# Practical Reasoning and Plan Execution with Active Logic *

**K. Purang, Darsana Purushothaman, David Traum, Carl Andersen, Don Perlis**
**Computer Science Department, University of Maryland**
**College Park, MD 20742 USA**
**phone: +1 (301) 405-1139 fax: +1 (301) 405-6707**
**{kpurang,darsana,traum,cfa,perlis}@cs.umd.edu**

## Abstract

We present an approach toward design of a rational agent, integrating aspects of theoretical reasoning, practical reasoning, and reasoning about and executing plans. The approach uses Active Logic, which combines reactivity and logical inference, taking resource bounds into account, and providing mechanisms for handling contradiction. We augment this logic with a formalization of practical reasoning and plan execution, which also makes uses of contradiction handling abilities to cope with plan failure. We conclude with a description of a preliminary implementation and plans for embedding that within a dialogue system.

## 1 Introduction

In this paper, we present an approach toward design of a rational agent, integrating aspects of theoretical reasoning, practical reasoning, and reasoning about and executing plans. The approach, based on Active Logic [Elgot-Drapkin and Perlis, 1990], couples a view of belief as resulting explicitly from inference (or observation), with a resource-bounded approach to inference. Thus not all consequences of an agent's beliefs will be believed (currently), and doing the inference necessary to establish these consequences as beliefs will take time, during which other changes to the world may happen. Also key to this approach is an ability to handle contradictory beliefs in a robust manner. The inference procedure is set up so that contradictions in beliefs will have only limited (and recoverable) effects on the inferability of other beliefs. Noticing contradictions drives much of the further inference, including both theoretical and practical reasoning.

We model the components of practical reasoning in a fairly intuitive, commonsense way rather than attempting a comprehensive account of the tricky issues involved in such notions as knowledge, intentions and obligations. Beliefs are represented directly as a sequence of sets of propositions (one set per time point), and also using an introspection operator. Part of the beliefs includes a theory of action, including plan recipes with pre- and post-conditions and linear decompositions including sub-actions and subgoal states. Practical reasoning is accomplished using modalities *Goal* (an end state), *Adopt* (marking the current state of execution of a plan), and *Expect* (marking the anticipated results of an adopted plan). A key feature of the approach is a natural integration of inference, (normal) plan execution, detection of plan failure, and re-planning and acting.

In the next section we highlight some of the main features of active logic. We then describe, in Section 3, an initial formalization for reasoning about action and practical reasoning within active logic. In section 4, we present initial efforts at implementing an agent using an architecture that gives active logic sensors and effectors to interact with the world (the electronic world). We conclude with some future directions, using this agent as the basis for a natural language dialogue system.

## 2 Active Logic

Active logics were developed as a means of combining the best of two worlds – inference and reactivity – without giving up much of either. This requires a special evolving-during-inference model of time.

A key example is deadline-coupled reasoning. An approaching deadline must be factored into one's reasoning, even seen as an evolving part of that reasoning, rather than as a separate concern outside the reasoning process. Thus the remaining time (deadline – current_time) shrinks steadily as one attempts to find a solution to the problem.

The formal changes required for such a logic are, in some respects, quite modest. The language can be that of a first-order logic, perhaps augmented with names for expressions to facilitate meta-reasoning. The principal change is that inference rules become time-sensitive. The most obvious case is that of reasoning about time itself, as in the rule

*This research was supported in part by the National Science Foundation (IIS-9724937)

```
i:              Now(i)
               ----------
i+1:            Now(i+1)
```

The above indicates that from the belief (at time i) that the current time is in fact $i$, one concludes that it *now* is the later time $i+1$. That is, time does not stand still as one reasons.

For instance, suppose you are driving en route to the airport and planning details of your route as you go. You wonder whether to take the more direct but more heavily traveled road, or another. There are many facts to consider (time of day, day of week, radio traffic and weather reports) and many implications to ferret out (the radio is not broadcasting any traffic news, but it may be due to lack of such news or to their obsession with announcing a baseball game, etc). You quickly realize that your flight will be gone before you can figure out ramifications to all these subtleties. So you decide to stop worrying about the best of all possible routes, and instead content yourself with any one that seems likely to work.

Using active-logic inference rules such as that above, deadline-coupled reasoning has been formalized and applied to planning problems (see [Nirkhe *et al.*, 1997] where time of plan-enactment is crucial.

Technically, an active logic consists of a first-order language, a set of time-sensitive inference rules, and an observation-function that specifies an environment in which the logic "runs". Thus an active logic is not pure formalism but is a hybrid of formal system and embedded inference engine, where the formal behavior is tied to the environment via the observations and the internal monitoring of time-passage (see [Elgot-Drapkin and Perlis, 1990] for a detailed description).

