Journal of Intelligen Information Systems, 6, 33–58 (1999)

Query Processing in Annotated Logic Programming: Theory and Implementation*

SONIA M. LEACH sml@cs.brown.edu
Department of Computer Science, Brown University, Providence, RI 02912

JAMES J. LU jameslu@bucknell.edu
Department of Computer Science, Bucknell University, Lewisburg, PA 17837

Abstract. Annotated logic is a formalism that has been applied to a variety of situations in knowledge representation, expert database systems, quantitative reasoning, and hybrid databases [6], [13], [19], [20], [21], [22], [23], [24], [30], [33], [35], [36]. Annotated Logic Programming (ALP) is a subset of annotated logics that can be used directly for programming annotated logic applications [22], [23]. A top-down query processing procedure containing elements of constraint solving, called ca-resolution, is developed for ALPs. It simplifies a number of previously proposed procedures, and also improves on their efficiency. The key to its development is in observing that satisfaction, as introduced originally for ALPs, may be naturally generalized. A computer implementation of ca-resolution for ALPs is described which offers important theoretical and practical insights. Strategies for improving its efficiency are discussed.

Keywords: expert database systems, heterogeneous systems, reasoning under uncertainty

1. Introduction

Annotated Logic is a formalism that can be used as the foundation for a variety of situations in knowledge representation, expert systems, quantitative reasoning, and hybrid databases [6], [13], [19], [20], [21], [22], [23], [24], [30], [33], [35], [36]. As a knowledge representation formalism, annotated logic can be applied to reasoning under uncertain, incomplete, and contradictory knowledge. In hybrid knowledge bases, annotated logic provides a logical framework for integrating heterogeneous systems. In addition, in [23], Kifer and Subrahmanian demonstrated how annotated logic programming can capture certain fragments of temporal reasoning, as well as bilattice logic programming introduced by Fitting [10].

The distinguishing feature of annotated logic is the incorporation of names for truth values directly into the language of the logic. Such signing of formulas has been considered elsewhere including by [32], [14] for analyzing multiple-valued logics, by [5] for evidential reasoning, and by [38] for fuzzy reasoning. In annotated logic, the set of underlying truth values is assumed to form a complete lattice.

As the utility of annotated logic becomes more apparent, the next important set of research questions to address includes the development of proof procedures for automated reasoning in annotated logic, and the examination of issues related to implementing such

* This material is based upon work supported by the NSF under Grant CCR9225037. A preliminary version of this paper appears in the proceedings of the International Conference on Logic Programming, 1994.

procedures. A subset of annotated logic that has been considered most frequently is annotated logic programming, which focuses on *annotated horn clauses* (cf. [19], [6], [22], [23]). Due to the procedural interpretation naturally associated with horn clauses, annotated logic programming can be used to directly implement many annotated logic applications.

The current paper investigates a query processing method, called ca-resolution, for annotated logic programming, and presents a prototype interpreter based on the method. Ca-resolution improves upon both the efficiency and the readability of several existing procedures. The procedure considered in [22], [23] requires the application of two inference rules for processing queries. The additional inference rule has presented difficulties in the development of an efficient top-down query processing procedure. Ca-resolution alleviates the necessity of the additional rule. The key to its development is in observing that the semantics of annotated logic programs can be naturally generalized.

Ca-resolution contains elements of constraint programming [26]. The inclusion of constraint solving mechanisms into logic programming is a topic of considerable interest in recent years [8], [18], [37].[1] The semantics of annotated logic programs extended with constraints have been studied in [23]. Here, we discuss how the mechanism developed in [23] corresponds to *local propagation* [26], and can be used to solve queries in our system.

There is an interesting difference between our uses of constraints and the typical Constraint Logic Programming (CLP) scheme [15]. In CLP, horn clauses are augmented by constraints. At each step of a computation, the interpreter must ensure the solvability of the constraint. Unsolvable constraints cause backtracking. On the other hand, in our system, the solvability of constraints *defines* the satisfiability of a query. Constraints are associated with atoms in the query; they do not exist independently of the atoms. An unsolvable constraint does not cause backtracking, but instead, prompts further searches until a solvable constraint is found. Therefore constraints are continually *modified* until a solvable constraint is constructed. In CLP, an existing constraint does not change. Additional constraints, however, may be added.

We describe what we believe is a first implementation of an annotated logic programming system. Compared to ordinary logic programming, the search space for a proof of a given query in annotated logic programming is more complex. Certain issues absent in classical horn clause programming surface in annotated logics programming. We illustrate these complexities and examine possible strategies for reducing the search space. Of special interest are those strategies that systematically utilize special properties embedded in the underlying lattice of truth values. It has long been argued that such semantically based strategies are necessary in improving the effectiveness of inference techniques [40], [7]. One promising restriction strategy along this line is briefly discussed Section 5.3.2.

2. Annotated Logic Programs

2.1. *The Lattice Δ and Annotations*

Underlying any system of annotated logic is a lattice Δ of objects. We denote the associated ordering by $\preceq$. Generally, Δ is assumed to be a complete upper semilattice. More interesting however is the case when Δ is assumed to be a complete lattice. This is the assumption taken in the current paper. In practice, the distinction between the two is slight as we may often "complete" the upper semilattice by adding a bottom element.

The least upper bound and the greatest lower bound operators are denoted $\sqcup$ and $\sqcap$, respectively, while the top and the bottom elements are denoted $\top$ and $\bot$, respectively. Objects in Δ are used for "signing" formulas in annotated logics. As mentioned in [23], elements in Δ may be thought of as confidence factors, as degrees of belief, or as truth values. The requirement that Δ forms a lattice stems from the desire to use Δ for modeling certain epistemic concepts such as inconsistency [6], [21], and evidence [24]. More recently, Subrahmanian [36] extended the scope of Δ to represent names of distributed databases, where databases that are "higher" in the lattice are regarded as supervisory databases – otherwise known as mediators [39] – of "lower" databases .

The lattice Δ may be viewed as a constraint domain. To that end, we assume the existence of a first order language Σ whose function and predicate symbols are interpreted over Δ. As in [23], we consider only total, computable continuous functions over Δ. In particular, included in the function symbols are the operators $\sqcup$, $\sqcap$, and included in the predicate symbols is the ordering relation $\preceq$. A *term* built in the usual way from the non-logical symbols of the language Σ is called an *annotation.* To maintain consistency with the terminology used in [23], we refer to a constant symbol in the language Σ as a c-annotation, a variable in Σ as a v-annotation, and a term involving a function symbol as a t-annotation. To ease the presentation, we do not distinguish between objects of Δ and terms of the language Σ whose denotations are in Δ. A *constraint* is a formula constructed in the usual way from atomic formulas and logical connectives.

Of particular interest is the class of *normal constraints*, which are constraints of the form

$$\kappa_1 \preceq \tau_1, ..., \kappa_m \preceq \tau_m$$

where each κ is a c- or v-annotation, each τ is an annotation, and if κ_i is a variable, then it does not occur in $\tau_1, ..., \tau_i$. For example, the constraint $V \preceq f(W), W \preceq g(V)$ is not a normal constraint since any ordering of the inequalities violates the required condition. The simplest lattice, used in the original study on logic programming for reasoning with inconsistent information [6], is the lattice FOUR shown in Figure 1.

Not every application of annotated logic involves the use of a lattice whose elements represent truth values. For example, Krishnaprasad and Kifer used lattices where the objects correspond to evidence [24]. Objects used in these lattices include `bird`, `Nixon`, and `Pennsylvanian`. In [13], a lattice of adjectives are used to denote user preference in answers to queries.

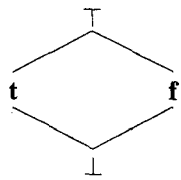

Figure 1. The Complete Lattice FOUR.

A still more complex lattice is the lattice $\mathcal{UNC}$ on which hybrid knowledge bases are formalized [31], [36]. The lattice $\mathcal{UNC}$ is used for reasoning about time and uncertainty. Formally, let $\mathbf{R}^+$ denote the set of non-negative real numbers, then $\mathcal{UNC}$ is the function space $\mathbf{R}^+ \rightarrow [0,1]$ ordered by

$$f_1 \preceq f_2 \text{ iff } f_1(x) \leq f_2(x) \text{ for all } x \in R^+.$$

The top element of the lattice, denoted $f_\top$, is the function that assigns every element in $\mathbf{R}^+$ to 1, while the bottom element $f_\bot$ maps each real number to 0. Observe that elements in $\mathcal{UNC}$ are functions. More interestingly however, is that they correspond to fuzzy subsets of $\mathbf{R}^+$ [4]. Intuitively, given a function f in $\mathcal{UNC}$, the domain $\mathbf{R}^+$ represents time points, while the range $[0,1]$ represents uncertainty.

2.2. *Annotated Logic Program Syntax*

Based on Δ, we may define an annotated logic. We assume a first-order language L. Atoms are built from constants, variables, and function and predicate symbols in the usual way (cf. [27]). Suppose A is an atom, and α is an annotation. Then $A : \alpha$ is an *annotated atom.* Suppose α is a c-annotation. Then $A : \alpha$ is said to be *c-annotated.* Similarly, $A : \alpha$ may be said to be *v-annotated* or *t-annotated.* If $A : \alpha$ is an annotated atom and $B_1 : \mu_1, ..., B_k : \mu_k$ are c- or v-annotated atoms, then

$$A : \alpha \leftarrow B_1 : \mu_1, ..., B_k : \mu_k$$

is an *annotated clause.* $A : \alpha$ is called the *head* of the clause, and $B_1 : \mu_1, ..., B_k : \mu_k$ is called the *body* of the clause. All variables (object or annotation) appearing in the clause are implicitly universally quantified at the beginning of the clause. A set of annotated clauses is called an *annotated logic program* (ALP). Suppose $A : \alpha$ is an annotated atom where α is a variable free t-annotation. We may replace μ by the lattice element denoted by α.

Example: The lattice FOUR can be used by ALPs for expressing inconsistent information. As an example taken from [21], the following ALP program asserts intuitively tweety is a bird and is not a bird, and that john receives a grade of 'A'.

