# On the Relationship between Annotated Logic Programs and Nonmonotonic Formalisms*

Krishnaprasad Thirunarayan
Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435.
**Email**: tkprasad@cs.wright.edu
**Phone**: (513)-873-5134

**Running Title**: Annotated logic and Nonmonotonic Formalisms

July 23, 2008

*This work was supported in part by NSF grant IRI-9009587.

**Abstract**

In the past, we developed a semantics for a restricted annotated logic language for inheritance reasoning. Here we generalize it to annotated *Horn* logic programs. We first provide a formal account of the language, describe its semantics, and provide an interpreter written in Prolog for it. We then investigate its relationship to Belnap's 4-valued logic, Gelfond and Lifschitz's semantics for logic programs with negation, Brewka's prioritized default logics and other annotated logics due to Kifer *et al*.

# 1 Introduction

Pragmatically, the logic programming paradigm can be thought of as a reasonable trade-off between expressive power and computational efficiency for knowledge representation and reasoning. The relational database community can view deductive databases as a suitable generalization of the relational languages to support recursive queries. The knowledge engineers can consider Horn clause logic as a restricted nonmonotonic logic that is "tractable" [8, 14].

If a set of first-order logic sentences is inconsistent then every sentence in the language is a theorem. Given that the detection of inconsistency is computationally hard (actually, impossible in the general case), and that there is no simple way to resolve an inconsistency even when it is detected, [2, 3] pursued the approach of "weakening" the logic to make it robust with respect to contradictory information. [2, 3] describe a four-valued logic as an alternative to classical predicate calculus to localize the effect of a contradiction. Subsequently, [6] used these ideas to provide a semantics to logic programs. Similarly, [18] used this logic to provide a semantics to a special class of inheritance hierarchies. [8] generalized the four-valued logic to multi-valued logics, to make precise the book-keeping operations performed by a nonmonotonic reasoning system that handles both strict and defeasible information. In all the above cases the underlying language is the traditional first-order language. But the semantics, instead of being two-valued, is given by mapping literals to values (or evidences) that are ordered on the basis of both truth-content and information-content. For example, Tweety being a bird contributes positive or supporting evidence to its flying ability, while Tweety being a penguin contributes negative or defeating evidence to its flying ability. This can be represented by mapping $fly(Tweety)$ to the constants $+\mathsf{bird}$ and $-\mathsf{penguin}$ respectively. Furthermore, knowing that Tweety is a penguin is more informative than knowing that it is a bird. This can be represented by letting the constant $\mathsf{penguin}$ have higher information-content

than the constant bird.

In approaches described in [10, 11, 12, 17], the underlying first-order language is extended using annotations. These annotations can be thought of as abbreviated justifications or pieces of evidence that can be ordered on two different scales — the truth scale and the information scale. In general, one can represent strict/defeasible evidence and various levels of defeasible evidence using this approach. The aforementioned papers illustrate the use of annotated logics in rule-based expert systems with uncertainty [10], temporal reasoning problems [12], reasoning with inconsistency [11], etc. In [17] we used such a language to represent nonmonotonic multiple inheritance networks, and developed a semantics for it by amalgamating concepts from logic programming [1] and multi-valued logics [8]. The basic motivation for all these upgrades is to make the language more expressive while simultaneously retaining the computational advantages of efficiency and simplicity of the underlying logic programming machinery.

Section 3 generalizes our work in [17] in various different directions to obtain an enriched representation language. In particular, the rule-body in [17] contains just one literal (which is sufficient for our modest purposes of representing nonmonotonic inheritance networks). Here we extend it in the direction of [12] by permitting rules to be Horn clauses. That is, we permit rule-bodies to be conjunctions of literals and rules to be recursive. Furthermore, we let function symbols to appear in object terms, and classical negation in literals. In Section 2, we present a simple example of annotated logic programs. We describe their syntax in Section 3.1, and develop a model-theoretic semantics in Section 3.2. (Refer to [17] for an informal account of the motivation for the technical apparatus used.) For concreteness, we also present a meta-interpreter written in Prolog for answering queries with respect to annotated logic programs in Section 3.3. We then investigate, in detail, its relationship to Belnap's 4-valued logic, Gelfond and Lifschitz's semantics for logic programs with negation, Brewka's prioritized default logics, and other annotated logics due to Kifer *et al* in Section 3.4. Finally, we

summarize our conclusions in Section 4.

# 2 Motivating Example

We illustrate our approach by specifying the normal and the faulty behavior of an inverter as an annotated logic program. The value of the output of the inverter is "opposite" of its input if the inverter is normal; they are the same if there is a short-circuit fault.

Let $Id$ be an inverter with input node $in(Id)$ and output node $out(Id)$. Let $val(N)$ stand for the value of the node $N$. A logical 1 (resp. 0) on node $N$ is represented by a positive (resp. negative) evidence supporting $val(N)$. So, the normal behavior of the inverter can be formalized as: *If there exists positive (resp. negative) evidence for* $val(in(Id))$*, then there is negative (resp. positive) evidence for* $val(out(Id))$, and the short-circuit behavior as: *If there exists positive (resp. negative) evidence for* $val(in(Id))$*, then there is positive (resp. negative) evidence for* $val(out(Id))$. Furthermore, under short-circuit conditions, the latter rule should dominate the former one. In order to capture this, we associate with each atom an annotation (which is a signed constant) that summarizes the nature and the strength of evidence. Thus the normal behavior is now formalized as: *If there exists some positive (resp. negative) evidence for* $val(in(Id))$*, then there is negative (resp. positive) evidence of strength* norm *for* $val(out(Id))$, and the short-circuit behavior as: *If there exists some positive (resp. negative) evidence for* $val(in(Id))$*, then there is positive (resp. negative) evidence of strength* short *for* $val(out(Id))$. Furthermore, the evidential constants are ordered as: $\pm$norm $<$ $\pm$short to override conclusions derived under the assumptions of normality, in the presence of short-circuit fault. Note that a net positive (resp. negative) evidence stands for a logical 1 (resp. 0).

In the language of annontated logic programs, the inverter can be specified as follows, assuming that $+\epsilon$ (resp. $-\epsilon$) stands for the smallest positive (resp.

negative) evidence:

$val(out(Id)) : +\mathsf{norm} \leftarrow val(in(Id)) : -\epsilon.$

$val(out(Id)) : -\mathsf{norm} \leftarrow val(in(Id)) : +\epsilon.$

An inverter with a short circuit can be modelled as:

$val(out(Id)) : +\mathsf{short} \leftarrow val(in(Id)) : +\epsilon, \quad fault(Id, short) : +\epsilon.$

$val(out(Id)) : -\mathsf{short} \leftarrow val(in(Id)) : -\epsilon, \quad fault(Id, short) : +\epsilon.$

where $\mathsf{norm} <_k \mathsf{short}$. In other words, the positive evidences such as $+\mathsf{norm}$ and $+\mathsf{short}$ (resp. the negative evidences such as $-\mathsf{norm}$ and $-\mathsf{short}$) support the truth value logical 1 (resp. logical 0) of $out(ID)$ with different degrees of support.