In the above example, the reactivity is not to external events but rather to the universal event of time-passage vis-a-vis one's own reasoning. One can conceptualize this externally in terms of looking at a clock but this is not necessary or particularly helpful. On the other hand, external events are often quite important, as we discuss later.

Active logics are able to react to incoming information (including dialogue utterances by a collaborative partner) while reasoning is ongoing, blending new inputs into its inferences without having to start up a new theorem-proving effort. Thus, any helpful communications of a partner (or user) – whether as new initiatives, or in response to system requests – can be fully integrated with the system's evolving reasoning. Similarly, external observations of actions or events can be made during the reasoning process and also factored into that process.

Thus the notion of theorem for active logics is a bit different from that of more traditional logics, in several respects:

1. **Time sensitivity.** Theorems come and go; that is, a wff once proved remains proved but only in the sense of its being a historical fact that was once proved. That historical fact is recorded for potential use, but the wff itself need not continue to be available for use in future inferences; it might not even be reprovable, if the "axioms" (belief) set has changed sufficiently. As a trivial example, suppose $Now(noon) \rightarrow Lunchtime$ is an axiom. At time t=noon, Now(noon) will be inferred from the rule given earlier, and Lunchtime will be inferred a step later. But then Now(noon+1) is inferred, and Lunchtime is no longer inferable since its premise Now(noon) is no longer in the belief set. Lunchtime will remain in the belief set until it is no longer "inherited"; the rules for inheritance are themselves inference rules. One such involves contradiction; see next item.

2. **Contradictions.** If a direct contradiction ($P$ and $\neg P$) occurs in the belief set at time t, that fact is noted at time t+1 by means of the inference rule

```
t:                P , ~P
             ------------------
t+1:         Contra(i+1,P, ~P)
```

Also, as a consequence of the contradiction having been noted, neither of these instances of $P$ or $\neg P$ will be used in future inferences. Thus the logic is partially shielded from using particularly blatant contradictands. $P$ and $\neg P$ remain theorems in the sense of having been proved, but are not available for further inference. For instance, if Lunchtime is contradicted by $\neg Lunchtime$, neither of these is inherited to the next time step; but $\neg Lunchtime$ may well be reproven and thus in a sense "wins". This can occur as a result of a further axiom, such as $\neg Now(noon) \rightarrow \neg Lunchtime$. We can also provide contradiction-handling axioms that are domain independent or domain dependent for specific domains. These can use information about the domain but also information about the knowledge base and proofs to arbitrate between the contradictands.

Much more subtle effects can occur from this feature. To return to the airport example: you are driving en route to the airport and planning details of your route as you go. Then your car gets a flat tire. Rather than complete your original plans, it is time to make major revisions, in fact even to change your goals for the time being.

Truth maintenance systems [Doyle, 1979] also tolerate contradictions and resolve them typically using justification information. This happens in a separate process which runs while the reasoning engine is waiting. We do not think that this will not work in general since the reasoning needed to resolve the contradiction will depend on the very information that generated it. Reasoning and the resolution of contradictions have to take place in the same reasoning process.

3. **Defaults.** Defaults can be given a straightforward representation in an evolving-time framework:

```
t:              ~Known(~P), Q
              -----------------------
t+1:                  P
```

Here from the facts that Q, and that $\neg P$ is not a belief at time t, P is inferred. This avoids the decidability issues of traditional default mechanisms, since only a linear lookup in the belief set for time t 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, these then can be treated as any other contradictands.

4. **Observations.** In active logic the flat tire in the previous example can be represented in terms of *observations*. And the reasoning simply goes on with this new information. There is no executive subsystem that turns off the route planner midstream and starts up a new planning action. Rather there is a single stream of reasoning, which can monitor itself by looking backwards at one moment to see what it has been doing in the past, including the very recent past. If the previous few steps in some way conflict with new information, then the next few steps can be devoted to sorting out enough of the apparent mismatch to allow a decision as to how to proceed. All of this is carried out in the same inferential process as the original planning, without the need for level upon level of meta-reasoners. This is not to say that there is no metareasoning here, but rather that it is "in-line" metareasoning, all at one level. The advantages of this are (i) simplicity of design, (ii) no infinite regress, and (iii) no reasoning time at higher levels unaccounted for at lower levels.

