A GENERAL FRAMEWORK FOR DISTRIBUTED REASON MAINTENANCE

Jacques Calmet, Joachim Schü, Morio Taneda

University of Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Postfach 6980, D-76128 Karlsruhe, Germany
e-mail: {**calmet,schue,taneda**}**@ira.uka.de**

ABSTRACT

In this paper we will present an efficient logical and computational framework for distributed reason maintenance systems (DRMS), part of our implementation of a CKBS-shell based upon results from distributed and deductive databases. We define a logic programming based foundation onto which paraconsistent, uncertain, temporal and all kinds of distributed reason maintenance capabilities coping with uncertain and temporal informations as well can be added modularly. Compared to previous approaches, we remove limitations on the number of components and the need to define scheme-specific semantics and resolution procedures. Our theoretical issues in this paper are backed by an ongoing implementation exploiting results from distributed databases.

1 INTRODUCTION

With the transition from isolated to cooperating knowledge bases, there is an urgent need for sound computational reasoning with a variety of circumstancial information. Tracking of sources and circumstances as well as revising distributed knowledge bases according to the foundations theory [7] of belief revision requires reason maintenance capabilities. Where it is not possible or desirable to remove inconsistencies, paraconsistent reasoning is needed. A unifying approach has to be defined in a way that is accessible from the logical viewpoint as well as applicable to the concurrent, transaction centered execution model of distributed databases. With such a conceptually complex task, modularity becomes indispensable. We propose that a little algebraic effort goes a long way in understanding and solving the task of providing such a framework.

Technically our approach is based upon generalized annotated logic (GAP) [12] which is briefly introduced in the next section. In the subsequent section we will briefly mention some possible features of GAP useful in Multi-Agent Computing. In section four we first introduce the building blocks of our unit-construction and provide two different algebraic products useful in combining them. We then extend the original notions of two-valued logic reason-maintenance systems (RMS) to multi-valued logic based distributed RMS. The paper is concluded by two examples and a final section which relates our approach to existing work and describes possible future research directions.

2 GENERALIZED ANNOTATED LOGIC

In this section we informally sketch *some* of the interesting features of the logical framework introduced by M. Kifer and coworkers. Besides the capability of providing an efficient approach to paraconsistent reasoning, this logic allows to formalize and to implement a flexible inference machine incorporating temporal and uncertain reasoning methods. For a comprehensive description the reader may refer to [12, 11].
Annotated clause:

$$p : l \leftarrow p_1 : l_1 \wedge \ldots \wedge p_k : l_k \wedge \ \mathbf{not}(B_{k+1} : l_{k+1}) \ldots \wedge \ \mathbf{not}(B_{k+n} : l_{k+1})$$

The literals p, p_i are from a function-symbol free language $\mathcal{L}$. The l_i's are called *annotations* and are constants or variables from a complete lattice $\mathcal{T}$ [1] of truth values.
Semantics: Annotations could be understood as the degree of *belief* in a proposition, e.g. $p : [0.5]$ means that the belief in the truth of A is *at least* $0.5 \in [0, 1]$. Let H be the Herbrand base of the program. An annotated logic interpretation $\mathcal{I}$ is a mapping $\mathcal{I} : H \rightarrow \mathcal{T}$ from the base onto a lattice. An annotated atom $p : l$ is satisfied by $\mathcal{I}$ iff $\mathcal{I}(A) \geq l$ where A is a *strictly ground instance*[2] $p_i : l_i$ of $p : l$. The other logical connectives as $\vee, \wedge$ and $\leftarrow$ are defined in the usual sense.
Fixed-point operator R_P: Let P be a GAP, $\mathcal{I}$ a GAP interpretation and $\mathcal{T}$ a complete lattice. An operator $R_P(\mathcal{I})$ which maps interpretations to interpretations is defined as follows $R_P(\mathcal{I})(p) := \sqcup\{\rho \mid p : \rho \leftarrow p_1 : l_1, \ldots p_n : l_n$ is a strict ground instance of a clause in P and $\mathcal{I} \models p_1 : l_1, \ldots p_n : l_n\}$.

R_P may reach the least fixed point (lfp) if for all strict ground instances A, $lfp(R_P(A))$ is reached after a finite number of iterations. This condition holds for many GAP: if the clause bodies of a program contain only variable- or constant-annotations, or if only finite or decreasing monotone functions[3] appear in the program.
Negation: There are basically two different kinds of negation in GAP, the k-monotonic epistemic (or explicit) negation $\neg$ and the non-monotonic Closed World Assumption based **not**. $\neg$ requires the symmetry between **true** and **false** provided by bilattices [8], for example $\neg A : \mathbf{t} = A : \mathbf{f}$. For a proposal on how to handle **not** by adapting the stable and well-founded semantics to GAP the reader may refer to [15].

3 GAP IN MULTI-AGENT COMPUTING

In a situation where several knowledge base systems cooperate more or less tightly, and have a varying portion of their knowledge in common, two approaches may be used to map them into an integrated view (a single KB which is equivalent to the group of KBs,

[1] A complete lattice $(\mathcal{T}, \preceq)$ is a partial ordering with a least upper bound $\sqcup$ and a greatest lower bound $\sqcap$ for every subset of $\mathcal{T}$.

[2] A strictly ground instance is a ground instance as in classical first-order logic and furthermore all annotation variables and functions are evaluated to constants of the lattice.