Assuming $+\mathsf{max}$ (resp. $-\mathsf{max}$) stands for the largest positive (resp. negative) evidence, if the input node $in(Id)$ is at logical 1 (e.g., $val(in(Id)) : +\mathsf{max}$ holds), we infer that the output is at logical 0 (e.g., $val(out(Id)) : -\mathsf{norm}$ holds). However, if we later learn that the inverter has a short (e.g., $fault(Id, short) : +\mathsf{max}$ holds), then we revise our belief and infer that the output is at logical 1 (e.g., $val(out(Id)) : +\mathsf{short}$ holds). Thus, even though we have conflicting evidences $-\mathsf{norm}$ and $+\mathsf{short}$ supporting $val(out(Id))$, the conflict can be resolved in favor of the conclusion $val(out(Id)) : +\mathsf{short}$.

We now provide a detailed formalization of these ideas along the lines of [17].

# 3 Annotated Logic Programs

## 3.1 Syntax

A *term* is an individual constant or a variable, or an n-ary function symbol $f$ applied to terms $t_1$,$t_2$, ..., $t_n$, that is, $f(t_1, t_2, \ldots, t_n)$. An *atom* is a propositional constant or a formula $q(t_1, \ldots, t_n)$, where $q$ is an n-ary predicate symbol and $t_i$'s are terms. *Literals* are of the form $p : \tau$, where $p$ is an atom and $\tau$ is an *annotation* drawn from the domain of annotations $\mathcal{A}$. A *rule* is an expression of the form $p : \tau \quad \leftarrow \quad q_1 : \gamma_1, \ \ldots, \ q_m : \gamma_m$ where $p$ and $q_i$'s

are atoms and $\tau$ and $\gamma_i$'s are annotations. The literal $p : \tau$ is referred to as the *rule-head*, while $q_1 : \gamma_1,\ \ldots,\ q_m : \gamma_m$ is referred to as the *rule-body*. A *fact* is a *ground* (i.e., variable-free) literal. A *clause* is either a fact or a rule. An *annotated logic program* is a set of clauses.

Following [2, 3, 6, 8, 17], the annotations are ordered along two different dimensions: in one, they are partially ordered by $\leq_k$ on the basis of their *information-content*; in the other, they are related by an equivalence relation $\approx_t$ on the basis of their *truth-content*. (In fact, we require that $\leq_k$ be a *semi-lattice*, that is, a partial order equipped with a least-upper-bound operation.) We define:

- $(\tau\ <_k\ \gamma)$ if and only if $(\tau\ \leq_k\ \gamma)\ \wedge\ (\tau\ \neq_k\ \gamma)$; and
- $(\tau\ \leq_{tk}\ \gamma)$ if and only if $(\tau\ \leq_k\ \gamma)\ \wedge\ (\tau\ \approx_t\ \gamma)$.

Furthermore, for our purposes, we assume that $\approx_t$ has only the following three equivalence classes: $\mathcal{C}^-$, $\mathcal{C}^*$, and $\mathcal{C}^+$, corresponding to the defeating, ambiguous, and supporting evidences, respectively. We also assert the existence of the smallest element, denoted $\epsilon$, and the largest element, denoted $\omega$, in each of the three classes.

We impose certain acyclicity restrictions on the annotated logic programs for reasons that will become clear in the next section. This acyclicity requirement is a generalization of the *local stratification* condition for ordinary logic programs [15]. We formalize this requirement by defining a relation $\ll$ on the ground atoms as follows: We say that $q \ll_{\mathbf{P}} p$ holds if there is a ground instance $p : \alpha \leftarrow q_1 : \gamma_1, \ldots, q : \beta, \ldots, q_n : \gamma_n$ of a rule in the program $\mathbf{P}$, *or* there is a ground instance $p : \alpha \leftarrow q_1 : \gamma_1, \ldots, r : \tau, \ldots, q_n : \gamma_n$ of a rule in $\mathbf{P}$ and $q \ll_{\mathbf{P}} r$. We demand that, for annotated logic programs of interest, the relation $\ll_{\mathbf{P}}$ be *acyclic*. In the sequel, we refer to annotated logic programs that satisfy this condition as *stratified annotated logic programs*, and abbreviate them as *SALPs*.

## 3.2 Model-theoretic Semantics

Let $\mathbf{P}$ be an annotated logic program. The *domain* $\mathcal{D}$ of interpretations of $\mathbf{P}$ is a collection of all individual constants mentioned in $\mathbf{P}$, and the ground (variable-free) terms that can be formed from them using function symbols; it is often called the *universe* of $\mathbf{P}$. A *base*, $\mathcal{B}_{\mathbf{P}}$, of $\mathbf{P}$ is a collection of all ground atoms $p$ that use only terms of $\mathcal{D}$ and the predicate symbols mentioned in $\mathbf{P}$.

An *interpretation* $\mathcal{I}$ of $\mathbf{P}$ is a *partial* mapping from the base, $\mathcal{B}_{\mathbf{P}}$, of $\mathbf{P}$ to the set of annotations $\mathcal{A}$. Given two interpretations $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{J} \sqsubseteq_k \mathcal{I}$ (resp., $\mathcal{J} \sqsubseteq_{tk} \mathcal{I}$) if and only if for every atom $p \in \mathcal{B}_{\mathbf{P}}$, such that $\mathcal{J}(p)$ is defined, $\mathcal{I}(p)$ is also defined and $\mathcal{J}(p) \leq_k \mathcal{I}(p)$ (resp., $\mathcal{J}(p) \leq_{tk} \mathcal{I}(p)$). An interpretation $\mathcal{I}$ is *minimal* in a set of interpretations $\mathcal{S}$ if and only if there is no $\mathcal{J} \in \mathcal{S}$ such that $\mathcal{J} \sqsubseteq_k \mathcal{I}$ and $\mathcal{J} \neq \mathcal{I}$; $\mathcal{I}$ is *maximal* if and only if there is no $\mathcal{J} \in \mathcal{S}$ such that $\mathcal{I} \sqsubseteq_k \mathcal{J}$ and $\mathcal{J} \neq \mathcal{I}$.

A ground literal (fact) $p : \tau$ is *satisfied* in $\mathcal{I}$, denoted $\mathcal{I} \models_k p : \tau$, if and only if $\mathcal{I}(p)$ is defined and $\tau \leq_k \mathcal{I}(p)$; it is *strongly satisfied* in $\mathcal{I}$, denoted $\mathcal{I} \models_{tk} p : \tau$, if also $\tau \leq_{tk} \mathcal{I}(p)$.

A conjunction of ground literals $q_1 : \gamma_1, q_2 : \gamma_2, \ldots, q_n : \gamma_n$ (denoted as $BODY$) is *satisfied* in $\mathcal{I}$ if and only if ( $\forall i : 1 \leq i \leq n : \Rightarrow \mathcal{I} \models_k q_i : \gamma_i$ ); it is *strongly satisfied* if and only if ( $\forall i : 1 \leq i \leq n : \Rightarrow \mathcal{I} \models_{tk} q_i : \gamma_i$ ).

