Active Logic for More Effective Human-Computer Interaction and Other Commonsense Applications*

Michael L. Anderson[1], Darsana P. Josyula[2], Don Perlis[1,2], Khemdut Purang[3]
(1)Institute for Advanced Computer Studies,
(2)Department of Computer Science,
University of Maryland, College Park, MD 20742
(3)Santa Clara, CA USA,
{anderson,darsana,perlis}@cs.umd.edu,kpurang@yahoo.com

June 1, 2004

Abstract

The demands of real-time commonsense reasoning—as evidenced for example in the pragmatics of human-computer dialog—put stringent requirements on the underlying logic, including those of (i) perturbation tolerance, (ii) contradiction tolerance and (iii) time situatedness. Active logic is an attempt to meet all of these needs. In this paper we present this work and its application to natural language dialog via time-sensitive meta-reasoning. We illustrate this with a description of ALFRED, a cooperative natural language interface to multiple task-oriented domains.

Keywords: first order logic, reasoning, meta-reasoning, human-computer dialog

1 Introduction

The demands of real-time commonsense reasoning—as evidenced for example in the pragmatics of human-computer dialog—put stringent requirements on the underlying logic, including those of (i) perturbation tolerance, (ii) contradiction tolerance and (iii) time situatedness. Perturbation tolerance is the ability to maintain effective reasoning in the face of unanticipated changes.[1] Because in a logic-based system change (especially when it is unexpected) can result in contradictions between what is believed or expected and what is actually being experienced, perturbation tolerance implies a need for contradiction tolerance. Time situatedness is the capacity to track (and reason about) the passage of time as reasoning proceeds; not only is this ability useful when reasoning in light of approaching deadlines, but the simple awareness, and explicit representation, of the fact that beliefs change over time can be an important part of effectively managing

---

[*] This research is supported in part by the AFOSR and ONR

[1] Perturbation tolerance is meant as an extension and generalization of John McCarthy's notion of "elaboration tolerance": the ability of a reasoning agent to appropriately adjust its beliefs and actions to respond to new information [19].

that change. Active logic (and its implementation in ALMA—Active Logic MAchine) is an attempt to meet all of these needs. We have, in recent years, gathered together some key features of Active Logic into an algorithm we call MCL, that seems to provide a powerful and general tool for commonsense reasoning, as well as a unifying view of our various applications.

Our current view of commonsense reasoning is that it depends in large measure on a special skill in meta-reasoning, in the service of error-handling. This view is motivated by the thought that the ability to "get along" in a wide range of changing environments is a key feature of human cognition. Our suspicion is that this ability, which largely amounts to perturbation tolerance, is the result of robust self-monitoring, and the ability to notice and respond to novelty, error, unexpected outcomes, and the like, which are the inevitable result of interaction with a complex and dynamic environment. More specifically, we hypothesize a three-part process that (i) notes anomalies in overall system performance, (ii) assesses options for dealing with it, and (iii) guides a response into place. We call this note-assess-guide, the metacognitive loop (MCL). That humans are skilled at such a process is clear; our efforts are aimed at seeing to what extent the same process can be effectively incorporated into automated systems as well.

Note that suitably simple versions of MCL need not employ logic at all. For instance, we have performed a number of experiments in reinforcement learning, where an "agent" (CHIPPY) meanders in a two-dimensional grid with various levels of reward (positive or negative) at different locations. Standard learning algorithms (such as Q-learning) work very well; however, when the reward locations *change* after the learning has gotten far along, then the re-learning takes significantly *longer* than the initial learning. This is because the agent must un-learn what it has learned, and (in non-MCL versions) the algorithm is unable to detect that what it has learned is now doing poorly; it simply keeps plugging along, adjusting its action policy little by little. But even the seemingly simple stratagem of throwing out all that it has learned will speed up the re-learning a great deal; and this is easily done by a trivial version of MCL, which, when it notices that (say) three mismatches have occurred between the rewards it actually received, and the rewards it expected, decides to forget its current action policy and start over. Of course, throwing out one's entire policy is drastic, and for many purposes unwise; we are currently working on more sophisticated versions of MCL that will reason about matters such as context and time, to widen the range of options that can be guided into place. We expect that as the demands on the meta-reasoning increase, logic-based implementations will become more attractive.

This is certainly the case when implementing MCL for use in natural language interfaces. As we will discuss in more detail below, natural language understanding is a difficult process full of the opportunity for error; thus the ability to notice and recover from errors is extremely important. But in the case of language understanding there is no single solution to apply in every case; to assess and decide how to fix an error requires careful considerations of its origin, and this in turn requires the ability to represent and reason about one's own past reasoning. To implement this ability in first-order logic, and in particular a contradiction-tolerant, and time-sensitive variation of FOL, is the natural choice.

In what follows, we will first discuss active logic in general, and its specific implementation in ALMA. Next we will discuss human-computer interfaces and their requirements, especially in conversation. This leads finally to our implementation of a natural-language based interface, ALFRED, which is presented in some detail.

2 Active Logic: An Introduction

Active logics are a family of formalisms that combine inference rules with a constantly evolving measure of time (a 'now') that itself can be referenced in those rules. An account of the basic concepts can be found in [11].

In active logic, aspects of the environment are represented as first order formulas in the knowledge base. Such formulas might represent perceptions of a user's utterance, observations about the state of the domain, or rules added by a system administrator. Inference rules provide the mechanism for "using" the knowledge for reasoning. One aspect of active logic especially important in the current context is its robust ability to continue to reason normally as formulas are added, changed or deleted from its knowledge base. In other words, the evolving knowledge base is naturally integrated into the ongoing reasoning processes. This makes active logic a good candidate for a reasoning agent which is expected to observe and interact in real time with a continually changing world or domain.