$flies(tweety) : \mathbf{t} \leftarrow$
$flies(tweety) : \mathbf{f} \leftarrow$

$grade(john, "A") : \mathbf{t} \leftarrow$

As Kifer and Lozinskii argued, such a set of knowledge may represent beliefs held by a reasoning agent R. Knowledge gathered based on different evidence may therefore introduce conflicting beliefs, such as $flies(tweety) : \mathbf{t}$ (i.e. R believes $tweety$ flies) and $flies(tweety) : \mathbf{f}$ (i.e. R believes $tweety$ does not fly). As will be seen when we introduce the semantics of annotated logic programming, unlike classical knowledge, such an inconsistency does not affect R's ability to reason rationally. □

Example: The following is an example of a hybrid knowledge base rule using the lattice $\mathcal{UNC}$. We adopt the Prolog convention of denoting variables by upper-case letters, and constants in the database by lower-case letters. Intuitively, the rule expresses the statement: "If the robot is at location (X, Y) with certainty V at time t, and it is certain that the robot is moving north at a constant speed of 0.2, then the robot will be at location $(X, Y + 1)$ with certainty V at time $(t + 1)$" may be expressed as

$$robot\_at(X, Y + 1) : (V, \{T + 1\}) \leftarrow robot\_at(X, Y) : (V, \{T\}) \,\&\, move(X, north, 0.2) : (1, \mathbf{R}^+).$$

Here, each of $(V, \{T\})$ and $(V, \{T + 1\})$ represent functions in $\mathcal{UNC}$ – formally called ℓ-representations. The operator $+1$ used inside the annotation $(V, \{T + 1\})$ is an unary annotation function f that maps ℓ-representations to ℓ-representations, i.e. $f(\mu, S) = (\mu, S')$ where $S' = \{\alpha + 1 | \alpha \in S\}$. More will be discussed on ℓ-representation in Section 6. □

2.3. *Annotated Logic Programs Semantics*

As usual in the treatment of the semantics of logic programs, attention is restricted to only Herbrand interpretations. Thus the domain of our interpretations consists of the set of variable free terms built out of the constants and the function symbols of the language L.

Definition (satisfaction). An interpretation I is a mapping from the set of variable free atoms (i.e. ground atoms) to Δ. It is said to satisfy

1. the ground c-annotated atom $A : \mu$ iff $\mu \preceq I(A)$.
2. the ground c-annotated conjunction $F_1, ..., F_n$ if it satisfies each of F_1 through F_n.
3. the ground c-annotated clause $A \leftarrow B_1, ..., B_n$ if whenever it satisfies $B_1, ..., B_n$, it also satisfies A.
4. the c-annotated clause $A \leftarrow B_1, ..., B_n$ if it satisfies each ground instance, obtained by replacing each object variable by a ground term. Different occurrences of the variable must be replaced with the same term.
5. the annotated clause $A \leftarrow B_1, ..., B_n$ if it satisfies each c-annotated instance, obtained by replacing each annotation variable by an element in Δ. Different occurrences of the variable must be replaced with the same object.

The symbol $\models$ is used to denote both satisfaction and logical consequence.

Example: [23] Let Δ be the lattice FOUR. Take P to be the ALP below.
P_1 $p : V \leftarrow q(X) : V$
P_2 $q(a) : \mathbf{t} \leftarrow$
P_3 $q(b) : \mathbf{f} \leftarrow$
Among the ground c-annotated instances of P_1 are:
$p : \mathbf{t} \leftarrow q(a) : \mathbf{t}$
$p : \mathbf{f} \leftarrow q(b) : \mathbf{f}$
According to the definition for satisfaction, any interpretation that satisfies the program must assign to p an element in FOUR that is greater than or equal to $\mathbf{t}$ and $\mathbf{f}$. Clearly, the only such element is $\top$ itself. Hence $P \models p : \top$. □

The previous example illustrates how annotated logics may be used as a *paraconsistent* logic – logic that tolerates inconsistent information without entailing all possible conclusions. In particular, referring back to the *tweety* example in Section 2.2, it is straightforward to see that the inconsistent belief held by the reasoner R regarding the flying ability of *tweety* does not allow R to draw arbitrary conclusions concerning the grade received by *john*.

Variables that occur in the head but not in the body of clauses are called *free variables*. Conversely, variables occurring only in the body but not in the head of a clause are called *local variables*. Note that as in the case of object variables, annotation variables in clauses are implicitly universally quantified. An immediate consequence of the semantics of annotated logic programs is that there is no loss of generality in assuming that no annotated clause contains a free annotated variable. Consider for example the ALP $p : V \leftarrow$ over FOUR. The program is equivalent to the four c-annotated clauses $p : \top \leftarrow$, $p : \mathbf{t} \leftarrow$, $p : \mathbf{f} \leftarrow$, and $p : \bot \leftarrow$, obtained by instantiating the variable V with all possible values in FOUR. As the first clause subsumes the last three, the original ALP is equivalent to the single c-annotated unit clause $p : \top \leftarrow$. It is easy to verify, with the assumption that annotated functions are continuous, and thus monotonic, that any free annotated variable may be replaced by the top element of the lattice Δ without changing the semantics of the program. This observation is noted in [23].

On the other hand, given the ALP

$$p : \top \leftarrow q : V, r : V$$

over the lattice FOUR, we may replace V by the four lattice elements in FOUR to yield the equivalent c-annotated ALP below.
$p : \top \leftarrow q : \top, r : \top$
$p : \top \leftarrow q : \mathbf{t}, r : \mathbf{t}$
$p : \top \leftarrow q : \mathbf{f}, r : \mathbf{f}$
$p : \top \leftarrow q : \bot, r : \bot$
The last clause says any interpretation that assigns to q and r at least the truth value $\bot$ must assign to p the value $\top$. As any interpretation assigns to each ground atom at least $\bot$, the condition $q : \bot, r : \bot$ is trivially satisfied. Thus, we may drop the antecedent of the clause. The result is a clause that subsumes the first three c-annotated clauses.

Therefore the original ALP is equivalent to $p : \top \leftarrow$. This analysis generalizes to arbitrary ALPs, and we may assume without loss of generality that ALPs do not contain local variables.

PROPOSITION 1 Suppose P is an ALP. Let P_1 be the ALP obtained from P by first eliminating from each clause each annotated atom that contains a local annotated variable, and then replacing each free variable by $\top$. Then P and P_1 have the same models.

3. Query Processing in ALPs

The most popular technique for answering queries in logic programs is based on SLD-resolution [27]. The main advantage in using an SLD-style proof procedure is, as noted in [23], the choice of clauses that need to be considered at each deduction step is restricted to the current goal and the program clauses. Thus far, finding an SLD-style proof procedure has proven elusive for ALPs [22], [23]. The difficulty lies in the need to compute *reductants* (defined below). First we give some necessary definitions.

An expression of the form $\leftarrow \Xi \parallel A_1 : \mu_1, ..., A_n : \mu_n$ where Ξ is a constraint, is called a *constrained query*. In [23], the constraint part of a constrained query need not be restricted to lattice constraints. However, here we focus only on lattice constraints since they must be handled for any system that includes an ALP component. The symbol $\parallel$ denotes conjunction, and is used to delineate the constraint part of the query from the atomic part. The notion of satisfaction for constrained queries is immediate. Following are two inference rules for answering query in ALPs, introduced in [23].

Definition (annotated resolution). Suppose $C = A : \rho \leftarrow B_1, \ldots, B_n$ is a clause and $Q = \leftarrow \Xi \parallel A_1 : \alpha_1, \ldots, A_m : \alpha_m$ is a constrained query with no variables in common with C. Moreover, suppose that for some $1 \leq i \leq m$, A_i is unifiable with A via mgu θ. Then the annotated resolvent of Q and C with respect to A_i is the query

$$\leftarrow \alpha_i \preceq \rho, \Xi \parallel (A_1 : \alpha_1, \ldots, A_{i-1} : \alpha_{i-1}, B_1, \ldots, B_n, A_{i+1} : \alpha_{i+1}, \ldots, A_m : \alpha_m)\theta.$$

Definition (reduction). Assume two annotated clauses

$A_1 : \mu_1 \leftarrow B_1 : \beta_1, ..., B_n : \beta_n$

$A_2 : \mu_2 \leftarrow C_1 : \gamma_1, ..., C_m : \gamma_m$

where A_1 and A_2 are unifiable via mgu θ. The *reductant* of the two clauses is the clause

$(A_1 : \sqcup\{\mu_1, \mu_2\} \leftarrow B_1 : \beta_1, ..., B_n : \beta_n, C_1 : \gamma_1, ..., C_m : \gamma_m)\theta$.

In terms of resolution theorem proving, an SLD-proof procedure does not exist for ALPs due to the incompatibility of the two rules of inference, annotated resolution and reduction, with the linear restriction strategy [28]. Kifer and Subrahmanian circumvent this difficulty by specifying that a deduction consists of only applications of annotated resolution. However, each inference may involve a resolution with an annotated clause obtained by implicit applications of the reduction procedure.

Example: Consider the program from the example in Section 2.3. We have shown that the program entails $p : \top$. Hence the query $\leftarrow p : \top$ should have a refutation. Observe

that if we do not use the reduction inference rule, the only possible annotated resolvent that can be obtained from the query is $\leftarrow \top \preceq V \parallel q(X) : V$. Resolving the new query with P_2 yields the query that contains the unsolvable constraint $\top \preceq V, V \preceq \mathbf{t}$. Similarly, resolving with P_3 produces an unsolvable constraint. Hence no proof can be found.

Using the reduction inference on the other hand, we first compute the reductant $p : \sqcup\{V_1, V_2\} \leftarrow q(X_1) : V_1, q(X_2) : V_2$ from two variants of P_1. This resolves with the original query, resulting in the following constrained query.

$$\leftarrow \top \preceq \sqcup\{V_1, V_2\} \parallel q(X_1) : V_1, q(X_2) : V_2$$

Resolving the new query with P_2 and P_3 in succession yields the constrained query $\leftarrow \top \preceq \sqcup\{V_1, V_2\}, V_1 \preceq \mathbf{t}, V_2 \preceq \mathbf{f}$. As the constraint in the query is solvable, we have a refutation. □

Unfortunately, the implicit use of reduction causes difficulties since a proof, which consists of only annotated resolution steps, may contain many deductions with clauses that are not in the original program. This makes the proof difficult to read. Moreover, application of the reduction inference is expensive since it must be allowed to occur any time during a deduction, thus greatly expanding the search space. The situation is analogous to the use of the restart rule in disjunctive logic programming [34].