A ground rule $HEAD \leftarrow BODY$ is *satisfied* in $\mathcal{I}$ if and only if $BODY$ is strongly satisfied implies $HEAD$ is satisfied. That is, a ground rule $p : \tau \leftarrow q_1 : \gamma_1, \ldots, q_n : \gamma_n$ is *satisfied* in $\mathcal{I}$ if and only if ( $\forall i : 1 \leq i \leq n : \Rightarrow \mathcal{I} \models_{tk} q_i : \gamma_i$ ) implies $\mathcal{I} \models_k p : \tau$. Note that this is consistent with thinking of facts as if they were rules with the empty body, which is always strongly satisfied in all interpretations. A non-ground rule $HEAD \leftarrow BODY$ is *satisfied* in $\mathcal{I}$ if and only if all its ground instances are satisfied in $\mathcal{I}$.

An annotated logic program $\mathbf{P}$ is *satisfied* in $\mathcal{I}$, if all its clauses are satisfied in $\mathcal{I}$. An interpretation $\mathcal{I}$ is a *model* of a set of clauses $\mathbf{P}$ if and only if $\mathbf{P}$ is satisfied under $\mathcal{I}$. A model $\mathcal{I}$ of $\mathbf{P}$ is *supported* by $\mathbf{P}$ if for every atom

$p$ such that $\mathcal{I}(p)$ is defined, we have

$$\mathcal{I}(p) \;=\; lub_k\{\tau \;\mid\; p:\tau \;\leftarrow\; BODY \text{ is a ground instance of a clause in } \mathbf{P} \text{ and } \mathcal{I} \;\models_{tk}\; BODY\}.$$

We assume that $\mathcal{I}$ is undefined on any atom that does not appear in the head of a ground instance of a rule in $\mathbf{P}$. Informally, $\mathcal{I}$ is supported by $\mathbf{P}$ if the evidences that it assigns to atoms are not stronger than what is warranted by $\mathbf{P}$. (Recall also that a fact can be thought of as a rule with the empty body.)

In general, there are annotated logic programs that admit zero or more supported models. This is in contrast with the definite clauses which admit a unique minimal supported model. The reason for the first discrepancy is the fact that general annotated logic programs can be recursive and the addition of new facts can cause the truth values of the conclusions to change nonmonotonically. For instance, consider the following program:

$p:-1 \leftarrow q:+1.$

$q:+1 \leftarrow p:+1.$

$p:+1.$

The minimal model of this program is: $\{p:*1,\; q:+1\}$, given that $lub_k(-1,+1) \;=\; *1$. This model is not supported because $q:+1$ is unsupported. The latter condition arises because $p:+1$ is not strongly satisfied in the model. On the other hand, the reason for the existence of multiple supported models can be traced to mutual recursion. For instance, the program

$p:\alpha \leftarrow q:\beta.$

$q:\beta \leftarrow p:\alpha.$

has two supported models: $\emptyset$ and $\{p:\alpha,\; q:\beta\}$. We can filter out the second model containing cyclic "ungrounded" support by insisting on the minimality condition. However this is not sufficient because stratified programs can have

several minimal supported models in the presence of recursion. For example, the following program:

$r : +1.$

$r : -2 \leftarrow p : -1.$

$p : +2 \leftarrow p : +2.$

$p : -1.$

admits two minimal supported models: $\{\{p : -1,\ r : -2\}$,$\{p : +2,\ r : +1\}\}$.

However, when we restrict our attention to SALPs, we can prove the following results by adapting techniques in [1], as described in [17].

**Lemma 3.1** *Every annotated logic program containing only facts has a unique supported model, if* $\leq_k$ *is a semilattice.*

**Theorem 3.1** *Every stratified annotated logic program has a unique supported model, if* $\leq_k$ *is a semilattice.*

**Theorem 3.2** *The supported model of an annotated logic program is always minimal.*

**Proof:** The supported model $\mathcal{M}$ is *minimal* because any interpretation $\mathcal{I}$ such that $(\mathcal{I} \sqsubseteq_k \mathcal{M} \ \wedge \ \mathcal{I} \neq \mathcal{M})$ holds cannot be a *model*. •

Another characteristic property of the SALPs is their *stability*.

**Theorem 3.3** *Let* $\mathbf{P}$ *be a stratified annotated logic program and* $r$ *be a clause. Let* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *be the unique supported models of* $\mathbf{P}$ *and* $\mathbf{P} \cup \{r\}$ *respectively. Then,* $\mathcal{M}_1 \models_{tk} r$ *if and only if* $\mathcal{M}_1 = \mathcal{M}_2$.

This important property does not hold for a number of popular non-monotonic theories based on first-order logic language such as [13, 16] because the language does not *explicitly* distinguish between strict and defeasible conclusions. In particular, a defeasible conclusion in the theories of [13, 16] is incorrectly accorded the status of a strict fact when it is added to the original theory.

## 3.3 A Meta-Interpreter in Prolog

In this section we present a meta-interpreter written in Prolog to answer queries with respect to annotated logic programs.

The Prolog encoding of the inverter example is essentially the same as the one described in Section 2 except that the constant $\epsilon$ has been replaced by an anonymous variable. We have also added two new facts to illustrate the handling of ambiguity.

```
:- op(1150, xfx, [ <- ]).
val(out(Id)) : +norm    <-   val(in(Id)) : -_.
val(out(Id)) : -norm    <-   val(in(Id)) : +_.
val(out(Id)) : +short   <-   val(in(Id)) : +_ , fault(Id,short) : +_.
val(out(Id)) : -short   <-   val(in(Id)) : -_ , fault(Id,short) : +_.
val(in(inv1)) : +max    <-   true.
val(in(inv0)) : +max    <-   true.
fault(inv0,short) : +max  <-  true.
val(node1) : +vdd  <-  true.
val(node1) : -gnd  <-  true.
```

The ordering and the equivalence of the strength of evidences on the information content scale (that is, the relations $<_k$ and $=_k$) can be represented in Prolog using the operators "`:<<`" (defined in terms of "`:<`") and "`:==`" (defined in terms of "`:=`") respectively as follows:

```
:- op(800, xfx, [ :<, :<< ]).
:- op(800, xfx, [ :=, :== ]).
norm  :< short.
short :< max.
vdd := gnd.
```

The operator "`:<<`" is the transitive closure of "`:<`", and the operator "`:==`" is the equivalence relation covering "`:=`".

```
M1 :<< M2 :- M1 :< M2.
M1 :<< M2 :- M1 :< M3, M3 :<< M2.
M1 :== M2 :- M1 == M2.
M1 :== M2 :- M1 := M2.
M1 :== M2 :- M2 := M1.
M1 :== M2 :- M1 := M3, M3 :== M2.
```

To incorporate numbers, add:

```
M1 :<< M2 :- number(M1), number(M2), M1 < M2.
```

