more efficiently and more reliably instead.

The event calculus is not a semantics of change. However, in theory, if events are reified, then the use of the event calculus to reason about change should commit an agent to a semantic structure in which events are individuals. But this is the case only if all symbols in an agent's goals and beliefs need to be interpreted directly in the semantic structure in which the agent is embedded. If some symbols can be regarded as defined symbols, for example, then they need not be so interpreted. Alternatively, in the same way that in physics it is possible to hypothesize and reason with the aid of theoretical particles, which can not be observed directly, it may also be possible in the event calculus to represent and reason about events without their being observable and without their corresponding to individuals in the world of experience.

In any case, the event calculus is compatible with a semantic structure in which changes in relationships are performed destructively, by deleting (terminating) old relationships and adding (initiating) new relationships. These destructive changes in the semantic structure are the ones that actually take place in the world, as opposed to the representation of events and the derivations of their consequences using the axiom of persistence, which might take place only in the mind of the agent.

6 Related Work

There is a vast literature dealing with the problem of reconciling and combining logic programming and object-orientation, most of which was published in the 1980s, when the two paradigms were still contending to occupy the central role in Computing that OO occupies today. Most of this early literature is summarized in McCabe's [14].

Perhaps the most prominent approach among the early attempts to reconcile logic programming and objects was the concurrent object-oriented logic programming approach exemplified by [15]. In this approach, an object is implemented as a process that calls itself recursively and communicates with other objects by instantiating shared variables. Objects can have internal state in the form of unshared arguments that are overwritten in recursive calls. Although this approach was inspired by logic programming it ran into a number of semantic problems, mainly associated with the use of committed choice. The problem of committed choice is avoided in ALP systems by incorporating it in the decision making component of individual agents.

In McCabe's language, *L&O* [14], a program consists of a labelled collection of logic programs. Each labelled logic program is like a set of beliefs belonging to the object or agent that is the label. However, in *L&O,* objects/agents interact by sending messages to other agents, asking for their help in solving sub-goals. This is like the object-oriented logic programs of section 3.4. It is also similar to the way multi-agent systems are simulated in GALATEA [22].

ALP systems differ from *L&O*, therefore, primarily in their use of a shared environment instead of messages. This use of a shared environment is similar to the use of tuple-spaces in Linda [2]. In this respect, therefore, ALP systems are closest to the various systems [16-18] that use a Linda-like environment to coordinate parallel execution of multiple logic programs. ALP systems can be viewed, therefore, as providing a logical framework in which the shared environment in such systems can be understood as a dynamic semantic structure.

A different solution to the problem of reconciling logic and objects is the language LO [19], which is a declarative logic programming language using linear logic. The language is faithful to the semantics of linear logic, which however is quite different from the model-theoretic semantics of traditional logic. Communication between objects in LO is similar to that in Linda.

7 Conclusions

In this paper, I have explored some of the relationships between OO systems and ALP systems, and have argued that ALP systems can combine the semantic and syntactic features of logic with the syntactic structuring and dynamic, local behaviour of objects. I have investigated a number of transformations, which show how OO systems and ALP systems can be transformed into one another. These transformations are relatively straight-forward, and they suggest ways in which the two kinds of system are related. Among other applications, the transformations can be used to embed one kind of system into the other, for example along the lines of [22], and therefore to gain the benefits of both kinds of systems.

However, the transformations also highlight a number of important differences, including problems with the treatment of relations and multi-owner methods in OO systems in particular. On the other hand, they also identify a number of issues that need further attention in ALP systems, including the need to clarify the distinction between active and passive observations, to organise agents into more general agent hierarchies, and possibly to structure the shared semantic environment, to take account of the fact that different agents can more easily access some parts of the environment than other parts. In addition, it would be useful to make the relationships between OO systems and ALP systems explored in this paper more precise and to prove them more formally.

As a by-product of exploring the relationships between logic and objects, the transformations also suggest a relationship between logic and Linda. On the one hand, they suggest that Linda systems can be understood in logical terms, in which tuple-spaces are viewed as semantic structures and processes are viewed as agents interacting in this shared semantic environment. On the other hand, they also suggest that ALP systems can be generalised into Linda-like systems in which different processes can be implemented in different languages, provided that the external, logical specification of the processes is unaffected by their implementation.

Acknowledgements I am grateful to Ken Satoh at NII in Tokyo, for valuable discussions and for providing a congenial environment to carry out much of the work on this paper. I would also like to thank Jim Cunningham, Jacinto Davila and Maarten van Emden for valuable comments on an earlier draft of this paper.

It is my pleasure to dedicate this paper to Gigina Aiello, for her inspiration in the early stages of this work, when we first started thinking about logic-based multi-agent systems in the Compulog Project in the late 1980s and early 1990s.

References

1. Kowalski, R., Sadri, F.: From Logic Programming towards Multi-agent Systems. Annals of Mathematics and Artificial Intelligence. (25) (1999) 391- 419
2 Gelernter, D.: Generative Communication in Linda, ACM Transactions on Programming Languages, (7)1 (1985) 80-112
3. Brown, G. Yule, G.: Discourse Analysis. Cambridge University Press (1983)
4. Williams, J.: Style: Towards Clarity and Grace. Chicago University Press (1990)
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-Oriented Programming. Proceedings of the European Conference on Object-Oriented Programming, vol.1241, (1997) 220–242
6. Kowalski, R., Sadri, F.: Logic Programming with Exceptions. New Generation Computing, (9)3,4 (1991) 387-400
7. Toni, F., Kowalski, R.: Reduction of Abductive Logic Programs to Normal Logic Programs. Proceedings International Conference on Logic Programming, MIT Press (1995) 367-381
8. Engelmore, R.S., Morgan, A. (eds.): Blackboard Systems. Addison-Wesley (1988)
9. Pearce, D.J., Noble, J.: Relationship Aspects. Proceedings of the ACM conference on Aspect-Oriented Software Development (AOSD'06). (2006) 75-86
10. Paton, N.W., Diaz, O.: Active Database Systems. ACM Computing Surveys 31(1) (1999)
11. McCarthy, J., Hayes, P.: Some Philosophical Problems from the Standpoint of AI. Machine Intelligence. Edinburgh University Press (1969)
12. Barwise, J., Perry, J.: Situations and Attitudes. MIT-Bradford Press, Cambridge (1983)
13. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. New Generation Computing. (4)1 (1986) 67-95
14. McCabe, F.G.: Logic and Objects. Prentice-Hall, Inc. Upper Saddle River, NJ (1992)
15. Shapiro, E., Takeuchi, A.: Object-Oriented Programming in Concurrent Prolog, New Generation Computing 1(2) (1983) 5-48
16. Andreoli, J.M., Hankin, ., Le Métayer, D. (eds.): Coordination Programming: Mechanisms, Models and Semantics - Imperial College Press, London (1996)

17. Brogi, A., Ciancarini, P.: The Concurrent Language, Shared Prolog. ACM Transactions on Programming Languages and Systems, 13(1) (1991) 99–123
18. De Bosschere, K., Tarau, P.: Blackboard-Based Extensions in Prolog. Software - Practice and Experience 26(1) (1996) 49-69
19. Andreoli, J.-M., Pareschi, R.: Linear Objects: Logical Processes with Built-In Inheritance. New Generation Computing, 9 (1991) 445-473
20. Kowalski, R.: How to be Artificially Intelligent. In Toni, F., Torroni, P. (eds.): Computational Logic in Multi-Agent Systems. LNAI 3900, Springer-Verlag (2006) 1-22
21. Armstrong, D.J.: The Quarks of Object-Oriented Development. Communications of the ACM 49 (2) (2006) 123-128
22. Davila,J., Gomez, E., Laffaille, K., Tucci, K., Uzcategui, M.: Multi-Agent Distributed Simulations with GALATEA. In Boukerche, A., Turner, T., Roberts, D., Theodoropoulos, G. (eds): IEEE Proceedings of Distributed Simulation and Real-Time Applications. IEEE Computer Society (2005) 165-170

Appendix 3

Reasoning with Conditionals in Artificial Intelligence

Robert Kowalski
Department of Computing
Imperial College London

January 2008

1 Introduction

Conditionals of one kind or another are the dominant form of knowledge representation in Artificial Intelligence. However, despite the fact that they play a key role in such different formalisms as production systems and logic programming, there has been little effort made to study the relationships between these different kinds of conditionals. In this paper I present a framework that attempts to unify the two kinds of conditionals and discuss its potential application to the modelling of human reasoning.

Among the symbolic approaches developed in Artificial Intelligence, production systems have had the most application to the modelling of human thinking. Production systems became prominent in the late 1960s as a technology for implementing expert systems in Artificial Intelligence, and then in the 1970s as a cognitive architecture in Cognitive Psychology. However, although they have been used extensively to study human skills, they seem to have had little impact on studies of human reasoning. This lack of interaction between cognitive architectures based on production systems and studies of human reasoning is especially surprising because both emphasise the central role of conditionals.

Conditionals in production systems (called production rules) have the form *if conditions then actions* and look like logical implications. Indeed, some authors, such as Russell and Norvig (2003), take the view that production rules are just logical implications used to reason forward to derive candidate *actions* from *conditions* (*modus ponens*). However, other authors, such as Thagard (2005) deny any relationship between logic and production rules at all.

In some characterisations of production rules, they are described as expressing procedural knowledge about what to do when. In this paper, I will argue that production rules can be given a logical semantics, and this semantics can help to clarify their relevance for modelling human reasoning.

Conditionals are also a distinctive feature of logic programming, which has been used widely, also since the 1970s, both to implement practical applications and to formalise knowledge representation in Artificial Intelligence. Conditionals in logic programming have both a logical interpretation as conditionals of the form *conclusion if conditions* and an interpretation as procedures that reduce the goal of establishing the *conclusion* to the sub-goals of establishing the *conditions.* Goal-reduction is a form of *backward reasoning.* Although forward reasoning has been studied extensively in psychology, backward reasoning seems to have received little attention in studies of human reasoning.

Instead, studies of reasoning in Psychology emphasize the use of classical negation, including its use in such classically valid inferences as *modus tollens* and in such classical fallacies as

denial of the antecedent. In contrast, classical negation has had relatively few applications in Artificial Intelligence, where negation as failure and the closed world assumption (*not P* holds if and only if *P* does not hold) have been more widely used in practice.

Studies of reasoning in Psychology treat affirmation of the consequent as a fallacy. In Artificial Intelligence, however, it is treated as abductive inference. Both abduction and negation as failure are forms of non-monotonic reasoning.

In this paper, I will outline an abductive logic programming (ALP) approach that aims to reconcile production rules and logic within a unifying agent-based framework. In this framework, logic programs are conditionals used to represent an agent's beliefs, and have a descriptive character. Production rules are conditionals in logical form, used to represent the agent's goals, and have deontic force. Abduction is used both to represent hypotheses that explain the agent's observations and actions that might achieve the agent's goals. Abductive hypotheses can have associated probabilities.

The ALP agent framework embeds goals and beliefs in an observation-thought-decision-action cycle, similar to the production system cycle, and similar to other agent cycles developed in Artificial Intelligence. In the ALP agent framework, the thinking component of the cycle is a combination of forwards reasoning, as in production systems, and backwards reasoning, as in logic programming. Negation is represented both by means of negation as failure and by means of integrity constraints.

Because this book is directed primarily towards a psychological audience and because my own expertise lies in Artificial Intelligence, I will focus on the potential contribution of Artificial Intelligence to studies of human reasoning. However, I anticipate that there is much to be gained from applying results about human reasoning to Artificial Intelligence. In particular, it seems likely that results about the psychology of reasoning with negation can suggest useful directions in which to extend AI reasoning techniques.

The remainder of the paper has four main parts, dealing with production systems, logic programs, the relationship between production systems and logic programs, and the ALP agent model. In addition, there is a section on integrity constraints, to help motivate the interpretation in ALP of stimulus-response production rules as a species of integrity constraint, and a section on the agent language AgentSpeak, which can be regarded as an extension of production systems in which actions are generalised to plans of action.

The semantics and proof procedures of logic programming and ALP are based on classical logic, but are sufficiently different to be a potential source of difficulty for some readers. For this reason, I have put details about the minimal model semantics in an appendix. I have also included an appendix about the interpretation of natural language conditionals in logic programming terms, because it interrupts the main flow of the paper.

Before settling down to the main part of the paper, I first present a motivating example, discuss the Wason selection task, and present some historical background.

2 A motivating example

Consider the following real world example of a sign posted in carriages on the London underground:

Emergencies:
Press the alarm signal button to alert the driver.
The driver will stop if any part of the train is in a station.

If not, the train will continue to the next station, where help can more easily be given.
There is a 50 pound penalty for improper use.

The sign is intended to be read and for its contents to be assimilated by passengers traveling on the underground. For this purpose, a passenger needs to translate the English text from natural language into a mental representation. In AI, such mental representations include logic, logic programs, production rules and other procedural representations. Here is the main part of a translation of the London underground sign into the ALP agent representation that I will describe later in the paper:

> *You alert the driver that there is an emergency*
> *if you are on the underground and there is an emergency*
> *and you press the alarm signal button.*
>
> *The driver will stop the train immediately if you press the*
> *alarm signal button and any part of the train is in a station.*
>
> *The driver will stop the train (and help can more easily be*
> *given) at the next station if you press the alarm signal button*
> *and not any part of the train is in a station.*
>
> *You will be liable to a 50 pound penalty*
> *if you press the alarm signal button improperly.*

The first sentence of the translation shows how procedures are represented in logic programming form as conditionals that are used backwards to reduce goals that match the conclusion of the conditional to sub-goals that correspond to the conditions. The first, second and third sentences show that the conditional form of natural language sentences should not to be taken literally, but may need to include extra conditions (such as *you press the alarm signal button*) and extra qualifications (such as the driver will stop *the train*) implicit in the context. The fourth sentence shows that the meaning of a natural language sentence may be a conditional whether or not the surface structure of the sentence contains an explicit mark of a conditional.

However, the translation of the sign does not represent its full import, which depends upon the reader's additional background goals and beliefs. These might include such goals and beliefs as[55]:

> *If there is an emergency then you get help.*
> *You get help if you are on the underground and you alert the driver.*

I will argue later that the first sentence is a maintenance goal, which is a special case of an integrity constraint of the kind used to maintain integrity in active database systems. Maintenance goals are similar to plans in the intelligent agent programming language AgentSpeak. I will also argue that they generalize production rules in production systems.

In the ALP agent framework, goals and beliefs are embedded in an observation-thought-decision-action cycle. Given an observation of an emergency, the maintenance goal would be used to reason forward, to derive the achievement goal of getting help. The logic program would be used to reason backward, to reduce the goal of getting help to the action sub-goal of

[55] These two sentences also help to explain the significance of the phrase *where help can more easily be given*. The passenger's background knowledge may also contain other ways of getting help. Arguably, the phrase is intended to help the passenger to evaluate alternative candidate actions, in deciding what to.

pressing the alarm signal button. Assuming that there are no other candidate actions to consider, the agent would commit to the action and attempt to solve it by executing it in the environment.

3 The Selection Task

I will not attempt to analyse the Wason selection task and its variants (Wason, 1968) in detail, but will outline instead how the ALP agent framework addresses some of the problems that have been observed with human performance in psychological experiments.

Consider the following simplified abstraction of the selection task presented to a subject in natural language:

> *Given some incomplete information/observations,*
> *what conclusions can be derived using the conditional if P then Q?*

The task is given with a concrete instance of the conditional, such as *if there is a vowel on one side of a card then there is an even number on the other side* and *if a person is drinking alcohol in a bar then the person is over eighteen years old*. To solve the task, the subject first needs to translate the conditional into a mental representation. The subject then needs to use the mental representation of the conditional, possibly together with other background goals and beliefs, to derive conclusions from the observations. I will argue that most of the reasons for the variation in subjects' performance on the selection task can be demonstrated using the ALP agent model with this simplified formulation of the task.

In the ALP agent model, the subject needs first to decide whether the English language conditional should be understood as a belief or as a goal. If it is understood as a belief, then it is represented as a clause in a logic program. If it is understood as a goal, then it is represented as an integrity constraint, to monitor updates. These two alternatives, understanding the conditional as a belief or as a goal, correspond roughly to the descriptive and deontic interpretations of the conditional respectively. In concrete instances of the task, the syntactic form of the conditional, its semantic content, and the way in which the task is formulated can all influence the interpretation.

Suppose the subject interprets the conditional as a belief, which is then represented as a logic programming clause:

> *Q if P*

Assuming that there are no other clauses, including background clauses, that have the same conclusion *Q*, the clause is then interpreted as the *only* way of concluding *Q*. In the completion semantics (Clark, 1978) of logic programming, this can be expressed explicitly in the form:

> *Q if and only if P*

The equivalence here is asymmetric in the sense that the formula on the right hand side is taken to be the definition of the predicate on the left hand side of the equivalence, and not the other way around.

In the ALP agent model described in this paper, given an observation *O*, the agent can reason both forwards from *O,* to derive consequences of the observation, and backwards from *O*, to derive explanations of the observation. In the selection task, therefore, forward reasoning can be used to derive *Q* from an observation of *P*, and backward reasoning can be used to derive

P as an explanation of an observation of *Q*. These are the classic responses to Wason's original card version of the selection task.

The modus tollens derivation of *not P* from *not Q* is also possible, but more difficult, because it is first necessary to derive *not Q* from some initial positive observation *Q'*. This is because, in general, an agent's observations are represented only by atomic sentences. Negations of atomic sentences need to be derived from positive observations. In card versions of the selection task, for example, the conclusion that a card does not have a vowel on one of its sides needs to be derived by a chain of reasoning, say from an observation that it has the letter B on that side, to the conclusion that it has a consonant on that side, to the conclusion that it does not have a vowel on that side. As Sperber, Cara, and Girotto (1995) argue, the longer the derivation, and the greater the number of irrelevant, alternative derivations, the less likely it is that a subject will be able to perform the derivation.

In the ALP agent model, the relationship between positive and negative concepts, needed to derive a negative conclusion from a positive observation, is expressed in the form of integrity constraints, such as:

if mortal and immortal then false i.e.*not(mortal and immortal)*
if odd and even then false i.e.*not(odd and even)*
if vowel and consonant then false i.e.*not(vowel and consonant)*
if adult and minor then false i.e.*not(adult and minor)*
etc.

and more generally as:

if Q' and Q then false i.e.*not(Q' and Q)*

where *Q'* is the positive observation or some generalization of the observation (such as the card has a consonant on the face) and *if Q then false* is an alternative syntax for the negation *not Q*.

