Using Uncertainty Within Computation

Overview

To reason about complex computational systems, researchers are starting to borrow techniques from the field of uncertainty reasoning. In some cases, this is because the algorithms contain stochastic components. For example, Markov decision processes are now being used to model the trajectory of stochastic local search procedures. In other cases, uncertainty is used to help model and cope with the stochastic nature of inputs to (possibly deterministic) algorithms. For example, Monte Carlo sampling is used to deal with uncertainty in game playing programs, whilst probability distributions are used to model variations in runtime performance. Uncertainty and randomness have also been found to be a useful addition to many deterministic algorithms. And a number of areas like planning, constraint satisfaction, and inductive logic programming which have traditionally ignored uncertainty in their computations are waking up to the possibility of incorporating uncertainty into their formalisms. The goal of this workshop is to encourage symbiosis between these different areas.

Topics

The aim is to bring together researchers from a number different areas of AI including (but not limited to) agents, constraint programming, decision theory, game playing, knowledge representation and reasoning, learning, planning, probabilistic reasoning, qualitative reasoning, reasoning under uncertainty, and search. Possible topics include (but are not limited to):

- Incorporating uncertainty into existing frameworks
- Modelling uncertainty in computation
- Monte Carlo sampling
- Probabilistic analysis and evaluation of algorithms
- Randomization of algorithms
- Stochastic vs. systematic algorithms
- Utility and computation

Handling Uncertainty with Active Logic

M. Anderson[a], M. Bhatia[b], P. Chi[b], W. Chong[c], D. Josyula[c], Y. Okamoto[ad], D. Perlis[ac], K. Purang[c]
a: Institute for Advanced Computer Studies, University of Maryland, College Park MD 20742
b: Department of Computer Science, Bowie State University, Bowie MD 20715
c: Department of Computer Science, University of Maryland, College Park MD 20742
d: Department of Linguistics, University of Maryland, College Park MD 20742
{mikeoda,yuan,darsana,yoshi,perlis,kpurang}@cs.umd.edu

Introduction

Reasoning in a complex and dynamic world requires considerable flexibility on the part of the reasoner; flexibility to apply, in the right circumstances, the right tools (e.g. probabilities, defaults, metareasoning, belief revision, contradiction-resolution, and so on). A formalism that has been developed with this purpose in mind is that of active logic. Active logic combines inference rules with a constantly evolving measure of time (a 'now') that itself can be referenced in those rules. As an example, $Now(6)$ [the time is now 6] is inferred from $Now(5)$ since the fact of such inference implies that (at least one 'step' of) time has passed.

From this feature come others, most notably:

- Ignorance-assessment amounts to a lookup at time t, of what was known prior to t.
- Contradictory information can (sometimes) be detected and used to curtail nonsensical inferences as well as to initiate repairs.
- Default conclusions can be characterized in terms of lookups to see whether one has information (directly) contrary to the default.
- Reasoning can be kept current, i.e., inferences can be triggered to occur when they should, and this itself is done declaratively so that it is also under control of (easily modifiable) inferences.

These features of active logic provide mechanisms to deal with various forms of uncertainties arising in computation.

A computational process P can be said to be *uncertain* about a proposition (or datum) X if

(i). it explicitly represents X as in the knowledge base (KB) but possibly a mistake;

(ii). it represents X as in the KB and initially correct but possibly no longer correct;

(iii). it is aware of X (and/or $\neg X$) – that is, X is a closed subformula of at least one item in the KB – but X is not present in the KB as a belief; or

(iv). X is known to be an item it cannot compute or infer. (This last case is often undecidable in its fullest form; active logic provides a convenient shortcut that we will return to below.)

Uncertainties of type (i) above lend themselves to representation by probabilistic reasoning, which involves the representation of explicit confidence levels for beliefs, for example, Bayesian Networks; and somewhat less so for type (ii); and even less for types (iii) and (iv). On the other hand, a suitably configured default reasoner (non-monotonic approaches) can represent all of these, and without special *ad hoc* tools; that is, active logic already has, in its time-sensitive inference architecture, the means for performing default reasoning in an appropriately expressive manner. It is the purpose of this paper to elaborate on that claim; the format consists of an initial primer on uncertainty in active logic, then its current implementation (Alma/Carne), existing applications, and finally a discussion of potential future applications.

Probabilistic and Non-monotonic Reasoning

Probabilistic approaches (Pearl 1988; Ramoni & Riva 1994) are very useful in reasoning with uncertainty; they can smoothly handle inconsistent inputs, and model belief change over time as evidence accrues, by adjusting the probabilities attached to beliefs and their connections. However, in a Bayesian net for instance, because the probabilities have a somewhat holistic character, with the probability of a given proposition depending not just on direct but indirect connections, it looks like adding new propositions or rules (connections between nodes) will be expensive and potentially require re-calculation of all connection weights. If one's world-model is well specified enough that reasoning about and interacting with the world is primarily a matter of coming to trust or distrust propositions already present in that model, a Bayesian net may provide a good engine for reasoning. However, if one's world-model is itself expected to be subject to frequent change, as novel propositions and rules are added (or removed) from one's KB, we think that a reasoning engine based on active logic will prove a better candidate.