   A potential disadvantage is the possibility of vicious self-reference. This matter is a topic of current investigation. However, another major advantage of such time-sensitive in-line metareasoning is that inconsistency in one's beliefs need not cause serious problems in general. The reason is largely that given above: a conflict in the reasoner's beliefs can be noted by the reasoner as a new belief, and the latter can lead to a decision to encapsulate the conflicting beliefs so that they do no harm. Now this cannot be a fully general process, since identifying contradictions is at best semi-decidable. However, deeply hidden contradictions usually do little harm; and so we have concentrated on inference rules for "direct" contradictions, that is, belief pairs that surface in the form $P$ and $\neg P$; see [Miller, 1993] for details including a theorem providing a rather general case in which such in-line metareasoning can cope with direct contradictions.

5. **Evolving state representation.** Another feature that comes directly out of the time-coupled nature of active logics is their ability to represent the evolving status of reasoning and actions. The representation of actions can avail itself of up-to-date time information. Thus an action A can be marked as Planned, Underway, and Done; and the logic can pass from one to another of these as actions are put into execution. Thus active logics not only reason about plans, but can make and execute them while keeping track of this changing state.

   It is easy to write an inference rule that updates at each time step whether a particular plan is currently being started, is already underway, or is completed. More detail than this, such as how long the plan execution has been going on, is also readily inferred. This is important for various purposes, such as:

   (i) avoiding re-initiation of a plan already underway

   (ii) assessing whether one is spending too much time on a goal

   (iii) distinguishing between various instances of a plan, one underway, another finished, perhaps a third and fourth on the to-do list. This is useful for repetitive activities, such as transporting objects one by one, or keeping track of how many times one is performing a certain action (for instance in dialogue, where one may repeat a request a few times for emphasis or as a reminder, but not indefinitely, without a kind of breakdown of coherence [Suchman, 1987]).

Active logics can be seen either as formalisms per se, or as inference engines that implement formalisms. This double-role aspect is not accidental: it is inherent to the conception of an active logic that it have a behavior, ie, the notion of theoremhood depends directly on two things that are not part of traditional logics: (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.

Active Logic provides the theoretical reasoning component of our framework. However it also has many convenient features for practical reasoning, particularly the time-situatedness and contradiction handling facilities. This provides a natural mechanism for plan reasoning and acting, as well as failure detection and re-planning. In the next section, we describe a preliminary formalization of reasoning about action and plan execution, using Active Logic.

# 3 Practical Reasoning and Plan Execution

Plan execution architectures (for instance CIRCA [Goldman *et al.*, 1997], ESL [Gat, 1996], PRS [Myers, 1997], RAPS [Firby, 1995]) are generally not based primarily on logic. However, we think that active logics are well suited to serve as plan execution architectures: failures of plans or of actions can be handled naturally as contradictions; the changing state of the world can be represented as time-situated changing beliefs of the agent; the reasoner can use logic to perform arbitrary reasoning. Active logics can therefore provide a uniform platform for reasoning and plan execution.

Using active logics as a plan execution architecture requires one to define representations for plans, goals and actions, to add axioms to describe the plan execution process, and to augment the contradiction handler to take care of the special cases of contradictions caused by plan execution.

We have begun developing a plan execution architecture in active logic. In this section we sketch our preliminary system and present some examples that illustrate it. We are still at the early stages of development, so we do not take complex plans (beyond a sequence of sub-actions) or situations into account yet.

## 3.1 Representations

The notation we use is as follows: predicates and functions are capitalized, variables are not, Greek letters are used for expression variables; $Know$ is a positive introspection predicate; the formulas we present are assumed to be universally quantified unless otherwise noted. We allow quantification over formulas that may be seen as being implicitly quoted. Lists are represented prolog style with [ ], and we use | to denote concatenation of lists. $Now$ is a unary predicate true of the current time step.

Plan recipes are represented as $Plan(name, pre, post, steps)$ where $name$ is the name of the plan, $pre$ is a formula describing the preconditions for the execution of the plan, $post$ is a formula describing the formulas that hold at the successful execution of the plan, $steps$ is a temporally ordered list of steps that constitute the plan. These steps can be either primitive actions that can be executed, or sub-goals, requiring a new plan to be adopted and executed. The plans we currently consider are simple plans with only sequencing of plans or actions allowed.

Actions are represented in a similar way as $Action(name, pre, post, act)$ where $name$ is the name of the action, $pre$ is a formula describing the preconditions for the execution of the action, $post$ is a formula describing the formulas that hold at the successful execution of the action, $act$ is the procedure that is to be executed to implement the action.