The key in overcoming the difficulties of annotated resolution and reduction is in observing that the semantics of annotated logics are naturally generalized to set membership. This is captured by the use of *constrained annotations*, introduced next.

3.1. *Constrained Annotated Atoms and CA-Resolution*

A *constrained annotation* is a pair $\langle \mu, S \rangle$ where μ is an annotation and S is a finite set of annotations. A *constrained annotated atom* is an expression $A : C$ where A is an atom and C is a constrained annotation.

Definition (satisfaction of constrained atom). An interpretation I satisfies a ground constrained annotated atom $A : \langle \mu, S \rangle$ iff $I(A) \in (\uparrow \mu) \cup (\uparrow \sqcup S)^c$.

Given a lattice element γ, $\uparrow\gamma$ denotes the upset of γ (cf. [9]). In other words, $\uparrow\gamma = \{\delta \in \Delta | \gamma \preceq \delta\}$. Set complementation of a set A is denoted A^c. Intuitively, a constrained annotated atom says that the truth value of A is either greater than or equal to μ, or is not greater than or equal to the least upper bound of S.

Example: Suppose I is the interpretation that assigns to p the value $\mathbf{t}$ in FOUR. Then I satisfies the constrained annotated atom $p : \langle \mathbf{t}, \{\} \rangle$, but it does not satisfy $p : \langle \top, \{\mathbf{t}\} \rangle$ since $(\uparrow\top) \cup (\uparrow \sqcup \{\mathbf{t}\})^c = \{\top, \mathbf{f}, \bot\}$, which does not contain $\mathbf{t}$. □

The following is immediate from the definition since $\mu \preceq \sqcup S$ iff $\uparrow \mu \cup (\uparrow \sqcup S)^c = \Delta$, for any $\mu \in \Delta$ and $S \subseteq \Delta$.

PROPOSITION 2 Let I be an interpretation and let $A : \langle \mu, S \rangle$ be a ground constrained atom such that $\mu \preceq \sqcup S$. Then I satisfies $A : \langle \mu, S \rangle$.

The definition of satisfaction of constrained atoms is the key generalization in the semantics of ALPs that enables us to introduce our proof procedure below. Note that the original definition of satisfaction of annotated atoms naturally fits into this more general definition. Since Δ is a lattice, $\sqcup\{\} = \bot$. If we have a ground constrained annotated atom $A : \langle \mu, \{\} \rangle$, then a satisfying interpretation I must assign, according to definition, an element in $(\uparrow \mu) \cup (\uparrow \bot)^c$ to A. As $\uparrow \bot = \Delta$, it follows that $(\uparrow \bot)^c = \{\}$. Hence $(\uparrow \mu) \cup (\uparrow \bot)^c = \uparrow \mu$ and $I(A) \in \uparrow \mu$, the same condition as $\mu \preceq I(A)$.

Based on this observation, we see that constrained annotated atoms generalize ordinary annotated atoms since $A : \mu$ may be regarded equivalently as the special constrained annotated atom $A : \langle \mu, \{\} \rangle$. A query $\leftarrow A_1 : \mu_1, ..., A_n : \mu_n$ can therefore be rewritten

$$\leftarrow A_1 : \langle \mu, \{\} \rangle, ..., A_n : \langle \mu_n, \{\} \rangle$$

without changing its meaning. In general, we call a query consisting of only constrained annotated atoms a ca-query (constrained annotation query).

Let C be a conjunction of constrained annotated atoms $A_1 : \langle \mu_1, S_1 \rangle, ..., A_n : \langle \mu_n, S_n \rangle$. The *constraint associated with* C, denoted Ξ_C, is the lattice constraint

$$\mu_1 \preceq \sqcup S_1, ..., \mu_n \preceq \sqcup S_n.$$

The following relates the solvability of Ξ_Q and the unsatisfiability of $\leftarrow Q$.

PROPOSITION 3 A ca-query $\leftarrow Q$ is unsatisfiable iff Ξ_Q is solvable with respect to Δ.

Proof: $\leftarrow Q$ is unsatisfiable iff for each interpretation I, $\leftarrow Q\theta$ is unsatisfiable for some ground substitution θ iff I satisfies $Q\theta$. Note θ is a substitution that substitutes for each object variable, as well as for each annotation variable. Let θ_1 represent the substitution θ restricted to all and only the annotation variables.

Suppose $\leftarrow Q\theta$ is of the form $\leftarrow A_1 : \langle \mu_1, S_1 \rangle, ..., A_n : \langle \mu_n, S_n \rangle$. By the previous paragraph, for every interpretation I, I satisfies $A_i : \langle \mu_i, S_i \rangle$ for each $i = 1, ..., n$. By definition, $I(A_i) \in \uparrow \mu_i \cup (\uparrow \sqcup S_i)^c$. As this holds for every I, it must follow that $\uparrow \mu_i \cup (\uparrow \sqcup S_i)^c = \Delta$ for otherwise, we may find an I such that $I(A_i) \notin \uparrow \mu_i \cup (\uparrow \sqcup S_i)^c$ (take I to be one where $I(A_i) \in (\Delta - (\uparrow \mu_i \cup (\uparrow \sqcup S_i)^c))$). Hence $\mu_i \preceq \sqcup S_i$ for each $i = 1, ..., n$, which is exactly the lattice constraint represented by $\Xi_Q\theta_1$. It follows that Ξ_Q is solvable with respect to Δ and θ_1 is a solution.

For the reverse direction, θ_1 is a solution to the constraint Ξ_Q. Let $A : \langle \mu, S \rangle$ denote a typical, but arbitrary constrained annotated atom in Q. We have $\mu\theta_1 \preceq (\sqcup S)\theta_1$. By Proposition 2, I satisfies $(A : \langle \mu, S \rangle)\theta$ for every interpretation I, and hence $\leftarrow Q$ is unsatisfiable. ■

Definition (ca-resolution). Let $A : \mu \leftarrow B_1 : \beta_1, ..., B_n : \beta_n$ be an annotated clause (with ordinary annotations) that does not share any object or annotation variables with the ca-query $\leftarrow A_1 : \mathcal{D}_1, ..., A_m : \mathcal{D}_m$, and suppose A_i and A are unifiable via mgu θ, for some $i \in \{1, ..., m\}$. Assuming that $\mathcal{D}_i$ is the constrained annotation $\langle \gamma, T \rangle$, then the ca-query

$$\leftarrow (A_1 : \mathcal{D}_1, ..., A_{i-1} : \mathcal{D}_{i-1}, A_i : \langle \gamma, T \cup \{\mu\} \rangle, B_1 : \langle \beta_1, \{\} \rangle, ..., B_n : \langle \beta_n, \{\} \rangle, A_{i+1} : \mathcal{D}_{i+1}, ..., A_m : \mathcal{D}_m)\theta$$

is a ca-resolvent of the given clause and ca-query.

Note that in creating the ca-resolvent, we first translate each annotated atom from the body of the annotated clause into an equivalent constrained annotated atom.

A deduction of a ca-query from a given ALP and an initial ca-query using ca-resolution is defined in the usual way. We call such a deduction a *ca-deduction*. A ca-deduction of a ca-query from an ALP is a *ca-proof* if the last clause in the deduction is unsatisfiable.

Example: Recall the program in the example in Section 2.3. The initial query $\leftarrow p : \top$ may be represented equivalently as $\leftarrow p : \langle \top, \{\} \rangle$. A ca-proof of the query is shown below. The constraint associated with each ca-query is displayed below the query.

$Q_0 :\leftarrow p : \langle \top, \{\} \rangle$ (Initial Query)
Constraint: $\top \preceq \bot$

$Q_1 :\leftarrow p : \langle \top, \{V_1\} \rangle, q(X_1) : \langle V_1, \{\} \rangle$ (ca-resolvent of Q_0 and P_1)
Constraint: $\top \preceq V_1, V_1 \preceq \bot$

$Q_2 :\leftarrow p : \langle \top, \{V_1\} \rangle, q(a) : \langle V_1, \{\mathbf{t}\} \rangle$ (ca-resolvent of Q_1 and P_2)
Constraint: $\top \preceq V_1, V_1 \preceq \mathbf{t}$

$Q_3 :\leftarrow p : \langle \top, \{V_1, V_2\} \rangle, q(X_2) : \langle V_2, \{\} \rangle, q(a) : \langle V_1, \{\mathbf{t}\} \rangle$ (ca-resolvent of Q_2 and P_1)
Constraint: $\top \preceq \sqcup\{V_1, V_2\}, V_1 \preceq \mathbf{t}, V_2 \preceq \bot$

$Q_4 :\leftarrow p : \langle \top, \{V_1, V_2\} \rangle, q(b) : \langle V_2, \{\mathbf{f}\} \rangle, q(a) : \langle V_1, \{\mathbf{t}\} \rangle$ (ca-resolvent of Q_3 and P_2)
Constraint: $\top \preceq \sqcup\{V_1, V_2\}, V_1 \preceq \mathbf{t}, V_2 \preceq \mathbf{f}$

We may verify that of the constraints associated with each of the queries in the above deduction, only the last one is solvable with respect to FOUR. Indeed, this is exactly the same constraint derived using reduction and annotated resolution shown in the earlier example (Section 3). Therefore in one sense, the use of ca-resolution amounts to an incremental computation of reduction. □

A point of interest worth mentioning is how constraints are used differently in ca-resolution as compared to conventional CLP systems. Similar to classical logic programming, a ca-proof in ALP is obtained when an unsatisfiable query is derived. In a ca-deduction, the determination of whether a query $\leftarrow A_1 : \langle \mu_1, S_1 \rangle, ..., A_n \langle \mu_n, S_n \rangle$ is unsatisfiable depends on the solvability of the associated constraint $\mu_1 \preceq \sqcup S_1, ..., \mu_n \preceq \sqcup S_n$ (cf. Proposition 3). The unsolvability of this constraint does not cause backtracking, as can be seen in the example above. The first ca-resolvent of the proof is a satisfiable ca-query since there is no solution to the lattice constraint $\top \preceq \sqcup\{V_1\}, V_1 \preceq \bot$. In CLP, only solvable constraints are allowed to appear in each query of a deduction. A related difference in our approach is that constraints are modified during a deduction. The above unsolvable constraint associated with the first ca-resolvent in the example is modified to $\top \preceq \sqcup\{V_1\}, V_1 \preceq \mathbf{t}$ in the second ca-resolvent. The goal of a ca-deduction therefore can be viewed as searching for a satisfiable constraint. In CLP, an existing constraint remains throughout a deduction.