In addition, and partly because a Bayesian net deals so smoothly with inconsistent incoming data, it can operate on the assumption that incoming data is accurate and can be taken at face value. Although of course it is not expected

that *all* incoming data will be accurate (for instance, it is expected to contain noise), it is expected that the system will get reliable inputs overall. We have two related concerns about this: first, an abnormally long string of inaccurate data – as might be expected from a faulty sensor or a deliberate attempt at deceit – would obviously reduce the probability of certain beliefs that, were the data known to be inaccurate, would have retained their original strengths. It has been suggested to us that one could model inaccurate incoming information by coding child nodes that would contain information regarding the expected accuracy of the incoming information from a given evidence node. This seems adequate when it is known at the outset that a given sensor operates with a certain reliability; but it is not clear how one might *learn* that an information source is unreliable, as one might wish to be able to do if a sensor breaks or a source begins lying. Second, it seems that in a Bayesian net, all beliefs are similarly sensitive to incoming data (if they are sensitive to it at all) in the sense that the net operates by a slow erosion or confirmation of probability. But it is not clear that all beliefs should fit this model; one can retain full confidence in a given belief for a long time in the face of continuing empirical evidence to the contrary, and then in light of a single further fact (which alone would not have caused this event) give up the belief entirely. (See (Bratman 1999) for a discussion of belief modeling in light of such considerations.)

Further, insofar as a Bayesian net is operating smoothly, the fact that incoming data contradicts currently held beliefs, or other incoming data, need not be explicitly recognized. But we believe that the recognition of contradiction should be a central and important part of information handling (Perlis 1997). For it seems that there are cases where one can learn from the fact of a contradiction (where the belief that there has been a contradiction can be useful in the reasoning process), as for instance in coming to the conclusion that there is a system malfunction. Although it is no doubt possible to modify a Bayesian net to explicitly encode the occurrence of a contradiction, it is less clear what use this information would be within that schema. And this brings us to our final reason for preferring non-monotonic approaches: active logic has much greater expressive power, including not just the ability to encode complex propositions, but also functions and quantification.

Let us consider the following example: The belief $Yellow(tweety)$ may be based on theoretical considerations that should not necessarily be weakened by indirect evidence, e.g. $TweetyBird(tweety)$, $If\ TweetyBird(X)\ then\ Yellow(X)$. In such case one will at least want to *consider* his reaction to a photograph of a green Tweety. The complexity of the case is not captured by deciding how to treat the veridicality of photographs in general (for assigning a lower probability to this sort of evidence just means it will take longer for photographic content to register in the KB, and this will not always be appropriate); the issue is one of coming to an intelligent assessment of a given piece of evidence (photograph or no) in light of current beliefs and their parents (if any – without theoretical considerations supporting a given belief we may adopt the new evidence at face value; with very strong empirical evidence, we might disbelieve the theoretical bases.) It looks as though in many cases we will want to make *decisions* about this, rather than letting things play out according to pre-determined confidence measures. On the other hand, it seems important that we recognize and remember the (apparent) contradiction, even when we resolve it by distrusting the photographs. For a system which is fully epistemically open to the world – that is, which is capable of changing its world model by adding or deleting rules and facts, and for which every belief in the world model is empirically sensitive (no dogmatic beliefs) – may encounter evidence that directly contradicts $TweetyBird(tweety)$. In such case, among the considerations motivating a decision to accept the new evidence may be the fact that it allows acceptance of the previously discarded photographic evidence of Tweety's greenness.

Primer on Active Logic and Uncertainty

Currently, active logic does not explicitly represent confidence levels of the KB (although it certainly can be made to do so). Instead, it has the flexibility to distrust any of its beliefs in the presence of suitable counter evidence. In effect, active logic treats its current beliefs as simply true until such time as reasons arise for doubting them, and then distrusts them until such time as they may be reinstated. One can thus regard (the current version of) active logic as a kind of time-sensitive nonmonotonic reasoning engine.

Two of the principal mechanisms that provide the flexibility of active logic, especially in regard to uncertainty, are *contradiction-detection* and *introspection*.

Contradiction-detection: If P and $\neg P$ are both in the KB at any step, then both become distrusted in the next step and a repair process is initiated (which may or may not be conclusive). Such a distrust and repair process can occur in cases (i) and (ii). For instance, if P is believed originally, but later $\neg P$ is either derived or observed (this could come about for various reasons: P might always have been false, and the belief that P mistaken; P might have become false; $\neg P$ could be false, and this new belief could therefore be a mistake), then a conflict occurs between P and $\neg P$. This would cause active logic to enter a state of "uncertainty" with respect to P and $\neg P$ leading to distrust both P and $\neg P$ and to initiate a repair process to adjudicate between the 'old' P and 'new' $\neg P$. (Unlike most belief-revision formalisms, active logic does not automatically assume that the newest data is more accurate). The repair process involves the identification and distrust of the parents (and any derived consequences) of the contradictands; reasoning about the parents; and possible reinstatement of one or another set of parents, which may allow one of the original contradictands to be re-derived.

Returning to our $Yellow(tweety)$ example, in the normal case such a belief would be taken (and used) simply at face value, even though it may (unknown to the reasoner) be incorrect, until such time as counterevidence appears (the photo showing a green Tweety). At that time, a contradiction would occur between $Yellow(tweety)$ and $\neg Yellow(tweety)$ (itself derived from $Green(tweety)$ and rules about how colors inhere in objects). Assessment would discover and distrust the parents of each contradictand, and attempt to discern which beliefs it ought to reinstate, as for

instance by utilizing preferences for some beliefs over others (the whole set need not be accepted or rejected together). Thus, assessment may lead to rejection of the initial belief; or it may lead to its reinstatement, and rejection of the photo data (not just if it had a preference for the theoretical bases of $Yellow(tweety)$, but also, for instance, if it knew that the photo was developed poorly, or taken in green light). (This instead of assuming confidences for various of the data items and combining them into a revised confidence measure for $Yellow(tweety)$.)