Exogenous actions are represented by $Action(name, pre, post, Nul)$. For instance, if the agent is in a train and it depends on the train getting to $X$, this is represented as $Action(GetTo, InTrain(Train1, Time1), At(X, Time2), Nul)$

Goals are represented as $Goal(\phi)$ where $\phi$ is a formula that is to be made to hold. $Goal(\phi)$ holds only when $\phi$ has not been accomplished and no plan has been adopted to achieve $\phi$.This goal can be a maintenance goal if $\phi$ quantifies over time. For example, we would represent keeping the cat fed as $Goal(\forall t\ Fed(Cat, t))$.

Plans that have been adopted and are being executed are represented by $Adopt(name, done, rest, goal)$ where $name$ is the name of the plan, $done$ is a list of those steps executed, $rest$ is the list of the remaining steps, and $goal$ is the goal.

## 3.2 Plan execution axioms

We adopt a plan for execution if its postcondition($\psi$) implies the goal($\phi$), the preconditions($\theta$), are met and the goal is not already true:

$$Goal(\phi) \wedge \theta \wedge Plan(n, \theta, \psi, s) \wedge (\psi \rightarrow \phi) \wedge \neg Know(\phi) \rightarrow \neg Goal(\phi) \wedge Adopt(n, [\,], s, \phi)$$

Note the assertion of $\neg Goal(\phi)$ here. This represents that $\phi$ is no longer a goal that needs to be processed–$Adopt(N, [\ ], S, \phi)$ indicates that $\phi$ is being worked on. The assertion of $\neg Goal(\phi)$ will give rise to a contradiction. This will be resolved by preferring the later formula, in this case $\neg Goal(\phi)$ (see below for more on contradiction resolution). We also require that all adopted plans for the same goal be the same:

$$Adopt(n, i, f, \phi) \wedge Adopt(m, i', f', \phi) \rightarrow n = m$$

If there are two different adopted plans for the same goal, a contradiction will be generated since $\neg n = m$. At this point, one can choose which plan to pursue.

If the precondition of the plan is not known to hold, we make it a goal:

$$\begin{array}{l} Goal(\phi) \wedge \neg Known(\phi) \wedge \neg Known(\theta) \wedge \\ Plan(n, \theta, \psi, steps) \wedge (\psi \rightarrow \phi) \rightarrow Goal(\theta) \end{array}$$

Here, $\phi$ is still a goal so that whenever the preconditions are made true, the main plan will be started.

We now consider executing the plan. If the next step of the plan is an action, we wait until the previous step of the plan is completed and verify that the preconditions of that step hold before executing the action. $Done(act)$ is asserted in the knowledge base once action $act$ is completed by the procedure execution module. $Do(act)$ causes $act$ to be performed by the agent. $H$ returns the head of a list, $Last$ returns the last element of the list and $T$ returns the tail.

$$\begin{array}{l} (Now(t) \wedge Adopt(n, i, r, \theta) \wedge Done(Last(i)) \wedge \\ \quad H(r) = Action(a, \phi, \psi, act) \wedge \phi) \rightarrow \\ (Do(act) \wedge Adopt(n, i|H(r), T(r)) \wedge \\ \quad Expect(\exists t_1\ t < t_1 \wedge \psi(t_1)) \wedge \neg Adopt(n, i, r, \theta)) \end{array}$$

We assert that we expect that the postconditions will hold sometime in the future. When the action is actually done, we will have confirmation of that by the postconditions being asserted in the knowledge base. If something happens that makes this impossible (for example, if the action fails), the agent will know that it has a problem.

If the next thing on the plan is a goal, we try to plan for it:

$$\begin{array}{l} (Now(t) \wedge Adopt(n, i, r, \theta) \wedge Done(Last(i)) \wedge \\ \quad H(r) = Goal(\phi)) \rightarrow \\ (Goal(\phi) \wedge Adopt(n, i|H(r), T(r), \theta) \wedge \\ \quad \neg Adopt(n, i, r, \theta) \wedge Expect(\exists t_1\ t < t_1 \wedge \phi(t_1))) \end{array}$$

If the preconditions do not hold, we make them a goal.

$$\begin{array}{l} (Adopt(n, i, r, \theta) \wedge Done(Last(i)) \wedge \neg Known(\phi) \wedge \\ Hd(r) = Action(a, \phi, \psi, act)) \rightarrow \quad Goal(\phi) \end{array}$$

If there is nothing left in the plan, we stop.

$$Adopt(n, i, r, \theta) \wedge r = [\ ] \rightarrow \neg Adopt(n, i, r, \theta)$$

This causes a direct contradiction that is resolved by retracting both contradictands.

