NORTH-HOLLAND

AMALGAMATING KNOWLEDGE BASES, III: ALGORITHMS, DATA STRUCTURES, AND QUERY PROCESSING

SİBEL ADALI AND V. S. SUBRAHMANIAN

▷ Integrating knowledge from multiple sources is an important aspect of automated reasoning systems. In the first part of this series of papers, we presented a uniform declarative framework, based on *annotated logics*, for amalgamating multiple knowledge bases when these knowledge bases (possibly) contain inconsistencies, uncertainties, and nonmonotonic modes of negation. We showed that annotated logics may be used, with some modifications, to *mediate* between different knowledge bases. The multiple knowledge bases are amalgamated by embedding the individual knowledge bases into a lattice. In this paper, we briefly describe an SLD-resolution-based proof procedure that is sound and complete w.r.t. our declarative semantics. We will then develop an OLDT-resolution-based query processing procedure, MULTI_OLDT, that satisfies two important properties: (1) *efficient reuse* of previous computations is achieved by maintaining a table —we describe the structure of this table, and show that table operations can be efficiently executed, and (2) *approximate, interruptable query answering* is achieved, i.e., it is possible to obtain an "intermediate, approximate" answer from the QPP by interrupting it at any point in time during its execution. The design of the MULTI_OLDT procedure will include the development of run-time algorithms to incrementally and efficiently update the table. ◁

*This work was supported by the Army Research Office under Grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under Grant F49620-93-1-0065, and by ARPA Order A716 administered by Rome Labs under Contract F30602-93-C-0241.

Address correspondence to V. S. Subrahmanian, Department of Computer Science, Institute for Advanced Computer Studies and Institute for Systems Research, University of Maryland, College Park, MD 20742.

Received September 1994; accepted September 1995.

THE JOURNAL OF LOGIC PROGRAMMING
© Elsevier Science Inc., 1996
655 Avenue of the Americas, New York, NY 10010

0743-1066/96/$15.00
SSDI 0743-1066(95)00148-4

1. INTRODUCTION

Complex reasoning tasks in the real world utilize information from a multiplicity of sources. These sources may represent data and/or knowledge about different aspects of a problem in a number of ways. Wiederhold and his colleagues [38, 39] have proposed the concept of a *mediator*—a device that will express how such an integration is to be achieved.

This is the third in a series of papers developing the theory and practice of integrated databases. In Part I of this series of papers, we developed a language for expressing mediators, and reasoning with them. In particular, we showed that an extension of the "generalized annotated program" (GAP) paradigm of Kifer and Subrahmanian [18] may be used to express mediators. We defined the concept of the "amalgam" of "local" databases $DB_1, \ldots, DB_n$ with a mediatory database, M, and proved a number of results linking the semantics of the local databases with the semantics of the amalgam. In Part II, we developed methods to distribute the mediator across a network of processors in such a way that the distributed semantics coincided with the nondistributed semantics—yet the former led to greater computational efficiency than the latter.

The primary aim of this paper is the *development of query processing procedures* (QPPs, for short) that possess various desirable properties. We will first develop a resolution-based QPP and show it to be sound and complete. However, it is well known that resolution proof procedures are notoriously inefficient, often solving previously solved goals over and over again. OLDT-resolution, due to Tamaki and Sato [33], is a technique which caches previously derived solutions in a table. The theory and implementation of OLDT have been studied extensively by several researchers, including Seki [29, 28] and Warren and his colleagues [9, 10]. Furthermore, it is known that OLDT and magic set computations [5, 6, 27] are essentially equivalent, although they differ in many (relatively minor) details. We will use the OLDT technique as our starting point, and extend it as follows:

(1) *Multiple Databases:* As different databases may provide different answers to the same query, OLDT-resolution needs to be modified to handle a multiplicity of (possibly mutually incompatible) answers to the same query.

(2) *Uncertainty and Time:* Previous formulations of OLDT-resolution did not handle time and uncertainty. We will show how temporal and uncertain answers can be smoothly incorporated into the OLDT paradigm.

(3) *Approximate, Interruptable Query Answering:* In some situations, the user may wish to interrupt the execution of the query processing procedure and ask for a "tentative answer." This kind of feature becomes doubly important when databases contain uncertain and temporal information. When processing a query Q such as: "Is the object O at location L an enemy aircraft?," it is desirable that uncertainty estimates of the truth of this query be revised upwards in a monotonic fashion as the QPP spends more and more time performing inferences. Thus, if the user interrupts the QPP's execution at time t and asks: "What can you tell me about query Q?," the KB should be able to respond with an answer of the form: "I'm not done yet, but at this point I can tell you that Q is true with certainty 87% or more."

(4) *Table Management:* Relatively little work has been done on the development of data structures for managing OLDT-tables (cf. Warren [9, 10]). When a single database with neither uncertainty nor time is considered, the struc-

ture of the OLDT-table can be relatively simple. However, when multiple database operations, uncertainty estimates (that are constantly being revised), and temporal reasoning are being performed simultaneously, the management of the OLDT-table becomes a significant issue. We will develop data structures and algorithms to efficiently manage the OLDT-table.

Our query processing procedure, called MULTI_OLDT, incorporates all of the above features and is described in detail in this paper. In particular, we prove that MULTI_OLDT is a sound and complete query processing procedure. Restricted termination results are also established.

The paper is organized as follows. In Section 3, we provide two examples motivating our work. These examples will be used throughout the paper to illustrate various definitions, data structures, and algorithms. Section 4 contains a brief description of a resolution-style proof procedure, including soundness and completeness results. The MULTI_OLDT procedure is described in detail in Section 5—in particular, this section contains details on the organization of the OLDT-table. We compare our results with relevant work by other researchers in Section 6.

2. PRELIMINARIES

In this section, we give a quick overview of GAPs and the amalgamation theory developed in the first of this series of papers [31].

2.1. Overview of GAPs *(Generalized Annotated Programs)*

The GAP framework syntax proposed in [18] is an extension of the logic programming. It has been proposed as a framework within which inconsistencies, temporal information, and probabilistic logic can be handled in a uniform way. The GAP framework assumes that we have a set $\mathcal{T}$ of truth values that forms a complete lattice under an ordering $\preccurlyeq$. For instance, $(\mathcal{T}, \preccurlyeq)$ may be any one of the following:

(1) *Fuzzy Values:* We can take $\mathcal{T} = [0,1]$—the set of real numbers between 0 and 1 (inclusive) and $\preccurlyeq$ to be the usual $\leq$ ordering on reals.

(2) *Time:* We can take $\mathcal{T}$ to be the set $\text{TIME} = 2^{\mathbf{R}^+}$ where $\mathbf{R}^+$ is the set of nonnegative real numbers, $2^{\mathbf{R}^+}$ is the power-set of the reals, and $\preccurlyeq$ is the inclusion ordering. The reader may note that interval time can therefore be represented. So can sets of time points like the set $\{1,3,7\}$ which refers to the time points 1, 3, and 7; furthermore, $\{1,7\} \preccurlyeq \{1,3,7\}$ since $\{1,7\} \subseteq \{1,3,7\}$.

(3) *Fuzzy Values + Time:* We can take $\mathcal{T} = [0,1] \times \text{TIME}$ and take $\preccurlyeq$ to be the ordering: $[u_1, T_1] \preccurlyeq [u_2, T_2]$ iff $u_1 \leq u_2$ and $T_1 \subseteq T_2$. Here, u_1, u_2 are real numbers in the $[0,1]$ interval and T_1, T_2 are sets of real numbers.

(4) *Four-Valued Logic:* Four-valued logic [8, 17] uses the truth values $\text{FOUR} = \{\bot, \mathbf{t}, \mathbf{f}, \top\}$ ordered as follows: $\bot \preccurlyeq x$ and $x \preccurlyeq \top$ for all $x \in \text{FOUR}$ (cf. Figure 1). In particular, $\mathbf{t}$ and $\mathbf{f}$ are not comparable relative to this ordering. References [7, 8, 17] show how this FOUR-valued logic may be used to reason about databases containing inconsistencies.

This is only a small sample of what $\mathcal{T}$ could be. Using the elements of $\mathcal{T}$, as well as variables ranging over $\mathcal{T}$ (called annotation variables), and function symbols of arity $n \geq 1$ on $\mathcal{T}$ (called annotation functions), *annotation terms* are defined as follows: (1) any member of $\mathcal{T}$ is an annotation term, (2) any annotation variable is an annotation term, and (3) if f is an n-ary annotation function symbol[1] and $t_1,\ldots,t_n$ are annotation terms, then $f(t_1,\ldots,t_n)$ is an annotation term. For instance, if $\mathcal{T}=[0,1]$ and $+,*$ are annotation function symbols interpreted as "plus" and "times," respectively, and V is an annotation variable, then $(V+1)*0.5$ is an annotation term. Strictly speaking, we should write this in prefix notation as: $*(+(V,1),0.5)$, but we will often abuse notation when the meaning is clear from the context.

If A is an atom (in the usual sense of logic) and μ is an annotation, then $A:\mu$ is an annotated atom. For example, when considering $\mathcal{T}=[0,1]$, the atom *broken*(c_1): 0.75 may be used to say: "there is at least a 75% degree of certainty that component c_1 is broken." If $\mathcal{T}=[0,1]\times \mathtt{TIME}$, then annotations are pairs, and an annotated atom for *at_robot*$(3,5):[0.4,\{1,3,7\}]$ says that at each of the time points 1, 3, 7, there is at least a 40% certainty that the robot is at xy-coordinates $(3,5)$.

An annotated clause is a statement of the form

$$A_0:\mu_0 \leftarrow A_1:\mu_1\&\ldots\&A_n:\mu_n$$

where: (1) each $A_i:\mu_i, 0\le i\le n$ is an annotated atom, and (2) for all $1\le j\le n$, μ_j is either a member of $\mathcal{T}$ or is an annotation variable, i.e., μ_j contains no annotation functions. In other words, annotation functions can occur in the heads of clauses, but not in the clause bodies. The above annotated clause (when the annotations are ground) may be read as: "A_0 has truth value at least μ_0 if A_1 has truth value at least μ_1 and $\ldots A_n$ has truth value at least μ_n."

Kifer and Subrahmanian developed a formal model theory, proof theory, and fixpoint theory for GAPs that accurately captures the above-mentioned notion of "firability." In brief, an *interpretation I* assigns to each ground atom an element of $\mathcal{T}$. Intuitively, if $\mathcal{T}=[0,1]$, then the assignment of 0.7 to atom A means that, according to interpretation I, A is true with certainty 70% or more. Interpretation *I satisfies* a ground annotated atom $A:\mu$ iff $\mu\preccurlyeq I(A)$. The notion of satisfaction of formulas containing other connectives, such as $\&$, $\vee$, $\leftarrow$, and quantifiers $\forall,\exists$ is the usual one [30]. In particular, I satisfies the ground annotated clause $A_0:\mu_0\leftarrow(A_1:\mu_1\&\ldots\&A_h:\mu_n)$ iff either $I\not\models(A_1:\mu_1\&\ldots\&A_n:\mu_n)$ or $I\models A_0:\mu_0$. The symbol "$\models$" is read "satisfies." I satisfies a nonground clause iff I satisfies each and every ground instance of the clause (with annotation variables instantiated to members of $\mathcal{T}$ and logical variables instantiated to logical terms).

2.2. *Overview of Amalgamation Theory*

Suppose we have a collection of "local" databases $DB_1,\ldots,DB_n$ over a complete lattice, $\mathcal{T}$, of truth values. In this section, we recall from [31] how the theory of GAPs may be successfully applied to define a *new* lattice of true values that forms the basis of a "mediatory" or "supervisory database." To do so, we first define the DNAME lattice; this is the power set, $2^{\{1,\ldots,n,\mathbf{m}\}}$. The integer i refers to database DB_i, while $\mathbf{m}$ refers to the mediator. Note, in particular, that $2^{\{1,\ldots,n,\mathbf{m}\}}$ is a complete lattice under the set inclusion ordering. Now, the new truth value

[1] As done by Kifer and Subrahmanian [18], we will assume that all annotation function symbols can be interpreted in only one fixed way.

lattice for the mediatory database is DNAME $\times \mathcal{T}$, where the ordering $\preccurlyeq'$ on this lattice is given as follows: $[D_1, \mu_1] \preccurlyeq' D_2, \mu_2]$ iff $D_1 \subseteq D_2$ and $\mu_1 \preccurlyeq \mu_2$ ($\preccurlyeq$ is the ordering for the lattice $\mathcal{T}$.)

We assume that we have a set of variables (called DNAME variables) ranging over $2^{\{1,\ldots,n,\mathbf{m}\}}$. If $A:\mu$ is an atom over lattice $\mathcal{T}$, V is a DNAME-variable, and $D \subseteq \{1,\ldots,n,\mathbf{m}\}$, then $A:[D,\mu]$ and $A:[V,\mu]$ are called *amalgamated atoms*. Intuitively, if $\mathcal{T}=[0,1]$, the amalgamated atom $at\_robot(3,4):[\{1,2,3\},0.8]$ says that, according to the (joint) information of databases 1, 2, and 3, the degree of certainty that the robot is at location $(3,4)$ is 80% or more.

An amalgamated clause is a statement of the form

$$A_0:[D_0,\mu_0] \leftarrow A_1:[D_1,\mu_1] \& \ldots \& A_n:[D_n,\mu_n]$$

where $A_0:[D_0,\mu_0], \ldots, A_n:[D_n,\mu_n]$ are amalgamated atoms. An *amalgamated database* is a collection of clauses of this form.

Mediatory Database. Suppose $DB_1,\ldots,DB_n$ are GAPs. A *mediatory database*[2] M is a set of amalgamated clauses such that every ground instance of a clause in M is of the form

$$A_0:[\{\mathbf{m}\},\mu] \leftarrow A_1:[D_1,\mu_1] \& \ldots \& A_n:[D_n,\mu_n]$$

where, for all $1 \le i \le n$, $D_i \subseteq \{1,\ldots,n,\mathbf{m}\}$.

Intuitively, ground instances of clauses in the mediator say: "If the databases in set D_i, $1 \le i \le n$, (jointly) imply that the truth value of A_i is at least μ_i, then the mediator will conclude that the truth value of A_0 is at least μ." This mode of expressing mediatory information is very rich—in [31], it is shown that it is possible to express prioritized knowledge about predicates, prioritized knowledge about objects, as well as methods to achieve a consensus in this framework.

We now define the concept of an *amalgam* of local databases $DB_1,\ldots,DB_n$ via a *mediator M*. First, each clause C in DB_i of the form

$$A_0:\mu_0 \leftarrow A_1:\mu_1 \& \ldots \& A_n:\mu_n$$

is replaced by the amalgamated clause, $AT(C)$:

$$A_0:[\{i\},\mu_0] \leftarrow A_1:[\{i\},\mu_1] \& \ldots \& A_n:[\{i\},\mu_n].$$

We use $AT(DB_i)$ to denote the set $\{AT(C) | C \in DB_i\}$. The *amalgam* of $DB_1,\ldots,DB_n$ via a *mediator M* is the amalgamated knowledge base $(M \cup \bigcup_{i=1}^{n} AT(DB_i))$. The model theory for amalgamated knowledge bases is (slightly) different from that of individual GAPs because it must account for a new type of variable, viz. the DNAME variables. An A-*interpretation*, J, for an amalgamated database is a mapping from the set of ground atoms of our base language to the set of functions from $\{1,\ldots,n,\mathbf{m}\}$ to $\mathcal{T}$. Thus, for each $A \in B_L$, $J(A)$ is a mapping from $\{1,\ldots,n,\mathbf{m}\}$ to $\mathcal{T}$. In other words, if $J(A)(i)=\mu$, then according to the interpretation J, DB_i says the truth value of A is at least μ. Given a subset, D, of $\{1,\ldots,n,\mathbf{m}\}$, we use $J(A)(D)$ to denote $\sqcup_{i \in D}(J(A))(i)$. An A-*interpretation*, J satisfies the ground amalgamated atom A: $[D,\mu]$ iff $\mu \preccurlyeq J(A)(D)$. Here, $\sqcup$ denotes "least upper bound (lub)." The concepts of A-model and A-consequence are defined in the usual way. All of the other symbols are interpreted in the same

[2] When the databases being integrated are geographically dispersed across a network, it is common to distribute the mediator so that bottlenecks (e.g., due to network problems) do not have a devastating effect. In this paper, we will not study issues relating to implementing distributed mediators (although we are doing so in a separate, concurrent effort).

way as for ordinary $\mathcal{T}$-valued interpretations, with the caveat that for quantification, DNAME variables are instantiated to subsets of $\{1,\ldots,n,\mathbf{m}\}$ and other annotation variables are instantiated to members of $\mathcal{T}$. Note that we will always use the word **A**-*interpretation* to denote an interpretation of an amalgamated KB, and use the expression "interpretation" of "$\mathcal{T}$-interpretation" to refer to an interpretation of a GAP.

3. MOTIVATION

In this section, we will present two motivating examples—the first is a set of deductive databases expressed using FOUR-valued logic describing a static robotic domain (i.e., one where the world remains constant). The second example extends this to reason about a dynamically changing world, and thus incorporates both uncertainty and time. These examples will be used throughout the paper to illustrate various intuitions as they arise in the paper.

We will assume that the reader is familiar with generalized annotated programs (GAPs) as defined in [18].

3.1. Robot Example

Consider two mobile robots, $r1$ and $r2$, that are operating in a common workspace. Each of these two robots have access to three databases; one of these databases represents information about the locations of objects in the workspace (cf. Figure 2), the second represents information about the weight of these objects, while the third represents information about the temperature of the objects. The last two databases also contain information about what kinds of loads the individual robots can lift. Each of these three databases is expressed over the lattice FOUR shown in Figure 1, and examples of clauses in each database are given below

$\boxed{DB_1:}$

$at(r1,1,3):\mathbf{t}\leftarrow$

$at(r2,2,4):\mathbf{t}\leftarrow$

$at(a,1,1):\mathbf{t}\leftarrow$

$at(b,2,2):\mathbf{t}\leftarrow$

$at(c,3,5):\mathbf{t}\leftarrow$

$at(d,4,2):\mathbf{t}\leftarrow$

$right(E1,E2):\mathbf{t}\leftarrow at(E1,X1,Y1):\mathbf{t}\&at(E2,\ X2,Y1):\mathbf{t}\&X1>X2.$

$left(e1,E2):\mathbf{t}\leftarrow at(E1,X1,Y1):t\&at(E2,X2,Y1):\mathbf{t}\&X1<X2.$

$above(E1,E2):\mathbf{t}\leftarrow at(E1,X1,Y1):\mathbf{t}\&at(E2,X1,Y2):\mathbf{t}\&Y1>Y2.$

$below(E1,E2):\mathbf{t}\leftarrow at(E1,X1,Y1):\mathbf{t}\&at(E2,X1,Y2):\mathbf{t}\&Y2>Y1.$

$at(E1,X,Y):\mathbf{f}\leftarrow at(E2,X,Y):\mathbf{t}\&E1\neq E2.$

This database specifies where the objects are located (including the robots), and also specifies relations such as "entity $E1$ is to the right of entity $E2$ if ...," and "entity $E1$ is to the left of $E2$ if...,". There is also a rule saying that two things

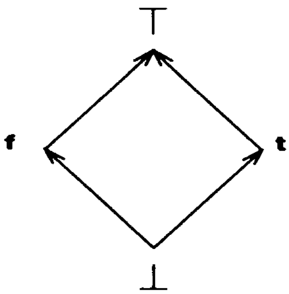

FIGURE 1. The truth value lattice FOUR.

cannot be at the same place. We assume that relations like $>$, $<$, and $=$ are evaluated in the standard way. Intuitively, the first rule above says: "If the entity $E1$ is at location $(X1, Y1)$ and entity $E2$ is at location $(X2, Y1)$ and $X1 > X2$, then $E1$ is to the right of $E2$."