The meta-interpreter processes queries of the form "?(*Atom : Evid*)". It first computes the cumulative evidence supporting *Atom*. If the variable *Evid* in the query is *bound*, then it checks to see whether the cumulative evidence strongly satisfies the value of *Evid*; otherwise, it returns the computed cumulative evidence in *Evid*.

```
:- op(800, fy, [ ? ]).
?( true ).
?( (G1,G2) )    :- ?( G1 ), ?( G2 ).
?( Atom : Evid ) :-  findall( E, ((Atom:E <- Body),?(Body)), EL),
                     lub( EL, Cum_Evid),
                     (var(Evid) -> Evid = Cum_Evid ;
                                   sign_mag(Evid,S,M),
                                   sign_mag(Cum_Evid,S,CM),
                                   (var(M) -> M = CM ;
                                              (M :<< CM ; M :== CM))).
```

The evidences are combined by computing their least upper bound with respect to $\leq_k$, by examining their sign and magnitude. If the magnitude of an evidence is clearly higher than that of the other, then the former dominates. If the two evidences have the same magnitude, but different signs, then there is a conflict/ambiguity/inconsistency.

```
:- op(500, fy, [ * ]).

lub( [E], E ).
lub( [E1,E2|EL], E ) :- combine(E1,E2,ET), lub( [ET|EL], E).

sign_mag(+M, +, M).
sign_mag(-M, -, M).
sign_mag(*M, *, M).

combine(E1,E2,E) :- sign_mag(E1,S1,M1), sign_mag(E2,S2,M2),
                    ( M1 :<< M2 -> E = E2 ;
                         ( M2 :<< M1 -> E = E1 ;
                              ( M1 :== M2 ->
                                   ( S1 = S2 -> E = E1 ;
                                          sign_mag(E,*,M1) )))).
```

The following query-response pairs illustrate the working of the meta-interpreter.

```
| ?- ?(val(out(inv1)) : V).
V = -norm
| ?- ?(val(out(inv0)) : V).
V = +short
| ?- ?(val(node1) : V).
V = *vdd
| ?- ?(val(out(inv0)) : +M).
M = short
| ?- ?(val(out(inv0)) : *M).
no
| ?- ?(val(out(inv0)) : -norm).
no
| ?- ?(val(out(inv0)) : +norm).
```

```
yes
```

## 3.4 Relationship to Nonmonotonic Formalisms

We now explore equivalence results between annotated logics and other nonmonotonic formalisms.

### 3.4.1 Belnap's four-valued logic

We first map the truth values in the four-valued logic into equivalent annotations and then study the relationship between the logical connectives used in the two formalisms.

We restrict our attention to the semilattice shown in Figure 1. The values $-\omega$, $+\omega$, and $*\omega$ in the semilattice correspond to the values $\mathsf{f}$, $\mathsf{t}$ and $\top$ respectively in Belnap's bilattice. The undefinedness in our approach is captured by $\bot$ in the bilattice. Consequently, we can associate a unique interpretation $\Theta(\mathcal{B})$ with every four-valued (Belnap) interpretation $\mathcal{B}$ as follows:

$\mathcal{B}(p) = \mathsf{t}$ iff $\Theta(\mathcal{B})(p) = +\omega$
$\mathcal{B}(p) = \mathsf{f}$ iff $\Theta(\mathcal{B})(p) = -\omega$
$\mathcal{B}(p) = \top$ iff $\Theta(\mathcal{B})(p) = *\omega$
$\mathcal{B}(p) = \bot$ iff $\Theta(\mathcal{B})(p)$ is undefined

However, the meaning of the connectives "," and "$\leftarrow$" in the rule

$$p : \alpha \leftarrow q_1 : \gamma_1, \ldots, q : \beta, \ldots, q_n : \gamma_n$$

is very different from the meaning associated with the conjunction "&" and the implication "$\Rightarrow$" respectively of Belnap's logic shown in Figure 2.

Recall that the notion of (strong) satisfaction of an annotated literal in an interpretation is two-valued. Thus, the connective ",", used in the annotated logic, is the classical (two-valued) "and", while the connective "&", used in Belnap's logic, is a conservative extension of "and" to all four values

of the bilattice. Similarly, the annotated logic rule with the connective "$\leftarrow$" resembles a classical logic inference rule, while "$\Rightarrow$", which represents entailment in Belnap's logic, resembles a classical logic implication. In particular, $p : +\omega \leftarrow q : -\omega$ is not equivalent to $q : +\omega \leftarrow p : -\omega$, while $p \Rightarrow q$ is equivalent to $\neg q \Rightarrow \neg p$.

We now explore equivalence results for the two theories.

**Definition 3.1** *An* $\omega$-program *(resp.* $\omega$-literal*) is a SALP (resp. an annotated literal) whose annotations are restricted to* $\{-\omega, +\omega\}$.

An $\omega$-program can be translated into Belnap's logic as follows:

$\Gamma(p : +\omega) \equiv p$

$\Gamma(p : -\omega) \equiv \neg p$

$\Gamma(p : \alpha \leftarrow q_1 : \gamma_1, \ldots, q_n : \gamma_n) \equiv \Gamma(q_1 : \gamma_1) \& \ldots \& \Gamma(q_n : \gamma_n) \Rightarrow \Gamma(p : \alpha)$

The following result is an immediate consequence of the definitions of $\Gamma$, $\Theta$, minimality with respect to information lattice [2, 3], and Theorem 3.1.

**Theorem 3.4** *Let* $\mathbf{P}$ *be an* $\omega$*-program that contains* only facts. *Then, a Belnap interpretation* $\mathcal{B}$ *is the minimum model of* $\Gamma(\mathbf{P})$ *if and only if the interpretation* $\Theta(\mathcal{B})$ *is the supported model of* $\mathbf{P}$.

**Theorem 3.5** *Let* $\mathbf{P}$ *be an* $\omega$*-program that is* negation-free*, (that is, it does not contain any negative evidence annotation). Then, a Belnap interpretation* $\mathcal{B}$ *is the minimum model of* $\Gamma(\mathbf{P})$ *if and only if the interpretation* $\Theta(\mathcal{B})$ *is the supported model of* $\mathbf{P}$.

**Proof:** The result follows by a simple induction on $\ll$-relation once we observe that (1) $\mathbf{P}$ is not recursive, and (2) In the absence of negation in $\Gamma(\mathbf{P})$, no new negative literals are entailed by contraposition of rules. •

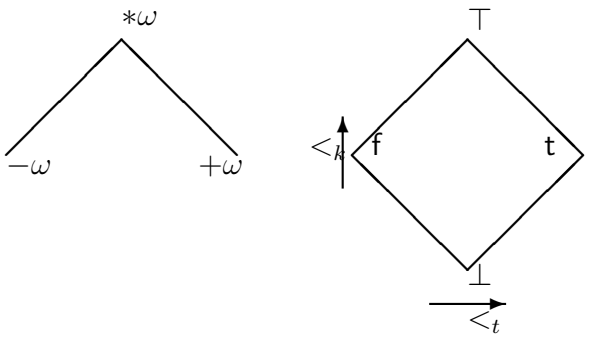


Figure 1: Semilattice and Belnap's Bilattice

| & | $\bot$ | f | t | $\top$ |
|---|---|---|---|---|
| $\bot$ | $\bot$ | f | $\bot$ | f |
| f | f | f | f | f |
| t | $\bot$ | f | t | $\top$ |
| $\top$ | f | f | $\top$ | $\top$ |

| $\Rightarrow$ | $\bot$ | f | t | $\top$ |
|---|---|---|---|---|
| $\bot$ | valid | invalid | valid | valid |
| f | valid | valid | valid | valid |
| t | invalid | invalid | valid | invalid |
| $\top$ | valid | invalid | valid | valid |

