Coherent Well-founded Annotated Logic Programs

Carlos Viegas Damásio[1], Luís Moniz Pereira[2], and Terrance Swift[3]

[1] A.I. Centre, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2825-114 Caparica, Portugal. (cd@di.fct.unl.pt)
[2] A.I. Centre, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2825-114 Caparica, Portugal. (lmp@di.fct.unl.pt)
[3] Department of Computer Science, University of Maryland, College Park, MD, USA. (tswift@cs.umd.edu)

Abstract. Extended logic programs and annotated logic programs are two important extensions of normal logic programs that allow for a more concise and declarative representation of knowledge. Extended logic programs add explicit negation to the default negation of normal programs in order to distinguish what can be shown to be false from what cannot be proven true. Annotated logic programs generalize the set of truth values over which a program is interpreted by explicitly annotating atoms with elements of a new domain of truth values. In this paper coherent well-founded annotated programs are defined, and shown to generalize both consistent and paraconsistent extended programs, along with several classes of annotated programs.

1 Introduction

The ability to concisely represent knowledge by a logic program, along with the ability to efficiently evaluate that program, can lead to important applications of logic programming. This has been seen to be the case in diagnosis, model checking, grammar processing, and many other applications. Indeed, a stream of research has focussed on how logic programming can be employed to better represent knowledge. For instance, extended logic programs add explicit negation to normal programs, and gain the fundamental ability to distinguish what can be shown to be false from what is false by default because it cannot be proven true. This distinction can be useful in representing knowledge that derives from separate, possibly contradictory, sources. These two negations are conveniently related through the principle of *coherence*, which states that an atom that is explicitly proven false must be considered default false as well. In fact, the coherence principle underlies two main semantics for extended programs: the answer set semantics [9] and the well-founded semantics with explicit negation [2]. A separate line of research, into annotated logic programs, has extended the domain of truth values over which logic programs are interpreted. Rather than mapping atoms into *true*, *false*, or *undefined*, they are mapped into domains that allow paraconsistent or quantitative information to be represented. This research direction is represented by formalisms such as GAPs [11] and Amalgamation Logic [13].

Each of these extensions is powerful in itself, but suffers from some deficiencies regarding knowledge representation. Extended logic programs, per se, cannot easily represent quantitative information such as probabilities or degrees of belief; annotated logic programs, per se, cannot easily relate what is explicitly known to be false to what is not known to be true. In this paper we propose a framework for coherent well-founded annotated programs that combines the expressivity of both annotated and extended logic programs. We show that several classes of annotated programs can be embedded into coherent well-founded annotated programs, as can both consistent and paraconsistent extended programs.

2 Generalized Annotated Logic Programs

Generalized annotated logic programs (GAPs) are an extension to ordinary definite logic programs. The language, semantics, and query answering procedures are covered in the joint article by Kifer and Subrahmanian [11]. In this section we recall their fundamental results required by our study.

A GAP is defined with respect to an underlying upper semi-lattice of truth-values $(\mathcal{T}, \preccurlyeq)$, representing a partial ordering among truth-values. This lattice can be used to represent fuzzy truth-values, time intervals, paraconsistent logics, qualitative degrees of truth, and the like [11, 13, 1]. In our work we assume this lattice is complete, and therefore the existence of the minimum and maximum elements is always guaranteed, represented by $\bot$ and $\top$ respectively.

Truth-values are referred to in the programs by means of *annotation terms*, while the basic syntactic elements of generalized annotated logic programs are annotated atoms. Given a set of atoms $\mathcal{A}$, an annotated atom has the form $A : \mu$, where A is an atom in $\mathcal{A}$ and μ is an annotation term. For instance, $low\_temp : 0.82$ may mean that the temperature is low with a certainty of at least 82%, or $box(a) : 4$ could signify that object a is a box with a confidence level of at least 4. Formally:

Definition 1 (Annotation terms). *Let $(\mathcal{T}, \preccurlyeq)$ be a complete lattice, $\mathcal{F}$ a set of function symbols of arity $n \geqslant 1$, and $\mathcal{V}$ a set of variables ranging over the truth-values $\mathcal{T}$. Then,*

1. *Every element of $\mathcal{T}$ is a (simple) annotation term;*
2. *Every annotation variable of $\mathcal{V}$ is a (simple) annotation term;*
3. *If f is a n-ary annotation function symbol of $\mathcal{F}$ and $t_1, \ldots, t_n$ are annotation terms, then $f(t_1, \ldots, t_n)$ is an annotation term;*
4. *Nothing else is an annotation term.*

A annotation function symbol is assumed to be computable and continuous, and hence monotonic [11].