3.2. *Completeness Issues*

The soundness and the completeness of ca-resolution is proved in this section. Completeness holds only for programs that possess the *fixpoint reachability property*, discussed in [23]. This restriction is not a drawback of ca-resolution. Rather, it is a general consequence of the semantics of ALPs. Intuitively, the fixpoint reachability property ensures that the least model of a given ALP is recursively enumerable. Clearly, failing this minimal condition, no query processing procedure can be expected to handle arbitrary queries with reasonable efficiency.

In this paper, we are not concerned with the conditions under which an ALP possesses a computable model. Such conditions have been investigated in detail in other work [22], [23], [31]. Instead, we show that ca-resolution is sufficiently general to handle any ALP that annotated resolution and reduction can handle. More formally, we say an inference rule R is *relatively complete* if given any ALP P and query $\leftarrow Q$, whenever there is a proof of $\leftarrow Q$ from P using annotated resolution and reduction, then there is proof of $\leftarrow Q$ from P using R. Clearly, a consequence of relative completeness is that ca-resolution is complete for any class of ALP for which annotated resolution and reduction is complete.

LEMMA 1 Suppose I satisfies both the ground clause $A : \mu \leftarrow B_1 : \mu_1, ..., B_n : \mu_n$ and the ground query $\leftarrow Q_1 : \langle \gamma_1, S_1 \rangle, ..., Q_m : \langle \gamma_m, S_m \rangle$ where $A = Q_i$ (i.e. they are identical atoms). Then I satisfies the ca-resolvent

$$\leftarrow Q_1 : \langle \gamma_1, S_1 \rangle, ..., Q_i : \langle \gamma_i, S_i \cup \{\mu\} \rangle, B_1 : \langle \mu_1, \{\} \rangle, ..., B_n : \langle \mu_n, \{\} \rangle, \\ ..., Q_m : \langle \gamma_m, S_m \rangle.$$

Proof: If I does not satisfy $B_1 : \mu_1, ..., B_n : \mu_n$, then the result follows. Similarly if I does not satisfy any $Q_j : \langle \gamma_j, S_j \rangle$ where $i \neq j$, then the result follows. Otherwise $I \models A : \mu$ and $I \not\models Q_i : \langle \gamma_i, S_i \rangle$. By the first condition, $\mu \preceq I(A)$. Therefore $I(A) \in \uparrow \mu$. By the second condition, $I(Q_i) \notin \uparrow \gamma_i \cup (\uparrow \sqcup S_i)^c$. Equivalently, $I(Q_i) \in \Delta - (\uparrow \gamma_i \cup (\uparrow \sqcup S_i)^c)$. Thus we have $I(Q_i) \in (\uparrow \gamma_i)^c \cap (\uparrow \sqcup S_i)$.

It follows that, as $Q_i = A$, $I(Q_i) \in (\uparrow \gamma_i)^c \cap (\uparrow \sqcup S_i) \cap \uparrow \mu$. Since $(\uparrow \sqcup S_i) \cap \uparrow \mu = \uparrow \sqcup (S_i \cup \{\mu\})$, we may conclude that $I(Q_i) \in (\uparrow \gamma_i)^c \cap \uparrow \sqcup (S_i \cup \{\mu\})$. By the reverse of the argument in the previous paragraph, $I(Q_i) \notin \uparrow \gamma_i \cup (\uparrow \sqcup (S_i \cup \{\mu\}))^c$. Hence I does not satisfy $Q_i : \langle \gamma_i, S_i \cup \{\mu\} \rangle$. ■

Based on this lemma, we may conclude that ca-resolution preserves soundness. It follows that any ca-deduction derives only sound conclusions from an ALP and a starting query. In particular, if we obtain a ca-proof, it must be that the given program and query are not satisfiable.

COROLLARY 1 *(soundness)* Suppose P is an ALP and there is a ca-proof of $\leftarrow Q$ from P. Then $P \cup \{\leftarrow Q\}$ is unsatisfiable.

For the proof of relative completeness, we show a translation from each deduction using annotated resolution and reduction to a ca-deduction.

LEMMA 2 Given a query $\leftarrow Q$ and an ALP P. Suppose there is a deduction of the query $\leftarrow \Xi$, where Ξ is a constraint, using annotated resolution and reduction from $P \cup \{\leftarrow Q\}$. Let $\leftarrow Q^*$ be the equivalent ca-query obtained from $\leftarrow Q$ by replacing each atom $A : \mu$ in Q by $A : \langle \mu, \{\} \rangle$. Then there is a ca-deduction of a query $\leftarrow L$ from $\leftarrow Q^*$ where L is a conjunction of constrained atoms and $\Xi_L = \Xi$.

Proof: The proof is by induction on the length n of the deduction.
(Base Case) $n = 1$. The original query $\leftarrow Q$ has the form $\leftarrow A : \mu$ and there is a reductant

$$B : \beta \leftarrow$$

such that A and B are unifiable with mgu θ. The annotated resolvent is $\leftarrow \mu \preceq \beta$. The clause $B : \beta \leftarrow$ is derived by applying the reduction inference rule to m program clauses. Suppose the program clauses involved are the following:

$C_1 \equiv_i B_1 : \beta_1 \leftarrow$
...
$C_m \equiv B_m : \beta_m \leftarrow$

We have $\beta = \sqcup\{\beta_1, ..., \beta_m\}$. Thus a ca-deduction from $\leftarrow Q^*$ may be obtained as follows.

$Q_0 \leftarrow A : \langle \mu, \{\} \rangle$	Initial Query
$Q_1 \leftarrow A_1 : \langle \mu, \{\beta_1\} \rangle$	ca-resolution of Q_0 and C_1
...	
$Q_m \leftarrow A_m : \langle \mu, \{\beta_1, ..., \beta_m\} \rangle$	ca-resolution of Q_{m-1} and C_m

The constraint associated with the single atom is $\mu \preceq \sqcup\{\beta_1, ..., \beta_m\}$, which is equivalent to $\mu \preceq \beta$ as $\beta = \sqcup\{\beta_1, ..., \beta_m\}$.
(Inductive Case) The original query $\leftarrow Q$ has the form

$$\leftarrow A_1 : \mu_1, ..., A_m : \mu_j$$

and there is a reductant

$$B : \beta \leftarrow B_1 : \beta_1, ..., B_k : \beta_k$$

such that A_i and B are unifiable via mgu θ. The annotated resolvent has the form

$$\leftarrow \mu_i \preceq \beta \parallel (A_1 : \mu_1, ..., A_{i-1} : \mu_{i-1}, B_1 : \beta_1, ..., B_k : \beta_k, A_{i+1} : \mu_{i+1}, ..., A_j : \mu_j)\theta.$$

Let $\leftarrow \Xi$ be an annotated resolvent obtained by a deduction using annotated resolution and reduction of the query

$$\leftarrow (A_1 : \mu_1, ..., A_{i-1} : \mu_{i-1}, B_1 : \beta_1, ..., B_k : \beta_k, A_{i+1} : \mu_{i+1}, ..., A_j : \mu_j)\theta.$$

By the induction hypothesis, there is a ca-deduction D of a ca-query $\leftarrow L$ from the above query, appropriately translated into its equivalent ca-query, such that $\Xi_L = \Xi$.

The reductant $B : \beta \leftarrow B_1 : \beta_1, ..., B_k : \beta_k$ is obtained from m program clauses using the reduction inference rule. We will denote these clauses $C_1, ..., C_m$. By similar argument as the base case, there is a ca-deduction from $\leftarrow Q^*$ of

$$\begin{aligned} \leftarrow \quad & (A_1 : \langle \mu_1, \{\}\rangle, ..., A_{i-1} : \langle \mu_{i-1}, \{\}\rangle, \\ & A_i : \langle \mu_i, \{\beta_1, ..., \beta_m\}\rangle, B_1 : \langle \beta_1, \{\}\rangle, ..., B_k : \langle \beta_k, \{\}\rangle, \\ & A_{i+1} : \langle \mu_{i+1}, \{\}\rangle, ..., A_j : \langle \mu_j, \{\}\rangle)\theta. \end{aligned}$$

Here $\beta_1, ..., \beta_m$ are the annotations of the respective heads from $C_1, ..., C_m$. The constraint associated with the atom $A_i : \langle \mu_i, \{\beta_1, ..., \beta_m\}\rangle$ is $\mu_i \preceq \sqcup\{\beta_1, ..., \beta_m\}$. Again this is the same as the constraint $\mu_i \preceq \beta$. From the above ca-resolvent, perform the same ca-deduction steps as D, we obtain the ca-query $\leftarrow L, A_i : \langle \mu_i, \{\beta_1, ..., \beta_m\}\rangle$, and the associated constraint is $\Xi, \mu_i \preceq \beta$. ■

In both the base case and the inductive case of the above proof, if the constraint Ξ in the statement of the above lemma is solvable, then the given deduction is a refutation. It follows that the corresponding ca-deduction is a ca-proof of the original query. We have the following corollary.

COROLLARY 2 *(completeness)* Suppose there is a proof using annotated resolution and reduction from $P \cup \{\leftarrow Q\}$ and $\leftarrow Q^*$ is the equivalent ca-query obtained from $\leftarrow Q$ by replacing each atom $A : \mu$ in Q by $A : \langle \mu, \{\}\rangle$. Then there is a ca-proof of $P \cup \{\leftarrow Q^*\}$.