In the ALP agent model, given a concrete observation that leads by a chain of forward reasoning to the conclusion *Q'*, a further step of forward reasoning with the integrity constraint is needed to derive *not Q*. Backward reasoning with the conditional (replacing *Q* by its definition *P*) then derives *not P*.

Thus in card versions of the Wason selection task, when the conditional is interpreted as a belief, the derivations of *P* from *Q* and of *Q* from *P* are straight-forward, the derivation of *not P* from some contrary positive observation that implies *not Q* is difficult, but possible. But the derivation of *not Q* from *not P* is not possible at all, because of the asymmetry of the equivalence *Q if and only if P*.

Suppose instead that the subject interprets the conditional as a goal. In the ALP agent model, this is represented by an integrity constraint of the form:

if P then Q.

Such integrity constraints are used to reason forwards, like production rules, in this case to derive *Q* from *P*. They are not used backwards to derive *P* from *Q*, which is consistent with experimental data for deontic versions of the selection task.

As in the case where the conditional is interpreted as a belief, negative premises, such as *not P* and *not Q*, need to be derived by forward reasoning from positive atomic observations, using integrity constraints of such form as:

if Q' and Q then false
if P' and P then false

and the same considerations of computational complexity apply.

Assuming that the positive observations imply both *Q'* and *P'* and that both:

if Q then false
if P then false

have been derived, then the only further inference that is possible is:

From *if P then Q*
and *if Q then false*
derive *if P then false*
i.e. *not P.*

However, this inference step, which is both easy and natural for human subjects, is not possible in some existing ALP proof procedures. I will argue later that this is because these proof procedures implement the wrong semantics for integrity constraints. The semantics needed to justify this inference step is the *consistency view of integrity constraint satisfaction*. I will discuss this problem and its solution again in section 10.2. In the meanwhile, it is interesting to note that the selection task seems to suggest a direction for solving technical problems associated with proof procedures developed for practical applications in AI.

4 A personal view of the history of logic in AI

To put the issues dealt with in this paper into context, it may be helpful to consider their historical background.

4.1 Heuristic versus formal approaches

It is widely agreed that the most important work concerning logic in the early days of AI was the Logic Theorist (Newell et al, 1957), which managed to prove 38 of the first 52 theorems in Chapter 2 of Principia Mathematica. The Logic Theorist pioneered the heuristic approach, employing three inference rules, "backward reasoning", "forward chaining" and "backward chaining", without traditional logical concerns about semantics and completeness. In contrast with formal theorem-proving techniques, these heuristic inference rules behaved naturally and efficiently.

Logic Theorist led to GPS, a general problem solver, not directly associated with logic, and later to production systems. Production rules in production systems resembled conditionals in traditional logic, but did not suffer from the inscrutability and inefficiencies associated with formal theorem-proving.

In the meanwhile, McCarthy (1958) advocated a formal approach, developing the Advice Taker, using formal logic for knowledge representation and using theorem-proving for problem-solving. In the Advice-Taker approach, the theorem-prover was a "black box", and there was no attempt to relate the behaviour of the theorem-prover to common sense techniques of human problem-solving. The theorem-proving approach led to the development of question-answering systems, using complete and correct theorem-provers, based on mathematical logic.

Resolution (Robinson, 1965) was developed within the formal logic tradition, with mainly mathematical applications in mind. Its great virtue was its simplicity and uniformity, compressing the rules and logical axioms of symbolic logic into a single inference rule. It could not be easily understood in human-oriented terms, but was presented as a machine-oriented logic. Its early versions were very inefficient, partly because the resolution rule did not have an intuitive interpretation.

Towards the end of the 60s, there were two main trends among symbolic approaches to AI: the heuristic (sometimes called "scruffy" or "strong") approach, mainly associated with production systems, which behaved in human-intelligible terms, uninhibited by mathematical concerns about semantics, but emphasizing the importance of domain-specific knowledge; and the formal ("neat" or "weak") approach, exemplified by resolution-based systems, emphasizing domain-independent, general-purpose problem-solving. Production systems were beginning to have applications in expert systems, and resolution was beginning to have applications in mathematics and question-answering.

4.2 Procedural representations of knowledge and logic programming

In the meanwhile, critics of the formal approach, based mainly at MIT, began to advocate procedural representations of knowledge, as superior to declarative, logic-based representations. This led to the development of the knowledge representation and problem-solving languages Planner and micro-Planner. Winograd's PhD thesis (1971), using micro-Planner to implement a natural language dialogue for a simple blocks world, was a major milestone of this approach. Research in automated theorem-proving, mainly based on resolution, went into sharp decline.

The battlefield between the logic-based and procedural approaches moved briefly to Edinburgh during the summer of 1970 at one of the Machine Intelligence Workshops organized by Donald Michie (van Emden, 2006). At the workshop, Pappert and Sussman from MIT gave talks vigorously attacking the use logic in AI, but did not present a paper for the proceedings. This created turmoil among researchers in Edinburgh working in resolution theorem-proving. However, I was not convinced that the procedural approach was so different from the SL-resolution system I had been developing with Donald Kuehner (1971).

During the next couple of years, I tried to reimplement Winograd's system in resolution logic and collaborated on this with Alain Colmerauer in Marseille. This led to the procedural interpretation of Horn clauses (Kowalski 1973/1974) and to Colmerauer's development of the programming language Prolog. I also investigated other interpretations of resolution logic in problem solving terms (Kowalski 1974/1979), exploiting in particular the fact that all sentences in resolution logic can be expressed in conditional form.

However, at the time, I was not aware of the significance of production systems, and I did not understand their relationship with logic and logic programming. Nor did I appreciate that logic programming would become associated with general-purpose problem-solving, and that this would be misunderstood as meaning that logic programming was suitable only for representing "weak" general-purpose knowledge. I will come back to this problem later in the paper.

4.3 Non-monotonic reasoning, abduction, and argumentation

In the mid-1970s, Marvin Minsky (1974) launched another attack against logic from MIT, proposing frames as a representation of stereotypes. In contrast with formal logic, frames focussed on the representation of default knowledge, without the need to specify exceptions

strictly and precisely. This time the logic community in AI fought back with the development of various non-monotonic logics, including circumscription (McCarthy, 1980), default logic (Reiter, 1980), modal non-monotonic logic (McDermott and Doyle, 1980), autoepistemic logic (Moore, 1985) and negation as failure in logic programming (Clark, 1978).

(Poole et al, 1987) argued that default reasoning can be understood as a form of abductive reasoning. Building upon their results, Eshghi and I (1989) showed that negation as failure in logic programming can also be understood in abductive terms. More recently, Dung and his collaborators (Bondarenko et al 1997 ; Dung et al, 2006) have shown that most non-monotonic logics can be interpreted in terms of arguments and counter-arguments supported by abductive hypotheses. Moreover, Poole (1993, 1997) has shown that Baysian networks can be understood as abductive logic programs with assumptions that have associated probabilities.

4.4 Intelligent agents

Starting in the late 1980s and early 1990s, researchers working in the formal logic tradition began to embed logic-based representations and problem-solving systems in intelligent agent frameworks. The intelligent agent approach was given momentum by the textbook of Russell and Norvig (2003), whose first edition was published in 1995. In their book, Russell and Norvig credit Newell, Laird and Rosenblum (Newell, 1990; Laird *et al*., 1987) with developing SOAR as the "best-known example of a complete agent architecture in AI".

However, SOAR was based on production systems and did not have a logical basis. Among the earliest attempts to develop formal, logic-based agent architectures were those of Rao and Georgeff (1991) and Shoham (1991), both of which were formulated in BDI (Belief, Desire, Intention) terms (Bratman, Israel, and Pollack,1988) . Although their formal specifications were partly formulated in modal logics, their implementations were in procedural languages that looked like extensions of production systems. The ALP agent model (Kowalski and Sadri,1999; Kowalski 2001, 2006) was developed to reconcile the use of logic for agent specification with its use for implementation. Instead of using modalities to distinguish between beliefs and desires, it uses an extension of the database distinction between data and integrity constraints, treating beliefs like data and desires (or goals) like integrity constraints.

5 Production systems

Production systems were originally developed by the logician Emil Post as a mathematical model of computation, and later championed by Alan Newell as a model of human problem solving (Newell, 1973; Anderson and Bower, 1973). Production systems restrict knowledge representation to a combination of *facts*, which are atomic sentences, and *condition-action rules*, which have the syntax of conditionals, but arguably do not have a logical semantics. They also restrict reasoning to a kind of modus ponens, called *forward chaining.*

A typical *production system* (Simon, 1999) involves a "declarative" memory consisting of atomic sentences, and a collection of procedures, which are *condition-action rules* of the form:

If conditions C, then actions A.

Condition-action rules are also called *production rules*, just plain *rules*, or *if-then rules.* The most typical use of production rules is to implement stimulus-response associations. For example:

If it is raining and you have an umbrella, then cover yourself with the umbrella.

If you have an essay to write, then study late in the library.

The action part of a rule is often expressed as a command, or sometimes as a recommendation, to perform an action.

Production rules are executed by means of a cycle, which reads an input fact, uses *forward chaining* (from conditions to actions) to match the fact with one of the conditions of a production rule, verifies the remaining conditions of the rule, and then derives the actions of the rule as candidates to be executed. If more than one rule is "triggered" in this way, then *conflict-resolution* is performed to decide which actions should be executed. Inputs and actions can be internal operations or can come from and be performed upon an external environment.

Conflict resolution can be performed in many ways, depending upon the particular production system language. At one extreme, actions can be chosen purely at random. Or they can be determined by the order in which the rules are written, so that the first rule to be triggered is executed first. At the opposite extreme, actions can be chosen by means of a full scale decision-theoretic analysis, analysing the expected outcomes of actions, evaluating their utility and probability, and selecting one or more actions with highest expected utility.

Compared with classical logic, which has both a declarative, model-theoretic semantics and diverse proof procedures, production systems arguably have no declarative semantics at all. Production rules look like logical implications, without the semantics of implications. Moreover, forward chaining looks like *modus ponens*, but it is not truth-preserving, because there is no notion of what it means for a condition-action rule to be true.

The similarity of production rules to implications is a major source of confusion. Some authors (Thagard, 2005) maintain that production systems enable *backward reasoning*, from actions that are goals to conditions that are sub-goals. MYCIN (Shortliffe, 1976), for example, one of the earliest expert systems, used backward reasoning for medical diagnosis. MYCIN was and still is described as a production system, but is better understood as using an informal kind of logic.

When production rules are used to implement stimulus-response associations, they normally achieve unstated goals implicitly. Such implicit goals and the resulting goal-oriented behaviour are said to be *emergent*. For example, the emergent goal of covering yourself with an umbrella if it is raining is *to stay dry*. And the emergent goal of studying late in the library if you have an essay to write is *to complete the essay*, which is a sub-goal of *completing the course*, which is a sub-goal of *getting a degree*, which is a sub-goal of the top-level goal of *becoming a success in life.*

In contrast with systems that explicitly reduce goals to sub-goals, systems that implement stimulus-response associations are an attractive model of evolutionary theory. Their ultimate goal, which is to enable an agent to survive and prosper in competition with other agents, is emergent, rather than explicit.

Although the natural use of production systems is to implement stimulus-response associations, they are also used to simulate backward reasoning, not by executing production rules backwards, but by treating goals as facts and by forward chaining with rules of the form:

If goal G and conditions C then add H as a sub-goal.

Indeed, in ACT-R (Anderson and Bower, 1973), this is the typical form of production rules used to simulate human problem solving in such tasks as the Tower of Hanoi.

Thagard (2005) also draws attention to this use of production rules and claims that such rules cannot be represented in logical form. He gives the following example of such a rule:

> *If you want to go home for the weekend, and you have the bus fare,*
> *then you can catch a bus.*

Here the "action" *you can catch a bus* is a recommendation. The action can also be expressed as a command:

> *If you want to go home for the weekend, and you have the bus fare,*
> *then catch a bus.*

The use of the imperative voice to express actions motivates the terminology "conflict resolution" to describe the process of reconciling conflicting commands.

The differences between production systems and logic programming are of two kinds. There are technical differences, such as the fact that production rules are executed in the forward direction, the fact that conclusions of production rules are actions or recommendations for actions, and the fact that production rules are embedded in a cycle in which conflict resolution is performed to choose between candidate actions. And there are cultural differences, reflecting a more casual attitude to issues of correctness and completeness, as well as a greater emphasis on "strong", domain-specific knowledge in contrast to "weak" general-purpose problem-solving methods.

The technical differences are significant and need to be treated seriously. In the ALP agent model, in particular, we take account of forward chaining by using logic programs and integrity constraints to reason forwards; we take account of the imperative nature of production rules by treating them as goals rather than as beliefs; and we take account of the production system cycle by generalising it to an agent cycle, in which thinking involves the use of both goals and beliefs.

However, the cultural differences are not technically significant and are based upon a failure to distinguish between *knowledge representation* and *problem-solving*. Knowledge can be weak or strong, depending on whether it is general knowledge that applies to a wide class of problems in a given domain, like axioms of a mathematical theory, or whether it is specialised knowledge that is tuned to typical problems that arise in the domain, like theorems of a mathematical theory.

However, some strong knowledge may not be derivable from general-purpose knowledge in the way that theorems are derivable from axioms. Strong knowledge might be more like a mathematical conjecture or a rule of thumb, which is incomplete and only approximately correct. Strong knowledge can exist in domains, such as expert systems, where there exists no weak knowledge that can serve as a general axiomatisation.

Logic is normally associated with weak knowledge, like mathematical axioms; and production systems are associated with strong knowledge, as in expert systems. But there is no reason, other than a cultural one, why logic can not also be used to represent strong knowledge, as in the case of mathematical theorems or conjectures.

In contrast with knowledge, which can be weak or strong, problem-solving methods are generally weak and general-purpose. Even production systems, with their focus on strong knowledge, employ a weak and general-purpose problem-solving method, in their use of forward chaining. Logic programs similarly employ a weak and general-purpose method, namely backward reasoning.

6 Logic programming

6.1 A short introduction to logic programming

Logic programming has three main classes of application: as a general-purpose programming language, a database language, and a knowledge representation language in AI. As a programming language, it can represent and compute any computable function. As a database language, it generalises relational databases, to include general clauses in addition to facts. And as a knowledge representation language it is a non-monotonic logic, which can be used for default reasoning. Its most well-know implementation is the programming language Prolog, which combines pure logic programming with a number of impure features.

In addition to the use of logic programming as a normative model of problem solving (Kowalski, 1974/79), Stenning and van Lambalgen (2004, 2005, 2008) have begun to investigate its use as a descriptive model of human reasoning.

Logic programs (also called *normal logic programs*) are sets of conditionals of the form:

If B_1 *and … and* B_n *then* H

where the *conclusion* H is an atomic formula and the *conditions* B_i are *literals*, which are either atomic formulas or the negations of atomic formulas. All variables are implicitly universally quantified in front of the conditional. Conditionals in logic programs are also called *clauses*. *Horn clauses* [56] are the special case where all of the conditions are atomic formulae. *Facts* are the special case where $n = 0$ (there are no conditions) and there are no variables. Sometimes clauses that are not facts are also called *rules,* inviting confusion with production rules.

Goals (or *queries*) are conjunctions of literals, syntactically just like the conditions of clauses. However, all variables are implicitly existentially quantified, and the intention of the goal is to find an instantiation of the variables that makes the goal hold.

For example, the three sentences:

If you have the bus fare and you catch a bus
and not something goes wrong with the bus journey,
then you will go home for the weekend.

If you have the bus fare and you catch a bus,
then you will go home for the weekend.

You have the bus fare.

are a clause, a Horn clause and a fact respectively. Notice that the second sentence can be regarded as an imprecise version of the first sentence. Notice too that the first two clauses both express "strong" domain-specific knowledge, rather than the kind of weak knowledge that would be necessary for general-purpose planning.

The sentence *you will go home for the weekend* is a simple, atomic goal.

[56] Horn clauses are named after the logician Alfred Horn, who studied some of their model-theoretic properties.

Backward reasoning (from conclusions to conditions) treats conditionals as goal-reduction procedures:

to show/solve H, show/solve B_1 and … and B_n.

For example, backward reasoning turns the conditionals:

If you study late in the library then you will complete the essay.
If you have the bus fare and you catch a bus,
then you will go home for the weekend.

into the procedures:

To complete the essay, study late in the library.
To go home for the weekend, check that you have the bus fare, and catch a bus.

Because conditionals in normal logic programming are used only backwards, they are normally written backwards:

H if B_1 and … and B_n.

so that backward reasoning is equivalent to "forward chaining" in the direction in which the conditional is written. The Prolog syntax for clauses:

H :- B_1 , … , B_n.

is deliberately ambiguous, so that clauses can be read either declaratively as conditionals written backwards or procedurally as goal-reduction procedures executed forwards.

Whereas positive, atomic goals and sub-goals are solved by backward reasoning, negative goals and sub-goals of the form *not G,* where *G* is an atomic sentence, are solved by *negation as failure: not G* succeeds if and only if backward reasoning with the sub-goal *G* does not succeed.

Negation as failure makes logic programming a non-monotonic logic. For example, given only the clauses:

An object is red if the object looks red and not the object is illuminated by a red light.
The apple looks red.

then the consequence:

The apple is red.

follows as a goal, because there is no clause whose conclusion matches the sub-goal *the apple is illuminated by a red light* and therefore the two conditions for the only clause that can be used in solving the goal both hold. However, given the additional clause *the apple is illuminated by a red light*, the sub-goal now succeeds and the top-level goal now fails, non-monotonically withdrawing the consequence *The apple is red.* However, the consequence *not the apple is red* now succeeds instead.

Goals and conditions of clauses can be generalised from conjunctions of literals to arbitrary formulae of first-order logic. The simplest way to do so is to use auxiliary predicates and clauses (Lloyd and Topor, 1984). For example, the goal:

Show that for all exams,
if the exam is a final exam, then you can study for the exam in the library.

can be transformed into the normal logic programming form:

Show that not the library is useless.
the library is useless if the exam is a final exam and
not you can study for the exam in the library.

This is similar to the transformation[57] noted by Sperber, Cara, and Girotto (1995) needed to obtain classically correct reasoning with conditionals in variants of the Wason Selection Task. It is important to note that the transformation applies only when the conditional is interpreted as a goal, and not when it is interpreted as a clause.

The computational advantage of the transformation is that it reduces the problem of determining whether an arbitrary sentence of first-order logic holds with respect to a given logic program to the two simple inference rules of backward reasoning and negation as failure alone.

6.2 Strong versus weak methods in logic programming

In practice, expert logic programmers use both the declarative reading of clauses as conditionals, so that programs correctly achieve their goals, and the procedural reading, so that programs behave efficiently. However, it seems that few programmers achieve this level of expertise. Many programmers focus primarily on the declarative reading and are disappointed when their programs fail to run efficiently. Other programmers focus instead on the procedural reading and loose the benefits of the declarative reading.