Definition 2 (Generalized Annotated Logic Program). *A generalized annotated logic program is a set of annotated clauses of the form:*

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

where $A_0 : \mu_0$ is an annotated atom, and the $A_i : \mu_i (1 \leq i \leq n)$ are atoms annotated with simple annotation terms. In a ground annotated clause all annotations are truth-values of $\mathcal{T}$.

For our purposes the above syntax suffices. However, in their original paper a full blown first-order logic like syntax is introduced, with the usual connectives and quantification symbols. The details can be found in [11].

The reading of an annotated clause of the form $A_0 : \mu_0 \leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n$ is "if A_1 is at least μ_1 and ... and A_n is at least μ_n then A_0 is at least μ_0." Mark that function symbols may only appear in the heads of annotated clauses. Furthermore, one can instantiate all the annotated variables and evaluate all the function symbol annotations in the heads of the resulting ground program, i.e. where all annotated clauses are replaced by all their ground instances. For simplicity, we assume from now on that this grounding operation has been performed on every program, which may result in an infinite program. This program is dubbed a "strictly ground instance" in [11].

An interpretation is a mapping from the set of atoms to the set of truth-values $\mathcal{T}$. This corresponds to the restricted interpretations of [11]. Given an interpretation, it is straightforward to define a satisfaction relation:

Definition 3 (Ground satisfaction). *Let I be an interpretation on $(\mathcal{T}, \preccurlyeq)$. We define the ground satisfaction relation as follows, denoted by $\models$, where all annotations are ground:*

- $I \models A : \mu$ *iff* $I(A) \succcurlyeq \mu$;
- $I \models A_1 : \mu_1 \& \ldots \& A_n : \mu_n$ *iff* $I \models A_1 : \mu_1$ *and ... and* $I \models A_n : \mu_n$;
- $I \models A_0 : \mu_0 \leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n$ *iff* $I \not\models A_1 : \mu_1 \& \ldots \& A_n : \mu_n$ *or* $I \models A_0 : \mu_0$

An interpretation I is a model of a ground GAP iff it satisfies all the annotated clauses in the program.

The ordering of the underlying lattice of truth-values is easily extended to the point-wise ordering between interpretations. As usual, we are interested in the minimal model of the program. It can be obtained by extending the T_P operator of van Emden and Kowalski [6] to this more general setting:

Definition 4 (Immediate consequences operator). *Let P be a generalized annotated logic program on the complete lattice $(\mathcal{T}, \preccurlyeq)$. The immediate consequences operator, a function mapping interpretations into interpretations, is defined by:*

$$T_P(I)(A) = lub\,\{\mu \mid I \models A_1 : \mu_1 \& \ldots \& A_n : \mu_n, where \\ A : \mu \leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n \text{ belongs to } P\,\}$$

Because this operator is monotonic by the Knaster-Tarski fixpoint theorem, i.e. if $I \preccurlyeq J$ then $T_P(I) \preccurlyeq T_P(J)$, we can conclude that T_P has a least fixpoint, which corresponds to the least model of P. It can be found by iterating from the least interpretation, where all atoms are initially assigned the truth-value $\perp$. However, this operator is not continuous. See [11] for more details.

We now provide some examples to illustrate the above concepts.

Example 1. Definite logic programs are easily captured by generalized annotated logic programs. Let $L_2 = \{\perp, \top\}$ with $\perp \prec \top$. For instance the classical $member/2$ predicate, written as a GAP over L_2 is:

$$member(X, [X|\_]) : \top. \qquad member(X, [\_|Y]) : \top \leftarrow member(X, Y) : \top.$$

In general, we obtain an equivalence between GAPs over L_2 and definite logic programs by adding the annotation ": $\top$" to every predicate symbol of the latter.

Example 2. [10] Consider Belnap's logic $\mathcal{FOUR} = (\{\perp, \mathbf{f}, \mathbf{t}, \top\}, \{\perp \prec \mathbf{f}, \perp \prec \mathbf{t}, \mathbf{f} \prec \top, \mathbf{t} \prec \top\})$. The tweety example can be encoded as:

$$\begin{array}{ll} flies(X) : \mathbf{t} \leftarrow bird(X) : \mathbf{t}. & penguin(fred) : \mathbf{t}. \\ flies(X) : \mathbf{f} \leftarrow penguin(X) : \mathbf{t}. & bird(tweety) : \mathbf{t}. \\ bird(X) : \mathbf{t} \ \leftarrow penguin(X) : \mathbf{t}. & \end{array}$$

In this example we conclude that $flies(tweety) : \mathbf{t}$ and $flies(fred) : \top$. Mark that the corresponding first-order theory does not have any model.

