Theory of Generalized Annotated Logic Programming and its Applications*

Michael Kifer†
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794.
E-Mail: kifer@cs.sunysb.edu

V.S. Subrahmanian
Department of Computer Science
University of Maryland
College Park, Maryland 20742.
E-Mail: vs@cs.umd.edu

Abstract

Annotated logics were introduced in [43] and later studied in [5, 7, 31, 32]. In [31], annotations were extended to allow variables and functions, and it was argued that such logics can be used to provide a formal semantics for rule-based expert systems with uncertainty. In this paper we continue to investigate the power of this approach. First, we introduce a new semantics for such programs based on ideals of lattices. Subsequently, some proposals for multivalued logic programming [5, 7, 32, 47, 40, 18] as well as some formalisms for temporal reasoning [1, 3, 42] are shown to fit into this framework. As an interesting by-product of this investigation, we obtain a new result concerning multivalued logic programming: a model theory for Fitting's bilattice-based logic programming, which until now has not been characterized model-theoretically. This is accompanied by a corresponding proof theory.

1 Introduction

Large knowledge bases can be inconsistent in many ways. Nevertheless, certain "localizable" inconsistencies should not be allowed to significantly alter the intended meaning of such knowledge bases. As classical logic semantics decrees that inconsistent theories have no models (and hence are meaningless from a model-theoretic point of view), classical logic is not the appropriate formalism for reasoning about inconsistent knowledge bases.

As a step towards the solution of this problem, annotated logic programs were introduced by Subrahmanian in [43] and were subsequently studied in [5, 7] by Blair and Subrahmanian. In [32, 33], Kifer and Lozinskii extended the theory to a full-fledged logic, and it was shown that a sound and complete proof procedure exists. More efficient proof procedures have been recently obtained, and implementations of these theorem provers have been designed (cf. [12, 26]). Kifer and Li [31] extended annotated programs in a different direction by allowing variables and evaluable function terms to appear as annotations. We will call such programs *generalized* annotated programs (*GAPs*, for short). The utility of annotated logics for reasoning with

*A preliminary report on this research has appeared in [34]
†Work supported in part by the NSF grant IRI-8903507.

inconsistency and for programming expert systems was well argued in [5, 7, 31, 32, 33]. In this paper we continue to investigate the power of this formalism.

First we extend the semantics of [7, 31, 32, 33] to allow annotation variables over arbitrary semilattices of truth values (in [31] only a special lattice - the Cartesian product of two unit intervals - was considered and in [7, 32, 33], the notion of annotation variable was not present). Then we present the model-theoretic, fixed-point, and operational semantics of GAPs. In Section 5.1 we show that van Emden's quantitative logic programming [47] is a special case of GAPs. Then, in Section 5.2, we show how Fitting's bilattice-based logic programming approach fits into the framework of GAPs. The consequence of this "fit" is that we can now characterize Fitting's approach model-theoretically (no model-theoretic semantics was previously proposed for this approach). By translating [47] and [18] into GAPs, we obtain a sound and complete proof procedure for these theories, thus strengthening van Emden's soundness and completeness theorems (which were obtained under some restrictions) and complementing Fitting's results. Lastly, we demonstrate how to incorporate two versions of temporal logic programming in the framework of GAPs – in the first, we consider a discrete linear version of time, i.e., each instant of time is a *time point*; in the second, we consider an interval-based temporal logic. We show that GAPs are sufficiently expressive to be able to cope with a large body of temporal problems and, in particular, subsume some of the earlier proposals for temporal logic programming [3]. Although our approach cannot directly represent certain constructs used in temporal specifications, we note that the implication problem in most full-fledged temporal logics is Π^1_1-complete and therefore such logics cannot be adequately implemented on a computer, anyway. In contrast, the corresponding problem for temporal specifications in GAPs is semi-decidable and thus they are more suitable for a computer implementation.

We believe that this paper unifies, and in some cases, generalizes various results and treatments of multivalued logic programming. Furthermore, it presents new applications of this formal setting. So far, research in multivalued logic programming has proceeded along three different directions:

1. Annotated logics as described in [5, 7, 32, 33];
2. Bilattice-based logics [23, 17]; and
3. Quantitative rule sets [47, 39, 29, 36, 41, 28, 40].

Earlier studies of these three approaches quickly identified various distinctions between these frameworks. For example, one of the key insights behind bilattices was the interplay between the truth values assigned to sentences and the notion of implication in the language under consideration. Thus, rules (implications) had weights (or truth values) associated with them as a whole. The problem was to study how truth values should be propagated "across" implications. Annotated logics, on the other hand, appeared to associate truth values with each *component* of an implication rather than the implication as a whole. The implication itself was then interpreted in a "classical logic" fashion. The two approaches had their own advantages and disadvantages: although associating truth values with implications, as in [23], has intuitive appeal, annotated logics provide a simpler formalism that is much closer to classical logic. Besides, in [33] it is shown that for many problems in nonmonotonic reasoning, it is easier to arrive at the intended semantics via nonmonotonic annotated logics, compared to the bilattice-based formalism of [23].

However, one of the principal results of this paper is to show that this dichotomy can be done away with. The GAP framework introduced here uses the "classical" definition of implication in the same way as in [43, 7, 32]. However, by appropriately generalizing the concept of an annotation, we are able to capture the propagation of truth values "across" implications (cf. Theorems 7 and 8), at least to the extent this propagation is treated in [17]. This is one of the key insights provided by this paper.

Additionally, this paper demonstrates that the GAP framework can be used to implement a semi-decidable fragment of temporal logics, which is a new application for GAPs. Ginsberg [24] has recently observed that there are various connections between bilattices and modal logics, temporal logics in particular. However, his treatment of temporal logics is very sketchy and no semidecidable proof theory for a large enough fragment of such logics is given.

2 Generalized Annotated Logic Programs

We assume an upper semilattice $\mathcal{T}$ of truth values, and denote the semilattice ordering on $\mathcal{T}$ by $\leq$ and the least upper bound operator by $\sqcup$. The semilattice needs not be complete. It is often convenient to assume the existence of a greatest element in $\mathcal{T}$, denoted $\top$, and some of our results will depend on this assumption. The greatest lower bound operator, when it exists, is denoted by $\sqcap$. Elements of $\mathcal{T}$ can be thought of as confidence factors [7, 31], or degrees of belief [7, 32, 33], or, as we shall see later, as truth values similar to those used in multivalued logics. In addition, sometimes it will be assumed that $\mathcal{T}$ has a unique least element, denoted $\perp$; in these cases this assumption will be made explicitly.

For each $i \geq 1$, we postulate that there is a family $\mathcal{F}_i$ of total continuous (hence monotonic) functions, each of type $\mathcal{T}^i \rightarrow \mathcal{T}$, called annotation functions. We denote $\mathcal{F} = \cup_{i\geq 1}\mathcal{F}_i$ and assume that all functions f in $\mathcal{F}$ are computable in the sense that there is a uniform procedure P_f such that if f is n-ary and $\mu_1, \ldots, \mu_n$ are given as input to P_f, then $f(\mu_1, \ldots, \mu_n)$ is output by P_f in a finite amount of time. We also assume that each $\mathcal{F}_j$ contains a j-ary function $\sqcup_j$, derived from the semilattice operator $\sqcup$, which, given inputs $\mu_1, \ldots, \mu_j$, returns the least upper bound of $\{\mu_1, \ldots, \mu_j\}$. Slightly abusing the notation, we will often write $\sqcup$ instead of $\sqcup_j$. Apart from the interpreted annotation functions, the language contains usual uninterpreted functions, constants, and predicate symbols, as commonly used in logic programs. We also postulate two disjoint sets of variable symbols – *object variables*, and *annotation variables*.

Definition 1 An *annotation* is either an element of $\mathcal{T}$, an *annotation variable*, or a *complex annotation term*. Annotation terms are defined recursively as follows: members of $\mathcal{T}$ and variable annotations are annotation terms. In addition, if $f \in \mathcal{F}_n$ and $x_1, \ldots, x_n$ are annotation terms, then $f(x_1, \ldots, x_n)$ is a complex annotation term.
If A is a usual atomic formula of predicate calculus (built out of object variables and uninterpreted predicate, function, and constant symbols) and α is an annotation, then $A : \alpha$ is an *annotated atom*. An annotated atom containing no occurrences of object variables is *ground*.
If $\alpha \in \mathcal{T}$ then $A : \alpha$ is *constant-annotated* (or *c-annotated*, for brevity). When α is an annotation variable, then $A : \alpha$ is said to be *variable-annotated* (*v-annotated*). If α is a complex annotation term then $A : \alpha$ is *term-annotated* (*t-annotated*).

Definition 2 If $A : \rho$ is an annotated atom and $B_1 : \mu_1, \ldots, B_k : \mu_k$ are c- or v-annotated atoms, then

$$A : \rho \leftarrow B_1 : \mu_1 \,\&\, \ldots \,\&\, B_k : \mu_k$$

is an *annotated clause*. $A : \rho$ is called the *head* of this clause, while $B_1 : \mu_1 \,\&\, \ldots \,\&\, B_k : \mu_k$ is called the *body*. All variables (object or annotation) are implicitly universally quantified.

Any set of annotated clauses is also a *generalized annotated program* (abbr., a *GAP*).

Intuitively, what this definition says is that members of $\mathcal{F}$ may occur in the annotation of the head of a clause, but that elements of $\mathcal{F}$ may not occur in the body of a clause. For instance, if we take $\mathcal{T}$ to be the interval $[0, 1]$ of real numbers, then

$$p(s(X)) : 0.5 \times \mu \leftarrow p(X) : \mu$$

is a clause that says: If the truth value assigned to $p(X)$ is μ, then $p(s(X))$ is assigned $0.5 \times \mu$. In general, using this intuition, a quantitative rule

$$r : A \leftarrow B_1 \,\&\, \ldots \,\&\, B_k$$

of van Emden [47] may be translated into the annotated clause:

$$A : r \times \min\{\mu_1, \ldots, \mu_k\} \leftarrow B_1 : \mu_1 \,\&\, \ldots \,\&\, B_k : \mu_k.$$

This issue will be further discussed in Section 5.1. We also note that

$$p(X) : \mu \leftarrow p(s(X)) : 0.5 \times \mu$$

is *not* an annotated clause because it contains a complex annotation term in the clause-body.

Definition 3 Suppose C is an annotated clause. A *c-annotated instance* of C is any annotated clause obtained by replacing all annotation variables occurring in C by members of $\mathcal{T}$. Different occurrences of the same annotation variable must be replaced by the same member of $\mathcal{T}$.

Definition 4 Suppose C is an annotated clause. A *strictly ground instance* of C is any ground instance of C which contains only c-annotations. Notice that since all functions in $\mathcal{F}$ are evaluable, annotation terms of the form $f(a_1, ..., a_n)$, where a_1, ..., $a_n \in \mathcal{T}$ and $f \in \mathcal{F}_n$, are also considered to be ground and are identified with the result of the computation of f on the a_i's.

We use the notation $SGI(C)$ to denote the set of all strictly ground instances of a clause C. Similarly, if P is a GAP, then we denote the set of all strictly ground instances of clauses in P by $SGI(P)$.

3 The General and Restricted Semantics

In this section, we propose two alternative model-theoretic semantics for GAPs. The first corresponds closely to that in [31, 34], while the second is ideal-theoretic in nature, first proposed in [33].

Definition 5 An *ideal* of an upper semilattice is any subset S such that:

- S is *downward closed*, i.e. $s \in S$ and $t \leq s$ imply $t \in S$; and

- S is closed with respect to *finite* least upper bounds, i.e. $s, t \in S$ implies $s \sqcup t \in S$.

An ideal S is *principal* if for some $p \in \mathcal{T}$, $S = \{s | s \leq p\}$. S is called the principal ideal *generated* by p and is denoted by $\|p\|$. The set of all ideals of $\mathcal{T}$ is denoted by $\mathcal{I}(\mathcal{T})$, and the set of principal ideals of $\mathcal{T}$ will be denoted by $\mathcal{PI}(\mathcal{T})$. □

Example 1 Ideals are not necessarily closed under *infinite* least upper bounds. For example, consider the complete lattice $[0, 1]$ (the unit interval of reals) ordered by the familiar "$\leq$" relation. Then the right-open interval

$$[0, 1) = \{x \mid 0 \leq x < 1\}$$

is an ideal which is not closed under the infinite least upper bound operator. □

It is easily seen that $\mathcal{I}(\mathcal{T})$ forms a complete lattice with the intersection operation serving as the greatest lower bound and the union operator serving as the least upper bound; the order on $\mathcal{I}(\mathcal{T})$ is determined by the usual set inclusion $\subseteq$. Furthermore, there is a homomorphic embedding of upper semilattices $\mathcal{T} \longmapsto \mathcal{I}(\mathcal{T})$ (that preserves finite least upper bounds) that maps elements of $\mathcal{T}$ into the corresponding principal ideals of $\mathcal{I}(\mathcal{T})$.

Definition 6 Let $\mathcal{L}$ be a language of annotated logic. The *Herbrand Base* of $\mathcal{L}$, $B_{\mathcal{L}}$, is the set of all ground atomic formulas of $\mathcal{L}$ (without annotations).
A *general Herbrand interpretation* (or just interpretation, for short) I is a mapping from the Herbrand Base of $\mathcal{L}$ to $\mathcal{I}(\mathcal{T})$. Since I is a function into a partially ordered set $\mathcal{I}(\mathcal{T})$, we can define a partial order on interpretations in the usual way: $I \leq J$ if and only if for every $p \in B_{\mathcal{L}}$, $I(p) \subseteq J(p)$.

