Alma/Carne: implementation of a time-situated meta-reasoner

K. Purang
kpurang@yahoo.com

Abstract

Agents need to operate in dynamic situations where the information they have about the world is incomplete, uncertain and quite possibly false. Active logic has been designed with capabilities that enable these aspects of the world to be taken into account, notably time-situatedness, contradiction tolerance and meta-reasoning. This paper presents a general-purpose implementation of active logic, Alma/Carne that is meant to be a basis on which to build and experiment with such agents. We illustrate the use of Alma/Carne in the implementation of a non-monotonic reasoner that is computable and that has been successfully tested on a large number of examples from the literature.

1. Introduction

We present Alma/Carne, an implementation of active logic that is time-situated, tolerates contradictions and is capable of meta-reasoning. These features make Alma/Carne a useful tool for specifying and executing agents that operate in a dynamic world with incomplete and uncertain information. In this paper we focus on the contradiction-tolerant and meta-reasoning features of Alma/Carne.

An agent in an incompletely specified dynamic world with uncertain knowledge is bound to make mistakes. In particular, it is bound to have false beliefs. These can lead to an inconsistency in its knowledge base (KB) where it believes both P and $\neg P$. It needs to detect this situation and resolve it appropriately if it is to operate in a reasonable way. The contradiction tolerance and meta-reasoning of active logic allows these circumstances to be detected and allows the contradiction to be reasoned about with the aim of resolving it. This happens while the rest of the inferences of the agent that are not affected by the inconsistency proceed normally. The inconsistency is seen as another fact about the world and is reasoned about just as the logic would reason about birds.

*This work was mostly done while the author was at the University of Maryland College Park and was supported in part by ONR and AFOSR.

Alma (Active Logic MAchine) is the reasoning engine and Carne executes procedures on behalf of Alma. Alma/Carne has been used in some applications and we here describe the implementation of a non-monotonic reasoner in Alma/Carne. This uses computable procedures to jump to conclusions, but in case these are mistaken, they are corrected automatically and the conclusions improve with time. This is an important component for an agent of the kind we are interested in and illustrates some properties and uses of Alma/Carne.

In the next section we introduce active logic. This provides the background for the description of Alma/Carne. We then illustrate some basic behaviors of the system. The non-monotonic reasoner is described after that. We next discuss related work and finally conclude and point to future work.

2. Active logic

In this section we present an overview of active logic. More details can be found in [9]. Active logic was developed to combine inference and reactivity in one formalism. To enable this, active logic has, as part of its language, a fluent that represents the current time and as inference takes place, the value of time changes. This enables the logic to act (and reason) at specific times and to be aware that the reasoning takes time. It also enables the logic to have a history that relates past times to the beliefs the logic held at those times. With this, the logic can reason about its own past reasoning. This meta-reasoning capability is key to our approach to incompleteness, uncertainty and error in the agent's view of the world. Note that active logic is to be viewed as an on-board agent reasoner, rather than as just an external specification for an agent.

2.1. Formalism

The language of active logic is that of a first-order logic, augmented with names for expressions to facilitate meta-reasoning.

The principal change is that inference rules become time-sensitive. For instance, Modus Ponens becomes

```
i:              P, if(P, Q)
               ----------
i+1:                 Q
```

From the beliefs at time i that P and $if(P, Q)$, we get at time $i + 1$ that Q. Inference takes time and this evolution of the database is recorded and can be explicitly reasoned about.

Active logic also represents the current time in the language through a fluent $Now(t)$. The value of t that makes this true changes as the logic executes.

Temporal logics [26, 2, 24] also have a notion of past, present and future, but these do not change as theorems are derived. These are specification logics external to the reasoner. This contrasts strongly with the agent-based on board character of active logic where the value of Now changes as the agent thinks.

2.2. Characteristics of active logic

Active logics are able to react to incoming information while reasoning is ongoing, blending new inputs into its inferences without having to start up a new theorem-proving effort. So 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.** The Now predicate enables the logic to reason with the current time and this makes formulas derivable at some time but not at others. For instance, `if(now(1200), lunch)` active logic will conclude `lunch` only at time `1200`.