Introspection: Another mechanism that provides the flexibility of active logic is its ability to note that it does not have a given belief P, represented as $\neg Know(P)$ and $\neg Know(\neg P)$. This ability can be applied to encode uncertainties in uncertainties of type (iii) above. Here it is crucial that $Know$ is interpreted as "currently-in-the-KB", and not as (an often undecidable) "possible-to-derive." Thus an intractable or even uncomputable problem is replaced with a simple lookup in the (always finite) KB. In the above yellow bird example, this can be used as part of a default, to wit: "if something looks yellow and if I (currently) have no knowledge that there is an irregularity about the situation, then I will conclude that it in fact is yellow."[1] Later, if the conclusion that Tweety is yellow is found to be problematic (e.g., it conflicts with other data) that conclusion can be retracted (or precisely, disinherited at subsequent time steps, since the actual inferential history is preserved).

Alma/Carne

Alma is our current implementation of active logic and *Carne* is a process that executes non-logical computations independent of Alma steps.

Alma: At each step, Alma applies the rules of inference to the formulas in the database at that step to produce a set of new formulas. These are added to the database, and the process repeats at each subsequent step. Some characteristics of Alma are:

- The current step number is represented in the KB as $now(T)$. Formulas can be written using the step number which makes it possible to reason about the current time.
- Alma maintains information about various properties of the formulas in the database, including the derivations of the formulas, their consequences and the time at which they were derived; indeed, the entire inferential history is preserved. This information is available for reasoning through reserved predicates.
- The formulas in the KB have names which allow the user to assert properties of the formulas and to reason about these properties. One can for instance, assert that a particular formula is to be preferred over another; that its probability is q; etc.
- If ϕ and $\neg\phi$ are present in the KB where ϕ is a literal, this fact is detected by the contradiction-detection rule. The outcomes of a contradiction between formulas ϕ and $\neg\phi$ named $N1$ and $N2$ are:[2]
  - A formula of the form `contra(N1, N2, T)` is added to the database where T is the step number at which the contradiction has been detected.
  - The contradictands and their consequences are "distrusted" so that they cannot be used for further inference but can be reasoned about.
  - Formulas of the form `distrusted(N)` are added to the database where N is the name of a formula that is distrusted.

  One can specify axioms to reason about the contradiction and decide which of the formulas, if any, to reinstate. Alma provides the reserved predicate $reinstate(N)$ for that purpose.
- Some computations that need to be done in the logic may be more easily, conveniently or efficiently done through procedures. To enable this, prolog programs can be specified as inputs to Alma. These can be invoked when needed through the formulas. An alternative for longer running procedures is Carne (see below).
- Alma can operate in both the forward and backward chaining modes. The usual mode of operation for Alma is in the forward direction. Alma also allows one to do backward chaining to find whether some specific formula is derivable. This is also done in a step by step fashion, just as for forward chaining.

Carne: Carne is a process that communicates with Alma but runs independently. The main use of Carne is to run non-logical computations asynchronous from Alma steps. One of its roles is to serve as an input-output interface to Alma. In this case Carne transforms external input to a form suitable for addition to the Alma KB and conversely.

Alma formulas can request computations to be done by Carne by asserting $call(X, Y, Z)$ in the database. This will trigger the program X in Carne. When the request is sent to Carne, $doing(X, Y)$ is asserted in Alma to record that the action has been started. When Carne returns with an answer, $doing(X, Y)$ is deleted and $done(X, Y)$ is added. If the action fails, we have $error(X, Y)$ replacing the $doing(X, Y)$.

Carne can add and delete formulas directly in Alma. This enables external inputs to be added to the Alma database whenever they become available.

Carne interacts with external processes and with the user at standard input and output. A KQML parser converts input to a form suitable for further processing in prolog. This causes a formula to be added to the Alma database. Alma can then request further processing of the incoming message based on user-defined message interpretation code.

Existing Applications

As indicated above, active logic provides a framework for reasoning in presence of uncertainties. Some of the application areas of active logic are discussed below.

[1] This formulation comes close, at least intuitively speaking, to McCarthy's notion of circumscription with an abnormality predicate; see (McCarthy 1986)

[2] Names for formulas play a technical role that we will not further detail here.

Deadline-Coupled Planning

(Nirkhe *et al.* 1997) addresses the problem of deadline coupled planning in active logic. Deadline coupled planning involves taking into account the uncertainties that could crop up in the planning process, while at the same time factoring in the ever decreasing time to deadline.

Consider for example, an agent planning a strategy to FedEx a hard-copy of a paper before the deadline. While the agent is planning the strategy to get to the nearest FedEx location, the clock is ticking. Therefore the time he has available to reach the location before it closes is fast decreasing. While he is trying to reach the nearest FedEx location, time is still passing and hence many uncertain events could happen which could mandate more planning. For instance, a traffic jam could delay the agent and the nearest FedEx location might close, so that he will have to go to another location which is open later. (This reasoning about the choice of locations to try next, based on distance and time-to-closing, is naturally expressible given active logic's time sensitivity and representation; further, since the reasoning itself explicitly takes place in time, this can be used to give preference for an easily-computable and workable plan over a more optimal possibility which might take too much time to compute.)

The time tracking and observation mechanisms of active logic render it useful in such applications that deal with uncertainties while trying to meet a deadline.