Example 3. By defining $\mathcal{C}$ to be a lattice of probability intervals, GAPs can be used to implement probabilistic reasoning. Specifically, if probabilities associated with atoms are assumed to be independent, the join operation of $\mathcal{C}$ can be defined as the intersection $< max(Low_1, Low_2), min(High_1, High_2) >$ of two intervals $[Low_1, High_1]$ and $[Low_2, High_2]$. Expanding on this idea, GAPs can be used to implement a significant subset of Hybrid Probabilistic Programs [5].

Under the lattice $\mathcal{C}$, GAPs have been used to model probabilistic association rules in a deductive database about aircraft spare parts implemented in XSB [7]. An instance of such a rule is:

$$\begin{array}{l} process(Part,'CADMIUM\ PLATING', Source) : [94.7, 100] \leftarrow \\ \quad nomenclature(Part,'BELL\ CRANK', Source) : [100, 100]\ \& \\ \quad federal\_supply\_class(Part,'A0500', Source)) : [100, 100]. \end{array}$$

This rule allows one to infer the finishing process of a part given other definite true facts about the part that may be present in a database.

3 Coherent Well-founded Annotated Programs

Both common sense and expert knowledge may be positive (stating the veracity of facts and conclusions) or negative (expressing their falsity). It is also important to have the ability to assume truth or falsity of facts non-monotonically. Generalized Annotated Logic Programs provide ease of expression of (monotonic) negative knowledge by means of epistemic negation. Epistemic negation, represented by the symbol "¬", is a unary operator on the truth-value lattice, subject to no other additional constraints. This negation corresponds to the notion of explicit negation in several semantics for extended logic programs.

Example 4. Continuing from Ex. 3, it is known that the primary material of certain types of parts, must be either all Steel, all Aluminum, or all Magnesium. This gives us a rule, using explicit negation, for instance of the form

$$\begin{aligned}&\neg material(Part,'STEEL', Source) : Prob \leftarrow\\ &\quad material(Part,'ALUMINUM', Source) : Prob\ \&\\ &\quad nomenclature(Part,'STRUT', Source) : [100, 100].\end{aligned}$$

The definition of the negation operator is $\neg[Low, High] = [1 - High, 1 - Low]$.

As originally presented [11], the GAP framework lacks a form of (non-monotonic) default negation (called *ontological* negation in [10]), i.e. a non-monotonic closed world assumption. This has been remedied in the more recent work [13], where a well-founded like [8] and answer-sets like semantics [9] have extended GAPs with a default negation operator. However the semantics of [13] ignores a fundamental relationship that default and explicit negation should obey. Namely, that if something is stated false then it should be assumed false: the *coherence* principle[1]. This principle has been extensively advocated in [2–4]. We will adopt coherency from now on and examine its consequences within the setting of annotated programs.

Example 5. Continuing from Ex. 4, it is also known that cadmium plating is only used on steel parts, so that, in the absence of more specific information about a part's material, the part may be inferred to be of generic steel. This requires default negation to be added to GAPs.

$$\begin{aligned}&material(Part,'STEEL', Source) : Prob \leftarrow\\ &\quad process(Part,'CADMIUM\ PLATING', Source) : Prob\ \&\\ &\quad not\ more\_specific\_material(Part,'STEEL', Source) : Prob.\end{aligned}$$

$$\begin{aligned}&more\_specific\_material(Part, Mat, Source) : Prob \leftarrow\\ &\quad material(Part, Mat1, Source) : Prob\ \&\\ &\quad subclass(Mat1, Mat, Source) : [100, 100].\end{aligned}$$

In these rules about parts, GAPs are used to reason with probabilistic data mining rules in Ex. 3, default negation is used to allow default inferences, while explicit negation is used to allow representation of contrary information in the rule of Ex. 4. Moreover, coherence ensures that default literal of the form

$$not\ material(Part,'Steel', Source) : Prob$$

[1] Even though answer-sets are coherent, their paraconsistent [12] and annotated [13] extensions are not. For details consult [4].

are true in the deductive database by virtue of explicit negative information.

Let us start by clarifying the notion of explicit negation.

Definition 5 (Explicit negation). *Let $(\mathcal{T}, \preccurlyeq)$ be a complete lattice. An explicit negation operator "$\neg$" is a total mapping from $\mathcal{T}$ into $\mathcal{T}$ such that the following two conditions are satisfied:*

1. *for every $\mu \in \mathcal{T}$ we have $\neg\neg\mu = \mu$;*
2. *if $\mu \preccurlyeq \vartheta$ then $\neg\mu \preccurlyeq \neg\vartheta$, for every $\mu, \vartheta \in \mathcal{T}$.*

An explicit negation operator enforces a symmetry transformation on the truth-values lattice. The use of negation is already covered by the original syntax of [11]. We extend the syntax with a default negation operator.