One of the original motivations for active logics was that of designing formalisms for reasoning about an approaching deadline; for this use it is crucial that the reasoning takes into account the ongoing passage of time as that reasoning proceeds. Such a formalism has the ability to explicitly track the individual steps of a deduction, making it a natural mechanism for reasoning about contradictions and their causes.

Thus, in active logic, the formulas at step i+1 include the formulas obtained by applying the rules of inference *once* to the formulas in step i as illustrated below:

$$
\begin{array}{cll}
 & i \quad\;\; : & A, A \rightarrow B, B \rightarrow C \\ \cline{3-3}
(1) & i+1 : & A, B, A \rightarrow B, B \rightarrow C \\ \cline{3-3}
 & i+2 : & A, B, C, A \rightarrow B, B \rightarrow C
\end{array}
$$

Here, at step $i+1$ B is derived using the axioms A and $A \rightarrow B$ and at step $i+2$ C is obtained from the axioms, B and $B \rightarrow C$.

Each "step" in an active logic proof itself takes one active logic time-step; thus inference always moves into the future at least one step and this fact can be recorded in the logic. The KB will at all times be finite since the finitely-many inference rules can produce only finitely-many conclusions in one time-step.[2]

By default, all beliefs from one step that are not directly contradicting are inherited to the next step. However some beliefs like the ones related to the current time are not inherited to the next step. (See below). The inheritance of formulas from one step to the next is controlled by inheritance rules. One simple version of such an "inheritance rule"[3], which also illustrates the use of firing conditions, is shown in 2:

---

[2] In ongoing work begun in [23] we have been exploring ways to keep the KB size not merely finite but bounded, analogous to human short-term memory.

[3] Inheritance and disinheritance are directly related to belief revision [12] and to the frame problem [18, 6]; see [23] for further discussion.

$$(2)\quad \begin{array}{l} \text{i} \quad : \underline{\text{A}\;} \\ \text{i+1} : \text{A} \end{array}$$

[condition: $\neg$A $\notin$ KB at step i and A $\neq$ Now(i)]

In addition to the formulas obtained from applying rules of inference to formulas at the previous step, new formulas can be added at each step. Step-wise reasoning coupled with this ability to add new formulas, ensure that the logic would not get stuck in a lengthy proof, oblivious of the other events that occur *during* the reasoning.

To achieve much of their reasoning, active logics employ a notion of "now" that is constantly updated by the "clock rule" shown in 3:

$$(3)\quad \begin{array}{lcl} \texttt{i} & : & \underline{\;\;\texttt{Now(i)}\;\;} \\ \texttt{i+1} & : & \texttt{Now(i+1)} \end{array}$$

The clock rule states that from the fact that it is step i at the current step, the step number of the next step is $i + 1$. This step-wise tracking model of time is very different from the "time-frozen" characterization of time that temporal logic [1, 29] has. The notion of past, present and future, that temporal logics have do not change while theorems are being derived. This sharply contrasts with the special evolving-during-inference model of time that active logics have. When an agent is reasoning about its own ongoing activity, or about another agent whose activity is highly interdependent, traditional "time-frozen" reasoning is at a disadvantage, and "time-tracking" active logics can bring new power and flexibility to bear. For instance, theorems can be marked with their time (step-number) of being proven, i.e., the current value of "now". This step-number is itself something that further inferences can depend on, such as inferring that a given deadline is now too close to meet by means of a particular plan under refinement if its enactment is estimated to take longer than the (ever shrinking) time remaining before the deadline.

It is the time-sensitivity of active logic inference rules that provides the chief advantage over more traditional logics. Thus, an inference rule can refer the results of all inferences *up until now*—i.e. thru step i—as it computes the subsequent results (for step $i + 1$). This allows an active logic to reason, for example, about its own (past) reasoning; and in particular about what it has *not* yet concluded. Moreover, this can be performed quickly, since it involves little more than a lookup of the current knowledge base.

In the sub-sections that follow, we will discuss some of the useful general features of active logics.

2.1 Contradiction Tolerance

As is well known, traditional formalisms, including most modal, temporal and nonmonotonic logics, suffer from the "swamping problem" (this is related to the "omniscience" problem of traditional logics of belief: all (infinitely-many) consequences of the axioms are theorems and hence are believed). As a result, in those logics, any possible clues as to how to proceed with reasoning when a contradiction is encountered are rendered ineffective by their own negations which are also derived from the contradiction.

There have been some attempts to overcome the swamping problem, but so far only in the propositional case, and even so the essential time-dependency for real-time capabilities is still missing there.

Even though the problem of inconsistency is treated by some logics like paraconsistent logics, in reality most of the traditional logics do not note or repair inconsistencies, they just carry on with them. Nor in general do they provide for any special real-time status as needed by a real-world agent. On the other hand, active logics are intended for on-board use by an agent, not as an external specification of an agent.

In active logics, since the notion of inference is time-dependent, it follows that at any given time only those inferences that have actually been carried out so far can affect the present state of the agent's knowledge. As a result, even if directly contradictory wffs, P and ¬P, are in the agent's KB at time `t`, it need not be the case that those wffs have been used by time `t` to derive any other wff, Q. Indeed, it may be that `t` is the first moment at which both P and ¬P have simultaneously been in KB.

By endowing an active logic with a "conflict-recognition" inference rule such as that in 4, *direct* contradictions can be recognized as soon as they occur, and further reasoning can be initiated to repair the contradiction, or at least to adopt a strategy with respect to it, such as simply avoiding the use of either of the contradictands for the time being. Unlike in truth maintenance systems [9, 10] where a separate process resolves contradictions using justification information, in an active logic the contradiction detection and handling [21] occur in the same reasoning process. The `Contra` predicate is a meta-predicate: it is about the course of reasoning itself (and yet is also part of that same evolving history).