Figure 2: Truth Tables for Belnap's logic

However, when programs contain both negation (that is, negative annotation) and entailment, the equivalence results no longer hold. For instance, consider the annotated program $\mathbf{P} = \{p : -\omega \leftarrow q : +\omega,\ p : +\omega,\ q : +\omega\}$. The unique supported model $\mathcal{M}$ of $\mathbf{P}$, according to our semantics, maps $p$ to $*\omega$ and $q$ to $+\omega$. The translation of $\mathbf{P}$ into Belnap's logic, $\Gamma(\mathbf{P})$, is $\{q \Rightarrow \neg p,\ p,\ q\}$. The minimum model $\mathcal{B}$ of $\Gamma(\mathbf{P})$, according to Belnap's logic, maps both $p$ and $q$ to $\top$. Thus, the supported model $\mathcal{M}$ and the image under $\Theta$ of Belnap's minimum model $\mathcal{B}$, agree on $p$, but differ on $q$.

### 3.4.2 Logic Programs with Classical Negation

We now investigate relationship to the semantics of logic programs that support both classical negation (denoted $\neg$) and negation-by-failure (denoted *not*) as developed by Gelfond and Lifschitz [7]. (The two negations differ in that the meaning associated with the program $\{p \leftarrow\ not\ q\}$ is the set {p}, while that of $\{p \leftarrow \neg q\}$ is the set {}.)

The two theories differ as follows: In our theory there is no counterpart for the *not*-operator. Secondly, even though recursion is permitted in the language, a ground instance of a SALP is recursion-free. This is in contrast with [7] which permits the ground instances to be recursive. On the other hand, our theory enables a satisfactory representation of a certain class of common-sense facts, because annotations can be used to make explicit meta-information (such as specificity) required to resolve conflicts among contradictory conclusions.

To better understand the relationship between the two theories, we study the class of $\omega$-programs. These programs can be naturally viewed as logic programs with classical negation according to the following translation:

$\Gamma(p : +\omega) \equiv p$

$\Gamma(p : -\omega) \equiv \neg p$

$\Gamma(p : \alpha \leftarrow q_1 : \gamma_1, \ldots, q_n : \gamma_n) \equiv \Gamma(p : \alpha) \leftarrow \Gamma(q_1 : \gamma_1), \ldots, \Gamma(q_n : \gamma_n)$

Recall that the annotations $\{-\omega, +\omega, *\omega\}$ are the maximal elements with

respect to $\leq_{tk}$-ordering.

**Definition 3.2** *Let* $\mathbf{P}$ *be an* $\omega$*-program. An interpretation* $\mathcal{I}$ *of* $\mathbf{P}$ *is* $\omega$-consistent *if it does not satisfy any literal of the form* $p : *\omega$.

**Definition 3.3** *An* $\omega$*-program* $\mathbf{P}$ *is said to be* consistent *if the supported model is* $\omega$*-consistent.*

We wish to show here that the two theories "agree on" the class of consistent $\omega$-programs. To move towards this goal, we review the notion of answer sets given in [7]. A word about the usage of the term "literal". We reserve "literal" to stand for a literal in ordinary logic programs, and use "$\omega$-literal" to refer to an annotated literal in $\omega$-programs.

**Definition 3.4** *Let* $\mathbf{Q}$ *be a logic program that does not contain* not, *and* Lit *be the set of all ground literals of* $\mathbf{Q}$. *Then, the* answer set $\alpha_{\mathbf{Q}}$ *of* $\mathbf{Q}$ *is the* smallest *subset of* Lit *such that*

*(1) for any ground instance* $l_0 \leftarrow l_1, \ldots, l_m$ *of a rule in* $\mathbf{Q}$,
*if* $\{l_1, \ldots, l_m\} \in \alpha_{\mathbf{Q}}$, *then* $l_0 \in \alpha_{\mathbf{Q}}$.

*(2) if* $\alpha_{\mathbf{Q}}$ *contains a pair of complementary literals, then* $\alpha_{\mathbf{Q}}$ *is* Lit.

**Lemma 3.2** *Let* $\mathbf{Q}$ *be a* consistent *logic program that does not contain* not. *Then, the answer set* $\alpha_{\mathbf{Q}}$ *is consistent and unique.*

**Proof:** Follows from Proposition 2 in [7] and the fact that definite clauses have unique answer sets. •

In order to relate answer sets to interpretations, we make the following definition.

**Definition 3.5** *Let* S *be a* consistent *set of ground literals. Then, the mapping* $\Phi(S)$ *is defined as:* $\Phi(S)(p) = +\omega$ iff $p \in S$ *and* $\Phi(S)(p) = -\omega$ iff $\neg p \in S$

**Lemma 3.3** *The mapping $\Phi(S)$ above is an $\omega$-consistent interpretation.*

**Proof:** As $S$ is consistent, $\Phi(S)$ is a partial function. •

**Lemma 3.4** *Let $p : \alpha$ be an $\omega$-literal, and $S$ be a* consistent *set of ground literals. Then,*

*(1)* $\Phi(S) \models_{tk} p : \alpha$ iff $\Gamma(p : \alpha) \in S$.

*(2)* $\Phi(S) \models_{tk} p : \alpha$ iff $\Phi(S) \models_{k} p : \alpha$.

**Proof:** (1) Follows from the definitions of $\Gamma$, $\Phi$, and $S$ is consistent.

(2) ($\Rightarrow$) Trivial. ($\Leftarrow$) Since $S$ is consistent, there does not exist an atom $p$ such that $p \in S \wedge \neg p \in S$. Given that the annotations $\{-\omega, +\omega\}$ are the *maximal* elements with respect to $\leq_k$-ordering, the result follows. •

**Lemma 3.5** *Let $\mathbf{P}$ be an $\omega$-program. A consistent subset $S$ is closed under a ground instance $\Gamma(p : \alpha) \leftarrow \Gamma(q_1 : \gamma_1), \ldots, \Gamma(q_n : \gamma_n)$ of a rule in $\Gamma(\mathbf{P})$ if and only if the interpretation $\Phi(S)$ of $\mathbf{P}$ satisfies the ground instance $p : \alpha \leftarrow q_1 : \gamma_1, \ldots, q_n : \gamma_n$ of the corresponding rule in $\mathbf{P}$.*

**Proof:** Follows from the definitions of $\Gamma$, $\Phi$ and Lemma 3.4. •

**Theorem 3.6** *Let $\mathbf{P}$ be a consistent $\omega$-program, $\mathcal{M}_{\mathbf{P}}$ be the supported model of $\mathbf{P}$, and $\alpha_{\mathbf{P}}$ be the answer set associated with $\Gamma(\mathbf{P})$. Then, $\mathcal{M}_{\mathbf{P}} = \alpha_{\mathbf{P}}$.*