Corollary 1 and Corollary 2 together proves Theorem 1.

THEOREM 1 CA-resolution is sound. Moreover, it is relatively complete.

Using ca-resolution, it is not necessary to compute reductants. This enhances the readability of proofs as they contain only program clauses, and increases efficiency by eliminating the expensive reduction inference rule. Ca-resolution thus represents the first complete top-down procedure for handling ALPs in its full generality.

In classical logic programming, an important theoretical result regarding SLD-resolution is the *independence of the computation rule* [27]. A computation rule specifies the literal in a query to resolve on at each step of a deduction. The result tells us that any computation rule will suffice. This enables for example, Prolog to safely choose the leftmost literal to resolve on at each step.[2] Unfortunately, in the case of ca-resolution, selecting systematically the leftmost or the rightmost literal causes incompleteness, even if we select clauses fairly.

Example: Suppose we adopt the selection rule of choosing the leftmost literal in a query. Consider the ALP over the lattice $[0, 1]$, where the ordering $\preceq$ is the relation $\leq$ on reals,

$q : (V + W) \leftarrow p : V, r : W$
$p : 0.2 \leftarrow$
$r : 0.2 \leftarrow$

and the ca-query $\leftarrow q : \langle 0.4, \{\}\rangle$. The first ca-resolvent is $\leftarrow q : \langle 0.4, \{(V + W)\}\rangle, p : \langle V, \{\}\rangle, r : \langle W, \{\}\rangle$. The associated constraint $0.4 \preceq (V + W), V \preceq 0, W \preceq 0$ is not

solvable with respect to $[0,1]$. It is easy to see that one may continue to resolve the leftmost literal with the first clause, but never obtain a solvable constraint.

Suppose on the other hand, we select the rightmost literal to resolve on. From the initial ca-resolvent we obtain the ca-resolvent $\leftarrow q : \langle 0.4, \{(V+W)\}\rangle, p : \langle V, \{\}\rangle, r : \langle W, \{0.2\}\rangle$. Again the associated constraint $0.4 \preceq (V+W), V \preceq 0, W \preceq 0.2$ is not solvable. Resolving on the rightmost literal in the next deduction step yields the same ca-query.

One can see that no fixed position selection of literals in this case can produce a proof of the original query, even though clearly $q : \langle 0.4, \{\}\rangle$ is a logical consequence of the program. Thus the independence of the computation rule does not hold for ca-resolution. One must select literals "fairly", in the sense that each literal will be tried eventually. This will be considered in Section 5.2 when the ALP interpreter is discussed.

□

3.3. *Unsatisfiability of CA-queries and Normal Constraints*

A key to implementing ca-resolution lies in determining solvability of constraints associated with queries. Kifer and Subrahmanian have devised an algorithm for determining satisfiability of the class of *normal constraints* [23].[3] It turns out that normal constraints are exactly the class of lattice constraints we need to consider.

According to the definition of annotated clauses, annotations that occur in the body of a clause must be either c- or v-annotations. A ca-resolution of a query with a clause adds a constraint onto the literal resolved on in the query, and may introduce additional literals. The following is straightforward.

PROPOSITION 4 Suppose $\leftarrow Q$ is a ca-query where Ξ_Q is a normal constraint and P is an ALP. Let $\leftarrow R$ be any ca-resolvent of Q with a clause in P. Then Ξ_R is a normal constraint.

Proof: Suppose $\leftarrow Q$ has the form

$$\leftarrow A_1 : \langle \mu_1, S_1\rangle, ..., A_n : \langle \mu_n, S_n\rangle.$$

The constraint $\Xi_Q = \mu_1 \preceq \sqcup S_1, ..., \mu_n \preceq \sqcup S_n$ is by assumption a normal constraint. Without loss of generality, assume the ca-resolvent $\leftarrow R$ is obtained by ca-resolving on the constrained atom $A_1 : \langle \mu_1, S_1\rangle$ and has the form

$$\begin{aligned} \leftarrow\ & A_1 : \langle \mu_1, S_1 \cup \{\beta\}\rangle, B_1 : \langle \beta_1, \{\}\rangle, ..., B_m : \langle \beta_m, \{\}\rangle, \\ & A_2 : \langle \mu_2, S_2\rangle, ..., A_n : \langle \mu_n, S_n\rangle. \end{aligned}$$

Note β is the annotation from the head of the clause with which $\leftarrow Q$ ca-resolved with, and the atoms $B_1 : \langle \beta_1, \{\}\rangle, ..., B_m : \langle \beta_m, \{\}\rangle$ form the body of the same clause. The constraint Ξ_R is

$$\beta_1 \preceq \bot, ..., \beta_m \preceq \bot, \mu_1 \preceq \sqcup(S_1 \cup \{\beta\}), ..., \mu_n \preceq \sqcup S_n$$

The first condition of normal constraint is satisfied since each of $\beta_1, ..., \beta_m$ is an annotation variable or constant. The second condition of normal constraint is also satisfied as no $\beta_1, ..., \beta_m, \mu_1$ occurs in any of the annotation terms to its left (they are all $\perp$s). Moreover, no $\mu_2, ..., \mu_n$ occurs in $S_1 \cup \{\beta\}$ as the ca-query and the clause in P are standardized apart prior to performing ca-resolution. ■

We call a ca-query $\leftarrow Q$ a *normal ca-query* if Ξ_Q is a normal constraint. Since the solvability of a normal constraint is decidable [23], we may adopt Kifer and Subrahmanian's algorithm to test for unsatisfiability of ca-queries. In order to make the paper self-contained, we include below their algorithm, which we have incorporated into our interpreter, and will analyze further in the next section.

1. Algorithm 1
2. Input: A normal constraint $\Xi = \kappa_1 \preceq \tau_1, ..., \kappa_n \preceq \tau_n$.
3. Output: Boolean
4. If C is an empty constraint, return True
5. Find $i_0 \geq 1$ such that i_0 is the maximal integer where
6. κ_{i_0} is the same expression as κ_1 and substitute
7. $\top$ for all variables that occur in $\tau_1, ..., \tau_{i_0}$.
8. ***If*** κ_1 is a constant then
9. if $\kappa_1 \preceq \tau_1, ..., \kappa_{i_0} \preceq \tau_{i_0}$ is false in Δ, ***return False***
10. Set C to $\kappa_{i_0+1} \preceq \tau_{i_0+1}, ..., \kappa_n \preceq \tau_n$,
11. rearrange the indices of the constraints so that they start
12. with 1 and goto step 4.
13. ***else*** /* κ_1 is a variable */
14. Let $v = \sqcap\{\tau_1, ..., \tau_{i_0}\}$.
15. Set C to $\kappa_{i_0+1}(v/\kappa_1) \preceq \tau_{i_0+1}(v/\kappa_1), ..., \kappa_n(v/\kappa_1) \preceq \tau_n(v/\kappa_1)$
16. and rearrange the indices as in 11, and goto step 4.

4. An Example

Consider the lattice SIX shown in Figure 2. SIX is a belief lattice [20], [21] that may be used for reasoning with inconsistency. It offers a finer granularity of truth values than FOUR. Intuitively, the lattice values It and If in SIX denote likely true, and likely false, respectively.

Joslyn recently inherited some stocks, among them are IBM and Digital stocks. These are represented by the following annotated facts.

$holds\_stock(joslyn, ibm) : \mathbf{t} \leftarrow$

$holds\_stock(joslyn, digital) : \mathbf{t} \leftarrow$

She consults a knowledgeable friend for tips on how to manage stocks, and is informed, among other things, that if a company is planning on putting out a new product, then the value of the stock of the company will likely increase. On the other hand, if a

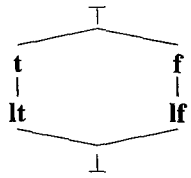

Figure 2. The Complete Lattice SIX

company is on the verge of laying off workers, it is probably the case that the company is in financial trouble, which in turn will likely drop the value of its stock. Thus Joslyn ventures into the stock market world equipped with the following knowledge.

1. $stock\_up(X) : g_1(V) \leftarrow has\_new\_product(X) : V$
2. $stock\_up(X) : g_2(V) \leftarrow financial\_trouble(X) : V$
3. $financial\_trouble(X) : V \leftarrow layoff\_planned(X) : V$

Here g_1 is the annotation function over SIX that maps $\mathbf{t}$ to $\mathbf{lt}$, and $\bot$ to $\bot$. We also assume that $g_2(\mathbf{t}) = \mathbf{lf}$, and $g_2(\bot) = \bot$. Note that g_1 and g_2, as well as g_3 and g_4 below, are annotated functions that model the intuitive relationships between the truth values of our predicates. Suppose Joslyn also has the following general knowledge about the effect of stocks going up or down.

4. $makes\_money(X, Y) : g_3(V, W) \leftarrow stock\_up(Y) : V, holds\_stock(X, Y) : W$
5. $high\_risk(X) : g_4(V) \leftarrow stock\_up(X) : V$

The function g_3 is assumed to satisfy $g_3(\mathbf{lt}, \mathbf{t}) = \mathbf{lt}$, $g_3(\mathbf{lf}, \mathbf{t}) = \mathbf{lf}$, and $g_3(\bot, \bot) = \bot$. In addition, $g_4(\top) = \mathbf{t}$ and $g_4(\alpha) = \bot$ for $\alpha \neq \top$.

Shortly thereafter, while reading the paper, Joslyn learns that IBM is coming out with a new personal computer. Meanwhile, Digital reportedly has plans to layoff workers, but at the same time it is also announcing a new supercomputer.

