Logic and Modules

Robert Kowalski
Department of Computing
Imperial College London

April 2005

Abstract

In this paper I investigate the implications for modularity of the view that the mind can be understood as a graph of connections (Kowalski 1975, 1979) among sentences expressed in the clausal form of logic. I argue that modularity is the property that such a graph may contain implicit or explicit sub-graphs, with a high degree of connectivity within sub-graphs and a low degree between sub-graphs. I use the computational interpretation of clauses as logic programs, to make a case for clausal logic as the high-level language of thought, which can be executed directly at the logical level or, alternatively, can be compiled into lower-level representation languages. Different modules can be compiled into different lower-level languages, provided their logical connections with other modules are preserved. I also illustrate a possible relationship between connection graphs and connectionism.

What's the problem?

How can we reconcile the apparent need for the mind to be organised into domain-specific, modular components with the undeniable need for the mind to integrate information from diverse domains when the occasion demands? The problem was posed by Fodor (2000), partly in reaction to Pinker's advocacy of mental modules in (Pinker, 1997).

How to solve it?

The modularity problem has sparked a number of responses in addition to those by Pinker himself in (Pinker, 2005a, 2005b). Carruthers (2003, 2004), for example, suggests that, although "the mind is more or less modular in structure, built up out of isolable, and largely isolated, components", the inputs and outputs of the various modules are integrated by means of natural language in the form of inner speech, co-opting the functions of the speech module for this purpose.

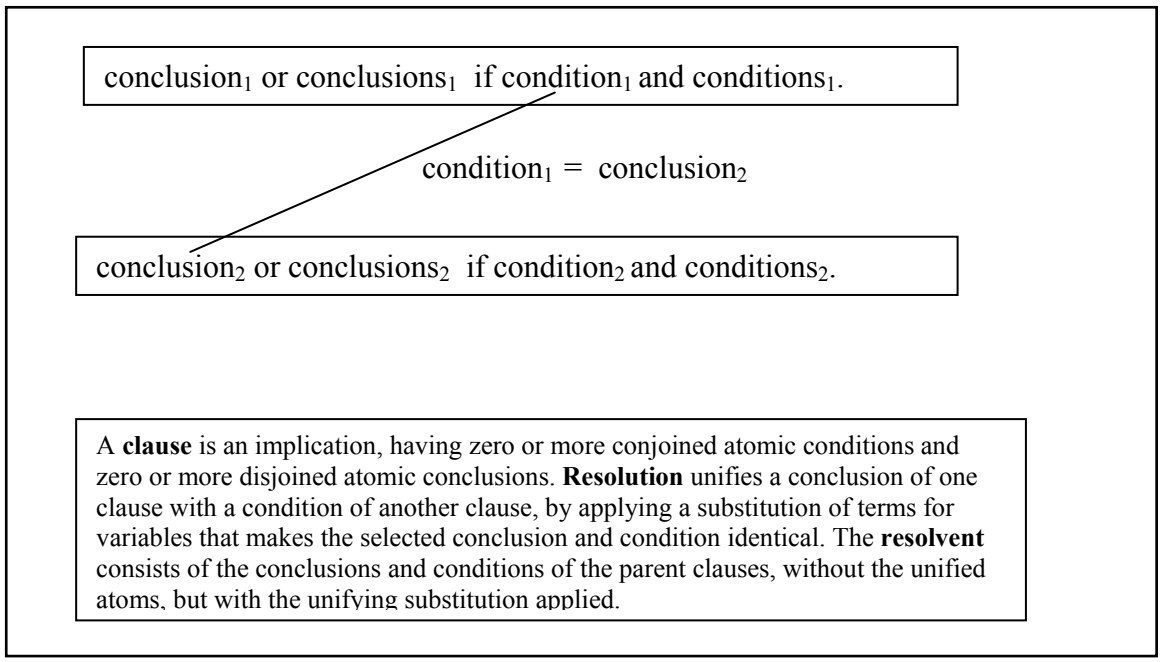
My proposed solution is similar to Carruthers', in that it too proposes that general-purpose, inter-modular thinking is performed in a language of thought. However, I propose that intra-modular thinking can be viewed as also taking place in the same language of thought, and that the language of thought is a simplified form of logic.

My proposal is also similar to Quine's web of belief (Quine, 1963, Quine and Ullian, 1970), in which the mind is viewed as a graph of connections (Kowalski 1975, 1979; Siekmann and Wrightson, 2002) among sentences expressed in a logical form. However, my connections are more specific, namely they are all the possible one step inferences that can be performed between sentences. Moreover, the sentences are in a special, simplified form, called *clausal form*; and the connections between sentences are also a simplified form of inference, called *resolution*.

Connection graphs have a computational, as well as a logical, interpretation: Activating a connection between a clause that represents a goal and a clause that represents a belief is a form of *backward reasoning*, which reduces the goal to sub-goals, similar to way in which a procedure call activates a procedure and invokes other procedure calls. Backward reasoning is the basic idea of logic programming (Kowalski, 1974) and the programming language Prolog.

Connection graphs can also simulate production systems. Activating a link between a clause that represents the record of an observation and a clause that represents a implicational goal is a form of *forward reasoning* (*modus ponens*), which derives a new goal, including the special case of a goal that is an atomic action.

The logic programming interpretation of connection graphs bears on the issue of modularity. Both for the sake of efficiency and because large problem domains are typically composed of smaller domains, connection graphs typically have an implicitly modular structure. Instead of connections being distributed arbitrarily and uniformly throughout the graph, they tend to be clustered in sub-graphs, with a high degree of connectivity within sub-graphs and a low degree between sub-graphs.



The argument for modularity

Perhaps the main argument is that modules are necessary to overcome the computational complexity that would result if the mind were organised as a general-purpose, unstructured web of belief. Without modularity, so the argument goes, every belief would be connected to every other belief, and thinking would become bogged down in a combinatorial explosion. Pinker (1997, p335), as quoted by Fodor (2005, p32), for example, says that "any true belief can spawn an infinite number of true but useless new ones". (In fact, this is untrue for most of the computational logics developed for automated reasoning, including connection graphs.)