Definition 6 (Annotated objective and default literals). *Let $A : \mu$ be an annotated atom constructed from the complete lattice $(\mathcal{T}, \preccurlyeq)$ with an explicit negation operator "$\neg$". For simplicity, assume that $\mu \in \mathcal{T}$. Then*

- *$A : \mu$, and $\neg A : \mu = A : \neg\mu$ are annotated objective literals. We use the notation $L : \mu$ to refer this type of literal;*
- *$not(A : \mu)$, and $not(\neg A : \mu) = not(A : \neg\mu)$ are annotated default literals. Similarly, we use $not(L : \mu)$ to denote annotated default literals.*

The extension of the satisfaction relation to objective literals is straightforward. By definition $\neg A : \mu$ equals $A : \neg\mu$. Therefore, $I \models \neg A : \mu$ iff $I \models A : \neg\mu$ iff $I(A) \succcurlyeq \neg\mu$. Notice that $\neg\mu$ is an element of $\mathcal{T}$. We conclude that in annotated programs without default negation, the explicit negation operator is just syntactic sugar. The syntax of Generalized Annotated Programs is appropriately extended:

Definition 7 (Normal Annotated Logic Programs). *A* normal annotated logic program *is a set of annotated clauses of the form:*

$$L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m \& not(H_1 : \vartheta_1) \& \ldots \& not(H_n : \vartheta_n) \quad (m, n \geq 0)$$

where the $L_i : \mu_i$ are annotated objective literals and the $not(H_j : \vartheta_j)$ are annotated default literals.

For default negation, the definition of the satisfaction relation is more intricate. One cannot simply define $I \models not(A : \mu)$ via $I \not\models A : \mu$, as can be seen from the next example.

Example 6. Consider the normal annotated logic program on the lattice L_2:

$$a : \top \leftarrow not(a : \top)$$

The single model of this program is $I(a) = \top$. However, this is contrary to the usual requirement of a logic program: every true literal should be supported, roughly meaning that it should be implied only by the set of rules with true body for it, whose conjuncts are each supported. One might conclude that under these conditions this program has no computationally relevant model, as this single rule becomes an equivalence, but the body is not supported.

One of the competing solutions to this problem is given by the well-founded semantics, for which the literal becomes undefined in the above situation. We adhere to this stream, and proceed by defining the meaning of the default negation operator via an alternating fixpoint construction, very similar to the original one of well-founded semantics [8] and amalgamation logic [13]. The crux of this technique is the notion of a Gelfond-Lifschitz like operator:

Definition 8 ($\Gamma_P^{\mathcal{T}}$ operator). *Let P be a normal annotated logic program, over the complete lattice $(\mathcal{T}, \preccurlyeq)$. Let I be an interpretation for P. The division of P by I over $\mathcal{T}$ is the generalized logic program*

$$\begin{aligned}\frac{P}{I}\mathcal{T} = \{\, & L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m | \\ & L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m \& not(H_1 : \vartheta_1) \& \ldots \& not(H_n : \vartheta_n) \in P \\ & \textit{and } I \not\models H_1 : \vartheta_1 \textit{ and } \ldots \textit{ and } I \not\models H_n : \vartheta_n \,\}\end{aligned}$$

Then the operator $\Gamma_P^{\mathcal{T}}$, maps interpretations to interpretations as follows:

$$\Gamma_P^{\mathcal{T}}(I) = \mathrm{lfp}\ T_{\frac{P}{I}\mathcal{T}}$$

That is, the least fixpoint of the immediate consequences operator T applied to the division of P by I with respect to $\mathcal{T}$.

Proposition 1 (Anti-monotonicity of $\Gamma_P^{\mathcal{T}}$). *[13] Let P be a normal annotated logic program over the complete lattice $(\mathcal{T}, \preccurlyeq)$. Let I and J be interpretations for P. Then $I \preccurlyeq J$ implies that $\Gamma_P^{\mathcal{T}}(J) \preccurlyeq \Gamma_P^{\mathcal{T}}(I)$.*