A *restricted* Herbrand interpretation (*r-interpretation*, for short) of $\mathcal{L}$ is any map from $B_{\mathcal{L}}$ to $\mathcal{T}$. Equivalently, a r-interpretation of $\mathcal{L}$ is a map from $B_{\mathcal{L}}$ to $\mathcal{PI}(\mathcal{T})$, the set of *principal* ideals of $\mathcal{T}$ (this explains the name "restricted" for such interpretations). □

Note here the distinction between the two notions of interpretations. Restricted Herbrand interpretations assign a single truth value, i.e., essentially a *principal* ideal to ground atoms; in contrast, general interpretations assign *arbitrary* ideals to atoms. Therefore, every r-interpretation is also a general interpretation, but not vice versa. In the sequel, we will be freely switching between the two views of r-interpretations, i.e. we will think of them either as mappings $B_{\mathcal{L}} \mapsto \mathcal{T}$ or $B_{\mathcal{L}} \mapsto \mathcal{PI}(\mathcal{T})$, depending on which of the views is more convenient at the moment. Following [32, 33], we could also define interpretations with arbitrary domains, but since in this paper we are mainly concerned with logic programming, we will restrict our attention to Herbrand interpretations only.

We assume that there is a unary operator $\neg : \mathcal{T} \to \mathcal{T}$, conceptually interpreted as *negation*. For the technical purposes of this paper, we do not need to impose any restrictions on $\neg$. However, sometimes one may wish $\neg$ to satisfy certain epistemological criteria, such as $\neg$ being a symmetric mapping, and the like. The proof theory for GAPs in Section 4 does not depend on these assumptions.

Definition 7 (Satisfaction) Suppose I is a general interpretation, $\mu \in \mathcal{T}$ is a c-annotation in $\mathcal{T}$ and A is a ground atom. Then:

1. If A is a ground atom, $I \models A : \mu$ if and only if $\mu \in I(A)$, where $\mu \in \mathcal{T}$.

2. $I \models \neg A : \mu$ if and only if $\neg(\mu) \in I(A)$.

3. $I \models F_1 \,\&\, F_2$ if and only if $I \models F_1$ and $I \models F_2$.

4. $I \models F_1 \vee F_2$ if and only if $I \models F_1$ or $I \models F_2$.

5. $I \models F_1 \leftarrow F_2$ if and only if $I \models F_1$ or $I \not\models F_2$.

6. $I \models F_1 \leftrightarrow F_2$ if and only if $I \models (F_1 \leftarrow F_2)$ and $I \models (F_2 \leftarrow F_1)$.

7. $I \models (\forall x)F$ if and only if $I \models F(x/t)$ for all ground terms t. Here x is an object or annotation variable and t must be of the same sort as x (i.e., either a usual ground first-order term, or an element of $\mathcal{T}$). $F(x/t)$ denotes the replacement of all free occurrences of x in F by t.

8. $I \models (\exists x)F$ if and only if $I \models F(x/t)$ for some ground term t, where x is an object or annotation variable.[1]

9. If F is not a closed formula, then $I \models F$ if and only if $I \models (\forall)F$, where $(\forall)F$ denotes the universal closure of F.

Definition 8 (r-Satisfaction) Suppose I is an r-interpretation, $\mu \in \mathcal{T}$ is a c-annotation in $\mathcal{T}$ and L is a ground literal. Then:

1. If A is a ground atom, $I \models^r A : \mu$ if and only if $I(A) \geq \mu$.

2. $I \models^r \neg A : \mu$ if and only if $\neg(\mu) \leq I(A)$.

The remaining cases (3) – (10) are defined in exactly the same way as for general satisfaction.

As usual, an interpretation I (or r-interpretation J) is said to be a *model* (resp., *r-model*) of a formula F if and only if $I \models F$ (resp., $J \models^r F$). I is a model of a set of formulas P (of a GAP, in particular) if and only if it is a model of each of the formulas in P. Also, if P is a set of formulas and ϕ is a formula, we write $P \models \phi$ (or $P \models^r \phi$) if and only if whenever $I \models P$ (resp., $I \models^r P$) then $I \models \phi$ (resp., $I \models^r \phi$).

In annotated logics, there are at least two different (but related) notions of negation [32, 33]. The *ontological* negation is close to the standard negation in predicate calculus; for annotated logics it was first studied in [32]. On the other hand, the negation defined in (2) of Definition 7 is close to the negation used in multivalued logics. For annotated logics it was first introduced in [43, 5, 7]; it was dubbed *epistemic* negation in [32]. One of the advantages of epistemic negation is that, given a c-annotated literal $\neg A : \mu$, there is a c-annotation $\rho = \neg(\mu)$ such that $\neg A : \mu$ is logically equivalent to $A : \rho$. This type of negation is *monotonic* and, therefore, is more tractable. The other negation, ontological [32], defines satisfaction of negated atoms as follows: $I \models \sim A : \mu$ if and only if $I \not\models A : \mu$. In this case, there is usually no ρ such that $A : \rho$ and $\sim A : \mu$ are logically equivalent. As a result, ontological negation is computationally more expensive. However, the primary reason for our use of epistemic negation in this paper is not computational, but the fact that the negation in Fitting's theory of logic programming over

[1] If t is an annotation ground term, it can be identified with a constant in $\mathcal{T}$, since all annotation functions are evaluable.

bilattices [18] directly translates into the epistemic negation of GAPs (see Section 5.2). Also, it is easy to see that the implication, $A \leftarrow B$, can be expressed via ontological negation as follows: $A \vee \sim B$. However, "$\leftarrow$" is not expressible via $\vee$, $\wedge$, and the epistemic negation $\neg$. Therefore, since the ontological negation is not used in this paper, we had to define the implication "$\leftarrow$" separately. Properties of ontological negation are discussed in detail in [32, 33, 13].

The above definition tells us what the models of a GAP are. Note that if some annotation term, ζ, appears in the head of a rule $A : \zeta \leftarrow Body$ and ζ has an annotation variable, x, which does not appear as an annotation in $Body$, then because of the monotonicity of annotation functions (i.e. functions in the $\mathcal{F}_i$'s) and due to the way the semantics is defined, we can replace all occurrences of x in ζ by $\top$ while preserving program equivalence (recall that x is implicitly universally quantified). Consider the following example:

Example 2 Suppose P is the following program over the unit interval $[0, 1]$ of truth values:

$$p : x \leftarrow q : 0.3$$

$$q : 0.4 \leftarrow$$

It is easy to see that this program is model-theoretically equivalent (using either general or restricted models) to:

$$p : 1 \leftarrow q : 0.3$$

$$q : 0.4 \leftarrow$$

(recall that when $\mathcal{T} = [0, 1]$ then $\top = 1$). □

Thus, we can assume without loss of generality that, in every clause, *variables occurring in the annotation of the clause head also appear as annotations of the body literals.* We will make this assumption throughout this paper. For the *facts* (clauses with an empty body) appearing in GAPs, this implies that *c-annotations can always be assumed*, which is done till the end of the paper.

Following the usual development of the semantics of logic programs, we associate two operators T_P and R_P with any GAP P: T_P maps interpretations to interpretations and R_P maps r-interpretations to r-interpretations. They are defined as follows:

Definition 9 Suppose I is an interpretation and $A \in B_{\mathcal{L}}$. Then $T_P(I)(A) =$ the *least* ideal of $\mathcal{T}$ containing the set $\{f(\mu_1, \ldots, \mu_n) \mid A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \,\&\, \ldots \,\&\, B_n : \mu_n$ is in $SGI(P)$, and $I \models (B_1 : \mu_1 \,\&\, \ldots \,\&\, B_n : \mu_n)\}$. It is easy to see that intersection of an arbitrary number of ideals is an ideal, and therefore for every subset $S \subseteq \mathcal{T}$ there is a unique least ideal containing S. Note also that $T_P(I)(A)$ is a subset, not an element, of $\mathcal{T}$. □

Definition 10 Suppose I is an r-interpretation and $A \in B_{\mathcal{L}}$. Assume also that $\mathcal{T}$ is a complete semilattice. Then $R_P(I)(A) = \sqcup\{f(\mu_1, \ldots, \mu_n) \mid A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \,\&\, \ldots \,\&\, B_n : \mu_n$ is in $SGI(P)$, $I \models (B_1 : \mu_1 \,\&\, \ldots \,\&\, B_n : \mu_n)\}$. □

Notice that if I is an r-interpretation (hence also an interpretation), then $R_P(I)(A) = \sqcup T_P(I)(A)$ for each atom A, where $\sqcup$ is the least upper bound operator (we postulate that $\sqcup\{\,\} = \perp$). Later we will establish a much more general result regarding the relationship between T_P and R_P that will be subsequently used in Section 5 to establish the relationship between GAPs and van Emden's [47] and Fitting's [18] works.

One intuition behind the ideal-theoretic definition is the following: Consider an interpretation I and a ground atom A. If there is a clause in $SGI(P)$ with head $A : \mu$ and whose body is satisfied by I, then we may use I to "conclude" that there is a derivation of $A : \mu$ by using modus ponens. We collect all such μ's together as a set called $\Gamma_P(I)(A)$, say. Now, using the finitary inference rule

$$\frac{A : \mu_1\ ,\ \ A : \mu_2}{A : \sqcup\{\mu_1, \mu_2\}}$$

we may conclude in a finitary way that $\Gamma_P(I)(A)$ should be closed under finite lubs. The main difference between R_P and T_P is that R_P would also allow *infinite* lubs to be present. It is precisely because of this finitary/infinitary distinction that T_P possesses some desirable properties (to be discussed shortly) that R_P does not possess. The question of which semantics is more intuitive depends on whether one believes that taking infinite lubs is a justified inference step. In any case, Theorem 3 below shows that for most practical purposes the two semantics yield the same results.

Theorem 1 Suppose P is a GAP, I is an interpretation and J is an r-interpretation. Then

- I is a model of P if and only if $T_P(I) \leq I$;
- J is an r-model of P if and only if $R_P(J) \leq J$;
- T_P is monotonic;
- R_P is monotonic.

Proof: A simple modification of the standard proof, e.g., from [35], with the use of the monotonicity property of annotated functions in $\mathcal{F}$. □

In what follows, we will often use a special "least" interpretation, Δ, which assigns the empty ideal $\{\}$ to every atom. In case of restricted interpretations, the least r-interpretation may not exist, unless we require $\mathcal{T}$ to have the least element $\perp$. In the latter case, the least r-interpretation, denoted Δ_r, assigns $\perp$ to every atom in $B_{\mathcal{L}}$.

Let us define the iterations of T_P as follows: $T_P \uparrow 0 = \Delta$. If α is a successor ordinal, then $T_P \uparrow \alpha = T_P(T_P \uparrow (\alpha - 1))$; if α is a limit ordinal, then $T_P \uparrow \alpha = \sqcup_{\beta<\alpha} T_P \uparrow \beta$. In the preceding sentence, $(\alpha - 1)$ denotes the immediate predecessor of the successor ordinal α. The iterations of R_P are defined similarly with the exception that $R_P \uparrow 0 = \Delta_r$. We will see that as in "classical" logic programming, T_P is continuous, and the equation $T_P \uparrow \omega = lfp(T_P)$ holds, but this is not always the case with R_P.

Theorem 2 Let P be a GAP. Then

1. T_P is continuous;
2. $T_P \uparrow \omega = lfp(T_P) =$ *the least model of* P;
3. For all annotated ground atoms $A : \mu$, $P \models A : \mu$ if and only if $\mu \in T_P \uparrow \omega(A)$.

Proof: The only non-obvious thing is the continuity of T_P. The rest of the claims follow from continuity in a standard way.

To show continuity, let $I_1, I_2, \ldots$ be a directed sequence of interpretations of P (i.e., every finite subsequence $I_{i_1}, ..., I_{i_k}$ has an upper bound I_l: $I_l \geq I_{i_j}$, $j = 1, ..., k$). We have to show that $T_P(\sqcup I_i) = \sqcup(T_P(I_i))$. It is easily seen from the definitions that for any set of interpretations, $\{J_k\}$, their least upper bound, $\sqcup J_k$, is such an interpretation J that for every ground atom A, $J(A)$ is the least ideal containing the set $\cup J_i(A)$.

Since T_P is monotonic, $T_P(I_k) \leq T_P(\sqcup I_i)$ for all k. Since, for every A, $T_P(\sqcup I_i)(A)$ is an ideal, we conclude that $T_P(\sqcup I_i) \geq \sqcup(T_P(I_i))$.

In the other direction, let A be a ground atom such that $\mu \in T_P(\sqcup I_i)(A)$. Then there must be a strict ground instance of a rule in P of the form $A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n$, where $\mu = f(\mu_1, \ldots, \mu_n)$ and the literals $B_j : \mu_j$ are satisfied by $\sqcup I_i$. This means that for every $j = 1, ..., n$, there are ν_{j1}, ..., ν_{jk_j} in $\mathcal{T}$ such that

1. each of the $B_j : \nu_{jl}$ is in some I_i; and
2. $\mu_j = \sqcup\{\nu_{j1}, ..., \nu_{jk_j}\}$.