An additional argument for modularity comes from neuro-anatomy. The brain, as we all know, is organised into distinct physical areas devoted to such specialised functions as language understanding, memory, vision, etc. The modular structure of the mind is, therefore, just the obvious way in which the mind mirrors the physical structure of the brain.

Other arguments come from psychology, where there is good evidence (as well-documented, for example, by Carruthers (2003)) that people compartmentalise their thinking into distinct domains. So much so, that they may apply a belief in one domain, but fail to apply it in another. The psychological evidence supports the view that there are distinct modules for "physical objects, living things, other minds and artefacts" (Pinker, 2005a, p15), among others. Pinker suggests that the number of modules needed might be "some two dozen emotions and reasoning faculties (distinguishing, for example fear from sexual jealousy from the number sense)" (Pinker, 2005b, p16). Fodor, however, sees the problem as stemming from the view that the mind is "massive modular".

The problem with modularity

Part of the problem is that different people mean different things by the term "modularity". They differ about whether modules are massively many or only moderately few. They differ also about the extent to which information is encapsulated inside modules, and about what such encapsulation might mean. And they differ about whether modules are associated with functions, such as cheater detection, or whether they are associated with domains, such as social interaction.

But all of these notions of module have the same problem: how does the mind combine information from different modules. Fodor, for example, dismisses Pinker's suggestion that "the mind is a network of subsystems that feed each other in criss-crossing but intelligible ways" (Pinker, 2005a, p17) as "entirely without content" (Fodor, 2005, p28). I will try to show that connection graphs can provide Pinker's missing content, although not the sort of content that Pinker himself seems to have in mind.

The problem of combining information from different modules is exacerbated when different modules use different, specialised implementations. As Pinker puts it, the mind "is not made up of mental Spam, but has a heterogeneous structure of many specialised

parts" (1997, p31). However, not only does heterogeneity necessitate transforming information from one representation into another, but it necessitates recognising when different items of information expressed in different representations are relevant to one another and can be combined.

The fox and the crow

The connection graph approach to the modularity problem can be illustrated by a version of Aesop's fable of the fox and the crow. We all know the story: The crow is perched in a tree with some cheese in its beak. The fox is on the ground under the tree and wants to have the cheese. To achieve its goal, the fox needs to combine its beliefs about the physical world, including the laws of gravity, with its beliefs about other minds, including that of the crow.

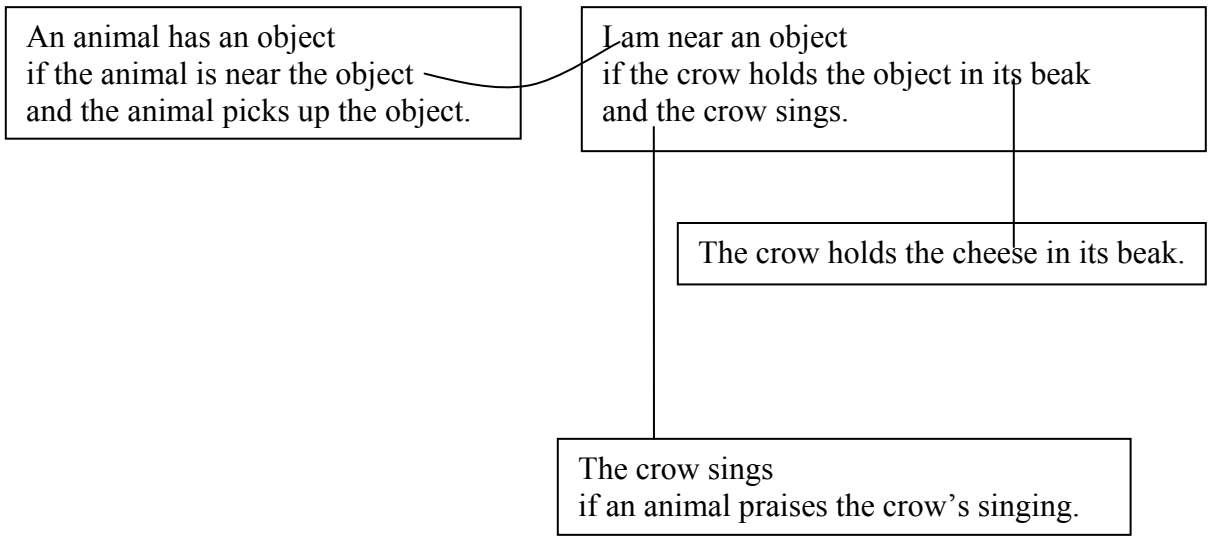
For example, the fox could use the following simplified beliefs to solve its problem:

> The crow holds the cheese in its beak.
>
> An animal has an object
> if the animal is near the object
> and the animal picks up the object.
>
> I am near an object
> if the crow holds the object in its beak.
> and the crow sings.
>
> The crow performs an action
> if an other animal praises the crow's action.

These beliefs are in the form of clauses, with some syntactic simplifications (such as the first occurrence of a typed variable in a clause is signalled by "a" or "an" and subsequent occurrences of the same variable in the same clause are signalled by "the".

The first three beliefs could be regarded as the output of a physical-world module, and the fourth belief as the output of an other-minds module.

The connection graph for the first four clauses (ignoring unifying substitutions and type inferences) has the simple form:



Connections in a connection graph can be selected and the associated resolution steps can be performed in any order. In logic programming they are triggered by a top-level goal, reducing it to sub-goals, and eventually to action sub-goals that are solved by executing them successfully in the environment.