With this operator, the well-founded semantics can be extended to the more general setting of normal annotated logic programs. This result is provided in [13]. Basically, the true atoms in the well-founded annotated semantics are given by the least fixpoint $T = \Gamma_P^{\mathcal{T}} \Gamma_P^{\mathcal{T}}(T)$, and the default ones are obtained from $F = \mathcal{H}_P^{\mathcal{T}} - \Gamma_P^{\mathcal{T}}(T)$, where $\mathcal{H}_P^{\mathcal{T}}$ is the set of all annotated objective literals (the annotated Herbrand base). We refer to this fixpoint semantics as *the well-founded annotated semantics.*

Example 7. Consider the normal annotated logic program over $\mathcal{FOUR}$:

$$\begin{aligned}& a : \mathbf{t} \leftarrow not(b : \mathbf{t}). \\ & b : \mathbf{t} \leftarrow not(a : \mathbf{t}). \\ & b : \mathbf{f}.\end{aligned}$$

According to the well-founded annotated semantics, these literals are entailed by the program:

$$\{a : \bot, b : \mathbf{f}, b : \bot\} \cup not\, \{a : \mathbf{f}, a : \top\}$$

For lattice $\mathcal{FOUR}$, there is a natural explicit negation operator, where $\neg\bot = \bot$, $\neg\mathbf{f} = \mathbf{t}$, $\neg\mathbf{t} = \mathbf{f}$, and $\neg\top = \top$. What is odd about the above result is that though we have $b : \mathbf{f}$ we do not have $not(\neg b : \mathbf{f})$, which is the same as $not(b : \mathbf{t})$, and therefore we cannot conclude $a : \mathbf{t}$!

This example shows the well-founded annotated semantics does not comply with the coherence principle, which would entail $not(\neg b : \mathbf{f})$ from $b : \mathbf{f}$. In our opinion, this is unsatisfactory. Nevertheless, the coherence property cannot be easily enforced on the well-founded annotated semantics. A naïve approach would resort to the semi-normal program, an approach used by the well-founded semantics with explicit negations (WFSX) [2]. However, in some situations the resulting semantics might not be coherent, in particular when we have undefined literals in the model. The next example illustrates this.

Example 8. Consider the lattice over the set of elements $\{\bot, \mathbf{t1}, \mathbf{t2}, \mathbf{f1}, \mathbf{f2}, \top\}$ and with ordering relation $\bot \prec \mathbf{t1}, \bot \prec \mathbf{f1}, \mathbf{t1} \prec \mathbf{t2}, \mathbf{f1} \prec \mathbf{f2}, \mathbf{t2} \prec \top, \mathbf{f2} \prec \top$. The explicit negation operator $\neg$ is given by $\neg\bot = \bot$, $\neg\mathbf{t1} = \mathbf{f1}$, $\neg\mathbf{f1} = \mathbf{t1}$, $\neg\mathbf{t2} = \mathbf{f2}$, $\neg\mathbf{f2} = \mathbf{t2}$, $\neg\top = \top$. Let P be the program:

$$a : \mathbf{t1}. \qquad a : \mathbf{f2} \leftarrow b : \mathbf{t1}. \qquad b : \mathbf{t1} \leftarrow not(b : \mathbf{t1}).$$

Its semi-normal version P_s is:

$$\begin{array}{ll} a : \mathbf{t1} \leftarrow not(a : \mathbf{f1}). & \quad b : \mathbf{t1} \leftarrow not(b : \mathbf{t1}) \ \& \ not(b : \mathbf{f1}). \\ a : \mathbf{f2} \leftarrow b : \mathbf{t1} \ \& \ not(a : \mathbf{t2}). & \end{array}$$

For extended programs without annotations, the well-founded semantics can be derived as the $\preccurlyeq$-least fixpoint of $\Gamma_P \Gamma_{P_s}$, where P_s denotes the semi-normal rewrite of P. The computation of the least fixpoint of $\Gamma_P \Gamma_{P_s}$ proceeds as follows:

$$\begin{aligned} I_0 &= \{a = \bot, b = \bot\} \\ \Gamma_{P_s} I_0 &= \{a = \top, b = \mathbf{t1}\} \\ I_1 = \Gamma_P \Gamma_{P_s} I_0 &= \{a = \mathbf{t1}, b = \bot\} \\ \Gamma_{P_s} I_1 &= \{a = \top, b = \mathbf{t1}\} \\ I_2 &= \Gamma_P \Gamma_{P_s} I_1 \end{aligned}$$

The annotated literals true in the model are $\{a : \bot, a : \mathbf{t1}, b : \bot\} \cup not\,\{b : \mathbf{f1}, b : \mathbf{t2}, b : \mathbf{f2}, b : \top\}$. Thus we have $a : \mathbf{t1}$ but $not(a : \mathbf{f1})$ is not entailed! Coherence is not satisfied.