(4)
$$\begin{array}{lcc} \mathtt{i} & : & \mathtt{P,}\ \neg\mathtt{P} \\ \hline \mathtt{i+1} & : & \mathtt{Contra(i,\ P,\ \neg P)} \end{array}$$

The idea then is that, although an indirect contradiction may lurk undetected in the knowledge base, it may be sufficient for many purposes to deal only with direct contradictions. Sooner or later, if an indirect contradiction causes trouble, it may reveal itself in the form of a direct contradiction. After all, a real agent has no choice but to reason only with whatever wffs it has been able to come up with *so far*, rather than with implicit but not yet performed inferences. Moreover, since consistency (i.e., the lack of direct or indirect contradictions) is, in general, undecidable, all agents with sufficiently expressive languages will be forced to make do with a hit-or-miss approach to contradiction detection. The best that can be hoped for, then, seems to be an ability to reason effectively in the presence of contradictions, taking action with respect to them only when they become revealed in the course of inference (which itself might be directed toward finding contradictions, to be sure).

But while disinheriting contradictands is a reasonable first step, it is often not enough even to "defuse" the contradiction for long. P and ¬P may have come into KB for reasons that are still in force and the system may re-derive P and ¬P, or other similar conflicts, later on. Thus, in [22, 13, 25] we have investigated ways to allow an active logic-based reasoner to retrace its history of inferences, examine what led to the contradiction, and perform meta-reasoning concerning which of these warrants continued belief.

2.2 Non-monotonicity

Unlike most NMR formalisms, we do not attempt to capture the (usually undecidable) absolute truth about what is consistent with what is known; this is in general impossible for real agents. If nothing is *already* known that would prevent a default conclusion, then the agent has little choice except to draw that conclusion, and this is what an active logic does. If later (with more time) the agent discovers a consequence of its beliefs that in fact should have prevented that conclusion, it is only at that later time that it can be withdrawn, and this is what active logic makes possible. In principle, in the limit, active logic should, in special cases at least, provide the same default conclusions as standard NMR formalisms; this is a topic of current investigation.

2.3 Introspection

In Active logic, negative introspection – the ability to determine that one does *not* know something – is often encoded as the following inference rule (where the notation [B] means that B is not present):

$$(5)\quad \begin{array}{ll} \texttt{i} & : \\ \texttt{i+1} & : \end{array} \; \frac{\texttt{...[B]}}{\texttt{-Know(i,B)}}$$

This mandates the conclusion at time i+1 that statement B was not known to the logic at time i (that is, B does not appear among those beliefs).

2.4 Defaults

Defaults are given a straightforward representation in active logics:

$$(6)\quad \begin{array}{ll} \texttt{i} & : \\ \texttt{i+1} & : \end{array} \; \frac{\texttt{Q, -Know(-P,i), Now(i)}}{\texttt{P}}$$

Thus if $\neg$P is not known at the current time, and if Q is known, then P is inferred by default at the next time step. This avoids the decidability issues of traditional default mechanisms, since only a linear lookup in the belief set for time i is needed to tell that $\neg P$ is not there (and that Q is there). This does not in itself deal with problems arising from interacting defaults. However, since such cases tend to involve contradictory conclusions (as when, evidence for $\neg$P becomes known), they can be treated as any other contradictands. In our past work, we have applied a very simple expedient in such cases: disinherit the default conclusion and accept the non-default evidence.

2.5 History Tracking

Active logic maintains a temporal history of its reasoning process that can be used by the logic for further reasoning. The history enables the logic to determine when each formula was added or deleted in its past and thus provides a mechanism to reason about the past reasoning.

2.6 Quotation

Quotation mechanism names the different formulas in Active logic. This allows an individual formula to be referenced using its name. The quotation and the history mechanism together provides a mechanism for meta-reasoning within the reasoning process itself.

2.7 Integration with non-logical processing

Finally, we extend the logic with one special proposition, *call*, which, if it is ever proved, will initiate external action (that can be reasoned about and tracked through observation). Our current implementation of active logic, as represented in ALMA [26], already has this ability to initiate, observe and respond to external events and non-logical processes.

Active logics can be seen either as formalisms per se, or as inference engines that implement formalisms. Unlike traditional logics, the notion of theoremhood in Active logic depends directly on (i) what is in the current evolving belief set, and (ii) what the current evolving time is.

The traditional markers of a logic are its syntax and its semantics. Active logics have both of these: the syntax is (usually) that of FOL; and the semantics can also be that of FOL with a few addenda such as that Now(x) has the meaning that the current evolving time is x. (There are also alternative semantics available.) What is missing is a soundness and completeness theorem, and for good reason: active logics are not intended to be sound or complete but rather to reflect the step-by-step process of reasoning of a real agent. Thus many true assertions will not be proven, and many things proven are not true. In fact, active logics are designed with inconsistent belief sets in mind; and these of course can never be true.

It is best to avoid a mere terminological squabble over the word "logic". However, in many important senses, active logics are formal specifications of notions of theoremhood appropriate to the study of real agents. If we are concerned about agents and their reasoning, rather than about an agent-independent notion of truth, then we should not expect or want a tight coupling between what is proven (or provable) and what is true. Agents can only do what they have the resources to do, and whatever logic an agent uses must therefore also have that property. Thus to the extent that logic is the study of reasoning, active logics are the study of reasoning as an active process.

3 Alma/Carne

Alma/Carne was designed to be the core reasoning engine for active logic based applications. Alma generates active logic inferences and Carne executes arbitrary processes and interfaces with the world on behalf of Alma. These processes can include non-logical reasoning procedures like probabilistic reasoners, parsers and enables close interaction between different kinds of reasoning. Alma differs from the description of active logic given above in two main ways. Firstly, additional features are required for interaction with external systems. Secondly, a practical system requires compromises to the ideal system so it can run effectively. We describe these differences below.