No matter in what order the connections are activated in the connection graph above, there is only one logical consequence that can ultimately be derived (far from the infinitely many imagined by Pinker):

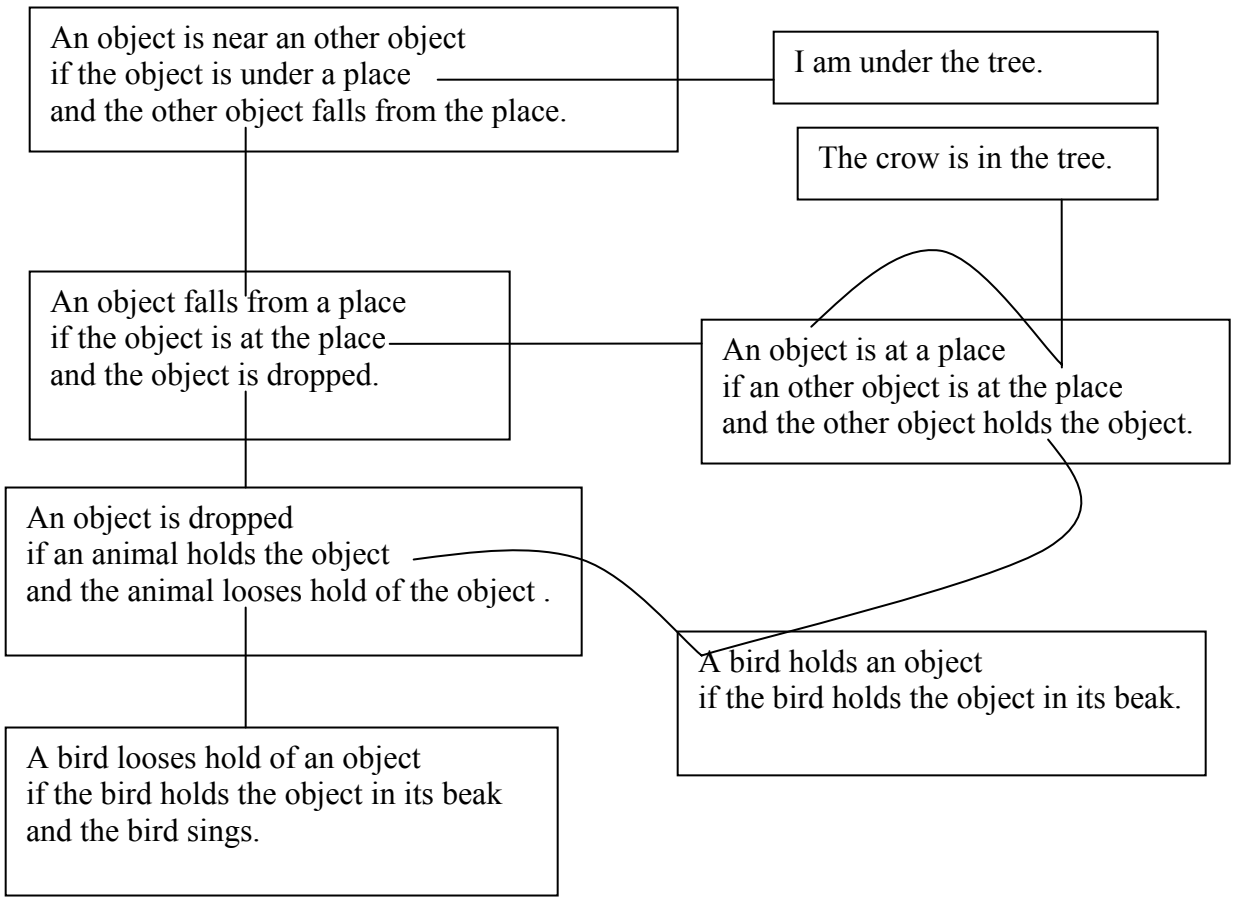
I have the cheese
if an animal praises the crow's singing.
and I pick up the cheese.

This sentence can be viewed as a plan consisting of two actions for the fox to achieve its goal. (For simplicity I have ignored temporal considerations, constraining the praising of the crow to occur before the fox tries to pick up the cheese. I have also ignored the likely fact that the fox will choose to instantiate the typed variable "an animal" with the term "I" representing itself.)

Of course, it is implausible that the fox's initial connection graph would explicitly contain such a specialised belief as:

I am near an object
if the crow holds the object in its beak
and the crow sings.

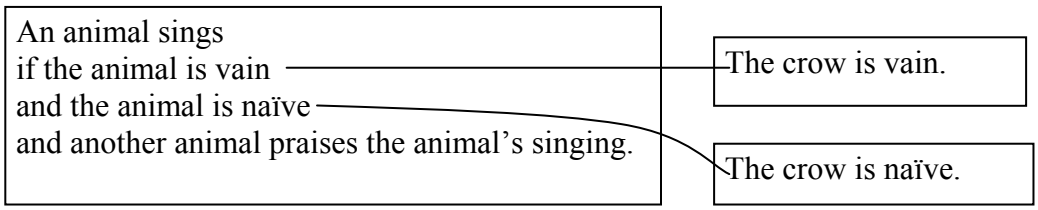
It is more likely that the belief would be implicit, and that the fox would derive the belief from knowledge about its current location and from more general beliefs about the physical world:



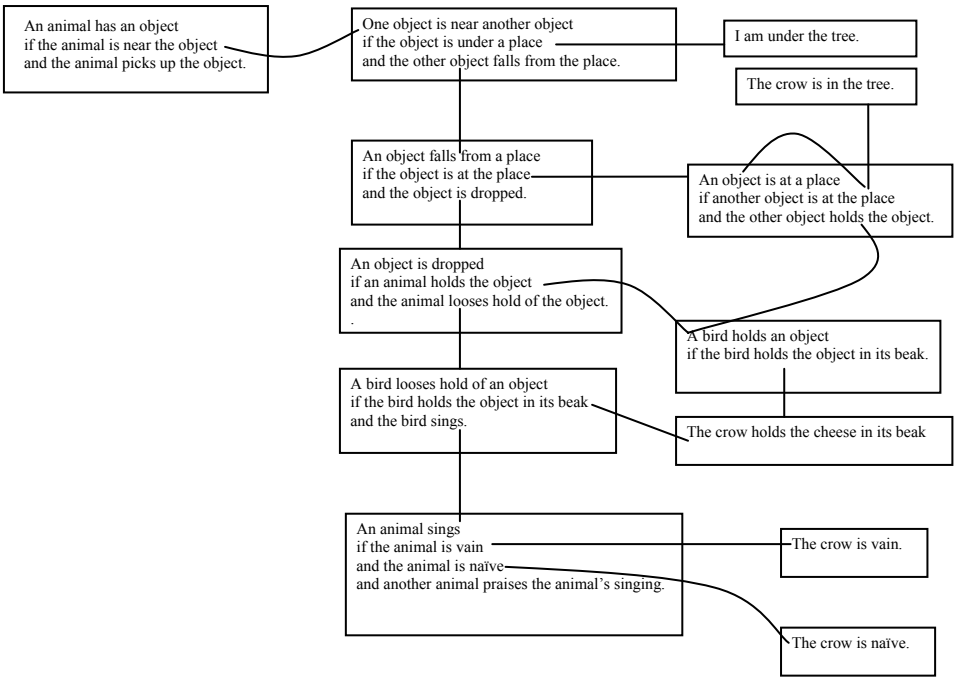
In a similar way, the fox's belief:

The crow sings
if an animal praises the crow's singing.

might also be derived from more general beliefs about animal psychology. For example, from such beliefs as:



The connection graph, which puts all of these beliefs together, looks like this:



Of course, the example is greatly simplified. Not only does it ignore the representation of time, but, more importantly, it ignores the representation of the many other beliefs that are needed for reasoning about the physical and psychological domains. However, as simple as it is, it shows that modules can be implicit, rather than explicit, and that reasoning within modules can be performed by the same logical mechanism as reasoning between modules.

Clausal Logic as the High-level Language of Computation

Viewed in computational terms, the clausal form of logic is a high-level language, which can be executed directly at the logical level by performing resolution inferences, as in

connection graphs for example. Or instead it can be compiled, like any other high-level computer language, into a lower-level representation and then can be executed at that lower level. Indeed compiling one level of representation into a lower level can be iterated any number of levels, ending ultimately in hardware or brainware.

In Computing, higher-level languages have the advantage of being easier to understand, develop and maintain. Lower-level languages have the advantage of greater efficiency. It is common to compile high-level representations into lower-level ones. But it is sometimes possible and desirable to decompile lower-level representations into higher-level ones.

The modular structure that is implicit in some connection-graphs can be made explicit by isolating the internal connections within modules from the internal connections within other modules. This isolation of internal structure encapsulates beliefs within modules, but has no impact on the over-all logic of all beliefs, provided the connections between modules are retained.

Different modules can be compiled into different, heterogenous lower-level representations, taking advantage perhaps of the different syntactic structure of different modules. The use of different lower-level representations need not have any impact on the over-all logic of beliefs, provided they correctly compile their respective higher-level logical representations.

Similar to the way in which simpler forms of life developed before more complex forms, in the history of Computing, lower-level programming languages were developed before higher-level languages. Legacy systems implemented in low-level computer languages often prove difficult to maintain when the environment and the system requirements change. In many such cases, however, it is possible to decompile the lower-level programs into higher-level languages. Sometimes, because of the undisciplined complexity of the lower-level programs, the corresponding higher-level programs only approximate and partially articulate the lower-level ones.

Clausal Logic as the Language of Thought

Logic is not very fashionable in Cognitive Science circles these days. Thagard (1996), for example, in his introductory textbook rates logic a long way behind production systems, concepts, neural networks and other more obviously computational formalisms.

Part of the problem seems to lie in the forms of logic that Cognitive Scientists have considered, and in the lack of consideration they have given to clausal logic in particular. Another part seems to be a lack of awareness of the computational character of clausal logic and its relatives.

Thagard (1996, p45), for example, incorrectly states that “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*.” In fact, all proof procedures for logic have two main components (Kowalski, 1979, p60). One is the set of

inference rules that make up the individual steps of proofs. The other is the search strategy for finding proofs. In connection graphs, for example, search is implicit in the strategy for selecting and activating connections. Different selection strategies implement different search strategies, including breadth-first, depth-first and branch-and-bound. Branch-and-bound, in particular, can be used both to incorporate additional "heuristics" into the search and to search for near-optimal solutions by resource-bounded successive approximation.

For example, in my version of the story of the fox and the crow, the fox might have the additional beliefs

> An animal is near an object
> if the object is at a place
> and the animal goes next to the place.
>
> An animal has an object
> if the animal is near the object
> and another animal has the object
> and the animal takes the object from the other animal by force.

It could use these beliefs to search for an alternative solution to the problem of having the cheese, in this case by going into the tree and taking the cheese from the crow by force. Heuristics and other techniques could be used to guide the search and to try to maximise some desirable characteristic, such as the expected utility of the solution.

Clausal logic divides the inference rules of standard logic into two groups. One group is subsumed by the resolution rule, which can be viewed as the execution mechanism of clausal form. The other group can be viewed as the mechanism that compiles the standard form of logic into clausal form or decompiles clausal form into standard form.

Resolution generalises *modus ponens*, transitivity of implication, and a number of other inference rules of standard logic. It also controls the instantiation of universally quantified variables, restricting instantiation to the most general substitution necessary for unification, avoiding a possibly infinite number of irrelevant instances.

The other group of inference rules of standard logic are needed to convert standard form into clausal form and vice versa. For example, both the inference rule, derive "p" from "not not p", and and-elimination, which derives "p" and "q" from "p and q", generate clausal form from standard form. The use of such inference rules is reminiscent of the use of transformational grammars to generate deep structure from natural language surface structure. In both cases many different, but equivalent sentences are converted into the same canonical form.

And-introduction, which derives "p and q" from "p" and "q", converts clausal form into standard form. Used indiscriminately, and-introduction, like the instantiation of universally quantified variables, gives rise to many useless consequences.

The thinning rule of inference, which derives “p or q” from “p”, is also used to generate standard form from clausal form, spawning an infinite number of useless new beliefs. It is also used for deriving any conclusion from a contradiction. Because thinning is absent from the clausal form of logic, clausal logic behaves as a para-consistent logic. In clausal logic, it is possible to derive useful consequences from an inconsistent set of beliefs without being able to derive everything.

Variations of clausal logic have been used for knowledge representation in Artificial Intelligence, since at least the early 1970s. Often in simplified *Horn clause* form (conclusions with at most one disjunct). Often in more complicated form (with negative conditions, for default reasoning). Often combining object level with meta-level reasoning. Sometimes using the combination of object level and meta-level to represent propositional attitudes, such as goals and beliefs.