$\boxed{DB_2:}$

$weight(a, 36) : \mathbf{t} \leftarrow$

$weight(b, 19) : \mathbf{t} \leftarrow$

$weight(c, 48) : \mathbf{t} \leftarrow$

$weight(d, 27) : \mathbf{t} \leftarrow$

$can\_lift(r1, X) : \mathbf{t} \leftarrow weight(X, W) : \mathbf{t} \,\&\, W < 50.$

$can\_lift(r1, X) : \mathbf{f} \leftarrow weight(X, W) : \mathbf{t} \,\&\, W \geq 50.$

$can\_lift(r2, X) : \mathbf{t} \leftarrow weight(X, W) : \mathbf{t} \,\&\, W < 30.$

$can\_lift(r2, X) : \mathbf{f} \leftarrow weight(X, W) : \mathbf{t} \,\&\, W \geq 30.$

$\boxed{DB_3:}$

$temp(a, 92) : \mathbf{t} \leftarrow$

$temp(b, 61) : \mathbf{t} \leftarrow$

$temp(c, 55) : \mathbf{t} \leftarrow$

$temp(d, 112) : \mathbf{t} \leftarrow$

$can\_lift(r1, X) : \mathbf{t} \leftarrow temp(X, T) : \mathbf{t} \,\&\, T < 60.$

$can\_lift(r1, X) : \mathbf{f} \leftarrow temp(X, T) : \mathbf{t} \,\&\, T \geq 60.$

$can\_lift(r2, X) : \mathbf{t} \leftarrow temp(X, T) : \mathbf{t} \,\&\, T < 120.$

$can\_lift(r2, X) : \mathbf{f} \leftarrow temp(X, T) : \mathbf{t} \,\&\, T \geq 120.$

Using DB_2 alone, we may conclude that $r1$ can lift any a, b, c, d, while using DB_3 alone, we may conclude that $r1$ can lift only c. Similarly, DB_2 alone tells us that $r2$ can lift b and d, while using DB_3 alone, we may conclude that $r2$ can lift all of a, b, c, and d. Clearly, this leads to an inconsistency. In addition to resolving such conflicts, we may wish to coordinate what should be done by the two robots $r1$ and $r2$. A mediatory database is a database that specifies how to resolve such conflicts and how to achieve the desired coordination. For instance, it may be the case that $r1$ moves easily in the vertical direction, while $r2$ moves easily in the horizontal

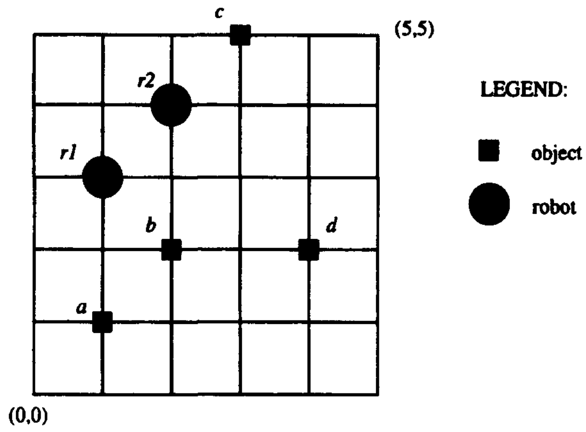

FIGURE 2. The location of objects in the workspace.

direction. If an object is above or below $r1$, and the mediator determines that $r1$ can lift that object, then the mediator may decide to command $r1$ to lift that object. Similarly, if an object is to the left or right of $r2$, and the mediator determines that $r2$ can lift that object, then the mediator may decide to command $r2$ to lift that object. If the object is not exactly above or below $r1$ or to the right, left of $r2$, then the mediator will first command $r1$ to lift the object. If no command is issued to $r1$ to lift an object, then $r2$ will be commanded to lift that object. These are formalized using the following "mediatory" knowledge base.

$$\begin{aligned}
&\mathit{can\_lift}(r1,X):[\{\mathbf{m}\},V) \leftarrow \mathit{can\_lift}(r1,X):[\{2,3\},V].\\
&\mathit{can\_lift}(r2,X):[\{\mathbf{m}\},V_1 \sqcap V_2] \leftarrow \mathit{can\_lift}(r2,X):[\{2\},V_1]\ \&\\
&\qquad\qquad \mathit{can\_lift}(r2,X):[\{3\},V_2].\\
&\mathit{command\_lift}(X,r1):[\{\mathbf{m}\},V] \leftarrow \mathit{can\_lift}(r1,X):[\{\mathbf{m}\},V]\ \&\\
&\qquad\qquad \mathit{above}(X,r1):[\{1\},\mathbf{t}].\\
&\mathit{command\_lift}(X,r1):[\{\mathbf{m}\},V] \leftarrow \mathit{can\_lift}(r1,X):[\{\mathbf{m}\},V]\ \&\\
&\qquad\qquad \mathit{below}(X,r1):[\{1\},\mathbf{t}].\\
&\mathit{command\_lift}(X,r2):[\{\mathbf{m}\},V] \leftarrow \mathit{can\_lift}(r2,X):[\{\mathbf{m}\},V]\ \&\\
&\qquad\qquad \mathit{left}(X,r2):[\{1\},\mathbf{t}].\\
&\mathit{command\_lift}(X,r2):[\{\mathbf{m}\},V] \leftarrow \mathit{can\_lift}(r2,X):[\{\mathbf{m}\},V]\ \&\\
&\qquad\qquad \mathit{right}(X,r2):[\{1\},\mathbf{t}].\\
&\mathit{command\_lift}(X,r1):[\{\mathbf{m}\},V] \leftarrow \mathit{can\_lift}(r1,X):[\{\mathbf{m}\},V].\\
&\mathit{command\_lift}(X,r2):[\{\mathbf{m}\},\mathbf{t}] \leftarrow \mathit{can\_lift}(r2,X):[\{2,3\},\mathbf{t}]\ \&\\
&\qquad\qquad \mathit{command\_lift}(X,r1):[\{\mathbf{m}\},\mathbf{f}].
\end{aligned}$$

The first two rules in the above mediatory knowledge base are very interesting. As far as robot $r1$ is concerned, the mediator is willing to accept the truth value provided by any of the databases—in other words, the mediator is indecisive, and acts as if both what DB_2 says is correct and what DB_3 says is correct (even though they may contradict each other). This may be an appropriate strategy when robot

$r1$ is a very inexpensive robot, and the task of lifting the objects is critical. The second rule says that the mediator only concludes that $r2$ can lift an object if both databases DB_2 and DB_3 say it can (consensus).

The amalgam of local database DB_1, DB_2, DB_3 with the mediatory database M is found as defined in [31]. To do this, $\mathscr{D}$-term annotation in all the clauses in database DB_i are set to $\{i\}$, and these modified clauses are added to the amalgam. For example, the clause

$$can\_lift(r1, X) : [\{3\}, \mathbf{t}] \leftarrow temp(X, T) : [\{3\}, \mathbf{t}] \& T < 60$$

is added to the amalgam by modifying the clause

$$can\_lift(r1, X) : \mathbf{t} \leftarrow temp(X, T) : \mathbf{t} \& T < 60$$

in database DB_3. Similarly, for the clause

$$t(E1, X, Y) : \mathbf{f} \leftarrow at(E2, X, Y) : \mathbf{t} \& E1 \neq E2$$

in database DB_1, the following clause is added to the amalgam:

$$at(E1, X, Y) : [\{1\}, \mathbf{f}] \leftarrow at(E2, X, Y) : [\{1\}, \mathbf{t}] \& E1 \neq E2.$$

4. A RESOLUTION-BASED QUERY PROCESSING PROCEDURE

In this section, we will develop a framework for processing queries to amalgamated databases. This procedure is a resolution-based procedure, and hence inherits many of the disadvantages of existing resolution-based strategies. It is similar to work by Lu et al. [23] who have independently developed a framework for query processing in GAPs. As stated by Leach and Lu [21], the work of [23] applies to not just the Horn-clause fragment of annotated logic (which is the case in our work), but to the full-blown logic. However, [23] does *not* deal with annotation variables and annotation functions—our results apply to those cases as well.

The work described here is intended as a stepping stone for the development of a more sophisticated procedure, called MULTI_OLDT, that will be described in Section 5.

We will now define the concept of the *up-set* of an annotation, or a set of annotations. Intuitively, given a set Q of annotations, the up-set of Q is simply the set of all elements in the truth value lattice that are larger than some element in Q.

Definition 4.1. Suppose $\langle \mathscr{R}; \leq \rangle$ is a partially ordered set and $Q \subseteq \mathscr{R}$. Then $\Uparrow Q = \{y \in \mathscr{R} | (\exists x \in Q) x \leq y\}$.

To better understand the intended meaning of up-sets, observe that whenever an A-interpretation I satisfies an atom A: $[D, \mu]$, we have $I(A)(D) \preccurlyeq \mu$ by the definition of satisfaction; hence, we also have that $I(A)(D) \in \Uparrow \mu$.

A *set expansion function* is a mapping $\mathscr{T} \rightarrow 2^{\mathscr{T}}$ from truth values to sets of truth values. Hence, $f(\mu) = \Uparrow \mu$ is a set annotation function. A *set-expanded atom* is an expression of the form $A : [D, \mu_s]$ where μ_s denotes a set of truth values. Hence, atoms $p : [\{1\}, \Uparrow \mu]$, $p : [\{1\}, \Uparrow \mathbf{t}]$, $p : [\{1\}, \{\mathbf{t}, \top\}]$ are all set expanded atoms. In the sequel, we will often use the notation μ_s to denote a *set of truth values* (*annotations*).

Using the concept of set expanded atoms, we now define the concept of a *regular representation* of a clause. Later in this section, we will define a resolution-based strategy that uses regular representations of amalgamated clauses instead of the amalgamated clauses themselves. The advantage is that the expensive *reductant* rule of inference introduced by Kifer and Lozinskii [17] and later studied by Kifer and Subrahmanian [18] can be eliminated by using regular representations.

Definition 4.2. Given a clause C of the form

$$A_0:[D_0,\mu_0] \leftarrow A_1:[D_1,\mu_1]\&\ldots\&A_n:[D_n,\mu_n]$$

the *regular representation* of C, denoted by C^*, is the expression

$$A_0:[D_0,\Uparrow\mu_0] \leftarrow A_1:[D_1,\Uparrow\mu_1]\&\ldots\&A_n:[D_n,\Uparrow\mu_n].$$

In other words, the regular representation is obtained by replacing the annotation terms by their up-sets.

Example 4.1. (Robot Example Revisited) Consider the following rule from DB_2 of the Static Robot example.

$$\textit{can\_lift}(r1,X):\mathbf{t} \leftarrow \textit{weight}(X,W):\mathbf{t}\&W<50.$$

The amalgamated form of this, as defined in [31], is

$$\textit{can\_lift}(r1,X):[\{2\},\mathbf{t}] \leftarrow \textit{weight}(X,W):[\{2\},\mathbf{t}]\&W<50.$$

The regular representation of this is

$$\textit{can\_lift}(r1,X):[\{2\},\Uparrow\mathbf{t}] \leftarrow \textit{weight}(X,W):[\{2\},\Uparrow\mathbf{t}]\&W<50,$$

and since $\Uparrow\mathbf{t}=\{\mathbf{t},\top\}$, the above clause becomes

$$\textit{can\_lift}(r1,X):[\{2\},\{\mathbf{t},\top\}] \leftarrow \textit{weight}(X,W):[\{2\},\{\mathbf{t},\top\}]\&W<50.$$

(We assume that the constraint $W<50$ is a predefined evaluable relation.) □

Definition 4.3. (S-Satisfaction) An **A**-interpretation I S-satisfies an expanded atom $A:[D,\mu_s]$ where $D\subseteq\{1,\ldots,n,\mathbf{m}\}$ and $\mu_s\in 2^{\mathcal{T}}$ iff $I\models^{\mathbf{A}}A:[D,\mu]$ for some $\mu\in\mu_s$.

The notion of an S-logical consequence is similar to that in classical logic—only now, S-satisfaction is considered instead of ordinary satisfaction.

Definition 4.4. A set-annotated amalgamated atom $A:[D_1,f_{s_1}(\mu_1)]$ is said to be an S-consequence of another set annotated amalgamated atom $B:[D_2,f_{s_2}(\mu_2)]$ (denoted by $B:[D_2,f_{s_2}(\mu_2)]\models^S A:[D_1,f_{s_1}(\mu_1)]$), iff any **A**-*interpretation* I that S-satisfies $B:[D_2,f_{s_2}(\mu_2)]$ also S-satisfies $A:[D_1,f_{s_1}(\mu_1)]$.

Example 4.2. Let the truth value lattice be FOUR, and let I be an **A**-interpretation such that $I(A)(1)=\bot$ and $I(A)(2)=\mathbf{t}$. $\bigsqcup_{d\in\{1,2\}}I(A)(d)=\mathbf{t}$. Hence, I S-satisfies $A:[\{1,2\},\{\mathbf{t},\mathbf{f},\top\}]$ since $\mathbf{t}\in\{\mathbf{t},\mathbf{f},\top\}$. □

Just as we defined the notion of "regular representation" of clauses, we also need to define the notion of "regular representation" of queries.

Definition 4.5. A query Q is a statement of the form

$$\leftarrow A_1:[D_1, \mu_1]\&\ldots\&A_n:[D_m, \mu_m]$$

where all of the free variables of the query are assumed to be universally quantified.[3] A *set-expanded query* is a query of the form

$$\leftarrow A_1:[D_1, \mu_s]\&\ldots\&A_n:\left[D_m, \mu_{s_m}\right]$$

where each $A_i:[D_i, \mu_{s_i}]$ is a set-expanded atom. Given a query, the regular representation of the query Q, denoted Q^*, is the expression

$$A_1:[D_1, \mathcal{T}-\Uparrow \mu_1] \vee \cdots \vee A_n:[D_m, \mathcal{T}-\Uparrow \mu_m] \leftarrow .$$

Thus, Q^* is a special kind of *set-expanded query.*

The following result follows immediately from the definitions, and is given without proof.

Proposition 4.1. Suppose I is an A-interpretation.

1. *I satisfies a ground clause C iff I S-satisfies C*.*
2. *I satisfies a ground query Q iff I S-satisfies Q*.* □

We now come to the central concept in this section, viz. that of an S-resolvent.

Definition 4.6. (S-Resolution) Let C^* be the regular representation of a clause C, given by

$$A_0:[D_0, \Uparrow \mu_0] \leftarrow A_1:[D_1, \Uparrow \mu_1]\&\ldots\&A_n:[D_n, \Uparrow \mu_n],$$

and let W^* be the following set annotated query:

$$B_1:\left[D_{q_1}, \mu_{q_{s_1}}\right] \vee \cdots \vee B_m:\left[D_{q_m}, \mu_{q_{s_m}}\right] \leftarrow$$

where $\mu_{q_{s_1}}$, $1 \leq j \leq m$, are in set expansion form. Suppose B_k and A_0 are unifiable via mgu θ, and suppose $D_0 \subseteq D_{q_i}$. Then the S-resolvent of W^* and C^* is the expression

$$\begin{aligned}&\Big(A_1:[D_1, \mathcal{T}-\Uparrow \mu_1] \vee \cdots \vee A_n:[D_n, \mathcal{T}-\Uparrow \mu_n] \vee\\ &B_1:\left[D_{q_1}, \mu_{q_{s_1}}\right] \vee \cdots \vee B_{i-1}:\left[D_{q_{i-1}}, \mu_{q_{s_{i-1}}}\right] \vee\\ &B_{i+1}:\left[D_{q_{i+1}}, \mu_{q_{s_{i+1}}}\right] \vee \cdots \vee B_m:\left[D_{q_m}, \mu_{q_{s_m}}\right] \vee\\ &B_i:\left[D_{q_i}, \mu_{q_{s_i}} \cap (\Uparrow \mu_0)\right]\Big)\theta \leftarrow .\end{aligned}$$

In case $\mu_{q_{s_i}}\theta \cap (\Uparrow \mu_0\theta) = \mu_s$ is ground and μ_s evaluated to $\emptyset$, then we simplify the above S-resolvent by removing the atom $(B_i:[D_{q_i}, \mu_{q_{s_i}} \cap (\Uparrow \mu_0)])\theta$.

All the atoms $(A_1:[D_1, \mathcal{T}-\Uparrow \mu_1])\theta, \ldots (A_n:[D_n, \mathcal{T}-\Uparrow \mu_n])\theta, (B_i:[D_{q_i}, \mu_{q_{s_i}} \cap (\Uparrow \mu_0)])\theta$ in the S-resolvent of W^* and C^* will be referred to as the *children* of the set annotated atom $B_i:[D_{q_i}, \mu_{q_{s_i}}]$. Conversely, $B_i:[D_{q_i}, \mu_{q_{s_i}}]$ is the *parent* of all the atoms in the S-resolvent. The atom $(A_0:[D_0, \mu_0])\theta$ will be referred to as the *twin* of $B_i:[D_{q_i}, \mu_{q_{s_i}}]$. These expressions will be used when MULTI_OLDT-resolution is introduced.

[3] A query can be thought of as a headless Horn-clause, i.e. $\forall(\leftarrow Q)$. The negation of the above query is the statement $(\exists)(A_1:[D_1, \mu_1]\& \cdots \&A_n:[D_m, \mu_m])$.

Two important points that distinguish S-resolution for amalgamated knowledge bases from GAPs are the following:

- First, it is possible that no atom may be "eliminated" during an S-resolution step. This occurs if μ_s above is not equal to $\emptyset$.
- Second, S-resolvents are inherently asymmetric due to the use of the inequality $D_0 \subseteq D_{q_i}$.

Dealing with variables in annotations and $\mathscr{D}$-terms requires extra attention. We will examine annotation variables more closely in the following sections. In this section, we only note that whenever the atoms $B_i : [D_i, \mu_{s_i}]$ in the query contain annotation variables, then the set μ_{s_i} cannot be compared with the empty set; hence, this atom cannot be removed from the query. Similarly, whenever the $\mathscr{D}$-terms contain variables, the condition $D_0 \subseteq D_{q_i}$ cannot be checked. It is possible to introduce constraints to the program specifying the possible values for D_0 and D_{q_i}, to circumvent this problem and expand the theory to deal with these constraints. However, in this paper, we will concentrate on the annotation variables, and assume that all of the $\mathscr{D}$-terms are ground.

Before proceeding to study soundness and completeness issues pertaining to S-resolution, we present an example.

Example 4.3. Consider the truth value lattice FOUR. Let C be the clause

$$p(a) : [\{1\}, \top] \leftarrow$$

and let Q be the query $\leftarrow p(X) : [\{1,2\}, \mathbf{t}]$. The regular representation, Q^*, of the above query is

$$p(X) : [\{1,2\}, \{\mathbf{f}, \bot\}] \leftarrow .$$

$\theta = \{X = a\}$ is the mgu of $p(a)$ and $p(X)$, and hence C^* and Q^* can be S-resolved, yielding

$$(p(X) : [\{1,2\}, \{\mathbf{f}, \bot\} \cap \{\top\}] \leftarrow)\{X = a\}$$

as the S-resolvent. This is reduced to the empty clause because $\{\mathbf{f}, \bot\} \cap \{\top\} = \emptyset$. □

Definition 4.7. An *S-deduction* from a query Q_0 and an amalgamated knowledge base is a sequence: $\langle Q_0^*, C_0^*, \theta_0 \rangle, \ldots, \langle Q_n^*, C_n^*, \theta_n \rangle$ such that Q_{i+1}^* is an *S-resolvent* of Q_i^* and C_i^* via mgu θ_i, $(0 \le i < n)$. Q_0^* is the regular representation of Q_0 and C_i^* is the regular representation of some clause C, $(0 \le i \le n)$ in the knowledge base.

An S-deduction is called an *S-refutation* if it is finite and the last query is the empty clause.

Theorem 4.1. (*Soundness of S-Resolution*) *Suppose I S-satisfies a clause* $C^* \equiv A_0 : [D_0, \Uparrow \mu_0] \leftarrow A_1 : [D_1, \Uparrow \mu_1] \& \ldots \& A_n : [D_n, \Uparrow \mu_n]$ *and a set-annotated query* $Q_k^* \equiv B_1 : [D_{q_1, \mu q_{s_1}}] \vee \cdots \vee B_m : [D_{q_m}, \mu_{q_{s_m}}] \leftarrow$. *Then, I S-satisfies the S-resolvent of* C^* *and* Q_k^*. □

The following definition from [31] is needed for proving the Completeness results for amalgamated knowledge bases. Given an amalgamated knowledge base

Q, it is possible to associate with Q an operator $\mathbf{A}_Q$ that maps A-interpretations to A-interpretations.