**Proof:** From Definition 3.4 it is clear that, for a consistent logic program, the answer set is the smallest set of ground literals closed under ground instances of the program clauses. Similarly, the supported model is the minimum interpretation that satisfies all the ground instances of the $\omega$-program clauses from Theorem 3.2. So the result follows from Lemma 3.5. •

However the two semantics differ in their interpretation of $\omega$-programs that are not consistent. For instance, the program $\mathbf{P} = \{p : +\omega,\ p : -\omega,\ q : +\omega\}$ has a unique supported model $\mathcal{M}_{\mathbf{P}} = \{p : *\omega,\ q : +\omega\}$ according to our semantics, while [7] associates with the translation $\Gamma(\mathbf{P}) = \{p,\ \neg p,\ q\}$ the

answer set $\alpha_{\mathbf{P}} = \{p, \neg p, q, \neg q\}$. In other words, the effect of inconsistent information about $p$ is "localized" in our approach, whereas it has "global" repurcussions in the approach of [7].

### 3.4.3 Relationship to Default Logics

We investigate relationship to Brewka's *prioritized default logic* (PDL) [4], which is a generalization of Reiter's default logic [16].

A *PDL-spec* is an $n$+1-tuple $\mathbf{T} = (D_1, D_2, \ldots, D_n, W)$, where $W$ is a set of classical first-order logic formulas, and each $D_i$ is a set of (Reiter's) defaults [4]. Furthermore, for $i > j$, the defaults in $D_i$ have priority over the defaults in $D_j$ in the event of a "conflict". This can be formally expressed by generalizing Reiter's notion of extension as follows [4]: $\mathcal{E}$ is a *PDL-extension* of $\mathbf{T}$ iff there exists a sequence of sets of formulas $E_1, \ldots, E_n$ such that

$E_n$ is an extension of $(D_n, W)$,
$E_{n-1}$ is an extension of $(D_{n-1}, E_n)$,
$E_{n-2}$ is an extension of $(D_{n-2}, E_{n-1})$,
$\ldots$,
$E_1$ is an extension of $(D_1, E_2)$, and $\mathcal{E} = E_1$.

We identify special subsets of SALPs and PDL-specs that can be regarded as equivalent. Toward this goal we make the following definitions.

**Definition 3.6** *A (Reiter) default rule* $\frac{P:\mathbf{M}A}{C}$ *(where the consequent C, the pre-requisite P and the assumption A are first-order formulas), is called a* normal default *if* $A \equiv C$*; it is called a* Horn default *if C is a (FOL) literal.*

**Definition 3.7** *A PDL-spec* $\mathbf{T} = (D_1, D_2, \ldots, D_n, W)$ *is called a* restricted PDL-spec *if*

1. (consistency) $W$ *is a* consistent *set of (FOL)* literals.
2. (normal and Horn defaults) *The default rules in each* $D_i$ *are both* normal *and* Horn.

3. (stratification) *The ground instances of literals in the pre-requisite $P$ of a default rule $\frac{P:\mathbf{M}C}{C}$ in $D_i$ appear as consequents of defaults* only *in* $\bigcup_{k=i+1}^{n} D_k$.

Now we identify special subsets of SALPs that can be shown to be equivalent to restricted PDL-specs. Consider the set of annotations ($\{-,+,*\} \times \{1,2,\ldots,\omega\}$) with the following semilattice ordering defined on it: (Here, the annotation $(+, 2)$ is denoted as $+2$. The former is referred to as the sign and the latter as the magnitude of the annotation.)
For annotations $\alpha$, $\beta$:

$\alpha <_k \beta$ **iff** $[mag(\alpha) < mag(\beta)] \vee$
$[(mag(\alpha) = mag(\beta)) \wedge (sign(\alpha) \in \{-,+\}) \wedge (sign(\beta) = *)]$

$\alpha <_{tk} \beta$ **iff** $(mag(\alpha) < mag(\beta)) \wedge (sign(\alpha) = sign(\beta))$

This semilattice is depicted in Figure 3, and is referred to as the *linear semilattice*.

**Definition 3.8** *A SALP is called* linear *if the annotations belong to the linear semilattice.*

One can show that there is a natural correspondence between restricted PDL-specs that admit unique PDL-extensions and linear SALPs that are consistent. More interestingly, it is possible to define the notion of a "skeptical" extension by taking the intersection of all extensions strata by strata, and prove a stronger equivalence result between restricted PDL-specs and linear SALPs.

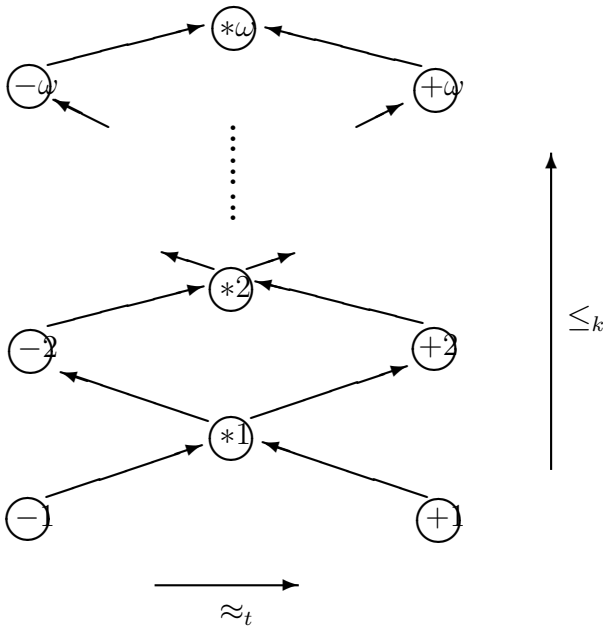


Figure 3: Linear Semilattice

$\mathcal{F}$ is a *skeptical PDL-extension* of a restricted PDL-spec $\mathbf{T}$ iff there exists a sequence of sets of literals $F_1, \ldots, F_n$ such that

$F_n$ is the intersection of all extensions of $(D_n, W)$,

$F_{n-1}$ is the intersection of all extensions of $(D_{n-1}, F_n)$,

$\ldots$,

$F_1$ is the intersection of all extensions of $(D_1, F_2)$, and $\mathcal{F} = F_1$.

In what follows, an extension of a restricted PDL-spec is represented as a set of (FOL) literals.

We now present a translation of restricted PDL-specs to SALPs.

**Definition 3.9** *Let $\mathbf{T} = (D_1, D_2, \ldots, D_n, W)$ be a restricted PDL-spec. The translation $\Psi$ of $\mathbf{T}$ into an annotated logic program, denoted $\Psi(\mathbf{T})$, is as follows:*

*For every positive fact in $W$:* $\Psi(\ p\ ) = p : +\omega$.

*For every negative fact in $W$:* $\Psi(\neg p) = p : -\omega$.