In addition to logic programming, the applications of clausal logic include both representations of common sense, as well as representations of domain specific expertise. They also include the representation of the semantics of natural language sentences.

The relationship I have in mind between clausal logic as the language of thought and natural language as a medium of communication is similar to the relationship between concepts and words proposed by Sperber and Wilson (1998). In this relationship, the language of thought, in which mental concepts are represented, is richer and more expressive than the natural languages and other means of communication that one human uses to convey her thoughts to another.

Algorithm = Logic + Control

Connection graphs represent the logic component of an agent’s beliefs. The manner in which connections are activated – in what sequence or in parallel – represents the control component. The combination of logic plus control determines the agent’s algorithmic behaviour, expressed by the equation: *algorithm = logic + control* (Kowalski, 1979). Most of the content of an algorithm is in its logic component.

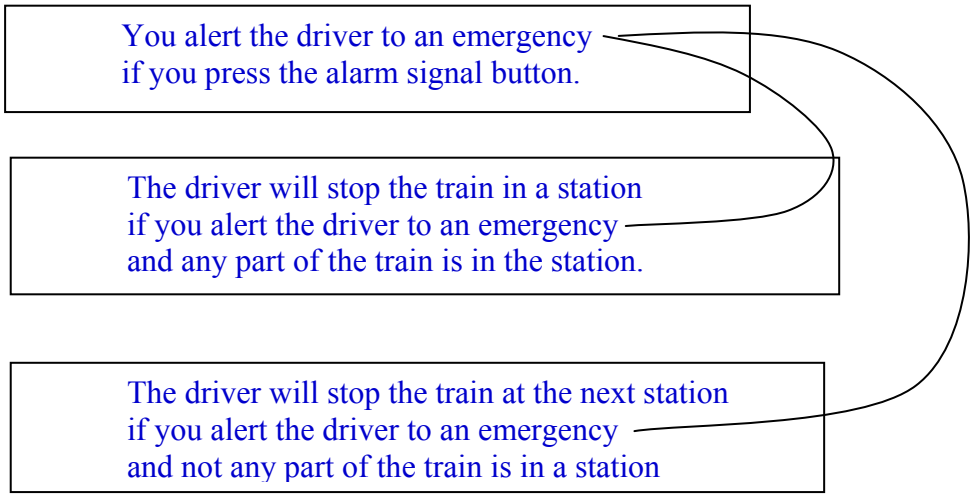
The equation can be used in many different ways. In particular, it can be used to turn a declarative belief into a procedure, by adding a control component; or it can be used to abstract a declarative belief from a procedure, by ignoring the control component.

Frawley (2002) 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 natural language, declarative and procedural expressions of belief often occur side by side, as for example in the following extract from a London underground emergency notice:

> "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,…"

The connection graph representing the underlying logical form of these sentences looks something like this[1]:



The fact that the first natural language sentence is expressed in procedural form and the other two are expressed in declarative form signals that the different sentences are to be used in different ways: the first, to reason backwards, to reduce the goal of alerting the driver to the action sub-goal of pressing the alarm signal button, the other two to reason forwards, to derive and monitor the consequences of the sub-goal.

The first sentence of the Emergency Notice illustrates the relationship between a goal-reduction procedure and its underlying logical form. Sometimes the relationship is not so obvious. Thagard (page 45), for instance, presents the sentence:

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

[1] Strictly speaking, the atomic part of the condition "not any part of the train is in a station" should be moved to the other side of the implication as a disjunct of the conclusion. However, it seems more natural to treat it as "negation as failure".

as an example of a condition-action rule, to support his claim that "unlike logic, rule-based systems can also easily represent strategic information about what to do".

However, in terms of the equation *algorithm* = *logic* + *control*, the same example can also be understood as a goal-reduction procedure, obtained from the *belief*:

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

expressed in logical form, using backwards reasoning as control.

Genuine condition-action rules, which are not goal-reduction procedures, can also be understood in *algorithm* = *logic* + *control* terms. For example, the condition-action rule:

> If I am hungry and I am near an object and the object is food
> then pick up the object and eat the object.

with its imperative conclusion, can be obtained from the *goal*:

> If I am hungry and I am near an object and the object is food
> then I pick up the object and I eat the object.

expressed in logical form, using forward reasoning as control.

In the same way that the Underground Emergency Notice contains both procedural and declarative sentences side by side, connection graphs can contain control pathways that are predetermined together with connections whose controls are determined only at the time of activation.

Sub-graphs whose control pathways are all predetermined and that have few external connections to other parts of the connection graph are candidates for optimisation, by compiling them into modules implemented in lower-level representations. For example, sub-graphs whose connections are all controlled by backward reasoning can be implemented in procedural languages, whereas sub-graphs whose connections are all controlled by forward reasoning can be implemented as production systems. Thus modules implemented in different representation languages can be combined in a single connection graph, having a higher-level, logical structure as a whole.

Natural language parsing is another example of the way in which a combination of logic plus control can be implemented in a specialised, lower-level representation. In 1971, Alain Colmerauer and I discovered that formal grammars can be represented in Horn clause form and that different parsing procedures can be obtained by applying different control strategies (Kowalski, 1979, Chapter 3). Top-down parsers, for example, are obtained by backward reasoning, and bottom-up parsers by forward reasoning. Colmerauer later invented *definite clause grammars*, as a specialisation of Horn clause logic with backwards reasoning and depth-first search, and integrated them into Prolog.

The Pragmatic Nature of Beliefs

Fodor (2005, p30), in his argument with Pinker, states that "cognition *per se* is interested in truth *per se*", proposing that creatures achieve "what they want by a division of mental labor according to which *cognitive* processes are specialised to deliver truths and *decision* processes are specialised for figuring our what to do in the sort of world that cognition reports".

The logic-based agent model we have developed (Kowalski and Sadri, 1999; Kowalski, 2001) supports Fodor's distinction between thinking and deciding, as separate components of an agent's observation-thought-decision-action cycle. However, it sides with Pinker in the view that "the belief system is prone to systematic error in circumstances where fixing true beliefs leads to lower fitness than satisficing or self-deceiving with false beliefs" (2005, p36).