In the case that we are at the very beginning of a plan, we know that the empty action is always done:

$$Done(Nul)$$

$Goal$ and $Expect$ have some properties of modalities, and the usual rules apply, including $Expect(\phi \wedge \psi) \leftrightarrow (Expect(\phi) \wedge Expect(\psi))$, and the Barcan formula (see for instance [Hughes and Creswell, 1996]), so that we get $Expect(\forall x\ \phi(x)) \leftrightarrow \forall x\ Expect(\phi(x))$. [1]

## 3.3 Contradictions

Some events in the execution of plans depend on the agent noticing contradictions and reacting appropriately. As mentioned above, contradictions in active logic are automatically flagged when both $P$ and $\neg P$ are derived. For plan execution, we also flag contradictions for $Expect(\phi)$ and $\neg\phi$: if the agent expects something to become true and the negation of it is found to be true, there is something wrong with the plan.

The contradictions are processed by a set of axioms that constitute the contradiction handler. These axioms depend on domain information as well as meta-information such as the derivation of the contradictands, their source and the time at which they were first asserted. This information is not explicitly represented in the knowledge base as formulas in the current implementation of active logic, but is instead represented in data structures associated with the formulas. Access functions allow the axioms to reason with these.

Some of the strategies for resolving contradictions between $\phi$ and $\neg\phi$ are: 1. if $\phi$ is of the form $Goal(\psi)$, then we reinstate the later one; 2. if $\phi$ is of the form $Expect(\psi)$ and $\neg\phi$ is $\neg\psi$ and results from an observation, then reinstate the goal that led to the expectation and remove the expectation. The rationale behind these will be made clearer below.

## 3.4 The domain

The domain we use to illustrate this system is part of the Washington area metro system. We assume that our agent is at College Park (CP) and wants to get to Union Station (US). The only train line that passes through CP is the green line. Since part of the green line is still under construction, there is no direct train from CP to US: one has to take the green train from CP to Fort Totten (FT) and there change to a red train that goes from FT to US. However, during rush hour, the green train bypasses FT altogether and goes to US. Therefore we have two plans to get to US from CP: one for rush hour, and one for non rush hour.

The examples we present are first a simple case of the agent getting on the train at CP during rush

[1] We intend to explore the relation between our use of modality in these cases and the uses of modality for agency as in [Belnap and Perloff, 1988; Horty and Belnap, 1995], for example.

hour and getting off at US. The second example we consider is the case of the agent thinking it is rush hour (by default), getting on the train at CP and expecting the train to go up to US. However, it is not rush hour and the train gets to FT and stops there. The agent observes this and that leads to a contradiction. This causes the agent to abandon the original plan and to form a new plan to get from FT to US.

**Plans**

If the agent is at $p$ at time $t$, $At(p,t)$, and there is a direct train $m$ that goes from $p$ to $q$, $DirectTrain(p,q,m)$, and that train is at $p$ at time $t$, $TrainAt(m,p,t)$, then the following plan will result in the agent not being at $p$ but at $q$ at some later time.

$$\begin{array}{ll} Plan(P1 & ,(At(p,t) \wedge DirectTrain(p,q,m) \wedge \\ & \quad TrainAt(m,p,t)), \\ & (\exists t_1 \; t_1 > t \wedge \neg At(p,t_1) \wedge At(q,t_1)), \\ & [\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3]) \end{array}$$

Here, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ stand for actions $A1$, $A2$, and $A3$ that we present below. We use $\mathcal{A}_1$ and so on for the convenience of not having to write the actions here. These are not part of the language.

Another plan is for the case that there is no direct train between the source and the destination:

$$\begin{array}{ll} Plan(P2, & (\exists m_1 m_2 \; At(x,t) \wedge DirectTrain(x,z,m_1) \\ & \quad \wedge \neg DirectTrain(x,y,m_0) \wedge \\ & \quad DirectTrain(z,y,m_2)), \\ & (\exists t_1 \; t_1 > t \wedge \neg At(x,t_1) \wedge At(y,t_1)), \\ & [Goal(\exists t_2 \; t_2 > t \wedge At(z,t_2)), \\ & \quad Goal(\exists t_3 \; t_3 > t \wedge At(y,t_3))]) \end{array}$$

**Primitive Actions**

The primitive actions used in the plans are as follows.