Since the set I_i of interpretations is directed, there is some I_{i_0} that satisfies all the $B_j : \nu_{jl}$. Because of (1) above, $I_{i_0} \models B_1 : \nu_{1m_1} \,\&\ldots\&\, B_n : \nu_{nm_n}$, for any m_1, ..., m_n such that $1 \leq m_1 \leq k_1$, ..., $1 \leq m_n \leq k_n$. Hence,

$$f(\nu_{1m_1}, \ldots, \nu_{nm_n}) \in T_P(I_{i_0})(A). \tag{1}$$

Therefore, by continuity of f (all annotation functions are continuous, by definition),

$$\begin{aligned}
&A : f(\mu_1, \ldots, \mu_n) = \\
&A : f(\sqcup\{\nu_{11}, ..., \nu_{1k_1}\}, ..., \sqcup\{\nu_{n1}, ..., \nu_{nk_n}\}) = \\
&A : \sqcup f(\nu_{1m_1}, ..., \nu_{nm_n}).
\end{aligned}$$

Thus, because of Equation (1) and since $T_P(I_{i_0})$, being an ideal, is closed under finite least upper bounds, we derive that $f(\mu_1, \ldots, \mu_n) \in \sqcup(T_P(I_i))(A)$, which concludes the proof. □

Corollary 1 If A is a ground atom such that $\mu \in T_P \uparrow \omega(A)$ then there is an integer n such that $\mu \in T_P \uparrow n(A)$.

Proof: Since $T_P \uparrow n \subseteq T_P \uparrow (n+1)$ for all $n \geq 0$, it follows that $(T_P \uparrow \omega)(A) = \cup(T_P \uparrow n)(A)$ (i.e. a plain union of sets instead of the least upper bound $\sqcup$). Therefore μ must belong to one of the $T_P \uparrow n(A)$'s. □

Unlike T_P, R_P may not be continuous, as illustrated in the following example:

Example 3 Suppose $\mathcal{T}$ is the interval of real numbers $[0, 1]$ with the usual ordering, and consider the following program:

$$\begin{aligned}
&p : 0 \leftarrow \\
&p : \tfrac{1+x}{2} \leftarrow p : x \\
&q : 1 \leftarrow p : 1,
\end{aligned}$$

where x is an annotation variable. It is easy to see that the interpretation $T_P \uparrow i$, $0 \leq i \leq \omega$, always assigns the empty ideal $\{\}$ to q. Hence, $(T_P \uparrow \omega)(q) = \{\}$. Now, according to the restricted semantics,

$$(R_P \uparrow \omega)(p) = \{a | a \leq 1\}, \tag{2}$$

while according to the general semantics,

$$(T_P \uparrow \omega)(p) = \{a | a < 1\}. \tag{3}$$

Therefore, the r-semantics yields $(R_P(R_P \uparrow \omega))(q) = \mathcal{T}$, because of the third rule, while by the general semantics we have $(T_P(T_P \uparrow \omega))(q) = \{\}$. This shows that $R_P \uparrow \omega$ is not a fixpoint of R_P in the r-semantics; however, in the general semantics, $T_P \uparrow \omega$ is a fixpoint of T_P, by Theorem 2. Notice that the only difference between Equations (2) and (3) is that "$\leq$" is used in (2), while in (3) "$<$" is employed. □

Thus, we see that one of the major differences between R_P and T_P is that the latter is continuous and hence attains a fixed-point at the ω-th step of its upward iteration, while R_P does not possess either of these properties. Interestingly, this profound difference is merely due to the fact that we used different notions of least upper bound (of infinite sets of annotations) to define T_P and R_P. In the remainder of this section we give a simple characterization of when R_P attains a fixed-point at ω.

Blair and Subrahmanian [5, 7] have shown that $lfp(R_P) = R_P \uparrow \omega$ whenever P is c-annotated, and $\mathcal{T}$ is a lattice. It follows immediately from that proof that the same result holds when no annotation variables appear in rule bodies. In parallel, Kifer and Li [31] showed that $lfp(R_P) = R_P \uparrow \omega$ holds at the other end of the spectrum: when P contains only v-annotations in rule bodies (in which case R_P is even continuous). This implies that R_P exhibits undesirable behaviour only when c- and v-annotations are intermixed in rule bodies.

Apart from the two important cases considered in [5, 7, 31], there is a large class of programs for which R_P is *not* necessarily continuous, but still $R_P \uparrow \omega = lfp(R_P)$ holds. Let us call the latter equation the *fixpoint reachability requirement.* Reachability of the least fixed point in at most ω iterations is important for a practical logic programming system, since otherwise it may often mean that no proof theory for the respective class of programs is likely to exist. In the following we identify a large class of programs for which the fixpoint reachability property of R_P holds.

First we need to introduce one additional operator, denoted $\coprod$, that maps general interpretations to r-interpretations. Given an interpretation, J, we define $(\coprod J)(A) = \sqcup\{\mu \mid \mu \in J(A)\}$, for every atom A of $B_{\mathcal{L}}$. Here $\sqcup$ is the operator that yields the unique least upper bound of a set (assuming that $\sqcup\{\,\} = \bot$). This operator will be used to establish a relationship between R_P and T_P. Its usefulness in this context becomes apparent if we recall that $R_P(I)(A) = \sqcup T_P(I)(A)$ – a property that immediately follows from the definition.

Definition 11 A program P is *acceptable* if and only if the following property holds for every c-annotated literal l in the body of P:

- If $\coprod(T_P \uparrow \omega) \models l'$ for some ground instance l' of l, then $T_P \uparrow \omega \models l'$. □

It is easily seen that all programs whose clause bodies are either entirely c-annotated or all are v-annotated, are acceptable. In the first case, this is because $(T_P \uparrow \omega)(A)$ is a finitely generated ideal for every atom A, and all such ideals are obviously principal. Hence $\coprod(T_P \uparrow \omega) = T_P \uparrow \omega$. In the second case, the acceptability follows because the condition in Definition 11 is vacuously satisfied (as there are no c-annotations in the body of P).

Example 4 Consider again the program P of Example 3. Here $(T_P \uparrow \omega)(p) = \{a|a < 1\}$ and thus $(T_P \uparrow \omega) \not\models p : 1$. On the other hand, $\coprod(T_P \uparrow \omega)(p) = \{a|a \leq 1\}$ and hence $\coprod(T_P \uparrow \omega) \models p : 1$. Since $p : 1$ is a c-annotated literal in the body of P, P is not acceptable. $\Box$

Theorem 3 If P is acceptable and $\mathcal{T}$ has a least element $\perp$ then

$$R_P \uparrow \omega = lfp(R_P) = \textit{ the least r-model of } P = \coprod(lfp(T_P)).$$

Surprisingly, this theorem hinges on the assumption about $\perp$ as much as it does on the assumption about acceptability of P. To see this, consider a semilattice without $\perp$, e.g., $\mathcal{T} = \{\mathbf{t}, \mathbf{f}, \top\}$. Assume that $\mathbf{t}$, $\mathbf{f}$ are incomparable to each other, but both are smaller than $\top$. Then the program $\{p : X \leftarrow q : X\}$ has two minimal models: $\{p : \mathbf{t}, q : \mathbf{t}\}$ and $\{p : \mathbf{f}, q : \mathbf{f}\}$, none of which is the least r-model. We thus assume the existence of $\perp$ till the end of this section. Likewise, we require $\mathcal{T}$ to be a complete semilattice in order for R_P to be well-defined.

Proof of Theorem 3: By (1) of Lemma 1 below and by (1) of Theorem 2, it follows that $R_P \uparrow \omega = \coprod(lfp(T_P))$. By (2) of Lemma 1, $R_P \uparrow \omega$ is a fixpoint of R_P; it is the least such fixpoint because of the monotonicity of R_P (Theorem 1). The claim about the least r-model being equal to all the rest, also easily follows from Theorem 1. $\Box$

Lemma 1 *Let P be an acceptable GAP and $\mathcal{T}$ be a complete upper semilattice with the least element $\perp$. Then*

1. $R_P \uparrow \omega = \coprod(T_P \uparrow \omega)$.
2. $R_P(R_P \uparrow \omega) = R_P \uparrow \omega$.

Proof: For Claim (1), we will show that $R_P \uparrow \omega \geq \coprod(T_P \uparrow \omega)$ and vice versa.

To see that $R_P \uparrow \omega \geq \sqcup(T_P \uparrow \omega)$ we first prove by induction that $R_P \uparrow i \geq T_P \uparrow i$, for all i. The base case is immediate:

$$T_P \uparrow 0 = \Delta \leq \Delta_r = R_P \uparrow 0.$$

For the inductive case,

$$T_P \uparrow (k+1) = T_P(T_P \uparrow k) \leq R_P(T_P \uparrow k) \leq R_P(R_P \uparrow k) = R_P \uparrow (k+1). \tag{4}$$

In Equation (4), the first inclusion follows because $T_P(I)(A) \subseteq R_P(I)(A)$ for all atoms A and the second one follows by the inductive assumption and because of the monotonicity of R_P. We now obtain that for all k, $T_P \uparrow k(A) \subseteq R_P \uparrow \omega(A)$ and, finally, that $R_P \uparrow \omega(A) \supseteq \sqcup(T_P \uparrow \omega)(A)$.

In the other direction, we show that for all i,

$$\sqcup (T_P \uparrow \omega)(A) \supseteq (R_P \uparrow i)(A). \tag{5}$$

It would then follow that for every A,

$$\sqcup(T_P \uparrow \omega)(A) \supseteq \sqcup\{(R_P \uparrow i)(A)|i = 1, 2, \ldots\} = R_P \uparrow \omega.$$

We prove (5) by induction on i. The base case is trivial, since $\Delta_r = \sqcup\Delta$, by definition. For the inductive step, assume that $\sqcup(T_P \uparrow \omega)(A) \supseteq (R_P \uparrow k)(A)$ for all A; we will show that this equality holds for $k+1$ as well.

By the definition of R_P, $R_P(R_P \uparrow k)(A) = \sqcup\{f(\mu_1,\ldots,\mu_n) \mid A : f(\mu_1,\ldots,\mu_n) \leftarrow B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n$ is in $SGI(P)$, $R_P \uparrow k \models (B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n)\}$. By the inductive assumption, for every such rule-instance in $SGI(P)$, $\coprod(T_P \uparrow \omega) \models (B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n)$. Because of the acceptability of P, for each i there is a sequence $\mu_i^1 \leq \mu_i^2 \leq \ldots$ such that $\sqcup\{\mu_i^j \mid j = 1,2,\ldots\} = \mu_i$ and $(T_P \uparrow \omega) \models (B_1 : \mu_1^j \,\&\ldots\&\, B_n : \mu_n^j)$ (acceptability is needed to ensure that logical implication holds also in cases when some of the μ_i are annotation constants appearing in the body of P).

Now, since annotation functions are continuous, $\sqcup\{f(\mu_1^j,\ldots,\mu_n^j) \mid j = 1,2,\ldots\} = f(\mu_1,\ldots,\mu_n)$. Therefore $R_P(R_P \uparrow k)(A) \subseteq \sqcup(T_P \uparrow \omega)(A)$.

For Claim (2), $R_P(R_P \uparrow \omega) \geq R_P \uparrow \omega$ follows from the monotonicity of R_P and since $R_P \uparrow \omega \geq R_P \uparrow i$, for all i. The other inclusion is proved similarly to the earlier proof that $R_P(R_P \uparrow k)(A) \subseteq \sqcup(T_P \uparrow \omega)(A)$, in Claim (1). □

4 Constrained Queries and GAPs

Following [44], we are now going to develop an SLD-style proof theory for GAPs based on the general semantics.

Definition 12 Suppose P is a GAP and $C_1,\ldots,C_n$ are renamed versions of clauses in P such that no pair C_i and C_j of clauses shares common variables. Further, let each C_r, $1 \leq r \leq n$, be of the form

$$A_r : \rho_r \leftarrow B_1^r : \mu_1^r \,\&\cdots\&\, B_{m_r}^r : \mu_{m_r}^r,$$

where ρ_r is an annotation term and each μ_k^r is an annotation variable or a constant. Suppose further that $A_1,\ldots,A_n$ are unifiable via an mgu θ and $\rho = \sqcup\{\rho_1,\ldots,\rho_n\}$. Then the clause

$$[\, A_1 : \rho \leftarrow B_1^1 : \mu_1^1 \,\&\ldots\&\, B_{m_1}^1 : \mu_{m_1}^1 \,\&\ldots\, \& B_1^n : \mu_1^n \,\&\ldots\&\, B_{m_n}^n : \mu_{m_n}^n \,]\theta$$

is called a *reductant* of P [32]. Here, the expression $\sqcup\{\rho_1,\ldots,\rho_n\}$ is evaluated only if all the ρ_i's are c-annotations; otherwise the complex term-annotation $\sqcup\{\rho_1,\ldots,\rho_n\}$ becomes the annotation in the head of the reductant. Note, in particular, that the clauses C_i, C_j above may be different renamings of the same clause from P. Furthermore, it should be clear that each clause in P is a reductant of itself.

Example 5 Suppose $\mathcal{T}$ is the lattice $\mathcal{FOUR}$ shown in Figure 1. This lattice has been used extensively in reasoning about knowledge bases which may contain inconsistent information (cf. [6, 46]). Intuitively, $\bot$ represents the Kleene's truth value "unknown", while $\top$ represents the truth value "inconsistent". Let P be the program:

$$p(X) : \mathbf{t} \leftarrow q(X,Y) : \mathbf{f}$$

$$p(b) : \mathbf{f} \leftarrow r(Z,a) : \mathbf{t}.$$

Then

$$p(b) : \top \leftarrow q(b,Y) : \mathbf{f},\ r(Z,a) : \mathbf{t}$$

is a reductant of P via the mgu $\theta = \{X \backslash b\}$.