The role of logic in our agent model is to help the agent achieve its goals in response to changes in its environment. For this purpose, its beliefs, expressed in logical form, need to be both *effective* in yielding successful responses and *efficient* in generating these responses in real time. In this role, true beliefs are normally more effective than false beliefs.

However, the idea that the top priority of logic is to represent the truth was one of the main reasons for the decline of logic programming in the 1990s. In its heyday in the 1980s, many enthusiasts believed you could simply write down the truth about a problem domain, feed it to Prolog, and Prolog would then be able to solve any problem in that domain.

Unfortunately, it doesn't always work that way. To write a program that works, it needs not only to be effective, in the sense of satisfying its specification, but it also needs to be efficient – which in many cases means that you need to write an efficient algorithm, instead of a program specification.

To some extent, the equation, *algorithm* = *logic* + *control*, may have given the wrong impression – that all you need to do is to write down the truth in the logic component and adjust the control until you get an efficient algorithm. The problem is that there are many domains in which no control of the logical specification yields acceptable efficiency. John McCarthy (1990), for example, showed this in detail for the map colouring problem.

To obtain acceptable efficiency, it is necessary to use both an appropriate logic and a suitable control. For example, as is well know in Computing, there are many ways to sort a list. One way is just to execute the specification of sorting as finding an ordered permutation of the list. But there is no way to control this specification and obtain acceptable efficiency. It is necessary, instead, to use some other formulation of the logic of the problem, corresponding, for example, to the quick-sort or the merge sort algorithms.

In many applications, when the problem is inherently difficult (say, exponentially hard), there may not exist any algorithm that completely and correctly implements the specification with acceptable efficiency. In such cases, it is often possible to employ heuristics instead. Heuristics are rules of thumb, which may be neither correct nor complete, but which are useful in many cases. The use of heuristics is a common way of building expert systems. The equation *algorithm = logic + control* includes the special case of heuristics as a special kind of algorthim.

Logic is concerned with the *form* of goals and beliefs, but not with their *contents*. It can be used to represent not only truth and specifications, but also the declarative content of conventional algorithms and heuristics. It treats all contents in the same way and enables them to be used for deriving logical consequences.

Modularity is a property of the logic component of algorithms. It contributes to efficiency in two ways. First, it limits the size of the search space for many problems, by confining the search to modules of manageable size. Second, it facilitates parallel processing, by enabling it to take place in different modules simultaneously with minimal or no interference. However, on its own, as a way of achieving efficiency, it is neither necessary nor sufficient.

Modularity, object-orientation and domain-specificity

Modularity, whether explicit or implicit, is associated with the natural decomposition of problem domains into sub-domains. However, in Computing, this decomposition is typically multi-layered, the different layers forming a hierarchy of domains and sub-domains. Such hierarchies are a defining characteristic of object-orientation, which is the dominant paradigm in Computing today.

The relationship between computational logic and object-orientation has been a topic of concern for many years. Suffice it to say here that, as far as the modularity of the mind is concerned, the lesson of object-orientation seems to be that mental modules, whether explicit or implicit, whether represented in logical form or in some other, more conventional form, might also be structured in multi-layered hierarchies.

This hierarchical structure of modules reflects the natural hierarchical structure of problem domains and implies a corresponding hierarchical classification of algorithms and beliefs. In contrast to the distinction often made between domain-specific and domain-general beliefs and procedures, it implies a more refined distinction between beliefs of varying levels of specificity and generality.

Except for beliefs at the lowest level of the hierarchy, all beliefs are of necessity domain-general to some degree. In the same way, unless there is a single top-most level of the hierarchy, rather than a number of top-most levels, all beliefs are also of necessity domain-specific. Thus, to take the example of Atkinson and Wheeler (2204, p155), the belief “that two physical objects cannot take up the same portion of space at the same time… looks like a paradigmatic example of genuinely domain-general information”. But

even it is specific to the domain of physical objects, and does not apply to non-material things.

In my proposal, only the connection graph mechanism, with resolution as its single rule of inference and with selection of connections as its single method of control, is entirely domain-general. Because *modus ponens* is subsumed by resolution, *modus ponens* is also domain-general, which is as Fodor (2000) maintains and Sperber (2002) denies.

Connection graphs and Connectionism

I have been deliberately vague about the detailed workings of the connection graph proof procedure. This is partly because many different implementations are possible, including, as I already mentioned, both sequential and parallel implementations, and partly because not all of the possibilities have been explored.

The standard implementation repeatedly

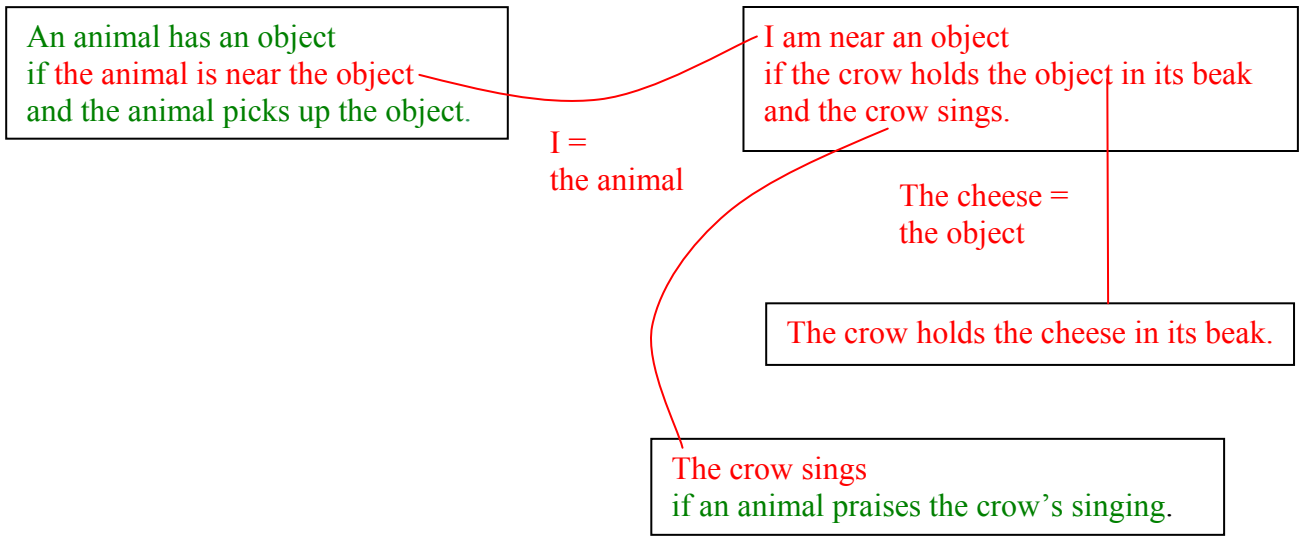
- selects a connection,
- deletes the connection,
- adds the associated resolvent to the connection graph,
- adds connections between the resolvent and other clauses in the graph, (inheriting them from the connections of the parent clauses), and
- deletes any clauses (starting with parent clauses) containing unconnected atoms.