If we are at the station at the same time as the train is, we can get on the train and we will no longer be considered to be at the station.

$$\begin{array}{ll} Action(A1, & (At(x,t_0) \wedge TrainAt(m,x,t_0)), \\ & (InTrain(m,t_0+1) \wedge \neg At(x,t_0+1)), \\ & GetOnTrain(m,x,t_0)) \end{array}$$

A2 is an exogenous event: if the agent is in the train at station $X$ and there is a direct connection to station $Y$, then, at some later time, the train and the agent will end up in $Y$.

$$\begin{array}{ll} Action(A2, & (InTrain(m,t_0) \wedge TrainAt(m,x,t_0) \wedge \\ & \quad DirectTrain(x,y,m)), \\ & (\exists t_2, \; t_2 > t_0 \wedge TrainAt(m,y,t_2) \wedge \\ & \quad InTrain(m,t_2)), \\ & Nul) \end{array}$$

The third action is to get off the train: if we are in the train and it is at a station, we can get off the train and we will be at the station.

$$\begin{array}{ll} Action(A3, & (InTrain(t_0) \wedge TrainAt(m,y,t_0)), \\ & (\neg InTrain(t_0+1) \wedge At(y,t_0+1), \\ & GetOffTrain(m,y,t_0)) \end{array}$$

**Domain Information**

During rush hour, there is a direct train from College Park to Union Station:

$$\begin{array}{l} Now(t) \wedge RushHour(t) \rightarrow \\ \quad DirectTrain(CP,US,Green) \end{array}$$

It is usually not rush hour:

$$Now(t) \wedge \neg Know(\neg RushHour(t)) \rightarrow \neg RushHour(t)$$

When it is not rush hour, there is a green train from CP to FT and a red train from FT to US:

$$\begin{array}{l} Now(t) \wedge \neg RushHour(t) \rightarrow \\ \quad DirectTrain(CP,FT,Green) \end{array}$$

$$Now(t) \wedge \neg RushHour(t) \rightarrow DirectTrain(FT,US,Red)$$

If the train reaches a terminal station, we have to get off:

$$\begin{array}{l} InTrain(m,t) \wedge TrainTerminus(m,x,t) \rightarrow \\ \qquad Do(GetOffTrain(m,x,t)) \end{array}$$

This is an instance of an action being done depending directly on the state of the agent and not being part of a plan. When getting off the train succeeds, the following: $At(x,t)$ and $\neg InTrain(m,t)$ are added to the knowledge base.

When a train reaches the terminus, it goes nowhere else:

$$\begin{array}{l} TrainTerminus(m,x,t_0) \wedge \neg x = y \wedge t_1 > t_0 \rightarrow \\ \qquad \neg TrainAt(m,y,t_1) \end{array}$$

### 3.5 Example 1

We present axioms for the first example: The goal is to get to Union Station, the agent is at College Park at time 0, and it is rush-hour and the train is at College Park.

$$\begin{array}{l} Goal(At(US,T)) \wedge Now(0) \wedge At(CP,0) \wedge \\ RushHour(0) \wedge TrainAt(Green,CP,0) \end{array}$$

We do not show the details of the plan execution, but highlight some of the aspects. At time 1, plan $P1$ is adopted and at the next step, the action $GetOnTrain(Green,CP,0)$ is executed. This succeeds and adds to the knowledge base the following: $Done(GetOnTrain(Green,CP,0),3) \wedge InTrain(Green,3), \neg At(CP,3)$. Since $Done(GetOnTrain(Green,CP,0),3)$ is a precondition for the next action, we can now execute it. However, the next action is a Nul action, so we can only wait until the preconditions for the action after are satisfied and we assert the expectations at this point: $Expect(\exists t_1 \; t_1 > 3 \wedge TrainAt(Green,US,t_1) \wedge InTrain(m,t_1))$. Later, say at time 10, the train does get to US, and these expectations are observed to be true. The agent then executes the last step of the plan, which is to get off the train, and that results in asserting $At(US,11)$ in the knowledge base.

### 3.6 Example 2

In this case too, the agent is at College Park and thinks it is rush hour and gets on the train just as before. Once it does get on the train, we expect $Expect(\exists t_1\ t_1 > 3 \wedge TrainAt(Green, US, t_1) \wedge InTrain(m, t_1))$. From this we can derive that $Expect(\exists t_1 TrainAt(Green, US, t_1))$.

However when the agent gets to action A2, the train reaches the terminus at FT. The agent observes $TrainTerminus(Green, FT, 10)$ (assume the time is 10). This leads to the agent getting off the train $Do(GetOffTrain(Green, FT, 10))$ which results in $At(FT, 10)$. The agent also concludes that this train is not getting anywhere: $\forall t, p\ \neg FT = p \rightarrow \neg TrainAt(Green, p, t)$. In particular, this train is not getting to Union Station: $\forall t\ \neg TrainAt(Green, US, t)$. This however contradicts the expectation that the green train will indeed get to Union Station: $Expect(\exists t_2\ TrainAt(Green, US, t_2))$.