Definition 4.8 [31]. Suppose Q is an amalgamated knowledge base. We may associate with Q an operator, $\mathbf{A}_Q$, that maps A-interpretations to A-interpretations as follows:

$\mathbf{A}^*_Q(I)(A)(D) = \sqcup\{\mu | A:[D,\mu] \leftarrow B_1:[D_1,\mu_1]\&\ldots\&B_n:[D_n,\mu_n]$ is a ground instance of a clause in Q, and for all $1 \leq i \leq n$, $\mu_i \leq I(B_i)(D_i)\}$.

$\mathbf{A}_Q(I)(A)(D) = \sqcup_{D' \subseteq D}\mathbf{A}'_Q(I)(A)(D')$, for all $D \subseteq \{1,\ldots,n,s\}$.

Subrahmanian [31] proved that $\mathbf{A}_Q$ is monotonic. Hence, $\mathbf{A}_Q$ has a least fixpoint which is identical to $\mathbf{A}_Q \uparrow \eta$ for some ordinal η. Unlike ordinary logic programs, even if η is ω, it is possible that $(\mathbf{A}_Q \uparrow \omega)(A)(i) = \mu$, but there is no integer $j < \omega$ such that $(\mathbf{A}_Q \uparrow j)(A)(i) = \mu$. This may occur because μ is the lub of an infinite sequence, $\mu_0, \mu_1, \ldots$ where $\mu_k = (\mathbf{A}_Q \uparrow k)(A)(i)$.

An amalgamated knowledge base is said to possess the fixpoint reachability property iff, whenever $(\mathbf{A}_Q \uparrow \eta)(A)(i) = \mu$, there is an integer $j < \omega$ such that $(\mathbf{A}_Q \uparrow j)(A)(i) = \mu$. The fixpoint reachability property is critical for completeness because, otherwise, we need to take recourse to infinitary proofs. It is well known [18] that, even in the case of GAPs, the fixpoint reachability property is critically necessary for obtaining completeness results. The proof of the following result is contained in the Appendix.

Theorem 4.2. (Completeness of S-Resolution) Suppose $P \models Q$, where P is an amalgamated knowledge base that possess the fixpoint reachability property. Then there is an S-refutation of $(\leftarrow Q)^*$ from P. □

The above completeness theorem specifies the existence of refutations of queries that are consequences of P. In this paper, we do not deal with computation rules [22]. The use of different fair computation rules in implementing a search strategy for resolution has been studied by many authors such as Vieille [35]. Our MULTI_OLDT procedure described in the rest of the paper may work with any of these computation rules.

5. MULTI_OLDT RESOLUTION

The previous section describes a sound and complete proof procedure for amalgamated knowledge bases. The completeness result for S-resolution asserts the existence of a refutation for $(\leftarrow Q)^*$ whenever Q is a logical consequence of a program P possessing the fixpoint reachability property.

Consider a query of the form $\leftarrow \mathit{can\_lift}(r1,a):V$ in the robot example. An S-resolution may terminate by setting $V = \bot$, which is a correct refutation—however, in this query, we are really interested in finding the *maximal* truth value μ such that $\mathit{can\_lift}(r1,a):\mu$ is true. The completeness of the S-resolution procedure described in the preceding section does not guarantee that this refutation will be found; it only guarantees that *some* substitution which causes $\mathit{can\_lift}(r1,a):V$ to be true will be found.

Furthermore, the robot may have a *hard deadline* within which to perform its action(s). Thus, it should have the ability to *interrupt* the query processing module and request the "best" answer obtained thus far.

How these two goals are achieved efficiently is the subject of this section of the paper. As a preview, we give a small example.

Example 5.1. Consider the databases DB_1, DB_2, and DB_3 in the static robot example, and suppose we ask the query

$$\leftarrow can\_lift(r1, b) : [\{1,2,3\}, V].$$

The query Q says: "What is the maximal truth value V such that $can\_lift(r1, b) : [\{1,2,3\}, V]$ can be concluded?" Q^* is: $can\_lift(r1, b) : [\{1,2,3\}, \mathcal{T} - \Uparrow V] \leftarrow$. Let us see what happens.

1. Resolving this query with the (regular representation of the) first rule in DB_2 yields, as resolvent, $Q_1^* \equiv$:

$$can\_lift(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t}] \vee$$
$$weight(b, W) : [\{2\}, \mathcal{T} - \Uparrow \mathbf{t}] \vee W \geq 50 \leftarrow .$$

2. Resolving this query with the (regular representation of the) second fact in DB_2 yields

$$can\_left(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t}] \vee$$
$$weight(b, 19) : [\{2\}, (\mathcal{T} - \Uparrow \mathbf{t}) \cap \Uparrow \mathbf{t}] \vee 19 \geq 50 \leftarrow .$$

As $(\mathcal{T} - \Uparrow \mathbf{t}) \cap \Uparrow \mathbf{t} = \emptyset$, the atom $weight(b, 19) : [(\mathcal{T} - \Uparrow \mathbf{t}) \cap \Uparrow \mathbf{t}]$ can be eliminated from the resolvent, and the evaluable atom $19 \geq 50$ may also be so eliminated, thus leaving us with the resolvent

$$can\_lift(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t}] \leftarrow .$$

Note that at this stage, we are in a position to conclude that V must be at least **t** *for the following reasons*:

- All atoms in the body of the first rule in DB_2 have been resolved away (i.e., the subgoals generated by atoms in the body of this rule have been achieved), and
- $V = \mathbf{t}$ represents the *maximal* lattice value such that

$$(\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t} = \emptyset.$$

Hence, we may conclude that V's truth value is *at least* **t** (w.r.t. the lattice ordering).

3. After concluding that V's truth value is *at least* **t**, we continue resolving the query from 2 above. We resolve it with the second clause in DB_3 to get

$$can\_lift(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t} \cap \Uparrow \mathbf{f}] \vee$$
$$temp(b, T) : [\{3\}, \mathcal{T} - \Uparrow \mathbf{t}] \vee T < 60 \leftarrow .$$

4. Resolving the above query with the second fact in DB_3 gives

$$can\_lift(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t} \cap \Uparrow \mathbf{f}] \vee$$
$$temp(b, 61) : [\{3\}, (\mathcal{T} - \Uparrow \mathbf{t}) \cap \Uparrow \mathbf{t}] \vee 61 < 60 \leftarrow .$$

As explained in 2, second and third atoms in the query can be eliminated, leaving us with the query

$$can\_lift(r1, b) : [\{1,2,3\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t} \cap \Uparrow \mathbf{f}] \leftarrow .$$

To evaluate this query to the empty clause, we must find the maximal truth value of V that satisfies the following equation $\equiv (\mathcal{T} - \Uparrow V) \cap \Uparrow \mathbf{t} \cap \Uparrow \mathbf{f} = \emptyset$. This is equivalent to $\equiv (\mathcal{T} - \Uparrow V) \cap \{\top\} = \emptyset$, and we conclude that $V = \top$ is the solution to this equation that maximizes the value of V. □

As we can see from the example above, finding the maximum truth value of an annotation variable that enables us to eliminate a query atom results in a maximization problem with some constraints. Each resolution with the atom introduces new restrictions on the set of truth values its annotation variable can legitimately have. Notice that these restrictions can be part of another maximization problem. As an example, suppose we have the following clause in the (regular representation of) DB_1:

$$can\_lift(X,b):[\{1\}, \Uparrow V_1] \leftarrow can\_lift(X,b):[\{2\}, \Uparrow V_1].$$

In other words, DB_1 contains the information that DB_2 is a more reliable source of information as far as the object b is concerned. When we resolve this clause with the original query in the above example, we get the following query:

$$can\_lift(r1,b):[\{1,2,3\},(\mathcal{T} - \Uparrow V) \cap \Uparrow V_1]$$
$$\vee\, can\_lift(X,b):[\{2\},\mathcal{T} - \Uparrow V_1] \leftarrow .$$

Here, V_1 is going to be maximized as well, and we want to know how the current maximum value of V is affected by the changes in the value of V_1. We are now going to formalize this idea.

5.1. Maximization Problems

5.1.1. Declarative Maximization. We would like to consider the situation where an end-user asks an amalgamated knowledge base P query Q of the form $\leftarrow A:[D,V]$ where D is ground and V is an annotation variable.

If A is an atom (not necessarily ground), then an answer to Q is a (substitution, truth value) pair (σ, μ) such that $(\forall)A\sigma:[D,\mu]$ is an **A**-logical consequence of P. A *maximal answer to query* Q is an answer (substitution, truth value) pair (σ, μ) such that there is no μ' where $\mu < \mu'$ such that $(\forall)A\sigma:[D,\mu']$ is an **A**-logical consequence of P.

Note that the above notions of maximal answers are defined completely declarative, and do not depend on any specific query processing procedure. Furthermore, note that in the second case listed above, multiple maximal answers may exist, depending on the substitution involved in the (substitution, truth value) pair. However, for any given substitution σ, at most one (substitution, truth value) pair of the form (σ, μ) may exist. A good part of the rest of this paper will be devoted to computing maximal answers, and we will develop techniques to do so in the rest of this section. Theorem 5.4 later in our paper shows that our procedure is sound and complete w.r.t. computation of maximal answers. The first step in this procedure is to interleave certain maximization problems during an S-resolution computation. This is the subject of Section 5.1.2 below.

Before proceeding to the interleaving of maximization problems and S-resolution, we observe that computing maximal answers is often much harder than just computing S-refutations. The reason for this is that many different S-refutations may be needed in order to compute the maximal answer to a query. In this paper, we try to develop procedures that are as efficient as possible to compute maximal answers, while recognizing that the problem of maximal answer computation is itself intrinsically very complex.

5.1.2. Maximization Problems in Query Processing. In this section, we will describe certain maximization problems that can be solved during the construction of S-deductions. These maximization problems will eventually help us in computing maximal answers as described in Section 5.1.1.

Definition 5.1. (Maximization Problem) Let $\mathcal{T}$ be a complete lattice of truth values, $V_1,\dots,V_n$ be annotation terms, and f_{obj}: $\mathcal{T}^n \to \mathcal{T}$. A maximization problem MP is given as follows:

$$\begin{array}{ll} \textbf{maximize} & f_{obj}(V_1,\dots,V_n) \\ \textbf{subject to} & T_1\Omega_{1_1}f_{1_1}(V_1)\Omega_{1_2}\dots\Omega_{1_n}f_{1_n}(V_n) = \varnothing \\ & \dots \\ & T_m\Omega_{m}f_{m_1}(V_1)\Omega_{m_2}\dots\Omega_{m_n}f_{m_n}(V_n) = \varnothing \end{array}$$

where $T_i \subseteq \mathcal{T}$, f_{i_j} is a map from $\mathcal{T}$ to $2^{\mathcal{T}}$, and $\Omega_{i_j} \in \{\cap, \cup, -\}$ for all $1 \le i \le m$, $1 \le j \le n$. Intuitively, the expressions on the left of the equalities above are unions/intersections/differences of terms denoting subsets of $\mathcal{T}$.

A mapping M: $\{V_1,\dots,V_n\} \to \mathcal{T}$ is said to be a maximal solution of MP iff: (1) the assignment of $M(V_i)$ to variable V_i $(1 \le i \le n)$ satisfies the constraints, and (2) for all other mappings M' that satisfy the constraints, the inequality $f_{obj}(M(V_1),\dots,M(V_n)) \not< f_{obj}(M'(V_1),\dots,M'(V_n))$ holds w.r.t. the given lattice ordering.

Example 5.2. Consider the truth value lattice FOUR, and suppose we wish to solve the maximization problem

$$\begin{array}{ll} \textbf{maximize} & V_1 \sqcup V_2 \\ \textbf{subject to} & \{\bot, \mathbf{f}\} \cup (\Uparrow V_1) \cap (\Uparrow V_2) = \varnothing. \end{array}$$

Then $V_1 = V_2 = \top$, $V_1 = \top$, $V_2 = \mathbf{t}$, and $V_2 = \top$ are all maximal solutions to the above problem. However, the solution $V_1 = \bot$, $V_2 = \mathbf{t}$ does *not* maximize $V_1 \sqcup V_2$; hence, it is not a maximal solution. □

When dealing with lattices, it is possible to have more than one maximal solution to a maximization problem. For example, the problem: **maximize** V **subject to** $\{V\} \cap \{\top\} = \varnothing$ has two maximal solutions: $V = \mathbf{t}$ and $V = \mathbf{f}$. It turns out that the maximization problems that arise as a result of successive S-resolutions have a special form. We will show that maximization problems generated during the course always have a unique solution.

As an example, consider the query $Q^* \equiv A$: $[D, \mathcal{T} - \Uparrow V_1] \leftarrow$. As has been illustrated in Example 5.1, when processing this query by performing successive S-resolutions, the atom A (when it occurs in successive resolvents in an S-deduction) will always have an annotation of the form

$$(\mathcal{T} - \Uparrow V_1) \cap (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n)$$

where $n \geq 1$. When attempting to evaluate the "current best" known truth value for A, we need to maximize the value of V_1 subject to the constraint

$$(\mathcal{T} - \Uparrow V_1) \cap (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n) = \varnothing.$$

This is because V_1 occurs in the query Q^*, and we wish to obtain maximal possible values of V_1. Theorem 5.1 below shows that there is a unique maximal solution to this problem, and it is obtained by setting $V_1 = V_2 \sqcup \cdots \sqcup V_n$. Prior to proving Theorem 5.1, we need to prove an elementary result.

Lemma 5.1. If $V_1 = V_2 \sqcup \cdots \sqcup V_n$, *then* $\Uparrow V_1 = (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n)$.

PROOF.

- Since $V_i \leq V_1$ $(2 \leq i \leq n)$, $V_1 \in (\Uparrow V_i)$. Hence, for all $V_1 \leq V'$, $V' \in (\Uparrow V_i)$ and $(\Uparrow V_1) \subseteq ((\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n))$.
- Let $V_s = (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n)$. For all $V' \in V_s$, we have that $V_i \leq V'$ $(2 \leq i \leq n)$. Since $V_1 = V_2 \sqcup \cdots \sqcup V_n$, it must be the case that $V_1 \leq V'$. Hence, $V \in \Uparrow V_1$ and $((\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n) \subseteq \Uparrow V_1$. □

Theorem 5.1. For any maximization problem MP given as follows:

maximize V_1
subject to $(\mathcal{T} - \Uparrow V_1) \cap (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n) = \varnothing$

where all the V_i, $1 \leq i \leq n$ *are annotation terms, the maximal solution is*: $V_1 = V_2 \sqcup \cdots \sqcup V_n$.

PROOF. The theorem will be proved by induction on the number, n, of annotation variables.

Basis. The problem MP_1 is given as follows:

maximize V_1
subject to $(\mathcal{T} - \Uparrow V_1) = \varnothing$.

Then the maximal solution to MP_1 is $V_1 = \bot$.

- $\sqcup\{\ \} = \bot$; therefore $V_1 = \bot$ is the solution given in the theorem.
- Since $\Uparrow V_1 = \mathcal{T}$, $(\mathcal{T} - \Uparrow V_1) = \varnothing$, and hence $V_1 = \bot$ is a solution to the constraint given in MP_1.
- There is no solution V_1' such that $\bot \leq V_1'$. Since that implies $\bot \in (\mathcal{T} - \Uparrow V_1')$, V_1' does not satisfy the constraint.

Inductive Step. For all $i < n$, let the solution to the problem MP_i,

maximize V_1
subject to $(\mathcal{T} - \Uparrow V_1) \cap \cdots \cap (\Uparrow V_i) = \varnothing$

be given as $V_1 = V_2 \sqcup \cdots \sqcup V_i$. Let the problem MP_n be

maximize V_1
subject to $(\mathcal{T} - \Uparrow V_1) \cap \cdots \cap (\Uparrow V_n) = \varnothing$.

Then the solution to MP_n is $V_1 = V_2 \sqcup \cdots \sqcup V_n$.

- Let $\alpha = V_2 \sqcup \cdots \sqcup V_{i-1}$ and $\beta = \alpha \sqcup V_i$. By the inductive hypothesis, α is a solution to MP_{i-1}. By Lemma 5.1, it is true that

  $$\Uparrow \alpha = (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_{i-1})$$
  $$(\Uparrow \alpha) \cap (\Uparrow V_i) = (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_{i-1}) \cap (\Uparrow V_i).$$

  By Lemma 5.1, $\Uparrow(\alpha \sqcup V_i) = (\Uparrow \alpha) \cap (\Uparrow V_i) = \Uparrow \beta$. Then

  $$(\mathcal{T} - \Uparrow \beta) \cap (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_i) = \emptyset$$

  and β is a solution to MP_i.
- β is the only solution, since for all $V' \not\leq \beta$, it is true that $\beta \notin \Uparrow V'$ and $\beta \in (\mathcal{T} - \Uparrow V')$. By the argument above, we know that

  $$\Uparrow \beta = (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_i)$$
  $$\beta \in ((\Uparrow V_2) \cap \cdots \cap (\Uparrow V_i))$$
  $$\beta \in [(\mathcal{T} - \Uparrow V') \cap ((\Uparrow V_2) \cap \cdots \cap (\Uparrow V_i))] \neq \emptyset.$$

  Hence, V' does not satisfy the constraints for MP_i, and cannot be a solution. □

Example 5.3. Consider the maximization problem

maximize V
subject to $(\mathcal{T} - V) \cap \Uparrow V_1 \cap \cdots \cap \Uparrow V_{n-1} = \emptyset$.

The solution to this problem is $V_{old} = V = V_1 \sqcup \cdots \sqcup V_{n-1}$. Now, suppose the term $\Uparrow V_n$ is added to the constraint. Then the new maximum value of V is $V = V_{old} \sqcup V_n$. In other words, having calculated V_{old} once, we can use it to solve larger problems maximizing the same variable. For instance, in the case of Example 5.1, we had calculated the maximal truth value of V to be **t** (in the second step). In step 4, we introduce the term $\Uparrow \mathbf{f}$ into the constraint. Then the new maximal value of became $V = \mathbf{t} \sqcup \mathbf{f} = \top$. Therefore, we can conclude that $V = \top$ without solving the maximization problem from scratch. □

When using the above theorem to compute the maximal value of V_1 subject to the constraint that

$$(\mathcal{T} - \Uparrow V_1) \cap (\Uparrow V_2) \cap \cdots \cap (\Uparrow V_n) = \emptyset$$

we need to address how the maximal value of V_1 changes when the value of one of the V_is changes. The following theorem shows how this may be easily computed.

Theorem 5.2. Let MP_n be the maximization problem given in Theorem 5.1, *and let $V_1 = \alpha = V_2 \sqcup \cdots \sqcup V_n$ be the maximum solution. The problem MP'_n is defined by replacing V_i by V'_i for some $2 \leq i \leq n$ where $V_i \leq V'_i$. The maximal solution to MP'_n is $V_1 = \alpha \sqcup V'_i$.*

PROOF. Since $\Uparrow V_i \cap \Uparrow V_i' = \Uparrow V_i'$, and by Lemma 5.1, $\Uparrow(\alpha \sqcup V_i') = \Uparrow \alpha \cap \Uparrow V_i'$, then

$$\begin{aligned}\Uparrow \alpha &= \Uparrow V_2 \cap \cdots \cap \Uparrow V_n\\ \Uparrow \alpha \cap \Uparrow V_i' &= \Uparrow V_2 \cap \cdots \cap \Uparrow V_i' \cap \cdots \cap \Uparrow V_n\\ (\mathscr{T} - \Uparrow(\alpha \sqcup V_i')) \cap \Uparrow V_2 \cap \cdots \cap \Uparrow V_i' \cap \cdots \cap \Uparrow V_n &= \emptyset.\end{aligned}$$

Hence, $V_1 = \alpha \sqcup V_i'$ satisfies the constraint given in MP_n', and it is the maximum such value as a result of the second equality above. □

We will now start defining a mathematical description of the data structures needed for an OLDT type proof processing procedure. First of all, a table is needed for catching information obtained in the intermediate levels of resolution. Just as [33] stores sets of atoms in the table, the table in our well framework will store a set of *annotated* atoms. This leads to two key distinctions behind our framework and that of Sato and Tamaki's [33]:

- As the atoms being inserted are annotated atoms, the insertion of new annotated atoms to the table and checking if an atom is true in the table are significantly more complicated operations compared to the simple case in [33]. This will necessitate the development of three new suboperations called *revision*, *merging*, and *simplification*. In the next section, we will define these operations in detail.
- In addition, in our framework, whenever a new atom is inserted into the table, there may be a need to (implicitly or explicitly) solve a maximization problem. This is not true in the framework of [33].