In general, a connection graph increases in size, when new resolvents and their connections are added. It decreases in size when clauses are deleted. In some cases, deleting a clause causes a chain reaction of deletions, which reduces the size of the graph dramatically. In any case, adding a new clause to a connection graph does not generally spawn the infinite number of consequences, which Pinker associates with logical reasoning.

Although the standard implementation adds resolvents explicitly, it is also possible to represent them implicitly, as is done, for example, in implementations of Prolog. For example, the resolvent

I have the cheese if an animal praises the crow's singing and I pick up the cheese.

can be represented implicitly by simply marking all the connections that were activated in the derivation of the resolvent with the same "colour". The resolvent implicitly represented in this way is the collection of atoms left over when all the atoms joined by the marked connections ignored with their unifying substitutions applied:



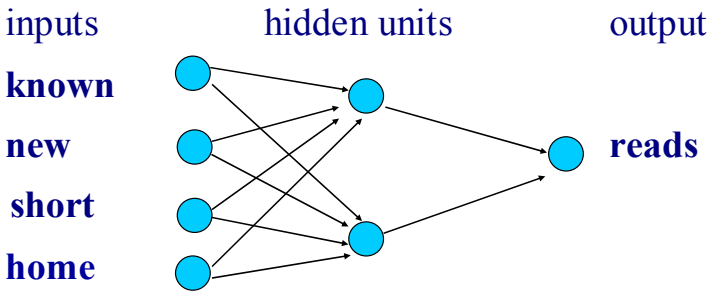
Connection graphs resolvents represented implicitly are similar to connectionist architectures of the mind.

Consider the following example of a neural network taken from the textbook, *Computational Intelligence: A Logical Approach*, by Poole, Mackworth & Goebel (1998). The network simulates a user's decision whether or not to read an article depending upon whether the author is known or unknown, the article starts a new thread or is a follow-up, the article is short or long, and the user is at home or at work.

Poole *et al* present the example as an illustration of a neural network with hidden units, which do not have a symbolic interpretation.

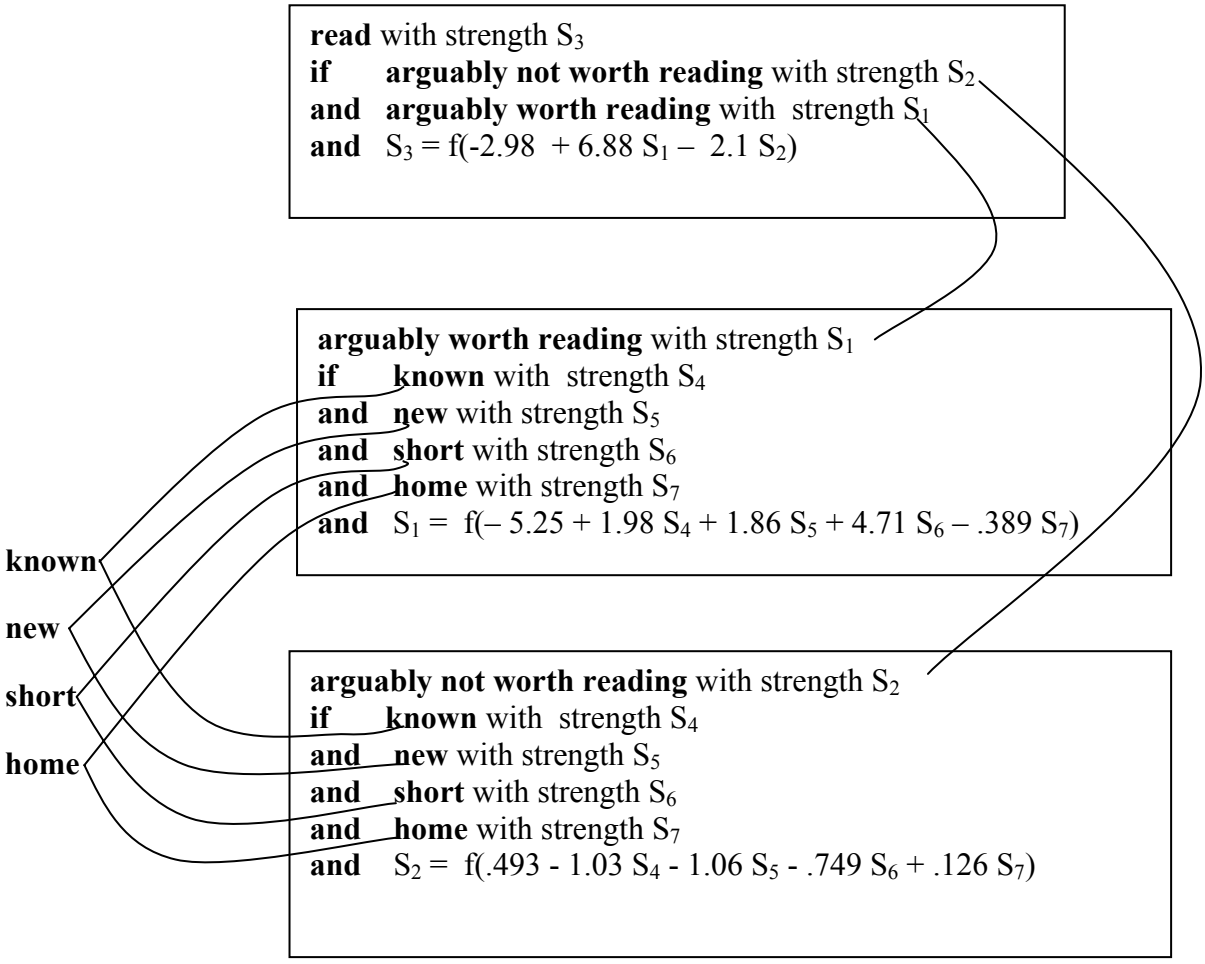
Example	Action	Author	Thread	Length	Where read
E1	**skip**	**known**	**new**	**long**	**home**
E2	**reads**	**unknown**	**new**	**short**	**work**
E3	**skips**	**unknown**	**follow-up**	**long**	**work**