*For every default rule in $D_i$:*

$\Psi(\frac{q_1, \ldots, q_u, \neg r_1, \ldots, \neg r_v : \mathbf{M}\, p}{p}) = p : +i \leftarrow q_1 : +1, \ldots, q_u : +1, r_1 : -1, \ldots, r_v : -1.$

$\Psi(\frac{q_1, \ldots, q_u, \neg r_1, \ldots, \neg r_v : \mathbf{M} \neg p}{\neg p}) = p : -i \leftarrow q_1 : +1, \ldots, q_u : +1, r_1 : -1, \ldots, r_v : -1.$

**Lemma 3.6** *Let $\mathbf{T} = (D_1, D_2, \ldots, D_n, W)$ be a restricted PDL-spec. Then, the translation $\Psi(\mathbf{T})$ is a* linear *SALP.*

**Proof:** The result follows trivially once we note that the set of ground instances of the defaults in $\mathbf{T}$ is *stratified*. •

We now relate a supported model to a PDL-extension. We will explicitly distinguish an FOL interpretation from an AL (annotated logic) interpretation.

**Definition 3.10** *Let $\mathcal{I}$ be an AL-interpretation. Then, the set of ground FOL-literals $\Upsilon(\mathcal{I})$ associated with $\mathcal{I}$ is defined as:*

$\mathcal{I} \models_{tk} p : +1$ iff $\quad p \in \Upsilon(\mathcal{I})$

$\mathcal{I} \models_{tk} p : -1$ iff $\quad \neg p \in \Upsilon(\mathcal{I})$

**Lemma 3.7** *Let $\mathcal{I}$ be an AL-interpretation. Then, the set of ground FOL-literals $\Upsilon(\mathcal{I})$ is consistent.*

**Proof:** Observe that $\{p : *n\} \not\models_{tk} p : +1$ and $\{p : *n\} \not\models_{tk} p : -1$, for any $n \geq 1$. So, the literals in $\mathcal{I}$ with "inconsistent" annotations do not contribute to $\Upsilon(\mathcal{I})$. •

**Lemma 3.8** *Let* **P** *be a restricted PDL-spec containing* only literals*, and* $\Psi(\mathbf{P})$ *be the corresponding linear SALP. Then, the set of FOL-literals* $\Upsilon(\mathcal{I})$ *is the skeptical extension of* **P** if and only if *the AL-interpretation* $\mathcal{I}$ *is the supported model of* $\Psi(\mathbf{P})$*.*

**Proof:** As **P** is consistent and contains FOL-literals alone, it has a unique extension. So the claim follows trivially from the definitons of $\Psi$ and $\Upsilon$. •

**Theorem 3.7** *Let* **P** *be a restricted PDL-spec, and* $\Psi(\mathbf{P})$ *be the corresponding linear SALP. If the AL-interpretation* $\mathcal{I}$ *is the supported model of* $\Psi(\mathbf{P})$*, then the set of FOL-literals* $\Upsilon(\mathcal{I})$ *is the skeptical extension of* **P***.*

**Proof:** Given that $\mathcal{I}$ is the supported model of $\Psi(\mathbf{P})$, $\exists n : \mathcal{I} \models_{tk} p : +n$ iff $p \in \Upsilon(\mathcal{I})$, and similarly, $\exists n : \mathcal{I} \models_{tk} p : -n$ iff $\neg p \in \Upsilon(\mathcal{I})$. To show that $\Upsilon(\mathcal{I})$ is the skeptical extension of **P**, we show that there exists a derivation of $p$ (resp. $\neg p$) in the restricted PDL-spec **P** for each $p$ (resp. $\neg p$) in $\Upsilon(\mathcal{I})$.

This can be proved by induction on the number of groups of defaults with different priorities (which is also the length of the derivation of $p$ (resp. $\neg p$)) in PDL-spec **P**. For the basis case, we have Lemma 3.8. For the induction step, we assume the result holds for the restricted PDL-spec $(D_{i+1}, \ldots, D_n, W)$ and show that it also holds for $(D_i, \ldots, D_n, W)$. Given the translation $\Psi$, it can be easily verified that if $\mathcal{I} \models_{tk} p : +i$ (resp. $\mathcal{I} \models_{tk} p : -i$) then $p$ (resp. $\neg p$) is derivable in $(D_i, \ldots, D_n, W)$ using a default in $D_i$. if $\mathcal{I} \models_{tk} p : *i$ then neither $p$ nor $\neg p$ belong to $\Upsilon(\mathcal{I})$. •

The converse, however, is not true as illustrated below: Consider the PDL-spec $\{ \{\frac{p:\mathbf{M}r}{r}\}\{\frac{q:\mathbf{M}p}{p}, \frac{q:\mathbf{M}\neg p}{\neg p}\}, \{\mathrm{q}\} \}$ and the corresponding translation

$\{r : +1 \leftarrow p : +1,\ p : +2 \leftarrow q : +1,\ p : -2 \leftarrow q : +1,\ q : +\omega\ \}$. $\{q\}$ is the skeptical extension of the PDL-spec, and $\{p : *2, q : +\omega\}$ is the supported model of the SALP. Furthermore, $\Upsilon(\{p : *2, q : +\omega\}) = \{q\}$. However, $\{p : *1, q : +\omega\}$ is not a supported model, even though $\Upsilon(\{p : *1, q : +\omega\}) = \{q\}$.

**Theorem 3.8** *Let* $\mathbf{P}$ *be a restricted PDL-spec, and* $\Psi(\mathbf{P})$ *be the corresponding linear SALP. If* $\Psi(\mathbf{P})$ *is consistent, then AL-interpretation* $\mathcal{I}$ *is a supported model of* $\Psi(\mathbf{P})$ if and only if $\Upsilon(\mathcal{I})$ *is a unique PDL-extension of* $\mathbf{P}$.

**Proof:** Observe that $\Psi(\mathbf{P})$ being consistent implies that there does not exist an atom $p$ for which $\mathcal{I} \models_{tk} p : *n$ holds. Now the result follows by induction on the length of the derivation as given in Theorem 3.7. •

Remarks

The above results are reminiscent of the following correpondence: The PDL-extensions associated with a restricted PDL-specs is in same relationship to the supported model of the corresponding linear SALP, as preferential semantics of negation as failure in logic programs is to well-founded semantics [5], which, in turn is the same as the relationship between a credulous theory of inheritance to the "corresponding" skeptical theory [9].

### 3.4.4 Relationship to other annotated logics

We now sketch the basic differences between the semantics of annotated logics given in [10, 11, 12] from the one prsesented here.

In [10] the annotations are complex *certainty* terms (containing function symbols and variables) whose values are in the range [0,1]. The annotations all represent supporting evidence. The rules with the same head-predicate are assumed to be *independent*. The combining-function, which pools together evidence from different rules, is predicate specific. In contrast, our approach uses constant annotations from a set with upper-semilattice structure. An annotation can represent either supporting or contradicting evidence. The

combining function is the *least-upper bound* operation. Furthermore, in the presence of exceptions, the different rules with identical head-predicates are effectively *mutually exclusive*. This is because, the evidence contributed by a rule to a conclusion may override the evidence contributed by another less specific rule, in case of conflict. The former semantics captures reasoning with uncertainty, while the latter semantics captures defeasible reasoning.