5.2. MULTI_OLDT *Table*

Kifer and Subrahmanian [18]l have defined how substitutions (in the ordinary sense; cf. Lloyd [22]) may be extended to apply to annotated atoms. The only difference is that, now, substitutions may assign terms to annotation variables, and these terms must range over the appropriate truth value lattice. Application of substitutions to annotated atoms may then be defined in the obvious way. For instance, when the truth value domain is the unit interval [0, 1], the substitution

$$\sigma = \{X = a, Y = f(Z, a), U = 0.25\}$$

when applied to the annotated atom $p(X, Y, X) : [\{1\}, ((U + 1)/2)]$ yields the annotated atom $p(a, f(Z, a), a) : [\{1\}, ((0.25 + 1)/2)]$; at this stage, we will assume that the annotation term $((0.25 + 1)/2)$ is evaluated to yield the annotated atom $p(a, f(Z, a), A) : [\{1\}, 0.625]$. *Throughout the rest of this paper, whenever we use the word "substitution," we will mean a substitution in he extended sense defined above.*

Definition 5.2. A MULTI_OLDT-table is a set of annotated atoms of the form A: $[D, \mu]$.

We will now specify how the MULTI_OLDT-table gets updated when a new atom is inserted. When such an insertion occurs, we would like the table to draw all possible conclusions (about the truth values of atoms) based on the insertion being made. For example, suppose the table contains the atom $p(X, Y) : [\{1\}, \mathbf{t}]$, and we insert the atom $p(a, Y) : [\{1\}, \mathbf{f}]$. Now, the atom $p(a, Y) : [\{1\}, \top]$ is a logical consequence of these two atoms. We have two options. In the first option, we may insert the first atom as it is. In this case, to solve the query $p(a,Y):[\{1\},\top]\leftarrow$, we will need to perform two resolutions with the table. In the second option, we can calculate the logical consequences of the new atom and insert them as well. In this case, we want to ensure that we have computed all the necessary logical consequences. Note that there are two kinds of logical consequences. (1) The first is those that change the atom being inserted as in the case above. For example, we now need to insert $p(a,Y):[\{1\},\top]$ instead of $p(a,Y):[\{1\},\mathbf{f}]$. Atoms containing a $\mathscr{D}$*-term* smaller than the $\mathscr{D}$*-term* of the atom being inserted should be considered for this case. (2) The second is the logical consequences that change the atoms already stored in the table. Suppose, now, that we have the atom $p(Z,f(Z))$: $[\{1,2\},\mathbf{t}]$ in the table. In this case, the atom $p(a,f(a)):[\{1,2\},\top]$ is the logical consequence of this atom and the new atom, i.e., $p(a,Y):[\{1\},\top]$. Observe that if we did not calculate case (1) before case (2), we could not find this logical consequence. Hence, we will separate these two cases and call (1) the *revision step* and (2) the *merge step*. After all of these new atoms are inserted into the table, we have to remove the redundant atoms. We will call this step the *simplification step*. For example, atom $p(a,Y):[\{1\},\top]$ subsumes the atom $p(a,f(a)):[\{1,2\},\top]$. In this case, we want to remove the latter from the table.

Definition 5.3. (Revision Step) Suppose Γ is a MULTI_OLDT-table and A_1: $[D_1,\mu_1]$ is an annotated atom. Given any set X of annotated atoms of the form A: $[D,\mu]$ where D is ground, we use $X^{[i]}$ to denote the set $\{A$: $[D,\mu] \| A$: $[D,\mu]\in X \& card(D)=i\}$ of all annotated atoms in set X whose D component has cardinality i.

The revision $\mathbf{R}$ of the atom A_i: $[D_1,\mu_1]$ with table Γ is given as follows:

- $\mathbf{R}^0=\{A_1$: $[D_1,\mu_1]\}$.
- $\mathbf{R}^{i+1}=\mathbf{R}^i\cup\{A'_1\sigma$: $[D_1,\sqcup(\mu'_1\sigma,\mu_2\sigma]\}|A'_1$: $[D_1,\mu'_1]\in\mathbf{R}^i$ and A_2: $[D_2,\mu_2]\in\Gamma^{[i+1]}$ and $A'_1:\mu'_1$ and $A_2:\mu_2$ are unifiable via mgu σ and $D_2\subseteq D_1\}$.
- $\mathbf{R}=\mathbf{R}^{card(D_1)}$.

5.2.1. The Revision Step. Intuitively, the revision step finds all of the atoms in a table that contain information relevant to the new atom, and updates the "maximal" truth value that may be associated with the new atom. This process starts by comparing the new atoms with the atoms having a singleton $\mathscr{D}$-term. Then it is compared with atoms with $\mathscr{D}$-terms of cardinality $2,3,\ldots$ until the cardinality of the current $\mathscr{D}$-term is reached. Since there cannot be a $\mathscr{D}$-term which is a subset of the current $\mathscr{D}$-term after this point, the execution stops. To ensure that all of the logical consequences are computed in an efficient way, we consider the atoms in the table in the increasing order of their $\mathscr{D}$-terms.

Example 5.4. Suppose Γ is as given below:

$$\Gamma=\{p(X,c):[\{1\},\mathbf{t}],p(f(Y),Y):[\{2\},\mathbf{f}],p(a,Y):[\{2\},\mathbf{t}],$$
$$p(a,Y):[\{1,2,3\},\mathbf{f}],p(f(Y),Y):[\{1,2,3\},\mathbf{t}]\}.$$

Then the revision of $p(U,b):[\{1,2\},\mathbf{f}]$ with the table Γ is the set $\mathbf{R}$:

$$\mathbf{R}=\{p(U,b):[\{1,2\},\mathbf{f}],p(f(b),b):[\{1,2\},\mathbf{f}],p(a,b):[\{1,2\},\top]\}.$$ □

Definition 5.4. (Merging Step) Suppose the set **R** contains a set of atoms that are to be inserted into a MULTI_OLDT-table Γ. Then the merge of **R** with Γ is the set $\mathbf{M} = \{A_2\sigma: [D_2, \sqcup(\mu_1\sigma, \mu_2\sigma)] \| A_1: [D_1, \mu_1] \in \mathbf{R}$ is unifiable with $A_2: [D_2, \mu_2] \in \Gamma$ via mgu σ and $D_1 \subset D_2\}$.

5.2.2. *The Merging Step.* The basic intuition (in the case when annotation variables are ground) behind merging is the following: when inserting an atom A_1: $[D_1, \mu] \in \mathbf{R}$ into the MULTI_OLDT-table Γ, we examine all atoms A_2: $[D_2, \mu_2] \in \Gamma$ such that $D_1 \subset D_2$ and such that A_1 and A_2 are unifiable via mgu σ—the insertion of A_1: $[D_1, \mu]$ may cause the truth value of A_2: $[D_2, \mu_2]$ to "increase" from μ_2 to $\sqcup(\mu_1, \mu_2)$. The above definition uses this intuition to define merging when annotation variables may be nonground. The following example illustrates how merging works.

Example 5.5. Consider the table Γ given in the example above. Let **R** be given by

$$\mathbf{R} = \{p(U,b): [\{1,2\},\mathbf{f}], p(f(b),b): [\{1,2\},\mathbf{f}], p(a,b): [\{1,2\}, \top]\}.$$

Since $\{1,2\} \subset \{1,2,3\}$, only the atoms $p(a,Y)$: $[\{1,2,3\},\mathbf{f}]$ and $p(f(Y),Y)$: $[\{1,2,3\},\mathbf{t}]$ in Γ will be considered for merging. The merge of **R** and Γ is the set

$$\mathbf{M} = \{p(a,b): [\{1,2,3\},\mathbf{f}], p(a,b): [\{1,2,3\}, \top], p(f(b),b): [\{1,2,3\}, \top]\}.$$

□

Definition 5.5. (Simplification Step) Suppose Γ is a MULTI_OLDT-table. Then a simplified version of Γ is a table Γ' where Γ' is a minimal subset of Γ such that, for all atoms A: $[D, \mu] \in (\Gamma - \Gamma')$, there exists an atom A': $[D', \mu'] \in \Gamma'$ such that A': $[D', \mu'] \models A$: $[D, \mu]$.

5.2.3. *The Simplification Step.* Note that given a MULTI_OLDT-table Γ, there may be many tables Γ' which are simplifications of Γ. Any of these will suffice for our purposes.

Example 5.6. Consider the sets **R**, **M**, and Γ given in Examples 5.4 and 5.5. The union of these sets is the set Γ^* given below:

$$\begin{aligned}\Gamma^* = \{&p(X,c): [\{1\},\mathbf{t}], p(f(Y),Y): [\{2\},\mathbf{f}], p(a,Y): [\{2\},\mathbf{t}],\\ &p(U,b): [\{1,2\},\mathbf{f}], p(f(b),b): [\{1,2\},\mathbf{f}], p(a,b): [\{1,2\}, \top],\\ &p(a,Y): [\{1,2,3\},\mathbf{f}], p(f(Y),Y): [\{1,2,3\},\mathbf{t}], p(a,b): [\{1,2,3\},\mathbf{f}],\\ &p(a,b): [\{1,2,3\}, \top], p(f(b),b): [\{1,2,3\}, \top]\}.\end{aligned}$$

Since $p(f(Y), Y)$: $[\{2\}, \mathbf{f}] \models p(f(b), b)$: $[\{1, 2\}, \mathbf{f}]$, $p(a, b)$: $[\{1, 2\}, \top] \models p(a,b)$: $[\{1,2,3\},\mathbf{f}]$ and $[\{1,2\}, \top] \models p(a,b)$: $[\{1,2,3\}, \top]$. Then the simplified version Γ' of the table Γ^* is given as

$$\Gamma' = \Gamma^* - \{p(f(b),b): [\{1,2\},\mathbf{f}], p(a,b): [\{1,2,3\}, \top]\}.$$

□

5.2.3. *Table Insertion.* In this section, we will show how the three operations of revision, merging, and simplification may be jointly used to update a given table.

Definition 5.6. (Table Insertion) Suppose Γ is a MULTI_OLDT-table, and A_1: $[D_1, \mu_1]$ is an annotated atom. The *result of inserting* A_1: $[D_1, \mu_1]$ *into* Γ is a new table Γ' constructed as follows:

```
1. Set $\mathbf{M}$ to $\{A_1\colon [D_1, \mu_1]\}$.
2. WHILE $\mathbf{M} \neq \emptyset$ DO
   BEGIN
   (a) Find the revision $\mathbf{R}_i$ of all the atoms $A_i\colon [D_i, \mu_i] \in \mathbf{M}$.
   (b) Set $\mathbf{R}'$ to $\bigcup_i \mathbf{R}_i$.
   (c) Set $\mathbf{R}$ to the simplified version of $\mathbf{R}'$.
   (d) Find the merge $\mathbf{M}'$ to $\mathbf{R}$ and $\Gamma$.
   (e) Set $\mathbf{M}$ to the simplified version of $\mathbf{M}'$.
   (f) Set $\Gamma$ to $\Gamma \cup \mathbf{R}$.
   END
3. Find the simplified version $\Gamma'$ of $\Gamma$; set the final table to $\Gamma'$.
```

That is, the insertion of A_1: $[D_1, \mu_1]$ into the table Γ is a two-step process (after initialization): in the first step, the atoms in the table are compared and merged with the new atom in a continuous loop. In the second step, the redundant atoms are removed. This process is guaranteed to terminate, as explained by the lemma below:

Lemma 5.2. At all times in the table insertion process, the following invariant is maintained:

"If i and i + 1 are two consecutive executions of the while loop in Definition 5.6 and $\mathbf{M}^i, \mathbf{M}^{i+1}$ *are the simplified versions of the merges obtained at the end of the ith and i + 1th executions of step 2(e), respectively, then*

- *either* $\mathbf{M}^{i+1}$ *is empty,*
- *or if* D^i_{min} *is a* $\mathscr{D}$*-term with the smallest cardinality among the* $\mathscr{D}$*-terms of the atoms in* $\mathbf{M}^i$*, then all the* $\mathscr{D}$*-terms* D^{i+1} *of the atoms in* $\mathbf{M}^{i+1}$ *satisfy the property that*

$$card(D^i_{min}) < card(D^{i+1})."$$

PROOF. Let $\mathbf{R}^i$ be the revision obtained at step 2(c), and let Γ^i be the table obtained at step 2(f) of the ith execution of the repeat loop. Since the loop is executed an $(i + 1)$th time, we know that $\mathbf{M}^i$ is not empty.

Now, observe that if A_k: $[D_k, \mu_k]$ is an atom in $\mathbf{M}^i$, then all of the atoms in the revision $\mathbf{R}_k$ of this atom with Γ^i have the same $\mathscr{D}$-term, namely, D_k. Hence, the cardinality of the $\mathscr{D}$-terms with the smallest cardinality in $\mathbf{R}'^{,i+1}$ obtained in step 2(b) is the same as that of $\mathbf{M}^i$, namely, $card(D^i_{min})$. Since the simplification step only removes atoms from the set, the same is true for $\mathbf{R}^{i+1}$, i.e., $\mathscr{D}$-terms with the smallest cardinality in $\mathbf{R}^{i+1}$ still have the cardinality $card(D^i_{min})$.

Now, consider the merge $\mathbf{M}'^{,i+1}$ of $\mathbf{R}^{i+1}$ and Γ^i. In case $\mathbf{M}'^{,i+1}$ is empty, the invariant is automatically maintained. If it is nonempty, we know from the definition of the *merging* step that for all atoms $A^{i+1}\sigma$: $D^{i+1}, \sqcup(\mu^{i+1}\sigma, \mu^k\sigma)]$ in $\mathbf{M}'^{,i+1}$, it is true that there exists an atom A^i: $[D^i, \mu^i]$ in $\mathbf{R}^{i+1}$ that is unifiable with an atom A^{i+1}: $[D^{i+1}, \mu^{i+1}]$ in Γ^i via mgu σ and such that $D^i \subset D^{i+1}$. Thus, $card(D^i) < card(D^{i+1})$. Since all the $\mathscr{D}$-terms with the smallest cardinality in $\mathbf{R}^{i+1}$

have the cardinality $card(D^i_{min})$, we have that $card(D^i_{min}) \leq card(D^i)$ and $card(D^i_{min}) < card(D^{i+1})$. This is true for all of the atoms in $\mathbf{M}'^{,i+1}$, and since the simplification step only removes atoms from $\mathbf{M}'^{,i+1}$, it is also true for all atoms in $\mathbf{M}^{i+1}$. □

Corollary 5.1. (*Termination of the Table Insertion Algorithm*) *Let D_{max} be a $\mathscr{D}$-term in Γ with the biggest cardinality. Then the insertion of an atom A: $[D, \mu]$ into Γ using the algorithm given in Definition* 6 *terminates after at most $card(D_{max})$ executions of the* **WHILE** *loop.*

PROOF. By Lemma 5.2, we know that at each execution of the **WHILE** loop in Definition 5.6, the cardinality of $\mathscr{D}$-terms with the smallest cardinality in **M** is strictly larger than that of the previous execution of the body of this loop. Moreover, we know that the $\mathscr{D}$-terms of the atoms obtained in the revision and the merge steps are either equal to the $\mathscr{D}$-term of the new atom A: $[D, \mu]$ or to the $\mathscr{D}$-term of an atom in Γ. Hence, $card(D_{max})$ remains constant. Then, if the **WHILE** loop is executed $card(D_{max}) - 1$ times, at the end of this set of iterations, all $\mathscr{D}$-terms in **M** with the smallest cardinality have cardinality $card(D_{max})$. At the $card(D_{max})$th execution of the repeat loop, the following happens: the revision step does not change the cardinality of the $\mathscr{D}$-terms in **M** since there are no atoms in Γ with $\mathscr{D}$-terms having cardinality strictly larger than $card(\mathscr{D}_{max})$, and consequently, the merge is empty. Hence, the **WHILE** loop is exited, and the algorithm terminates. □

The following examples illustrates the notion of insertion into a MULTI_OLDT-table.

Example 5.7. Suppose we consider the MULTI_OLDT-table

$$\Gamma = \{p(a,b)\colon [\{1,2\},0.5], q\colon [\{1,2\},0.7], r\colon [\{2\},0.3]\}.$$

The table that results from the insertion of $p(a, X)$: $[\{1,2\}, 0.6]$ is

$$\Gamma^* = \{p(a,X)\colon [\{1,2\},0.6], q\colon [\{1,2\},0.7], r\colon [\{2\},0.3]\}.$$

Note that the atom $p(a,b)$: $[\{1,2\}, 0.5]$ is implies by the universal closure of the atom being inserted, viz. $p(a, X)$: $[\{1,2\}, 0.6]$, and hence, $p(a,b)$: $[\{1,2\}, 0.5]$ is eliminated from the table Γ.

A slightly more complicated example is the following:

Example 5.8. Suppose we are considering the lattice FOUR, and $\Gamma = \{p\colon [\{1,2\}, \mathbf{t}]\}$, and we are inserting the atom p: $[\{1\}, \mathbf{f}]$. The merge of these atoms is p: $[\{1\}, \top]$. The table Γ before the execution of the simplification step consists of

$$\Gamma = \{p\colon [\{1,2\}, \top], p\colon [\{1,2\}, \mathbf{t}], p\colon [\{1\}, \mathbf{f}]\}.$$

A minimal subset Γ' is

$$\Gamma' = \{p\colon [\{1,2\}, \top], p\colon [\{1\}, \mathbf{f}]\}.$$

The following example illustrates the execution of the revision and merge steps of the table insertion routine.

Example 5.9. Let us consider the lattice 2^{N} of time points. An atom of the form p: $[\{1,2\},\{t_3,t_4\}]$ in this lattice can be read as: "p is true at time points 3 and 4 according to databases 1 and 2 jointly." Now, suppose the table in this example contains the atoms

$$\Gamma=\{p\colon [\{1\},\{t_1,t_3\}], p\colon [\{2\},\{t_1,t_2\}], p\colon [\{3\},\{t_7\}], p\colon [\{1,2,3\},\{t_6\}],\\ p\colon [\{1,2,3,4\},\{t_4,t_5\}]\}$$

and the atom p: $[\{1,2\},\{t_3\}]$ is being inserted into Γ. The following operations take place:

- *Step* 1. $\mathbf{M}$ is set to $\{p$: $[\{1,2\},\{t_3\}]\}$.
- *Step* 2$(a$–$b)$. The atom p: $[\{1,2\},\{t_3\}]$ is revised according to atoms p: $[\{1\},\{t_1,t_3\}]$ and p: $[\{2\},\{t_1,t_2]\}$ in Γ to give p: $[\{1,2\}, \sqcup(\{t_3\},\{t_1,t_3\},\{t_1,t_2\})] \equiv p$: $[\{1,2\},\{t_1,t_2,t_3\}]$. $\mathbf{R}$ is set to $\{p$: $[\{1,2\},\{t_1,t_2,t_3\}]\}$.
- *Step* 2$(c$–$d)$. $\mathbf{M}$ is set to $\{p$: $[\{1,2,3\},\{t_1,t_2,t_3,t_6\}]$, p: $[\{1,2,3,4\},\{t_1,t_2,t_3,t_5\}]\}$. Γ is set to $\Gamma \cup \mathbf{R}$.
- *Step* 2$(a$–$b)$ $\mathbf{R}$ is set to the revision of all atoms in $\mathbf{M}$, i.e., $\mathbf{R}=\{p$: $[\{1,2,3\},\{t_1,t_2,t_3,t_6,t_7\}]$, p: $[\{1,2,3,4\},\{t_1,t_2,t_3,t_5,t_6,t_7\}]\}$.
- *Step* 2$(c$–$d)$. $\mathbf{M}$ is set to $\{p$: $[\{1,2,3,4\},\{t_1,t_2,t_3,t_5,t_6,t_7\}]\}$. Γ is set to $\Gamma \cup \mathbf{R}$.
- *Step* 2$(a$–$d)$. $\mathbf{R}$ is set to $\{p$: $[\{1,2,3,4\},\{t_1,t_2,t_3,t_5,t_6,t_7\}]\}$ and $\mathbf{M}$ is set to the empty set. Γ is set to $\Gamma \cup \mathbf{R}$
- *Step* 3. The table before simplification contains the atoms

  $$\Gamma=\{p\colon [\{1\},\{t_1,t_3\}], p\colon [\{2\},\{t_1,t_2\}], p\colon [\{3\},\{t_7\}], p\colon [\{1,2,3\},\{t_6\}],\\ p\colon [\{1,2,3,4\},\{t_5\}], p\colon [\{1,2\},\{t_1,t_2,t_3\}],\\ p\colon[\{1,2,3\},\{t_1,t_2,t_3,t_6,t_7\}], p\colon [\{1,2,3,4\},\{t_1,t_2,t_3,t_5,t_6,t_7\}]\}.$$

  This table is simplified to give the final table:

  $$\Gamma'=\{p\colon [\{1\},\{t_1,t_3\}], p\colon[\{2\},\{t_1,t_2\}], p\colon [\{3\},\{t_7\}],\\ p\colon [\{1,2\},\{t_1,t_2,t_3\}], p\colon [\{1,2,3\},\{t_1,t_2,t_3,t_6,t_7\}],\\ p\colon [\{1,2,3,4\},\{t_1,t_2,t_3,t_5,t_6,t_7\}]\}.$$ □

Lemma 5.3. (Soundness of Table Insertion) Suppose Γ *is a* MULTI_OLDT*-table,* A: $[D,\mu]$ *an annotated atom and* I *an* $\mathbf{A}$*-interpretation. Let* Γ' *be the table obtained by inserting* A: $[D,\mu]$ *into* Γ. *Then* I $\mathbf{A}$*-satisfies all the atoms in* Γ' *iff* I $\mathbf{A}$*-satisfies* A: $[D,\mu]$ *and all the atoms of* Γ.