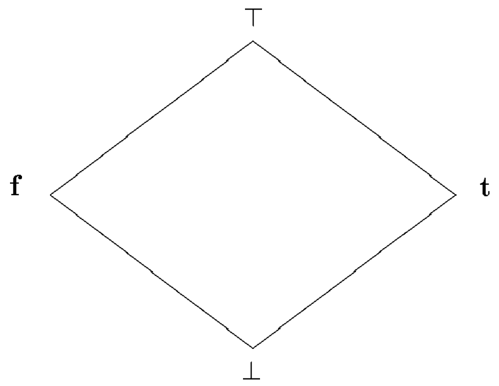

Figure 1: The four-valued lattice $\mathcal{FOUR}$

Theorem 4 If C is a reductant of P, then $P \models C$.

Proof: As C is a reductant of P, it is obtained from clauses $C_1, \ldots, C_n$, where each C_r is a renamed version of a clause in P and no pair C_i and C_j shares common variables. Hence, C is of the form

$$(A_1 : \rho \leftarrow B_1^1 : \mu_1^1 \,\&\ldots\&\, B_{m_1}^1 : \mu_{m_1}^1 \,\&\ldots\,\& B_1^n : \mu_1^n \,\&\ldots\&\, B_{m_n}^n : \mu_{m_n}^n)\theta$$

where $\rho = \sqcup\{\rho_1, ..., \rho_n\}$, each C_j, $1 \leq j \leq n$ is of the form

$$A_j : \rho_j \leftarrow B_1^j : \mu_1^j \,\&\ldots\&\, B_{m_j}^j : \mu_{m_j}^j,$$

and $\{A_1, \ldots, A_n\}$ are unifiable via an mgu θ. Suppose now that I is a model of P, and $C\sigma$ is a strict ground instance of C such that

$$I \models (B_{m_1}^1 : \mu_{m_1}^1 \,\&\ldots\,\& B_1^n : \mu_1^n \,\&\ldots\&\, B_{m_n}^n : \mu_{m_n}^n)\theta\sigma.$$

Hence, I satisfies the body of $C_j\theta\sigma$ for all $1 \leq j \leq i$. I is a model of P and hence a model of each C_j (as the C_j's are only renamed versions of clauses in P). Hence, $I(A_1\theta\sigma) \geq \rho_j$ for all $1 \leq j \leq i$, and thus $I(A_1\theta\sigma) \geq \rho = \sqcup\{\rho_1, \ldots, \rho_i\}$. This completes the proof. □

Proposition 1 Suppose P is a GAP and $A : \mu$ is an annotated atom. If $P \models A : \mu$, then there is a reductant of P having the form:

$$A : \rho \leftarrow B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n$$

such that $\rho \geq \mu$ and $P \models B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n$.

Proof: Suppose $P \models A : \mu$ where A is a ground atom. By Theorem 2, $\mu \in T_P \uparrow \omega(A)$, and by Corollary 1 there is an integer k such that $\mu \in (T_P \uparrow k)(A)$. We proceed by induction on k.

Base Case: $k = 1$. In this case, $\mu \in (T_P \uparrow 1)(A)$. Hence, there are clauses $C_1, \ldots, C_r$, $r \geq 1$, in $SGI(P)$, of the form (empty bodies):

$$A : \rho_1 \leftarrow$$
$$\ldots$$
$$A : \rho_r \leftarrow$$

such that $\mu \leq \rho = \sqcup\{\rho_1, \ldots, \rho_r\}$. Since $A : \rho \leftarrow$ is the reductant of the above clauses, the base case follows.

Inductive Step: $k = i + 1$. The proof proceeds along the lines of the Base Case. □

Example 6 (S. Morishita) Suppose $\mathcal{T}$ is the power set of $\{a, b\}$ and is ordered under inclusion. Consider the program P:

$$p : V \leftarrow q(X) : V$$
$$q(a) : \{a\} \leftarrow$$
$$q(b) : \{b\} \leftarrow$$

Clearly, $P \models p : \{a, b\}$. The clause

$$p : \sqcup\{V_1, V_2\} \leftarrow q(X_1) : V_1 \,\&\, q(X_2) : V_2$$

is a reductant of P involving two different renamings of the first clause. Without being able to take reductants, the proof theory given in this section would not be complete, as it would be impossible to prove $p : \{a, b\}$ from P.

Definition 13 A *query* is a statement of the form $? -\ A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k$, where the $A_i : \rho_i$'s are atoms annotated by a constant or a variable.[2]

Unless explicitly stated otherwise, we will assume that queries are not necessarily c-annotated, i.e. they may contain annotation variables. If Q is a query $? -\ A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k$ then $(\exists)Q$ will denote the existential closure of the conjunction of its body literals, $(\exists)\,(A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k)$.

Definition 14 A *constrained query* Q is a statement of the form:

$$? -\ A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k \,\&\, Constraint_Q\,,$$

where

$$A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k$$

is the *query-part* of Q and $Constraint_Q$ is its *constraint-part*. Here, each ρ_j is either a constant from $\mathcal{T}$ or an annotation variable. In the most general case, $Constraint_Q$ can be any conjunction of decidable predicates[3] over $\mathcal{T}$, but in all specific theories considered in Sections 5 and 6, constraints will be of the form

$$\tau_1 \geq \kappa_1 \,\&\ldots\&\, \tau_n \geq \kappa_n \,\&\, \alpha_1 = \beta_1 \,\&\ldots\&\, \alpha_m = \beta_m,$$

where the τ_i's, κ_j's, α_k's, β_l's may be arbitrary term annotations (clearly, $\leq$ and $=$ are decidable predicates over any lattice with a computable $\sqcup$). □

[2] Notice that, as usual, a query can be viewed as a headless clause $\leftarrow (A_1 : \rho_1 \,\&\ldots\&\, A_k : \rho_k)$. This explains our restriction on query annotations, since only c- and v-annotations are allowed in clause bodies.

[3] A k-ary predicate p over $\mathcal{T}$ is *decidable* if there is an algorithm $\mathcal{A}$ such that for every tuple $(\kappa_1, \ldots, \kappa_k) \in \mathcal{T}^k$, $\mathcal{A}$ can correctly decide whether $p(\kappa_1, \ldots, \kappa_k)$ is true over $\mathcal{T}$ or not. In particular, we assume that constraints do not involve quantifiers.

Similarly, we can define *constrained clauses* of the form

$$A : \psi \leftarrow B_1 : \mu_1 \;\&\ldots\&\; B_m : \mu_m \;\&\; Constraint_C,$$

which is a clause in the old sense, augmented by a constraint, $Constraint_C$. The notion of satisfaction of such clauses by an interpretation is immediate.

Definition 15 Suppose C is a constrained clause $A : \psi \leftarrow B_1 : \mu_1 \& \ldots \& B_m : \mu_m \& Constraint_C$ and Q is a query $?- \; A_1 : \rho_1 \;\&\ldots\&\; A_k : \rho_k \;\&\; Constraint_Q$ such that:

1. C and Q have no (annotation or object) variables in common; and
2. A_i and A are unifiable via mgu θ.

Then the *resolvent* of Q and C with respect to A_i is the constrained query Q' below:

$$?- \; [A_1 : \rho_1 \,\& A_{i-1} : \rho_{i-1} \& \; B_1 : \mu_1 \;\&\ldots\&\; B_m : \mu_m \;\&\; A_{i+1} : \rho_{i+1} \;\&\ldots\&\; A_k : \rho_k]\theta$$

$$\& \; (Constraint_C \;\&\; \psi \geq \rho_i \;\& Constraint_Q). \qquad (6)$$

In the above, if θ is not required to be a most general unifier (i.e. θ is allowed to be any unifier), then Q' is called an *unrestricted resolvent* of C and Q with respect to A_i. □

A constraint $\mathbf{C}$ is *solvable* with respect to the semilattice $\mathcal{T}$ and the set $\mathcal{F}$ of interpreted annotation functions if and only if there is an assignment σ of elements in $\mathcal{T}$ to the annotation variables of $\mathbf{C}$, such that $\mathbf{C}$ has a solution with respect to $\mathcal{T}$ and $\mathcal{F}$ ($\mathbf{C}\sigma$ is evaluated using the intended interpretation of the annotation functions in $\mathcal{F}$).

There is an important class of constraints, called normal constraints, which we define next. A constraint $(\tau_1 \geq \kappa_1 \;\&\ldots\&\; \tau_n \geq \kappa_n)$ is *normal* if

1. Each κ_i is an annotation variable or a constant;
2. If κ_i is a variable, then it does not occur in τ_1, ..., τ_i.

Queries, clauses, and GAPs constrained by normal constraints are called *normal* queries, clauses, and GAPs, respectively.

Lemma 2 Suppose $\mathcal{T}$ is a lattice (not necessarily complete). Then

1. If C and Q are a normal clause and a normal query, respectively, then the resolvent of Q and C is a normal query.
2. Satisfiability of any normal constraint is decidable.

Proof: (1) Notice that ρ_i in Equation (6) does not appear in $Constraint_C$ and in ψ, since variables have been renamed before performing the resolution step. Similarly, none of the right-hand sides of inequalities in $Constraint_Q$ appears in ψ. Therefore, if both $Constraint_C$ and $Constraint_Q$ in the above equation are normal, the constraint in the resolvent, ($Constraint_C$ &

$\psi \geq \rho_i \; \& Constraint_Q$), is also normal. Observe that the order of constraints in Equation (6) above is crucial.

(2) Let $\mathbf{C}$ be a normal constraint of the form $\tau_1 \geq \kappa_1 \& \ldots \& \tau_n \geq \kappa_n$. Without loss of generality, we assume that the inequalities in $\mathbf{C}$ with identical variable in the right-hand side are grouped together, i.e. if κ_i and κ_j are the same variable, then for all s, $i \leq s \leq j$, κ_s is the same variable as κ_i and κ_j. This grouping can be achieved by the following re-grouping operation: Suppose $\mathbf{C}$ has a subsequence of conjuncts $\ldots \& \tau^1 \geq x \& \ldots \& \tau^2 \geq y \& \ldots \& \tau^3 \geq x \ldots$. Because of normality of $\mathbf{C}$, x does not appear in τ^2, and hence the whole block of inequalities between $\tau^1 \geq x$ and $\tau^3 \geq x$ can be moved in front of $\tau^1 \geq x$. Clearly, the resulting constraint will still be normal and equivalent to $\mathbf{C}$. Repeating this process, we will achieve the desired grouping of conjuncts in $\mathbf{C}$.

The test for satisfiability of $\mathbf{C}$ in $\mathcal{T}$ now follows:

a. If $\mathbf{C}$ is an empty constraint, return (**satisfiable**).

b. Let $i_0 \geq 1$ be the maximal integer such that κ_{i_0} is the same symbol as κ_1. Substitute $\top$ for each of the variables occurring in τ_1, ..., τ_{i_0} (the substitution must be done throughout $\mathbf{C}$). Since $\mathbf{C}$ is normal, none of these variables appears on the right-hand side of $\mathbf{C}$. Let the resulting constraint and annotation terms be also denoted by $\mathbf{C}$ and τ_i's, respectively. Notice that now each of the τ_1, ..., τ_{i_0} can be evaluated to an element of $\mathcal{T}$.

c. If κ_1 is a constant then

   If $\tau_1 \geq \kappa_1 \& \ldots \& \tau_{i_0} \geq \kappa_{i_0}$ is false in $\mathcal{T}$

   then return(**unsatisfiable**)

   /* The if-condition is verifiable since the τ_i's are ground */

   /* Otherwise */

   Set $\mathbf{C}$ to $\tau_{i_0+1} \geq \kappa_{i_0+1} \& \ldots \& \tau_n \geq \kappa_n$

   Rearrange indices of the τ_i's and κ_i's in $\mathbf{C}$ so that they will start with 1, and then go to (a).

d. If κ_1 is a variable then replace it by the greatest lower bound of τ_1, ..., τ_{i_0}, which exists since it was assumed that $\mathcal{T}$ is a lattice. This replacement should be done everywhere in $\mathbf{C}$. Then delete the conjuncts $\tau_1 \geq \kappa_1 \& \ldots \& \tau_{i_0} \geq \kappa_{i_0}$ from $\mathbf{C}$, as in (c), and go to (a).

Correctness of this algorithm follows immediately from the fact that all functions used in the τ_j in $\mathbf{C}$ are monotonic (see assumptions at the beginning of Section 2). Termination of the algorithm follows from the assumption that all functions in $\mathcal{F}$ are computable. $\square$.

It is easy to see that the above result can be strengthened somewhat by replacing the requirement that $\mathcal{T}$ must be a lattice by a weaker requirement that every finite subset of $\mathcal{T}$ has a (not necessarily greatest) lower bound. However, then we will have to restrict $\mathbf{C}$ to be not only a normal constraint but also such that all conjuncts $\tau_i \geq \kappa_i$, where κ_i is a constant, appear in front of $\mathbf{C}$. Indeed, it is easy to verify that $\mathcal{T}$ needs be a lattice only in step (d) of Lemma 2 and that this step will go through under the modified requirements. The next result says that if $\mathcal{T}$ is finite then the requirement of normality can be dropped altogether.

Lemma 3 For finite semilattices $\mathcal{T}$, satisfiability of every constraint is decidable.

Proof: Suppose **C** is a constraint over $\mathcal{T}$. Let GRD be the set of all instances of this constraint obtained by (uniformly) replacing all occurrences of annotation variables by annotation constants (in particular, constraints in GRD are free of annotation variables). As $\mathcal{T}$ is finite, GRD is a finite set of ground constraints (since constraints contain no quantifiers, by definition). Now, **C** is solvable if and only if *some* constraint in GRD is solvable. But for ground constraints satisfaction is obviously decidable since they are conjunctions of ground atoms involving decidable predicates only. $\square$

Of course, the algorithm of Lemma 3 is impractical and we just used it as a decidability argument; efficient algorithms for constraint solving over $\mathcal{T}$ are presented in [22].