[3] A function f is finite if $\{f(x) | x \in DOM(f)\}$ is finite and f is decreasing if for arbitrary arguments $x_1, \ldots x_n$ $f(x_1, \ldots x_n) \leq x_i$ for all $1 \leq i \leq n$.

at least model-theoretically). One possibility is to annotate the predicates with sets of KB ids stemming from the lattice $(\mathcal{P}(\{Ag_1, \ldots, Ag_n\}), \subseteq)$, the powerset of KB ids. In this case the disagreement of KBs increases notational complexity. Rules common to multiple KBs require the least effort. Connections between same-named predicates are implied by the structure of subset lattice. Or one could simply rename the predicates such that the alphabets of the different agents are disjoint. Any identities between predicates in different KB must then be reestablished explicitly. Where KBs agree, rules, inference steps and results may be duplicated. Rules for mediation may be written as clauses whose bodies consist of multiple instances of a predicate. In the rest of this section we provide some examples of how GAP can be used in the context of distributed knowledge bases:

Paraconsistent reasoning. Definite Horn-clauses used in ordinary logic programming languages lack the capability to express logical inconsistencies, inherent to collaborative environment. Using bilattices like FOUR [11] we are able to explicitly represent clauses which state how to deal with conflicting knowledge, e.g.

$$buy\_stock(IBM) : [\mathbf{f}] \leftarrow buy\_stock\_opinions(IBM) : [\top]$$

if there is any inconsistency concerning the idea of buying stocks from IBM then we should better not buy it.

Majority decision-making. For example expressing a majority of two in a group of three requires three clauses:

$$\begin{aligned} p_{maj} : [v] &\leftarrow p_1 : [v], p_2 : [v] \\ p_{maj} : [v] &\leftarrow p_1 : [v], p_3 : [v] \\ p_{maj} : [v] &\leftarrow p_2 : [v], p_3 : [v] \end{aligned}$$

Which of the former two transformations is advantageous obviously depends on the similarity and coherence of KB's. Also, the rules established for the mediation of disagreements may be more easily expressed in one or in an other form.

Preference of agents. The variety of reasoning modes beyond local or global consistency applicable to mediation suggests that it may be better not to build a single mode of mediation into the RMS, but to provide the means to write clauses for mediation. In the following clause we are able to express a preference of agent one for a single proposition:

$$\begin{aligned} buy\_Stock : [v, \text{Agent3}] \leftarrow \quad & buy\_Stock : [w, \text{Agent2}] \\ \wedge \quad & buy\_Stock : [v, \text{Agent1}] \end{aligned}$$

This kind of reasoning is particularly useful for result recomposition as in the contract-net.

Reasoning with temporal data. In [12] it was shown how some kinds of temporal reasoning could be subsumed by GAP, e.g. clauses of the form

$$\text{Buy\_HK\_Dollar} : succ(T) \leftarrow \text{US\_Dollar\_low} : T$$

where $succ()$ denotes a function which increases every member of the set T, e.g. $succ(\{1,2,3\}) = \{2,3,4\}$. There are furthermore some interval-based temporal reasoning formalisms which could be expressed in GAP [12].

4 DISTRIBUTED REASON MAINTENANCE

An alternative approach to paraconsistent knowledge is to compute maximal consistent subsets by a reason maintenance system. The key idea in redefining ATMS as GAPs is to write labels as a form of annotation, i.e. defining a suitable lattice of labels. Our system departs therefore from most other RMS since it amalgamates inference machine and the RMS. On one hand, it appears tempting to have a separation between the reason maintenance system and the inference machine leading to a reusable RMS system and a gain in concurrence between the RMS and the inference machine. On the other hand, following the argumentation of [17] a RMS *itself* should be able to detect inconsistencies and to compute automatically the dependencies of new beliefs on older ones, instead of just passively recording them. Besides, the amount of time spent for recording the assumptions is reduced since this is done without any extra costs during the inference process. In our implementation we are not recording the annotated literal in the labels, but just a pointer to the according clauses which allows a more compact representation as in systems where the RMS component is separated from the inference machine. For these reasons we decided to rely on the above sketched framework to formalize the operational semantics of the reason maintenance process. We believe that the building block nature of our framework makes up well for the loss of overall modularity. In fact, one might say that in our proposal the former RMS component has become just one of the factors of the annotation lattice.

In some application domains one would like to reason paraconsistently and furthermore to record the assumptions leading to inconsistency, without propagating such a Nogood through all labels, causing high complexity. In the example at the end of this section we show how a third agent may draw conclusions from mutually contradicting facts from two other agents and record these facts in his own label as ordinary assumptions.

4.1 Basic Components

As mentioned above the truth values a GAP interpretation assigns to a proposition stem from a complete lattice. In the context of distributed reason maintenance the following lattices appear to be useful:

The lattice $([0,1],\leq)$ with the max function as lub represents degrees of certainty.

A sufficiently well-behaved family of subsets of $\mathbf{R}^+$, e.g. all sets made up of points and closed and open intervals, or finite sets of time points, with the usual subset ordering and the set-union $\cup$ as least upper bound is used to represent temporal extents. Actual temporal reasoning, such as taking into account possible and impossible sequences of events, is outside the interest of this report, and presumably requires extensions to the

GAP.

4.2 Products

Next, we introduce two kinds of products of lattices which we use to describe composite annotation lattices.