PROOF. Suppose I is an $\mathbf{A}$-interpretation that $\mathbf{A}$-satisfies A: $[D,\mu]$ and all the atoms in Γ. If I $\mathbf{A}$-satisfies all the atoms introduced in the revision and merging steps, then I $\mathbf{A}$-satisfies all the atoms in Γ' (the simplification step only reduces the size of the table). Assume A': $[D',\mu']$ is an atom in Γ that is unifiable with A: $[D,\mu]$ via some substitution σ and such that $D' \subseteq D$. Then the atom $A\sigma$: $[D, \sqcup(\mu\sigma,\mu'\sigma)]$ is added to the table. Clearly, I $\mathbf{A}$-satisfies both $A\sigma$: $[D,\mu\sigma]$ and $A'\sigma$: $[D',\mu'\sigma]$. By the definition of $\mathbf{A}$-satisfaction, $\mu\sigma \leq \sqcup_{i\in D} I(A\sigma)(i)$ and $\mu'\sigma \leq \sqcup_{i\in D'} I(A'\sigma)(i)$. But $D' \subseteq D$; hence, $\sqcup(\mu\sigma,\mu'\sigma) \leq \sqcup_{i\in D} I(A\sigma)(i)$, and I $\mathbf{A}$-satisfies $A\sigma$: $[D, \sqcup(\mu\sigma,\mu',\sigma)]$. □

It follows from the above lemma that if A: $[D, \mu]$ is true in the table Γ at a given point in time during the computation of a query, this annotated atom will continue to be true at all times in the future—the main difference is that A: $[D, \mu']$ may also be known to be true where $\mu \leq \mu'$. In other words, the set of consequences of the table is growing monotonically as more time is spent processing a query.

5.3. Complexity of Table Insertion

Complexity of Revision. Suppose $\Gamma^{[i]} = \{A\text{: } [D, \mu] \| A\text{: } [D, \mu] \in \Gamma \text{ and } card(D) = i\}$. Then the worst case time complexity of computing $\mathbf{R}^{i+1}$ from $\mathbf{R}^{i}$ is $O(card(\mathbf{R}^i)card(\Gamma^{[i+1]})l)$ where l is the cost of checking whether two annotated atoms are unifiable and checking whether $D_2 \subseteq D_1$. An unification is a well-known linear time problem (cf. Martelli and Montanari [25]), it follows immediately that l is linear in the number of symbols in the atoms.

It is easy to see that $card(\mathbf{R}^{i+1}) \leq card(\mathbf{R}^i) + card(\Gamma^{[i+1]})$. Thus, the total cost of the revision step for an atom A_j: $[D_j, \mu_j]$ with Γ where $card(D_j) = d$ is given by

$$C_{\mathbf{R}} \leq \sum_{i=1}^{d} \left(card(\Gamma^{[i]})l \left(1 + \sum_{k=1}^{i-1} card(\Gamma^{[k]}) \right) \right).$$

Assuming that $card(\Gamma^{[i]}) = \alpha$ for all i, the above upper bound on $C_{\mathbf{R}}$ reduces to $C_{\mathbf{R}} \leq (\alpha^2 dl(d+1)/2) - (\alpha^2 - \alpha)dl$. Hence, the worst case complexity of revision is $O(\alpha^2 d^2 l)$ and $card(\mathbf{R}) \leq d\alpha + 1$. *In short, revision is a polynomial-time operation.*

Complexity of Merging. Suppose Γ is a MULTI_OLDT-table and $\Gamma^{[i]} = \{A\text{: } [D, \mu] \| A\text{: } [D, \mu] \in \Gamma \,\&\, card(D) = i\}$. Suppose there are n deductive databases in the amalgamated system. Then the time complexity of merging Γ with a set $\mathbf{R}$ is given by

$$C_{\mathbf{M}} = \sum_{i=d}^{n} card(\Gamma^{[i]})card(\mathbf{R})l.$$

Assuming again that card$(\Gamma^{[i]}) = \alpha$, $C_{\mathbf{M}} \leq card(\mathbf{R})l\alpha(n - d + 1)$ and $card(\mathbf{M}) \leq \alpha(n - d + 1)$. *Thus, the complexity of merging is polynomial-time.*

We now come to the third and final step that is used in defining the insertion of an annotated atom A: $[D, \mu]$ into a MULTI_OLDT-table. This step is called *simplification.* The basic idea in simplification is that "redundant" atoms in a table should be eliminated..

Complexity of Simplification. In the worst case, the simplified version of a set $\mathbf{M}$ of atoms may be computed in $O(card(\mathbf{M})^2 l)$. The reason for this is the following: consider the ordering $\preccurlyeq$ on $\mathbf{M}$ defined as follows: A_1: $[D_1, \mu_1] \preccurlyeq A_2$: $[D_2, \mu_2]$ iff A_2: $[D_2, \mu_2] \models A_1$: $[D_1, \mu_1]$. $\preccurlyeq$ is a reflexive and transitive ordering on $\mathbf{M}$, and hence induces an equivalence relation $\sim$ on $\mathbf{M}$ defined as: A_1: $[D_1, \mu_1] \sim A_2$: $[D_2, \mu_2]$ iff A_1: $[D_1, \mu_1] \preccurlyeq A_2$: $[D_2, \mu_2]$ and A_2: $[D_2, \mu_2] \preccurlyeq A_1$: $[D_1, \mu_1]$. The $\preccurlyeq$ relation can now be extended to the equivalence classes generated by $\sim$ as follows: $[A_1$: $[D_1, \mu_1] \preccurlyeq {}^*[A_2$: $[D_2, \mu_2]$ iff A_1: $[D_1, \mu_1] \preccurlyeq A_2$: $[D_2, \mu_2]$. $\preccurlyeq^*$ is a partial ordering on equivalence classes. The simplification step corresponds to finding the $\preccurlyeq^*$-maximal equivalence classes, and then choosing exactly one member from each of these $\preccurlyeq^*$-maximal equivalence classes.

The step of computing whether A_1: $[D_1, \mu_1] \preccurlyeq A_2$: $[D_2, \mu_2]$ is a linear time operation as it only involves checking whether there exists a substitution σ such that: (1) $A_2\sigma = A_1$, and (2) $D_2 \subseteq D_1$ and (3) $\mu_1\sigma \leq \mu_2\sigma$. Computing equivalence relations can be performed in time that is quadratic in the number of annotated atoms in **M** (cf. Knuth [19, Alg. E, p. 354]). Finding the $\preccurlyeq^*$ maximal elements of the $\sim$-equivalence classes can be done in linear time using standard topological sorting (cf. Knuth [19, pp. 258–265]). *In short, the complexity of simplification is quadratic.*

In the worst case, the cardinality of the simplified version of a set is the same as the cardinality of the original set.

Complexity of Table Insertion Algorithm. The table insertion procedure (Definition 5.6) is a polynomial-time procedure. To see this, we observe that the loop in the table insertion procedure can be executed at most n times, where n is the total number of deductive databases being integrated. Each iteration of the loop takes polynomial-time as the steps of revision, merging, and simplification are all polynomial-time operations. Hence, the overall complexity of table insertion is polynomial-time.

Improving the Efficiency of Table Insertion. The running time of the table insertion algorithm given in Definition 5.6 can be reduced if certain assumptions are made about the MULTI_OLDT-table. Consider MULTI_OLDT-table Γ that satisfy the following two conditions at all times:

- (Complete information) Whenever there are two atoms A_1: $[D_1, \mu_1]$ and A_2: $[D_2, \mu_2]$ in the table that are unifiable via mgu σ and such that $D_1 \subseteq D_2$, then there must be an atom in Γ that subsumes the atom $A_2\sigma$: $[D_2, \sqcup(\mu_1\sigma, \mu_2\sigma)]$.
- (No redundant information) At all times, the simplified version of the table is the same as the original table.

These conditions will be referred to as the *compactness* conditions.

Furthermore, suppose the table is organized in such a way that all the atoms A: $[D, \mu]$ having the same predicate symbol are stored consecutively in nondecreasing order of the cardinality of their $\mathscr{D}$-terms. In other words, the atoms with singleton sets as $\mathscr{D}$-terms come first, then the atoms having D-terms with two elements, and so on. For instance, the table Γ of Example 5.4 can be stored in the order shown below:

$$\Gamma = \{p(X,c):[\{1\},\mathbf{t}], p(f(Y),Y):[\{2\},\mathbf{f}], p(a,Y):[\{2\},\mathbf{t}],$$
$$p(a,Y):[\{1,2,3\},\mathbf{f}], p(f(Y),y):[\{1,2,3\},\mathbf{t}]\}.$$

However, storing it in the order

$$\Gamma = \{p(f(Y),Y):[\{1,2,3\},\mathbf{t}], p(f(Y),Y):[\{2\},\mathbf{f}], p(a,Y):[\{2\},\mathbf{t}],$$
$$p(a,Y):[\{1,2,3\},\mathbf{f}], p(X,c):[\{1\},\mathbf{t}]\}$$

is not permitted.

Given that the tables satisfies the above conditions, the insertion routine for inserting the atom A_1: $[D_1, \mu_1]$ into the table Γ can be modified as follows:

1. Set **R** to the simplified version of the revision of A_1: $[D_1, \mu_1]$ with Γ.
2. Set Γ for $\Gamma \cup$ **R**.

```
3. FOR $i = card(D_1)$ TO $card$(largest $\mathscr{D}$-term) DO begin
   (a) Find the set $\Gamma^{[i]} = \{A_j: [D_j, \mu_j] \mid A_j: [D_j, \mu_j] \in \Gamma \,\&\, card(D_j) = i\}$.
   (b) Find the simplified version $\mathbf{M}$ of the merge of $\mathbf{R}$ and $\Gamma^{[i]}$.
   (c) Set $\Gamma$ to $\Gamma \cup \mathbf{M}$.
   (d) Set $\mathbf{R}$ to $\mathbf{R} \cup \mathbf{M}$.
   end
4. Find the simplified version $\Gamma'$ of $\Gamma$; set the final table to $\Gamma'$.
```

The difference between this algorithm and the original insertion algorithm is that this algorithm does not perform the revision operation in each iteration of the loop —instead, it is performed only once (viz. in step 1 above). The set **R** stores the set of new atoms, i.e., atoms that were produced as a result of revision or merge steps. Unlike the pervious algorithm, the merge operation is performed with the set **R** of new atoms and the atoms in nondecreasing order of the cardinality of their $\mathscr{D}$-terms. As a result, every atom in the original table will be processed only once. In other words, if the size of the table storing atoms with the same predicate symbol as A is κ, then the for loop is executed at most κ times.

The running time of the simplification step can be further reduced if special data structures are used to store the atoms in the MULTI_OLDT-table. One such arrangement is that atoms having the same $\mathscr{D}$-terms are arranged according to a secondary key. In other words, if A_1: $[D, \mu_1]$ subsumes A_2: $[D, \mu_2]$, then A_1: $[D, \mu_1]$ comes before A_2: $[D, \mu_2]$ in the table. Moreover, A_2: $[D, \mu_1]$ contains links that can be traversed to reach A_2: $[D, \mu_2]$ and all the other atoms that are subsumed by A_1: $[D, \mu_1]$. One advantage of such a data structure is that whenever it is determined that the atom being inserted subsumes an atom B already in the table, then all of the atoms that are subsumed by B can be removed without processing the entire list of such atoms by simply dereferencing a pointer. More details about the actual data structures will be given later.

5.4. *Dynamic* MULTI_OLDT-*Computation*

In this section, we will show how deductions may be constructed using the MULTI_OLDT-table.

Definition 5.7. Suppose Γ is a MULTI_OLDT-table, and let W be the expression

$$B_1: \left[D_{q_1}, \mu_{q_{s_1}}\right] \vee \cdots \vee B_m: \left[D_{q_m}, \mu_{q_{s_m}}\right] \leftarrow$$

where $[D_{q_i}, \mu_{q_{s_i}}]$, $1 \leq i \leq m$, are in set expansion form. (Note that every query has a regular representation of this form.) Then a MULTI_OLDT-*child of W w.r.t.* Γ is:

1. any S-resolvent of W with (the regular representation of) a clause in the amalgamated knowledge base P, or
2. any S-resolvent of W with (the regular representation of) an annotated atom in Γ.

Note that the above definition only specifies how queries make use of the MULTI_OLDT-tables. We will now specify how MULTI_OLDT-tables get "built up" as queries are being processed. The process of solving queries and caching intermediate answers in an intermixed fashion will be called a *dynamic* MULTI_OLDT*-computation.*

Definition 5.8. (Dynamic MULTI_OLDT Computation) Given a query Q and an amalgamated knowledge base P, a *dynamic* MULTI_OLDT *computation* associated with Q, denoted $\mathrm{DYN}_P(Q)$, is a sequence of *distinct*[4] queries $Q_1^*, Q_2^*, \ldots, Q_n^*$ and a sequence of (not necessarily distinct) tables $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ where:

1. Q_1^* is the regular representation of the original query Q.
2. $\Gamma_1 = \emptyset$.
3. Q_{i+1}^* is a MULTI_OLDT-child of Q_j for some $j \leq i$.
4. $\Gamma_{i+1} = \Gamma_i$ if Q_{i+1}^* is a MULTI_OLDT-child of Q_i^* w.r.t. the table Γ_i using condition 2 of Definition 5.7.
5. If Q_{i+1}^* is a MULTI_OLDT-child of Q_i^* w.r.t. the table Γ_i using condition 1 of Definition 5.7, then Γ_{i+1} is obtained from Γ_i as follows:
   (a) If the clause in P with which Q_i^* S-resolves has one or more atoms in its body, then $\Gamma_{i+1} = \Gamma_i$.
   (b) Otherwise, Γ_{i+1} is obtained as follows:
      i. Let T_1 be the table obtained by inserting (into table Γ_i^*) the annotated atom A_i: $[D_i, \mu_i]\theta$ where A_i: $[D_i, \mu_i]$ is the head of the clause in P that participated in the S-resolution step that generated Q_{i+1}^* and θ is the unifying substitution used in performing that resolution.
      ii. Consider, now, the parent, P, of the atom A_j: $[D_{s_j}, \mu_{s_j}]$ occurring in Q_i^*. If there exists a substitution σ such that all of the children B: $[D^*, \mu_s^*]$ of the parent are true in T_1 via substitution σ (i.e., there exists an atom B': $[D', \mu']$ in T_1 such that $B\sigma$ is an instance of $B', D' \subseteq D^*$ and $\mu_s^* \sigma \cap \Uparrow \mu\sigma = \emptyset$), then insert $P_t \sigma$ into the table T_1 where P_t is the twin of P. Repeat Step 5(b)ii until either no such substitution exists, or until no parent exists.

   The final result of this construction is the table Γ_{i+1}.

Before we give examples of dynamic MULTI_OLDT-computations, we remind the reader of the concept of the "twin" of an atom. Suppose we are processing the query $\leftarrow q$: $[\{1,2\}, \mathbf{t}]$, and we resolve (the regular representation of) this query with the (the regular representation of) the clause q: $[\{1\}, \mathbf{t}] \leftarrow$ *Body*. In this case, the "twin of the (regular representation of) atom q: $[\{1,2\}, \mathbf{t}]$ will be q: $[\{1\}, \mathbf{t}]$. Now, suppose that later we solve *Body*, and this allows us to conclude that the query is solved, and hence q: $[\{1,2\}, \mathbf{t}]$ is a logical consequence of the program. In fact, we know a stronger fact, viz. that q: $[\{1\}, \mathbf{t}]$ is a logical consequence. Since q: $[\{1\}, \mathbf{t}]$ implies q: $[\{1,2\}, \mathbf{t}]$, we would like to store the former rather than the latter. For this reason, we want to remember the twins of the atoms when building dynamic MULTI_OLDT-computations. The following examples show both how dynamic MULTI_OLDT-computations are done and how twins are used.

[4] Two set annotated queries Q_1^* and Q_2^* are called *distinct* if they are not variants/permutations of each other. The requirement of "distinctness" ensures that loops are eliminated during MULTI_OLDT-computations.

Example 5.10. Suppose we consider the lattice FOUR. Let P be the very simple program

$$p\colon [\{1\},\mathbf{t}] \leftarrow \tag{5.1}$$

$$p\colon [\{1\},\mathbf{f}] \leftarrow \tag{5.2}$$

$$q\colon [\{2\},\mathbf{t}] \leftarrow r\colon [\{1\},\mathbf{t}] \,\&\, p\colon [\{1,2\},\top] \tag{5.3}$$

$$r\colon [\{1\},\mathbf{t}] \leftarrow p\colon [\{1\},\top] \tag{5.4}$$

and let Q be the query $\leftarrow q$: $[\{1,2\},\mathbf{t}]$. The regular representation of Q is q: $[\{1,2\},\{\bot,\mathbf{f}\}] \leftarrow$. Figure 3 shows an example of a dynamic MULTI_OLDT-computation.

We now explain how Figure 3 corresponds to a dynamic MULTI_OLDT computation:

1. The regular representation of the original query is q: $[\{1,2\},\{\mathbf{f},\bot\}] \leftarrow$ —this resolves with the regular representation of clause (3), yielding

   $$r\colon [\{1\},\{\mathbf{f},\bot\}] \vee p\colon [\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow .$$

   Note that both set-annotated atoms in this S-resolvent having q: $[\{1,2\},\{\mathbf{f},\bot\}]$ as their parent, and this is shown by dotted links in the diagram. Similarly, the "twin" of the atom q: $[\{1,2\},\{\mathbf{f},\bot\}]$ is the head of the clause, i.e., q: $[\{2\},\mathbf{t}]$. The broken lines with arrows at both ends shown in the diagram link atoms and their twins. As this resolution did not occur with a program clause that had an empty body, the table remains empty after the S-resolution step.
2. At this stage, r: $[\{1\},\{\mathbf{f},\bot\}] \vee p$: $[\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow$ S-resolves with the regular representation of clause (4) in the program, yielding

   $$p\colon [\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}] \vee p\colon [\{1\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow .$$

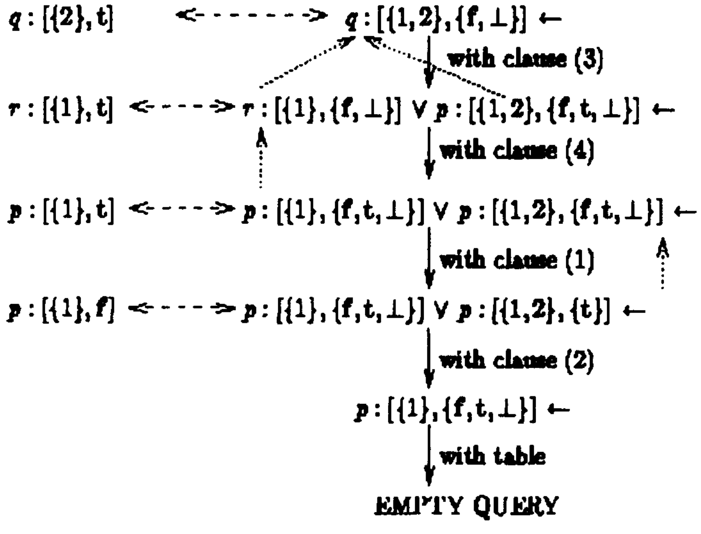

FIGURE 3. Dynamic MULTI_OLDT-computation of $\leftarrow q$: $[\{1,2\},\mathbf{t}]$.