Definition 16 A *deduction* of a constrained query Q_0 from a GAP P is a sequence:

$$Q_0, \langle C_0, \theta_0 \rangle, Q_1, \ldots, Q_n, \langle C_n, \theta_n \rangle, Q_{n+1}$$

such that:

1. Q_{i+1} is a resolvent of Q_i and C_i via mgu θ_i; and
2. C_i is a reductant of P that contains no variables in common with Q_i.

When the θ_i's in the above deduction are required to be unifiers but not necessarily mgu's (i.e. the Q_i's, $i \geq 1$, are only required to be unrestricted resolvents), then the above deduction is called an *unrestricted deduction.*

Definition 17 The deduction $\Re = Q_0, \langle C_0, \theta_0 \rangle, Q_1, \ldots, Q_n, \langle C_n, \theta_n \rangle, Q_{n+1}$ of the query Q_0 from P is a *refutation* if and only if

1. Q_{n+1}, the resolvent of Q_n and C_n, has an empty query-part (i.e., Q_{n+1} is just a constraint); and
2. Q_{n+1} is solvable with respect to the lattice $\mathcal{T}$ and the set of annotation functions $\mathcal{F}$.

In what follows, we will use SOL($\Re$) to denote the set of solutions of the constraint-part of Q_{n+1}.

Unrestricted refutation is defined similarly (where "deduction" must be replaced by "unrestricted deduction").

The implementation of the above refutation procedure hinges upon two things:

- The ability to solve lattice constraints; and
- The ability to restrict the choice of reductants.

Studying the ways of solving constraints is beyond the scope of this paper; [22] deals with efficient serial and parallel algorithms for this task. The need to use reductants of P rather then just the clauses of P is another major obstacle. Indeed, the main appeal of SLD-resolution is that the choice of clauses that need to be considered is restricted to the current goal and the program clauses. However, if reductants are to be used, one may generate an infinite number of them out of a finite set of program clauses. Therefore, for GAPs, SLD-resolution with reduction is no better than the general resolution. Fortunately, for a large class of semilattices, we can effectively limit the number of reductants to be considered in refutations.

Definition 18 An upper semilattice $\mathcal{T}$ is *n-wide* if for every *finite*[4] set $E \in \mathcal{T}$, there is a finite subset $E_0 \subseteq E$ of at most n elements such that $\sqcup E_0 = \sqcup E$.
A *n-reductant* of a program P is a reductant involving no more than n clauses of P.

Many popular semilattices have finite width. Clearly, all finite semilattices are of this kind. Among the infinite ones, the semilattice of the form $[0,1]^n$ has width n (here $(a_1, ..., a_n) \sqcup (b_1, ..., b_n) = (a_1 \sqcup b_1, ..., a_n \sqcup b_n)$). In particular, $[0,1]$ and $[0,1]^2$ are frequently used in expert systems. To show that, e.g., $[0,1]^2$ is 2-wide, let $\alpha_1 = [a_1, b_1]$, ..., $\alpha_k = [a_k, b_k]$ be a finite set of pairs of real numbers in the interval $[0,1]$. Let a_i (resp., b_j) be the maximal element among the a_1, ..., a_k (resp., b_1, ..., b_k). Then, $\alpha_i \sqcup \alpha_j = \sqcup\{\alpha_1, ..., \alpha_k\}$, which proves that $[0,1]^2$ is 2-wide.

As we shall see, if $\mathcal{T}$ is n-wide, then in building refutations it suffices to consider n-reductants only. This limits the choice of clauses to resolve with to a *finite* set of n-reductants.

Theorem 5 (Soundness) Suppose P is a GAP and Q is a constrained query such that

$$Q_0, \langle C_0, \theta_0 \rangle, Q_1, \ldots, Q_n, \langle C_n, \theta_n \rangle, Q_{n+1}$$

is a refutation of Q_0 from the GAP P. Let σ be any solution for the constraint-part of Q_{n+1}. Then $Q_0\sigma$ is an annotation-variable-free query obtained by replacing all annotation variables in Q_0 by the annotation constants specified in σ. We claim that:

$$P \models (\forall)(Q_0\sigma)\theta_0\theta_1 \cdots \theta_n$$

(recall that $(\forall)Q_0$ denotes a conjunction of body literals of query Q_0, universally quantified).

Proof: We proceed by induction on n, the length of the refutation of Q_0 from P.

Base Case: $n = 1$. Then Q_0 contains exactly one annotated atom, denoted $A : \mu$. Hence, C_0 is of the form

$$D_0 : \rho_0 \leftarrow$$

such that $A\theta_0 = D_0\theta_0$ and the constraint $(\rho_0 \geq \mu)$ is solvable. Let σ be any solution of this constraint and let I be a model of P. Then I is a model of C_0 (as P entails all its reductants) and, in particular,

$$I \models (\forall)(D_0 : \rho)$$

where ρ is obtained from ρ_0 by instantiating all annotation variables to $\top$, the top element of $\mathcal{T}$. In particular,

$$I \models (\forall)A_0\theta : \rho$$

and as $\rho = \rho_0\sigma \geq \mu\sigma$ (due to the monotonicity of annotation functions in $\mathcal{F}$), it follows that

$$I \models (\forall)A\theta_0 : \mu.$$

Inductive Step: $n > 1$. In this case,

$$Q_1, \langle C_1, \theta_1 \rangle, Q_2, \ldots, Q_n, \langle C_n, \theta_n \rangle, Q_{n+1}$$

is a refutation of Q_1, where the body of Q_{n+1} is a pure constraint. Let σ be any solution of the constraint-part of Q_{n+1}. Suppose Q_0 is

$$? -\ A_1 : \mu_1 \,\&\, \ldots \,\&\, A_k : \mu_k \,\&\, Constraint_{Q_0}.$$

[4] Finiteness is crucial here; we do not care if infinite sets do not have the property described in this definition.

If C_0 is of the form

$$A : \rho \leftarrow B_1 : \rho_1 \,\&\, \ldots \,\&\, B_r : \rho_r$$

and $A\theta_0 = A_i\theta_0$ (i.e. Q_0 and C_0 resolve on atom $A_i : \rho_i$), then Q_1 is of the form:

$$? - \;\; A_1 : \mu_1 \,\&\, \ldots \,\&\, A_{i-1} : \mu_{i-1} \,\&\, B_1 : \rho_1 \,\&\, \ldots \,\&\, B_r : \rho_r \& A_{i+1} : \mu_{i+1} \,\&\, \ldots \,\&\, A_n : \mu_n$$

$$\& \; (Constraint_{Q_0} \,\&\, \rho \geq \mu_i).$$

By the inductive hypothesis, we may assume that

$$P \models (\forall)(Q_1 \sigma \theta_1 \ldots \theta_n).$$

Suppose now that I is a model of P. Then, as

$$I \models (\forall)(Q_1 \sigma \theta_1 \ldots \theta_n), \tag{7}$$

it follows that

$$I \models (\forall)(B_1 : \rho_1 \sigma \,\&\, \ldots \,\&\, B_r : \rho_r \sigma)\theta_1 \ldots \theta_n.$$

Hence, since $I \models C_0$ (as C_0 is a reductant of P):

$$I \models (\forall) A\theta_1 \ldots \theta_n : \rho\sigma.$$

As $A\theta_0 = A_i\theta_0$, we conclude that:

$$I \models (\forall) A_i\theta_0 \ldots \theta_n : \rho\sigma.$$

Recall that the constraint-part of Q_1 (and hence of Q_{n+1}) contains the constraint $\rho \geq \mu_i$. As σ is a solution of the constraint Q_{n+1}, it follows that $\rho\sigma \geq \mu_i\sigma$ holds true in $\mathcal{T}$. Thus,

$$I \models (\forall) A_i\theta_0 \ldots \theta_n : \mu_i.$$

Finally, this and (7) imply

$$I \models (\forall) Q_0 \sigma \theta_0 \cdots \theta_n,$$

which completes our proof. $\Box$

Lemma 4 (Mgu Lemma) Suppose P is a GAP and Q is a query. Suppose there is an *unrestricted* refutation $\Re$ of Q such that $\sigma \in \mathrm{SOL}(\Re)$ (SOL was defined in Definition 17). Then there is a refutation $\Re'$ of Q such that $\sigma \in \mathrm{SOL}(\Re')$.

Proof: Similar to the proof of the mgu lemma in classical logic programming (Lloyd, [35]). $\Box$

Lemma 5 (Lifting Lemma) Suppose P is a GAP and Q is a normal query. Suppose σ is an assignment of c-annotations to some (not necessarily all) annotation variables in Q and let θ be a substitution for object variables. If there is a refutation $\Re$ of $Q\sigma\theta$ from P, then there is a refutation $\Re'$ of Q from P.

Proof: Similar to the proof of the classical lifting lemma (cf. Lloyd [35]). $\Box$

Theorem 6 (Completeness Theorem) Suppose that $\mathcal{T}$ is a lattice, P is a GAP and Q is a *normal* query. Suppose $P \models (\exists)Q$. Then there is a refutation of Q from P. Moreover, if $\mathcal{T}$ is n-wide then Q can be refuted solely using n-reductants of P.

Proof: Suppose Q is

$$? - \; A_1 : \mu_1 \,\&\ldots\&\, A_k : \mu_k \,\&\, \mathbf{C}_Q,$$

where $\mathbf{C}_Q$ is the constraint-part of Q. As $P \models (\exists)Q$, it follows from Theorem 2 that $T_P \uparrow \omega \models (\exists)Q$ and hence, there is an integer n such that $T_P \uparrow n \models (\exists)Q$. We first proceed by induction on n to show that there is an *unrestricted* refutation of Q from P.

Base Case: $m = 1$. In this case, $k = 1$ and there is a reductant C of P of the form:

$$A_1 : \rho \leftarrow$$

such that the constraint $C_1 \equiv (\mathbf{C}_Q \,\&\, \rho \geq \mu_1)$ is solvable. Hence, $Q, \langle C, \theta\rangle, Q_1$, where Q_1 is the goal $\;? - \; C_1$, is an unrestricted refutation of Q from P.

Inductive Step: $m = n + 1$. Suppose now that $T_P \uparrow (n + 1) \models (\exists)Q$. In particular, there is a variable-free instance $Q\sigma\theta$ of Q (here, σ is an assignment of c-annotations to annotation variables and θ is a ground substitution for object variables) such that $T_P \uparrow (n+1) \models Q\sigma\theta$. By the definition of T_P (Definition 9), this implies that for each $1 \leq i \leq k$, there is a reductant of P, denoted C_i, having a ground instance of the form

$$A_i : \rho_i \leftarrow B^i_1 : \psi^i_1 \,\&\ldots\&\, B^i_{r_i} : \psi^i_{r_i}$$

such that $T_P \uparrow n \models (B^i_1 : \psi^i_j \,\&\ldots\&\, B^i_{r_i} : \psi^i_{r_i})$ and $(\rho_1 \geq \mu_1 \& \ldots \& \rho_k \geq \mu_k \,\&\, \mathbf{C}_Q)$ is solvable.

Furthermore, if $\mathcal{T}$ is n-wide, then we can choose each C_i above to be a n-reductant. Indeed, in Definition 9, in order to obtain $T_P(I)(A)$ one needs to take all possible finite least upper bounds of the elements of the set

$$\{f(\mu_1,\ldots,\mu_n) \mid A : f(\mu_1,\ldots,\mu_n) \leftarrow B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n \text{ is in } SGI(P), \text{ and } I \models (B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n)\},$$

which amounts to taking all possible reductants of $SGI(P)$. However, if $\mathcal{T}$ is n-wide, one only needs to take least upper bounds of up to n elements of the above set, which amounts to taking n-reductants only.

As $\mathbf{C}_Q$ is normal, and as annotation variables are renamed prior to resolution, $(\rho_1 \geq \mu_1 \& \ldots \& \rho_k \geq \mu_k \,\&\, \mathbf{C}_Q)$ is a normal query. Hence, for all $1 \leq i \leq k$ and for all $1 \leq j \leq r_i$, we may assume, by the induction hypothesis, that there is an unrestricted refutation, denoted $\Re^i_j$ of $B^i_j : \psi^i_j$. Then, the k resolution steps that involve Q and the clauses $C_1, \ldots, C_k$ above, followed by

$$\Re \;=\; \Re^1_1 \Re^1_2 \ldots \Re^1_{r_1} \ldots \Re^k_1 \Re^k_2 \ldots \Re^k_{r_k}$$

is an unrestricted refutation of $Q\sigma\theta$ from P. By the Lifting Lemma, there is an unrestricted refutation of Q from P. This completes the proof of the inductive step.

Thus, we know that there is an unrestricted refutation of Q from P. By the Mgu Lemma, it now follows that there is a refutation of Q from P. $\Box$

It should be noted that there is no similar completeness result for r-entailment, even in the case of acceptable GAPs.

Example 7 Consider the following program P:

$$p : 0$$
$$p : \frac{x+1}{2} \leftarrow p : x$$

The query $?-p:1$ cannot be refuted by the above proof procedure, even though $P \models^r p:1$. Notice that $P \not\models p:1$ and so this argument is applicable to the restricted semantics but not to the general one. $\square$

In fact, Example 7 shows that Herbrand's theorem does not hold for r-entailment even for acceptable GAPs, which indicates that there is no sufficiently general proof procedure for the restricted semantics.