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(t+1, P, ~P)
   ```

   Details on handling contradictions are presented later. See also [17].

   Truth maintenance systems [7] 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 work in general since the reasoning needed to resolve the contradiction will depend on the very information that generated it in addition to justification information. Resolution of contradictions is itself, in general, a reasoning process much like any other.

3. **Meta-Reasoning** In active logic, 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. All of this is carried out in the same inferential process, without the need for level upon level of meta-reasoners. A potential disadvantage is the possibility of vicious self-reference. However the contradiction handling capability should be a powerful tool even there.

3. The Alma/Carne system

Active logic has been used for solving several problems [8, 18, 20, 14] by having a new implementation for each solution which is not effective. While individual problems may require some special representations and procedures, all of the active logic solutions share a common set of characteristics.

We implemented *Alma/Carne* [21], with the aim for it to be the core reasoning engine for active logic applications. Alma is the logical reasoning engine and Carne is a process that runs separately from Alma and executes procedures on its behalf. They are implemented for the most in Prolog with Java being used for the graphical interface. The Alma language is used to specify the reasoning domain. It is similar to a first order language with a set of reserved predicates some of which are described below.

Alma/Carne is the core of an active logic application and not the complete application because in addition to the axiomatic domain description, the user can specify non-logical representations and procedures that are specific to the problem to be solved. These representations and procedures can be accessed from the axioms describing the system. This feature gives the system the flexibility to be used in different applications. We do not need all aspects of the solution of a problem to be expressed in logic for Alma/Carne to be used. We can incorporate other forms of reasoning within the system and reason logically about their results. There may be, for instance, parts of the problem that are more appropriately solved using Bayesian networks [19]. In that case, the information needed by the network can be provided by Alma through Carne and the results of the computation returned to Alma through Carne. This also allows Alma/Carne to be embedded in larger systems.

3.1. Top-level control

Alma runs in "steps" which are sequentially numbered. The step number is used as the time value for Alma. In each step, the rules of inference are applied to extend the derivations by at most one inference rule application. This roughly results in an incremental breadth first forward chaining proof procedure. At each step some formulas are

added and some deleted. A formula that is deleted at each step is the $now(t)$ formula. At step n, the KB contains $now(n)$, and this is deleted and $now(n+1)$ is added for the next step, step $n+1$. Alma/Carne is also capable of abckward chaining which is implemented through the forward chaining mechanism.

3.2. Method of inference

The main rule of inference used in Alma is resolution. While this does not result in a complete reasoner in forward chaining, backward chaining does gives completeness for first order logic. The advantage of resolution is that there is no need to choose which rule of inference to apply, the disadvantage, conversely, is that there is less control than if one used natural deduction rules, for instance.

3.3. Not repeating inferences

Alma does not apply all inference rules to all formulas at each step since this is inefficient. It ensures that each rule of inference applied involves at least one newly derived formula so that we do not repeat inferences that have already been done. Doing so reduces the amount of computation that needs to be done. However, this heuristic causes problems with formulas with negative introspection (see below). Negative introspection terms are satisfied if some formula is *not* present in the KB. Therefore removing a formula may make it possible for these formulas to participate in an inference. This is not possible if we strictly follow the strategy above. The solution used is to verify at each step, whether these formulas can be used. This is still effective since the applications we have considered have relatively few instances of negative introspection.

3.4. Language features

Alma has a rich set of features that are crucial for reasoning with mistakes in beliefs. We describe some of them here.