Part of the problem is purely technical, because many declarative programs, like the clause:

Mary likes a person if the person likes Mary

reduce goals to similar sub-goals repeatedly without termination. This problem can be solved at least in part by employing more sophisticated ("stronger"), but still general-purpose problem-solvers that never try to solve similar sub-goals more than once (Sagonas et al, 1994).

However, part of the problem is also psychological and cultural, because many programmers use logic only to specify problem domains, and not to represent useful knowledge for efficient problem-solving in those domains.

The planning problem in AI is a typical example. In classical planning, the problem is specified by describing both an initial state and a goal state, and by specifying the preconditions and postconditions of atomic actions. To solve the problem, it is then necessary to find a sequence of atomic actions that transforms the initial state into the goal state. This is sometimes called *planning from first principles*, or by *brute force*. In many domains it is computationally explosive.

[57] The use of the auxiliary predicate is merely a technical device, which is useful for maintaining the simple syntax of goals and clauses. However, it is also possible to employ a more natural syntax in which the conditional is written directly in the form of a denial: *Show that not there is an exam, such that (the exam is a final exam and not you can study for the exam in the library).*

The alternative is to use a collection of precompiled plan schemata that apply to a wide class of commonly occurring problems in the given domain. These plan schemata typically reduce a top-level goal to high-level actions, rather than to atomic actions, in a hierarchy of reductions from higher-level to lower-level actions. This is sometimes called *planning from second principles*, and it can be very efficient, although it may not be as complete as planning from first principles.

Algorithms and their specifications are another example. For instance, the top-level specification of the problem of sorting a list can be expressed as:

list L' is a sorted version of list L if L' is a permutation of L and L' is ordered.

No "weak" general-purpose problem-solving method can execute this specification efficiently.
The Prolog implementation of backward reasoning, in particular, treats the specification as a procedure:

To sort list L obtaining list L', generate permutations L' of L,
until you find a permuation L' that is ordered.

However, there exist a great variety of efficient sorting algorithms, which can also be expressed in logic programming form. For example, the top-level of the recursive part of quicksort:

To quicksort a list L into L', split L into two smaller lists L_1 & L_2 and
quicksort L_1 into L'_1 and quicksort L_2 into L'_2 and shuffle L_1 & L_2 together into L'.

I leave the Horn clause representation of the procedure to the reader. Suffice it to say that quicksort is a sufficiently "strong" problem-solving method that it behaves efficiently even when executed by a weak problem-solving method, such as backward reasoning. Stronger, but still general-purpose methods, such as parallelism, can improve efficiency even more.

6.3 Two kinds of semantics

There are two main kinds of semantics for logic programs, with two main ways of understanding what it means for a goal to hold. Both of these semantics interpret the clauses of a program as the *only* clauses that can be used to establish the conclusions of clauses. This is also known as the *closed world assumption* (Reiter, 1978).

In the *minimal model semantics*, a logic program defines a set of minimal models, and a goal holds if it is *true* in one or more of these models. The minimal model semantics comes in a *credulous* version, in which a goal holds if it is true in *some* minimal model; and a *sceptical* version, in which a goal holds if it is true in *all* minimal models.

The notion of minimal model is very simple in the case of Horn clause programs, because every Horn clause program has a unique minimal model, in which an atomic sentence is true if and only if it is true in all models. However, it is more complicated for normal logic programs with negative conditions. For this reason, a more detailed discussion of the minimal model semantics is given in the appendix.

The main (and arguably simpler) alternative to the minimal model semantics is the *completion semantics*, in which clauses are understood elliptically as the if-halves of definitions in if-and-only-if form (Clark, 1978). If a predicate does not occur as the conclusion of a clause, then it is deemed to be *false.* This interpretation is called the *predicate*

(or Clark) completion. A goal holds in the completion semantics if it is a *theorem*, logically entailed by the completion of the logic program.

Consider, for example, the logic program:

you get help if you press the alarm signal button.
you get help if you shout loudly.

The completion[58] of the program is:

you get help if and only if you press the alarm signal button or you shout loudly.
you press the alarm signal button if and only if false.
(i.e. *not you press the alarm signal button.*)
you shout loudly if and only if false. (i.e. *not you shout loudly.*)

which logically entails the conclusion: *not you get help*.

Negation as *finite* failure is *sound* with respect to the completion semantics:

not G is logically entailed by the completion
if *G fails finitely*.

It is also complete in many cases.

The completion and minimal model semantics have different notions of what it means for a goal, which in both cases can be any formula of first-order logic, to hold with respect to a given logic program. In both semantics, the same proof procedures of backward reasoning and negation as failure can be used. However, in the minimal model semantics, negation as failure includes possibly infinite failure, whereas in the completion semantics, negation as failure is finite.

Given a program and goal, in most cases these two semantics give the same results. In many cases, these results differ from those sanctioned by classical logic, but are similar to those observed in psychological studies of human reasoning. However, as we have seen at the beginning of the paper, the selection task presents a challenge, which we will discuss again in section 10.2.

6.4 Two kinds of proof procedures

The standard proof procedure for logic programming, used in Prolog for example, generates a *search tree*, whose *nodes* are labelled by goals. The *root* of the tree is labelled by the initial goal. Given any node in the tree, labelled by a goal, say:

$B \& B_1 \& ... \& B_n$

backward reasoning proceeds by first selecting a sub-goal for solution, say *B* for simplicity.

If there is more than one sub-goal that can be selected, then, as with conflict-resolution in production systems, a decision needs to be made, selecting only one. Any selection strategy can be used. Prolog selects sub-goals in the order in which they are written. However, more intelligent, but still "weak", problem-solving strategies, such as choosing the most

[58] The use of *if and only if* here does not make the two sides of the "equivalence" symmetric. The left-hand side is a predicate defined by the formula on the right-hand side.

constrained sub-goal first (the one that has fewest candidate solutions), can also be used (Kowalski, 1974/79).

If the selected sub-goal *B* is a positive, atomic goal, then backward reasoning continues by looking for a clause whose conclusion matches *B*. In the propositional case, if *B* is identical to the conclusion of a clause *B if* B'_1 *&...* $\&$ B'_m then *backward reasoning* replaces the selected sub-goal by the conditions of the clause, obtaining a *child* of the node labelled by the new goal:

$B'_1 \&... \& B'_m \& B_1 \&... \& B_n$

In the more general case, the selected sub-goal needs to be unified with the conclusion of a clause, and the unifying substitution is applied to the new goal. A node has as many children as there are such clauses whose conclusion matches the selected sub-goal.

If the selected sub-goal *B* is the negation *not B'* of an atomic formula *B'* containing no variables, then *negation as failure* is applied, generating a subsidiary search tree with the same program, but with the initial goal *B'*. The selected sub-goal *not B'* succeeds if and only if the subsidiary search tree contains no solution, and the *only child* of the node is labelled by the new goal:

$B_1 \&... \& B_n$

A *solution* is a finite branch of the search tree that starts from the initial node and ends with a node labelled by the empty conjunction of sub-goals (where $n = 0$). Because of infinite branches, the proof procedure is semi-decidable.

Given a program and an initial goal, the search tree is generated and explored, to find a solution. Prolog uses a depth-first search strategy, exploring branches in the order in which clauses are written. Depth-first search is risky, because it can get lost down an infinite branch, when there are solutions lying around on alternative finite branches. However, when it works, it works very efficiently. But other search strategies, such as breadth-first, best-first and parallel search strategies, which are guaranteed to find a solution if one exists, are also possible.

If a search tree is too big, no search strategy can search it efficiently. Sometimes the problem is inherently difficult, and there is no alternative but to search the space as patiently as possible. But in many domains, as we have seen, the problem can be reformulated, using "stronger" knowledge, which results in a smaller, more manageable search tree.

The alternative to generating a search tree is to reason explicitly with the program completion, representing the current state of the search by a single formula, which corresponds to the expanding frontier of the search tree. Reasoning with the completion was introduced as a proof procedure for abductive logic programming by (Console et al, 1991) and extended by (Fung and Kowalski, 1997).

Backward reasoning is performed by replacing predicates with their definitions, resulting in a logically equivalent formula. For example:

Goal: *you get help and you study late in the library*

Program: *you get help if and only if*
you press the alarm signal button or you shout loudly.

New goal: *[you press the alarm signal button and you study late in the library] or*

[you shout loudly and you study late in the library].

Reasoning with the completion represents the current state of the search to solve a goal as a disjunction of conjunctions of literals, say:

$(A \& A_1 \& ... \& A_\alpha)$ *or* $(B \& B_1 \& ... \& B_\beta)$ *or...or* $(C \& C_1 \& ... \& C_\nu)$.

Each disjunct corresponds to the goal at the end of an unextended branch in a search tree. The entire disjunction corresponds to the set of all goals at the current frontier of the search tree. The initial goal is represented by an initial disjunction with only one disjunct. Replacing an atom by its definition, called *unfolding,* corresponds to *replacing* a node at the tip of a branch by *all* of its children.

In the propositional case, unfolding a selected positive goal atom, say *B,* replaces it with its definition:

B if and only if D_1 *or...or* D_n

and distributes conjunction over disjunction, so that the new state of the search is represented by a new, logically equivalent formula of the form:

$(A \& A_1 \& ... \& A_\alpha)$ *or* $(D_1 \& B_1 \& ... \& B_\beta)$ *or…or* $(D_n \& B_1 \& ... \& B_\beta)$ *or...or* $(C \& C_1 \& ... \& C_\nu)$.

In the special case where the definition has the form:

B if and only if true

then the new formula is simplified to:

$(A \& A_1 \& ... \& A_\alpha)$ *or* $(B_1 \& ... \& B_\beta)$ *or...or* $(C \& C_1 \& ... \& C_\nu)$.

In the special case where the definition has the form:

B if and only if false

then the new formula is simplified to:

$(A \& A_1 \& ... \& A_\alpha)$ *or ...or* $(C \& C_1 \& ... \& C_\nu)$.

In the propositional case, the initial goal *succeeds* if and only if the formula *true* is derived by means of a finite number of such equivalence-preserving inference steps. The goal *fails finitely* if and only if the formula *false* is derived[59].

If the selected atom is a negative literal, say *not B'*, then a subsidiary derivation is initiated with the initial goal *B'* (viewed as a disjunction with only one disjunct, which is a conjunct with only one conjunct). If the new initial goal fails finitely, then the selected atom is replaced by *true* and the new formula is simplified to:

$(A \& A_1 \& ... \& A_\alpha)$ *or* $(B_1 \& ... \& B_\beta)$ *or...or* $(C \& C_1 \& ... \& C_\nu)$.

[59] In the general case, the process of matching goals with conclusions of definitions introduces equalities, which represent the instantiations of variables need to solve those goals. The initial goal succeeds if and only if a consistent set of equalities is derived.

If the initial goal succeeds, then the selected atom is replaced by *false* and the new formula is simplified to:

(A & A_1 &...& A_α) or ...or (C & C_1 & ... & C_ν).

For example, consider the problem of determining whether the consequence:

The apple is red.

follows from the program:

An object is red if the object looks red and not the object is illuminated by a red light.
The apple looks red.

In the completion semantics this can be represented in the form[60]:

$Goal_1$: *The apple is red.*
Program: *An object is red if and only if*
the object looks red and not the object is illuminated by a red light.
The apple looks red if and only if true
The apple is illuminated by a red light if and only if false

Using the completion, it is possible to derive the following sequence of formulae:

$Goal_2$: *The apple looks red and not the apple is illuminated by a red light.*
$Goal_3$: *not the apple is illuminated by a red light.*

Subsidiary $goal_1$: *the apple is illuminated by a red light.*
Subsidiary $goal_2$: *false*

$Goal_4$: *true*

The example shows that reasoning with the if-halves of clauses and generating a search tree simulates reasoning explicitly with the completion. Indeed, reasoning informally it can be hard to tell the two proof procedures apart.

7 The relationship between logic programs and production rules.

Viewing logic programs and production rules informally, it can also be hard to tell them apart, because in many cases they generate the same goal-reduction behaviour. Indeed, Simon (1999) includes Prolog, along with ACT-R, "among the production systems widely used in cognitive simulation". Stenning and van Lambalgen (2004, 2005, 2008) also observe that forward chaining with production rules of the form:

If goal G and conditions C then add H as a sub-goal.

behaves the same as backward reasoning with clauses of the form:

G if C and H.

[60] This is not the real completion, which makes the stronger closed world assumption that nothing is illuminated by a red light. In fact, it would be more appropriate to represent the predicate *is illuminated by a red light* by an abducible predicate, to which the closed world assumption does not apply. The form of the completion used in this example has been chosen as a compromise, to simplify the example.

The production rule and the clause both behave as the same goal-reduction procedure:

To achieve G, show C, and achieve H as sub-goal.

The production rule can be understood as representing the procedure *subjectively* (or *intentionally*), in terms of the causal effect of the goal *G* on an agent's state of mind. The clause, on the other hand, views it *objectively* (or *extensionally*), in terms of causal effects in the agent's environment. However, in both cases, the direction of the conditional is from conditions that are causes to conclusions that are effects. This switching of the direction of the conditional is related to the switching associated with *evidential conditionals*, *epistemic* and *inferencial conditionals* as noted by Pearl (1988), Dancygier (1998) and Politzer and Bonnefon (2006). In Appendix B, I argue that the completion semantics helps to explain the switching and truncation of conditionals observed in many psychological studies, as discussed by Politzer and Bonnefon (2006)

Thagard's example of the production rule:

If you want to go home for the weekend, and you have the bus fare,
then you can catch a bus.

can be understood as a subjective representation of the goal-reduction procedure:

To go home, check that you have the bus fare, and catch a bus.

which can be represented objectively by the logic programming clause:

you will go home for the weekend if
you have the bus fare and you catch a bus.

Notice that the causal relationship in both formulations of the procedure has as an implicit associated degree of uncertainty. In the case of the production rule, the uncertainty is associated with whether or not the agent's conflict resolution strategy will chose the action of the rule if other rules with other incompatible actions are triggered at the same time.

In the case of the clause, the uncertainty is associated with whether or not other implicit and unstated conditions also hold. This implicit condition can be stated explicitly using negation as failure:

you will go home for the weekend if
you have the bus fare and you catch a bus and
not something goes wrong with the bus journey.

The formulation without this extra condition can be regarded as an approximation to the fully specified clause. It is an approximation, not only because it is missing the condition, but also because it is not sufficient to take any old bus to get home, but it is necessary to take a bus that is on a route that goes home.

This representation with an extra condition, using negation as failure, does not attempt to quantify or even to qualify the associated degree of uncertainty. However, as we will see later, in abductive logic programming (ALP) it can be quantified by using instead an abducible predicate, say *the bus journey is successful,* and by associating a probability with the abducible predicate. For example:

you will go home for the weekend if
you have the bus fare and you catch a bus and
the bus journey is successful.

the bus journey is successful (with probability .95).
something goes wrong with the bus journey (with probability .05).

In the meanwhile, it is worth noting that the ALP formulation provides a link with Bayesian networks (Pearl, 1988). David Poole (1993, 1997) has shown, in general, that assigning a probability *p* to a conditional *A if B* is equivalent to assigning the same probability *p* to an abducible predicate, say *normal*, in the clause *A if B and normal.* In particular, he has shown that abductive logic programs with such probabilistic abducible predicates have the expressive power of discreet Bayesian networks. A similar result, but with a focus on using statistical learning techniques to induce logic programs from examples, has also been obtained by Taisuke Satoh (1995). In recent years this has given rise to a booming research area combining probability, learning and logic programming (De Raedt and Kersting, 2003).

The relationship between logic programs that are used to reason backwards and production rules that are used to represent goal-reduction procedures does not solve the problem of the relationship between logic programs and production systems in general. In particular, it does not address the problem of understanding the relationship when production rules are used instead to represent stimulus-response associations. For example, it is unnatural to interpret the condition-action rule:

If it is raining and you have an umbrella, then cover yourself with the umbrella.

as a goal-reduction procedure, to solve the goal of covering yourself with an umbrella by getting an umbrella when it rains.

I will argue that stimulus-response production rules are better understood as *maintenance goals*, which are integrity constraints in abductive logic programming.

8 Integrity constraints

In conventional databases, integrity constraints are used passively to prevent prohibited updates. But in abductive logic programming, they are like integrity constraints in active databases, where they both prevent prohibited updates and perform corrective actions to ensure that integrity is maintained. Production rules, used as stimulus-response associations, can be understood as integrity constraints of this kind.

Thus the condition-action rule for covering oneself when it rains can be understood as an active integrity constraint:

If it is raining and you have an umbrella, then you cover yourself with the umbrella.

This is identical in syntax to the condition-action rule, except that the action part is not expressed as a command or recommendation, but as a statement in declarative form. Unlike the original condition-action rule, the integrity constraint has the syntax of a logical implication.

In fact, integrity constraints are just a particular species of goal; and, like other kinds of goals, they can be sentences of first-order logic. They also have the same semantics as goals in logic programs (Godfrey *et al*, 1998). This includes both the credulous and sceptical versions of the minimal model semantics, in which integrity constraints are sentences that are

true in minimal models of a logic program. It also includes the completion semantics, in which they are sentences logically entailed by the if-and-only-if completion of the program.

However, it is also natural to understand the semantics of integrity constraints in consistency terms: An integrity constraint holds if and only if it is *consistent* with respect to the program. The consistency view comes in two flavours: In the minimal model semantics, the consistency view is equivalent to the credulous semantics: An integrity constraint is consistent with respect to the minimal model semantics of a logic program if and only if it is true in some minimal model of the program. In the completion semantics, an integrity constraint is satisfied if and only if it is consistent with the completion of the program relative to classical model theory. We will see later in section 10.2 that the consistency view of constraint satisfaction seems to be necessary in the selection task.

The difference between integrity constraints and conventional goals is partly terminological and partly pragmatic. In problem-solving, a conventional goal is typically an *achievement goal*, which is a one-off problem to be solved, including the problem of achieving some desired state of the world. In contrast, a typical integrity constraint is a *maintenance goal* that persists over all states of the world. In the context of intelligent agents, the term *goal* generally includes both *maintenance goals* and *achievement goals*.

Maintenance goals typically have the form of conditionals and can be hard to distinguish from clauses[61]. The distinction between clauses and maintenance goals can be better understood by viewing them in terms of the data base distinction between data and integrity constraints (Nicolas and Gallaire, 1978).

In conventional relational databases, data is defined by relationships, which can be viewed in logical terms as facts (variable-free atomic sentences). For example:

The bus leaves at time 9:00.
The bus leaves at time 10:00. etc.