6. $has\_new\_product(ibm) : \mathbf{t} \leftarrow$
7. $layoff\_planned(digital) : \mathbf{t} \leftarrow$
8. $has\_new\_product(digital) : \mathbf{t} \leftarrow$

According to Joslyn's knowledge, since Digital is now considered a high risk stock while IBM is likely to be profitable, she quickly calls to sell her Digital stocks and to purchase more IBM stocks. In Figure 3, we illustrate a ca-proof for the ca-query

$$\leftarrow makes\_money(joslyn, Y) : \langle \mathbf{lt}, \{\} \rangle.$$

It returns ibm as the answer. The normal constraint associated with each query is shown below the query in the deduction. Since $g_3(\mathbf{lt}, \mathbf{t}) = \mathbf{lt}$, the constraint associated with the last query is solvable with the solution

$$U = \mathbf{t}, W = \mathbf{t}, V = \mathbf{lt}$$

The ca-refutation of the ca-query $\leftarrow high\_risk(X) : \langle \mathbf{t}, \{\} \rangle$ is given in Figure 4. Note as $g_4(\alpha) \neq \bot$ only in the case that $\alpha = \top$, it is necessary that the deduction uses both rules 1 and 2 to generate a value of $\top$ for the annotation variable Z. The solution to the final constraint is

G_0: $\leftarrow makes\_money(joslyn, Y) : \langle \mathbf{lt}, \{\}\rangle$
Constraint: $\mathbf{lt} \preceq \bot$

G_1: $\leftarrow makes\_money(joslyn, Y) : \langle \mathbf{lt}, \{g_3(V, W)\}\rangle, stock\_up(Y) : \langle V, \{\}\rangle,$
$holds\_stock(joslyn, Y) : \langle W, \{\}\rangle$
Constraint: $\mathbf{lt} \preceq g_3(V, W), V \preceq \bot, W \preceq \bot$

G_2: $\leftarrow makes\_money(joslyn, Y) : \langle \mathbf{lt}, \{g_3(V, W)\}\rangle, stock\_up(Y) : \langle \mathbf{lt}, \{g_1(U)\}\rangle,$
$has\_new\_product(Y) : \langle U, \{\}\rangle, holds\_stock(joslyn, Y) : \langle W, \{\}\rangle$
Constraint: $\mathbf{lt} \preceq g_3(V, W), V \preceq g_1(U), W \preceq \bot, U \preceq \bot$

G_3: $\leftarrow makes\_money(joslyn, ibm) : \langle \mathbf{lt}, \{g_3(V, W)\}\rangle, stock\_up(ibm) : \langle \mathbf{lt}, \{g_1(U)\}\rangle,$
$has\_new\_product(ibm) : \langle U, \{\mathbf{t}\}\rangle, holds\_stock(joslyn, ibm) : \langle W, \{\}\rangle$
Constraint: $\mathbf{lt} \preceq g_3(V, W), V \preceq g_1(U), W \preceq \bot, U \preceq \mathbf{t}$

G_4: $\leftarrow makes\_money(joslyn, ibm) : \langle \mathbf{lt}, \{g_3(V, W)\}\rangle, stock\_up(ibm) : \langle \mathbf{lt}, \{g_1(U)\}\rangle,$
$has\_new\_product(ibm) : \langle U, \{\mathbf{t}\}\rangle, holds\_stock(joslyn, ibm) : \langle W, \{\mathbf{t}\}\rangle$
Constraint: $\mathbf{lt} \preceq g_3(V, W), V \preceq g_1(U), W \preceq \mathbf{t}, U \preceq \mathbf{t}$

Figure 3. CA-refutation of $\leftarrow makes\_money(joslyn, Y) : \langle \mathbf{lt}, \{\}\rangle$.

$$W = \mathbf{t}, U = \mathbf{t}, V = \mathbf{t}, Z = \top.$$

This example presents a rather simplistic world view. A more realistic representation of the scenario may be obtained by using the lattice $\mathcal{UNC}$ in place of SIX where both time and uncertainty are captured. For example, typically layoff plans are uncertain, and are usually time bound. Hence fact 7 may be more appropriately expressed as:

7_a. $layoff\_planned(digital) : (0.9, \{t_1\}) \leftarrow$
7_b. $layoff\_planned(digital) : (0.8, \{t_0, t_2\}) \leftarrow$

Informally, 7_a asserts that at time point t_1, there is a 0.9 certainty that digital will layoff employees. On the other hand, 7_b asserts a certainty of 0.8 for layoff at time points t_0 and t_2.

5. An ALP Interpreter

We have implemented in C an interpreter for ALPs based on ca-resolution. In this section, we focus on the important components in our interpreter. Following the approach of [17], our interpreter for ALP consists of a Prolog-like inference engine and a constraint solver. A third component distinct to our system is a function to select the next literal to resolve on. In Prolog, the usual method is to apply resolution to the leftmost literal of the query in each step of a deduction. The theoretical issue of the independence of computation

G_0: $\leftarrow high\_risk(X) : \langle \mathbf{t}, \{\} \rangle$
Constraint: $\mathbf{t} \preceq \bot$

G_1: $\leftarrow high\_risk(X) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(X) : \langle Z, \{\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq \bot$

G_2: $\leftarrow high\_risk(X) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(X) : \langle Z, \{g_1(V)\} \rangle,$
$has\_new\_product(X) : \langle V, \{\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq g_1(V), V \preceq \bot$

G_3: $\leftarrow high\_risk(digital) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(digital) : \langle Z, \{g_1(V)\} \rangle,$
$has\_new\_product(digital) : \langle V, \{\mathbf{t}\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq g_1(V), V \preceq \mathbf{t}$

G_4: $\leftarrow high\_risk(digital) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(digital) : \langle Z, \{g_1(V), g_2(U)\} \rangle,$
$financial\_trouble(digital) : \langle U, \{\} \rangle, has\_new\_product(digital) : \langle V, \{\mathbf{t}\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq \sqcup\{g_1(V), g_2(U)\}, V \preceq \mathbf{t}, U \preceq \bot$

G_5: $\leftarrow high\_risk(digital) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(digital) : \langle Z, \{g_1(V), g_2(U)\} \rangle,$
$financial\_trouble(digital) : \langle U, \{W\} \rangle, layoff\_planned(digital) : \langle W, \{\} \rangle,$
$has\_new\_product(digital) : \langle V, \{\mathbf{t}\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq \sqcup\{g_1(V), g_2(U)\}, V \preceq \mathbf{t}, U \preceq W, W \preceq \bot$

G_6: $\leftarrow high\_risk(digital) : \langle \mathbf{t}, \{g_4(Z)\} \rangle, stock\_up(digital) : \langle \mathbf{t}, \{g_1(V), g_2(U)\} \rangle,$
$financial\_trouble(digital) : \langle U, \{W\} \rangle, layoff\_planned(digital) : \langle W, \{\mathbf{t}\} \rangle,$
$has\_new\_product(digital) : \langle V, \{\mathbf{t}\} \rangle$
Constraint: $\mathbf{t} \preceq g_4(Z), Z \preceq \sqcup\{g_1(V), g_2(U)\}, V \preceq \mathbf{t}, U \preceq W, W \preceq \mathbf{t}$

Figure 4. CA-refutation of $\leftarrow high\_risk(X) : \langle \mathbf{t}, \{\} \rangle$.

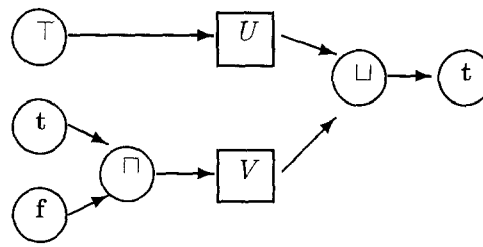

Figure 5. A Constraint Graph

rule was considered in Section 3.2. In Section 5.2, we discuss strategies for selecting literals within a query.

The inference engine performs ca-resolution. Thus its task includes performing unification on object variables, and constructing and modifying constraints associated with each ca-query in a deduction. The constraint solver is used to determine unsatisfiability of a ca-query. Efficient lattice algorithms exist for computing greatest lower bounds and least upper bounds (e.g. [3]) of finite lattices and they will be incorporated into our interpreter. Currently, the algorithm used by the constraint solver is adapted from Algorithm 1.

5.1. The Constraint Solver

The constraint Ξ_Q associated with each ca-query $\leftarrow Q$ can be represented graphically. For example, the normal constraint $U \preceq \top, V \preceq \mathbf{t}, V \preceq \mathbf{f}, \mathbf{t} \preceq \sqcup\{U, V\}$ associated with the ca-query

$$\leftarrow p_1 : \langle \mathbf{t}, \{U, V\}\rangle, p_2 : \langle V, \{\mathbf{f}\}\rangle, p_3 : \langle V, \{\mathbf{t}\}\rangle, p_4 : \langle U, \{\top\}\rangle$$

has the graphical representation shown in Figure 5. Circular nodes represent operators. Constants are operators that require no arguments. Square nodes represent variables. We call the nodes occurring on the far left *origins* of the graph, and the single node occurring on the far right the *destination* of the graph. This representation allows us to view constraints associated with ca-queries as constraint graphs (cf. [26]), which may be solved by *local propagation*. A constraint graph is satisfiable when the variables take on values that make the specified relation true. The relation implicitly specified by the graph above is that the value associated with the arc that terminates at the destination is greater than or equal to the value of the destination. In the given example, the graph in Figure 5 is satisfied since the value that is associated with the right most arc, which emanates from the node $\sqcup$, is $\top$.

Representing normal constraints as constraint graphs in the above manner, one can see that Algorithm 1 corresponds to local propagation. In steps 14 – 16, the algorithm computes the values of the origins, and propagates the results along the arcs to the right to fill in the values for the variables. When the least upper bound left of the destination

is eventually computed, the result is checked against the value of the destination in step 9. The order in which values are computed is thus left to right, delineated by operator nodes. In particular, to compute the value for each of the nodes containing $\sqcup$ and $\sqcap$, all nodes whose arcs terminate at the operator node must first be evaluated, and the resulting values propagated.

The node labeled with $\sqcap$ is obtained from the two inequalities $V \preceq \mathbf{t}$ and $V \preceq \mathbf{f}$, since any lattice element γ that satisfies both inequalities must satisfy $\gamma \preceq \sqcap\{\mathbf{t}, \mathbf{f}\}$. This is also the basis behind steps 14 – 16 of Algorithm 1, which can be stated in the next proposition.