- As mentioned above, time is associated to step numbers. This is available for reasoning though the predicate `now(t)`. There is just one instance of `now(t)` in the KB at any time and the value of `t` that makes this true changes with time.
- Alma can compute arbitrary Prolog programs through the form `eval_bound(P, L)`. `P` is the program to evaluate and `L` is a list of variables that need to be bound before `P` is evaluated. The list allows the user to control when the evaluation is done. This feature allows Alma to be customized easily to various domains that require non-logical computations.
- Alma records the changes in the KB at each step of the computation. This record constitutes the history of the reasoning and can be accessed though reserved predicates. These then enable the logic to reason about its own past reasoning which is useful in case of errors, for instance.
- Alma can introspect in its KB to verify whether a formula is present at that time. The negative introspection can be used as a time-situated approximation for non-provability. Instead of finding "is it the case that ϕ is not provable", Alma computes "is it the case that ϕ has not been derived until now".
- To refer to and assert properties of the formulas in its KB, Alma names all formulas. Names can be assigned by the user too, which makes it possible for the user to assert properties of formulas as part of the description of the domain. The names can be parameterized by variables appearing in the formula which makes it easy to refer to specific instances of universally quantified formulas.
- In addition to the properties of formulas asserted by the user, Alma records various properties of the formulas during execution. These are available to the user for meta-logical computations. Some of the properties that Alma stores or computes when requested by the user are the time at which a formula was first derived and the formulas that were derived from that formula.
- Detection of a direct contradiction is done by the Contradiction Detection rule. This results in the contradictands and their consequences being "distrusted" and a formula $contra(N1, N2, T)$ being added to the KB. $N1$ and $N2$ are the names of the contradictands and T is the time at which the contradiction was found. When a formula is distrusted, it cannot be used in any further inference. However, it can be inspected and reasoned *about*. The $contra(N1, N2, T)$ assertion can be used to trigger reasoning about the contradiction towards its resolution.

  A distrusted formula `P` can be *reinstated* by asserting `reinstate(P)`. Once reinstated, a formula can be used in inferences whereas the formula not reinstated remains unavailable. This facility is useful when the contradiction has been resolved. This resolution is typically specified through axioms provided in describing the domain.

3.5. 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 [25] where Alma/Carne implements the dialog manager in a larger planning system.

Alma directs Carne to do program `P` through `do(P, ID)` where `ID` is an identifier for this execution of `P`. When Carne starts doing the program (which can take a long time to run), `doing(P, ID)` is asserted. Then depending on whether the program succeeds or fails, `doing(P, ID)` is replaced by `done(P, ID)` or `error(P, ID)`. This allows Alma to keep track of and reason about the status of the computation.

Carne can also write arbitrary formulas into the Alma database. This is used for asserting the results of the programs. External input is asserted in Alma in the same way.

3.6. Control issues

Logic typically brings flexibility but at the cost of efficiency. In addition to the restriction to the application of inference rules to new formulas, Alma has the following features to help to mitigate these inefficiencies.

Alma allows the user to limit the number of inferences done at each step. The inferences chosen to be executed are those at the top of a list of possible inferences for that step. The user can specify a sorting procedure to move the more relevant inferences to the beginning of the list. Note that the relevance and the ordering can themselves be computed in the logic.

Another feature of Alma is that there are three kinds of implications instead of the usual one. fif is like the usual implication except that it only asserts its consequent when all the antecedents are satisfied at one step. The result is that fif formulas cannot combine with other implications to give various combinations of implications and that the fif formula cannot be contraposed.

bif also specifies an implication, but one that is only used in backward chaining contexts. These are therefore only used in proofs by contradiction and will not affect the usual forward chaining procedure.

if are the usual implications that can be used both in forward and backward reasoning.

For example, if there are the following formulas:

```
if(and(p, t), s).
fif(and(p, q), r).
bif(p, u)
```

If p is added, we get $if(t, s)$ but not u nor $fif(q, r)$. If we then add q so that both p and q are in the KB, we will get r. u will only be obtained if we start a backward search for it.

4. Illustrations

We now illustrate some simple behaviors of Alma. The initial formulas asserted in the KB will be presented first, followed by the formulas derived at successive steps. The integer to the left of a formula is its name.

4.1. Time situatedness

This example will simply assert a formula at a given time. The input is:

```
if(now(5), lunch).
```

Nothing happens, except for time passage until time 5:

```
0: now(5) ---> lunch
5: now(5)
```

At this point the antecedent of formula 0 is true and `lunch` is asserted at the *next* step, step 6:

```
6: lunch
0: now(5) ---> lunch
7: now(6)
```

4.2. Contradiction detection

If we have `p` and `not(p)`, the contradiction is detected and the formulas are distrusted. The initial KB is:

```
p.
not(p).
```

We get at the next step:

```
4: distrusted(0,1)
6: distrusted(1,1)
3: contra(1,0,1)
1: not(p)
0: p
7: now(2)
```

The arguments of `distrusted` are the formula name and the time at which they were distrusted. The arguments for `contra` are the formula names and the time.

Note that the contradiction with `p` and `not(p)` does not stop inference from going on with other formulas that may be present, and the distrust prevents possibly false formulas from deriving more falsehoods.

5. Non-monotonic reasoning in Alma/Carne

An agent in a dynamic world with incomplete and uncertain information will have to come to conclusions based on

that information. There are several approaches to this problem, one of them being non-monotonic logics. Further, as new information becomes available, the agent should modify its beliefs appropriately, perhaps disbelieving some previously held beliefs. In this section we describe an implementation of a time-situated non-monotonic reasoner in Alma/Carne that solves these problems. This can be a central component of an agent of the type we are interested in.

The features of active logic and Alma/Carne enable us to specify a non-monotonic reasoner in Alma/Carne that jumps to plausible conclusions as soon as possible but then corrects itself if these conclusions are later found to be mistaken. This fast jumping to conclusions has been the intent of non-monotonic logics from the start, but such logics are generally not computable [5], notable exceptions being [6, 13]. Our reasoner has been tested on a suite of non-monotonic examples collated from the literature and performs well overall. In this section, we give an overview of this application of Alma/Carne.

5.1. Non-monotonic reasoning

Reasoning is non-monotonic when, given a set of formulas Π_1, $\Pi_1 \models \phi$ for some formula ϕ but for a superset of Π_1, Π_2, $\Pi_2 \not\models \phi$. This phenomenon is common in commonsense reasoning where getting more information about an object can cause us to change our view of it. The paradigm example is that if we know that Tweety is a bird, we can conclude that Tweety flies. But if we also know that Tweety is a penguin, then we no longer conclude that Tweety flies. Adding the fact that Tweety is a penguin removes the conclusion that Tweety flies. Non-monotonic reasoning seems essential for an agent operating in the world since the information it has tends to be incomplete and uncertain and it can rarely know for sure that a belief it concludes is certain.

There have been several formalizations of non-monotonic reasoning [15, 22, 16]. A major problem with these though, is that they are typically undecidable. While these formalisms might be good at describing the relations in a non-monotonic theory, they are not suitable for computing non-monotonic conclusions as is our aim.

Non-monotonicity can be expressed by defaults which express facts that are typically but not always true, like the fact that birds usually fly. Defaults have exceptions, for example, if the bird is a penguin in which case it does not fly. We represent defaults as formulas of the form $\alpha_i \hookrightarrow \beta_i$ where α_i is called the premise and β_i is the consequent of the default. Such a default will be named δ_i. We say that a default δ_i holds in the KB at some time if at that time α_i and β_i are both in the KB. We also allow preferences between defaults. Consider the defaults "Animals typically don't fly", "Birds typically fly". If we know that Joe is an animal and a bird, we will prefer the second default and say that Joe flies. Preferences are expressed as relations among the names of defaults, for instance $Prefer(\delta_i, \delta_j)$. We assume that the preferences induce a partial order on the defaults. The preferences are expected to be asserted in the KB as part of the domain specification. This seems to be something that is more easily specified for common-sense domains than are probabilities.

The undecidability of non-monotonic logics is typically related to the fact that to be sure that we can apply the "Animals typically don't fly" default to Joe say, we need to know that Joe is not a bird. But non-provability is not decidable and that makes the whole process undecidable.

5.2. Our approach