First, the usual cross-product of lattices with componentwise ordering, least upper bound and greatest lower bound. Let $(\mathcal{T}_1, \preceq_1, \sqcap_1, \sqcup_1) \ldots (\mathcal{T}_n, \preceq_n, \sqcap_n, \sqcup_n)$ be lattices and $(x_1 \ldots x_n), (y_1 \ldots y_n) \in \mathcal{T}_1 \times \cdots \times \mathcal{T}_n$.

- $(x_1, \ldots, x_n) \preceq (y_1, \ldots, y_n) \quad \Leftrightarrow \quad \forall i \in \{1, \ldots, n\} \quad x_i \preceq_i y_i$
- $(z_1, \ldots, z_n) = (x_1, \ldots, x_n) \sqcap (y_1, \ldots, y_n) \quad \Leftrightarrow \quad \forall i \in \{1, \ldots, n\}\, z_i = x_i \sqcap_i y_i$
- $(z_1, \ldots, z_n) = (x_1, \ldots, x_n) \sqcup (y_1, \ldots, y_n) \quad \Leftrightarrow \quad \forall i \in \{1, \ldots, n\}\, z_i = x_i \sqcup_i y_i$

For example $[0, 1] \times [0, 1]$ is of use in uncertain reasoning where the first component refers to the certainty that a proposition is true and the second to its falsehood. Consider as another example the lattice FOUR, which may be written as the cross-product of $\{\bot, \mathbf{t}\}$ and $\{\bot, \mathbf{f}\}$.

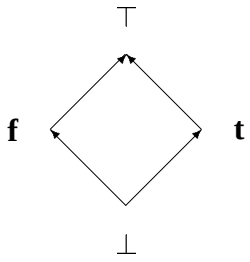

Figure 1: The lattice FOUR

Second, the *free product*, written $\otimes$, which is similar to the cross product except that the lub is not defined pointwise but as a formal operation with the rules

$$\begin{aligned} (a \sqcup b) \otimes c &= (a \otimes c) \sqcup (b \otimes c) \\ a \otimes (b \sqcup c) &= (a \otimes b) \sqcup (a \otimes c) \end{aligned}$$

and additional rules which express commutativity and associativity of $\sqcup$ and distributivity of $\sqcup$ with $\sqcap$.

Definition 4.1 *An expression of the form $a \otimes b$ is called a* term. *An arbitrary element of $A \otimes B$ is the lub of a finite number of terms, i.e. $x = \bigsqcup_{i=1}^{n} a_i \otimes b_i$. Terms with $\bot$ as one of the factors may be omitted because $\bot \otimes x = x \otimes \bot = \bot_{A \otimes B}$.*

The set of time-uncertainty annotations [5] is a typical example. It can be written as $[0, 1] \otimes 2^{\mathbf{R}^+}$.

Possibility of truth and falsehood	$N = [0,1] \times [0,1]$
Time and uncertainy	$T \otimes N$
Multiple agents and uncertainty	$\mathcal{P}(1 \ldots n) \otimes N$
Time, multiple agents and uncertainty	$T \otimes \mathcal{P}(1 \ldots n) \otimes N$

Table 1: Selected examples of product lattices

4.3 Mixed variable and constant annotations

In product annotation lattices a specific form of "term annotation" as defined in [12] is particularly simple and flexible. These are annotations by terms – as defined here – whose factors are either variables or constants.

Definition 4.2 (m-annotations) *An m-annotated literal is a literal* $p : (a_1 \otimes a_2 \otimes \cdots \otimes a_n)$ *where each* a_i *is either a variable or a constant of the appropriate lattice. We shall use the more conventional notation* $p : [a_1, \ldots, a_n]$ *for* $p : (a_1 \otimes \cdots \otimes a_n)$.

In clauses, variables may not appear in different positions, nor be shared between annotations and arguments of predicates. Like in general GAPs, a variable that occurs in the head of the clause is expected to appear in the same position in at least one of the body literals. Since the individual lattices that make up a product are usually distinct from each other and from the Herbrand universe, this is not a severe restriction. At the same time, it is of major help in reducing complexity, especially compared to the monotonic, computable functions permitted by [12].

4.3.1 Annotated clauses

The following two clauses are interpreted as being equivalent:

$$\begin{aligned} p &\leftarrow q_1, q_2 \ldots \\ p : (\alpha_1 \sqcap \alpha_2 \sqcap \ldots) &\leftarrow q_1 : \alpha_1, q_2 : \alpha_2 \ldots \end{aligned}$$

This means, that if one simply states that p whenever q_1, q_2 etc., then p holds under the meet of all circumstances under which the body literals hold. If annotations are times, then p holds only at those times at which all q_i hold, if they are fuzzy values, p is guaranteed only to the lowest certainty found in the q_i, and if they are environments, p is justified by the meet of the justifications of the q_i, which is, by reversal of order, the union of the assumptions needed for the q_i.

A constant or variable in an annotation overrides this implicit behavior.

Definition 4.3 (partially m-annotated clauses) *A clause*

$$p : [a, b, \ldots] \leftarrow q_1 : [a_1, b_1, \ldots], q_2 : [a_2, b_2, \ldots], \ldots$$

where $a, a_i, b, b_i \ldots$ *are constants, variables or empty, is read as follows:*

(i) Replace each empty a or a_i with the same new variable a_0, each empty b or b_i with the same b_0 etc.

(ii) Each body literal is processed as a subquery, returning, if successful, with a substitution for each of the variables in the annotation.