As the example shows, coherence is impeded because $a : \mathbf{f2}$ is undefined. $a : \mathbf{f2}$ is not being falsified via its semi-normal rule because $a : \mathbf{t1}$ is not strong enough; $a : \mathbf{t2}$ being required to do so. One way of guaranteeing coherence, and the one we follow in this paper, is by avoiding these situations. We achieve that by removing rules from the program which can destroy coherence. This is accomplished by an extension to the semi-normal program called the *down semi-normal program*, whose construction requires that the down-set[2] of every element in the lattice must be finite. Another solution, to be expounded elsewhere, is to introduce a tuneable coherence in complete lattices.

[2] In an ordered set P the down-set of x, denoted by $\downarrow x$ is $\{y \in P | y \leq x\}$.

Definition 9 (Down semi-normal program). *Let P be a normal annotated logic program over the finite lattice $(\mathcal{T}, \preccurlyeq)$ with explicit negation operator "$\neg$". The down semi-normalized program version of P, denoted by P_{ds}, is the normal annotated logic program obtained as follows:*

If $L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m \& not(H_1 : \vartheta_1) \& \ldots \& not(H_n : \vartheta_n) \in P$ then let $\{\tau_1, \ldots, \tau_o\} = (\downarrow \neg\mu_0) - \{\bot\}$. The following rule is in P_{ds} :

$$
\begin{aligned}
L_0 : \mu_0 \leftarrow & L_1 : \mu_1 \& \ldots \& L_m : \mu_m \& \\
& not(H_1 : \vartheta_1) \& \ldots \& not(H_n : \vartheta_n) \& not(L_0 : \tau_1) \& \ldots \& not(L_0 : \tau_o)
\end{aligned}
$$

Note that the finiteness condition is necessary to guarantee that each body in the down semi-normal program is finite. This simplifies the presentation in the finite case. The down semi-normal can then be used to define the new operator on programs $\Gamma^{\mathcal{T}}_{P_{ds}}$. However, a more general operator can be easily defined to work on arbitrary complete lattices, by including the down semi-normalization condition directly in the program division operation:

Definition 10 ($\digamma^{\mathcal{T}}_P$ operator). *Let P be a normal annotated logic program, over the complete lattice $(\mathcal{T}, \preccurlyeq)$. Let I be an interpretation for P. The down division of P by I over $\mathcal{T}$ is the generalized logic program*

$$
\begin{aligned}
P/^{\mathcal{T}} I = \{\, & L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m | \\
& L_0 : \mu_0 \leftarrow L_1 : \mu_1 \& \ldots \& L_m : \mu_m \& not(H_1 : \vartheta_1) \& \ldots \& not(H_n : \vartheta_n) \in P \\
& \textit{and } I \not\models H_1 : \vartheta_1 \textit{ and } \ldots \textit{ and } I \not\models H_n : \vartheta_n \textit{ and} \\
& \textit{for all } \tau \in (\downarrow \neg\mu_0) - \{\bot\} \textit{ then } I \not\models L_0 : \tau \,\}
\end{aligned}
$$

The operator $\digamma^{\mathcal{T}}_P$, mapping interpretations to interpretations, is defined by:

$$
\digamma^{\mathcal{T}}_P(I) = \text{lfp } T_{P/^{\mathcal{T}} I}
$$

The proof of anti-monotonicity of $\digamma^{\mathcal{T}}_P$ is straightforward.

Proposition 2 (Anti-monotonicity of $\digamma^{\mathcal{T}}_P$). *Let P be a normal annotated logic program over the complete lattice $(\mathcal{T}, \preccurlyeq)$. Let I and J be interpretations for P. Then $I \preccurlyeq J$ implies that $\digamma^{\mathcal{T}}_P(J) \preccurlyeq \digamma^{\mathcal{T}}_P(I)$.*

Proof. Our proof relies on the fact that when $I \preccurlyeq J$, then the program $P/^{\mathcal{T}} J$ has fewer rules than $P/^{\mathcal{T}} I$. By monoticity of the immediate consequences operator on the program the result immediately follows (if we have more rules one can derive more truths). Assume that a rule of P with head $L : \mu$ is removed in program $P/^{\mathcal{T}} I$. We show that this rule is also removed in $P/^{\mathcal{T}} J$. This is due to at least one of the following cases:

1. There is a default annotated literal $not(H : \vartheta)$ in the body of the rule such that $I \models H : \vartheta$. But since $I \preccurlyeq J$ then $J \models H : \vartheta$. Therefore the rule also does not belong to $P/^{\mathcal{T}} J$.