However, in deductive databases, relationships can also be defined by more general clauses. For example:

The bus leaves at time X:00 if X is an integer and $9 \leq X \leq 18$.

Given appropriate definitions for the predicates in its conditions, the clause replaces the 10 separate facts that would be needed in a conventional relational database.

Compare this clause with the conditional:

If the bus leaves at time X:00,
then for some integer Y, the bus arrives at its destination at time X:Y & $20 \leq Y \leq 30$.

The existential quantifier in the conclusion of the sentence means that the sentence cannot be used to define data, but can only be used to constrain it. In a passive database, it would be used to reject any update that records an arrival time earlier than 20 minutes or later than 30 minutes after departure. However, in an active database, it could attempt to make its conclusion true and self-update the database with a record of the arrival, generating an appropriate value for Y.

[61] In this respect, they resemble the conditionals in the selection task, which can be interpreted descriptively or deontically.

Obviously, the capability to make such an update lies outside the powers of even an active database. However, it is a feature of many intelligent agent systems, such as Agent0 (Shoham, 1991). Whereas actions in production systems can be executed as soon as all of their conditions are satisfied, actions in Agent0 have associated times, and they can be executed when their associated times equal the actual time.

An intelligent agent might use integrity constraints to maintain the integrity of its "knowledge base", in much the same way that an active database system uses integrity constraints to maintain the integrity of its data. However, whereas backward reasoning is adequate for solving achievement goals, maintaining integrity is more naturally performed by combining backward and forward reasoning (Kowalski, Sadri and Soper, 1987):

> Integrity checking is triggered by an observation that updates the agent's knowledge base. Forward reasoning is used to match the observation either with a condition of an integrity constraint ***I*** or with a condition of a clause ***C***. It proceeds by using backward reasoning to verify any remaining conditions of the integrity constraint ***I*** or clause ***C***; and then, if these condtions are satisfied, it derives the conclusion. If the derived conclusion is the conclusion of an integrity constraint ***I***, then the conclusion is a new goal, typically an achievement goal. However, if it is the conclusion of a clause ***C***, then it is treated as a derived update and processed by further forward reasoning.

An achievement goal can be solved by backward reasoning, as in normal logic programming. But, instead of failing if a sub-goal cannot be reduced to further sub-goals, if the sub-goal is an action, then an attempt can be made to make it true by executing it successfully[62]. The result of the attempt is added as a new update to the knowledge base.

This informal description of forward and backward reasoning with clauses and forward reasoning with integrity constraints is compatible with different semantics and can be formalised both in terms of search trees and in terms of reasoning with the completion. In both cases, it gives a logical semantics to production rules of the stimulus-response variety and it facilitates combining them with logic programs. It also facilitates the modelling of agents that combine the ability to plan pro-actively with the ability to behave reactively.

9 Intelligent Agents and AgentSpeak

The ability to combine planning for a future state of the world with reacting to the present state is a characteristic of both human and artificial agents. In the next section, we will see how these abilities are combined in ALP agents. ALP agents can be viewed in BDI (Belief, Desire, Intention) (Bratman, Israel, and Pollack, 1988) terms, as agents whose beliefs are represented by clauses, whose desires (or goals) are represented by integrity constraints, and whose intentions are collections of actions to be performed in the future.

Arguably, the most influential of the BDI agent models is that of Rao and Georgeff (1991), its successor dMARS (d'Inverno, 1998), and their abstraction and simplification AgentSpeak (Rao, 1996). Although the earliest of these systems were specified in multi-modal logics, their procedural implementations bore little resemblance to their logical specifications. AgentSpeak abandoned the attempt to relate the modal logic specifications with their procedural implementations, observing instead that "…one can view agent programs as multi-threaded interruptible logic programming clauses"

[62] If the abducible sub-goal is an observable predicate, then the agent can actively attempt to observe whether it is true or false. Active observations turn the simplified formulation of the selection task discussed at the beginning of the paper into one that more closely resembles standard formulations, in which the subject is asked to perform an action.

However, this view of AgentSpeak in logic programming terms is restricted to the procedural interpretation of clauses. In fact, procedures in AgentSpeak are much closer to production rules than they are to clauses in logic programming. Like production systems, AgentSpeak programs have both a declarative and a procedural component. The declarative component contains both *belief literals* (atoms and negations of atoms) and *goal atoms*, whereas the procedural component consists of *plans*, which are an extension of production rules. Plans are embedded in a cycle similar to the production system cycle, and have the syntactic form:

Event E: conditions C ⇐ goals G and actions A.

Here the event *E* can be the addition or the deletion of a belief or the addition of a goal. Like production rules, plans are executed in the direction in which they are written, by a kind of forward chaining. Production systems are the special case where the set of goals *G* is empty.

Here are some typical AgentSpeak plans:

+!quench_thirst: have_glass ⇐
!have_soft_drink;
fill_glass, drink.

+!have_soft_drink: soft_drink_in_fridge ⇐
open_fridge;
get_soft_drink.

+rock_seen(R): not battery_low ⇐
?location(R,L);
!move_to(L);
pick_up(R).

+ there is a fire: true ⇐ +there is an emergency.

Observations and actions do not have associated times, and the declarative memory provides only a snapshot of the current state of the world. To compensate for this lack of a temporal representation, the prefixes *+,-, !,* and *?* are used to stand for *add, delete, achieve,* and *test* respectively.

Notice that the first two plans are goal-reduction rules, the third plan is a stimulus-response rule, and the fourth plan behaves like a logical implication used to reason forwards.

Like rules in production systems, plans in AgentSpeak do not have a declarative reading. However, in the special case where the triggering event *E* is the addition of a goal, and the plan has the form:

Goal E: conditions C ⇐ goals G and actions A.

the plan can be reformulated as a corresponding logic programming clause:

E' if C' and G' and A'and temporal constraints.

where the prefixed predicates of AgentSpeak are replaced by predicates with explicit associated times. The corresponding clause is identical in behaviour to the plan, but also has

a declarative reading. Using an (overly) simple, explicit representation of time[63], the clauses corresponding to the first two plans illustrated above are:

quench_thirst at time T_5 if have_glass at time T_1 and
have_soft_drink at time T_2 and
fill_glass at time T_3 and drink at time T_4 and $T_1<T_2<T_3<T_4<T_5$.

have_soft_drink at time T_4 if soft_drink_in_fridge at time T_1 and
open_fridge at time T_2 and
get_soft_drink at time T_3 and $T_1<T_2<T_3<T_4$.

As in the case of stimulus-response production rules, plans in which the triggering event is the addition or deletion of a belief can be represented by means of an integrity constraint. For example, the integrity constraint corresponding to the third plan above is:

If rock_seen(R) at time T and not battery_low at time T and location(R,L) at time T then move_to(L) at time T+1 and pick_up(R) at time T+2.

In addition to goal-reduction rules and stimulus-response associations, there is a third kind of production rule and AgentSpeak plan, illustrated by the fourth plan above, in which the event *E* is an observation and the goals and actions are simply the addition of beliefs. The fourth plan corresponds to a logic programming clause:

there is an emergency if there is a fire.

that is used to reason forwards in abductive logic programming, rather than backwards in normal logic programming, as we shall see again below.

10 Abductive logic programming (ALP)

Abductive logic programming (Kakas et al, 1998) extends normal logic programming by combining closed predicates that are defined in the conclusions of clauses with abducible (open or undefined) predicates that occur in the conditions, but not in the conclusions of clauses. Abducible predicates are instead constrained, directly or indirectly, by integrity constraints.

10.1 A brief introduction to ALP

In general, given a logic program ***P*** and integrity constraints ***I***, the *abductive task* is given by a problem ***G***, which is either an achievement goal or an observation to be explained. The task is solved by finding a set Δ of atomic sentences in the abducible predicates, such that:

Both ***I*** and ***G*** hold with respect to the extended, normal logic program $\boldsymbol{P} \cup \Delta$.

Δ is called an *abductive explanation* of ***G.***

This characterisation of abductive logic programming is compatible with different semantics[64] defining what it means for a goal or integrity constraint to hold with respect to a normal logic program. It is compatible, in particular, with the minimal model semantics, the

[63] Notice that the explicit representation of time draws attention to implicit assumptions of persistence, for example the assumption that *have_glass at time T_1* persists until time T_5.
[64] Sometimes the set Δ is allowed to contain the negations of atomic sentences in the abducible predicates, in which case there is an additional requirement that Δ be consistent.

completion semantics and the consistency view of constraint satisfaction. It is also compatible with different proof procedures, including both search tree proof procedures and proof procedures that reason explicitly with the completion.

Consider the following abductive logic program (with a much simplified representation of time), in which the integrity constraint is a denial:

Program: *Grass is wet if it rained.*
Grass is wet if the sprinkler was on.

Abducible predicates: *it rained, the sprinkler was on*, *the sun was shining.*

Integrity constraint: *not (it rained and the sun was shining*) or equivalently
if it rained and the sun was shining then false.

Observation: *Grass is wet.*

In abductive logic programming, reasoning backwards from the observation (treated as a goal), it is possible to derive two alternative explanations of the observation:

it rained
or the sprinkler was on.

Depending on the proof procedure, the two explanations are derived either on different branches of a search tree or as a single formula with an explicit disjunction.

Suppose that we are now given the additional observation:

Observation: *the sun was shining.*

Backward reasoning with the new observation is not possible. But, depending on the proof procedure, as we will see in the next section, the new observation and the integrity constraint reject the hypothesis *it rained*, leaving *the sprinkler was on* as the only acceptable explanation of the observation that the *Grass is wet.*

10.2 Proof procedures for ALP

The standard proof procedure for ALP augments the search tree proof procedure for normal logic programming in two ways. First, it includes all integrity constraints in the initial goal, expressed in the form of denials. Second it accumulates abductive hypotheses as they are needed to solve the initial goal, and it adds these to the program as the search to find a solution continues.

For example, the initial goal, at the root of the search tree, needed to explain the observation that the *Grass is wet* in the previous section, would be expressed in the form[65]:

Grass is wet and not (it rained and the sun was shining)

Selecting the first sub-goal *Grass is wet*, backward reasoning generates two alternative branches in the search tree, with two children of the root node:

[65] Strictly speaking, the denial should be formulated as a negative literal, by introducing an auxiliary predicate, but this is just a technical requirement, which can be avoided by slightly complicating the syntax of conditions of clauses, as in this example.

it rained and not (it rained and the sun was shining)
the sprinkler was on and not (it rained and the sun was shining)

On each branch, the proof procedure solves the abducible sub-goal by adding it to the set of hypotheses *Δ*:

not (it rained and the sun was shining)Δ = {it rained}
not (it rained and the sun was shining)Δ = {the sprinkler was on}

On both branches, negation as failure is applied to the remaining negative sub-goal, generating subsidiary search trees with new initial goals, and with the program augmented by the hypotheses generated so far:

it rained and the sun was shining Δ = {it rained}
it rained and the sun was shining Δ = {the sprinkler was on}

Given no additional information, both subsidiary search trees fail to contain a solution (in both cases *the sun was shining* fails). So the negative goals on the two branches of the original search tree succeed, resulting in two alternative solutions of the initial goal:

Δ = {it rained} *Δ = {the sprinkler was on}*

However, given the additional observation *the sun was shining* added to the program, the first subsidiary search tree, with the goal *it rained and the sun was shining*, now contains a solution, because both sub-goals now succeed, and the corresponding negative goal now fails, leaving only the one solution to the initial goal:

Δ = {the sprinkler was on}

Although the search tree proof procedure can be extended to include forward reasoning with clauses and integrity constraints, it is easier to incorporate forward reasoning into completion proof procedures instead. There are different proof procedures that use the completion explicitly.
For example, the proof procedure of Console et al (1991) represents integrity constraints as denials and uses the consistency view of constraint satisfaction, whereas the IFF proof procedure of Fung and Kowalski (1997) and the SLP proof procedure of Kowalski, R., Toni, F. and Wetzel, G. (1998) represents integrity constraints as implications. The IFF and SLP proof procedures differ mainly in their view of constraint satisfaction. IFF uses the theorem-hood view, and SLP uses the consistency view.

Given the problem of explaining the observation that the *Grass is wet,* the IFF and SLP proof procedures behave identically. The initial goal and program are expressed in the form:

$Goal_1$: *Grass is wet and (if it rained and the sun was shining then false)*
Program: *Grass is wet if and only if it rained or the sprinkler was on.*

Unfolding the first sub-goal, replacing it by its definition and distributing conjunction over disjunction, we obtain the equivalent new goal:

$Goal_2$: *[it rained and (if it rained and the sun was shining then false)] or*
[the sprinkler was on and (if it rained and the sun was shining then false)]

Forward reasoning within the first disjunct of the new goal adds the conclusion *if the sun was shining then false* to the disjunct:

$Goal_3$: *[it rained and (if it rained and the sun was shining then false) and (if the sun was shining then false)] or [the sprinkler was on and (if it rained and the sun was shining then false)]*

The proof procedure now terminates, because there are no further inferences that can be performed. It is a property of the IFF and SLP proof procedures that both of the hypotheses *it rained* and *the sprinkler was on* entail the observation, and the integrity constraint is both entailed and is consistent with the completion of the program extended with the completion of the hypotheses.

If the new observation *the sun was shining* is now added to both disjuncts of the current state of the goal, we obtain:

$Goal_4$: *[it rained and (if it rained and the sun was shining then false) and (if the sun was shining then false) and the sun was shining] or [the sprinkler was on and (if it rained and the sun was shining then false) and the sun was shining]*

Forward reasoning with the new observation in both disjuncts, adds *false* to the first disjunct and *if it rained then false* to the second disjunct. Simplifying the first disjunct to *false* leaves only the second disjunct in the form:

$Goal_5$: *[the sprinkler was on and (if it rained and the sun was shining then false) and the sun was shining and (if it rained then false)]*

The proof procedure terminates again. This time, the hypothesis that *the sprinkler was on* is now the only hypothesis that entails the observation and satisfies the integrity constraint. In this example, the same proof procedure implements both the theorem-hood and the consistency views of constraint satisfaction.

However, this inference can be added naturally and with little extra work, by extending the forward reasoning procedure with a refutation complete resolution method, such as SL-resolution (Kowalski and Kuehner, 1971). SL resolution generalises the SLD proof procedure (Kowalski, 1973/1974), on which the search tree proof procedure for Horn clauses is based.

The IFF proof procedure is sound and, with certain modest restrictions on the form of clauses and integrity constraints, complete with respect to the completion of the program in the Kunen (1987) three-valued semantics. The extension of the proof procedure to implement the consistency view of integrity satisfaction can not be complete, because consistency is not semi-decidable. However, this is not an argument against the consistency view - any more than the fact that there is no proof procedure for proving all truths of arithmetic is not an argument against the notion of truth in arithmetic. (See, in particular, the Horn clause program for arithmetic in Appendix A.)

The IFF and SLP completion proof procedures, with or without the extension to implement the consistency view of constraint satisfaction, differ from the search tree proof procedure mainly in their incorporation of forward reasoning to reason with integrity constraints. The informal proof procedure, which extends IFF and SLP by incorporating the integrity checking method of Kowalski, Sadri and Soper (1987) to include forward reasoning with clauses, is necessary to obtain the full effect of forward chaining with production rules.

11 ALP agents

The notion of ALP agent, in which abductive logic programming is embedded as the thinking component of an observation-thought-decision-action cycle, was introduced in (Kowalski, 1995) and was developed further in (Kowalski and Sadri, 1999; Kowalski 2001, 2006). It is also the basis of the KGP (Knowledge, Goals and Plans) agent model of (Kakas el al, 2004). The ALP proof procedure of (Kowalski and Sadri, 1999) is the IFF proof procedure, whereas the informal proof procedure described in this paper and in (Kowalski 2001, 2006) extends IFF by incorporating forward reasoning with clauses. The proof procedure of KGP extends IFF with constraint handling procedures.

The observation-thought-decision-action cycle of an ALP agent is similar to the production system and AgentSpeak cycles. The agent's beliefs are represented by clauses. Achievement and maintenance goals, as well as prohibitions, are represented by integrity constraints. Observations (or hypotheses that explain observations) and actions are represented by abducible predicates.

In the ALP agent cycle, reasoning can be interrupted, as in AgentSpeak, both by incoming observations and by outgoing actions. An incoming observation, for example, might trigger an integrity constraint and derive an action that needs to be performed immediately, interrupting the derivation of plans and the execution of actions that need to be performed only later in the future.

Consider the English sentences in the following example:

If you have an essay to write, then you study late in the library.
If there is an emergency, then you get help.
If you press the alarm signal button, then you get help.
If there is a fire, then there is an emergency.

The four sentences all have conditional form. Moreover, the second and third sentences have the same conclusion *you get help.* However, despite their similar syntax, the first two sentences would function as *maintenance goals* and would be represented by integrity constraints in the ALP agent model. The last two sentences would function as beliefs and would be represented by clauses. The predicates *"there is a fire"* and *"you press the alarm signal button"* are abducible predicates, representing possible observations and actions respectively. The predicate *you study late in the library* would be defined by other clauses.

To be precise, all of the predicates would need to include associated times. These times would reflect the fact that getting help in an emergency is more urgent than studying late in the library.

Given an update recording an observation that *you have an essay to write*, forward reasoning with the first sentence would derive the achievement goal *you study late in the library* and set in train a process of backward reasoning and action execution to solve the achievement goal.

A second update recording an observation that *there is a fire* would interrupt this process and initiate a chain of forward reasoning to determine the consequences of the new update. Forward reasoning with the fourth sentence derives the consequence *there is an emergency*. Forward reasoning with the second sentence derives the new achievement goal *you get help.* Backward reasoning with the third sentence reduces this goal to the abducible sub-goal, *you press the alarm signal button*. Because the sub-goal is an action abducible, it can be solved only by making it true, executing it successfully in the environment. Because the action is urgent, it interrupts the process of studying late in the library, which can be returned to later.

In production systems and AgentSpeak, all four sentences would be represented uniformly as production rules or plans, and they would all be executed by forward chaining. Different uses of a belief would be represented by different rules or plans. For example, in AgentSpeak, the four sentences above would be represented by such plans as:

+ you have an essay to write: true ⇐ !you study late in the library
+there is an emergency: true ⇐ !get help.
+!get help: true ⇐ press the alarm signal button.
+ ?there is a fire: there is an emergency ⇐ true.
+ there is a fire: true ⇐ +there is an emergency.
+! there is an emergency: true ⇐ ! there is a fire.

Because AgentSpeak plans do not have a uniform logical reading, they can not easily be used to represent weak domain-general knowledge, as is possible in logic programming. On the one hand, this is a limitation, but on the other hand it probably encourages programmers to write more efficient domain-specific problem-solving methods.