An application is specified by a set of logical sentences that are loaded into Alma and a set of procedures loaded into Carne. As the system runs, inferences are made in Alma, some of which trigger procedures in Carne. These procedures can perform some computation or cause effects in the world. Alma's state is updated with the status of the procedures which enables reasoning about the processes Alma triggered. Failure of a procedure, for instance, can lead to reasoning that causes retraction of earlier assumptions. Carne can also monitor the world and assert formulas about the state of the world into Alma, implementing the observation functionality of active logic. This enables Alma to react to changes in the world.

3.1 Alma implementation highlights

Following are some of the more interesting aspects of Alma's implementation of active logic.

3.1.1 Representation

Formulas input to Alma are converted to clausal form and asserted in the database. They are indexed by the predicate and the first two arguments of their atoms. All formulas are named, by the user or otherwise, by Alma. The names are used to refer to the formulas.

The system maintains information about the formulas, for example, how they were derived, at what step, and which derivations they were involved in. An important such piece of information is the status of the formula: trusted or distrusted. Distrusted formulas are not used for futher inference but remain present in the database. Formulas typically become distrusted when they are involved in a direct contradiction. All of that meta-information is available through reserved predicates and form the basis of the meta-reasoning capabilities of Alma. Other reserved predicates cause changes in the database (including meta-information) when asserted. For example, user-controlled change of the status of a formula can be used to remove from further inference potentially false formulas, or, on the other hand, to reinstate formulas that were previously distrusted.

3.1.2 Computation

Inferences, including those for meta-reasoning are done by applying *the resolution and factoring rules.* Alma is therefore not complete in the forward direction. This requires more care on the part of the user in choosing formulas to assert in Alma. The procedural as well as logical aspects of Alma have to be considered when specifying domains.

Alma can perform arbitrary Prolog computation with the *eval_bound* form. The computations performed should be limited to those that will complete fast. Longer running computations are to be done using Carne.

The *contradiction detection rule* verifies whether the negation of any newly added atom is present in the database. Inconsistencies are therefore only detected when complementary atoms are derived. The presence of an atom and its complement causes the status of these formulas to change to being 'distrusted'. The fact that there is a contradiction and the identity of the contradictands are asserted in the database. Strategies for dealing with the contradiction can be expressed as user-specified formulas which depend on the contradiction assertions being present in the database. These can use the properties of the formulas to decide on a resolution to the contradiction. Alma stops the immediate spread of potentially bad facts by distrusting them and provides the means for the user to handle the contradiction but has no further built-in strategy.

The *time rule* deletes the previous $now(i)$ and asserts $now(i+1)$ and thereby defines each step.

As alluded to above, there is no explicit *inheritance rule*. All formulas are inherited by default, except for those that are distrusted or explicitly removed.

Introspection is achieved by determining the truth value of $Know(P)$ by merely inspecting the database once P is ground. This limits the expressivity somewhat but provides substantial gains in efficiency.

3.1.3 Control

The rules above (resolution, contradiction detection and time) are applied at each step. While contradiction detection is applied to all formulas, the application of resolution can be restricted by the user. Prolog filters can be specified that determine which formulas resolution is to be applied to. The user can also specify an ordering on pairs of formulas that are candidates for resolution application. This, together with a limit on the time to be spent on each step can lead to a bounded step duration which only does the most important resolutions. Here as in contradiction handling, Alma provides the means for the user to specify strategies. It is possible for user-specified strategies to indefinitely delay inferences.

A further device that Alma provides for controlling the inference is the availability of three kinds of conditional according to whether they are to be used for forward chaining, or backward chaining or both. This can be a crucial distinction practically. There may be implications that are sometimes required but will typically generate "useless" formulas in forward chaining which can then swamp the database. Using the backward chaining implication eliminates the generation of these useless inferences while allowing the intended inferences to be made.

Backward chaining is done step by step in a breadth first fashion. Breadth first search has the advantage of finding the shortest solution but the disadvantage of requiring more resources. The search tree is maintained as a set of formulas in the Alma database. This means that new information can inform proofs started earlier and sprout new branches in the interior of the tree. Alma provides several ways to control the proof, including a depth limit.

3.1.4 Implementation

The system is implemented in Prolog with a GUI written in Java which displays ongoing inferences in Alma. Alma can output a history of its inference. This history can then be replayed in the GUI. This feature has proved useful for debugging domain descriptions.

3.2 Carne

Carne is a process separate from Alma that communicates with Alma to run procedures that would take too long to run in the Alma process. Carne also serves as a link from Alma to other external processes. This allows Alma to be embedded in a larger system with Alma providing reasoning services, as in [30] where Alma/Carne implements the dialog manager in a larger planning system.

The following table presents the simple interface between Alma and Carne.

Formula	Asserted by	Effect
$do(P, ID)$	Alma reasoning	Carne invokes program P
$doing(P, ID)$	Carne	Informs Alma that P has been invoked.
$done(P, ID)$	Carne	Informs Alma that P has completed.
$error(P, ID)$	Carne	Informs Alma that P has failed.
ϕ	Carne	Informs Alma of ϕ

P is a program that is defined in Carne, and ID is a unique identifier for that invocation instance. This simple interface together with Alma's inference allows several types of behavior. For example, the user can specify that a program should be run and aborted if it does not complete within a time limit. This can be done by comparing the time at which *doing* is asserted with the current time and invoking another program to abort the first.

The last row in the table represents Carne asserting formulas in the Alma database. Notice that the do/doing/done terms do not allow complex results of computation to be asserted in the database. Programs run by Carne can use a reserved term to assert formulas in the Alma database.