(iii) For every variable occurring in annotations in the body, form the meet of all values substituted for it by the subqueries.

(iv) Substitute this value for any variable in the head annotation. Fail if any of the variables has the value $\perp$.

This description, which may be used almost directly in a SLD- or OLDT-type procedure, is equivalent to the usual definition which refers to strictly ground instances of clauses.

4.4 Assumptions

The basic idea is to code the assumptions under which a proposition holds into its annotations. This idea was originally used in Ginsberg's MVL theorem proving system [8].

To illustrate how assumption based reason maintenance is in principal in being processed let P be the GAP database below and consider the following example:

$$\begin{array}{rcl} A:[A] & \leftarrow & \\ D:[D] & \leftarrow & \\ B:V & \leftarrow & A:V \\ C:V & \leftarrow & B:V \\ D:V & \leftarrow & C:V \end{array}$$

According to the GAP fixed-point operator R_P in section 2, the following computations are then performed:

$$\begin{array}{rcl} R_P \uparrow 0(\emptyset) & = & \{A:[A], D:[D]\} \\ R_P \uparrow 1(\emptyset) & = & R_P \uparrow 0(\emptyset) \cup \{B:[A]\} \\ R_P \uparrow 2(\emptyset) & = & R_P \uparrow 1(\emptyset) \cup \{C:[A]\} \\ R_P \uparrow 3(\emptyset) & = & \{A:[A], D:[\{D\},\{A\}], B:[A], C:[A]\} \\ R_P \uparrow 4(\emptyset) & = & R_P \uparrow 3(\emptyset) = lfp(R_P(\emptyset)) \end{array}$$

At this point we should note that departing from most other ATMS our system is being able to perform reason-maintenance within a backward-chaining inference or a mixed bottom-up and top-down procedure like OLDT or Magic Sets as well. In our CKBS implementation the query proof procedure is based upon an adaption of OLDT-resolution [14] which is basically a top-down search with respect to a certain query

where already computed subqueries are cached and bottom-up applied to a repeatedly appearing subquery. The following proof tree shows how assumptions are being handled in a pure backward-chaining procedure:

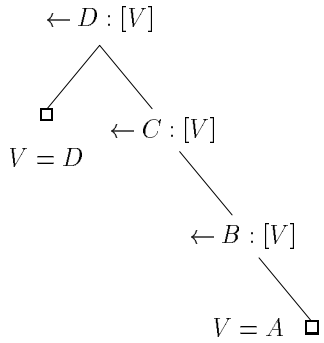

Figure 2: Proof path

4.4.1 Multivalued Assumptions

In an annotated logic database, it is not sufficient to record dependency based only on predicates. One proof may rely on a predicate being true at one time, while another proof may rely on the same predicate being false at another. Therefore we record *annotated literals* in labels. This enables in particular the representation of negative assumptions using the explicit negation $\neg$.

In forward chaining inference processes, the literals in nodes contain no variables. If an unbound and thus implicitly universally quantified variable occurs in an annotation, the value $\top$ is substituted. Even with backward chaining, labels are only collected when returning from subqueries and thus contain no variables.

We now define a lattice which, when used in annotations in the way defined in section 4.3.1, implements the label calculations of an annotated ATMS.

Definition 4.4 (Annotated Environments) *Let L be a (composite) annotation lattice not including assumptions and $H_A = \{p_1 \ldots p_m\}$ the finite subset of the Herbrand base which appears in assumptions, and H_{AG} the set of ground elements of H_A. We define the lower semilattice Env = Env(H_A, L) of annotated environments as:*

- *Env* $= \{e : H_A \to L\}$
- *An annotated environment e will be written $(p_{i_1} : l_1, \ldots p_{i_n} : l_n)$ where $l_j = e(p_{i_j}) \neq \bot$, omitting any $p \in H_A$ for which $e(p) = \bot$.*
- *The order on Env is the reverse* **pointwise** *ordering*

$$\textit{Let } d, e \in \textit{Env, then } d \preceq e \quad \textit{iff} \quad d(p) \geq e(p) \quad \forall p \in H_{AG}$$

Likewise, the meet on Env is derived pointwise from the lub on L.

- $\top_{Env} = ( )$
- $(\bot_{Env})(p) = \top_L \quad \forall p \in H_{AG}$

Thus the meet of two sets of assumptions A_1, A_2 is defined as the *minimal* set of assumptions which implies both A_1 and A_2. Hence the ordering of sets of assumptions is now based upon the GAP implication and generalizes the usual $\subseteq$ ordering of powersets of assumptions in classical two-valued logic. In other words, if $L = \{\mathbf{t}, \mathbf{f}\}$, then environments are sets of predicates and $\sqcap$ is simply the set union.

Definition 4.5 (Annotated Labels) *The set of annotated labels, Lab(H_A, L), H_A, L as above, is defined as:*

$$\mathit{Lab}(H_A, L) = \{\{e_1, \ldots, e_n\} \mid e_i \in \mathit{Env}(H_A, L), \forall i, j = 1 \ldots n : e_i \not\preceq e_j\}\}$$

Write Lab $= \mathit{Lab}(H_A, L)$ for short, let $D = \{d_1, \ldots, d_n\}$, $E = \{e_1, \ldots, e_m\}$ $E, D \in$ Lab and let max(X) be the set of maximal elements of X. Then define:

- $\bot_{Lab} = \{\bot_{Env}\}$
- $\top_{Lab} = \{( )\}$
- $D \sqcap E = \mathit{max}(\{d_i \sqcap e_j \mid d_i \in D, e_j \in E\})$
- $D \sqcup E = \mathit{max}(D \cup E)$

With these definitions, Lab is a lattice.