PROPOSITION 5 Given a normal constraint $\Xi = \kappa_1 \preceq \tau_1, ..., \kappa_n \preceq \tau_n$. Suppose for some $1 \leq i < j \leq n$, κ_i and κ_j are the same variable. Then the constraint

$\kappa_1 \preceq \tau_1, ..., \kappa_{i-1} \preceq \tau_{i-1}, \kappa_{i+1} \preceq \tau_{i+1}, ..., \kappa_{j-1} \preceq \tau_{j-1},$

$\kappa_j \preceq \sqcap\{\tau_i, \tau_j\}, \kappa_{j+1} \preceq \tau_{j+1}, ..., \kappa_n \preceq \tau_n$

is solvable iff Ξ is solvable.

Typically a constraint associated with a ca-query consists of several disjoint constraint graphs. For example, the constraint associated with the query

$$\leftarrow p : \langle \mathbf{t}, \{V\}\rangle, q : \langle V, \{\top\}\rangle, r : \langle \mathbf{t}, \{\mathbf{f}\}\rangle$$

contains two disjoint constraint graphs. The first is formed by the constraints associated with the p and q atoms, namely $\mathbf{t} \preceq V, V \preceq \top$. The second is formed from the r atom with the single inequality $\mathbf{t} \preceq \mathbf{f}$. We refer to each set of atoms that forms an independent constraint graph as a *network*. Equivalently, a network of a ca-query $\leftarrow Q$ is an element in the partition π of the atoms in Q where for each pair $N_1, N_2 \in \pi$, no annotation variable occurs in both N_1 and N_2. Thus the two networks in the above ca-query are $N_1 = \{p : \langle \mathbf{t}, \{V\}\rangle, q : \langle V, \{\top\}\rangle\}$, and $N_2 = \{r : \langle \mathbf{t}, \{\mathbf{f}\}\rangle\}$. A result that parallels the independence of computation rule in classical logic programming is that for ALPs, networks can be solved independently.

PROPOSITION 6 Suppose $\leftarrow Q$ is a query that contains networks $N_1, ..., N_k$. Then $\leftarrow Q$ has a ca-proof iff each of $\leftarrow N_1, \leftarrow N_2, ..., \leftarrow N_k$ has a ca-proof, and the resulting substitutions of the object variables are compatible.

Proof: As $N_1, ..., N_k$ do not share any annotation variables, a substitution σ for the annotation variables is a solution to Ξ_Q iff it is a solution to each of $\Xi_{N_1}, ..., \Xi_{N_k}$. Thus the if part of the statment is immediate. The only if part follows from the additional assumption that the substitutions for the object variables are compatible. ■

5.2. *Literal Selection Strategies*

The example in Section 3.2 demonstrated the importance of eventually trying each literal in a network. In this subsection, we analyze possible strategies for selecting literals. Interestingly, each literal selection strategy relates to how the constraint graph is modified. Recall the earlier discussion on the difference between how constraints are used in our

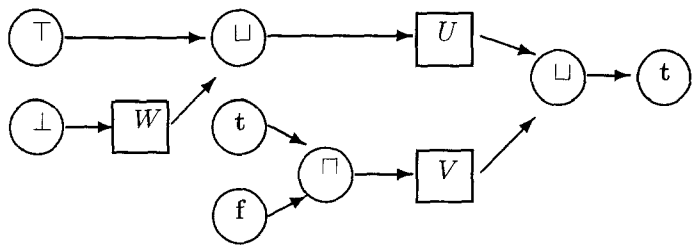

Figure 6. A Modified Constraint Graph

system as compared to their use in conventional CLP systems. In a ca-deduction, a constraint is continually modified until a solvable constraint is constructed. As constraints can be represented by constraint graphs, it is therefore not surprising that the order in which literals are selected for performing ca-resolution determines how the associated constraint graph changes. We illustrate this point with an example.

Example: Let us reconsider the constraint graph corresponding to the query

$$\leftarrow p_1 : \langle \mathbf{t}, \{U, V\}\rangle, p_2 : \langle V, \{\mathbf{f}\}\rangle, p_3 : \langle V, \{\mathbf{t}\}\rangle, p_4 : \langle U, \{\top\}\rangle$$

shown in Figure 5. If we select the atom $p_4 : \langle U, \{\top\}\rangle$ to perform ca-resolution, and if our program contains the clause $p_4 : W \leftarrow p_5 : W$, then the ca-resolvent is

$$\leftarrow p_1 : \langle \mathbf{t}, \{U, V\}\rangle, p_2 : \langle V, \{\mathbf{f}\}\rangle, p_3 : \langle V, \{\mathbf{t}\}\rangle, p_4 : \langle U, \{\top, W\}\rangle, p_5 : \langle W, \{\}\rangle.$$

The constraint graph that represents the associated constraint is shown in Figure 6. One sees that the origin of the graph has been modified, and the overall depth of the graph is increased since the longest path now is of length five. Suppose on the other hand, the atom $p_1 : \langle \mathbf{t}, \{U, V\}\rangle$ is chosen first for applying ca-resolution. This does not change the depth of the graph, but instead modifies the graph by adding new arcs that terminate at the node $\sqcup$ left of the destination. □

For our interpreter, we have classified constrained atoms into four categories, which has enabled the development of literal selection strategies. With respect to the constraint graph, one of the literal selection strategies corresponds to modifying the graph in a "depth-first" manner, where the length of some path in the graph is increased, as illustrated in the previous example. Orthogonally, literals may be selected so as to modify the constraint graph "breadth-first". Each of these strategies for selecting literals is a *guidance strategy*, in the same spirit as guidance strategies are used in theorem proving [40]. Such a strategy imposes an ordering on the search space, but does not eliminate any part of it. It is generally accepted in theorem proving that the usefulness of guidance strategies typically can only be supported experimentally. We are currently experimenting with several literal selection strategies to determine their effectiveness.

5.3. *Restriction Strategies*

We consider briefly restriction strategies for pruning the search space, some of which have been incorporated into the interpreter. The most interesting aspect about these strategies is that they are semantically based, where properties related to the underlying lattice Δ are used to systematically eliminate irrelevant search paths.

5.3.1. *Linear Δ restriction*

In the case where Δ is a totally ordered set, it is straightforward to verify that $P \models Q$ iff there is a proof by annotated resolution of $\leftarrow Q$ from P without using reduction. Using ca-resolution, we may state this equivalently in the following way.

PROPOSITION 7 $P \models Q$, where Q has the form $p : \mu$ iff there is a ca-proof R of $\leftarrow p : \langle \mu, \{\} \rangle$ from P, and R does not contain a ca-query in which there is a constrained annotated atom $q : \langle \beta, S \rangle$ such that $|S| > 1$.

Indeed, this result can be generalized based on the notion of *n-wide* lattices introduced in [23]. A lattice Δ is n-wide if for every finite subset E of Δ, there is a finite subset $E_0 \subseteq E$ containing at most n elements such that $\sqcup E = \sqcup E_0$. Clearly, a totally ordered set is 1-wide. For an n-wide lattice, we may generalize the above result to $|S| > n$.

5.3.2. *Conditional CA-resolution*

Another interesting strategy, when given a constraint graph, is to use the value of the destination to determine "backwards" what the value of the origins might be. This idea is also found in constraint programming [26].

Example: Suppose the lattice Δ is FOUR and that we arrive at the query $\leftarrow p : \langle \top, \{\mathbf{t}, V\} \rangle, q : \langle V, \{\} \rangle$ during a deduction. Realizing that to solve the constraint $\top \preceq \sqcup\{\mathbf{t}, V\}$ associated with the p atom, V must be either $\mathbf{f}$ or $\top$, we may then substitute the lower of the two for V throughout the query and remove the p atom, thus obtaining the ca-query $\leftarrow q : \langle \mathbf{f}, \{\} \rangle$. This indicates that if we are able to show q has a truth value greater than or equal to $\mathbf{f}$ from the program, then the original query is proved. □

This approach, where we begin with the destination of the graph, and compute backward towards the origins, may allow greater flexibility for incremental computation within a network. In the above example, the query $\leftarrow q : \langle \mathbf{f}, \{\} \rangle$ represents a condition for solvability of the original query. Once the condition has been established, we no longer need to keep the atom p in the query. This in turn reduces the search space as the number of literals from which we may select for ca-resolution decreases. A similar idea was considered in [29] for theorem proving with c-annotated only clauses.

The difficulty with this approach however, is that backward solving may involve a considerable amount of non-determinism. Returning to the example above, if the underlying

lattice is large, there may be a number of values that can be substituted for V, but only a few such values can lead to a refutation. In other words, there may be truth values $\mu_1, ...\mu_n$ other than f such that the query $\leftarrow q : \langle \mu_i, \{\} \rangle$, $1 \leq i \leq n$, also represents a correct condition for solvability of the original query, but for the given program, only one or two of these values for q can be shown to hold.

It appears unlikely that a general solution to this problem can be found. More promising is that there may exist special classes of lattices possessing properties that can assist in determining which of the possible truth values represent the "better" choices – ones that are more likely to lead to a refutation. In [25], an optimal such class of lattices has been identified in which a single "correct" choice of truth value is guaranteed to exist. The class is non-trivial as it encompasses many of the lattices that have been used in annotated logic programming applications. Our current work involves giving a precise characterization of the class, along with analyzing the efficiency that can be gained by ca-resolution when augmented with procedures that take advantage of this special property.

6. Summary and Discussion

Kifer and Subrahmanian developed the first query processing procedure for annotated logic programming [22], [23]. However, their proof procedure requires complex machinery, namely both annotated resolution and reduction, for answering queries. The bottleneck rests with the restrictive semantics currently defined for satisfaction of annotated atoms. In this paper, we presented a top-down resolution procedure with constraint solving for processing queries in ALPs based on the observation that the semantics of annotated atoms may be naturally generalized to set membership. The new procedure captures the effect of earlier approaches through a considerably smaller search space. Constraints within our procedure are handled differently from conventional CLP systems, and certain completeness issues absent in classical horn clause programming surface with the new procedure. We have attempted to identify the key implementation issues of our procedure along with possible optimization techniques that include semantically-based restriction strategies. All the theoretical development in this paper has been driven by practical implementation concerns. The study presented in this paper has already contributed to the work on amalgamating databases at the University of Maryland [2], [1], [36].