4 Some Aspects of Interfaces

Human-computer interfaces are Janus-faced, with one side facing outward and interacting with the user, and another side facing inward, interacting with and controlling some task-oriented system. This system can be limited and specific, as with a digital video recorder or camera, or more general, as with a desktop computer. A typical interface design strategy is to let the shape of the interface be primarily determined by the function(s) of the device itself, relying on familiar controls for each function. Thus some simple microwaves have just a timer dial and a power-level knob. The advantage of this approach is two-fold. First, the capabilities of the device are displayed on its face, and thus the way to access these functions is obvious to the user. And second, because this is so, very little thought needs to be given to the inward-facing side of the interface; for the user is essentially controlling the device directly. However, the approach also has some important drawbacks. First, as devices get more complex, it is not generally possible to "show" all its capabilities without making them resemble airplane cockpits. Second, as users acquire more electronic devices (computers, cameras, dvd players, mp3 players, cell phones, PDAs, etc.) they may find learning each new interface increasingly burdensome. Thus, as consumer electronics become more complex and various, designers will need to consider the possibility of employing a single, shared, general and flexible interface which is easy for users to learn, and can control many different devices. For this purpose, it is hard to beat natural language.

Implementing a general purpose, natural-language based interface, however, offers some difficult challenges. First, and most obvious, is the difficulty and complexity of language itself. Second is the fact that, insofar as the interface is not specifically designed to control a particular device, the interface must itself be intelligent enough to translate what a user says into something a given device can do. Deeply related to this latter fact is that the user is no longer directly manipulating and controlling the target device; thus much of the agency and proactive responsiveness which would normally be the provenance of the user must be taken over by the interface.

In the subsections that follow, we will discuss each of these challenges in turn. Then in section 5 we outline our general approach to these challenges, and highlight our use of active logic. The remainder of the paper will discuss the various aspects of ALFRED, our flexible, natural-language based interface to several task-oriented domains.

4.1 The Nature of Conversation

The development of a truly viable natural language computer interface will require advances in computer reasoning, in particular in the ability to monitor for, reason about and manage the particular uncertainties inherent in interactive dialog. This, in fact, is what human dialog partners do. They manage the uncertainty inherent in dialog by continually monitoring their conversations, their own comprehension, and the apparent comprehension of their interlocutor. This ability is apparently very basic, and fundamental to language use. Clark [8] presents empirical evidence for the use of meta-linguistic skills by young children. Among many other things, these skills include monitoring one's ongoing speech, checking the results of that speech, predicting the effects of specific utterances, and deliberately trying to learn.

These behaviors are not just necessary to children learning a language, but in fact pervade conversation between fully competent language users [3, 27, 28]. The ability to ask questions about what's been said and understood—that is, the ability to engage in meta-dialog (dialog about the dialog or its elements)—and to use the results of these meta-dialogic interchanges to help understand otherwise problematic utterances, is essential to conversation.[4] We believe that one of the problems facing natural language HCI at the present time is that the computer partners in dialog are not generally equipped with this ability to elicit and utilize appropriate feedback; this can make these systems difficult and frustrating to use [5].

Thus, we have been inching closer (we hope) to a viable natural language computer interface by focusing our efforts on understanding the importance, role, and extent of human dialog monitoring and repair—as well as the meta-reasoning and meta-linguistic skills this monitoring and repair involves—and implementing these behaviors in a task-oriented natural language computer interface [4, 31, 16, 2]. The specific goal we have set for ourselves is the design of a system modeled not on conversation with a fluent colleague, but rather, for example, on a task-oriented interaction with a stranger who doesn't speak much of a common language. Despite the difficulties of understanding the language, interactive dialog behaviors and ongoing repairs allow humans to overcome problems and communicate.

---

[4] Indeed, as any teacher—and student—knows, it is important to natural language comprehension more generally.

4.2 The Requirements for a Flexible Domain Controller

As mentioned earlier, as electronic devices become more complex and various, users find it increasingly frustrating to learn how to operate each one separately. Thus, it would be a great advantage for a single agent to be able to interface with the user, and with all these different task-oriented systems (TOSs). If a single agent can interface users with different TOSs, then the users would need not learn how to interact with each TOS separately; instead, they would just need to learn how to interact with one agent. In addition, if the agent is rational, users can more flexibly adapt the operation of the TOSs by interacting with the agent.

However, for such an interfacing agent to effectively control different TOSs it should have the capability not only to translate a user request into a TOS instruction, and to issue that instruction to the TOS at the appropriate time(s), but, because the user is insulated from direct control of (and may be unable to monitor) the TOS, the interfacing agent should also be able to track the effect of those commands and to detect any perturbations, such as contradictory information, or a difference between expected and actual outcomes. In addition, because the user may be unaware of the actual capabilities of any given device, the interfacing agent will itself have to be able to do such things as distinguish between achievable and unachievable goals, informing the user of the latter. Finally, to manage its own reasoning it will have to distinguish between these and achieved goals (desires/intentions/expectations) so it can attempt achievable goals, resist attempting unachievable goals (until they become achievable) and ignore achieved goals.

In what follows, we will detail our approach to these different challenges, and describe some of the behaviors we have been able to generate in our natural language HCI systems.

5 Active Logic and the Interfacing Agent

Insofar as the above analysis of the nature of conversation and the requirements for an effective, flexible, error-tolerant domain controller is correct, it suggests that any agent hoping to be a good human-device interface should be:

1. Self-monitoring: An interfacing agent should track its own language comprehension, maintain a history of its translations of user utterances into domain-specific instructions, and track its own beliefs, desires, intentions, expectations and achievements, so as to be able not just to notice errors (such as mistakes of translation, mismatches between expectations and observations, or between implications of past utterances and current ones) but also to be able to trace their origins and fix them.

2. Contradiction-tolerant: Having encountered problems, which are inevitable not just in conversation, but whenever one is dealing with a complex, dynamic environment, an interfacing agent should be able to gracefully handle these situations.