2. There is a $\tau \in (\downarrow \neg\mu) - \{\bot\}$ such that $I \models L : \tau$. But then $J \models L : \tau$. Therefore the rule does not appear in $P/^{\mathcal{T}} J$.

We finally obtain the intended alternating fixpoint operator construction:

Proposition 3 (Monotonicity of $\Gamma_P^{\mathcal{T}} F_P^{\mathcal{T}}$). *Let P be a normal annotated logic program over the complete lattice $(\mathcal{T}, \preccurlyeq)$ with explicit negation operator "$\neg$". Let $I \preccurlyeq J$ be two interpretations for P. Then*

$$I \preccurlyeq J \textit{ implies } \Gamma_P^{\mathcal{T}}\left(F_P^{\mathcal{T}}(I)\right) \preccurlyeq \Gamma_P^{\mathcal{T}}\left(F_P^{\mathcal{T}}(J)\right)$$

When the program and associated truth-value lattice are clear from context we omit them from the operators. Also, it should be clear to the reader that $F_P^{\mathcal{T}}$ coincides with $\Gamma_{P_{ds}}^{\mathcal{T}}$ when $\mathcal{T}$ is finite. To further simplify notation we denote the combination of operators $\Gamma_P^{\mathcal{T}} F_P^{\mathcal{T}}$ with the notation $\Gamma\Gamma_{ds}$, whenever confusion does not arise.

Since the alternating fixpoint construction $\Gamma\Gamma_{ds}$ is monotonic it has always a least fixpoint, which can be "obtained" by iterating from the least interpretation, Δ, where for every atom A in the language we have $\Delta(A) = \bot$. The semantics of normal annotated logic programs follows.

Definition 11 (Down coherent well-founded semantics). *Let P be a normal annotated logic program over the complete lattice $(\mathcal{T}, \preccurlyeq)$ with explicit negation operator "$\neg$". Let $\mathcal{M}$ be the least fixpoint of $\Gamma\Gamma_{ds}$. Its down coherent well-founded semantics is given by*

$$\{A : \mu \mid \mathcal{M}(A) = \vartheta \textit{ and } \mu \preccurlyeq \vartheta\} \cup \{not(A : \mu) \mid (\Gamma_{ds}\mathcal{M})(A) = \vartheta \textit{ and } \mu \not\preccurlyeq \vartheta\}$$

The least fixpoint $\mathcal{M}$ of $\Gamma\Gamma_{ds}$ determines the true annotated literals while the default ones are those not belonging to $\Gamma_{ds}\mathcal{M}$.

Example 9. Consider the program and lattice of Ex. 7. First, note that the down semi-normal version of P is:

$$\begin{aligned}
&a : \mathbf{t} \leftarrow not(b : \mathbf{t}) \ \&\ not(a : \mathbf{f}).\\
&b : \mathbf{t} \leftarrow not(a : \mathbf{t}) \ \&\ not(b : \mathbf{f}).\\
&b : \mathbf{f} \leftarrow not(b : \mathbf{t}).
\end{aligned}$$

The semantics of the program is iteratively obtained as follows:

$$\begin{aligned}
I_0 &= \Delta & &= \{a = \bot, b = \bot\}\\
\Gamma_{ds} I_0 &= \mathrm{lfp}\,\{a : \mathbf{t} \leftarrow; b : \mathbf{t} \leftarrow; b : \mathbf{f} \leftarrow\} & &= \{a = \mathbf{t}, b = \top\}\\
I_1 = \Gamma\Gamma_{ds} I_0 &= \mathrm{lfp}\,\{b : \mathbf{f} \leftarrow\} & &= \{a = \bot, b = \mathbf{f}\}\\
\Gamma_{ds} I_1 &= \mathrm{lfp}\,\{a : \mathbf{t} \leftarrow; b : \mathbf{f} \leftarrow\} & &= \{a = \mathbf{t}, b = \mathbf{f}\}\\
I_2 = \Gamma\Gamma_{ds} I_1 &= \mathrm{lfp}\,\{a : \mathbf{t} \leftarrow; b : \mathbf{f} \leftarrow\} & &= \{a = \mathbf{t}, b = \mathbf{f}\}\\
\Gamma_{ds} I_2 &= \mathrm{lfp}\,\{a : \mathbf{t} \leftarrow; b : \mathbf{f} \leftarrow\} & &= \{a = \mathbf{t}, b = \mathbf{f}\}\\
I_3 = \Gamma\Gamma_{ds} I_2 &= I_2
\end{aligned}$$

Applying now Def. 11 we get the model:

$$\mathcal{M} = \{a : \bot, a : \mathbf{t}, b : \bot, b : \mathbf{f}\} \cup not\,\{a : \mathbf{f}, a : \top, b : \mathbf{t}, b : \top\}$$

One can easily check this is the expected model, and that coherence is verified.