Let now P be a GAP program consisting of literals $p : [l_1, \ldots, l_n]$ and

$$\pi_i(p : [l_1, \ldots, l_n]) = p : [l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n], \quad \text{and} \quad \Pi_i(P) = \{\pi_i(c) \mid c \in P\}$$

Assume that the label occupies the n-th, i.e. last position. That ATMS labels consist of sets of minimal environments may be expressed in the following integrity constraints:

Lemma 4.1

- $\forall l \in Lab(H, L), e \in l : P \vdash_{GAP} p : [l_1, \ldots, l_{n-1}, l] \Rightarrow \Pi_n(P) \cup \{e\} \vdash_{GAP} p : [l_1, \ldots, l_{n-1}])$
- $\forall l \in \mathit{Lab}(H, L), d \in \mathit{Env} : \Pi_n(P) \cup \{d\} \vdash_{GAP} p : [l_1, \ldots, l_{n-1}] \Rightarrow \exists e \in l : e \geq d$
- $\forall \alpha, \beta \in (l \in Lab(H, L)) : \alpha \vdash_{GAP} \beta \Rightarrow \alpha = \beta$

which state, that every environment in a label is sufficient to prove p, that every environment is maximal in that respect, and that within a label no environment subsumes another.

Following the conventions introduced in section 4.3.1, if the label lattice is part of the annotation but never explicitly specified in clauses, the meet of the labels in the body of a clause instance is assigned to the node resulting from the head of that clause. Thus dependency tracking is performed without one's having to rewrite existing clauses.

Lemma 4.2 *If P is a GAP with the base H and the annotation lattice L which does not include labels, then an interpretation of the same set of clauses with the annotation lattice $L \otimes Lab(H, L)$ contains in the last component of its annotations the ATMS labels.*

In order not to jeopardize the integrity of labels it makes sense to completely forbid explicit specification of labels in clause heads.

4.4.2 Facts and assumptions

Traditionally, it is the responsibility of the RMS component to distinguish facts and assumptions when initializing nodes. An assumption would result from a query for a predicate which doesn't appear anywhere in a clause head, while a fact would result from a bodiless clause in the KB. In annotated logic, it may be desirable to mark some extent of a predicate as fact and another as assumption. We also want to integrate the RMS and inference components as far as possible, thus the KB is the preferable place to specify assumptions as well as facts. An ATMS will initialize nodes for assumptions with labels referring only to themselves, i.e. $A : [A]$. In a GAP with labels, such a node would also result from a bodiless clause with a head of this form, which is normally not allowed. However since some form of initialization is necessary and writing initial nodes syntactically as facts integrates them nicely, we allow this form of explicit labeling of literals.

Definition 4.6 *In a GAP with labels, with the annotation lattice $L \otimes Lab(H, L)$, define:*

(i) A fact is a clause $a : [\alpha]$, $\alpha \in L$. It has the empty label, i.e. it holds without any further preconditions.

(ii) An (initial) assumption is a clause $a : [\alpha, \{(a : [\alpha])\}]$, $\alpha \in L$, or $a : [\alpha, \sim]$ for short.

(iii) No other bodiless clauses are allowed.

Example. Let $L = ([0, 1] \times [0, 1]) \otimes T$. To specify that p is true at time 0 and may be assumed anything later, write:

$$\begin{aligned} p : [(1, 0), \{0\}] &\leftarrow . \\ p : [\top, (0, \infty), \sim] &\leftarrow . \end{aligned}$$

We now have the necessary elements to describe various constructs in a common framework. Some examples of annotations incorporating justifications are given in table 2, where: $B = \{\bot, \mathbf{t}\}$, $T =$ time, $P = [0, 1] \times [0, 1]$, $A =$ agents, $H =$ ground literals.

4.5 Nogood computation and detection

Since assumptions themselves are annotated, the original definition that a Nogood is a minimal set of assumption inconsistent with the knowledge base has to be extended.

'Classic' ATMS [8]	FOUR $\otimes$ *Lab*(H, B)
A distributed ATMS	FOUR $\otimes A \otimes$ *Lab*$(H, B \otimes A)$
A temporal-possibilistic distributed ATMS	$T \otimes P \otimes A \otimes$ *Lab*$(H, B \otimes A)$
A temporal-possibilistic distributed ATMS with temporal uncertain justifications	$T \otimes P \otimes A \otimes$ *Lab*$(H, T \otimes P \otimes A)$

Table 2: Composite annotation lattices

In our case, inconsistency depends on the chosen lattices. The same (bi)lattice structure with different inconsistent sets may express very different views of what lattice elements mean. For example, the bilattice $\mathcal{L} = [0,1] \times [0,1]$ where the first component indicates some degree of truth and the second falsehood, with the inconsistent set $\mathit{Inc}_1(\mathcal{L}) = \{(x,y) \mid x > 0 \wedge y > 0\}$ is used in possibilistic logic [4] while the same bilattice with $\mathit{Inc}_2(\mathcal{L}) = \{(x,y) \mid x + y > 1\}$ is closer to probabilistic logic.