3. Time-sensitive: Dialog is governed by a number of time-based expectations. If someone makes a conversational overture, or asks a question, a response is expected

in a certain amount of time.[5] An interfacing agent should be sensitive to these demands. Likewise, it should have expectations about the time it will take its human user to respond, as deviations could indicate a problem. For instance, a long pause could signal a turn change, or that the user is confused, or even that the user is no longer engaged in the conversation. Further, when the interfacing agent gives a command to a domain, it should have specific expectations for what should result, when. These time-specific expectations can be used to monitor progress, and notice and address problems as they arise. Alternately, a command (or series of commands) may have to be given at certain times and/or in a specific order; time-sensitivity can help make this possible. Finally, time sensitivity can help with deadline-sensitive planning, allowing the reasoner to properly allocate resources or choose strategies for acting and reasoning in light of the time remaining before a looming deadline.

4. Multi-modal: A dialog agent should ideally be able to monitor all the various aspects of speech, including affect, tone and gesture, as well as to match its own utterances with appropriate accompaniment. Unfortunately, monitoring, interpreting and producing these effectively and freely (without prior scripting) is currently beyond the state of the art.[6] However, the ability to attend to multiple contexts is not, and is something which ought to be included in dialog systems whenever possible. For, to understand an utterance it may not be enough to attend to the dialog context (e.g. the subject under consideration, open questions under discussion, turn) but also to the larger environment. If one says "I guess he's had enough," the "he" might equally well refer to the fellow under discussion who quit his job, or the one who just fell from his barstool.

Our approach to implementing these abilities involves three conceptual planks. First, there is the issue of representation of aspects of the dialog processing as it moves from user utterance to domain instructions to the setting of expectations. For many object-level behaviors, it is not necessary to have an explicit representation of the processes that the system performs, other than the programming or mechanism that produces the behavior at appropriate times. However, to engage in meta-behavior, such as dialog about dialog, or reasoning about the reasoning process, rich representations are needed. In the case of meta-dialog we are influenced by work such as [14, 15, 32, 24, 20], that proposes detailed logical representations of a range of dialog phenomena. Likewise for the interfacing agent's beliefs, expectations, and the like. To effectively reason about these, especially when faced with an anomaly such as contradictory information, it is important for these to be richly represented. Second, the system must be able to effectively use such representations in inference to be able to notice interesting phenomena, such as implications of what has been done, recognizing resulting anomalies, and deciding what can be done about it. For use in a real-time system, this reasoning cannot be off-line, but must be integrated within the normal behavior of such a system. This leads to the third plank, integration of reasoning with acting and non-logical processing. Reasoning

---

[5] Exactly how much time is context dependent. For a conversational overture or a very easy question, a near immediate response is required. If a question is perceived to be difficult, or require complex thinking, a longer delay may well be tolerated.

[6] Although some impressive work is being done, see, e.g. [7]

about anomalies and meta-dialog is not enough. To be effective such reasoning should be integrated with normal functioning, being able to affect object level processes.

As detailed in section 2, active logic is a formalism which appears to have all the requirements for implementing the abilities listed above. Thus, we have been using active logic to implement the real-time meta-reasoning needed for natural-language based interfaces. Because active logic uses an inference mechanism that takes account of the passage of time as it performs inferences, it has both the expressive and the inferential power to monitor its own reasoning in a real-time fashion, as that very reasoning is going on, thus watching for errors (such as mismatches between conveyed and intended meanings); noting temporal conversational cues such as pauses that may signal a turn change; and re-examining its beliefs and altering them appropriately. In what follows we will describe in detail our progress so far.

6 ALFRED

The logical engine ALMA/CARNE has been used to create an interfacing agent ALFRED (Active Logic For Reason Enhanced Dialog) that can exhibit some of the metalinguistic skills that are necessary for a truly viable natural language human-computer interface. To reduce the complexity of the dialog problem, ALFRED is always connected to some task-oriented system (TOS) and the communication between ALFRED and a human user is related to the operation of that task-oriented system. The ultimate task of ALFRED is to communicate with a human user in natural language to control different task-oriented systems. The implementation details and the architecture for ALFRED are discussed in [17].

In general, ALFRED interprets the user requests and creates desires to accomplish the requests. Based on knowledge and availability of time and resources, ALFRED creates intentions to achieve the desires and thus issues the appropriate instruction to the TOS. In order to determine whether an instruction that it issued to a TOS has resulted in the desired action, ALFRED creates an expectation regarding the outcome, whenever it initiates an action. It then compares the actual result that it obtains from either the user or the TOS with the expected outcome.

The current implementation of ALFRED has been successfully interfaced with the following domains:

- Simulated Pool: ALFRED controls the temperature settings of a pool based on user needs.
- Movie Player: ALFRED plays different movies based on user requests.
- Toy Train: ALFRED moves various trains to different cities based on user requests.
- Simulated House: ALFRED controls different appliances in a house based on user needs.
- Home Designer: ALFRED helps the user create and move different objects in a house model.

- Chess Player: ALFRED plays chess for the user by sending the user requested moves to a chess program.

Switching from one domain to another is achieved by loading the domain specific information which includes the names of the domain objects and the syntax and semantics of the valid domain instructions.

6.1 Capabilities due to Active logic

Some of the skills that the current version of ALFRED exhibit and how the properties of active logic contribute towards implementing these skills are discussed below:

1 **Recognizing a pause**:

The time-sensitive, history-tracking Active logic provides ALFRED the capability to notice the passage of idle-time and thus recognize pauses in conversation. If no new formulas (observations or inferences) are added into the logic at step i then, the agent notes at step $i+1$ that it was idling in step i. ALFRED associates idle-time with pauses in the on-going conversation and makes use of such pauses to capture the turn in the conversation if or when it needs to do so. This is illustrated in the following example:

```
User:  Send Bullet to Balty
ALFRED: I do not know the city Balty. Which city is Balty?
User:  Send Metroliner to Baltimore.
ALFRED: Ok. [sends Metroliner to Baltimore]
ALFRED: Could you tell me which city is Balty?
```

Here, since ALFRED did not get a reply from the user for a previous question, when there is a pause in the conversation, it takes the opportunity to reiterate its question.