Note that the parent of p: $[\{1\},\{\mathbf{f},\mathbf{t},\bot\}]$ is r: $[\{1\},\{\mathbf{f},\bot\}]$. As this resolution did not occur with a program clause that had an empty body, the table remains empty after the S-resolution step.

3. p: $[\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}] \vee p$: $[\{1\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow$ now resolves with regular representation of clause (1), yielding

   $$p\colon [\{1,2\},\{\mathbf{t}\}] \vee p\colon [\{1\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow.$$

   Note that the parent of the set-annotated atom p: $[\{1,2\},\{\mathbf{f},\bot\}]$ is p: $[\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}]$. As this resolution occurs with a program clause that had an empty body, this means that the annotated atom p: $[\{1\},\mathbf{t}]$ gets added to the table, i.e.,

   $$\Gamma=\{p\colon [\{1\},\mathbf{t}]\}.$$

4. In the next step, p: $[\{1,2\},\{\mathbf{t}\}] \vee p$: $[\{1\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow$ S-resolves with clause (2), yielding p: $[\{1\},\{\mathbf{f},\mathbf{t},\bot\}] \leftarrow$. As this resolution also occurred with a program clause having an empty body, the annotated atom p: $[\{1\},\mathbf{f}]$ must be inserted into the MULTI_OLDT-table, Γ. As Γ already contains the atom p: $[\{1\},\mathbf{t}]$ which is not implied by the atom p: $[\{1\},\mathbf{f}]$ being inserted, these two atoms must be "merged"; this leads to the new table

   $$\Gamma=\{p\colon [\{1\},\top]\}.$$

   Since all the children of p: $[\{1,2\},\{\mathbf{t}\}]$ are solved, its twin, which is p: $[\{1\},\mathbf{f}]$, should be inserted into Γ. Since p: $[\{1\},\top]$ subsumes p: $[\{1\},\mathbf{f}]$, Γ remains unchanged. Now, all of the children of p: $[\{1,2\},\{\mathbf{f},\mathbf{t},\bot\}]$ are solved, and its twin p: $[\{1\},\mathbf{t}]$ is inserted into the table. Since this atom is also subsumed by the atom in Γ, the table remains the same. At the next step of the propagation, the twin of the atom r: $[\{1\},\{\mathbf{f},\bot\}]$, which is r: $[\{1\},\mathbf{t}]$, is inserted into the table, giving

   $$\Gamma=\{p\colon [\{1\},\top], r\colon [\{1\},\mathbf{t}]\}.$$

   At the final step of the propagation, the atom q: $[\{1,2\},\{\mathbf{f},\bot\}]$ is solved, and its twin q: $[\{2\},\mathbf{t}]$ is added to the table to give the final table:

   $$\Gamma=\{p\colon [\{1\},\top], r\colon [\{1\},\mathbf{t}], q\colon [\{2\},\mathbf{t}]\}.$$

5. Finally, the set-annotated atom p: $[\{1\},\{\mathbf{f},\mathbf{t},\bot\}]$ resolves with (the regular representation of) p: $[\{1\},\top]$ in the table, yielding the empty query.

The preceding example does not show how the MULTI_OLDT-table gets modified when an atom has more than one child or when atoms contain annotation variables. To illustrate this better, consider the following example.

Example 5.11. Consider the amalgamated knowledge base in Example 5.10, and suppose we add the following clauses to it:

$$s\colon [\{\mathbf{m}\},V_1] \leftarrow p\colon [\{1\},V_1] \,\&\, q\colon [\{1,2\},\mathbf{t}]. \tag{5.5}$$

$$t\colon [\{\mathbf{m}\},V_2] \leftarrow s\colon [\{\mathbf{m}\},V_2] \,\&\, r\colon [\{1\},V_2]. \tag{5.6}$$

Let us now consider the simple query $Q = \leftarrow t$: $[\{\mathbf{m}\},V]$. The regular representation of Q is $Q^* = t$: $[\{\mathbf{m}\},\mathcal{T}- \Uparrow V] \leftarrow$. The dynamic MULTI_OLDT-computation

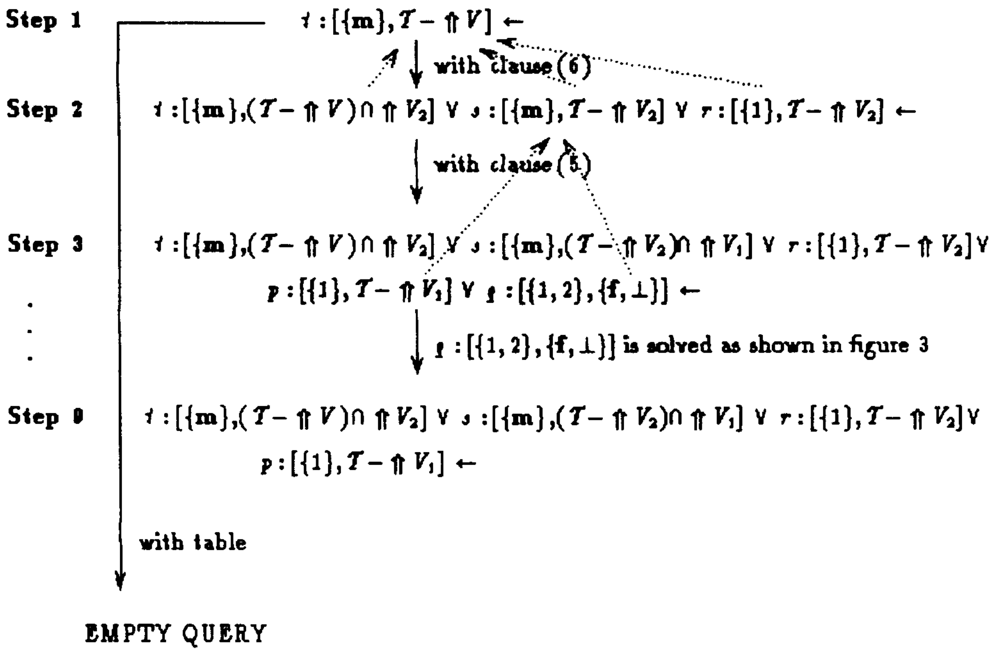
Step 1
t : [{m}, T − ⇑ V] ←
with clause (6)
Step 2
t : [{m}, (T − ⇑ V) ∩ ⇑ V₂] ∨ s : [{m}, T − ⇑ V₂] ∨ r : [{1}, T − ⇑ V₂] ←
with clause (5)
Step 3
t : [{m}, (T − ⇑ V) ∩ ⇑ V₂] ∨ s : [{m}, (T − ⇑ V₂) ∩ ⇑ V₁] ∨ r : [{1}, T − ⇑ V₂] ∨
p : [{1}, T − ⇑ V₁] ∨ q : [{1, 2}, {f, ⊥}] ←
q : [{1, 2}, {f, ⊥}] is solved as shown in figure 3
Step 8
t : [{m}, (T − ⇑ V) ∩ ⇑ V₂] ∨ s : [{m}, (T − ⇑ V₂) ∩ ⇑ V₁] ∨ r : [{1}, T − ⇑ V₂] ∨
p : [{1}, T − ⇑ V₁] ←
with table
EMPTY QUERY

FIGURE 4. Dynamic MULTI_OLDT-computation of $\leftarrow t\colon [\{\mathbf{m}\}, V]$.

associated with this query is shown in Figure 4—the twins of the atoms are not shown in the figure. Let us examine how this query is processed.

1. Initially, the MULTI_OLDT-table is empty, and the root of the dynamic MULTI_OLDT-computation has $Q^* = t\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V] \leftarrow$ as its label. This S-resolves with the clause (6), yielding the S-resolvent

   $$t\colon [\{\mathbf{m}\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow V_2] \vee s\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V_2] \vee r\colon [\{1\}, \mathcal{T} - \Uparrow V_2] \leftarrow .$$

   The three new set-annotated atoms all have $t\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V]$ as their parent (cf. dotted lines in Fig. 4). The twin of $t\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V]$ is $t\colon [\{\mathbf{m}\}, V_2]$. Since the body of clause (5.6) is not empty, the table remains empty.
2. In the next step, the atom $s\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V_2]$ is S-resolved with clause (5.5) in the program to give the resolvent

   $$t\colon [\{\mathbf{m}\}, (\mathcal{T} - \Uparrow V) \cap \Uparrow V_2] \vee sL\ [\{\mathbf{m}\}, (\mathcal{T} - \Uparrow V_2) \cap \Uparrow V_2] \vee$$
   $$r\colon [\{1\}, \mathcal{T} - \Uparrow V_2] \vee p\colon [\{1\}, \mathcal{T} - \Uparrow V_1] \vee q\colon [\{1,2\}, \{\mathbf{f}, \perp\}] \leftarrow .$$

   The three new atoms $p\colon [\{1\}, \mathcal{T} - \Uparrow V_1], q\colon [\{1,2\}, \{\mathbf{f}, \perp\}]$, and $s\colon [\{\mathbf{m}\}, (\mathcal{T} - \Uparrow V_2) \cap \Uparrow V_1]$ all have $s\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V_2]$ as their parent. The twin of $s\colon [\{\mathbf{m}\}, \mathcal{T} - \Uparrow V_2]$ is $s\colon [\{\mathbf{m}\}, V_1]$. Again, the table Γ remains empty.
3. At this stage, $q\colon [\{1,2\}, \{\mathbf{f}, \perp\}]$ is chosen, and it is processed as shown in Figure 3, and the table at the end of this process is as follows:

   $$\Gamma = \{p\colon [\{1\}, \top], q\colon [\{2\}, \mathbf{t}], r\colon [\{1\}, \mathbf{t}]\}.$$
4. (Propagation Step) Since one of the atoms in the query at step 3 is solved, the propagation of the result starts from this point. Consider the substitution

$\sigma_1 = \{V_1 = \top, V_2 = V_1\}$. The atom $(p\colon [\{1\}, \mathscr{T} - \Uparrow V_1])\sigma$ is true in Γ, and the atom $(s\colon [\{\mathbf{m}\}, (\mathscr{T} - \Uparrow V_2) \cap \Uparrow V_1])\sigma_1 \equiv s\colon [\{\mathbf{m}\}, \varnothing]$ is a tautology; hence, all the children of $s\colon [\{\mathbf{m}\}, \mathscr{T} - \Uparrow V_2]$ are solved via substitution σ. Hence, the twin of $(s\colon [\{\mathbf{m}\}, \mathscr{T} - \Uparrow V_2]])\sigma_1$, which is $(s\colon [\{\mathbf{m}\}, V_1])\sigma_1 \equiv s\colon [\{\mathbf{m}\}, \top]$, is inserted into Γ, giving

$$\Gamma = \{p\colon [\{1\}, \top], q\colon [\{2\}, \mathbf{t}], r\colon [\{1\}, \mathbf{t}], s\colon [\{\mathbf{m}\}, \top]\}.$$

The propagation continues. Since an atom in the query at step 2 is solved, its parent should be checked. Now, consider the substitution $\sigma_2 = \{V_2 = V = \mathbf{t}\}$. This satisfies all of the children in $t\colon [\{\mathbf{m}\}, (\mathscr{T} - \Uparrow V)]$ since the atoms $(r\colon [\{1\}, \mathscr{T} - \Uparrow V_2])\sigma_2 \equiv r\colon [\{1\}, \{\mathbf{f}, \perp\}]$ and $(s\colon [\{\mathbf{m}\}, \mathscr{T} - \Uparrow V_2]])\sigma_2 \equiv s\colon [\{\mathbf{m}\}, \{\mathbf{f}, \perp\}]$ are both true in Γ and the atom $(t\colon [\{\mathbf{m}\}, (\mathscr{T} - \Uparrow V) \cap \Uparrow V_2])\sigma_2 \equiv t\colon [\{\mathbf{m}\}, \varnothing]$ is a tautology. Hence, the twin of $(t\colon [\{\mathbf{m}\}, \mathscr{T} - \Uparrow V])\sigma_2$, which is the atom $t\colon [\{\mathbf{m}\}, \mathbf{t}]$, is inserted into Γ to give the table

$$\Gamma = \{p\colon [\{1\}, \top], q\colon [\{2\}, \mathbf{t}], r\colon [\{1\}, \mathbf{t}], s\colon [\{\mathbf{m}\}, \top], t\colon [\{\mathbf{m}\}, \mathbf{t}]\}.$$

5. Finally, the original query is resolved with the atom $t\colon [\{\mathbf{m}\}, \mathbf{t}]$ in Γ via substitution $\{V = \mathbf{t}\}$ to give the empty clause.

5.4.1. Soundness and Completeness of Dynamic MULTI_OLDT-Computation. We are now in a position to establish the soundness and completeness of dynamic MULTI_OLDT-computations.

Theorem 5.3. (Soundness and Completeness of Dynamic MULTI_OLDT-*Computation) Suppose P is an amalgamated knowledge base and Q is a query. Then*:

1. If $Q_1^*, \ldots, Q_n^*$ is $\Gamma_1, \ldots, \Gamma_n$ is a dynamic MULTI_OLDT computation associated with Q, and if $A\colon [D, \mu]$ is in Γ_n, then $P \models A\colon [D, \mu]$.
2. Suppose $C = (A_1\colon [D_1, \mu_1] \& \ldots \& A_k\colon [D_k, \mu_k])$. If $P \models (\forall) C\sigma$ for some substitution σ, then there exists a dynamic MULTI_OLDT computation associated with $Q = \leftarrow C$, and a table Γ, in this MULTI_OLDT-computation such that for all $1 \leq i \leq k$, either $\mu_i \sigma = \perp$ or there exists an atom $A'_i\colon [D'_i, \mu'_i] \in \Gamma_j$ such that $A'_i\colon [D'_i, \mu'_i]$ subsumes $A_i \sigma\colon [D_i, \mu_i \sigma]$.

PROOF. (1) By induction on n.

Base Case ($n = 1$). $A\colon [D, \mu] \in \Gamma_1$ means $\mu = \perp$, which means that $A\colon [D, \mu]$ is a tautology in the logic, and so $P \models A\colon [D, \mu]$.

Inductive Case ($n = m + 1$). In this case, by the induction hypothesis, all atoms $A\colon [D, \mu] \in \Gamma_m$ are logical consequences of P. Q_{m+1} is obtained in one of two ways:

1. The first possibility is that Q_{m+1} is the S-resolvent of an atom $A_1\colon [D_1, \mu_1] \in \Gamma_n$ and Q_j ($j \leq n$) on an atom $A_2\colon [D_2, \mu_2]$ via mgu θ. In this case, no atoms are added to the table as a result of the resolution. But, if it is the case that $(\mu_{2_s} \theta) \cap \Uparrow(\mu_1 \theta) = \varnothing$, then the parents of $A_2\colon [D_2, \mu_{2_s}]$ have to be checked. For this, we will prove the soundness of propagation in item 3 below.
2. The second possibility is that Q_{m+1} is obtained by S-resolving a clause C with Q_j ($j \leq n$) on an atom $A_2\colon [D_2, \mu_{2_s}]$ via mgu θ. If the body of the clause C is nonempty, then no atoms are added to the table. Otherwise, suppose $C = A_1\colon$

$[D_1, \mu_1] \leftarrow$. Then this atom is added to the table. Clearly, $P \models A_1$: $[D_1, \mu_1]$. Now, the propagation step starts.

3. (Soundness of propagation) Suppose A: $[D, \mu_s]$ is an atom in Q_k $(k \leq m)$, and suppose its twin is the atom A_t: $[D, \mu_t]$. The children of this atom are A_i: $[D_i, \mathcal{T} - \Uparrow \mu_i](1 \leq i \leq m)$ and $A\theta$: $[D, \mu_s \theta \cap \Uparrow \mu_t]$ for some mgu θ. Clearly, A_t: $[D_t, \mu_t] \leftarrow A_1$: $[D_1, \mu_1] \& \cdots \& A_m$: $[D_m, \mu_m]$ is an instance of a clause C in P; hence, $P \models C$. Now, suppose there exists a substitution σ such that all of the children $(A_i$: $[D_i, \mathcal{T} - \Uparrow \mu_i])\sigma$ and $(A\theta$: $[D, (\mu_s \cap \Uparrow \mu)\theta])\sigma$ of A: $[D, \mu]$ are true in Γ_n. In this case, $(A_t$: $[D_t, \mu_t])\sigma$ is added to Γ_{n+1}. By the definition of an atom being true in the table, there must be an atom A'_i: $[D'_i, \mu'_i] \in \Gamma_n$ such that $A_i \sigma$ is an instance of $A'_i, D'_i \subseteq D_i$ and $(\mathcal{T} - \Uparrow \mu_i \sigma) \cap \Uparrow \mu'_i = \emptyset$. By the induction hypothesis, $P \models A'_i$: $[D'_i, \mu'_i]$. Thus, $P \models A_i \sigma$: $[D_i, \mu'_i]$. Since $(\mathcal{T} - \Uparrow \mu_i \sigma) \cap \Uparrow \mu'_i = \emptyset$, then $\mu_i \sigma = \mu'_i$ is a solution to this equation; this implies that $P \models A_i \sigma$: $[D_i, \mu_i \sigma]$. Thus, all of the atoms in the body of C are logical consequences of P, and the same is true for the head of C, i.e., $P \perp (A_t$: $[D_t, \mu_t])\sigma$.

(2) Clearly, every MULTI_OLDT-resolution corresponds to an S-deduction. By definition of dynamic MULTI_OLDT-computations, an annotated atom is placed in the corresponding MULTI_OLDT table as soon as it is solved. Hence, by the completeness of S-resolution, if P is an amalgamated knowledge base having the fixpoint reachability property, and if $P \models \forall(A_i$: $[D_i, \mu_i])\sigma$, then either $\mu_i \sigma = \perp$ or there exists an S-refutation for the atom A'_i: $[D'_i, \mu'_i]$ where A'_i: $[D'_i, \mu'_i]$ subsumes $(A_i$: $[D_i, \mu_i])\sigma$. In the first case, the statement of the theorem is proven automatically. In the second case, the atom A'_i: $[D'_i, \mu'_i]$ will be placed in the MULTI_OLDT-table. □

5.4.2. Computation of Maximal Answers Using MULTI_OLDT-*Computations*. Having established the soundness and completeness of dynamic MULTI_OLDT-computations, it is now relatively straightforward to show that all maximal answers (in the declarative sense of Section 5.1.1) are guaranteed to be found by dynamic MULTI_OLDT-computations.

Theorem 5.4. (*Completeness of Maximal Answers Found by Dynamic* MULTI_OLDT-*Computation*) *Suppose P is an amalgamated knowledge base and Q is a query. Then*:

1. *Suppose $Q = A$: $[D, V]$ where A is a ground atom, and suppose μ is the maximal answer to this query. Then there is a dynamic* MULTI_OLDT-*computation consisting of $Q_1^*, \ldots, Q_n^*$ and $\Gamma_1, \ldots, \Gamma_n$ associated with Q such that A: $[D, \mu]$ is subsumed by some atom in Γ_n.*
2. *Suppose $Q = A$: $[D, V]$ where A is a nonground atom, and suppose (σ, μ) is a maximal answer to this query. Then there is a dynamic* MULTI_OLDT-*computation consisting of $Q_1^*, \ldots, Q_n^*$ and $\Gamma_1, \ldots, \Gamma_n$ associated with Q such that $A\lambda$: $[D, mu']$ is in Γ_n and $\mu \leq \mu'$ and λ is more general than σ.* □

PROOF. (1) As $P \models A$: $[D, \mu]$, it follows immediately that $P \models (\forall)[A$: $[D, V]]\sigma$ where $\sigma = \{V = \mu\}$. Therefore, it follows by part (2) of Theorem 5.3 that there exists a dynamic MULTI_OLDT-computation $Q_1^*, \ldots, Q_n^*$ and $\Gamma_1, \ldots, \Gamma_n$ associated with Q such that A: $[D, \mu]$ is subsumed by an atom in Γ_n.