Definition 4.7 *Given lattices $L = L_1 \otimes \ldots \otimes L_n$ with inconsistent sets Inc(L_i), a term $x_1 \otimes \cdots \otimes x_n \in L$ is inconsistent iff for at least one i, $x_i \in$ Inc(L_i).*

An element $x = \bigsqcup_{j=1}^{k} x_j \in L$, with $x_j = x_{j1} \otimes \cdots \otimes x_{jn}$, is inconsistent iff there is an inconsistent term $t \leq x$.

Such an inconsistent term is not necessarily one of the x_j. Detection of inconsistencies thus involves finding *maximal* terms $t \leq x$ and checking them for inconsistency. The latter is simple since in most cases only one of the lattices used in product annotations has a non-empty *Inc*; time, agents etc. are always consistent. The integrity constraints stated above are extended to account for the consistency checks.

Lemma 4.3 *Let l be a Nogood, then the following constraints must hold:*

- $\exists p \in H : \Pi_n(P) \cup \{l\} \vdash_{GAP} \pi_i(p : [\ldots, l_j, \ldots]), l_j \in$ *Inc*(L_j)
- $\forall \alpha, \beta \in l_i : \alpha \vdash \beta \Rightarrow \alpha = \beta$
- $\Pi_n(P) \cup \{l_i'\} \vdash_{GAP} \Rightarrow l_i' \subseteq l_i$

Removing just-discovered nogoods from the labels of all nodes has a number of drawbacks. First and well known is its complexity. Furthermore, we expect that with removed labels it will be more expensive to add and remove clauses from the knowledge base; also removing an environment may not be as easy with arbitrary lattices as it is with sets of predicates.

The problem of finding maximal terms is of great importance to all lattice computations and comparisons.

Since we want to allow paraconsistent reasoning to coexist with nogoods within our framework, an arbitrary mechanism for excluding some predicates from all inconsistency checks is needed and presumed to be in place.

4.6 Hypothetical reasoning

Consider an annotated literal which specifies a label in the body of a clause ($\alpha \in L$, $\mathcal{A} \in \mathit{Lab}(H, L)$):

$$q : [\beta] \leftarrow p : [\alpha, \mathcal{A}]$$

The literal on the right, when interpreted as a subquery according to the GAP rules, means "can $p : [\alpha]$ be proven from the program and the assumptions in $\mathcal{A}$?" Such a statement is separated from the TMS process, since its validity no longer depends on other assumptions, but only on the clauses used in the proof. It is therefore correct to assign an empty label to the node $q : [\beta]$.

The conventions established so far allow this kind of reasoning to coexist with ATMS type dependency tracking, without violating integrity constraints.

Note that this is only an elementary form of hypothetical reasoning, in that there is no connection between the hypothetical and the believed environments.

4.7 Database issues

So far we have identified the necessary ingredients to formalize different kinds of distributed assumption-based reason maintenance systems. In the case of Multi-Agent computing we tagged each proposition with an index of its origin. In our implementation every piece of communicated data will be replicated. The copies are called *physical data items*, the original is named *logical data item* [21]. The location of the logical data items (propositions) will correspond to the index of it. For example the logical data items corresponding to the ground instances of a predicate $P(X) : [Ag_1, l]$ will always be in Agent one. Mutual consistency will be maintained by the *read-once/write all* protocol which we realize by a distributed two phase primary locking (D2PL) algorithm. We deem this protocol as sufficient since updates will only occur at the location of the logical data items. We note that this implies a directed communication, since only the site possessing the logical data item can change it. After such a fact got updated we use the afore mentioned algorithm for incremental maintenance of mediated views which is capable of handling recursive and non-recursive clauses as well.

It is furthermore important to have an appropriate locking granularity, as only ground instances of a fact are being locked in simultaneous read/write access on shared or even local data. This should guarantee a higher level of concurrence among the different RMS. Besides, due to the above mentioned amalgamation of RMS and inference machine, the locking granularity within a local RMS is of crucial importance.

5 POSSIBLE APPLICATIONS

There is broad spectrum of conceivable applications as abductive reasoning and knowledge assimilation [13], to mention a few, to which our framework applies in the distributed case as well. In the area of planning, temporal *probabilistic* logics [6] have gained popularity and we plan in the future to assess the usefulness of our framework

in this area. The examples in this section are intended to illustrate the effectiveness of multi-valued assumptions for certain application domains as robots and medical expert systems. For the purpose of readability we did not include the justification lattice in our sample knowledge bases.

5.1 Paraconsistent distributed reason maintenance

In this subsection we provide a conceivable scenario, where it appears useful not to remove inconsistent information from the knowledge bases like in traditional distributed RMS, as [10]. Suppose we have two different doctors and one patient consulting both of them.