2 **Recognizing the unknown**:

Using the introspection rule (5), ALFRED can determine that it does not know something. This knowledge, that it does not know X can be used to acquire X where X could be the meaning or referent of a word. In the above example, once ALFRED determines that it does not know the referent for the city "Balty" it deliberately makes an effort to identify the referent for the city "Balty" by engaging in a meta-dialog with the human user.

3 **Learning new words**:

In active logic new observations can be introduced into the reasoning process at any step. This allows ALFRED to learn new words as in the following example.

```
User:   Balty is Baltimore.
ALFRED: Ok. [learns the new word Balty]
User:   Send Metroliner to Balty.
ALFRED: Ok. [sends Metroliner to Balty]
```

In this example, ALFRED learns the new word Balty and makes use of this new information in the ongoing conversation. In other words, the newly acquired information is used in the inference mechanism used to find the referent for the city Balty during the interpretation of the second utterance. Alma/Carne makes such onboard reactive reasoning feasible by allowing new formulas (observations) to be added at any step and using all formulas at one step to derive the formulas in the next step.

4 **Interpreting elliptical utterances**:

Active logic's history mechanism provides the framework for maintaining the context of the ongoing conversation. At any point in time, ALFRED has access to all the interpretations that it has assigned to past user utterances. In addition, it has access to its own beliefs, desires, intentions, expectations and achievements at all previous steps. This means that ALFRED can know the expectations that have been achieved and those that have not yet been achieved. ALFRED uses this information to interpret elliptical utterances. For instance, in the example below, ALFRED expects a response from the user that would provide the meaning of the word Richy. So, ALFRED is able to interpret the word "Richmond" in the context of the ongoing conversation.

```
User:   Send Metroliner to Richy.
ALFRED: I do not know the city Richy. Which city is Richy?
User:   Richmond.
ALFRED: Ok. [learns that Richy is Richmond]
```

5 **Resolving the references**:

Reference resolution involves identifying the objects that are referenced in the ongoing conversation. ALFRED identifies the referent by introspection, interacting with the task-oriented system and/or getting help from the user. This is illustrated in the following dialog where initially introspection fails to provide the referent for the word "Metro" and hence the agent asks the user for help. Once it learns that Metro is Metroliner, the agent identifies the referent for the word "Metro" in the ongoing conversation, using introspection alone. In the last scenario, introspection provides the information that the Chicago train refers to the train at Chicago. The agent, then interacts with the task-oriented system to identify the train at Chicago.

```
User:   Send Metro to Baltimore
ALFRED: I do not know the train Metro. Which train is Metro?
User:   Metro is Metroliner
ALFRED: Ok.  [sends Metroliner to Baltimore]
User:   Send Metro to Richmond
ALFRED: Ok.  [sends Metroliner to Richmond]
User:   Send the Chicago train to Richmond
ALFRED: Ok.   [sends Bullet to Richmond]
```

6 **Identifying communication problems**:

Active logic's ability to detect contradictions provides the agent the capability to identify communication problems and hence attempt appropriate actions to correct these problems. For instance, in an example originally described in [31], the user initially says:

"Send the Boston train to New York"

But the agent moves a train other than the one the user intended by the phrase "the Boston train". Thus the user says:

"No, send the *Boston* train to New York"

This creates a contradiction—do X, don't do X—which the agent must recognize and consider. In order for the agent to properly interpret the correction in the latter portion of the above utterance, it must come to recognize that "no" is not a change of mind on the user's part (as it might have been), nor is it an incoherent self-contradiction by the user (don't send it and do send it), but rather an implicit correction of the intervening action taken by the agent (sending, say, the Burlington train instead of the Boston train). Recognizing this, the agent reconsiders its initial interpretation of the phrase "Boston train", and chooses another appropriate candidate to move.

In the example given above, the interpretation of the first utterance causes the belief $move(train1, new\_york)$ to be asserted into the logic at some step i. The user's "No", causes the agent to believe that the interpretation that it originally assigned to the first utterance is not correct and hence $not(move(train1, new\_york)$ gets asserted at some step j where $j > i$. This causes the contradiction rule (4) to be triggered and at step $j + 1$, the agent recognizes that there is a contradiction in its beliefs. The agent then initiates its contradiction-handling mechanism to decide which of the contradictands can be trusted. In this particular example, the contradiction handler causes the latter belief to be trusted over the earlier belief. (The assumption here is that a user's later commands overrides the earlier commands). Thus, the agent comes to believe $not(move(train1, new\_york))$. It uses this belief to eliminate $train1$ from the candidate list, when it attempts to resolve the reference "the Boston train" the second time.

8 **Switching between domains**:

Active logic allows new formulas to be added or existing formulas to be deleted at any step. This means that the agent can easily be switched from one domain to the other at step $i + 1$ by doing the following operations at step $i + 1$ (i) delete all the formulas in the KB that exist at step i and (ii) load the set of domain-specific formulas for the new domain into the logic. As a result the following dialog is feasible.

```
User:   Move Metroliner to Baltimore
ALFRED: I do not know the command ‘‘Move’’. What does ‘‘move’’ mean?
User:   Change to chess domain
ALFRED: OK.  [connects to chess]
```

```
User:   Move the king from e2 to e3
ALFRED: Ok.  [moves king from e2 to e3]
```

In this example, when ALFRED is connected to the simulated trains system, since "move" is not a valid command, it asks the user the meaning of the word "move". However, when it is connected to the chess domain. "move" is a known command and hence it issues the appropriate instruction to the chess player to make the move.