12 Decision-making in ALP agents – another role for probability

The ALP agent observation-thought-decision-action cycle includes a decision-making component to choose between alternatives courses of action, including alternative ways of solving the same goal. Different alternatives may solve an initial goal with different degrees of utility, as well as have other desirable or undesirable consequences as side effects.

This decision-making process is further complicated by the uncertainty of the consequences of an agent's candidate actions, due to the uncertain behaviour of other agents and the uncertainty of the environment. Thus the problem of deciding between different courses of actions requires consideration of both utility and uncertainty. According to the ideals of classical decision theory, a rational agent should choose a course of actions that optimises overall expected utility, including the utility of any side effects, and taking into account any associated uncertainties.

Decision theory is a normative ideal, which requires a large body of knowledge and huge computational resources. As a consequence, it is often more practical for an agent to employ heuristic strategies that approximate the normative ideal. One of the simplest of these strategies is to use a fixed priority ordering of goals and beliefs, to give preference to one set of actions over another.

Consider, for example, the following two ways of going home for the weekend:

you will go home for the weekend if you have the bus fare and you catch a bus home.
you will go home for the weekend if you have a car and you drive the car home.

Both alternatives can be used to solve the goal of going home, but with different utilities and probabilities. Taking into account the state of the car and the reliability of the bus service, it may be possible to associate explicit probabilities with the two clauses. For example:

you will go home for the weekend if you have the bus fare and you catch a bus home.
(with probability .95)

you will go home for the weekend if you have a car and you drive home.
(with probability .80)

However, as Poole (1993, 1997) has shown, the same effect can be achieved by associating probabilities with abducible predicates:

you will go home for the weekend if you have the bus fare and you catch a bus home
and the bus journey is successful.
you will go home for the weekend if you have a car and you drive the car home
and the car journey is successful.
the bus journey is successful. (with probability .95)
the car journey is successful. (with probability .80)

To do a full-scale decision-theoretic analysis, in addition to determining the probabilities, it would also be necessary to determine the degree to which the alternatives accomplish their intended goal, as well as to the quantify the costs and benefits of other possible consequences[66]. For example, driving the car may be more comfortable than taking the bus, and the bus schedule may be inconvenient. But taking the bus may cost less money, and, by contributing to the reduction of global carbon emissions, help to save the planet.

You could do a PhD calculating the probabilities and utilities and on combining them to find the optimal solution. But, the result might be equivalent to just giving preference to the use of one alternative over the other. This could be done, for example, just using normal logic programming clauses, Prolog-fashion in a fixed order. For example, in the order:

you will go home for the weekend if you have the bus fare
and you catch a bus home
and not something goes wrong with the bus journey.

you will go home for the weekend if you have a car
and you drive home
and not something goes wrong with the car journey.

13 Conclusions

I have argued that the two main kinds of conditionals used in AI, namely production rules and logic programming clauses, can be combined naturally in the ALP agent model and can simulate human performance on a number of reasoning tasks. Because production systems have been used widely as a cognitive architecture, but have not been much used to model human reasoning, there are grounds for believing that the ALP agent model may have value as a general cognitive architecture.

Psychological studies of human reasoning are a test and a challenge for agent models developed in Artificial Intelligence. I have argued that, although the ALP agent model performs well on some of these tests, the basic model needs to be extended, to include forward reasoning with clauses and to implement the consistency semantics of integrity constraints. I believe that these extensions, and possibly others that might be needed to simulate human reasoning in other psychological experiments, also suggest improvements of the ALP agent model, which can be used for computer applications in AI.

[66] In classical Decision Theory, the possible outcomes of candidate actions are given in advance. The extension of the IFF proof procedure to include forward reasoning with clauses makes it possible to derive consequences of candidate actions to help in deciding between them (Kowalski, 2006), using clauses of the form *effect if cause.* The same clauses can also be used backwards to generate plans to achieve desired *effects* by reducing them to possible *causes.*

Acknowledgements

Many thanks to Fariba Sadri, Michiel van Lambalgen and Keith Stenning for helpful discussions, and to Keith Clark, Luis Pereira, Keith Stenning and Guy Politzer for useful comments on an earlier draft of this paper. Thanks too to Mike Oaksford for the invitation to contribute to this volume.

References

Anderson, J. and Bower, G. (1973). *Human Associative Memory*. Washington, D.C.: Winston.

Bondarenko, A., Dung, P. M., Kowalski, R., and Toni, F. (1997) An Abstract Argumentation-theoretic Approach to Default Reasoning. *Journal of Artificial Intelligence 93 (1-2), 1997*, 63-101.

Bonnefon, J.-F. and Politzer, G. (2007) Pragmatic Conditionals, Conditional Pragmatics, and the Pragmatic Component of Conditional Reasoning. *This volume.*

M. E. Bratman, D. J. Israel, and M. E. Pollack (1988) Plans and resource-bounded practical reasoning, *Computational Intelligence, vol. 4,* 349–355.

Clark, K.L. (1978) Negation by failure. In Gallaire, H. and Minker, J. [eds], *Logic and Databases*, Plenum Press, 293-322.

Colmerauer, A., Kanoui, H., Pasero, R. and Roussel, P. (1973) *Un systeme de communication homme-machine en Francais.* Research report, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille II, Luminy.

Console, L., Theseider Dupre, D. and Torasso, P. (1991) On the relationship between abduction and deduction. *Journal of Logic and Computation. 1(5)* 661-690

Dancygier, B. (1998) *Conditionals and Prediction. Time, Knowledge, and Causation in Conditional Constructions*. Cambridge: Cambridge University Press.

De Raedt, L. and Kersting, K. (2003) Probabilistic Logic Learning. In: SIGKDD *Explorations 5/1* 31-48.

Dung, P. M., Kowalski, R., and Toni, F. (2006) Dialectic proof procedures for assumption-based, admissible argumentation. *Journal of Artificial Intelligence 170(2), 2006*,114-159.

van Emden, M. (2006) The Early Days of Logic Programming: A Personal Perspective *The Association of Logic Programming Newsletter,* Vol. 19 n. 3, August 2006. http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/aug06/

van Emden, M. and Kowalski, R. (1976) The Semantics of Predicate Logic as a Programming Language *JACM* , Vol. 23, No. 4, 733-742.

Eshghi, K. and Kowalski, R. (1989) Abduction Compared with Negation by Failure. In *Sixth International Conference on Logic Programming,* (eds. G. Levi and M. Martelli) MIT Press, 234-254.

d'Inverno, Luck, M., M., Georgeff, M. P., Kinny, D., and Wooldridge, M. (1998) A Formal Specification of dMARS. In: *Intelligent Agents IV: Proceedings of the Fourth International*

Workshop on Agent Theories, Architectures and Languages. Lecture Notes in Artificial Intelligence, 1365. Springer-Verlag, 155-176.

Fung, T.H. and Kowalski, R. (1997) The IFF Proof Procedure for Abductive Logic Programming. *Journal of Logic Programming.*

Gelfond, M. and Lifschitz, V. (1990) Logic programs with classical negation. *Proceedings of the Seventh International Conference on Logic Programming,* MIT Press, 579-597.

Godfrey, P., Grant, J., Gryz, J. and Minker, J. (1998) Integrity constraints: semantics and applications. In *Logics for databases and information systems.* Kluwer, Norwell, MA, USA. 265 – 306.

Kakas, A., Kowalski, R., Toni, F. (1998) The Role of Logic Programming in Abduction. In: Gabbay, D., Hogger, C.J., Robinson, J.A. (eds*.): Handbook of Logic in Artificial Intelligence and Programming 5*. Oxford University Press 235-324.

Kakas, A., Mancarella, P., Sadri, S., Stathis, K. and Toni, F. (2004) *The KGP model of agency,*
ECAI04, General European Conference on Artificial Intelligence, Valencia, Spain, 33-37.

Kowalski, R. and Kuehner, D. (1971) Linear Resolution with Selection Function. *Artificial Intelligence,* Vol. 2, 227-60. Reprinted in *Anthology of Automated Theorem-Proving Papers*, Vol. 2, Springer-Verlag, 1983, 542-577.

Kowalski, R., (1973) Predicate Logic as Programming Language. Memo 70, Department of Artificial Intelligence, Edinburgh University. Also in *Proceedings IFIP Congress*, Stockholm, North Holland Publishing Co., 1974, 569-574. Reprinted In *Computers for Artificial Intelligence Applications*, (eds. Wah, B. and Li, G.-J.), IEEE Computer Society Press, Los Angeles, 1986, 68-73.

Kowalski, R. (1974) *Logic for Problem Solving.* DCL Memo 75, Department of Artificial Intelligence, U. of Edinburgh. Expanded edition published by North Holland Elsevier 1979. Also at http://www.doc.ic.ac.uk/~rak/.

Kowalski, R. (1995) Using metalogic to reconcile reactive with rational agents.
In: Meta-Logics and Logic Programming (K. Apt and F. Turini, eds.), MIT Press

Kowalski, R. (2001) Artificial intelligence and the natural world. *Cognitive Processing, 4,* 547-573.

Kowalski, R. (2006) The Logical Way to be Artificially Intelligent. *Proceedings of CLIMA VI* (eds. F. Toni and P. Torroni) Springer Verlag, LNAI, 1-22.

Kowalski, R. and Sergot, M. (1986) A Logic-based Calculus of Events. In New Generation Computing, Vol. 4, No.1, 67-95. Also in The Language of Time: A Reader (eds. Inderjeet Mani, J. Pustejovsky, and R. Gaizauskas) Oxford University Press. 2005.

Kowalski, R., Sadri, F. and Soper, P. (1987) Integrity Checking in Deductive Databases. In: *Proceedings of VLDB,* Morgan Kaufmann, Los Altos, Ca. 61-69.

Kowalski, R., Sadri, F. (1999) From Logic Programming towards Multi-agent Systems. *Annals of Mathematics and Artificial Intelligence.* Vol. 25 391-419.

Kowalski, R., Toni, F. and Wetzel, G. (1998) Executing Suspended Logic Programs. *Fundamenta Informatica 34 (3)*, 1-22.

Kunen, K. (1987) Negation in logic programming. *Journal of Logic Programming 4:4* 289 - 308

Laird, J.E., Newell, A. and Rosenblum, P. S. (1987) SOAR: an architecture for general intelligence. *Artificial Intelligence,* 33:1, 1 – 64.

Lloyd, J W, Topor, R W (1984) Making PROLOG more expressive. *Journal of Logic Programming.* Vol. 1, no. 3, 225-240.

McCarthy, J. (1958) Programs with common sense, *Symposium on Mechanization of Thought Processes*. National Physical Laboratory, Teddington, England.

McCarthy, J. (1980) Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27-39.

McDermott, D. and Doyle, (1980) Nonmonotonic logic I," *Artificial Intelligence*, 13:41-72.

Minsky, M. (1974) A Framework for Representing Knowledge. Technical Report: AIM-306,

Massachusetts Institute of Technology, Cambridge, MA.

Moore, R. C. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75-94.

Newell, A., Shaw, J. C., and Simon, H.A. (1957) Empirical explorations of the logic theory machine. Proc. Western Joint Comp. Conf, 218-239.

Newell, A. (1973). Production Systems: Models of Control Structure. In W. Chase (ed): *Visual Information Processing* 463-526 New York: Academic Press 463-526.

Newell, A. (1990). *Unified theories of cognition.* Harvard University Press Cambridge, MA, USA.

Nicolas, J.M., Gallaire, H. (1978) Database: Theory vs. interpretation. In Gallaire, H., Minker, J. (eds*.): Logic and Databases.* Plenum, New York.

Pearl, J. 1988: *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann.

Politzer G. and Bonnefon J-F. (2006) Two varieties of conditionals and two kinds of defeaters help reveal two fundamental types of reasoning. *Mind and Language,***21**, 484-503.

Politzer G. and Bonnefon J-F. (this volume)

Poole, D., Goebel, R. and Aleliunas R. (1987) Theorist: a logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla (Eds.) *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer Verlag, New York, 1987, 331-352.

Poole, D. (1993) Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1) 81–129.

Poole, D. (1997) The independent choice logic for modeling multiple agents under uncertainty. *Artificial Intelligence.* Vol. 94 7-56.

Rao, A. S., & Georgeff, M. P. (1991). *Modeling rational agents with a BDI-architecture*. Second International Conference on Principles of Knowledge Representation and Reasoning. 473-484.

Rao, A.S. (1996) Agents Breaking Away, *Lecture Notes in Artificial Intelligence, Volume 1038,* (eds Walter Van de Velde and John W. Perrame) Springer Verlag, Amsterdam, Netherlands.

Reiter, R. (1978) On closed world data bases. In H. Gallaire and J. Minker, editors: *Logic and Data Bases*, Plenum, New York. 119-140.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81-132.

Robinson, J. A. (1965) A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM,* 12(1) 23 – 41.

Russell, S.J. & Norvig, P. (2003) Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall.

Sagonas, K., Swift, T. & Warren, D. (1994) XSB as an efficient deductive database engine. In ACM SIGMOD, ACM SIGMOD Record archive
Volume 23 , Issue 2 442 - 453

Sato, T. (1995) A statistical learning method for logic programs with distribution semantics In *Proceedings of the 12th International Conference on Logic Programming.* MIT Press. 715-729.

Shoham, Y. (1991) *AGENT0:* A Simple Agent Language and Its Interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (AAAI-91), AAAI Press/MIT Press, Anaheim, California, USA, 704-709.

Shortliffe, E. (1976) *Computer-based Medical Consultations: MYCIN.* North Holland Elsevier.

Simon, H. (1999) Production Systems. In Wilson, R. and Keil, F. (eds.): *The MIT Encyclopedia of the Cognitive Sciences.* The MIT Press. 676-677.

Sperber, D., Cara, F., & Girotto, V. (1995). Relevance theory explains the selection task. *Cognition, 52*, 3-39.

Stenning, K. & van Lambalgen, M. (2004) A little logic goes a long way: basing experiment on semantic theory in the cognitive science of conditional reasoning. *Cognitive Science*, 28:4, 481-529.

Stenning, K. & van Lambalgen, M. (2005) Semantic interpretation as computation in nonmonotonic logic: the real meaning of the suppression task. *Cognitive Science*, 29(6), 919–96

Stenning, K. and van Lambalgen M., (2008) *Human reasoning and cognitive science.* MIT Press. (in press).

Thagard, P. (2005) *Mind: Introduction to Cognitive Science.* Second Edition. MIT Press.

Wason, P. C. (1964). Reasoning. In Foss, B. (Ed.), New Horizons in Psychology. Harmondsworth: Penguin Books

Wason, P. C. (1968) 'Reasoning about a rule', The Quarterly Journal of Experimental Psychology, 20:3, 273 - 281

Winograd, T. (1971) *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language* MIT AI Technical Report 235, February 1971. Also published as a full issue of the journal *Cognitive Psychology* Vol. 3 No 1, 1972, and as a book, *Understanding Natural Language* (Academic Press, 1972).

Appendix A The minimal model semantics of logic programming

A.1 Horn clause programs

The minimal model semantics is consistent with the declarative reading of clauses as implications in classical logic. However, in programming and database contexts and with Horn clause programs ***P***, the semantics is also much simpler than in classical logic. In such cases, because clauses always have positive conclusions, it is natural to regard the set of clauses ***P*** as defining the set of positive, atomic sentences ***A*** that are logically implied by ***P***.

This relationship between ***P*** and an atomic sentence ***A*** can be understood purely in terms of classical, logical consequence:

> ***A*** is true in every model of ***P***.

However, sets of Horn clauses ***P*** have the special property (van Emden and Kowalski, 1976) that there exists a *unique minimal model* ***M*** of ***P*** such that:

> For every atomic sentence ***A,***
> ***A*** is true in every model of ***P*** if and only if ***A*** is true in ***M***.

This minimal model ***M*** can be represented syntactically as the set of all atomic sentences ***A*** that are true in ***M***. The model ***M*** is *minimal* in the sense that it is the smallest set of atomic sentences that is a model of ***P***.

For example, the Horn clauses, which define addition and multiplication for the natural numbers:

$$X + 0 = X$$
$$X + (Y + 1) = (Z + 1) \text{ if } X + Y = Z$$
$$X \cdot 1 = X$$
$$X \cdot (Y + 1) = (X \cdot Z) \text{ if } X \cdot Y = Z$$

have a unique minimal model, which is the standard model of arithmetic. In contrast, the completion of the Horn clause program is equivalent to the Peano axioms of arithmetic, without the axioms of induction.

The minimal model can be constructed in the limit, simply by repeatedly applying *modus ponens* and universal instantiation in all possible ways to all the clauses in ***P.*** This model construction property can also be interpreted as the completeness property of forward reasoning for atomic consequences. Backward reasoning, in the form of SLD-resolution (Kowalski, 1973/1974), which is equivalent to the search tree proof procedure, is also sound and complete for atomic consequences of Horn clauses.

A.2 The minimal model semantics of negation

The simplest case is that of a Horn clause program ***P*** and the negation *not G* of a positive atomic sentence *G*. According to the minimal model semantics:

> *not G holds* if and only if *not G* is *true* in the minimal model ***M*** of ***P***.

More generally, for any sentence *G*, including any sentence of first-order logic:

> *G holds* if and only if *G* is *true* in ***M.***

Backward reasoning and negation as infinite failure are sound and complete inference rules for this semantics. In other words:

G succeeds if and only if *G* is *true* in ***M.***

This property depends upon the rewriting of first-order formulae into goals that are conjunctions of literals augmented with auxiliary clauses, as mentioned in section 6.1.

The minimal model semantics can be extended in a number of different ways to the more general case of normal logic programs ***P***, which contain negative literals in the conditions of clauses. One of the simplest and most natural of these extensions interprets *not G* literally as *G does not* hold and expands ***P*** with a set Δ of true sentences of the form *not G*. The phrase *does not hold* can be treated as a positive auto-epistemic predicate, so that the expansion $\boldsymbol{P} \cup \Delta$ can be treated as a Horn clause program, which has a unique minimal model ***M***:

M is a *model* of ***P*** if all the sentences in Δ are true in ***M***.[67]

It is common to restrict auto-epistemic expansions Δ to ones that are total or maximal.[68] For simplicity, in this paper, I refer to this semantics and its variants simply and collectively as the *minimal model semantics* of negation as failure.

A program with negation as failure can have several such minimal models. According to the *credulous* version of this semantics,

a goal holds if and only if it is true in some minimal model.

According to the *sceptical* version of the same semantics,

a goal holds if and only if it is true in all minimal models.