Neural network



51/27

They also present a logic program that corresponds to the neural network. The following connection graph contains a variant of their program, with weights that are learned after backwards propagation using a set of training examples, and with "meaningful" predicate names. Here "f" is the sigmoidal 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*.



The neural network example and its corresponding logic program and connection graph not only illustrate a possible relationship between connectionism and connection graphs, but they also illustrate a possible relationship between the clausal logic as the language of thought and natural language. They show, in particular, that clausal logic can represent mental concepts that can only be approximated in natural language. They also show that it can represent concepts, such as "arguably worth reading" and "arguably not worth reading" that do not have a direct interpretation in the external world.

Conclusion

The proposals put forward in this paper build upon work in three closely related, but independent research areas: automated reasoning, logic programming and logic-based agents. The area of automated reasoning contributes the connection graph proof procedure, including the use of clausal form. Logic programming shows how computational and logical models of the mind can be combined. And logic-based agent models show how logic can serve as the thinking component of an intelligent agent interacting with its environment.

Because these research areas are semi-autonomous, there are issues at the boundaries that remain to be resolved. I have already referred to one of them in a footnote, namely that some form of default reasoning, as found in negation as failure in logic programming, needs to be incorporated in connection graphs. One way to do so is to use the assumption-based approach to default reasoning of Dung, Kowalski and Toni (2005), which can be used to extend any monotonic logic into a dialectic argumentation system.

Perhaps even more importantly, the connection graph proof procedure needs to be extended to incorporate a greater distinction between goals and beliefs. Some of the related issues concerning goals and beliefs are being investigated in (Kowalski, 2005).

In addition, relationships to other disciplines need to be investigated further - the most obvious being the relationship between connection graphs and connectionism.

References

Atkinson, A. & Wheeler, M. 2004: The grain of domains: The evolutionary psychological case against domain general cognition. *Mind and Language*, 19(2), 147-176.

Carruthers, P. 2003: On Fodor's Problem. *Mind & Language*, 18(5), 502-523.

Carruthers, P. 2004: Practical Reasoning in a Modular Mind. *Mind & Language*, 19(3), 259-278.

Dung, P.M., Kowalski, R. & Toni, F. 2005: Dialectic proof procedures for assumption-based, admissible argumentation. To appear in *Artificial Intelligence*. Online at http://www.doc.ic.ac.uk/~rak/.

Fodor, J. 1983: The modularity of mind: An essay on faculty psychology. Cambridge, MA: Bradford/MIT Books.

Fodor, J. 2000: The Mind Doesn't Work That Way, Cambridge: MIT Press.

Fodor, F. 2005: A reply to Steven Pinker “So How Does the Mind Work?” *Mind & Language*, 20(1), 33-38.

Frawley, W. 2002: Control and Cross-Domain Mental Computation: Evidence from Language Breakdown. *Computational Intelligence,* 18(1), 1-28.

Hitzler, P., Hoelldobler, S. & Seda A. 2004: Logic programs and connectionist networks. *J. Appl. Logic*.

Kowalski, R. 1974: Predicate Logic as Programming Language. In *Proceedings IFIP Congress,* Stockholm, North Holland Publishing Co., 569-574.

Kowalski, R. 1975: A Proof Procedure Using Connection Graphs, *JACM*, 22(4), 572-595.

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

Kowalski, R. 2001: Artificial intelligence and the natural world *Cognitive Processing*, 4, 547-573. Online at http://www.doc.ic.ac.uk/~rak/.

Kowalski, R. 2005: How to be artificially intelligent. Online at http://www.doc.ic.ac.uk/~rak/.

Kowalski, R. and Sadri, F. 1999: From Logic Programming towards Multi-agent Systems. *Annals of Mathematics and Artificial Intelligence*, 25, 391-419.

Mc Carthy, J. 1990: Coloring Maps and the Kowalski Doctrine. *In Formalizing Common Sense: Papers by John McCarthy.* V Lifschitz Ablex, Norwood, NJ.

Pinker, S. 1997: *How the Mind Works.* New York: Norton.

Pinker, S. 2005: So how does the mind work? *Mind & Language*, 20(1), 1-24.

Pinker, S. 2005: A reply to Jerry Fodor on How the Mind Works. *Mind & Language*. 20(1), 33-38.

Poole, D., Mackworth, A, & Goebel, R. 1998: *Computational Intelligence: A Logical Approach*. Oxford University Press.

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

Quine, W. O. and Ullian, J. 1970: *The Web of Belief.* NY: Random Nord.

Siekmann, J. and Wrightson, G. 2002: An Open Research Problem: Strong Completeness of R. Kowalski's Connection Graph Proof Procedure. *Logic Journal of the IGPL.*

Sperber, D. 2002: In defense of massive modularity. In Dupoux, E. *Language, Brain and Cognitive Development: Essays in Honor of Jacques Mehler*. Cambridge, Mass. MIT Press. 47-57.

Sperber, D. and Wilson, D. 1998: The mapping between the mental and the public lexicon. In P. Carruthers & J. Boucher (eds) *Thought and language*. CUP, Cambridge, 184-200.

Thagard, P. 1996: *Mind: Introduction to Cognitive Science.* Massachusetts Institute of Technology, Bradford Book.