7 Conclusion

Our primary claim is that the demands of real-time commonsense reasoning require a logic that is (i) perturbation tolerant, (ii) contradiction tolerant and (iii) time situated. We believe that active logic has the necessary features for use in commonsense reasoning applications.

More specifically, as illustrated by our dialog system ALFRED, we have shown that one can use active logic to enhance the interactive capability of a task-oriented computer system, by providing the ability to detect and recover from miscommunication problems, including ambiguous references, incompatible or contradictory user intentions, and the use of unknown words.

This technology can already be usefully applied to current application domains, such as home-control software, and we expect that the techniques employed can be refined and extended to handle more sophisticated domains.

We are working on methods of structuring and manipulating conceptual relations to allow ALFRED to learn not just new words for known objects or actions, but also genuinely new objects and concepts. We are also continuing to study human dialog behavior for clues as to what strategies can be employed by competent language users to ensure adequate communication even under difficult circumstances. Our long range goal is to be able to emulate the various behaviors and clarification devices employed by a human learning a foreign language or a novice learning a new subject.

References

[1] James F. Allen and George Ferguson. Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation*, 4(5), 1994.

[2] C. Andersen, D. Traum, K. Purang, D. Purushothaman, and D. Perlis. Mixed initiative dialogue and intelligence via active logic,. In *Proceedings of the AAAI'99 Workshop on Mixed-Initiative Intelligence*, 1999.

[3] Michael L. Anderson, Andrew Fister, Bryant Lee, and Danny Wang. On the frequency and types of meta-language in conversation: a preliminary report. In *Proceedings of the 14th Annual Conference of the Society for Text and Discourse*, 2004.

[4] Michael L. Anderson, Darsana Josyula, and Don Perlis. Talking to computers. In *Proceedings of the Workshop on Mixed Initiative Intelligent Systems, IJCAI-03*, 2003.

[5] Peter Bohlin, Johan Bos, Staffan Larsson, Ian Lewin, Colin Matheson, and David Milward. Survey of existing interactive systems. *TRINDI Technical Report*, D1.3:1–57, 1999.

[6] F. Brown, editor. *The Frame Problem in Artificial Intelligence*. Morgan Kaufmann, 1987.

[7] Justine Cassell, Joseph Sullivan, Scott Prevost, and Elizabeth Chuchill. *Embodied Conversational Agents*. MIT Press, Cambridge, MA, 2000.

[8] Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13:259–294, 1989. Also appears as Chapter 5 in [**?**].

[9] Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979.

[10] Jon Doyle. *A Model for Deliberation, Action, and Introspection*. PhD thesis, Massachusetts Institute of Technology, 1980.

[11] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[12] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA, 1988.

[13] J. Gurney, D. Perlis, and K. Purang. Interpreting presuppositions using active logic: From contexts to utterances. *Computational Intelligence*, 1997.

[14] Jerry Hobbs. Ontological promiscuity. In *Proceedings ACL-85*, pages 61–69, 1985.

[15] C. H. Hwang and L. K. Schubert. Episodic Logic: A comprehensive, natural representation for language understanding. *Minds and Machines*, 3:381–419, 1993.

[16] Darsana Josyula, Michael L. Anderson, and Don Perlis. Towards domain-independent, task-oriented, conversational adequacy. In *Proceedings of the Eighteenth international Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1637–8, 2003.

[17] Darsana P. Josyula, Michael L. Anderson, and Don Perlis. Domain-Independent Reason-Enhanced Controller for Task-ORiented systems - DIRECTOR. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[18] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[19] John McCarthy. Elaboration tolerance. In *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, 1998.

[20] Susan W. McRoy, Susan Haller, and Syed Ali. Uniform knowledge representation for language processing in the b2 system. *Journal of Natural Language Engineering*, 3(2/3):123–145, 1997.

[21] M. Miller. *A View of One's Past and Other Aspects of Reasoned Change in Belief.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.

[22] M. Miller and D. Perlis. Presentations and this and that: logic in action. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, Boulder, Colorado, 1993.

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

[24] Massimo Poesio and David R. Traum. Conversational actions and discourse situations. *Computational Intelligence*, 13(3), 1997.

[25] K. Purang. *Systems that detect and repair their own mistakes.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 2001.

[26] K. Purang, D. Purushothaman, D. Traum, C. Andersen, D. Traum, and D. Perlis. Practical reasoning and plan execution with active logic. In *Proceedings of the IJCAI'99 Workshop on Practical Reasoning and Rationality*, 1999.

[27] Matthew Purver. A clarification request markup scheme for the BNC. Technical Report TR-02-02, Department of Computer Science, King's College London, February 2002.

[28] Matthew Purver, Jonathan Ginzburg, and Patrick Healey. On the means for clarification in dialogue. In R. Smith and J. van Kuppvelt, editors, *Current and New Directions in Discourse and Dialogue.* Kluwer Academic Publishers, 2002.

[29] N. Rescher and A. Urquhart. *Temporal Logic.* Springer-Verlag, New York, 1971.

[30] D. Traum and C. Andersen. Representations of dialogue state for domain and task independent meta-dialogue. In *Proceedings of the IJCAI99 workshop: Knowledge And Reasoning in Practical Dialogue Systems*, pages 113–120, 1999.

[31] David Traum, Carl Andersen, Yuan Chong, Darsana Josyula, Michael O'Donovan-Anderson, Yoshi Okamoto, Khemdut Purang, and Don Perlis. Representations of dialogue state for domain and task independent meta-dialogue. *Electronic Transactions on Artificial Intelligence*, 3:125–52, 1999.

[32] David R. Traum, L. K. Schubert, M. Poesio, N. G. Martin, M. Light, C. H. Hwang, P. Heeman, G. Ferguson, and J. F. Allen. Knowledge representation in the TRAINS-93 conversation system. *International Journal of Expert Systems*, 9(1):173–223, 1996.