In both cases, the goal can be any sentence of first-order logic, and backward reasoning and negation as infinite failure can be used determine whether or not a goal holds.

The auto-epistemic interpretation of negation as failure can be formulated in abductive logic programming terms by treating ground literals *not G* as abducible atomic sentences, and by letting the set of integrity constraints be all sentences of the form *if G and not G then false.* This gives an abductive interpretation to negation as failure (Eshghi and Kowalski, 1989).

[67] In other words, there is no *not G* in Δ such that *G* belongs to ***M***.

[68] If Δ consists of *all* such true sentences, then the resulting model is called a *stable model* (Gelfond and Lifschitz, 1990). A program can have many or no stable models. For example, the program {*p if not q, q if not p*} has two stable models {*p, not q*} and {*q, not p*}, but the program {*p if not p*} has no stable models. However, if we drop the requirement that Δ consists of *all* such true sentences, then the program {*p if not p*} has the semantics in which Δ is just the empty set.

Appendix B Conditionals in natural language

Part of the problem with understanding natural language conditionals is the problem of translating them into logical form. We saw an example of this in the motivating example at the beginning of the paper. However, the completion semantics suggests that there can be an added difficulty with choosing between different logical forms, corresponding to the two directions of the completion of a definition.

B.1 Switching the direction of conditionals

The completion semantics shows that it can be difficult to decide the direction for a conditional. Consider, for example, the two clauses:

A if B.
A if C.

whose completion is

A if and only if B or C.

The only-if-half of the completion is implicit in the original set of two clauses. But it is also possible to write the two clauses in the *switched form,* in which the only-if half is explicit and the if-half is implicit:

B or C if A.

The disjunction in the conclusion of the switched form can be eliminated by using negative conditions, yielding the switched clauses:

B if A and not C.
C if A and not B.

For example, the two clauses for getting help in section 6.3 can be transformed into any of the following switched forms:

you have pressed the alarm signal button or you shouted loudly if you get help
you have pressed the alarm signal button if you get help and you have not shouted loudly
you shouted loudly if you get help and you have not pressed the alarm signal button

The completion semantics, therefore, justifies four ways of representing the same conditional relationship: the if-form, the if-and-only-if form, the only-if form and the only-if form with negative conditions. Both the first and the fourth representation can be represented as logic programming clauses, and it can be difficult for a programmer to know which representation to use.

In applications such as fault diagnosis in AI, different kinds of representation are common. The first representation is common because it models causality in the form *effect if cause.* But this representation requires the use of abduction to explain observations. The fourth representation, which models causality in the switched form *cause if effect and not other-causes*, is also common because it requires only the use of deduction.

B.2 Truncated conditionals

Bonnefon and Politzer (2007) observe that many conditionals in natural language have a truncated form $A \rightarrow \Phi$ that, to be understood fully, needs to be put into a wider context of the form:

$(A \ \& \ A_1 \ \&...\& \ A_\alpha)$ *or* $(B \ \& \ B_1 \ \&... \ \& \ B_\beta)$ *or...or* $(C \ \& \ C_1 \ \& \ ... \ \& \ C_\nu) \rightarrow \Phi$

In this wider context, there may be both additional, unstated conditions $(A_1 \ \&... \ \& \ A_\alpha)$ and additional, unstated, alternative ways $(B \ \& \ B_1 \ \&...\& \ B_\beta)$ *or...or* $(C \ \& \ C_1 \ \& \ ... \ \& \ C_\nu)$ of establishing the conclusion Φ.

But, as we have seen, this wider context is exactly a logic program, which can also be expressed in the completion form:

$(A \ \& \ A_1 \ \&...\& \ A_\alpha)$ *or* $(B \ \& \ B_1 \ \&... \ \& \ B_\beta)$ *or...or* $(C \ \& \ C_1 \ \& \ ... \ \& \ C_\nu)) \leftrightarrow \Phi$

The completion suggests the alternative possibility that a conditional might, instead, be a truncated form of the switched, only-if half $\Phi \rightarrow A$ of the wider context. I will come back to this possibility in B.3.

But, first, consider the following example of a simple truncated conditional, in logic programming form:

X flies if X is a bird

The wider context is a more precise statement that includes extra conditions and alternative ways of flying. In most AI knowledge representation languages, it is common to express the conditional more precisely (although not as completely as in Bonnefon and Politzer's wider context) by lumping all of the extra conditions into a single condition, which is assumed to hold by default. For example:

X flies if X is a bird and not X is unable to fly

Here the predicate *X is unable to fly* can be defined by other clauses, such as:

X is unable to fly if X is crippled
X is unable to fly if X has not left its nest etc.

which is equivalent to specifying other conditions for the clause. The minimal model and completion semantics both incorporate the closed world assumption that, unless there is evidence that a bird is unable to fly, then the bird is assumed to fly by default.

In general, the full alternative:

$(A \ \& \ A_1 \ \&...\& \ A_\alpha) \rightarrow \Phi$

which spells out all the missing conditions of a truncated conditional $A \rightarrow \Phi$ can be expressed in logic programming style with a default condition, which can be defined separately. For example in the form:

Φ *if A & not exceptional*
exceptional if not A_1 ....
exceptional if not A_α

Typically, the condition A of a truncated conditional $A \rightarrow \Phi$ is the most significant of the conditions $A \,\&\, A_1 \,\&...\&\, A_\alpha$, and the truncated conditional can be regarded as an *approximation* of the full alternative $(A \,\&\, A_1 \,\&...\&\, A_\alpha) \rightarrow \Phi$.

The precise form, say A *& not exceptional* $\rightarrow \Phi$, of a truncated conditional can be viewed as a solution of the *qualification problem*: How to suitably qualify a conclusion to adequately account for all its conditions, without specifying them all in advance and in detail. This precise form of the truncated conditional has the advantage that the conclusion can be qualified precisely from the outset by means of a single default condition, and exceptions to that condition can be spelled out separately in successively more precise formulations. The qualification problem is one aspect of the notorious *frame problem* in Artificial Intelligence, which has a similar solution.[69]

B.3 Truncated and switched conditionals

In B.1, I argued that the completion semantics explains why it may be difficult to determine the right direction for a conditional. This difficulty is compounded when a conditional is a truncated conditional $A \rightarrow \Phi$, which is part of a wider context

$(A \,\&\, A_1 \,\&...\&\, A_\alpha)$ *or* $(B \,\&\, B_1 \,\&...\&\, B_\beta)$ *or...or* $(C \,\&\, C_1 \,\&\, ...\,\&\, C_\nu) \rightarrow \Phi$

because the converse of the truncated conditional $\Phi \rightarrow A$ is itself a truncation of the switched form:

$\Phi \rightarrow (A \,\&\, A_1 \,\&...\&\, A_\alpha)$ *or* $(B \,\&\, B_1 \,\&...\&\, B_\beta)$ *or...or* $(C \,\&\, C_1 \,\&\, ...\,\&\, C_\nu)$

Understanding these two different ways in which a natural language conditional relates to its wider context can help to explain how people reason with conditionals. Consider the following example from (Politzer and Bonnefon, 2006).

Given the two sentences:

If an object looks red, then it is red.
This object looks red.

it is natural to draw the conclusion:

This object is red.

However, given the additional sentence:

If an object is illuminated by red light, then it looks red.

it is natural to withdraw the conclusion. But it is not natural to combine the two conditionals and draw the obviously false conclusion:

If an object is illuminated by red light, then it is red.

[69] For example, in the event calculus (Kowalski and Sergot, 1986), the fact that a property is assumed to hold from the time it is initiated until there is reason to believe that it has been terminated is expressed in the form:

P holds at time T_2 *if an event E initiates P at time* T_1 *and* $T_1 < T_2$
and not (an event E' terminates P at time T and $T_1 < T < T_2$*)*

This way of reasoning can be explained if the two conditionals both refer to the same wider context, but the first one is switched. In logic programming, this wider context can be expressed in the clausal form:

An object looks red if it is red and not abnormal.
An object looks red if it is illuminated by a red light and not abnormal'.

In this analysis, the first inference that *this object is red* is explained by interpreting the first clause as the *only* clause that can be used in establishing that *an object looks red* and interpreting the first conditional as the switched, only-if half of the completion of the clause.

The withdrawal of the inference is explained by interpreting the new conditional as indicating that there is an additional alternative way of establishing that *an object looks red*, as represented by the second clause of the logic program. The only-if half of the two clause program no longer implies the withdrawn inference.

The failure to draw the obviously false conclusion is explained by the fact that in the wider context, the meaning of the two conditionals is actually the two clauses, which express alternative ways of establishing the same conclusion, and which do not combine to imply the false conclusion.

In summary, the example illustrates that a pair of natural language conditionals whose surface structure has the form:

$\Phi \rightarrow A$
$B \rightarrow \Phi$

Might actually have a very different deep structure, say:

$(A \text{ \& not abnormal}) \text{ or } (B \text{ \& not abnormal'}) \leftrightarrow \Phi$

or in clausal form:

Φ *if A & not abnormal*
Φ *if B & not abnormal'*

Appendix 4

Legislation as Logic Programs*

Robert A. Kowalski

Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ, UK

January 1991
Revised June 1992

Abstract. The linguistic style in which legislation is normally written has many similarities with the language of logic programming. However, examples of legal language taken from the British Nationality Act 1981, the University of Michigan lease termination clause, and the London Underground emergency notice suggest several ways in which the basic model of logic programming could usefully be extended. These extensions include the introduction of types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning and procedural notation.

In addition to the resemblance between legislation and programs, the law has other important similarities with computing. It needs for example to validate legislation against social and political specifications, and it needs to organise, develop, maintain and reuse large and complex bodies of legal codes and procedures. Such parallels between computing and law suggest that it might be possible to transfer useful results and techniques in both directions between these different fields. One possibility explored in this paper is that the linguistic structures of an appropriately extended logic programming language might indicate ways in which the language of legislation itself could be made simpler and clearer.

1 Introduction

The characteristic feature of the language of legislation is that it uses natural language to express general rules, in order to regulate human affairs. To be effective for this purpose, it needs to be more precise than ordinary language and, as much as possible, it needs to be understood by different people in the same way. In this respect legislation can be viewed as programs expressed in human language to be executed by humans rather than by computers.

Thus the language of legislation might also serve as a model for computing, suggesting ways in which programming languages might be made more like human languages, while still remaining machine executable. In this paper I shall focus on a comparison between the language of legislation and the language of logic programming. I shall argue that, although logic programming fares well in this

*

comparison, it needs to be improved by incorporating such extensions as types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning, and procedural notation. I shall also argue that in some cases legislation itself can be improved by re-expressing it in a style more closely resembling such an extended logic programming form.

I shall investigate three examples. The first consists of several sections from the British Nationality Act 1981; the second is the University of Michigan lease termination clause; and the third is the London underground emergency notice. The first example was investigated earlier by the author and his colleagues [24] as an illustration of the use of logic programming for representing legislation. The second was investigated by Allen and Saxon [1] as an example of the use of logic to eliminate ambiguities in the formulation of a legal contract. The third was identified by the author [13] as an example of a public notice which is meant not only to be precise but also to be as clear as possible to ordinary people.

In our earlier investigation of the British Nationality Act 1981 [10] we emphasized both the prospects of using logic programming to build legal applications as well as the problems of attempting to use logic programming for knowledge representation. In this paper I am concerned only with the second of these matters, but more specifically with investigating linguistic similarities and differences between logic programming and legislation, and more generally with exploring other parallels between computing and the law.

2 The British Nationality Act 1981

The following four examples from the British Nationality Act illustrate some of the complexity and precision of legal language. They also illustrate the treatment of time, default reasoning, negative conclusions and reasoning about belief.

2.1 Acquisition by Birth

The first subsection of the British Nationality Act deals with the case of acquisition of citizenship by virtue of birth in the United Kingdom after commencement (1 January 1983, the date on which the Act took affect).

> 1.-(1) A person born in the United Kingdom after commencement shall be a British citizen if at the time of the birth his father or mother is -
> (a)a British citizen; or
> (b)settled in the United Kingdom.

The English of this clause is already close to logic programming form, even to the extent of expressing the conclusion before (most of) the conditions. Using infix notation for predicates and upper case letters for variables, 1.1 can be paraphrased in logic programming form by:

> X acquires british citizenship by section 1.1
> if X is born in the uk at T
> and T is after commencement
> and Y is parent if X

and Y is a british citizen at T or
Y is settled in the uk at T

This has the propositional form

A if [B and C and D and [E or F]]

which is equivalent to two rules

A if B and C and D and E
A if B and C and D and F

in normal logic programming form.

In this paper I shall use the term logic program to refer to any set of sentences which are equivalent to a set of universally quantified implications in the normal logic programming form

A if B1 and ... and Bn

where A is an atomic formula, Bi for $0 < i < n$ is an atomic formula or the negation of an atomic formula, and all variables, e.g. X1, ..., Xm occurring in the implication are assumed to be universally quantified, i.e.

for all X1, ..., Xm [A if B1 and ... and Bn].

The logic programming representation of 1.1 can be made more like the English, while remaining formal, by introducing types and various forms of syntactic sugar. For example:

a person who is born in the uk at a time
which is after commencement
acquires british citizenship by section 1.1
if a parent of the person is a british citizen at the time,
or a parent of the person is settled in the uk at the time.

Here "person" and "time" are type identifiers; "a person" is the first occurrence of a variable of type "person"; "a time" is the first occurrence of a variable of type "time"; "the person" and "the time" stand for later occurrences of the same variables. The relative pronouns "who" and "which" also stand for additional occurrences of the variables they follow. "who" stands for an occurrence of type "person", whereas "which" stands for an occurrence of any type of variable. Relative clauses in expressions of the form

... V which P ***

for example, are syntactic sugar for

... V *** if V P

where "V" is a variable, and "P" is a predicate which applies to "V".
Similarly an expression of the form

... a R of T P ***

is syntactic sugar for

... V P *** if V R of T

where "R of" is a binary predicate, "T" is a term, and "V" is any variable not occurring elsewhere in the sentence.

Notice that the two transformations described above need to be combined with the simplication of formulae of the form

(A if B) if C

to the form

A if B and C

This kind of typing and syntactic sugar can be defined more precisely and can be extended to deal with several variables of the same type, pronouns, and more flexible kinds of relative clauses. In this way English can serve as a model to improve the naturalness of logic programming without sacrificing its precision.

I shall argue elsewhere in this paper that, conversely, the use of conclusion-conditions form, which characterises the syntax of logic programming, can sometimes improve the clarity of natural languages such as English.

2.2 Representation of Time

In the representation of 1.1 time has been represented by an explicit parameter of type "time". The expression

... after ***

is interpreted as short-hand for

... at a time which is after ***
i.e.

... at T if T is after ***.

This explicit representation of time contrasts with modal representations, where temporal relationships are represented by modal operators, and time itself is implicit rather than explicit.

As mentioned for example in [11], to reason about time, an explicit axiom of persistence can be formulated to express that

a property holds at a time which is after another time
if an event occurs at the other time
and the event initiates the property
and it is not the case that
another event occurs at yet another time
which is between the time and the other time
and the other event terminates the property.

"a person acquires british citizenship by section 1.1" initiates
"the person is a british citizen".

Perhaps this is an example where symbolic notation with explicit representation of variables might be easier for some people to follow. Here "a time", "another time", and "yet another time" introduce different variables of the same type "time". Notice that the English suggests that the variables refer to distinct individuals, whereas the usual logical convention is that different variables of the same type can refer to the same individual. This is one of several discrepancies which would need to be attended to in a more systematic study of the correspondence between logic and a precise style of English.

Notice also in the two axioms above how events and properties are treated metalogically as names of sentences.

2.3 Abandoned Children and Default Reasoning

The second subsection of the British Nationality Act is conceptually one of the most complex sentences of the Act.

1.-(2) A new-born infant who, after commencement, is found
abandoned in the United Kingdom shall, unless the contrary is shown,
be deemed for the purposes of subsection (1)-
(a) to have been born in the United Kingdom after commencement; and
(b) to have been born to a parent who at the time of the birth
was a British citizen or settled in the United Kingdom.

Under the procedural interpretation of logic programs, conclusions of sentences are interpreted as goals and conditions as subgoals. According to this interpretation, the conclusion of a sentence identifies its purpose. Thus we can interpret the phrase "the purposes of subsection (1)" as a metalevel reference to the logical conclusion of 1.1, namely to acquire British citizenship. Moreover the object level phrases 1.2.a and 1.2.b are exactly the logical conditions of 1.1. Thus we can regard the entire sentence 1.2 as a mixed object level and metalevel sentence which expresses that

the conditions of 1.1 shall be assumed to hold for a person
if the person is found newborn abandoned in the uk
at a time which is after commencement.
and the contrary of the conditions of 1.1 are not shown

This can be reformulated at the object level alone by replacing the metalevel descriptions by their object level counterparts:

a person who is found newborn abandoned in the uk at a time
    which is after commencement
    acquires british citizenship by section 1.2
if it is not shown that it is not the case that
    the person is born in the uk at a time
    which is after commencement
and either it is not shown that it is not the case that
        a parent of the person is a british citizen at the time of birth
    or it is not shown that it is not the case that
        a parent of the person is settled in the uk at the time of birth

This seems to be a case where the mixed object-level meta-level expression may be easier to understand than the purely object level representation.

Conditions of the form

    it is not shown that it is not the case that P

in the object level sentence above, can be interpreted as combining negation as failure "not" and ordinary negation "¬", i.e.

    not ¬ P.

Thus, for another example, the statements

    A bird flies if it is not shown that it is not the case that the bird flies.
    It is not the case that an ostrich flies.

can be formalised by

    a bird flies if not ¬ the bird flies
    ¬ an ostrich flies

Just such an extension of logic programming to include both negation by failure and ordinary negation has been investigated by Gelfond and Lifschitz [8] and by Kowalski and Sadri [14].

Negation by failure is a form of default reasoning and is non-monotonic. Thus a person who acquires citizenship by 1.2 might non-monotonically have citizenship withdrawn in the light of new information. It is unlikely, however, that parliament intended that citizenship be withdrawn in this way. Both such an intention and the opposite intention can be catered for by introducing an extra layer of time concerned with the time for which beliefs are held in addition to the historical time for which properties hold true in the world. A logic programming approach to such a joint representation of belief time and historical time has been developed by Sripada [25].

It is important to emphasize that when formalising legislation as (extended) logic programs we do not attempt to define concepts which occur in conditions of the legislation but are not defined in the legislation itself. Thus, for example, we do not attempt to define the concept "new born infant" which occurs in the conditions of 1.2. This means, as a consequence, that a formalisation of the British Nationality Act has only limited applicability by itself. To be used in a particular case it would need to be supplemented, if not by a set of definitions of such vague terms, at least by a set of facts or assumptions which express judgements about whether or not such terms apply to the case in hand.