We see that there is a close relationship between annotated logic programming and constraint logic programming [30]. As will be shown later, there is also a close connection between annotated programs and certain fragments of temporal logics. Thus, there is hope that in the future a single unifying framework for multivalued, temporal, and constraint logic programming will emerge.

In related works, Morishita [36] and Subrahmanian [44] have also studied multivalued logic programming where annotations were associated with clauses, rather than with individual atoms. Morishita's framework is as follows: Associated with each atom is a lattice, and associated with each clause is a function that maps the product of the lattices associated with the atoms in the body of the clause to the lattice associated with the head. Soundness of the proof procedure is established for queries that have a finite AND/OR tree associated with them (cf. [36, Theorem 4.11]). This restriction is not needed in our work. It must also be pointed out that our ideal-theoretic semantics differs from Morishita's semantics. Hence our completeness result applies to Example 7, whereas Morishita's completeness result is inapplicable to that example. Results on query processing procedures for programs whose clauses are c-annotated were obtained in Subrahmanian [44] and Kifer and Lozinskii [32, 33].

5 Multivalued Logics and GAPs

The principal aim of this section is to show that quantitative logic programming as proposed by van Emden[47] and also the bilattice-based logic programs of Fitting [18], all fit into the framework of GAPs.

5.1 van Emden's Quantitative Deduction

A *quantitative rule* [47] is of the form:

$$r : \qquad A \leftarrow B_1 \,\&\, \ldots \,\&\, B_k,$$

where $r \in (0,1]$. A *quantitative rule set* (QRS) is a collection of finitely many rules. Interpretations map ground atoms into the unit interval $[0,1]$. Interpretations of QRSs, as defined in [47], are the same as r-interpretations of GAPs over the lattice $\mathcal{T} = [0,1]$ (with the standard ordering on $[0,1]$). *Throughout this section, the word "interpretation" will mean "r-interpretation."*

Associated with a QRS, P, is an operator S_P that maps interpretations to interpretations, and is defined as follows:

$$S_P(I)(A) = \sqcup\{r \times k \mid r: \quad A \leftarrow B_1 \,\&\ldots\&\, B_n \text{ is a ground instance of a clause in } P \text{ and } \min\{I(B_1),\ldots,I(B_n)\} = k\}.$$

If P is a QRS, then it can be translated into a GAP, $\mathbf{tr}(P)$, as follows:

$$\mathbf{tr}(P) = \{A : r \times \min\{T_1,\ldots,T_n\} \leftarrow B_1 : T_1 \,\&\ldots\&\, B_n : T_n \mid \text{where } r: \quad A \leftarrow B_1 \,\&\ldots\&\, B_n \text{ is a clause in } P\}.$$

Note that according to this translation, all QRSs get translated into GAPs whose bodies contain only v-annotated literals. Hence, for any QRS P, $\mathbf{tr}(P)$ is an acceptable GAP and thus $R_{\mathbf{tr}(P)}$ has the fixpoint reachability property.

Theorem 7 Suppose P is a QRS. Then $S_P = R_{\mathbf{tr}(P)}$.

Proof: The proof is a direct consequence of the definition of $\mathbf{tr}(P)$. □

Theorem 7 shows, in particular, that $lfp(S_P) = lfp(R_{\mathbf{tr}(P)})$, and since $\mathbf{tr}(P)$ is always acceptable, we also have that $lfp(S_P) = \coprod lfp(T_{\mathbf{tr}(P)})$.

Furthermore, for *finite* programs (i.e. programs with a finite set of rules and facts) we can show that $\coprod lfp(T_{\mathbf{tr}(P)}) = lfp(T_{\mathbf{tr}(P)})$. Indeed, because of the special form of their annotation functions, the rules in $\mathbf{tr}(P)$ never produce new annotation constants when applied in the computation of $T_{\mathbf{tr}(P)} \uparrow \omega$. Therefore, for any atom A, $(T_{\mathbf{tr}(P)} \uparrow \omega)(A)$ will be a finitely generated ideal. Since every such ideal is principal, the $\coprod$ operator has no effect on $lfp(T_{\mathbf{tr}(P)})$. Furthermore, note that the clauses in $\mathbf{tr}(P)$ have empty constraint-parts and thus are normal.

As a consequence of the fact that $lfp(S_P) = lfp(T_{\mathbf{tr}(P)})$, one can study the least model of a QRS P by studying the least model of the GAP $\mathbf{tr}(P)$, and the remark about normality of $\mathbf{tr}(P)$ in the previous paragraph implies that the proof procedure for GAPs described in the preceding section yields a sound and complete proof procedure for answering existential queries to QRS. This improves upon van Emden's "weak soundness and completeness" results in two ways:

1. van Emden's evaluation procedure [47] works under the conditions that the AND/OR tree associated with a program P and a query Q is finite. No such restriction is needed here.

2. van Emden's proof procedure applies to ground queries. The procedure for GAPs described in this paper applies to non-ground existential queries as well.

Thus, the theorems about GAPs given in the previous section shed new light on the operational aspects of van Emden's QRSs. However, unlike QRSs, GAPs are not restricted to the interval [0,1] of truth values, and the results are applicable to any multi-valued logic based on a complete lattice that possesses Henkin's existential property.

5.2 Bilattice-valued Logics

For the purpose of this section, we assume that the reader is familiar with the basics of Fitting's theory of bilattice-based logic programming developed in [18]. Bilattices, due to Ginsberg [23], provide an elegant epistemological framework for studying multivalued logics. Intuitively, a bilattice (also known as an interlaced bilattice) is a set $\Re$ having two orderings: the knowledge order, $\leq_k$, and the truth order, $\leq_t$, such that $(\Re, \leq_k, \sqcap_k, \sqcup_k)$ and $(\Re, \leq_t, \sqcap_t, \sqcup_t)$ are both complete lattices. In addition, meets and joins with respect to $\leq_k$ are monotone with respect to $\leq_t$ and vice versa. Fitting [17, 18] has developed a theory of fixpoints for logic programs whose associated set of truth values forms a bilattice.

In Fitting's formulation, the syntax of a bilattice logic program is similar to that of an ordinary logic program, except that the body of a clause may be an arbitrary first order formula constructed out of $\wedge_t$, $\vee_t$ ("and" and "or" with respect to $\leq_t$), $\wedge_k$, $\vee_k$ ("and" and "or" relative to $\leq_k$), and $\neg$. Negation is interpreted as a unary function on truth values such that $\mu_1 \leq_k \mu_2$ if and only if $\neg\mu_1 \leq_k \neg\mu_2$, and $\mu_1 \leq_t \mu_2$ if and only if $\neg\mu_2 \leq_t \neg\mu_1$.

Unlike annotated clauses that essentially have a two-valued satisfaction relation, in bilattice-based logics formulae may assume any truth value from $\Re$. The fundamental role of the truth order is to allow defining the logical connectives $\wedge_t$, $\vee_t$, and $\neg$ without having to bother with specifics of the set of truth values $\Re$. In contrast, the truth order plays no role in the semantics of annotated logics. The fundamental role of the knowledge order is to give meaning to logical implication, and it is used in a similar way by both annotated and bilattice-based logics. In a sense, the results of this section show that Fitting's theory of bilattice-based logic programming uses the knowledge order in a more essential way than the truth order.

Interpretations of Fitting's programs are the same as r-interpretations of GAPs. In other words, they are functions from Herbrand base of P to $\Re$. These functions are extended to arbitrary formulas by distributing them through the connectives $\wedge_t$, $\vee_t$, $\wedge_k$, $\vee_t$, and $\neg$. Associated with a program, P, is an operator V_P that maps interpretations to interpretations as follows:

$$V_P(I)(A) = \sqcup\{\mu \mid A \leftarrow L_1 \,\&\ldots\&\, L_n \text{ is a ground instance of a clause in } P \text{ and } \mu = I(L_1 \,\&\ldots\&\, L_n)\}.$$

Given a bilattice-based logic program, we can translate it into a GAP, denoted $\mathbf{bl}(P)$, clause-wise as follows: Let C denote a clause $A \leftarrow Body(L_1, ..., L_n)$ in P, where $Body(L_1, ..., L_n)$ is a Fitting's formula involving atomic literals L_1, ..., L_n (different occurrences are considered as different literals). Then the corresponding clause, $\mathbf{bl}(C)$, has the form

$$A : f_{Body(L_1,\ldots,L_n)}(T_1, \ldots, T_n) \leftarrow L_1 : T_1 \,\&\ldots\&\, L_n : T_n,$$

where T_1, ..., T_2 are annotation variables and the function $f_{Body(L_1,\ldots,L_n)}$ is defined as follows:

1. If $Body(L_1, ..., L_n) = Body_1(L_1, ..., L_k) \vee_t Body_2(L_{k+1}, ..., L_n)$ then $f_{Body}(T_1, ..., T_n) = f_{Body_1}(T_1, ..., T_k) \sqcup_t f_{Body_2}(T_{k+1}, ..., T_n)$
2. If $Body(L_1, ..., L_n) = Body_1(L_1, ..., L_k) \vee_k Body_2(L_{k+1}, ..., L_n)$ then $f_{Body}(T_1, ..., T_n) = f_{Body_1}(T_1, ..., T_k) \sqcup_k f_{Body_2}(T_{k+1}, ..., T_n)$
3. If $Body(L_1, ..., L_n) = Body_1(L_1, ..., L_k) \wedge_t Body_2(L_{k+1}, ..., L_n)$ then $f_{Body}(T_1, ..., T_n) = f_{Body_1}(T_1, ..., T_k) \sqcap_t f_{Body_2}(T_{k+1}, ..., T_n)$

4. If $Body(L_1, ..., L_n) = Body_1(L_1, ..., L_k) \wedge_k Body_2(L_{k+1}, ..., L_n)$ then
$f_{Body}(T_1, ..., T_n) = f_{Body_1}(T_1, ..., T_k) \sqcap_k f_{Body_2}(T_{k+1}, ..., T_n)$

5. If $Body(L_1, ..., L_n) = \neg Body_1(L_1, ..., L_n)$ then
$f_{Body}(T_1, ..., T_n) = \neg f_{Body_1}(T_1, ..., T_k)$.

In the above, $\vee$ and $\wedge$ are logical connectives, while $\sqcup$ and $\sqcap$ are meet and join on $\Re$ relative to the appropriate orderings ($\leq_t$ or $\leq_k$). Furthermore, for the purpose of this translation, we assume that the GAP $\mathbf{bl}(P)$ uses the same negation operator $\neg : \Re \longrightarrow \Re$ as the one used by the bilattice-based program P.

Note that according to this translation, if P is a bilattice program, then $\mathbf{bl}(P)$ is a GAP such that all literals occurring in clause bodies are v-annotated. Hence, $\mathbf{bl}(P)$ is acceptable.

As mentioned earlier, the truth-order plays no role in the semantics of GAPs, while it does in the semantics of multivalued programs. However, as can be seen from the definition of $\mathbf{bl}(C)$, the truth-order of bilattices is *encoded* in the annotation functions of clauses of $\mathbf{bl}(C)$. This explains why GAPs can successfully simulate multivalued programs.

Theorem 8 Suppose P is a bilattice-based logic program. Then $V_P = R_{\mathbf{bl}(P)}$.

Proof: The proof is a straightforward consequence of the translation of P into a GAP. It is clear that the annotation functions in the GAP $\mathbf{bl}(P)$ have been designed precisely so that they would simulate the computation of truth values for the rule heads in the bilattice-based program P. □

Since rule bodies in $\mathbf{bl}(P)$ are v-annotated, such programs are acceptable and we have $lfp(V_P) = \coprod lfp(T_{\mathbf{bl}(P)})$. In general, this equality does not guarantee the existence of a complete proof theory for bilattice-based logic programs. However, for *distributive* bilattices such a proof procedure does exist.

A *distributive bilattice* [18] is a bilattice satisfying all twelve distributive laws for various combinations of the operators $\sqcup_k$, $\sqcup_t$, $\sqcap_k$, and $\sqcap_t$.

Now, as in the case of van Emden's QRSs, finite programs will result in only a finite number of annotations being mentioned in $\mathbf{bl}(P)$. Since in a distributive bilattice one can always convert any expression involving $\sqcup_k$, $\sqcup_t$, $\sqcap_k$, and $\sqcap_t$ into a normal form (e.g., a disjunctive normal form with k-operators inside and t-operators outside), such expressions can yield only a finite number of annotation constants, given a finite number of such constants as an input. Therefore, arguing as in the previous subsection, we conclude that $\coprod$ has no effect on $lfp(T_{\mathbf{bl}(P)})$, i.e. $lfp(V_P) = lfp(T_{\mathbf{bl}(P)})$. Furthermore, as with QRS, we can observe that constraints arising in $\mathbf{bl}(P)$ are normal and therefore the proof procedure developed for GAPs applies.

As a consequence, once again, just as in the case of van Emden's QRSs [47], we can study the semantics of bilattice logic programming by studying the semantics of the corresponding GAPs, and we can use the corresponding proof procedure to answer queries. There are several advantages in doing so:

1. Fitting did not define the notion of a model for his logic programs. However, when bilattice-based logic programs are converted into GAPs, this notion becomes clear: I is a model of a clause C in a bilattice-based program if and only if I is an r-model of the translation of C into a generalized annotated clause (see below).

2. We can use the operational semantics of GAPs developed in the previous section to process queries to bilattice-based programs. In [19], Fitting described a tableau-based proof procedure for his logic programs. This procedure is restricted to the case of the four-valued bilattice depicted in Figure 1. Since this bilattice is distributive, our results about the proof theory subsume the corresponding results in [19].