Our approach is to apply a default if there is no reason not to (a more preferred default that holds or the negation of the consequence being non-defeasibly derived) *at that time*. If there is one, we do not apply the default. If, on the other hand, a more preferred default is later found to apply, we remove the less preferred one and its consequences from the KB which preempts a contradiction. There are two ways then that contradictions can appear. First, contradictions can result when the logic non-defeasibly derives a formula that is the negation of the consequence of a default. The solution then is to withdraw that default and its consequences. Second is the case when a contradiction depends on a set of defaults between which there is no preference (a well known example is Reiter's Nixon diamond problem [23]). In this case, we cannot choose between the contradictands and all of the defaults and their consequences have to be withdrawn. We call a minimal set of such defaults an ID.

Note that as a result, the answer we get from Alma will vary with time in such a non-monotonic system. For instance, at first Alma may conclude that Joe does not fly because he is an animal, then when it finds that he is a bird, it retracts the old conclusion and asserts that Joe does not fly. If Alma then finds that Joe is a penguin, it will change its conclusion once again. This behavior is inevitable if we are not to wait forever for all there is to know about Joe to be made available. Intuitively, the later answers are better than the earlier ones because they are concluded while taking more information into account. So, the quality of the answer is expected to improve with time.

5.3. Rules of inference

The approach described above is implemented as two rules of inference in Alma: one to apply defaults (the DA rule) and one to resolve contradictions (the CR rule). Recall that detection of the direct contradictions and their distrusting is done by Alma itself. Further computation, including

identifying the mistaken beliefs and restoring the true formulas is specified by the CR rule.

5.3.1 DA: Applying a default

From the above discussion, the rule for applying a default is straightforward. Given a default (instance) $\delta_i = \alpha_i \hookrightarrow \beta_i$, if α_i is in the KB, we add β_i to the KB unless either of these conditions hold:

- $\neg\beta_i$ is in the KB and has been derived non-defeasibly. This can be computed by verifying that $\neg\beta$ is in the KB and examining its derivation for the presence of any defaults.
- There is some default δ_j such that $Prefer(\delta_j, \delta_i)$ and δ_j holds in the KB. This is easy to verify too since the logic represents the preference relations explicitly and it is easy to verify whether α_j and β_j are in the KB.
- There is some ID that δ_i is in. This can be done simply by looking up the IDs recorded by the logic.

With this rule, defaults are not added if it is apparent that there is a stronger reason not to do so. The application of a default is clearly computable–it mainly involves KB lookups. The cost for this fast default application or jumping to conclusions is that we may be mistaken. We do not have the assurance that once a default is applied it will hold forever. But this is a cost we are willing to pay since we can handle the mistakes and the alternative–not to apply the default unless we are sure it is correct to do so–condemns us to not doing anything.

However, when a default is applied, we also need to verify whether there are less preferred defaults that hold in the KB but that should not. Taking this approach rather than allowing the contradiction to appear later simplifies the CR rule and reduces inconsistency in the KB. So another computable task for the rule is:

- Given that δ_i has been applied, for each default instance δ_x such that $Prefer(\delta_i, \delta_x)$, if δ_x holds in the KB, it is removed and its consequences undone.

5.3.2 CR: Resolving contradictions

The other rule that we need is the CR rule to respond to contradictions. These indicate mistakes in applying defaults. Once the contradictions are detected, the contradictands and their consequences are distrusted automatically by Alma which results in these formulas not generating any new consequences. The CR rule needs to find the cause of the mistake and repair it. The repair involves reinstating some formulas and deleting others. It also serves to detect IDs.

The first task of the CR rule is to decide which of the contradictands is mistaken. This is a diagnostic task that is compiled into the inference rule. This is possible in this case since there are just a few possibilities given the structure of the reasoning that we have specified. Determining the cause of an inconsistency is not possible in general, however, and even here, the logic might be mistaken.

From the default application rule, we can see that a clash among defaults that have preferences among them will not result in a contradiction because the less preferred default is removed when the more preferred one is added. Therefore the only possibility is an ID: a set of defaults is jointly inconsistent and such that there is no preference among these defaults. The problem then is to decide from all the defaults involved in the inconsistency, which are those that are to be taken to be in the ID. We need to find the minimal set of such defaults. Note that the ID could be a singleton set in which case this is a default the negation of whose consequent is obtained non-defeasibly from the KB. In this case, the resolution is to simply undo the application of the default. It will subsequently not be reapplied.