Several other developments with similar aim as our research are by Frühwirth [11], and by Adalı and Subrahmanian [2]. The work of Frühwirth is interesting in that query processing of annotated logics is handled completely within the framework of CLP. A cursory examination indicates a close relationship between his work and ca-resolution with respect to the structure of the search space. Details are currently under investigation. The procedure of Adalı and Subrahmanian is an adaptation of s-resolution [30], augmented with OLDT-resolution to improve termination behavior and efficiency. It uses the device of sets of annotations, which plays a prominent role in ca-resolution.

6.1. *On the Language of Annotations*

An interesting aspect regarding the work on hybrid knowledge bases [31] is that an SLD-like query processing procedure can be defined without the need to resort to either the reduction inference rule, or the use of constrained annotations. The simplicity and elegance of such a query processing procedure is very appealing both theoretically and practically. However, in general, such a procedure does not exist for annotated logic programs.

The conditions under which annotated logic programs have a straight forward SLD-like query processing procedure is not entirely clear at this point, and is a topic of current investigation. One possibility, arising out of a closer examination of the theory of hybrid knowledge bases, is that the existence of such a procedure may be due to certain restrictions placed on the language of annotations. A key difference between the theory of hybrid knowledge bases and the theory of annotated logic programming in general is that the annotations used in hybrid knowledge bases are not elements taken directly from the underlying lattice. Rather, a different set of syntactic objects, called ℓ-representations, are introduced as annotations. Formally, suppose f is a function in $\mathcal{UNC}$ such that for all inputs $S \subseteq \mathbf{R}^+$ to which f assigns a non-zero output, f gives the same output, say α, to each $x \in S$. Then the pair (α, S) is called the ℓ-representation of f. A function f is said to be *ℓ-representable* if there is a $\alpha \in [0,1]$ and $S \subseteq \mathbf{R}^+$ such that $f = (\alpha, S)$. Not every function in Δ is ℓ-representable; any function whose set of outputs contains more than two elements cannot be so represented. Consequently, the set of elements in $\mathcal{UNC}$ that may be used as annotations constitute a rather small subset of all the elements in $\mathcal{UNC}$. It has been shown in [31] that such a restriction causes a slight loss of expressiveness. However, we postulate that such a specialization may very well be what is necessary to formulate a simple SLD-like query processing procedure for annotated logic programming without using either reduction, or constrained annotation.

6.2. *On the Efficiency of CA-resolution*

The claim is made in this paper of the superior efficiency of ca-resolution as compared to annotated resolution and reduction. The astute readers will have observed that the lengths of ca-deductions may be considerably longer than corresponding deductions obtained from annotation resolution and reduction, in which the only steps displayed use the annotated resolution inference rule. This observation may give the appearance that we are simply modifying the structure of the search space, but without any real reduction in the size.

But surely a fair comparison must take into account the number of reduction steps performed, even if these steps do not explicitly appear. There is, moreover, a simple argument that supports the efficiency claim. Theoretically, an infinite number of reductants may be computed from a finite number of clauses. Thus, the possible number of clauses on which annotated resolution may be applied is infinite at each deduction step. This, in comparison with the finite number of choices – the program clauses – that is available for the application of ca-resolution, indicates a much larger search space. In

practice, the disparity between the sizes of the search spaces is unlikely to be so drastic. Analytically, the examples that we have examined, though small, support the improved behavior of ca-resolution. Presently, no experimental comparisons are available as we do not have an implementation of annotated resolution and reduction.

Acknowledgments

The useful comments of the referees are gratefully acknowledged. Discussions with Thom Frühwirth have improved our understanding of the role of annotations. Barbara Messing pointed out a mistake in an earlier draft.

Notes

1. The articles [12], [16] provide excellent accounts of the field.
2. Of course, Prolog is still incomplete due to its use of depth-first search.
3. Recall that normal constraints were defined in Section 2.1.

References

1. Adalı, S., Subrahmanian, V.S., "Amalgamating Knowledge Bases II: Distributed Mediators", *Journal of Intelligent and Cooperative Information Systems*, December, 1994.
2. Adalı, S., Subrahmanian, V.S., "Amalgamating Knowledge Bases III: Algorithms, Data Structures, and Query Processing", CS-TR-3124, University of Maryland.
3. Aït Kaci, H., Boyer, R., Lincoln, R., Nasr, R., "Efficient Implementation of Lattice Operations", *ACM Transactions on Programming Language and Systems*, 11, 1, 1989, 115–146.
4. Bezdek, J. C., "Fuzzy models – what are they, and why?", *IEEE Transactions on Fuzzy Systems*, 1, 1, 1993, 1–6.
5. Baldwin, J.F., "Evidential Support Logic Programming", *Fuzzy Sets and Systems*, 24, 1987, 1–26.
6. Blair, H.A., Subrahmanian, V.S., "Paraconsistent Logic Programming", *Theoretical Computer Science*, 68, 1989, 135–154.
7. Bundy, A., "A Science of Reasoning: Extended Abstract", *Proceedings of the Conference on Automated Deduction*, 1990, 633–640.
8. Cohen, J., "Constraint Logic Programming Languages", *Communications of the ACM*, 33, 7, 1990, 52–68.
9. Davey B.A., Priestley, H.A., *Introduction to Lattices and Order*, Cambridge University Press, 1990.
10. Fitting, M., "Bilattices and the Semantics of Logic Programming", *Journal of Logic Programming*, 11, 1991, 91–116.
11. T. Frühwirth, Annotated Constraint Logic Programming Applied to Temporal Reasoning, *Proceedings of the Symposium on Programming Language Implementation and Logic Programming*, 1994, 230–243.
12. Frühwirth, T., Herold, A., Küchenhoff, V., Le Provost, T., Lim, P., Wallace, M., "Constraint Logic Programming – An Informal Introduction", *Logic Programming in Action*, 1992, 3–35.
13. Gaasterland, T., Lobo, J., "Qualified Answers that Reflect User Needs and Preferences", *Proceedings of the 20th International Conference on Very Large Databases*, 1994, 309–320.
14. Hähnle, R., "Uniform Notation of Tableau Rules for Multiple-valued Logics", *Proceedings of the International Symposium on Multiple-Valued Logic*, 1991, 26–29.
15. Jaffar, J., Lassez, J-L., "Constraint Logic Programming", *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, 1987, 111–119.
16. Jaffar, J., Maher, M., "Constraint Logic Programming: A Survey", *J. of Logic Programming*, 19/20, 1994, 503–581.

17. Jaffar, J., Michaylov, S., "Methodology and Implementation of a CLP System", tutorial notes given in the Fourth IEEE Symposium on Logic Programming, 1987.
18. Jaffar, J., Michaylov, S., Stuckey, P., Yap, R., "The CLP(R) Language and System", *ACM Transactions on Programming Languages and Systems*, 14, 3, 1992, 339–395.
19. Kifer, M., Li, A., "On the Semantics of Rule-based Expert Systems with Uncertainty", *Proceedings of the 2nd International Conference on Database Theory*, 1988, 102–117.
20. Kifer, M., Lozinskii, E., "RI: A Logic for Reasoning with Inconsistency", *IEEE Symposium on Logic in Computer Science*, 1989, 253–262.
21. Kifer, M., Lozinskii, E., "A Logic for Reasoning with Inconsistency", *Journal of Automated Reasoning*, 9, 1992, 179–215.
22. Kifer, M., Subrahmanian, V.S., "On the Expressive Power of Annotated Logics", *Proceedings of the North American Conference on Logic Programming*, 1989, 1069–1089.
23. Kifer, M., Subrahmanian, V.S., "Theory of Generalized Annotated Logic Programming and its Applications", *Journal of Logic Programming*, 12, 1992, 335–367.
24. Krishnaprasad, T., Kifer, M., "A Theory of nonmonotonic inheritance based on annotated logic", *Artificial Intelligence*, 60, 1, 1993, 23–50.
25. Leach, S., "D-resolution: A Semantically Based Query Processing Procedure for Annotated Logic Programming", Honors Thesis in Computer Science, Bucknell University, May 1994. Computer Science TR94-2.
26. Leler, W., *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, 1988.
27. Lloyd, J.W., *Foundations of Logic Programming*, 2nd ed., Springer, 1988.
28. Loveland, D.W., "A Unifying View of some Linear Herbrand Procedures", *Journal of the ACM*, 19, 1972, 366–384.
29. Lu, J.J., Henschen, L.J., "The Completeness of gp-resolution for Annotated Logics", *Information Processing Letters*, 44, 1992, 135–140.
30. Lu, J.J., Murray, N.V., Rosenthal, E., "Signed Formulas and Annotated Logics", *Proceedings of the 23rd International Symposium on Multiple-Valued Logics*, 1993, 48–53.
31. Lu, J.J., Nerode, A., Subrahmanian, V.S., "Hybrid Knowledge Bases", *IEEE Transactions on Knowledge and Data Engineering*, to appear. TR93-14, Cornell University, revised March 1994.
32. Murray, N.V., Rosenthal, E., "Signed Formulas: A Liftable Meta-Logic for Multiple-Valued Logics", *Proceedings of International Symposium on Methodologies for Intelligent Systems*, 1993, 275–284.
33. Ng, R., Subrahmanian, V.S., "A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases", *Journal of Automated Reasoning*, 10, 1993, 191–235.
34. Reed, D.W., Loveland, D.W., "Near-Horn Prolog and the Ancestry Family of Procedures", presented at the symposium *Logic in Databases, Knowledge Representation and Reasoning* at the University of Maryland Institute for Advanced Computer Studies, Nov. 1992.
35. Subrahmanian, V.S., "Paraconsistent Disjunctive Databases", *Theoretical Computer Science*, 93, 1992, 115–141.
36. Subrahmanian, V.S., "Amalgamating Knowledge Bases", *ACM Transactions on Database Systems*, 19, 2, 1994, 291–331.
37. Van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.
38. Weigert, T.J., Tsai, J-P., Liu, X., "Fuzzy Operator Logic and Fuzzy Resolution", *Journal of Automated Reasoning*, 10, 1993, 59–78.
39. Wiederhold, G., "Mediators in the Architecture of Future Information Systems", *IEEE Computer*, March 1992, 38 – 49.
40. Wos, L.,*Automated Reasoning: 33 Research Problems*, Prentice Hall, 1988.