(2) In this case, $P \models (\forall) A\sigma$: $[D, \mu]$. Therefore$P \models (\forall)(A$: $[D, V])\gamma$ where $\gamma = \sigma \cup \{V = \mu\}$. The result now follows immediately from part (2) of Theorem 5.3. □

5.4.3. Termination Properties of Dynamic MULTI_OLDT-*Computations.* In the definition of dynamic MULTI_OLDT-computations, the sequence of queries and the sequence of tables are finite. Conceptually, there is no reason these sequences cannot be infinite (although computational considerations benefit from finiteness). Suppose P is an amalgamated knowledge base, Q is a query, and $\text{DYN}_P(Q)$ is a dynamic MULTI_OLDT-computation associated with Q of the distinct sequence of queries $Q_1^*, Q_2^*, \ldots, Q_n^*$ and the sequence of MULTI_OLDT tables $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$. An *infinitary extension* of $\text{DYN}_P(Q)$ is any infinite sequence of queries Q_1^*, $Q_2^*, \ldots, Q_n^*, Q_{n+1}^*, \ldots$ and any sequence of MULTI_OLDT-tables, $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \Gamma_{n+1}, \ldots$ having $Q_1^*, Q_2^*, \ldots, Q_n^*$ and $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, respectively, as prefixes. These sequences satisfy all of the same conditions as MULTI_OLDT-tables —the only difference is that they are infinite. We would like to ensure that such computations do not arise when an interpreter attempts to construct dynamic MULTI_OLDT-computations. We now define conditions on MULTI_OLDT-computations that will allow infinite computations to be eliminated.

Definition 5.9. (Finitary Dynamic MULTI_OLDT-Computation) Suppose P is an amalgamated knowledge base, Q is a query, and $\text{DYN}_P(Q)$ is a dynamic MULTI_OLDT-computation associated with Q of the distinct sequence of queries $Q_1^*, Q_2^*, \ldots, Q_n^*$ and the sequence of MULTI_OLDT tables $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$. Then $\text{DYN}_P(Q)$ is said to be *finitary* iff, whenever an atom A: $[D, \mu_s]$ in query Q_i^*, $i < n$ is S-resolved with a clause C in the program to give Q_i^*, $i < j \leq n$, it is the case that:

1. (Subsumption Rule) neither A: $[D, \mu_s]$ nor any of the parents of A: $[D, \mu_s]$ are entailed by any of the children A: $[D, \mu_s]$,
2. (Irredundancy Rule) there is no other query in the above sequence of queries that was obtained from Q_i^* by S-resolving on atom A: $[D, \mu_s]$ with clause C in the program, and
3. (Halting Rule) if Q_i^* is the empty query, then $i = n$.

Definition 5.10. (Subcomputations) Suppose P is an amalgamated knowledge base, Q is a query, $[(Q_1^*, Q_2^*, \ldots, Q_n^*, Q_{n+1}^*, \ldots), (\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \Gamma_{n+1}, \ldots)]$ is an infinite MULTI_OLDT-computation of Q w.r.t. P. Any MULTI_OLDT-computation

$$\left[(Q_{\alpha(1)}^*, Q_{\alpha(2)}^*, \ldots, Q_{\alpha(m)}^*), (\Gamma_1', \Gamma_2', \ldots, \Gamma_m')\right]$$

where

1. $Q_1^* = Q_{\alpha(1)}^*$,
2. $\Gamma_1 = \Gamma_1'$, and
3. each $Q_{\alpha(i)}^* = Q_j^*$ for some $1 \leq i \leq j$
is said to be a *subcomputation* of the infinite MULTI_OLDT-computation given above.

The following result shows that given any atom A: $[D, \mu]$ that is true in a table associated with an infinitary MULTI_OLDT-computation, there is an equivalent subcomputation.

Lemma 5.4. Suppose P is an amalgamated knowledge base that has no function symbols (logical and annotation). Suppose Q is a query. Then for every infinite MULTI_OLDT*-computation,*

$$\mathscr{C} = \left[(Q_1^*, Q_2^*, \ldots, Q_n^*, Q_{n+1}^*, \ldots), (\Gamma_1, \Gamma_2 2, \Gamma_n, \Gamma_{n+1}, \ldots)\right]$$

of Q w.r.t. P, there exists a finitary MULTI_OLDT*-subcomputation*

$$\mathscr{C}' = \left[(Q_{\alpha(1)}^*, Q_{\alpha(2)}^*, \ldots, Q_{\alpha(m)}^*), (\Gamma_1', \Gamma_2', \ldots, \Gamma_m')\right]$$

of $\mathscr{C}$ such that if A: $[D, \mu]$ is entailed by an atom in $\bigcup_{i=1}^{\infty}\Gamma_i$, then A: $[D, \mu]$ is entailed by an atom in Γ_m'.

PROOF. As there is no annotation functions and Datalog function symbols, only finitely many annotated atoms (up to renaming of variables) can be generated in $\mathscr{C}$. Thus, there exists an integer i such that Γ_i implies A: $[D, \mu]$ iff $\bigcup_{j=1}^{\infty}\Gamma_j$ implies A: $[D, \mu]$. □

5.4.4. Search Strategy. Lemma 5.4 says that whenever an amalgamated knowledge base contains no annotation functions and no ordinary function symbols, then all MULTI_OLDT-computations can be made "finitary." The space of MULTI_OLDT-computations associated with a query may be viewed as a finitely branching tree, all of whose paths are MULTI_OLDT-computations. Let us call this tree Γ_P. Thus, each node in Γ_P is labeled with Q^* for some query Q. As each path can, by the above lemma, be "truncated" at a finite level, this means that this tree is finite. Hence, there exists a search procedure that searches this space (the well-known A^* algorithm [26] can be used) with guaranteed termination. Note that algorithms such as A^* can search such search spaces even if cycles occur in the graph. Furthermore, A^* is guaranteed to terminate as well.

Another issue that needs to be addressed is the question: "When should we use the table?" We say that a search strategy is *fair* iff, whenever a node N in Γ_P is labeled with query Q^* containing an atom A: μ', the following conditions hold:

1. if there is a clause C in P of the form: $A: \mu$ where μ is a constant-annotation and μ is greater than $\sqcup\{\rho | A'$: ρ is an atom in the MULTI_OLDT-table and A is an instance of $A'\}$, then C is eventually used in an S-resolution step to generate a node in the subtree of Γ_P rooted at N *or*
2. if there is a clause C in P of the form $A: V$ is a variable-annotation, then C is eventually used in an S-resolution step to generate a node in the subtree of Γ_P rooted at N.

The first condition above guarantees that we never use a rule unless that rule is likely to generate a truth value for an atom that exceeds the truth value of the atom currently implies by the MULTI_OLDT-table. The two conditions jointly state that any rule that is likely to enhance the truth value of an atom is eventually used. It is easy to see that *any* fair search strategy is guaranteed to preserve completeness. These include fair computation strategies such as the strategies proposed by Chen and Warren in [9]. However, different fair search strategies may lead to different degrees of efficiency in query processing; an empirical evaluation that is beyond the scope of this paper is needed for this purpose.

5.5. *Implementation of* MULTI_OLDT *Resolution*

5.5.1. Overview. Two different data structures are needed for the implementation of dynamic MULTI_OLDT-computations: a table and a list of queries. These structures will be referred to as TABLE and QUERY, respectively. The detailed description of these data structures, together with the pseudocode for the dynamic MULTI_OLDT-computations, can be found in [1]. All of these data structures and algorithms have been implemented by Kullman [20] (with minor modifications).

There are a few differences between the mathematical model of dynamic MULTI_OLDT-computations and the real data structures used to implement them. In the implementation, QUERY is just a list of atoms. In contrast, in the mathematical model, an atom in a query contained pointers to its parent, its children, and its twin (when applicable). This information will not be stored in the QUERY data structure. Instead, this information will be encapsulated within the TABLE data structure. Jointly, the TABLE and QUERY data structures will contain the same information present in the mathematical framework given in the preceding section.

Recall the dynamic $\Gamma_1, \ldots, \Gamma_n$ of MULTI_OLDT tables. At step n, the TABLE data structure contains all of the atoms in Γ_n, and the QUERY data structure will contain all of the atoms in $Q_1^* - Q_n^*$. Since the queries in the sequence may have some atoms in common, duplication will thus be eliminated. As the sequence of queries is flattened to a single query, links are established in the TABLE to indicate the relative positions of atoms in QUERY.

In contrast to the queries in dynamic MULTI_OLDT-computations, the atoms in QUERY are atoms of the form A: $[D, V]$—note that a query of the form A: $[D, \mu]$ can be viewed as: "Find a value V such that A: $[D, V]$ is true and where V is greater than or equal to the desired value, μ."

TABLE is a linked list of records. Each record in the list contains information about an atom in QUERY. Hence, if A: $[D, V]$ is at atom in QUERY, then the TABLE record R corresponding to this atom contains links to the parent, the children, and the twin in A: $[D, V]$. R also has a field that stores the list of substitutions (for both ordinary and annotation variables) θ such that $A\theta$: $[D, \mu\theta]$ is in Γ_n. The table insertion routine will update this field only.

5.5.2. Description of Data Structures. In this section, we briefly describe data structures to implement the dynamic MULTI_OLDT-computations described above.

The QUERY *Data Structure.* As explained in Section 5.1, every resolution step results in a new maximization problem in the annotation term. Consider the situation when the query A: $[D, \mathcal{T} - \Uparrow T_0] \leftarrow$ is S-resolved with the clause A: $[D, T_1] \leftarrow$ to given the resolvent A: $[D, (\mathcal{T} - \Uparrow T_0) \cap \Uparrow T_1]$. As explained in Section 5.1, the maximal truth value of T_0 that causes the annotation term in the above resolvent to evaluate to $\varnothing$ is $T_0 = T_1$. Suppose, now, that the above resolvent is further resolved with the clause A: $[D, T_2] \leftarrow$. The new resolvent is A: $[D, (\mathcal{T} - \Uparrow T_0) \cap \Uparrow T_1 \cap \Uparrow T_2]$, and the maximum truth value of T_0 that causes the annotation term to evaluate to the $\varnothing$ is $T_0 = T_1 \sqcup T_2$.

As we can see from the above example, there are no need to explicitly store the set-annotations produced during intermediate levels of resolution. Instead, only the current maximal truth value of the annotation term (there is always a unique solution to the corresponding maximization problem by Theorem 5.1) and the unifying substitution for this term need to be saved. If the annotation term T_0 in the original query had been ground ($T_0 = \mu_0$), then it can be replaced with an annotation variable, V_0. The query A: $[D, \mathcal{T} - \Uparrow \psi_0] \leftarrow$ will be solved when the current maximal truth value, μ_0^* of V_0 exceeds μ_0. (If $\mu_0 \leq \mu_0'$, then $(\mathcal{T} - \Uparrow \mu_0) \cap \Uparrow \mu_0' = \emptyset$.) Note that annotation functions are only allowed in the head of clauses; hence, T_0 can only be a variable or a constant.

If the annotation term T_1 above is a variable V_1, then initially V_1's truth value is unknown; hence, its maximal (current) truth value is $\perp$. Whenever V_1's value changes, this change should be reflected to T_0 since $T_0 = V_1$. In case T_1 is a complex function of the form $f(V_1, \ldots, V_m)$ and initially the values of $V_1, \ldots, V_m$ are all unknown, then the initial truth value of T_0 is $T_0 = f(\perp, \ldots, \perp)$. This value is updated every time the value of one of V_i, $1 \leq i \leq m$ changes.

Hence, only the atoms A: $[D, V]$ are stored in QUERY, whereas details such as the name of the annotation variable, its truth value if it is ground, or the address of the code implementing the annotation function are stored in the TABLE. Finally, when the atom A: $[D, \mathcal{T} - \Uparrow V_0] \leftarrow$ is S-resolved with another clause A: $[D, \Uparrow V_0'] \leftarrow$, the maximum truth value of V_0 is set to lub of T_0' and the current maximum truth value of V_0 by Lemma 5.1.

The TABLE *Data Structure.* The TABLE data structure is a linked collection of records. Each record contains information about an atom A: $[D, V]$ in the query. This information can be categorized as follows:

- *Information about the annotation term V:* If V was a variable substituted for a ground term μ, then the value of μ is stored. Otherwise, a status bit indicating that V was nonground is set.
- *Information about the position of the atom in the query sequence:* A link is set to the parent of this atom, and all of the children of the same atom are placed next to each other in the table. This way, all of the children of an atom will constitute a block of atoms in TABLE having the same parent link.
- *Information about the twin of an atom:* Note that twins are obtained when an S-resolution is performed. After an S-resolution, the children of A: $[D, V]$ are placed consecutively in the table. Then an additional record for the twin of A: $[D, V]$ is added to the end of the block containing the records for the children in A: $[D, V]$. If the twin (the head of a clause) contains an annotation function, the address of the code implementing this function is also placed in this additional record.
- *Information about the current instances of A: $[D, V]$ that have been proven to be true:* This field is stored as a list of substitutions.

Suppose C is a clause in the amalgamated knowledge base. Then the head of C may contain an annotation function of the form $f(V_1, \ldots, V_m)$. In this case, the following assumptions are made about C.

- All of V_i, $1 \leq i \leq m$ are variables only.
- There is no nesting among the function symbols in $f(V_1, \ldots, V_m)$.
- Every variable occurs only once in the term.
- All of the variables $F(V_1, \ldots, V_m)$ occur at least once in one of the atoms in the body of C.

- The atoms in the body of C are arranged so that all of the atoms with the annotation variable V_1 come first, then those with V_2, and so on.

There is no loss of generality in the above assumptions—for instance, the annotation term $f(V_1, \ldots, h(V_m))$ which contains nested functions can be replaced by another term $f(V_1, \ldots, W)$ where $W = f(V_m)$.

The Interruptability of MULTI_OLDT-*Computations.* At any given stage during an MULTI_OLDT-computation, the user may wish to halt processing and examine the MULTI_OLDT-table. As the information in the MULTI_OLDT-table is monotonically improving (i.e., the set of annotated atoms entailed by the table increases as more and more time is spent processing the query), this means that the user can halt processing when he needs to, and do the best he can with the answers obtained thus far (if he has no further time to continue processing).

The reader who is interested in details of the algorithms manipulating the QUERYl and TABLE data structures may read the technical report for the required pseudocode [1]. They implement the algorithms described in Section 5.2 using the TABLE and QUERY data structures described above. The pseudocode has also been implemented by Kullman in Germany [20].

6. RELATED WORK

A great deal of work has been done in *multidatabase* systems and *interoperable* database systems [40, 16, 37]. However, most of this work combines standard relational databases (no deductive capabilities). Not much has been done on the development of a semantic foundation for such databases. The work of Grant et al. [16] is an exception: the authors develop a calculus and an algebra for integrating information from multiple databases. This calculus extends the standard relational calculus. Further work specialized to handle the interoperability of multidatabases is critically needed. However, our paper addresses a different topic—that of integrating multiple deductive databases containing (possibly) inconsistencies, uncertainty, nonmonotonic negation, and possibly even temporal information. Zicari et al. [40] describe how interoperability may be achieved between a rule-based system (deductive DB) and an object-oriented database using special *import/export* primitives. No formal theory is developed in [40]. Perhaps closer to our goal is that of Whang et al. [37] who argue that Prolog is a suitable framework for schema integration. In fact, the approach of Whang et al. is in the same spirit as that of metalogic programming discussed earlier. Whang et al. do not give a formal semantics for multidatabases containing inconsistency and/or uncertainty and/or nonmonotonicity and/or temporal information.

Baral et al. [2, 3] have developed algorithms for combining different logic databases which generalizes the update strategy by giving priorities to some updates (when appropriate), as well as not giving priorities to updates (which corresponds to combining two theories without any preferences). Combining two theories corresponds, roughly, to finding maximally consistent subsets (also called flocks by Fagin et al. [13, 14]). As we have shown in [31], our framework can express maximal consistency as well. References [2, 3] do not develop a formal model-theoretic treatment of combining multiple knowledge bases, whereas our method does provide such a model theory. References [2, 3] are unable to handle nonmonotonicity (in terms of stable/well-founded semantics), or uncertainty, or time-stamped information—our framework is able to do so.

Dubois et al. [12] also suggest that formulas in knowledge bases can be annotated with, for each source, a lower bound of a degree of certainty associated with that source. The spirit behind their approach is similar to ours, although interest is restricted to the $[0,1]$ lattice, the stable and well-founded semantics are not addressed, and amalgamation theorems are not studied. However, for the $[0,1]$ case, their framework is a bit richer than ours when nonmonotonic negations are absent.

In [15], Fitting generalizes results in [34, 4], to obtain a well-founded semantics for bilattice-based logic programs. We have given a detailed comparison of our declarative framework with Fitting's in [31].

Our work builds upon works by Lu et al. [23] who have independently developed a framework for query processing in GAPs. As stated by Leach and Lu [21], the work of [23] applies not just to the Horn-clause fragment of annotated logic (which is the case in our work), but to the full-blown logic. However, [23] does *not* deal with annotation variables and annotated functions—our results apply to those cases as well. Finally, our development of MULTI_OLDT-resolution is new.

Warren and his co-workers [10, 9] have studied OLDT-resolution for ordinary logic programs (both with and without nonmonotonic forms of negation). In this paper, we have dealt only with the monotonic case, and have focused on: (1) how truth value estimates of atoms can be monotonically improved as computation proceeds, and how this monotonic improvement corresponds to solving certain kinds of incremental optimization problems over a lattice domain, and (2) how OLDT tables must be organized so as to efficiently support such computations. As shown by Warren [36], OLDT-resolution is closely related to magic set computations, and hence our work enjoys the same relationships with magic sets discussed in [36].

7. CONCLUSIONS

Wiederhold has proposed mediators as a framework within which multiple databases may be integrated. In the first of this series of papers [31], it has been shown that certain forms of annotated logic provide a simple language within which mediators can be expressed. In particular, it was shown that the semantics of "local" databases can be viewed as embeddings within the semantics of amalgamated databases. In [31], we did not develop an operational theory for query processing in amalgamated KBs. In this paper, we have provided a framework for implementing such a query processing paradigm. This framework supports:

- *incremental, approximate query processing* in the sense that truth value estimates for certain atomic queries will increase as we continue processing the query. Thus, if a user (or a machine) wishes to interpret the processing, then at least an approximate estimate will be obtained, based on which a knowledge-based system may take some actions.
- *reuse of previous computations* using the table data structure(s). In particular, we have specified access paradigms for updating answers, i.e., (substitution, truth-value) pairs as processing continues.

In future work, we will extend the above paradigm to handle nonmonotonic modes of negation. The work being described here is being implemented as part of system called HERMES (Heterogeneous Reasoning and Mediator System) that allows not only for the integration of multiple databases, but also multiple data structures, software packages, and reasoning paradigms [32].

APPENDIX: PROOFS OF RESULTS ON S-RESOLUTION

PROOF OF THEOREM 4.1. Suppose C^* is the regular representation of a clause and Q^* is a set-annotated query as specified in Definitions 4.2 and 4.5. Let θ be the mgu of A_0 and B_i.