Computing IDs The defaults involved in the inconsistency are simply the defaults appearing in the derivation of that inconsistency. If there are multiple derivations of some formulas, we might get several sets of defaults. Defaults not appearing in the derivation are not related to the contradiction and are of no concern. However, not all these defaults are taken to be in the IDs. We take the *leaf defaults* only–those defaults in the derivation tree that are closest to te contradiction. The IDs are then asserted in the KB.

The computation of the ID can still be mistaken by being non-minimal. The solution is to try to prove that subsets of the consequents of the IDs we computed are inconsistent using only non-defeasible formulas. If any of these proofs succeeds, the subset is an ID and the previously computed one was mistaken. These proofs are done in parallel with the usual computation of the logic and we do not (and cannot) wait for them to succeed or fail before going on with using the ID. Here too, if the logic is wrong, this will be detected and resolved later.

The CR rule

Therefore, when a contradiction is found, the CR rule does the following:

1. Find the leaf defaults.
2. If there is just one, undo the effects of detecting the contradiction and undo the application of that default.
3. If there are several leaf defaults, undo the effects of the contradiction detection rule and undo the application of the leaf defaults. Further, record the set of distrusted formulas in an ID and start proofs to look for subset IDs. If any of these succeeds, undo the distrust of the

defaults that are in the old ID and take the new one to be the ID.

5.3.3 Results

This non-monotonic reasoner has been implemented in Alma/Carne (in a slightly simplified form) and tested with a large variety of problems in the literature. Out of 93 problems gathered from the non-monotonic reasoning literature, 46 were solved as expected by our system, 23 were similar enough that we are confident that they would also have been solved if we had attempted them and 23 could not be solved.

The problems that could not be solved could not be expressed in the language or were cases where there are no procedures in the language to solve the problems. These include problems where assumptions need to be made, or where one needs to explicitly minimize the set of objects that satisfy a predicate or where diagnostic reasoning is required.

6. Related work

We first consider the RETE algorithm, an efficient pattern matcher that could have been used as a basis of Alma and that we in fact considered using initially. We also consider Executable temporal logics that have a notion of time situatedness that approaches that of active logic.

6.1. RETE

The RETE algorithm [11] and its successor RETE++ [10] are fast pattern matchers that compile productions into tree structures that enable efficient computation of conflict sets (sets of productions that could be applied) in production systems. RETE++ also includes the possibility to do backward chaining.

The RETE family and Alma/Carne are at different points of the expressivity/efficiency scale. Whereas RETE is concerned with very efficient searches for some limited kinds of patterns, Alma/Carne is more concerned with the ability to represent a wide array of information. Some of the more prominent differences are as folows.

Firstly, RETE compiles the rules into tree structures and typically only facts in the KB change. We need to have more flexibility where one can add and delete arbitrary formulas including defaults and RETE rules. The rules in Alma/Carne are mainly logical rules and the rules that RETE compiles into trees are maintained as axioms in Alma/Carne which allows more flexible reasoning with these.

Related to that problem, Alma/Carne maintains a history which does not consist only of facts but of complex formulas too, including formulas that can be seen as RETE productions. It is not clear how that could be done in RETE.

Third, it is not clear how meta-reasoning could be done in RETE. We would need to be able to refer to RETE productions in other productions, and state, maintain and infer properties of these. This is easily done in Alma/Carne.

6.2. Executable temporal logic

Temporal logics [26, 1] are used to reason about changing worlds and are used for specifying and verifying dynamic systems. Executable temporal logics attempt to build models of temporal logic formulas by executing the formulas.

We focus in discrete temporal logics where the formulas can refer to individual steps in the execution, and in particular to the Metatem system [3]. The approach is to see formulas as expressing what the future should be, given the past [12]. The slogan being "Declarative past and imperative future". Any formula can be rewritten in the form *Condition about past* $\rightarrow$ *Condition about the present and future* so that executing a formula requires the interpreter to verify that the antecedent was true in the past and to then constrain the future based on the consequent of the formula. The logic is monotonic in that if previous steps constrain a future state to make some formula true, there cannot be a later state that makes that false. The interpreter may have to make choices in the execution that lead it to the impossibility to satisfy some formulas. In that case, the system backtracks and makes alternate choices.