The semantics of annotated logics given in [11, 12] differs from our approach in the interpretation of the $\leftarrow$-operator. In particular, $\leftarrow$-operator differs from both epistemic implication and ontological implication of [11] as illustrated below. Assume that the truth values $\mathtt{t}$ and $\mathtt{f}$ correspond to $+\omega$ and $-\omega$ respectively.

- Consider the program: $\{p : \mathtt{t} \leftarrow q : \mathtt{t}, q : \mathtt{t}\}$. Our approach associates with it the supported model: $\{p : \mathtt{t}, q : \mathtt{t}\}$. The $\leftarrow$-operator differs from the epistemic implication because the latter does not allow us to draw the conclusion $p : \mathtt{t}$, to guard against the possibility that the premise may turn out to be inconsistent.

- Consider the program: $\{p : \mathtt{t} \leftarrow q : \mathtt{t}, q : \top\}$. Our approach associates with it the supported model: $\{q : \top\}$. The $\leftarrow$-operator differs from the ontological implication because the latter allows us to draw the conclusion $p : \mathtt{t}$, in spite of the fact that there is inconsistent information about $q$.

More formally, one can define the "standard" $\mathcal{T}_{\mathbf{P}}$-operator [1] that maps interpretations to interpretations, and study its properties.

**Definition 3.11** *Let* $\mathbf{P}$ *be an annotated logic program and* $\mathcal{I}$ *be an interpretation of* $\mathbf{P}$*. Then,*

$$\mathcal{T}_{\mathbf{P}}(\mathcal{I})(p) \;=\; lub_k\{\tau \;\mid\; p : \tau \;\leftarrow\; BODY \text{ is a ground instance of a clause in } \mathbf{P} \text{ and } \mathcal{I} \models_{tk} BODY\}.$$

According to Lemma 1 of [10] (assuming that certainty functions and combining functions are monotonic) and Theorem 1 of [12],

**Theorem 3.9** $\mathcal{T}_{\mathbf{P}}$ *is monotonic, that is,* $\mathcal{I} \subseteq \mathcal{J} \;\Rightarrow\; \mathcal{T}_{\mathbf{P}}(\mathcal{I}) \subseteq \mathcal{T}_{\mathbf{P}}(\mathcal{J})$.

On the contrary, the $\mathcal{T}_{\mathbf{P}}$-operator according to our semantics is nonmonotonic [17]. In particular,

$$\mathcal{I} \leq_k \mathcal{J} \;\not\Rightarrow\; \mathcal{T}_{\mathbf{P}}(\mathcal{I}) \leq_k \mathcal{T}_{\mathbf{P}}(\mathcal{J}).$$

$$\mathcal{I} \leq_{tk} \mathcal{J} \;\not\Rightarrow\; \mathcal{T}_{\mathbf{P}}(\mathcal{I}) \leq_{tk} \mathcal{T}_{\mathbf{P}}(\mathcal{J}).$$

However it satisfies the following weaker property:

$$\mathcal{I} \leq_{tk} \mathcal{J} \;\Rightarrow\; \mathcal{T}_{\mathbf{P}}(\mathcal{I}) \leq_k \mathcal{T}_{\mathbf{P}}(\mathcal{J}).$$

# 4 Conclusion

In this paper we extended the annotated language of [17] in various directions to obtain an enriched representation language. In particular we permitted rule-bodies to be conjunction of literals, and the rules to be recursive. (However, the literals in our language do not contain variables as annotations (cf. [12]).) We identified a class of annotated logic programs called the *stratified* programs which can be given a *unique supported minimal model* as their meaning. We also described a Prolog interpreter for the language. We then proved certain equivalence results with respect to Belnap's 4-valued logic, Gelfond and Lifschitz's semantics for logic programs with negation, Brewka's prioritized default logics and Kifer *et al*'s annotated logics, and presented simple examples to illustrate situations where the equivalence results break down.

# References


[1] K. R. Apt, H. A. Blair and A. Walker (1987) Towards a theory of declarative knowledge, In Jack Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, (Morgan Kaufman), 89–148.

[2] N. Belnap (1977) How a computer should think. In G. Ryle (ed.), *Contemporary Aspects of Philosophy* (Oriel Press), 30–56.

[3] N. Belnap (1977) A useful four-valued logic, in: J. Dunn and G. Epstein (ed.), *Modern uses of multi-valued logic* (D. Reidel), 8–37.

[4] G. Brewka (1989) Preferred Subtheories: An Extended Logical Framework for Default Reasoning, In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1043–1048.

[5] P. M. Dung (1991) Negation as hypotheses: an abductive foundation for logic programming, In *Proceedings of the Eighth International Conference on Logic Programming*, 3–17.

[6] M. Fitting (1991) Bilattices and the semantics of logic programming, In *Journal of Logic Programming*, **11**:91–116.

[7] M. Gelfond and V. Lifschitz (1990) Logic Programs with Classical Negation, In *Proceedings of the Seventh International Conference on Logic Programming*, 579–597.

[8] M. L. Ginsberg (ed.) (1987) *Readings in Nonmonotonic Reasoning*, Morgan-Kaufmann.

[9] J. Horty (1992) Some Direct Theories of Nonmonotonic Inheritance, In D. Gabbay and C. Hogger (eds.), *Handbook of Logic in Artifical Intelligence and Logic Programming*, (Oxford University Press).

[10] M. Kifer, and A. Li (1988) On the semantics of rule based expert systems with uncertainties, In *Proceedings of the Second International Conference on Database Theory* (Springer Verlag), **326**:102–117.

[11] M. Kifer and E.L. Lozinskii (1992) A logic for reasoning with inconsistency, In *Journal of Automated Reasoning*, **9**(2):179–215.

[12] M. Kifer and V.S. Subrahmanian (1992) Theory of generalized annotated logic programming and its applications, In *Journal of Logic Programming*, **12**(4):335-368.

[13] J. McCarthy (1986) Applications of Circumscription to Formalizing Common-Sense Knowledge, In *Artificial Intelligence*, **28**:89–116.

[14] J. Minker (ed.) (1988) *Foundations of Deductive Databases and Logic Pogramming*, (Morgan Kaufmann).

[15] T.C. Przymusinski (1988) On the declarative semantics of deductive databases and logic programs, In J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, ( Morgan Kaufmann) 193–216.

[16] R. Reiter (1980) A Logic for Default Reasoning, In *Artificial Intelligence*, **13**:81–132.

[17] K. Thirunarayan and M. Kifer (1993) A theory of nonmonotonic inheritance based on annotated logic, In *Artificial Intelligence*, **60**:23-50.

[18] R. Thomason, J. Horty, and D. Touretzky (1987) A calculus for inheritance in monotonic semantic nets, In Z. Ras and M. Zemankova (eds.), *Proceedings of the Second International Symposium on Methodologies for Intelligent Systems* (North Holland), 280–287.