Common Sense Reasoning

Active logic finds applications from fully-decidable default reasoning to reasoning in the presence of contradictions. Some examples are listed below (working examples can be found at
http://www.cs.umd.edu/~kpurang/alma/demo/demo.html)

Simple Default Reasoning Given facts *Birds generally fly* and *Tweety is a bird*, active logic can conclude *Tweety flies.*

Default Reasoning with Preferences In active logic, one can specify default preferences like "*Penguins do not fly* is preferred over *Birds generally fly*". Then, if at any instant, *Birds generally fly. Tweety is a bird. Tweety is a penguin.* and *Penguins do not fly.* are in the database, active logic can conclude *Tweety does not fly.*

Maintaining world view An accurate world view cannot be specified without keeping track of current facts, because of the associated uncertainty. Current knowledge can have gaps (e.g., not knowing what constitutes black holes) or it may even be wrong (e.g., earth is flat). As time evolves, facts might change or cease to be true (e.g., the current president, cold war) or even new facts might arise (e.g., existence of the International Space Station). In order to deal with the ever changing plethora of facts, active logic has mechanisms to add, modify or delete facts on the fly.

Reasoning with Contradictions Traditional logics generate all consequences in the presence of contradictions, whereas active logic uses contradictions to help in its reasoning process. An agent can believe that Tweety flies until he gets contrary information through observation or reasoning. In the presence of contradiction, active logic distrusts both the contradictands (Tweety flies and Tweety does not fly) until it has enough facts to trust one over the other.

Dialog

Active logic has been applied to various dialog problems, including presupposition failures (Gurney, Perlis, & Purang 1997), cancellation of implicatures (Perlis, Gurney, & Purang 1996) and dialog management (Perlis *et al.* 1999). The following briefs on these issues.

Uncertainty often arises when Cooperative Principle (Grice 1975) is not observed among the discourse participants. For instance, when the speaker provides insufficient amount of information, or when it is false, irrelevant, ambiguous, vague, or when it lacks adequate evidence, the addressee is uncertain about the speaker's intention. Even when the Cooperative Principle *is* being followed, uncertainty can just as easily arise; e.g. if a speaker uses an unknown word or reference, or when the answer to a question is implicit rather than explicit. In some cases, conversation or communication just stops there, maybe because the speaker is infelicitous and the addressee does not wish to participate in conversation. In most cases, however, the addressee reasons about the speaker's intention and tries to stay in conversation. Despite the fact that there is a potential risk of misunderstanding that could lead to further uncertainty, we take advantage of reasoning when it comes to resolving uncertainty. In the following subsections, we will discuss how intelligent reasoning can be effectively woven into uncertainty resolution in the context of dialog applications.

Assumption In ambiguous cases, people make assumptions based on their previous knowledge. Consider the following example:

"Send the Boston Train to New York." (1)

In this example, the referent of "the Boston Train" may be ambiguous: It may mean the train currently at Boston, or the train going to Boston, or the train which left Boston this morning (and many other things besides). Furthermore, there may be more than one candidate for each case, as for instance, if there is more than one train currently in Boston. Nevertheless, we can deal with this ambiguity by making assumptions based on context.

Relevant context might be the following: The speaker once used the phrase "the Chicago Train", and meant the train currently at Chicago. Hence, we suppose that "the Boston train" means the train at Boston (although we know that there are other possibilities); likewise, given several trains at Boston (e.g. Northstar, Acela, Metroliner) we will choose one candidate, again with an eye to the overall context of the dialog. For instance, Northstar is leaving soon for Cleveland; Acela has mechanical problems. Here we would be led to assume that Metroliner is the train meant.

It is important to note, however, that this reasoning may be mistaken: for it could be that the speaker wanted to send Northstar to New York *instead* of Cleveland. Any reasoning system that employs assumptions should be able to repair false assumptions (see "Repair" below for details).

We have implemented a simplified form of reasoning into the Rochester TRAINS (Ferguson *et al.* 1996) system. In our version of TRAINS, the system makes the assumption that phrases like "the Boston train" mean "the train currently at Boston", and then utilizes context to make a choice among the trains at Boston. It utilizes context in that it chooses the first candidate, the choosing of which will not cause a contradiction in the KB. (For instance, if the sentence "Do not send Northstar to New York" is in the KB, interpreting "Send the Boston train to New York" as "Send the Northstar to New York" will cause a contradiction.)

If the user denies the system's choice of train, that denial becomes part of the relevant context, and will be taken into account when the system considers the alternative candidates in its ongoing efforts to interpret and act on the sentence. Thus our implementation of TRAINS has a rudimentary ability to repair incorrect assumptions. We are currently working on ways to expand this ability, as for instance by adding the capacity to consider other interpretations of phrases like "the Boston train".

Implicature Each expression we utter can mean more than it literally means. Consider the following example:

Q: "Are the roses fresh?"A: "They are in the fridge." (2)

In this example, the answer "They are in the fridge" appears to give information about the location of the roses, rather than their state (which is what was asked about). However, it is not hard to see, that, *given* the location, we are meant to infer that the roses are, indeed, fresh. One way to handle implicature is to assign a default interpretation to the expression. The better way, however, is for the system to be able to reason about the context, and provide an interpretation that is most fitting for the context. (Perlis, Gurney, & Purang 1996) describes an active logic implementation of a reasoner which correctly concludes that the roses are fresh from a dialog like example 2.