The plan has failed and since the agent is executing the plan, it cannot back up to a preceding state–it has to try to accomplish its goal from its current state. A contradiction is generated and the handling of the contradiction results in reinstating the original goal $Goal(At(US, t))$ and the removal of the expectations. Now the agent is at FT and has the goal of getting to US and knows there is a red train that goes there directly, so it can get to Union Station using the same procedure as in the previous example.

## 4 Alma/Carne: An Active Logic Agent

Our concern is not just with "theoretical" practical reasoning, but with using this reasoning about rationality in a "practical" way, as a specification of an artificial agent. We have thus been constructing a test-bed system both for testing the ideas above and for attempting to apply the general approach for practical problems such as human-computer natural language dialogue. Alma/Carne is an implementation of active logic that includes a facility for representing and using procedural knowledge. This gives the active logic the ability to interact in arbitrary ways with the environment and to execute procedures the details of which are of no interest inferentially. Alma and Carne are separate processes with Alma the reasoner and Carne the action execution module. This gives us a clear separation between the procedural and the declarative parts of the model of the agent while requiring declarative knowledge about the procedures to be explicitly stated.

### 4.1 Alma

Alma implements active logic and is the repository for declarative knowledge in the agent. All inferences and all decisions to act are done in Alma, controlled by domain axioms and active logic rules of inference. Alma has a few features that enhance the efficiency of the logic including: 1. applying the inference rules to new formulas only; 2. allowing the programmer to specify in what sorts of proofs each formula is to be used (forward or backward or both); 3. allowing the programmer to specify policies that determine which inferences to actually do at each step.

The problem of controlling the logic is a crucial one, which will get worse as the agent is used in more realistic settings and these features are just the start of our attempt to address this problem.

Alma also has the capability to interact with Carne, in particular, using Carne to "execute" basic actions. We describe that following a description of Carne.

### 4.2 Carne

Carne contains the procedural knowledge of the reasoner. It allows the programmer to specify programs in Prolog that fall into the following main categories:

- Programs triggered by Alma to effect a change in the environment.
- Programs that are responsive to events in the environment and that automatically update Alma's knowledge base with observations.
- Programs that do computations on behalf of Alma.

These give Alma the ability to effectively interact with the world and to offload resource intensive computations to a separate process. A simple interface is used to link Alma and Carne.

### 4.3 The Alma/Carne interface

On the Alma side, there are special purpose rules of inference and predicates. These predicates can be used in axioms to initiate programs in Carne, and to reason about the status of the programs.

$call(\phi, Id)$If a formula of the form $call(\phi, Id)$ is derived in Alma, an inference rule comes into play that sends a message to the Carne process for it to execute program $\phi$ (which, of course, has to be known to Carne). The rule also results in the assertion $doing(\phi, Id)$ in the knowledge base. The $Id$ is a unique identifier used to distinguish between multiple invocations of the same program with the same arguments. An alma rule to perform an action of a plan would be to call a program whenever a $Do(Act)$ proposition of the appropriate type is inferred.

$doing(\phi, Id)$ This asserts that Carne is in the process of executing $\phi$.

$done(\phi, Id)$ Once the program has completed successfully in Carne, a message is sent to Alma that results in the assertion of $done(\phi, Id)$ in the knowledge base and the deletion of $doing(\phi, Id)$ (although that remains in the Alma history).

$error(\phi, Id)$ In case the program fails to execute in Carne, $error(\phi, Id)$ is added to and $doing(\phi, Id)$ is deleted from the Alma database.

These predicates track the status of the programs in Alma and enables decisions to be made about actions as described in the previous section.

On the Carne side, a Prolog predicate $af$ (add formula) is provided to the Carne programs that allows them to assert formulas to the Alma knowledge base. This facility is independent of the above status predicates and is used to assert the results of computations and to include input from the environment, into Alma in the appropriate form. Similarly, *df* (delete formula) can be used by Carne programs to remove formulas from the Alma knowledge base.

## 5 Current and Future Work: A Conversationally Adequate Dialog Agent

Using the Alma/Carne implementation, we are designing and implementing a natural-language-dialogue and commonsense-reasoning engine that has a heavy emphasis on metareasoning [Traum and Andersen, 1999]. The hypothesis we wish to test is that metareasoning is essential to flexible discourse and cognition, in which (miscommunication and other) errors must be detected and repaired during the same episode of reasoning (see [Perlis *et al.*, 1998]). An agent capable of doing this will have to reason with and represent: (1) ongoing time; (2) history; (3) linguistic objects; (4) meanings; (5) contradictions.

The architecture we have designed involves traditional modules (e.g., speech-processor, parser, dialogue manager, problem solver, output/action manager), but organized in terms of logical and non-logical behaviors. Thus our logic engine, Alma, receives and sends communications from the rest of the system (via Carne – whose only job is to facilitate such internal messages, see Figure 1). As has been suggested often before (e.g., [Rieger, 1974]) we view dialogue as simply one special kind of problem-solving.