2.4 Deprivation of Citizenship and Negative Conclusions

Except for its occurrence in conditions of the form

not ¬ P

ordinary negation ¬ seems to be needed only in the conclusions of rules. In such cases, a negative conclusion typically expresses an exception to a general rule, as in the example

It is not the case that an ostrich flies.

which expresses an exception to the general rule that all birds fly.

Exceptions, expressed by sentences with negative conclusions, are common in legislation [12]. The provisions for depriving British citizens of their citizenship exemplify this use of negation:

> 40.-(1) Subject to the provisions of this section, the Secretary of State may by order deprive any British citizen to whom this subsection applies of his British citizenship if the Secretary of State is satisfied that the registration or certificate of naturalisation by virtue of which he is such a citizen was obtained by means of fraud, false representation or the concealment of any material fact.
>
> 40.-(5) The Secretary of State -
> (a) shall not deprive a person of British citizenship under this section unless he is satisfied that it is not conducive to the public good that that person should continue to be a British citizen; ...

40.1 has the logical form

P if Q

whereas 40.5 has the form

¬ P if not R

If both conditions Q and not R hold, then by ordinary logic it would be possible to

deduce a contradiction

P and ¬ P.

But this is not the intention of the legislation, which is rather that the exception should override the rule, or equivalently that the rule should be understood as having an extra, implicit condition.

P if Q and not ¬P.

In fact, the metalevel phrase "subject to the provisions of this section" at the beginning of 40.1 can be regarded as a caution that the meaning of 40.1 cannot be understood in isolation of the rest of the section as a whole.

The extension of logic programming to allow negative conclusions, for the purpose of representing exceptions, has been investigated by Kowalski and Sadri [14]. They also show that such extended logic programs can be transformed into normal logic programs. In particular a rule with a single exception

P if Q
¬ P if not R

can be transformed into the simpler rule

P if Q and R.

Both representations can be useful for different purposes. A representation in terms of rules and exceptions is often easier to develop and to maintain. However, the simpler representation as normal logic programs is usually clearer and easier to understand. The first representation, accordingly, might be preferred by a draftsman, who codifies the law; the second might be preferred by an administrator who executes the law.

In this discussion of the provisions for deprivation of citizenship we have considered only the propositional structure of the English sentences. We have not considered the meaning of such conditions as

"he is satisfied that it is not conducive
to the public good that that person
should continue to be a British citizen".

This is partly because it would be very difficult to do so; but also because we have restricted our attention to representing formally only what is defined explicitly in the legislation itself. Nonetheless, reasoning about reasoning can, at least to some extent, be formalised by metalogic or by logics of knowledge and belief.

2.5 Naturalisation and the Representation of Belief

Like the provisions for deprivation of citizenship, the provisions for naturalisation contain conditions concerning the Secretary of State's beliefs. In addition, however,

they also contain rules governing the subject matter of those beliefs. This leads us to consider whether we can establish a logical connection between the two.

Section 6.1 contains the main provision for naturalisation:

> 6.-(1) If, on an application for naturalisation as a British citizen made by a person of full age and capacity, the Secretary of State is satisfied that the applicant fulfills the requirements of Schedule 1 for naturalisation as such a citizen under this sub-section, he may, if he thinks fit, grant to him a certificate of naturalisation as such a citizen.

At the propositional level this is equivalent to a sentence in conclusion-conditions form:

> the secretary of state may grant a certificate of naturalisation to a person by section 6.1
> if the person applies for naturalisation
> and the person is of full age and capacity
> and the secretary of state is satisfied that the person fulfills the requirements of schedule 1 for naturalisation by 6.1
> and the secretary of state thinks fit to grant the person a certificate of naturalisation.

The last two conditions vest considerable powers of discretion in the Secretary of State. The last condition is totally inscrutable and can only be given as part of the input for a given case. But the meaning of the penultimate condition ought at least to be constrained by the meaning of Schedule 1. This schedule is quite long, and it is convenient therefore to summarise its contents:

> a person fulfills the requirements of schedule 1 for naturalisation by 6.1
> if either the person fulfills residency requirements specified in subparagraph 1.1.2
> or the person fulfills crown service requirements specified in subparagraph 1.1.3
> and the person is of good character
> and the person has sufficient knowledge of english, welsh, or scottish gaelic
> and either the person intends to reside in the uk in the event of being granted naturalisation
> or the person intends to enter or continue in crown service in the event of being granted naturalisation.

To understand the connection between 6.1 and Schedule 1, it is necessary to understand the connection between meeting the requirements for naturalisation specified in Schedule 1 and satisfying the Secretary of State that those requirements are met. Fortunately, this can be done, at least in part, by regarding satisfaction as a kind of belief. The appropriate rules of belief can be formalised in both modal logic

and metalogic. The following formalisation in metalogic has the form of a metainterpreter.

a person is satisfied that P
  if the person is satisfied that P → Q
  and the person is satisfied that Q

a person is satisfied that P ⌉ Q
  if the person is satisfied that P
  and the person is satisfied that Q

a person is satisfied that P v Q
  if the person is satisfied that P
  or the person is satisfied that Q

Here "→", "⌉", and "v" are infix function symbols naming implication, conjunction, and disjunction respectively.

We may safely assume that

the secretary of state is satisfied that P
if P is a representation of the meaning
of a provision of the british nationality act 1981

Thus the Secretary of State is satisfied in particular that the implication which represents the meaning of Schedule 1 holds. This assumption and the metainterpreters above are all we need to establish a logical connection between 6.1 and Schedule 1. This connection can be made more explicit, however, if we transform the metainterpreter using the technique of partial evaluation [7, 26]:

the secretary of state is satisfied that a person fulfills
the requirements for naturalisation by 6.1
if either the secretary of state is satisfied that
the person fulfills residency requirements specified in paragraph 1.1.2
  or the secretary of state is satisfied that
    the person fulfills crown service requirements specified in paragraph 1.1.3
and the secretary of state is satisfied that the person is of good character
and the secretary of state is satisfied that
    the person has sufficient knowledge of english, welsh, or Scottish gaelic
and either the secretary of state is satisfied that
    the person intends to reside in the uk in
    the event of being granted naturalisation
  or the secretary of state is satisfied that
    the person intends to enter or continue in
    crown service in the event of being granted naturalisation.

The result is an explicit, though somewhat tedious, statement of what it means to satisfy the Secretary of State concerning the requirements for naturalisation. Clearly the statement could be made a little less tedious if we used a pronoun, "he" or "she" for all references to the Secretary of State after the first.

The language of the British Nationality Act 1981 is for the most part extraordinarily precise. It is also very complex. Most of this complexity is inherent in the meaning of the Act. However, some of the complexity can be reduced by the explicit use of conclusion-conditions form and by the use of meaning-preserving transformations of the kind illustrated in the last two examples.

By comparison with ordinary language and even with legal language in general, the Act is also surprisingly unambiguous. However, as we have already seen, it does contain vague terms and undefined concepts. Such vagueness is often confused with ambiguity. Although, like genuine ambiguity, vagueness causes problems of interpretation, it is also useful, because it allows the law to evolve and adapt to changing circumstances.

Genuine ambiguity, on the other hand, generally serves no useful purpose. Moreover, whereas logic can easily accommodate vagueness, it cannot tolerate ambiguity.

The University of Michigan lease termination clause, presented in the next section, was originally investigated by Allen and Saxon [1] to illustrate the use of propositional logic to formulate a precise interpretation of an ambiguous legal text. I shall argue that the use of logic programming conclusion-conditions form has the further advantage of rendering many of the possible interpretations logically implausible.

3 The University of Michigan Lease Termination Clause

The clause consists of a single, long sentence which has the underlying, logically ambiguous form

A if A1 and A2 or A3 or A4 or A5 or A6 or A7
unless B1 or B2 or B3 or B4 or B5 in which cases B.

Different ways of introducing parentheses produce different interpretations. Some of these are logically equivalent because of the associativity of "or", for example. After accounting for these equivalences, Allen and Saxon identify approximately 80 questions that might need to be asked in order to distinguish between the different parenthesizations. As a result of this analysis they identify one intended interpretation which has the form

((A if (A1 and(A2 or A3)) or A4 or A5 or A6 or A7)
if not (B1 or B2 or B3 or B4 or B5)) and
(if (B1 or B2 or B3 or B4 or B5) then B)

where "unless" has been translated as "if not". It is interesting that this interpretation has a logic programming form.

The logic programming representation can be simplified if, as Allen and Saxon maintain, conditions B1-B5 are the only ones under which conclusion B holds. In

that case the conditions not(B1 or B2 or B3 or B4 or B5) can be replaced by not B. Thus the intended interpretation can be represented by the simplified, normal logic program:

A if A1 and A2 and not B
A if A1 and A3 and not B
A if A4 and not B
A if A5 and not B
A if A6 and not B
A if A7 and not B
B if B1
B if B2
B if B3
B if B4
B if B5

This logical analysis of the propositional structure of the sentence should be compared with the English text of the sentence:

"The University may terminate this lease when the Lessee, having made application and executed this lease in advance of enrollment, is not eligible to enroll or fails to enroll in the University or leaves the University at any time prior to the expiration of this lease, or for violation of any provisions of this lease, or for violation of any University regulation relative to Resident Halls, or for health reasons, by providing the student with written notice of this termination 30 days prior to the effective time of termination; unless life, limb, or property would be jeapordized, the Lessee engages in the sales or purchase of controlled substances in violation of federal, state or local law, or the Lessee is no longer enrolled as a student, or the Lessee engages in the use or possession of firearms, explosives, inflammable liquids, fireworks, or other dangerous weapons within the building, or turns in a false alarm, in which cases a maximum of 24 hours notice would be sufficient".

Notice how the conclusion A of the first half of the sentence is split into two parts by the insertion of the conditions A1-A7. Notice also that the language of the sentence is so complicated and so confused that the drafters mistakenly wrote "maximum of 24 hours" when they must have meant "minimum of 24 hours".

In fact I have slightly misrepresented Allen and Saxon's analysis of the sentence. In addition to identifying the intended placement of parentheses, they analyse for each of the three occurrences of "if" in the apparent meaning of the sentence whether or not "if and only if" is really intended. They conclude that in the first two cases (of the words "when" and "unless") it is not intended, whereas, in the third case (of the words "in which cases") it is. Thus their real analysis of the intended interpretation has the form

((A if (A1 and (A2 or A3)) or A4 or A5 or A6 or A7)
if not (B1 or B2 or B3 or B4 or B5)) and
(if (B1 or B2 or B3 or B4 or B5) then B) and
(if not (B1 or B2 or B3 or B4 or B5) then not B).

In contrast, with this change of representation using ordinary logic, the logic programming representation is not affected by this change of interpretation. In the logic program there is no difference between the representation of "if" and the representation of "if and only if". The difference between the two interpretations depends upon whether or not the "closed world assumption" [6] is applied. The closed world assumption for a predicate P is the assumption that all the implications

P if Q1
P if Q2
:
P if Qn

with conclusion P in a program represent all the conditions under which conclusion P holds. It is this assumption that justifies the negation as failure rule:

not P holds if P fails to hold, i.e.
not P holds if all ways of trying to show P result in failure.

Thus, in the example of the lease termination clause, in the case of the word "when", the interpretation "if and only if" is not intended because there are other situations referred to elsewhere in the lease under which the University may terminate the lease with 30 days written notice. But in the case of the words "in which case", the interpretation "if and only if" is intended because there are no other cases under which the University may terminate the lease with 24 hours notice. In the case of the word "unless", the question is not relevant because in the context in which it occurs the closed world assumption is not applicable.

Allen and Saxon argue that the logical representation of the lease termination clause does not express what the drafters must have actually intended. After all the ambiguities have been resolved, the English text expresses that for the University to be able to terminate the lease with 30 days written notice, not only must one of the conditions

(A1 and (A2 or A3)) or A4 or A5 or A6 or A7

hold but none of the conditions

B1 or B2 or B3 or B4 or B5,

under which it may terminate the lease with 24 hours notice, may hold. But these extra negative conditions play no useful role. They serve only to make the conditions under which conclusion holds exclusive of the conditions under which conclusion B holds.

The simpler rules

A if ((A1 and (A2 or A3) or A4 or A5 or A6 or A7)
B if (B1 or B2 or B3 or B4 or B5)

are more flexible. Compared with the original rules they give the university the extra option of giving students 30 days notice in cases where they would otherwise be forced to give 24 hour notice.

Using indentation, and the expressions "both ... and", and "either ... or" in place of parentheses, this new interpretation can be written in a form which arguably has both the precision and simplicity of logic programming and the naturalness of English:

The university may terminate this lease by providing the lessee with written notice of the termination 30 days prior to the effective time of termination
    if both the lessee has applied for and executed
        this lease in advance of enrollment
        and either the lessee is not eligible to enroll
            or the lessee fails to enroll
    or the lessee leaves the university at any
        time prior to the expiration of this lease
    or the lessee violates any provisions of this lease
    or the lessee violates any university regulations
        relative to residence halls
    or there are health reasons for terminating this lease.

The university may terminate this lease by providing the lessee with notice of the termination a minimum of 24 hours prior to the effective time of termination
    if life, limb or property would be jeopardized by
        continuation of the lease
    or the lessee engages in the sale or purchase of
        controlled substances in violation of federal, state or local law
    or the lessee is no longer enrolled as a student
    or the lessee engages in the use or possession of
        firearms, explosives, inflammable liquids,
        fireworks, or other dangerous weapons within the building
    or the lessee turns in a false fire alarm.

The University of Michigan lease termination clause is not a good illustration of our thesis that legal language can be a good guide for improving computer languages. If anything, it seems to suggest the converse, that some computer languages might be a useful guide for improving the language of the law.

In fact, few legal documents are written to the standards of precision found in the acts of parliament; and hardly any legal documents at all are written not only to be precise but also to be clear and easy to understand. However, public notices, which are meant to be understood by ordinary people, are for the most part an important exception to this rule. The London underground emergency notice is a good example of such an exception.

4 The London Underground Emergency Notice

The notice has many characteristics of a logic program, but with some interesting differences:

EMERGENCIES

Press the alarm signal button
to alert the driver.

The driver will stop immediately
if any part of the train is in a station.

If not, the train will continue to the next station,
where help can more easily be given.

There is a £50 penalty
for improper use.

From a knowledge representation point of view, the first sentence is probably the most interesting. Expressed in a procedural style, it shows that a procedural form of expression can sometimes be more appropriate than an "equivalent" statement in declarative style:

You alert the driver
if You press the alarm signal button.

Notice, however, that the procedural form can be regarded as a compiled version of the procedural interpretation of the declarative form. Like most compiled representations of knowledge, it requires less work on the part of the recipient to put the knowledge into effect.

This example and others like it suggest that logic programming could be made more like natural language if it allowed both declarative and procedural syntax. Under the procedural interpretation of logic programming, both the declarative syntax

A if B and C

and the procedural syntax

to do A do B and do C

would be equivalent. In fact both styles of expression would have the same declarative meaning

A if B and C

and the same procedural meaning

to do A do B and do C.

A procedural syntax for logic programs would not, however, include arbitrary imperative programming language constructs. It would not, for example, without further extension, include such purely imperative statements as

press the alarm signal button.

All imperative statements in a logic programming language would have to be imbedded in a procedure, which contains an expression of its purpose. I shall discuss the possible extension of logic programs to include purposeless procedures, viewed as integrity constraints, in sections 5.1 and 5.2.

To simplify the discussion of the emergency notice, I have ignored and, for the most part, will continue to ignore the temporal relationships between the different actions and situations referred to in the notice. We should note however, that to be accurate the title of the notice should be incorporated into the conclusion of the sentence:

press the alarm signal button,
to alert the driver to an emergency.

The second sentence of the notice is explicitly expressed in a logic programming form. However, even allowing for the fact that the phrase

the driver will stop immediately

is shorthand for

the driver will stop the train immediately,

the sentence fails to express its intended meaning, because it is missing an entire condition. The meaning of the sentence can be made explicit, by supplying the missing condition from the conclusion of the previous sentence:

the driver will stop the train immediately
if You alert the driver to an emergency
and any part of the train is in a station.

Certainly this precise expression of the meaning of the sentence is more cumbersome that the English. However, it is hard to see how the logic programming representation could be simplified so that it more closely resembles the English, without loosing its precision.

The third sentence begins with an allusion to the explicitly stated condition of the previous sentence. Ignoring for the moment, the comment at the end, the sentence with all its conditions made fully explicit has the logical form:

the train will continue to the next station
if You alert the driver to an emergency
and not any part of the train is in a station.

But this alone cannot be all that it is intended by the English, because the train will generally continue to the next station whether or not the driver is alerted to an emergency. Surely, what is meant is that the train will stop at the next station and that help will be given there. This is part of the meaning of the phrase

where help can more easily be given.

Moreover, presumably help will be given at a station whether it is the next station or not. Thus we can obtain a better approximation to the intended meaning of the third sentence with the two sentences:

the train will stop at the next station
if You alert the driver to an emergency
and not any part of the train is in a station.

help will be given in an emergency
if You alert the driver to the emergency
and the train is stopped in a station.

This second sentence of the revised formulation of the sentence captures part of the meaning of the comment at the end of the sentence. Presumably the rest of its meaning could be expressed by the meta statement that this procedure for getting help is better than the alternative procedure of stopping the train when it is not in a station.

The last sentence of the notice has a simple formulation in conclusion-conditions form:

there is a £50 penalty
if You use the alarm signal button improperly.

This contrasts with a purely imperative statement, which expresses a prohibition without expressing a purpose:

do not use the alarm signal button improperly.

In contrast with the purely imperative statement of prohibition, the procedural interpretation of the English sentence contains a clear expression of purpose:

if You want a £50 penalty,
then press the alarm signal button improperly!

Notice, by the way, how different the procedural syntax of a sentence can be from its declarative meaning. The English procedural sentence

if You want A, then do B

actually has the underlying declarative meaning

A if B.

Although the English of the London underground notice can be improved, it is undoubtably clear and easy to understand. I believe its clarity is due to at least three characteristics

- the explicit use of conclusion-conditions form

- the appropriate use of procedural form, and
- the use of ellipsis to avoid unnecessarily stating the obvious.

The first two characteristics can usefully be applied to the design and improvement of computer languages today. The third characteristic is harder to achieve, although some progress along these lines might be possible in the future.

5 Other Computing Paradigms

The preceding examples illustrate some of the typical characteristics of legal language and its relationship to logic programming form. It is also possible, however, to find indications of other computing paradigms.