Meta-Dialogue When uncertainty arises, one way to avoid further uncertainty and potential discourse failure is to ask for clarification. In natural language discourse, clarification takes place at all times. In the cases discussed, one might confirm the notion that the roses are fresh: "Yes, but are they fresh?". Likewise one might ask: "By Boston Train do you mean Northstar?" Our version of TRAINS is currently equipped with an extremely rudimentary meta-dialog ability, triggered only when its own reasoning reaches an impasse (i.e. when it cannot find a candidate which does not cause a contradiction). In such case it returns a message to the users which says: *Please specify the name of the train.*

We are working on a more robust representation of Question-Answer dialog exchanges that will support a more impressive range of meta-dialog abilities. There are difficult issues to be faced even in the simplest of cases, however. For when the system says: "Please specify the train by name" it looks as though the system should encode for itself an expectation that some future response of the user will be relevant to that request. But in determining whether any given response *is* relevant, all the same issues of uncertainty in dialog interpretation assert themselves. It seems that all of the following user responses should be appropriate: "Metroliner"; "I mean Metroliner"; "Send Metroliner to New York". However, it looks like "Send the Yankee Clipper to Philadelphia" should *not* be understood as relevant (although we could always tell a story in which it *would* be relevant, for it could be an implicit cancellation of the original request, perhaps prompting the question to the user: "Do you still want me to send the Boston train to New York?"). Likewise, were the user to have responded "Send the Yankee Clipper to New York", this could be an indication that the Yankee Clipper (which originated in Boston) was actually the Boston train; and it was the assumption that "Boston train" meant "train at Boston" which was incorrect.

The question of determining relevance is a large, difficult, important open area of research. Our approach, as with other forms of uncertainty, is to model determinations of relevance with explicit reasoning of the type already described (although it should be noted that for applications where the range of choices is circumscribed at the outset, a probabilistic approach looks promising. See, e.g. (Paek & Horvitz 1999))

Repair When false assumptions are made, it is important for an intelligent system to be able to repair the mistake as we do in natural language discourse. In example 2, the answerer might add: "but they are not fresh." In this case, the implication that the roses *are* fresh would need to be retracted. (Perlis, Gurney, & Purang 1996) describes an active logic implementation which can handle such cancellation of implicature.

Likewise, in the TRAINS example, suppose the system were to choose the Metroliner as the referent of "the Boston train" through a course of reasoning. It should be open to the user to say "No", and have the system retract its assumption (and remember the retraction, for it will likely be relevant). As mentioned, our version of TRAINS has this ability to retract the assertion "Send the Metroliner to New York" and remember that retraction to help in future interpretations of user utterances. It is worth pointing out however that the system has made *two* assumptions in order to get to its interpretation: first, that the "Boston train" means the train at Boston, and second, that the Metroliner is the relevant train at Boston. We are working on refining our representation and use of the ongoing dialog context to make it possible not just to reason about other interpretations of phrases like "the Boston Train", but to be able to choose which of these assumptions to negate in light of the user's "No."

Change in meaning

Active logic has been also used to model changes in the meaning associated with terms (Miller 1993). The following is the kind of problem treated in that work.

Imagine an ongoing conversation in which B initially understands A's use of *Bush* to mean George Walker Bush, the 43rd president of the United States. However, A meant his father *George Herbert Walker Bush*, the 41st president. A then tells B that *Bush* is the "read my lips" president. Supposing that B does not know anything about the "read my lips" speech that the 41st president made, B will understand the 43rd president to be the "read my lips" president. Later,

however, when A tells B that *Bush* is married to Barbara, B realizes his misunderstanding. At this point, B has to re-associate previous information with the appropriate referent; B then understands that the "read my lips" president is the 41st president. This is a slightly different sort of repair from those mentioned in the section above, although obviously related.

Potential Applications

Time Sensitive Automated Theorem Prover

Can an automated theorem prover function – to its advantage – more like a (human) mathematician? One way in which this might be possible is via time-sensitivity. A human is highly aware of the passage of time, and in particular to time well-spent as opposed to ill-spent.

Thus a mathematician might, after months (or hours, or even minutes) of pursuing a particular line of investigation, decide that it is not paying off, and instead try a different track. Active logic, with its built-in time-sensitivity, provides a potential mechanism for exploring this possibility. The obvious advantage is that, although a computer may not care that it runs a given program for hundreds of years in a search for a proof, *we* certainly care and we will want it to try a different tack long before many years go by.

There is another likely possible advantage here, something inherent in the active logic framework: it avoids the KR inflexibility of most traditional AI systems. In creative activities such as mathematical reasoning, often the introduction of a key new concept, or even an improved (and possibly inequivalent) reformulation of an old concept, can vastly shorten the argument or even recast it entirely. But AI systems typically are stuck in a fixed means of representing their knowledge and cannot, for instance use Set(X) to refer to finite sets at one time, and later to sets that are finite or infinite. Indeed, such systems cannot easily give up a belief, and when they do ("belief revision"), it is lost rather than available for further consideration. This is not to say that active logic has a built-in general-purpose concept-formation mechanism; but it does have the expressive power to represent and reason with such formations, if they were made available, perhaps along lines of AM (Lenat 1982; Lenat & Brown 1984).

Furthermore, as seen earlier, active logic allows for recognition that a given statement is already known, or that it's negation is known, or that neither is known, thereby avoiding re-derivation of a theorem. Similarly, if such a purported human-style-theorem-prover (HSTP) that is allowed to run in continual mode (rather than started up at a user's request) already working on a proof of X, it can respond, if asked (again) whether X is true, that it is uncertain but that a proof is already underway and that it has established such-and-such lemmas, and so on; or that it has given up since the considerable time spent has not resulted in results that support further effort.