Suppose I S-satisfies C^* and Q_k^* and $(Q_{k+1}^*)\sigma$ is a ground instance of (Q_{k+1}^*). Since $Q_k^*\theta\sigma$ and $C^*\theta\sigma$ must be ground and $I \models^{\mathrm{S}} Q_k^*, I \models^{\mathrm{S}} C^*$, it must be the case that $I \models^{\mathrm{S}} Q_k^*\theta\sigma$ and $I \models^{\mathrm{S}} C^*\theta\sigma$. We need to show that I S-satisfies (Q_{k+1}^*). Since I S-satisfies $Q_k^*\theta\sigma$, it must S-satisfy one of the amalgamated atoms B_j: $[D_{q_j}, \mu_{q_{s_j}}]\theta\sigma$. There are two cases to consider:

- *Case* 1. $(j \neq i)$ In this case $(B_j$: $[D_{q_j}, \mu_{q_{s_j}}])\theta\sigma$ occurs in $(Q_{k+1}^*)\sigma$ and I S-satisfies this atom in $(Q_{k+1}^*)\sigma$, and therefore satisfies the resolvent.
- *Case* 2. $(j = i)$ In this case, I must S-satisfy B_i: $([D_{q_i}, \mu_{q_{s_i}}]\}\theta\sigma$ in $Q_k^*\theta\sigma$. Since I S-satisfies $C^*\theta\sigma$, there are two cases to consider:

  —*Case* 2.1. I falsifies the body of $C^*\theta\sigma$. Then there must be at least one atom $(A_k$: $[D_i, \Uparrow \mu_k])\theta\sigma$ that is not S-satisfied in I. Let $\mu_I = \sqcup_{d \in D_k} I(A\theta\sigma)(d)$. Since $\mu_I \notin \mu_k$, it must be the case that, $\mu_I \in (\mathcal{T} - \mu_k)$. Then $(A_k$: $[D_k, \mathcal{T} - \Uparrow \mu_k])\theta\sigma$ must be S-satisfied in I. Since this atom occurs in $(Q_{k+1}^*)\sigma$, I satisfies (Q_{k+1}^*).

  —*Case* 2.2. I S-satisfies both the body and the head of the clause $C^*\theta\sigma$. Then, by the definition of S-satisfaction, there exists a truth value $\mu' \in \Uparrow \mu_0$ such that I **A**-satisfies $(A_0$: $[D_0, \mu])\theta\sigma$. Then, since $D_0 \subseteq D_{q_i}$, I must **A**-satisfy an annotation $(B_i$: $[D_{q_i}, \mu''])\theta\sigma$ such that, $\mu'' \geq \mu' \geq \mu$. This implies that $\mu'' \in \Uparrow \mu$, and this annotation occurs in the resolvent. Therefore, I S-satisfies the resolvent. □

The proof of the Completeness Theorem (Theorem 4.2) for S-resolution needs several intermediate theorems that are stated below.

Theorem 7.1. (*Ground Completeness of S-Resolution*) *Suppose Q is the ground query $\leftarrow A$: $[D, \mu]$, $P \models A$: $[D, \mu]$, and that P possesses the fixpoint reachability property. Then there is an unrestricted S-refutation of $(\leftarrow Q)^*$ from P^*.* (*An unrestricted refutation does not require the unifier used at each deduction step to be the most general unifier.*)

PROOF. As P satisfies the fixpoint reachability property, we know that $\mathbf{A}_Q \uparrow k$ satisfies A: $[D, \mu]$ for some $k < \omega$. We proceed by induction on k.

Base case $(k = 1)$. According to the definition of $\mathbf{A}_Q$, there exist ground instances

$$A\colon [D_1, \mu_1] \rightarrow$$

$$A\colon [D_2, \mu_2] \leftarrow$$

$$\dots$$

$$A\colon [D_m, \mu_m] \leftarrow$$

of a finite set of clauses

$$A_1\colon [D'_1, \mu'_1] \leftarrow$$
$$A_2\colon [D'_2, \mu'_2] \leftarrow$$
$$\cdots$$
$$A_m\colon [D'_m, \mu'_m] \leftarrow$$

In P, $m \geq 1$, such that $\sqcup\{\mu_1, \ldots, \mu_m\} \geq \mu$ and $\bigcup_{1 \leq j \leq m} D_j \subset D$. Note that for all $1 \leq i \leq m$, there is a substitution θ_i such that $A_i\theta_i = A$, $[D'_i, \mu'_i]\theta = [D_i, \mu_i]$. By the definition of regular representation, P^* contains ground instances

$$A\colon [D_1, \Uparrow \mu_1] \leftarrow$$
$$A\colon [D_2, \Uparrow \mu_2] \leftarrow$$
$$\cdots$$
$$A\colon [D_m, \Uparrow \mu_m] \leftarrow$$

of unit clauses

$$A_1\colon [D'_1, \Uparrow \mu'_1] \leftarrow$$
$$A_2\colon [D'_2, \Uparrow \mu'_2] \leftarrow$$
$$\cdots$$
$$A_m\colon [D'_m, \Uparrow \mu'_m] \leftarrow$$

and $(\leftarrow Q)^* = A\colon [D, \mathcal{T} - \Uparrow \mu] \leftarrow$. Since for all $1 \leq i \leq m$, $D_i \subseteq D$, $(\leftarrow Q)^*$ resolves with all $A_i\colon [D_i, \Uparrow \mu_i]$. It follows that there is an S-refutation

$$\langle A\colon [D, \mathcal{T} - \Uparrow \mu] \leftarrow, A\colon [D_1, \Uparrow \mu_1] \leftarrow \theta_1 \rangle,$$
$$\langle A\colon [D, \mathcal{T} - \Uparrow \mu \cap (\Uparrow \mu_1)])) \leftarrow, A\colon [D_2, \Uparrow \mu_2] \leftarrow, \theta_2 \rangle,$$
$$\ldots,$$
$$\langle A\colon [D, (\mathcal{T} - \Uparrow \mu) \cap \cap_{1 \leq i \leq m} \Uparrow \mu_i] \leftarrow, -, - \rangle.$$

We must show that the last query evaluates to $\varnothing$. Let $\mu_{lub} = \sqcup\{\mu_1, \ldots, \mu_m\}$. Since $\mu_{lub} \geq \mu$, we have $\Uparrow \mu_{ub} \subseteq \Uparrow \mu$; hence, $\Uparrow \mu_{lub} \cap (\mathcal{T} - \Uparrow \mu) = \varnothing$. Then it suffices to show that $(\bigcap_{1 \leq i \leq m} \Uparrow \mu_i) \subseteq \Uparrow \mu_{lub}$. For all $\mu_k \in (\bigcap_{1 \leq i \leq m} \Uparrow \mu_i)$, we have that $\mu_k \geq \mu_j$ for all j. Since μ_{lub} is the smallest such truth value, we must have $\mu_k \geq \mu_{lub}$, and therefore $\mu_k \in \Uparrow \mu_{lub}$.

Inductive Case $(k > 1)$. By the definition $\mathbf{A}_Q$, there exist ground instances $C_1\theta_1, \ldots, C_m\theta_m$ of the form

$$A\colon [D_1, \mu_1] \leftarrow B_1^1\colon [D_1^1, \mu_1^1] \,\&\cdots\&\, B_{k_1}^1\colon [D_{k_1}^1, \mu_{k_1}^1]$$
$$A\colon [D_2, \mu_2] \leftarrow B_1^2\colon [D_1^2, \mu_1^2] \,\&\cdots\&\, B_{k_2}^2\colon [D_{k_2}^2, \mu_{k_2}^2]$$
$$\cdots$$
$$A\colon [D_m, \mu_k] \leftarrow B_1^m\colon [D_1^m, \mu_1^m] \,\&\cdots\&\, B_{k_m}^m\colon [D_{k_m}^m, \mu_{k_m}^m]$$

of clauses $C_1, \ldots, C_m$

$$A_1\colon [D'_1, \mu'_1] \leftarrow B_1^1\colon [D_1^{1'}, \mu_1^{1'}] \,\&\cdots\&\, B_{k_1}^1\colon [D_{k_1}^{1'}, \mu_{k_1}^{1'}]$$
$$A_2\colon [D'_2, \mu'_2] \leftarrow B_1^2\colon [D_1^{2'}, \mu_1^{2'}] \,\&\cdots\&\, B_{k_2}^2\colon [D_{k_2}^{2'}, \mu_{k_2}^{2'}]$$
$$\cdots$$
$$A_m\colon [D'_m, \mu'_m] \leftarrow B_1^m\colon [D_1^{m'}, \mu_1^{m'}] \,\&\cdots\&\, B_{k_m}^m\colon [D_{k_m}^{m'}, \mu_{k_m}^{m'}]$$

in P, $m \geq 1$ such that $\sqcup\{\mu_1,\ldots,\mu_m\} \geq \mu$, $\bigcup_{1 \leq j \leq m} D_j \subseteq D$ and $\mathbf{A}_Q \uparrow (k-1) \models B_1^i$: $[D_1^i, \mu_1^i] \& \cdots \& B_{k_i}^i$: $[D_{k_i}^i, \mu_{k_i}^i]$, and there is a substitution θ_i such that $A_i\theta_i = A$, $[D_i', \mu_i']\theta = [D_i, \mu_i]$ for all $1 \leq i \leq m$. By the definition of regular expression, P^* contains ground instances $C_1^*\theta_1,\ldots,C_m^*\theta_m$

$$A\colon [D_1, \Uparrow \mu_1] \leftarrow B_1^1\colon \left[D_1^1, \Uparrow \mu_1^1\right] \& \cdots \& B_{k_1}^1\colon \left[D_{k_1}^1, \Uparrow \mu_{k_1}^1\right]$$

$$A\colon [D_2, \Uparrow \mu_2] \leftarrow B_1^2\colon \left[D_1^2, \mu_1^2\right] \& \cdots \& B_{k_2}^2\colon \left[D_{k_2}^2, \Uparrow \mu_{k_2}^2\right]$$

$$\ldots$$

$$A\colon [D_m, \Uparrow \mu_m] \leftarrow B_1^m\colon [D_1^m, \Uparrow \mu_1^m] \& \cdots \& B_{k_m}^m\colon \left[D_{k_m}^m, \Uparrow \mu_{k_m}^m\right]$$

of clauses $C_1,\ldots,C_m$

$$A_1\colon [D_1', \Uparrow \mu_1'] \leftarrow B_1^1\colon \left[D_1^{1'}, \Uparrow \mu_1^{1'}\right] \& \cdots \& B_{k_1}^1; \left[D_{k_1}^{1'}, \Uparrow \mu_{k_1}^{1'}\right]$$

$$A_2\colon [D_2', \Uparrow \mu_2'] \leftarrow B_1^2\colon \left[D_1^{2'}, \Uparrow \mu_1^{2'}\right]) \& \cdots \& B_{k_2}^2\colon \left[D_{k_2}^{2'}, \Uparrow \mu_{k_2}^{2'}\right]$$

$$\ldots$$

$$A_m\colon (D_m', \Uparrow \mu_m'] \leftarrow B_1^m\colon [D_1^{m'}, \Uparrow \mu_1^{m'}] \& \cdots \& B_{k_m}^m\colon \left[D_{k_m}^{m'}, \mu_{k_m}^{m'}\right].$$

By the inductive hypothesis, there is an S-refutation R_i of

$$B_1^i\colon \left[D_1^i, \mathcal{T} - \Uparrow \mu_1^i\right] \& \cdots \& B_{k_i}^i\colon \left[D_{k_i}^i, \mathcal{T} - \Uparrow \mu_{k_i}^i\right] \leftarrow$$

for all $1 \leq i \leq m$. By the same argument as above, $(\mathcal{T} - \Uparrow \mu) \cap \bigcap_{1 \leq i \leq m} \Uparrow \mu_i = \varnothing$. Therefore, $(\leftarrow Q)^*$ has an unrestricted S-refutation as follows:

$$\langle A\colon [D, \mathcal{T} - \Uparrow \mu] \leftarrow, C_i^*, \theta_i\rangle,$$

$$\ldots,$$

$$\langle A\colon \left[D, (\mathcal{T} - \Uparrow \mu) \cap \bigcap_{1 \leq i \leq m} \Uparrow \mu_i = \varnothing\right] \leftarrow, -, -\rangle,$$

$$R_1,\ldots,R_m,$$

$$\langle \leftarrow, -, -\rangle. \qquad \square$$

The completeness of S-resolution may now be established from the ground completeness result using standard techniques.

Lemma 7.1. (mgu Lemma) Suppose there is an unrestricted S-refutation $(\leftarrow Q)^*\theta$ from an amalgamated knowledge base P. Then there is an S-refutation of $(\leftarrow Q)^*$ from P. □

Lemma 7.2. (Lifting Lemma) Suppose there is an S-refutation of $(\leftarrow Q)^*\theta$ from an amalgamated knowledge base P. Then there is an S-refutation of $(\leftarrow Q)^*$ from P. □

The completeness of S-resolution is an immediate consequence of the ground completeness theorem and mgu lemma.

We have benefited from conversations with Mike Kifer, Jim Lu and Terry Swift. We are particularly grateful to Jim Lu for helping us determine the differences between S-resolution and work in [23].

REFERENCES

1. Adalı, S., and Subrahmanian, V. S., Amalgamating Knowledge Bases: II: Algorithms, Data Structures and Query Processing, Technical Report CS-TR-3124, Computer Science Department, University of Maryland, College Park, 1993.
2. Baral, C., Kraus S., and Minker J., Combining Multiple Knowledge Bases, *IEEE Trans. Knowledge and Data Eng.* 3(2):200–220 (1991).
3. Baral, C., Kraus S., Minker, J., and Subrahmanian, V. S., Combining Knowledge Bases Consisting of First Order Theories, *Computational Intell.* 8(1):45–71 (1992).
4. Baral, C., and Subrahmanian, V. S., Dualities between Alternative Semantics for Logic Programming and Nonmonotonic Reasoning, in: *Proc. 1991 Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, MIT Press, 1991. Full version in *J. Automated Reasoning* 10:339–420 (1993).
5. Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J., Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. 5th Symp. on Principles of Database Syst.*, 1986, pp. 1–15.
6. Beeri, C., and Ramakrishnan, R., On the Power of Magic, in: *Proc. 6th Symp. on Principles of Database Syst.*, 1987, pp. 269–283.
7. Belnap, N. D., Jr., A Useful Four Valued Logic, in: G. Epstein and J. M. Dunn (eds.), *Modern Uses of Many Valued Logic*, McGraw-Hill, New York, 1977, pp. 8–37.
8. Blair, H. A., and Subrahmanian, V. S., Paraconsistent Logic Programming, *Theoretical Comput. Sci.* 68:35–54 (1989). Preliminary version in: *LNCS 287*, Springer, Dec. 1987.
9. Chen, W., and Warren, D. S., A Goal-Oriented Approach to Computing Well-Founded Semantics, in: *Proc. 1992 Int. Conf. on Logic Programming*, K. R. Apt (ed.), MIT Press, 1992.
10. Dietrich, S., Extension Tables: Memo Relations in Logic Programming, in: *Proc. 1987 IEEE Symp. on Logic Programming*, 1986, pp. 264–272.
11. Dubois, D., Lang, J., and Prade, H., Towards Possibilistic Logic Programming, in: *Proc. 1991 Int. Conf. on Logic Programming*, K. Furukawa (ed.), MIT Press, 1991, pp. 581–595.
12. Dubois, D., Lang, J., and Prade, H., Dealing with Multi-Source Information in Possibilistic Logic, in: *Proc. 10th European Conf. on Artificial Intelligence*, Wiley, 1992.
13. Fagin, R., Ullman, J. D., and Vardi, M. Y., On the Semantics of Updates in Databases, in: *Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Syst.*, pp. 352–365.
14. Fagin, R., Kuper, G., Ullman, J., and Vardi, M., Updating Logical Databases, in: *Advances in Computing Research, Vol. 3*, 1986, pp. 1–18.
15. Fitting, M. C., Well-Founded Semantics, Generalized, in: *Proc. 1991 Int. Logic Programming Symp.*, MIT Press, 1991, pp. 71–83.
16. Grant, J., Litwin, W., Roussopoulos, N., and Sellis, T. , An Algebra and Calculus for Relational Multidatabase Systems, in: *Proc. 1st Int. Workshop on Interoperability in Multidatabase Syst.*, IEEE Computer Society Press, 1991, pp. 118–124.
17. Kifer, M., and E. Lozinskii, RI: A Logic for Reasoning with Inconsistency, in: *Proc. 4th Symp. on Logic in Comput. Sci.*, Asilomar, CA, 1989, pp. 253–262.
18. Kifer, M., and Subrahmanian, V. S., Theory of Generalized Annotated Logic Programming and its Applications, *J. Logic Programming* 12(4):335–368 (1992). Preliminary version in: *Proc. 1989 North American Conf. on Logic Programming*, MIT Press, 1989.
19. Knuth, D. E., *The Art of Computer Programming, Vol. 1. Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1973.
20. Kullman, P., Master's Thesis, Univ. of Karlsruhe, Germany, 1994.
21. Leach, S., and Lu, J., Computing Annotated Logic Programs, in: *Proc. 11th Int. Conf. on Logic Programming* P. Van Hentenryck (ed.), MIT Press, 1994, pp. 257–271.
22. Lloyd, J. W. *Foundations of Logic Programming*, Springer-Verlag, 1987.
23. Lu, J., Murray, N., and Rosenthal, E., Signed Formulas and Annotated Logics, draft manuscript. Preliminary version in: *Proc. Int. Symp. on Multiple-Valued Logic*, IEEE Computer Society Press, 1993, pp. 48–53.

24. Lu, J., Nerode, A., and Subrahmanian, V. S., Hybrid Knowledge Bases, *IEEE Trans. Knowledge and Data Eng.*, submitted May 1993, revised Jan. 1994, accepted Nov. 1994.
25. Martelli, A., and Montanari, U., An Efficient Unification Algorithm, *ACM Trans. Prog. Lang. and Syst.*, 4(2):258–282 (1982).
26. Nilsson, N. J., Principles of Artificial Intelligence, Morgan Kaufmann, CA, 1980, pp. 76–84.
27. Ramakrishnan, R. Magic Templates: A Spellbinding Approach to Logic Programs, *J. Logic Programming* 11:189–216 (1991).
28. Seki, H., and Itoh, H., A Query Evaluation Method for Stratified Programs under the Extended CWA, in: *Proc. 5th Int. Conf./Symp. on Logic Programming*, K. Bowen and R. Kowalski (eds.), 1989, pp. 195–211.
29. Seki, H., On the Power of Alexander Templates, in: *Proc. 8th ACM Symp. on Principles of Database Syst.*, 1989, pp. 150–159.
30. Shoenfield, J., *Mathematical Logic*, Addison-Wesley, Reading, MA, 1967.
31. Subrahmanian, V. S., Amalgamating Knowledge Bases, *ACM Trans. Database Syst.* 19(2):291–331 (1994).
32. Subrahmanian, V. S., Adah, S., Emergy, R., Rajput, A., Rogers, T. J., and Ross, R., *HERMES: A Heterogeneous Reasoning and Mediator System*, Univ. of Maryland, Technical Report, 1994.
33. Tamaki, H., and Sato, T., OLD Resolution with Tabulation, in: *Proc. 3rd Int. Conf. on Logic Programming*, E. Shapiro (ed.), Springer, 1986, pp. 84–98.
34. van Gelder, A., The Alternating Fixpoint of Logic Programs with Negation, in: *Proc. 8th ACM Symp. on Principles of Database Syst.* 1989, pp. 1–10.
35. Vieille, L. A Database-Complete Proof Procedure Based on SLD-Resolution, in: *Proc. 4th Int. Conf. on Logic Programming*, J.-L. Lassez (ed.), MIT Press, 1987, pp. 74–103.
36. Warren, D. S., Memoing for Logic Programs, *Commun. ACM* 35(3):94–111 (1992).
37. Whang, W. K., Navathe, S. B., and Chakravarthy, S., Logic-Based Approach for Realizing a Federated Information Systems, in: *Proc. 1st Int. Workshop on Interoperability in Multidatabase Syst.*, IEEE Computer Society Press, 1991, pp. 92–100.
38. Wiederhold, G., Jajodia, S., and Litwin, W., Dealing with Granularity of Time in Temporal Databases, in: *Proc. 3rd Nordic Conf. on Advanced Inform. Syst. Eng.*, Lecture Notes in Comput. Sci., Vol. 498, R. Anderson et al. (eds.), Springer-Verlag, 1991, pp. 124–140.
39. Wiederhold, G., Jajodia, S., and Litwin, W., Integrating Temporal Data in a Heterogeneous Environment, in: *Temporal Databases*, Benjamin Cummings, Jan. 1993.
40. Zicari, R., Ceri, S., and Tanca, L., Interoperability between a Rule-Based Database Language and an Object-Oriented Language, in: *Proc. 1st Int. Workshop on Interoperability in Multidatabase Syst.*, IEEE Computer Society Press, 1991, pp. 125–135.