Now, the models of Fitting's logic programs can be defined as follows:

Definition 19 We say that I is a *model* of a ground clause

$$A \leftarrow Body$$

in a bilattice-based program if and only if $I(A) \geq_k I(Body)$. I is a model of a nonground such a clause, C, if and only if I is a model of each ground instance of C. As usual, I is a model of a bilattice-based logic program, P, if and only if I is a model of each clause in P.

Theorem 9 Let I be an interpretation of a bilattice-based program P. Then I is a model of P if and only if $V_P(I) \leq I$ if and only if I is an r-model of $\mathbf{bl}(P)$. □

As a consequence of the last two theorems, the models of a bilattice logic program P are also the models of the GAP $\mathbf{bl}(P)$, and their least models coincide. Thus, bilattice model theory can be studied through the model theory of GAPs. Moreover, in the case of finite databases and distributive bilattices, processing of existential queries to bilattice logic programs can be converted into the equivalent problem of processing existential queries for GAPs.

There is at least one intriguing result due to Fitting that does not fit in our framework. This is the elegant theorem that states the connection between the least fixed-point of V_P relative to the $\leq_k$ ordering, and the greatest fixed-point of the V_P operator in the $\leq_t$ ordering. This result cannot be obtained in the GAP framework simply because our formalization assumes only one ordering on $\mathcal{T}$, namely $\leq_k$.

6 Temporal Reasoning

There are several different kinds of temporal logics. One of the fundamental differences of opinion between temporal logicians concerns the issue of the nature of time. Various representations of time are possible – each representation is accompanied by a host of philosophical and epistemological arguments. Here, we will consider two widely accepted representations:

1. Linear time stretching either finitely or infinitely back.

2. Interval-based time where one considers "time periods." rather than "time points"

6.1 Reasoning about Linear Time using GAPs

Let us assume that $\mathcal{Z}$ is the set of all integers and that $\mathcal{Y}$ is some subset of $\mathcal{Z}$ that is upward closed under $\leq$, i.e. if $x \in \mathcal{Y}$ and $x \leq y$, then $y \in \mathcal{Y}$. Thus, $\mathcal{Y}$ may be $\mathcal{Z}$ itself (in which case we are assuming that the world has been around infinitely long) or $\mathcal{Y}$ may be, say, the set of all non-negative integers (in which case we are subscribing to the theory that the world was created at some time). So, in any case, let us assume that $\mathcal{Y}$ is fixed.

A *temporal Herbrand interpretation* I assigns a truth value to a ground atom, A, at each *time point*. Thus, I may say that A is true at times $1, 3, 5, \ldots$ and false at times $2, 4, 6, \ldots$. Intuitively, we may view an interpretation I as a mapping from the Herbrand Base, B_L, of our language L, to the power set, $\mathcal{P}(\mathcal{Y})$, of $\mathcal{Y}$. The epistemic interpretation of this is as follows:

> A is true at time t according to interpretation I if and only if $t \in I(A)$.

Now we can reason about time in the framework of GAPs by taking as our set $\mathcal{T}$ of truth values, the set $\mathcal{P}(\mathcal{Y})$ ordered by subset inclusion. Given any truth value μ, i.e. μ is a subset of $\mathcal{Y}$, define an annotation function *succ* as follows:

$$succ(\mu) = \{t + 1 \mid t \in \mu\}.$$

Thus, if proposition p is assigned μ by interpretation I, i.e. interpretation I says that p is true at all those times in μ, and if I assigns $succ(\mu)$ to q then I says that q is true at a time point $(t + 1)$ if p is true at time point t. For instance, the clause

$$get\_out\_of\_way : succ(\mu) \leftarrow see\_car\_coming : \mu$$

says that if you see a car coming at you at a certain moment then get out of its way at the next time moment. To say that p is true at all even times, we write

$$\begin{array}{rcl} p : \{0\} & \leftarrow & \\ p : succ(succ(\alpha)) & \leftarrow & p : \alpha. \end{array}$$

Baudinet [3] has developed a semantic framework for temporal logic programming, with $\mathcal{Y}$ taken to be the set of non-negative integers. According to this framework, a temporal logic program can be considered as a (possibly infinite) set of clauses of the form:

$$C : \qquad \bigcirc^{i_0} A_0 \leftarrow \bigcirc^{i_1} A_1 \,\&\ldots\&\, \bigcirc^{i_n} A_n,$$

where $\bigcirc$ is the modal operator *next* and $\bigcirc^j$ is a shorthand for j applications of $\bigcirc$. If A is an atom and $i \geq 0$, then $\bigcirc^i A$ is called a *next-atom*. An interpretation is just a collection of ground next-atoms.

The above clause C can be translated into the following c-annotated clause $\mathbf{an}(C)$:

$$A_0 : \{i_0\} \leftarrow A_1 : \{i_1\} \& \ldots \& A_n : \{i_n\}\,.$$

For a temporal program, P, $\mathbf{an}(P) = \{\mathbf{an}(C) | \, C \in P\}$.

To compare GAPs with [3], let $\mathcal{Y}$ be the set of all non-negative integers $\{0, 1, \ldots\}$, and $\mathcal{T} = \mathcal{P}(\mathcal{Y})$ be the lattice of all sets of non-negative integers ordered by inclusion, which we call the *temporal lattice*. Thus, for instance, Baudinet's interpretation $\{\bigcirc \bigcirc A\}$ that says that A is true at time 2 and all other propositions are false at all times is captured by our multivalued interpretation I defined as:

$$I(A) = \{2\} \text{ and for all } B \neq A,\ I(B) = \{\,\}$$

Formally, let I be a collection of Baudinet's next-atoms. The translation of I into an r-interpretation, $\mathbf{an}(I)$, for annotated logic is:

$$(\mathbf{an}(I))(A) = \{i \mid \bigcirc^i A \in I\}.$$

Baudinet then defines an operator, denoted Z_P, that maps sets of ground next-atoms to sets of ground next-atoms as follows:

$$Z_P(I) = \{\bigcirc^i A \mid \bigcirc^i A \leftarrow B_1 \,\&\ldots\&\, B_n \text{ is a ground instance of a clause in } P \text{ and } \{B_1, \ldots, B_n\} \subseteq I\}.$$

Likewise, in the context of annotated logics, we may consider the $R_{\mathbf{an}(P)}$ operator defined on GAPs:

Theorem 10 For any temporal program, P, and a Baudinet's interpretation, I,

$$\mathbf{an}(Z_P(I)) = R_{\mathbf{an}(P)}(\mathbf{an}(I)). \qquad \Box$$

The above theorem establishes that one way of studying Baudinet's temporal logic programming is within the framework of GAPs. For instance, as in Section 5, we can argue that for finite programs $R_{\mathbf{an}(P)} = T_{\mathbf{an}(P)}$ and this allows us to use the theorems in Section 4 to define a proof procedure for answering existential queries to temporal programs.

A more expressive temporal logic was proposed by Abadi and Manna in [1]. There, besides the "next" operator, $\bigcirc$, other modalities, such as "eventuality", $\Diamond$, and "always true", $\Box$, are allowed. The full temporal logic of [1] is, according to the authors, computationally expensive, and a fragment amenable to an efficient implementation by means of SLD resolution was proposed. This fragment consists of clauses of the following form. First, rule bodies are drawn from the class of formulas, $\mathcal{B}$, which contains all next-atoms, and is closed under conjunction and under the application of $\Diamond$. The *initial* clauses, i.e., clauses that are true at time 0 are of the form

1. $B \leftarrow A$; or
2. $\Box B \leftarrow A$,

where $A \in \mathcal{B}$ and B is a next-atom. The *permanent* clauses, i.e., rules that are always true, have the form $\Box(B \leftarrow A)$, where, as before, $A \in \mathcal{B}$ and B is a next-atom.

Let $\mathcal{Y}$ and $\mathcal{T}$ be the same as before. We have already shown how the next-atoms are represented using annotations. The necessity operator, $\Box$, corresponds to the topmost element $\top \in \mathcal{T}$ (which represents the whole set $\mathcal{Y}$). For instance, $\Box A$ is represented as $A : \top$. Finally, our annotated rules correspond to the *permanent* rules of [1].

We show below some simple examples of information that can be represented using the GAP formalism. Suppose in the sequel that $\mathcal{Y}$ is the set of all integers. For example, to say that if X is a president at time T, then X must have been rich at time $(T-1)$ (think of time units as "years"), we can say:

$$rich(X) : previous(\mu) \leftarrow president(X) : \mu,$$

where

$$previous(\mu) = \{t - 1 \mid t \in \mu\}$$

is an annotation function. In TEMPLOG ([1]), we could express this as:

$$\Box(rich(X) \leftarrow \bigcirc president(X)).$$

The statement: "if sometime she becomes a president then she must be rich" is written as

$$\Box(rich(X) \leftarrow \Diamond president(X))$$

in TEMPLOG, and as

$$rich(X) : \{0\} \leftarrow president(X) : \mu$$

in our logic.

On the other hand, the statement "if X is a life-long president, then X is a ruthless murderer", written as

$$\Box(murderer(X) \leftarrow \Box president(X))$$

in temporal logic, is not allowed in TEMPLOG. In contrast, representing this as a GAP is straightforward:

$$murderer(X) : \top \leftarrow president(X) : \top.$$

In general, since we allow arbitrary computable functions (subject to the restrictions of Section 4) in rule heads while both Baudinet and Abadi and Manna restrict the rule heads to be next-atoms (i.e. c-annotated literals, in our setting), GAPs can express several fancy temporal problems that are beyond the scope of [1, 3]. Also, we do not restrict bodies of temporal programs to be $\Box$-free, since atoms of the form $A : \top$ are perfectly acceptable.

There are, however, situations where GAPs are weaker than [1]. For instance, GAPs cannot express a clause with the following body: $\Diamond(p \wedge \Diamond q)$. Likewise, we cannot represent directly the initial clauses of Abadi and Manna, since GAP rules are permanently true.

This difficulty could be overcome by using metaprogramming techniques (e.g. [45]). In these formalisms, formulae can be encoded by terms, and thus can be reasoned about. For instance, if $< B \leftarrow A >$ is an encoding of a clause $B \leftarrow A$, then we could write $clause(< B \leftarrow A >) : \{0\}$, stating that the respective clause is true at time 0, which corresponds to the initial clause $B \leftarrow A$ of [1]. We will not discuss this issue any further in this paper. More information about encodings and logics for meta-reasoning can be found in [14, 37, 38, 27, 45].

6.2 Multivalued Temporal Reasoning

One advantage of the GAP formalism as opposed to Baudinet's and Abadi and Manna's is that it also allows one to deal with "epistemically inconsistent worlds", i.e. interpretations in which at certain times t the information about certain ground atoms is inconsistent. More generally, assume that now our domain of truth values is the set of functions from $\mathcal{Y}$ to some complete lattice $\mathcal{T}'$, i.e.

$$\mathcal{T} = (\mathcal{Y} \rightarrow \mathcal{T}').$$

For instance, we can take $\mathcal{T}'$ to be the four-valued Belnap's lattice [4] shown in Figure 1, or we can take $\mathcal{T}'$ to be the set $[0, 1] \times [0, 1]$ often used for modeling uncertainty in expert systems [7, 31, 18]. An interpretation is a map from B_L to $\mathcal{T}$. The underlying intuition is that an interpretation I assigns to any ground atom A, a function f_A from $\mathcal{Y}$ to $\mathcal{T}'$. If $\mathcal{T}'$ is the four-valued Belnap's lattice of Figure 1, and $f_A(3) = \top$, $f_A(4) = t$ then we can think of this as an assertion that at time 3, A was inconsistently defined, but at time 4 it became true.

6.3 Interval Based Temporal Logic

Let us assume that time is linearly represented by the set of all non-negative integers. In an interval based temporal logic, we can use closed intervals of integers as truth values. We use the notation:

$$[a, b] \; = \; \{n \mid a \leq n \leq b\}.$$

Let $\Re$ be the set

$$\Re = \{[a, b] \mid a \leq b \text{ and } a, b \text{ are non-negative integers}\}.$$

The intervals in $\Re$ are partially ordered by inclusion and we denote this ordering by $\leq_{\Re}$. Let us take $\mathcal{T}$ to be the set of subsets of $\Re$ such that every $\alpha \in \mathcal{T}$ satisfies the following two properties:

1. If $[a, b] \in \alpha$ and $[c, d] \subset [a, b]$ then $[c, d] \in \alpha$;
2. If $[a, b] \subset \cup\alpha$, where $\cup\alpha$ is the union of all intervals in α, then $[a, b] \in \alpha$.

Intuitively, assignment of a set $\alpha = \{[a_1, b_1], \ldots, [a_j, b_j], \ldots\}$ to an atom A by an interpretation I means that A is true in each of the intervals $[a_j, b_j]$. This epistemic interpretation of elements of $\mathcal{T}$ makes the above conditions self-explanatory: if an event takes place over a time interval then it also takes place over a subinterval; if an event takes place over a group of time intervals then this event also takes place over any interval that is covered by the group. In fact, it is easy to see that the second condition above implies the first one.

There exist several popular orderings of power-domains over partial orders. The one that is particularly useful to model time is the following: for $\alpha, \beta \in \mathcal{T}$, $\alpha \leq_{\mathcal{T}} \beta$ if and only if for every interval $\imath \in \alpha$ there exists an interval $\jmath \in \beta$ such that $\imath \leq_{\Re} \jmath$.