$\boxed{KB_1:}$ $\quad Patient\_Disease(TBC) : [\{Ag_1\}, \mathbf{t}] \leftarrow$

$\boxed{KB_2:}$ $\quad Patient\_Disease(TBC) : [\{Ag_2\}, \mathbf{f}] \leftarrow$

$\boxed{KB_3:}$

$$\begin{aligned} Find\_Specialist(X) : [\{Ag_3\}, \mathbf{t}] &\leftarrow Patient\_Disease(X) : [\{Ag_1, Ag_2\}, \top] \\ Visit\_Specialist(X,Y) : [\{Ag_3\}, \mathbf{t}] &\leftarrow Find\_Specialist(X) : [\{Ag_3\}, \mathbf{t}], \\ &\quad\ still\_hurts(Y) : [\{Ag_3\}, \mathbf{t}] \\ still\_hurts(Lung) &\leftarrow \end{aligned}$$

Suppose agent 3 queries for $\leftarrow Visit\_Specialist(X,Y) : [\{Ag_3\}, \mathbf{t}]$. According to the proof procedure of GAP he receives the answer

$$\begin{aligned} Visit\_Specialist(TBC, Lung) \ : \ & [\{Ag_3\}, \mathbf{t}, \\ & \{(Patient\_Disease(TBC) : [\{Ag_1\}, \mathbf{t}], \\ & Patient\_Disease(TBC) : [\{Ag_2\}, \mathbf{f}], \\ & still\_hurts(Lung) : [\{Ag_3\}, \mathbf{t}])\}] \end{aligned}$$

which states that he should visit a lung specialist under the *assumptions* that the two family doctors he usually visits *disagree* and his lung still hurts. Suppose now Doctor (Agent) 1 changes his opinion and updates his database to $Patients\_Disease(TBC) : [\{Ag_1\}, \mathbf{f}] \leftarrow$. In this case the replica of this fact in Agent **3** becomes invalid and gets updated according to the update contract which has been established during the inference with Agent 1. Although this a very simple example it shows the usefulness of our incremental view maintenance algorithm [20] for maintaining the knowledge-bases of Agent 1 and 3 after an update.

5.2 Temporal possibilistic reason maintenance

To illustrate how temporal and possibilistic information can be handled within our reason maintenance system consider the following scenario consisting of three mobile robots. The annotations consist of origin, certainty and time resp.:

$\boxed{KB_1}$:

$$\begin{aligned} Robot\_at(X,Y):[\{Ag_1\},\sqcap(V_1,V_2),T+2] &\leftarrow Robot\_at(X,Y):[\{Ag_2\},V_1,T], \\ &\quad Part\_at(X,Y):[\{Ag_3\},V_2,T] \\ Call\_supervisor(X,Y):[\{Ag_1\},1,V] &\leftarrow Robot\_at(X,Y):[\{Ag_1\},0.2,V], \\ &\quad Robot\_at(X,Y):[\{Ag_2\},0.2,V] \end{aligned}$$

$\boxed{KB_2}$: $\quad Robot\_at(1,1):[\{Ag_2\},1,\{2,3\}] \leftarrow$

$\boxed{KB_3}$:

$$\begin{aligned} Robot\_at(1,1):[\{Ag_3\},1,\{2,3\}] &\leftarrow \\ Part\_at(X,Y):[\{Ag_3\},0.9,V] &\leftarrow Robot\_at(X,Y):[\{Ag_3\},1,V] \end{aligned}$$

Suppose Robot 1 queries now for $\leftarrow Robot\_at(X,Y):[\{Ag_1\},V,W]$, he then receives the answer

$$\begin{aligned} \leftarrow Robot\_at(1,1):[\{Ag_1\},0.9,\{4,5\}, \quad \{ \quad & Robot\_at(1,1):[\{Ag_2\},1,\{2,3\}], \\ & Robot\_at(1,1):[\{Ag_3\},1,\{3,4\}] \ \} \ ] \end{aligned}$$

which states that under the *assumptions* that robot 2 is at time points $2,3$ and robot 3 is at time points $3,4$ at location $(1,1)$, Robot 1 will be there with a certainty of at least 0.9 at time points $4,5$. Note that robot 1 is also able to perform simple kinds of hypothetical reasoning, as "if robot 2 were with a certainty of at least 0.75 at time point 3 at location $(1,1)$ at what location with which certainty should I be" by adding the fact $Robot\_at(1,1):[\{Ag_2\},0.75,\{3\}] \leftarrow$ to its *own* local knowledge base, independently if this fact really holds in robot 2. In this case all changes of the real position of robot 2 are not being propagated to robot 1 since $Robot\_at(1,1):[\{Ag_2\},0.75,\{3\}] \leftarrow$ is not a replicated fact and therefore no update contract between robot 1 and robot 2 has been established.

6 CONCLUSION AND OUTLOOK

In this paper we have described several distributed assumption-based RMS within a single framework and showed that distributed reason maintenance has a well-understood *semantics*. Using two kinds of lattice products and well known lattices for time, uncertainty and assumptions we can handle distributed belief revision within the context of distributed deductive databases. Though it is based upon the ideas of [8] our work departs in several points from theirs. We use a different inference machine, annotated logics is more expressive which has been shown in [12]. To summarize, bilattice-based logic programming could be subsumed be GAP. [8] does not deal with distributed knowledge bases and uncertain and temporal assumptions, the free product has not

been addressed. More similar to our ideas is the work of [2], though it does not deal with time, uncertainty or distributed knowledge bases.

There are several other distinctions to former approaches of DRMS. As stated previously there is a distinction between distributed RMS and cooperating RMS. There is a true distributed realization of an ATMS downwards compatible to the original ATMS in [1] which has recently been extended to include simple temporal informations. But there is neither a fixpoint semantics for distributed reason maintenance provided nor there are temporal *and* uncertain assumptions expressible.