Interactive Mathematical Companion - IMC

We envision an active-logic-based interactive system, which, in conjunction with an appropriate computational mathematical package (MP) as well as a conventional theorem prover (TP), can act as a virtual mathematical assistant and/or companion (Benzmüller *et al.* 1997; Chang & Lee 1973). It may be used in a variety of modes as illustrated by the following scenario outline.

A client may wish to use IMC to ask a (mathematical) question. IMC can try to understand the question on the basis of its existing KB and quite possibly be able to answer the query. Or it may not understand some of the terms used by the client and ask the client to explain them. The desired interactions such as IMC trying to clarify the meaning of terms can be achieved easily by active logic with its ability of detecting and handling ignorance. Thus a dialog may ensue in which IMC may take note of the new definitions or facts relevant to the query that the client furnishes in much the same manner as an interested professional colleague might. After this preliminary dialog, IMC may find that it is unable to answer the question from its (local) KB (even though it '*thinks* it understands the question'). At this point, IMC may consult the TP to see if TP has an answer to the client's question. If TP can respond affirmatively (within the time specified by IMC), IMC can capture the answer, convey the same to the client and also update its KB. In this way IMC is 'learning' from the interaction with the client.

If TP is unable to provide the answer, uncertainty prevails for the client as well as IMC. TP's inability to provide the answer may be because one of the two reasons. Either it is not given sufficient time, or it may just be unable to prove it from the hypotheses contained in the query. In any case IMC can tell the client that it does not know the answer to the question. Uncertainty persists.

The client may then try to pursue some thought of her own. She may come to a point where it is necessary to compute something (possibly as an intermediate step). She may choose to ask IMC to do the computation. IMC interprets client's command and formulates the necessary computation as a request and submits it to the MP. When (and if) it receives the response from MP it passes it back to the client. Such an interaction between the client and IMC can continue indefinitely until the client decides to terminate it. IMC is thus able to mobilize the mathematical resources for the client.

At each step, during the course of an interaction such as outlined above, IMC checks any new facts that are submitted to it by the client, or that are responses it gathers from MP or TP, against its current KB. If the new facts do not directly contradict the KB, they are recorded as assertions in its KB along with the (time) step number when it was entered. If any fact is in conflict with any existing item in the current KB, the contradiction is noted and client is made aware of the contradictands. active logic provides a convenient framework in which IMC can be implemented.

The ability of IMC to keep track of facts as they develop and to detect contradictions can be put to good use by any client who might be trying to check the truth of (or construct a proof of) a proposition.

Continual Computation

A student of AI will soon find out that, almost without exception, any interesting problem is NP-hard. When a computer scientist is confronted with a hard problem, there are several options to deal with it. One is to simplify the problem, or identify a simpler subproblem so that it can be solved algorithmically and automated, and leave the hard part for the human. Another option is for the scientist to study the problem carefully, derive some heuristics, and hope that they will be adequate most of the time. But none of these is quite satisfying: ideally, we would like the computer to do as much work for us as possible, and hopefully, be able to derive the heuristics by itself. A promising approach toward realizing this ideal is the notion of *continual computation* (Horvitz 1997).

The main motivation behind continual computation is to exploit the *idle time* of a computation system. As exemplified by usage pattern of desktop computers, workstations, web-servers, etc of today, most computer systems are under utilized: in typical employments of these systems, relatively long spans of inactivity are interrupted with bursts of computation intensive tasks, where the systems are taxed to their limits. How can we make use of the idle time to help improve performance during critical time?

Continual computation generalizes the definition of a *problem* to encompass the uncertain stream of challenges faced over time. One way to analyze this problem is to put it into the framework of probability and utility, or more generally, rational decision making(Horvitz 2001). However, an implicit assumption of the utility-based work in continual computation is that the future is somehow predictable. But in many cases, this cannot be expected. For example, for long term planning, most statistics will probably lose their significance. Here is a place where logic-based systems with the capability to derive or discover theorems by its own (e.g., Lenat's AM system) can play a complementary role, in a similar way where mathematics plays a complementary role to engineering principles. Just as mathematicians usually do not rely on immediate reward to guide their research (yet discover theorems of utmost utility), AM can function in a way independent of the immediate utility of its work.

More precisely, if we adopt logic as our base for computation and look at problem solving as theorem proving (Bibel 1997), a system capable of discovering new theorems can become a very attractive model of a continual computation system. In such a system, every newly discovered theorem has the potential of simplifying the proof of future theorem; so in essence, theorem becomes our universal format for caching the result of precomputation and partial solutions to problems.

A simplistic embodiment of the model can just be a forward chaining system capable of combining facts in its database to produce new theorems using modus ponens, for instance. Such a system is not likely to be very useful, however, because it will spend most of its time deriving uninteresting theorems. So the success of this model of continual computation will hinge on whether we can find meaningful criteria for the "interestingness" of a theorem. In the classical AM (Lenat 1982; 1983; Lenat & Brown 1984), the system relies largely on human to provide the judgment of interestingness. In a survey of several automated discovery programs, (Colton & Bundy 1999) identify several properties of concepts which seem to be relevant to their interestingness, such as novelty, surprisingness, understandability, existence of models and possibly true conjectures about them. Although these properties seem plausible, it is not obvious they are precise enough to be operational to guide automated discovery programs toward significant results.