One major ongoing application of active logics is that of building a "conversationally adequate" dialogue agent. Conversationally adequate agents should be able to engage in "free-ranging" conversation: successfully exchanging information with another agent over the course of a conversation covering any arbitrary topic. Such an agent will have the ability to learn in McCarthy's sense of advice-taking, via conversation [McCarthy, 1958]. We hypothesize that the ability to use meta-reasoning (coupled with other crucial skills like learning) to correct errors is an ability that, once sufficiently sophisticated, allows agents to engage in free-ranging conversation.

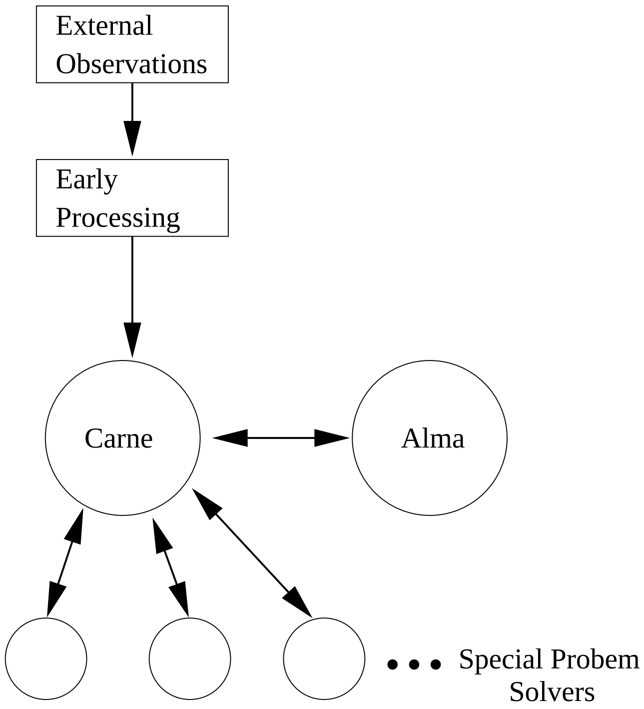


Figure 1: The conversational agent architecture

Preliminary work on applying active logics to problems in language processing has been done [Gurney *et al.*, 1997; Perlis *et al.*, 1996], and we have proposed an abstract view of how we would build such a conversationally adequate agent [Perlis *et al.*, 1998]. We view metareasoning to be a crucial part of that type of agent and believe that active logics are well suited for that. We are currently investigation use of the plan execution framework presented above in addressing dialog performance.

## References

[Belnap and Perloff, 1988] N. D. Belnap and M. Perloff. Seeing to it that: a canonical form for agentives. *Theoria*, 54:175–199, 1988.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[Elgot-Drapkin and Perlis, 1990] 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.

[Firby, 1995] R. J. Firby. The RAP language manual. Animate Agent Project Working Note AAP-6, University of Chicago, 1995.

[Gat, 1996] E. Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Procs. of the AAAI Fall Symposium on Plan Execution*. AAAI Press, 1996.

[Goldman *et al.*, 1997] R. P. Goldman, D. J. Musliner, M. S. Boddy, and K. D. Krebsbach. The circa model of planning and execution. In *Working Notes of the AAAI Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, 1997.

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

[Horty and Belnap, 1995] J. F. Horty and N. D. Belnap. The deliberative stit: A study of action, omission, ability, and obligation. *Journal of Philosophical Logic*, 24(6):583–644, 1995.

[Hughes and Creswell, 1996] G. E. Hughes and M. J. Creswell. *A new introduction to modal logic*. Routledge, 1996.

[McCarthy, 1958] J. McCarthy. Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, Teddington, England, 1958. National Physical Laboratory.

[Miller, 1993] 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.

[Myers, 1997] K. L. Myers. *User Guide for the Procedural Reasoning System*. SRI International,, Menlo Park, CA, 1997.

[Nirkhe *et al.*, 1997] 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.

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

[Perlis *et al.*, 1998] D. Perlis, K. Purang, and C. Andersen. Conversational adequacy: Mistakes are the essense. *International Journal of Human Computer Studies*, pages 553–575, 1998.

[Rieger, 1974] C. Rieger. *Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural-Language Utterances*. PhD thesis, Department of Computer Science, Stanford University, Palo Alto, California, 1974.

[Suchman, 1987] Lucy A. Suchman. *Plans and Situated Actions*. Cambridge University Press, 1987.

[Traum and Andersen, 1999] 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.