This approach is close to the active logic point of view in terms of the agent being situated in time. The backtracking of the program in the case of failure cannot happen in active logic–we cannot undo a choice made earlier but have to move on from there. Another difference is that the monotonicity of Metatem contrasts with the non-monotonicity of active logic. These constitute quite basic differences between the two formalisms.

In [4], Metatem is extended to MML and includes meta-reasoning. The domain of the logic in this case includes names of the object level formulas. Variables are divided into two sorts: the object and the meta variables. This logic allows one to use MML for meta-interpreters, and for control of inference for example.

This extension of Metatem allows the logic to refer to its own formulas but it is not clear that there are facilities to reason about the derivation or other properties of these formulas that have proved to be useful in active logic.

7. Conclusion

We have given an account of Alma/Carne, an implementation of active logic that provides representational and in-

ferential capabilities that we believe are useful in specifying agents that reason and act in worlds with incomplete, uncertain and dynamic information. We have also shown an application of Alma/Carne to non-monotonic reasoning which is one of the tasks such an agent will need to perform. Our non-monotonic reasoner jumps to conclusions fast but may later change its mind if new information is available or if it does more computation. The implemented reasoner has been tested on an almost 100 example test suite from the non-monotonic reasoning literature and in 75% of the cases, we get the expected answers.

There are several areas open for future work. We mention some of them here. First, Alma provides facilities for control of the inferenence by reordering the agenda and limiting the number of inferences done in a step. We need to investigate heuristics for doing that.

Second, the non-monotonic reasoning algorithm we devised is not sound or complete in the conventional sense. However, it does seem to be an anytime algorithm. Further, there seem to be special cases where the logic results in the correct answers after some delay. These intuitions have to be formally characterized.

Third, if an agent is to act in the world, we would typically want it to use plans, but executing plans in a logical setting might not be efficient. We would like to have the efficiency of plan execution architectures toether with the flexibility that logic provides, especially for reasoning about and correcting mistakes in the behavior of the agent. One way to do that would be to use our logic to maintain the beliefs in one of the many plan execution architectures, for example PRS [27].

References

[1] J. Allen. An interval-based representation of temporal knowledge. In *Proceedings of the 7th Int'l Joint Conference on Artificial Intelligence*, pages 221–226, 1981.

[2] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5), 1994.

[3] H. Barringer, M. Fisher, D. Gabbay, G. Gough, R. Owens, and M. An. Metatem: An introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[4] H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-reasoning in executable temporal logic". In *KR'91: Principles of Knowledge Representation and Reasoning*, pages 40–49, 1991.

[5] M. Cadoli and M. Schaerf. A survey on complexity results for nonmonotonic logics. *Journal of Logic Programming*, 17(2–4):127–160, 1993.

[6] M. Cadoli and M. Schaerf. Approximate inference in default logic and circumscription. *Fundamentae Informaticae*, 23(1):123–143, 1995.

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

[8] J. Elgot-Drapkin. Step-logic and the three-wise-men problem. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 412–417, 1991.

[9] 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.

[10] T. H. Enterprise. Rete++: Seamless integration of rules and objects using the rete algorithm and c++, 1993.

[11] C. L. Forgy. RETE:a fast algorithm for many pattern / many objectpattern-match problems. *Artificial Intelligence*, 1982.

[12] D. Gabbay. The declarative past and the imperative future: executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Temporal logic in specification*, volume 398 of *Lecture notes in computer science*. Springer-Verlag, 1989.

[13] A. K. Ghose and R. G. Goebel. Anytime default inference. In *Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence*, 1996.

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

[15] J. McCarthy. Circumscription–a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[16] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13(1,2):41–72, 1980.

[17] 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.

[18] 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.

[19] J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, 1988.

[20] 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.

[21] K. Purang. *The Alma/Carne manual*. University of Maryland, College Park, 2001.

[22] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132, 1980.

[23] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276. AAAI, 1981.

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

[25] 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.

[26] J. van Benthem. *The logic of time*. D. Reidel, 1983.

[27] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *JETAI*, pages 197–227, 1995.