Mathematical concepts are characterized by their abstractness. In fact, it is unclear whether the interestingness property of concepts is even meaningful at such abstract level, where the concepts are stripped to their bare essentials, sanitized from any domain specificity: in real life, our interests seem always to be tied to our biological existence in the physical world. When isolated from all real-world interpretations, is there an objective way to tell one theorem is more "interesting" than another? Although we don't have an answer to this question, we have a reason to be optimistic: mathematics has worked so well for us.

Seven Days in the Life of Al To put things into perspective, we will consider Al, a robot powered by active logic. Al is an "office robot", who roams the CS office building delivering documents, coffee, etc. He was endowed at birth with the desire to make people happy. We will see how active logic enabled Al to develop into an excellent office robot through his first week of work (Chong *et al.* 2001).

- **1st day**: The first day Al began his day as the office robot, he was given a tour of the building. Among other things, he was shown the power outlets scattered around the building so that he can recharge himself. Not wanting to overwhelm Al in the new environment, the supervisor let Al off early.
- **2nd day**: The morning went well: Al delivered everything on target, making good use of the map he constructed from the tour given the day before. But a problem occurred during the afternoon: Al found himself unable to move! The problem was soon diagnosed — it was simply low battery. (Since thinking draws less energy than moving, Al could still think.) It turned out that although Al knew he needed power to operate and he could recharge himself to restore his battery, it had never occurred to him that, "he would need to reach an outlet before the power went too low for him to move!" The movement failure triggered Al to derive the above conclusion, but it was too late; Al was stuck, and could not deliver coffee on request. Caffeine deprived computer scientists are not happy human beings; Al had a bad day.
- **3rd day**: Al was bailed out of the predicament by his supervisor at the morning. Having learned his lesson, Al decided to find an outlet a few minutes before the battery got too low. Unfortunate for Al, optimal route planning for robot navigation is an NP-complete problem. When Al finally found an optimal path to the nearest power outlet, his battery level was well below what he needed to move, and Al was stuck again. Since there was nothing else he could do, Al decided to surf the web (through the newly installed wireless network in the building), and came upon an in-

teresting article titled "Deadline-Coupled Real-time Planning" (Kraus, Nirkhe, & Perlis 1990).

- **4th day**: After reading the paper, Al understood that planning takes time, and decided to quickly pick the outlet in sight when his battery was low. Unfortunately, the outlet happened to be too far away, and Al ran out of power again before reaching it. Actually, there was a nearer outlet just around the corner; but since Al used a non-optimal planning algorithm, he was unable to find it. Again, stuck with nothing else to do, Al kicked into the "meditation" mode where he called the Automated Discovery (AD) module to draw new conclusions and theorems based on the facts he accumulated these few days. Al made some interesting discoveries: upon inspecting the history of his observations and reasonings, Al found that there were only a few places he frequented; he could actually precompute the optimal routes from those places to the nearest outlets. Al spent all night computing those routes.
- **5th day**: This morning, Al's AD module derived an interesting theorem: "if the battery power level is above 97% of capacity when Al starts (and nothing bad happened along the way), he can reach an outlet before the power is exhausted." Finally, Al didn't get stuck that day. But people were not quite happy; Al seemed not very responsive. Later, it was found that Al spent most of his time around the outlets recharging himself — since Al's power level dropped 3% for every 10 minutes, the theorem above led him to conclude that he needed to go to the outlet every 10 minutes.
- **6th day**: After Al's routine introspection before work, it was revealed to him that his knowledge base was populated with millions of theorems similar to the one he found the day before, but with the power level at 11%, 12%, ..., and so on. In fact, the theorem is true when the power level is above 10% of capacity. Luckily, there was a meta-rule in Al knowledge base saying that "a theorem subsumed by another was less interesting;" thus all the theorems with parameter above 10% were discarded. Equipped with this newer, more accurate information, Al concluded that he could get away with recharging every 5 hours. Although this might not be an optimal rule, it seemed to work well: Al did his job, and people were happy.
- **7th day**: That happened to be Sunday. Nobody was coming to the office. Al spent his day contemplating the meaning of life.

We would like to bring to attention several features of active logic which helped Al tremendously: time awareness of active logic enabled Al to realized that optimality is not necessarily desirable, especially when there is a deadline approaching; this realization improved his chances of success. More fundamentally, the time awareness allows Al to keep track of changes: the battery level is X now does not imply that is still true 5 hours later. Active logic's Now(i) predicate provides a natural and efficient way to deal with that. The history mechanism in active logic gave Al an opportunity to spot certain pattern in his past behavior, which helped him improve his future behavior. The expressiveness of active logic made it possible to store the meta-rules about interestingness of theorems, which gave Al a good "taste" in evaluating them. Finally, because of the uniformity of the logic-based system, precomputation is seamlessly integrated into goal based problem solving through the forward- and backward-chaining mechanisms of active logic.

The so called *No Free Lunch Theorem* (Wolpert & Macready 1997) states that "all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions." In other words, without domain specific structural assumptions of the problem, no algorithm can be expected to perform better on average than simple blind search. This result appears to be a cause for pessimism for researchers hoping to devise domain-independent methods to improve problem solving performance. But on the other hand, this theorem also provides compelling reason for embracing the notion of continual computation, which can be seen as a way to exploit domain dependent information in a domain independent way. However, to take advantage of continual computation, we cannot avoid the issue of interestingness. Interestingness touches on the ultimate uncertainty: what to do next? Although utility theory has its place, we argued that there are aspects of interestingness not susceptible to utility based analysis. We believe that a forward and backward chaining capable logic system such as active logic, with its expressiveness,time sensitivity, and reflective ability to reason about theorems, proofs and derivations, is well-positioned to take advantage of the opportunity offered by continual computation and explore the possibilities of interestingness.