The first approach to distributed reason maintenance [10] was based upon Doyle's justification based truth maintenance system. We emphasize that our framework is also capable of simulating distributed *justification-based* truth maintenance if we do not propagate the base assumptions through the clauses of our logic program but all antecedents of a derived fact. The annotation will then represent the footprint of a proof. The different levels of well-foundedness and (in)consistency as defined in [10, 1] can therefore be captured within our framework as well. According to our experiences one should not rely on a single method in dealing with inconsistent knowledge, but use several different strategies. Due to the high complexity of assumption-based RMS [19] only in some small parts of a CKBS where a set of solutions is absolutely required, a distributed assumption based RMS might be useful. In other parts the higher efficiency of a justification based RMS could be of greater use. Paraconsistent reasoning on the other hand has its place too, since computing maximal consistent subsets of knowledge-bases removes useful information. In some cases one would prefer to draw conclusions from inconsistent knowledge rather than throwing it away.

While GAP is capable of first-order logic, our DRMS is currently based on a function-symbol free, Datalog-like language. We have to investigate appropriate generalizations along the lines of Ginsberg's first-order ATMS [8].

It has been shown in [3] that GAP could be viewed as certain instance of constraint logic programming (CLP) which allows to express certain lattice-based constraints in a more compact and natural manner than in CLP. From viewpoint of generality our approach appears similar to that of [18] whose system allowed arbitrary boolean constraints on propositions. We plan in the future to permit *non-ground* annotated assumptions since there is a non-ground fixpoint semantics for CLP [9] which could probably forwarded to GAP.

For practical applications, the expressivity needs to be enhanced to allow more forms of hypothetical reasoning. There are some indications that this is only a special case of a more differentiated query capability which is already implicitly in use at some points. We will also address the practical need for some monotonic functions in annotations.

We furthermore plan to develop strategies to minimize the costs for communication and to develop some networking strategies for re-routing informations through other agents. In our current system data could only be received from the original source. In some cases it might be useful to ask a third agent for data which he has already received, e.g. if the agent is closer. In the case of link-failures voting based protocols might be

of interest. Even if an agent is currently not available, if a majority of agents agrees on certain propositions from this agent, all conclusions based upon these facts may be still considered as valid. Besides, investigations in possible distributed reliability protocols such as the three-phase commit protocol [21] have to be carried on.

REFERENCES

[1] C. Beckstein, R. Fuhge and G. Kraetzschmar. Supporting Assumption-Based Reasoning in a Distributed Environment. *Draft, Bavarian Research Center for Knowledge Based Systems, 1993*

[2] M. Cayrol, O. Palmade, T. Schiex. A Fixed point Semantics for the ATMS. *Journal of Logic Computation, Vol.3, No.2, pp.115-130, 1993*

[3] D. Debertin. Parallel inference algorithms for distributed knowledge bases (in german). *Project work, Institute for Algorithms and Cognitve Systems, University of Karlsruhe, 1994*

[4] D. Dubois, H. Prade and J. Lang. A possibilistic ATMS. *Report IRIT/90-54/R, December 1990, Université Paul Sabatier*

[5] D. Dubois, H. Prade and J. Lang. Timed possibilistic logic. *Fundamentae Informaticae, vol. XV, No. 3-4, 1991, pp. 211-234*

[6] T. Dean, M.P. Wellman. Planning and Control. *Morgan-Kaufman, San Mateo, 1991*

[7] P. Gärdenfors (Editor) Belief Revision. *Cambridge Tracts in Theoretical Computer Science, 1993*

[8] M. L. Ginsberg. Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence. *Computational Intelligence, 1990*

[9] M. Gabrielli, G. Levi. Modelling answer constraints in Constraint Logic Programs. *ICLP'91, pp.238-251*

[10] M. Huhns, D. Bridgeland. Distributed Truth Maintenance. *IEEE Transactions on Systems, Man and Cybernetics, 1990*

[11] M. Kifer and E. L. Lozinskii. A Logic for Reasoning with Inconsistency. *Journal of Automated Reasoning, Vol. 9, 1992, pp. 179-215*

[12] M. Kifer, V. S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming Vol. 12, 1992, pp. 335-367*

[13] A.C. Kakas, R.A. Kowalski, F.Toni. Abductive Logic Programming *Journal of Logic Computation, Vol.2, No.6, pp. 719-770*

[14] P. Kullmann. Implementing a deductive database based on generalized annotated logic. *Project work, University of Karlsruhe, Institute for Algorithms and Cognitve Systems, Department of Computer Science, March 1994*

[15] J. Lu, A. Nerode, J. Remmel, V.S. Subrahmanian. Towards a theory of hybrid knowledge bases. *Technical Report TR-93-14, Mathematical Science Institute, Cornell University*

[16] C. Mason, R. R. Johnson. DATMS: A Framework for Distributed Assumption Based Reasoning. *Readings in DAI II, 1990, pp. 293-317, M. Huhns (Editor)*

[17] J. P. Martins, S. C. Shapiro. A Model for Belief Revision. *Artificial Intelligence, 35, 1988, pp. 25-79*

[18] D. McDermott. A general framework for reason maintenance. *Artificial Intelligence 50, 1991, pp. 289-329*

[19] G. Provan. The conceptual complexity of truth maintenance systems. *Proceeding of the ECAI-90, London, pp.512-521*

[20] J. Schü, V.S. Subrahmanian. Maintaining mediated views. *Draft, 1994*

[21] M. Tamer Özsu, P. Valduriez. Principles of Distributed Database Systems. *Prentice Hall, 1991*