To see that GAPs can represent some forms of interval temporal reasoning, we show that the logic of Shoham [42, pp. 61–62] can be expressed within the GAP framework. Shoham extends classical propositional logic to an interval modal logic by adding six new modal connectives as follows:

1. $\langle\mathbf{A}\rangle$ is true in the interval $[t_1, t_2]$ if and only if there exists $t_2 \lhd t_3$ (here $\lhd$ denotes the regular order on integers) such that p is true in the interval $[t_2, t_3]$. Intuitively, modality $\langle A\rangle$ represents the intuition that p is true in some interval immediately following the current interval.
2. $\langle\mathbf{B}\rangle p$ is true in $[t_1, t_2]$ if and only if there exists a t_3 such that $t_1 \unlhd t_3 \lhd t_2$ and p is true in the interval $[t_1, t_3]$. Intuitively, $\langle\mathbf{B}\rangle$ represents the idea that p is true in some subinterval of the current interval that starts at the same time as the current interval.
3. $\langle\mathbf{E}\rangle p$ is true in the interval $[t_1, t_2]$ if and only if there is a t_3 such that $t_1 \lhd t_3 \unlhd t_2$ and p is true in the interval $[t_3, t_2]$. In other words, $\langle\mathbf{E}\rangle p$ becomes true earlier than p and remains true while p was true.
4. $\langle\overline{\mathbf{A}}\rangle p$ is true in $[t_1, t_2]$ if and only if there exists a t_3 such that $t_3 \lhd t_1$ and p is true in the interval $[t_3, t_1]$. Here $\langle\overline{\mathbf{A}}\rangle$ means that p is true at some interval ending immediately before the current interval starts.

5. $\langle\overline{\mathbf{B}}\rangle p$ is true in the interval $[t_1, t_2]$ if and only if there exists a t_3 such that $t_2 \lhd t_3$ and p is true in $[t_1, t_3]$. Intuitively, the $\langle\overline{\mathbf{B}}\rangle$ modality represents the intuition that p is true in some interval of which the current interval is the beginning.

6. $\langle\overline{\mathbf{E}}\rangle p$ is true in the interval $[t_1, t_2]$ if and only if there exists a t_3 such that $t_3 \lhd t_1$ and p is true in $[t_3, t_2]$. Intuitively, the $\langle\overline{\mathbf{E}}\rangle$ modality represents the intuition that p is true in some interval of which the current one is the end.

Given any n-ary predicate symbol p, we can express these six modalities as GAPs by using the following clauses:

$p_{\langle\mathbf{A}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; end(\alpha) = start(\beta) \;\&\; end(\alpha) < \beta.$
$p_{\langle\mathbf{B}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; start(\alpha) = start(\beta) \;\&\; \beta < \alpha.$
$p_{\langle\mathbf{E}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; end(\alpha) = end(\beta) \;\&\; \beta < \alpha.$
$p_{\langle\overline{\mathbf{A}}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; end(\beta) = start(\alpha) \;\&\; end(\beta) < \alpha.$
$p_{\langle\overline{\mathbf{B}}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; start(\alpha) = start(\beta) \;\&\; \alpha < \beta.$
$p_{\langle\overline{\mathbf{E}}\rangle}(X_1, \ldots, X_n) : \alpha \leftarrow p(X_1, \ldots, X_n) : \beta \;\&\; end(\alpha) = end(\beta) \;\&\; \alpha < \beta.$

In the above, $X_1, \ldots, X_n$ are object variables and α, β are annotation variables. Annotation functions *end* and *start* are defined as follows:

$$start(\alpha) = \{[a, a] \mid \; [a, b] \text{ is a } \mathit{maximum} \text{ interval in } \alpha\};$$

$$end(\beta) = \{[b, b] \mid \; [a, b] \text{ is a } \mathit{maximum} \text{ interval in } \beta\}.$$

An interval $[a, b] \in \alpha \in \mathcal{T}$ is *maximum* if it is not properly contained in another interval in α. Also note that satisfiability of constraints in the above clauses is decidable since this is a very simple case of linear programming, and thus the proof theory for GAPs applies.

The above translation precisely captures the intended meaning of the modal operators of Shoham [42] but, in general, we cannot simulate full-fledged interval-based temporal logics. This, of course, does not come as a surprise, since even the propositional temporal logic ITL has an undecidable validity problem [25].

7 Conclusions

There are many alternative formalisms for multivalued and temporal logic programming. However, the relationship between these different formalisms is not well understood. In this paper, we have made a first contribution towards the understanding of different methodologies for logic programming based on non-standard logics.

We have shown that annotated logics can serve as common grounds for several of the multivalued and temporal formalisms. Besides the theoretical interest, this has numerous practical benefits. First, already known results about GAPs can be used to obtain a direct characterization of certain kinds of temporal reasoning. Second, GAPs can be used to identify semi-decidable fragments of temporal logics by translating them into GAPs, as suggested in this paper (recall that the implication problem for GAPs is recursively enumerable). This is important because, we believe, logics with non-recursively enumerable implication problem cannot be effectively implemented on a computer. Third, it gives a proof theory to formalisms based on multivalued logics, such as [47, 18], which can be naturally translated into GAPs. Finally, when all else fails, one may prefer to program in terms of GAPs *directly* rather than using different formalisms (subsumed by GAPs) for different purposes.

Acknowledgements. We thank Mel Fitting, and Raymond Ng for useful suggestions on preliminary versions of this paper. S. Morishita pointed out to a mistake in an earlier version of Theorem 6. We are also grateful to the anonymous referees for many helpful suggestions.

References

[1] M. Abadi and Z. Manna. (1987) *Temporal Logic Programming*, Proc. 4th IEEE Symp. on Logic Programming, pp 4–16, Computer Society Press.

[2] K. R. Apt, H. A. Blair and A. Walker. (1988) *Towards a Theory of Declarative Knowledge*, in: Foundations of Deductive Databases and Logic Programming, (ed. Jack Minker), Morgan-Kauffman.

[3] M. Baudinet. (1989) *Temporal Logic Programming is Complete and Expressive*, Proc. of the ACM Conference on Principles of Programming Languages, Austin, Texas.

[4] N.D. Belnap. (1977) *A Useful Four-Valued Logic*, in G. Epstein and J.M. Dunn, eds. Modern Uses of Many-valued Logic, (eds. G. Epstein and J.M. Dunn), D. Reidel, pp. 8-37.

[5] H. A. Blair and V.S. Subrahmanian. (1989) *Paraconsistent Logic Programming*, Theoretical Computer Science, 68, pp 135–154.

[6] H. A. Blair and V. S. Subrahmanian. (1988) *Strong Completeness Results for Paraconsistent Logic Programming*, submitted.

[7] H. A. Blair and V. S. Subrahmanian. (1988) *Paraconsistent Foundations for Logic Programming*, Journal of Non-Classical Logic, 5, 2, pp 45–73.

[8] L. Farinas del Cerro. (1986) *Molog: A System That Extends Prolog with Modal Logic*, New Generation Computing.

[9] N.C.A. da Costa. (1974) *On the Theory of Inconsistent Formal Systems*, Notre Dame J. of Formal Logic, 15, pp. 497–510.

[10] N.C.A. da Costa and E.H. Alves. (1977) *A Semantical Analysis of the Calculi* $\mathcal{C}_n$, Notre Dame J. of Formal Logic, 18, pp. 621-630.

[11] N.C.A. da Costa and E.H. Alves. (1981) *Relations between Paraconsistent Logic and Many-valued Logic*, Bulletin of the Section of Logic, 10, pp. 185-191.

[12] N.C.A. da Costa, L. J. Henschen, J. J. Lu and V.S. Subrahmanian. (1989) *Automatic Theorem Proving in Paraconsistent Logics: Theory and Implementation*, in: Proc. 10th International Conference on Automated Deduction, Lecture Notes in Computer Science Vol. 449, pp. 72–86, July 1990.

[13] N.C.A. da Costa, V.S. Subrahmanian and C. Vago. (1989) *The Paraconsistent Logics* $\mathbf{P}\mathcal{T}$, to appear in: Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, Vol. 37, 1991.

[14] S. Feferman. (1984) *Toward Useful Type-Free Theories*, Journal of Symbolic Logic, 49, pp. 75-111.

[15] M. C. Fitting. (1987) *Bilattices and the Theory of Truth*, to appear in: Journal of Philosophical Logic.

[16] M. C. Fitting. (1987) *Enumeration Operators and Modular Logic Programming*, Journal of Logic Programming, 4, pp. 11–21.

[17] M. C. Fitting. (1988) *Logic Programming on a Topological Bilattice*, Fundamenta Informaticae, 11, pp. 209–218.

[18] M. C. Fitting. (1991) *Bilattices and the Semantics of Logic Programming*, Journal of Logic Programming, 1991. to appear.

[19] M. C. Fitting. (1989) *Negation as Refutation*, Proc. 4th Symp. on Logic in Computer Science, Asilomar, CA, pp. 63-70.

[20] D. M. Gabbay. (1987) *Modal and Temporal Logic Programming*, in: Temporal Logic in Computer Science, (ed. A. Galton), Academic Press.

[21] A. Galton. (1987) *Temporal Logic in Computer Science*, Academic Press.

[22] D.D. Ganguly and S. Ranka. (1990) *A Space Efficient Coding Algorithm for Lattice Computations*, draft manuscript.

[23] M. Ginsberg. (1988) *Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence*, Computational Intelligence, vol. 4, 1988, pp. 265–316.

[24] M. Ginsberg. (1990) *Bilattices and Modal Operators*, in: Proc. 1990 Intl. Conf. on Theoretical Aspects of Reasoning About Knowledge, Morgan Kaufmann.

[25] J.Y. Halpern, Z. Manna and B. Moszkowski. (1983) *A High-Level Semantics Based on Interval Logic*, Proc. ICALP 1983, pp 278–291, Springer.

[26] L.J. Henschen, J. J. Lu and V.S. Subrahmanian. (1989) *An Improved Resolution Procedure for Paraconsistent Logics* , submitted.

[27] P.M. Hill and J.W. Lloyd. (1988) *Analysis of Meta-Programs*, Tech. Report CS-88-08, University of Bristol, UK.

[28] M. Ishizuka. (1983) *Inference Methods Based on Extended Dempster and Shafer's Theory with Uncertainty/Fuzziness*, New Generation Computing, 1, 2, pps 159–168.

[29] M. Ishizuka and N. Kanai. (1985) *PROLOG-ELF Incorporating Fuzzy Logic*, New generation Computing, 3, pps 479–486.

[30] J. Jaffar and J-L Lassez. (1987) *Constraint Logic Programming*, POPL-87, Munich, W. Germany.

[31] M. Kifer and A. Li. (1988) *On the Semantics of Rule-Based Expert Systems with Uncertainty*, 2-nd Intl. Conf. on Database Theory, Springer Verlag LNCS 326, (eds. M. Gyssens, J. Paredaens, D. Van Gucht), Bruges, Belgium, pp. 102–117.

[32] M. Kifer and E.L. Lozinskii. (1989) *RI: A Logic for Reasoning with Inconsistency*, 4-th Symposium on Logic in Computer Science, Asilomar, CA, pp. 253–262.

[33] M. Kifer and E.L. Lozinskii. (1991) *A Logic for Reasoning with Inconsistency*, Journal of Automated Reasoning, Kluwer Academic Publ., 1991, to appear.

[34] M. Kifer and V.S. Subrahmanian. (1989) *On the Expressive Power of Annotated Logic Programs*, Proc. 1989 North American Conference on Logic Programming, Cleveland, Ohio, pp. 1069–1089, MIT Press.

[35] J.W. Lloyd. (1987) *Foundations of Logic Programming*, Springer-Verlag.

[36] S. Morishita. (1989) *A Unified Approach to Semantics of Multi-Valued Logic Programs*, Tech. Report RT 5006, IBM Tokyo, April 9th, 1990.

[37] D. Perlis. (1985) *Languages with Self-Reference I: Foundations* , Artificial Intelligence, 25, pp. 301-322.

[38] D. Perlis. (1988) *Languages with Self-Reference II: Knowledge,Belief, and Modality*, Artificial Intelligence, 34, pp. 179-212.

[39] D. C. Rine. (1986) *Some Relationships Between Logic Programming and Multiple-Valued Logic*, Proc. IEEE Intl. Symp. on Multiple-Valued Logic, pps 160–163.

[40] E. Shapiro. (1983) *Logic Programs with Uncertainties: A Tool for Implementing Expert Systems*, Proc. IJCAI '83, pp. 529–532, William Kauffman.

[41] E. Shortliffe. (1976) *Computer-Based Medical Consultation: MYCIN*, Elsevier Science.

[42] Y. Shoham. (1988) *Reasoning About Change*, MIT Press.

[43] V.S. Subrahmanian. (1987) *On the Semantics of Quantitative Logic Programs*, Proc. 4th IEEE Symposium on Logic Programming, Computer Society Press, Washington DC, pp. 173-182.

[44] V. S. Subrahmanian. (1990) *Mechanical Proof Procedures for Many Valued Lattice Based Logic Programming*, to appear in: Journal of Non-Classical Logic, 1990.

[45] V. S. Subrahmanian. (1988) *Foundations of Metalogic Programming*, in: Proc. of the Workshop on Meta-Programming in Logic Programming, (ed. J. Lloyd), Bristol, UK.

[46] V.S. Subrahmanian. (1989) *Paraconsistent Disjunctive Deductive Databases*, to appear in: Theoretical Computer Science, early 1992.

[47] M.H. van Emden. (1986) *Quantitative Deduction and its Fixpoint Theory*, Journal of Logic Programming, 4, 1, pp. 37-53.