Importing Intelligence Into Computation

In this paper we hope to have supported the view that (explicit, time-sensitive, and flexible) reasoning can be advantageous wherever there is uncertainty. The underlying architecture can be thought of as (one or more) procedural algorithms with a suitably attached reasoning component so the (system of interacting) algorithms becomes capable of self-monitoring in real time. The reasoning component affords certain protections such as recognizing and repairing errors, informed guiding of strategies, and incorporation of new concepts and terminologies.

In our view, reasoning should apply almost across the board, not only to AI programs, or to automated theorem-provers, but also, for example, to operating systems. Imagine an OS that had an attached active logic. Such an OS-Alma pair not only could gather statistics on its ongoing behavior (this is not new) but could infer and initiate real-time alterations to its behavior as indicated by the circumstances. We envision such an application as largely default-powered, along the line of error-diagnosis tools such as (deKleer 1986). But an active logic version has the additional virtues of accepting dynamically changing observations, and of having a genuine real-time default capability via its introspective lookup interpretation of ignorance.

Acknowledgments

We would like to acknowledge support for this research from AFOSR and ONR.

References

Benzmüller, C.; Cheikhrouhou, L.; Fehrer, D.; Fiedler, A.; Huang, X.; Kerber, M.; Kohlhase, M.; Konrad, K.; Meier, A.; Melis, F.; Schaarschmidt, W.; Siekmann, J.; and Sorge, V. 1997. ΩMEGA: Towards a mathematical assistant. In McCune, W., ed., *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, 252–255. Berlin: Springer.

Bibel, W. 1997. Let's plan it deductively! In *IJCAI*, 1549–1562.

Bratman, M. 1999. *Faces of Intention*. Cambridge, UK: Cambridge University Press.

Chang, C. L., and Lee, R. C. T. 1973. *Symbolic Logic and Mechanical Theorem Proving*. New York, NY: Academic Press.

Chong, W.; O'Donovan-Anderson, M.; Okamoto, Y.; and Perlis, D. 2001. Seven days in the life of a robotic agent. In *To be presented at the GSFC/JPL Workshop on Radical Agent Concepts*.

Colton, S., and Bundy, A. 1999. On the notion of interestingness in automated mathematical discovery. In *AISB Symposium on AI and Scientific Discovery*.

deKleer, J. 1986. Problem solving with the ATMS. *Artificial Intelligence* 28:197–224.

Ferguson, G. M.; Allen, J. F.; Miller, B. W.; and Ringger, E. K. 1996. The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant. TRAINS Technical Note 96-5, University of Rochester.

Grice, H. P. 1975. Logic and conversation. In Cole, P., and Morgan, J. L., eds., *Syntax and semantics 3: Speech Acts*. Academic Press. 41–58.

Gurney, J.; Perlis, D.; and Purang, K. 1997. Interpreting presuppositions using active logic: From contexts to utterances. *Computational Intelligence* 13(3):391–413.

Horvitz, E. 1997. Models of continual computation. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, 286–293. Menlo Park: AAAI Press.

Horvitz, E. 2001. Principles and applications of continual computation. *Artificial Intelligence* 126(1-2):159–196.

Kraus, S.; Nirkhe, M.; and Perlis, D. 1990. Deadline-coupled real-time planning. In *Proceedings of 1990 DARPA workshop on Innovative Approaches to Planning, Scheduling and Control*, 100–108.

Lenat, D. B., and Brown, J. S. 1984. Why AM and EURISKO appear to work. *Artificial Intelligence* 23(3):269–294.

Lenat, D. B. 1982. *AM: Discovery in Mathematics as Heuristic Search*. New York, NY: McGraw-Hill. 1–225.

Lenat, D. B. 1983. Theory Formation by Heuristic Search. *Artificial Intelligence* 21:31–59.

McCarthy, J. 1986. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence* 28(1):89–116.

Miller, M. 1993. *A View of One's Past and Other Aspects of Reasoned Change in Belief*. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, Maryland.

Nirkhe, M.; Kraus, S.; Miller, M.; and Perlis, D. 1997. How to (plan to) meet a deadline between *now* and *then*. *Journal of logic computation* 7(1):109–156.

Paek, T., and Horvitz, E. 1999. Uncertainty, utility and misunderstanding: A decision-theoretic perspective on grounding in conversational systems. In *Proceedings, AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems*.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann.

Perlis, D.; Purang, K.; Purushothaman, D.; Andersen, C.; and Traum, D. 1999. Modeling time and meta-reasoning in dialogue via active logic. In *Working notes of AAAI Fall Symposium on Psychological Models o f Communication*.

Perlis, D.; Gurney, J.; and Purang, K. 1996. Active logic applied to cancellation of Gricean implicature. In *Working notes, AAAI 96 Spring Symposium on Computational Implicature*. AAAI.

Perlis, D. 1997. Sources of, and exploiting, inconsistency: Preliminary report. *Journal of APPLIED NON-CLASSICAL LOGICS* 7.

Ramoni, M., and Riva, A. 1994. Belief maintenance in bayesian networks. In *Proceedings of UAI 1994*.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82.