5.1 Condition-Action Production Rules

Condition-action rules were developed by Newell and Simon [19] as a model of human psychology and have been used to implement expert systems [27]. They can also be found in the language of public notices. For example, the following notice is displayed in the carriages of the London underground

Please give up this seat
if an elderly or handicapped person needs it

This is a distinct improvement over the earlier, ambiguous, and potentially disturbing notice

please give up this seat
to an elderly or handicapped person.

But even with the explicit use of the word "if", the sentence falls short of logic programming form, because the apparent conclusion

please give up this seat

is imperative rather than declarative. Moreover the sentence does not express a purpose.

The condition-action form in which the rule is expressed can be converted into logic programming form by making the purpose, e.g.

to do a good deed

explicit rather than implicit. The resulting statement can be expressed procedurally

to do a good deed
give up this seat
if an elderly or handicapped person needs it

or declaratively

You do a good deed
if You give up Your seat to a person
who needs Your seat and
who is elderly or handicapped.

The claim that every command has an explicit or implicit purpose is an important theory in legal philosophy. The use of logic programming form, which forces purposes to be made explicit, is in the spirit of this theory. Associating explicit purposes with commands makes it possible to reason about the relative merits of conflicting commands and even to reason whether a command is appropriate in a given context.

Nonetheless, natural language does allow the expression of commands without purpose, and there even seems to be a logic programming analogue of this in the form of integrity constraints.

5.3 Integrity Constraints

For many years the London underground displayed the following notice above the automatic doors of its carriages

Obstructing the doors causes
delay and can be dangerous.

In other words

there will be a delay
if You obstruct the doors.

there can be danger
if You obstruct the doors.

As long as delay and danger are regarded as undesirable, a thinking person will conclude that obstructing the doors is undesirable too.

But the London underground authorities have recently changed the wording of the notice on some of its trains. The new sign reads

Do not obstruct the doors.

A sad reflection of our changing times. Either delay and danger are no longer regarded as undesirable, or the public cannot be relied upon to reason about the consequences of its behaviour.

But for a logic programmer the new notice is worrying, not only because it indicates the possibly deteriorating state of British underground society, but also because it represents a move away from a logic programming style of communication to a more imperative style. But on closer consideration, the change of wording is reminiscent

of recent efforts to extend logic programming by the inclusion of integrity constraints.

This extension is motivated by database applications of logic programming. For these applications, a number of studies [5, 17, 21, 22] have investigated the nature of integrity constraints in logic programming and the development of efficient integrity checking methods. In all of these approaches integrity constraints are viewed as properties which a database or program must satisfy as it changes over the course of time. To the extent that the contents of a database describe states of affairs in the world, commands, which impose obligations or prohibitions on states of the world, can be interpreted as integrity constraints on states of the database.

An integrity constraint can be expressed in the form of any sentence of first-order logic including a denial. Thus the command

do not obstruct the doors

might be represented by a denial

not You obstruct the doors

which expresses an integrity constraint on descriptions of events which take place in the world.

Similarly the condition-action rule

please give up this seat
if an elderly or handicapped person needs it

might be interpreted as an integrity constraint which has the form of an implication

You give up a seat to a person
if You are sitting in the seat
and the person needs Your seat
and the person is elderly or handicapped.

Thus, given a database that records events that take place in the world, the integrity of the database will be violated if the database records that a person is sitting in a seat which an elderly or handicapped person needs and the database does not contain a record of that person giving up the seat to the elderly or handicapped person. It is another problem, if integrity has been violated, to decide how integrity should be restored. Perhaps this is where "purpose" or "sanctions" might play a useful role.

Thus commands without purpose seem to be compatible with logic programs extended by the inclusion of integrity constraints. Moreover, there is even a transformation between integrity constraints and logic program rules, which is analogous to a transformation between commands without purpose and procedures with purpose:

Given an integrity constraint expressed as a first-order sentence C, introduce a new predicate S and convert the constraint to the rule

S if not C

together with the new constraint

not S.

The new predicate S can be interpreted as a "sanction" which applies if the original constraint is violated. This transformation has been used in the literature on integrity constraints in deductive databases to convert arbitrary first-order integrity constraints into denial form.

The analogy between this transformation and the legal doctrine of sanctions suggest the possibility of adapting legal techniques for dealing with violations of commands to the problem of restoring integrity in deductive databases. This is an intriguing possibility that merits closer investigation.

5.3 Object-Oriented Programming

The paradigm of object-oriented programming has become increasingly important in computing in recent years. It is interesting to investigate, therefore, to what extent it has analogues in natural language and in legislative language more particularly.

We have already seen some characteristics of object-orientation in English when we saw the use of common nouns such as "person", "time" and "lessee" as a kind of object-oriented typing of variables. Other manifestations of object-orientation seem to be more difficult to find in the actual language of legislation, but easier to find both in descriptions of individual cases and in the organisation of law as a whole.

In natural language descriptions, it is common to group sentences together around a single topic placed at the beginning of each of the sentences. Such topics help to organise communication similar to the way in which objects can be used to organise knowledge in computing.

Compare, for example, the pair of sentences

The Prime Minister stepped out of the plane.
Journalists immediately surrounded her.

with the pair

The Prime Minister stepped out of the plane
She was immediately surrounded by journalists.

Psycho-linguists have found that the second pair of sentences is easier to understand than the first, despite the fact that the second pair uses the passive rather than the active voice. The two sentences in the more comprehensible pair have the same topic, whereas the two sentences in the other pair have different topics. Such

examples suggest that organising knowledge around objects makes the knowledge more coherent and easier for humans to understand.

In the domain of law, it is common to organise the different areas of law into hierarchies, which are similar to hierarchies of objects. Thus a country might have one statute governing criminal law in general, another statute covering behaviour in public places, and yet another dealing with behaviour in public buildings. Assault and battery, for example, might be prohibited everywhere, whether in public places or not. Going about naked, however, might be prohibited only in public places, but be allowed in the privacy of one's own home. Smoking, on the other hand, might be prohibited only in public buildings but be allowed everywhere else.

Thus natural language seems to support two notions of objects: objects in the small, which are used like types and topics to organise descriptions of individuals; and objects in the large, which are used in hierarchies to organise whole areas of knowledge. From this point of view, logic programming and object-orientation correspond to different aspects of natural language and are complementary.

However, the notion of object in computing has other characteristics, such as change of state, which do not have such obvious counterparts in natural language. These characteristics seem to be more closely associated with simulating the behaviour of objects in the world than with describing their behaviour.

There have been several attempts to apply object-orientation to legal reasoning. Some of these, like Gordon's Oblog [9], are based on a view of objects as types and topics, which is entirely compatible both with logic programming and with the representation of natural language meanings. Others, like the treatment of patent law by Nitta et al [20] are based on the use of objects to simulate behaviour.

The use of objects for simulation in the patent law example is especially interesting because of the way in which patent procedures, obligations and prohibitions are used to generate and filter changing states of the simulation of a patent application. It seems possible that, if the changing states of the simulation are viewed as database states, then the obligations and prohibitions expressed in the patent law might be viewed as integrity constraints. This possibility would establish an interesting link between imperative statements in object-oriented programming and integrity constraints in deductive databases and logic programming.

No matter what the outcome of a more detailed investigation of these possibilities, there can be little doubt that legislation provides a rich domain outside computing science itself within which relationships between different computing paradigms can be studied. These studies need not be confined to programming languages alone, but could usefully be extended to many other aspects of computing.

6 Other Relationships Between Computing and Law

To the extent that we can truly regard legislation as programs to be executed by people, we can also expect to find analogues in the law of such other computing matters as program specification and software management.

6.1 An Analogy Between Specifications and Policies

In the same way that programs are written to meet specifications, laws are drafted to achieve policies, which are social or political objectives. The purpose of the British Nationality Act 1981, for example, was "to make fresh provisions about citizenship and nationality, and to amend the Immigration Act 1971 as regards the right of abode in the United Kingdom", and in particular to restrict immigration to the United Kingdom by residents of the former British colonies. The purposes of the University of Michigan lease termination clause presumably include such goals as discouraging unsociable behaviour in the halls of residence, restricting residency to legitimate students, and not causing undue hardship for individuals who are obliged to terminate their residence. The rules for dealing with London Underground emergencies, on the other hand, are designed to facilitate the provision of help as effectively and quickly as possible in the case of genuine emergencies and to avoid inconvenience and unnecessary trouble in the case of false alarms.

Program specifications have many characteristics in common with the policies of legal documents. In the same way, for example, that the primary obligation of a program might be to meet its specification, the primary duty of a legal document should be to achieve its social and political objectives. In both cases, moreover, specifications and policies are often ill-defined, inconsistent, or the result of compromise between conflicting demands.

The formal methods developed in computing to verify that programs meet their specifications are much more advanced than any corresponding methods developed for the law. A pilot study of the possibility of adapting formal methods of logic-based software verification to the problem of verifying social security regulations has been made by Bench-Capon [2].

Thus the transfer of techniques for program verification is one area in which the field of law might be able to benefit from its similarities with computing. In other areas, such as software management, the benefits might apply more equally to both fields.

6.2 An Analogy Between Software Maintenance and Maintenance of the Law

In the same way that programs need to be modified to meet changing specifications, legislation needs to be modified to meet changing social and political needs. But programs are both difficult to construct and difficult to change. So much so in fact, that programs are often still in use long after they have outlived their specifications.

The situation is not much better in the law, where legislation often lags far behind social and political changes. Obsolete and incorrect legislation is enforced simply for the sake of "law and order".

But the drafters of legislation have developed some ingenious devices for adapting, modifying and revising old legislation. The liberal use of vague and undefined terms such as "good character", "life, limb or property would be jeopardized" and

"improper use" greatly contribute to the flexibility of legislation and to its ability to adapt to change. Such use of vague terms is reminiscent of the use of data abstraction and encapsulation in computer programming, which allow the lower levels of a program to change, while leaving the higher levels intact.

Much legislation is explicitly concerned with the repeal or amendment of earlier legislation. The British Nationality Act 1981, for example, repeals the British Nationality Acts 1948 to 1965 and amends the Immigration Act 1971. Amendments in particular are typically expressed by metalevel statements which describe how an old piece of text should be edited to create a new text. Metalevel statements are also used to create a new provision from a similar provision in the same act.

Section 6.2 of the British Nationality Act 1981, for example, makes special provision for naturalisation of people who are married to British citizens. The requirements are similar to those for people who apply under section 6.1, but include shorter residency requirements, omit the requirement of having sufficient knowledge of English, Welsh, or Scottish Gaelic, and include

> "the requirement specified in paragraph 1(1)(b)".

This metalevel reference to 1(1)(b) is in fact a reference to the requirement

> "that he is of good character".

This particular use of metalanguage is rather unusual in that the English expression of the metalinguistic form is actually longer than the equivalent object level expression. Usually the metalinguistic formulation is more concise than the object level formulation.

Thus the source code of legislation often mixes object level statements about the domain of discourse with metalevel statements about the text of other legislation or other provisions in the same legislation. The principle objective of using such metalevel statements in preference to equivalent object level statements is to make explicit the relationship between different but similar texts.

The language of legislation also employs remarkable techniques for reusing previous legislation. In the British Nationality Act 1981, for example, it states that one of the conditions for being a British citizen by descent under the 1981 Act is to be a person who

> under any provision of the British Nationality Acts 1948 to 1965,
> was deemed for the purposes of the proviso to section 5(1) of the 1948 Act to
> be a citizen of the United Kingdom and Colonies by descent only,
> or would have been so deemed if male.

The last phrase is an example of a counterfactual condition. A metalogical interpretation of such counterfactuals has been proposed by Bench-Capon [3]. It is possible to imagine how metaprogramming might be used to implement such counterfactual reuse of software in a logic programming environment.

6.3 The Relationship Between Case-Based and Rule-Based Reasoning

In artificial intelligence a contrast is sometimes made between case-based and rule-based reasoning, and a conflict is often held to exist between these two kinds of reasoning [23]. People, it is argued, reason by means of analogies between different cases rather than by means of the deductive application of rules.

The distinction between these two kinds of reasoning also lies at the heart of law. To some extent it is even reflected among the distinguishing features of the two main western legal traditions. Common law systems, such as those in England and the United States, place greater emphasis on reasoning by means of cases. Civil law systems, such as those on the continent of Europe, place greater emphasis on reasoning by means of codified rules. In fact, in both systems of law the two kinds of reasoning interact and complement one another.

In rule-based legislation, for example, case-based reasoning plays a fundamental role in determining the meaning of vague concepts. Previous cases of a concept serve as precedents for new cases.

On the other hand, in case-based legal argumentation, the justification for a decision in a precedent setting case is often expressed in general terms and appeals to general principles. Moreover, authorative restatements of case law effectively reformulate the precedents set in individual cases into general, rule-based form, even though such case-based rules do not have the same binding force as rules in legislation. Indeed it can be argued that there is a natural evolution in the law from reasoning by means of cases to reasoning by means of rules.

7. Conclusion

The similarities between computing and the law seem to cover all areas of computing software. Moreover, the linguistic style in which legislation is drafted combines in one language the expressive power of computer languages for such diverse areas as programming, program specification, database description and query, integrity constraints, and knowledge representation in artificial intelligence. This linguistic style might be a good guide therefore to how these different areas of computing might be unified in the future.

The similarities between computing and law go beyond those of linguistic style. They extend also to the problems that the two fields share of developing, maintaining and reusing large and complex bodies of linguistic texts. Here too, it may be possible to transfer useful techniques between the two fields.

In this paper I have concentrated on similarities between logic programming and legislation. I have indicated several ways in which the language of legislation suggests that the basic model of logic programming can usefully be extended, to include types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning, and procedural notation. I believe that with the aid of such extensions logic programming can provide the foundations for a future, single computer language that will be suitable for all areas of computing in the same way that natural language is suitable for all areas of law.

Acknowledgement

This work was supported initially by the Science and Engineering Research Council and more recently by the ESPRIT Basic Research Action, "Computational Logic". I am especially indebted to my colleagues, Trevor Bench-Capon, Fariba Sadri and Marek Sergot, whose work on legislation and logic programming has provided much of the background for this paper.

References

[1]Allen, L. E., and Saxon, C.S. [1984] "Computer Aided Normalizing and Unpacking: Some Interesting Machine-Processable Transformation of Legal Rules", Computing Power and Legal Reasoning (C. Walter, ed.) West Publishing Company, pp. 495-572.

[2]Bench-Capon, T.J.M. [1987]: "Support for policy makers: formulating legislation with the aid of logical models", Proc. of the First International Conference on AI and Law, ACM Press, pp. 181-189.

[3]Bench-Capon, T. [1989] "Representing Counterfactual Conditionals". Proceedings of Artificial Intelligence and the Simulation of Behaviour (A. Cohn, Ed.) Pitman Publishing Co.

[4]Bowen, K. A. and Kowalski, R. A. [1982]: "Amalgamating Language and Metalanguage in Logic Programming", in Logic Programming (Clark, K.L. and Tärnlund, S.-Å., editors), Academic Press, pp. 153-173.

[5]Bry, F., Decker, H., and Manthey, R. [1988] "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases", Proceedings of Extending Database Technology, pp. 488-505.

[6]Clark, K. L. [1978]: "negation by failure", in "Logic and databases", Gallaire, H. and Minker, J. [eds], Plenum Press, pp. 293-322.

[7]Gallagher, J. [1986] "Transforming Logic Programs by Specializing Interpreters", Proc. of 7th European Conference on Artificial Intelligence, pp. 109-122.

[8]Gelfond, M. and Lifschitz, V. [1990]: "Logic programs with classical negation", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 579-597.

[9]Gordon, T. F. [1987] "Oblog-2 a Hybrid Knowledge Representation System for Defeasible Reasoning" Proc. First International Conference on Artificial Intelligence and Law. ACM Press, pp. 231-239.

[10] H.M.S.O. [1981]: "British Nationality Act 1981", Her Majesty's Stationery Office, London.

[11] Kowalski, R. A. and Sergot, M. J. [1986]: "A logic-based calculus of events", New Generation Computing, Vol. 4, No. 1, pp. 67-95.

[12] Kowalski, R. A. [1989]: "The treatment of negation in logic programs for representing legislation", Proceedings of the Second International Conference on Artificial Intelligence and Law, pp. 11-15.

[13] Kowalski [1990] "English as a Logic Programming Language", New Generation Computing, Volume 8, pp. 91-93.

[14] Kowalski, R. A. and Sadri, F. [1990], "Logic programs with exceptions", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 598-613.

[15] Kowalski, R. A., Sergot, M. J. [1990]: "The use of logical models in legal problem solving", Ratio Juris, Vol. 3, No. 2, pp. 201-218.

[16] Lloyd, J. W. and Topor, R. W. [1984]: "Making Prolog more expressive", Journal of Logic Programming, Vol. 3, No. 1, pp. 225-240.

[17] Lloyd, J. W. and Topor, R. W. [1985] "A Basis for Deductive Database Systems", J. Logic Programming, Volume 2, Number 2, pp. 93-109.

[18] Mitchell, T. M., Keller, R. M. and Kedar-Cabelli [1986] "Explanation-based Generalization: A Unifying View" Machine Learning, Volume 1, pp. 47-80.

[19] Newell, A. and Simon, H. A. [1972] "Human problem solving", Prentice-Hall.

[20] Nitta, K., Nagao, J., and Mizutori, T., [1988] "A Knowledge Representation and Inference System for Procedural Law", New Generation Computing, pp. 319-359.

[21] Reiter, R. [1990]: "On asking what a database knows", Proc. Symposium on Computational Logic, Springer-Verlag.

[22] Sadri, F. and Kowalski, R. A. [1987]: "A theorem proving approach to database integrity", In Foundations of deductive databases and logic programming (J. Minker, editor), Morgan Kaufmann, pp. 313-362.

[23] Schank, R. C. [1983] "The current state of AI: One man's opinion", AI Magazine, Volume 4, No. 1, pp. 1-8.

[24] Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P. and Cory, H. T. [1986]: " The British Nationality Act as a logic program", CACM, Vol. 29, No. 5, pp. 370-386.

[25] Sripada, S. M. [1991] "Temporal Reasoning in Deductive Databases". Department of Computing, Imperial College, London.

[26] Takeuchi, A. and Furukawa, K. [1986] "Partial evaluation of PROLOG programs and its application to metaprogramming", Proc. of IFIP 86, North-Holland, pp. 415-420.

[27] Waterman, D. A. and Hayes-Roth [1978] "Pattern-directed Inference Systems", Academic Press, New York.

Global References

Wason, P. C. (1968) 'Reasoning about a rule', The Quarterly Journal of Experimental Psychology, 20:3, 273 - 281
To link to this article: DOI: 10.1080/14640746808400161