Example 10. Let us return to the program and lattice of Ex. 8. The down semi-normal version P_{ds} is:

$$\begin{aligned}
&b : \mathbf{t1} \leftarrow not(b : \mathbf{t1})\ \&\ not(b : \mathbf{f1}).\\
&a : \mathbf{t1} \leftarrow not(a : \mathbf{f1}).\\
&a : \mathbf{f2} \leftarrow b : \mathbf{t1}\ \&\ not(a : \mathbf{t2})\ \&\ not(a : \mathbf{t1}).
\end{aligned}$$

Note that in the last rule we have added the default literals $not(a : \mathbf{t2})$ and $not(a : \mathbf{t1})$ to the body of the rule. We now get the expected results:

$$\begin{aligned}
I_0 &= \Delta & &= \{a = \bot, b = \bot\}\\
\Gamma_{ds} I_0 &= \mathrm{lfp}\,\{a : \mathbf{t1} \leftarrow; a : \mathbf{f2} \leftarrow b : \mathbf{t1}; b : \mathbf{t1} \leftarrow\} & &= \{a = \top, b = \mathbf{t1}\}\\
I_1 = \Gamma \Gamma_{ds} I_0 &= \mathrm{lfp}\,\{a : \mathbf{t1} \leftarrow; a : \mathbf{f2} \leftarrow b : \mathbf{t1}\} & &= \{a = \mathbf{t1}, b = \bot\}\\
\Gamma_{ds} I_1 &= \mathrm{lfp}\,\{a : \mathbf{t1} \leftarrow; b : \mathbf{t1} \leftarrow\} & &= \{a = \mathbf{t1}, b = \mathbf{t1}\}\\
I_2 = \Gamma \Gamma_{ds} I_1 &= I_1
\end{aligned}$$

The literals true in the model are:

$$\{a : \bot, a : \mathbf{t1}, b : \bot\} \cup not\,\{a : \mathbf{f1}, a : \mathbf{t2}, a : \mathbf{f2}, a : \top, b : \mathbf{f1}, b : \mathbf{t2}, b : \mathbf{f2}, b : \top\}$$

Clearly coherence is obeyed.

However the semantics in some cases is "overly" coherent. The following illustrates this fact.

Example 11. Consider the lattice over the set of elements $\{\bot, \mathbf{t1}, \mathbf{t2}, \top\mathbf{1}, \mathbf{f1}, \mathbf{f2}, \top\mathbf{2}\}$ and with ordering relation $\bot \prec \mathbf{t1}, \bot \prec \mathbf{f1}, \mathbf{t1} \prec \top\mathbf{1}, \mathbf{f1} \prec \top\mathbf{1}, \top\mathbf{1} \prec \mathbf{t2}, \top\mathbf{1}, \prec \mathbf{f2}, \mathbf{t2} \prec \top\mathbf{2}, \mathbf{f2} \prec \top\mathbf{2}$. The explicit negation operator $\neg$ is given by $\neg\bot = \bot$, $\neg\mathbf{t1} = \mathbf{f1}$, $\neg\mathbf{f1} = \mathbf{t1}$, $\neg\top 1 = \top 1$, $\neg\mathbf{t2} = \mathbf{f2}$, $\neg\mathbf{f2} = \mathbf{t2}$, $\neg\top 2 = \top 2$. The program consisting of the single fact $a : \mathbf{t2}$ has model M:

$$\begin{gathered}
\{a : \bot, a : \mathbf{t1}, a : \mathbf{f1}, a : \top\mathbf{1}, a : \mathbf{t2}\}\\
\cup\\
not\,\{a : \bot, a : \mathbf{t1}, a : \mathbf{f1}, a : \top\mathbf{1}, a : \mathbf{t2}, a : \mathbf{f2}, a : \top 2\}
\end{gathered}$$

In which both $a : \mathbf{t2}$ and $not(a : \mathbf{t2})$ are present.

Te presence of both $a : \mathbf{t2}$ and $not(a : \mathbf{t2})$ in the model M requires some explanation. Consider the atom $a : \top\mathbf{1}$. Now, since $M \models a : t2$, by Definition 3 $M \models a : \top 1$. But $\neg a : \top 1 = a : \top 1$, so that $M \models \neg a : \top 1$. By coherency, we should have $M \not\models a : \top 1$, which by Definition 3 means that $M \not\models a : t2$, accounting for the paraconsistency.

This approach to coherency, may be termed *strong* in that it dictates that if a literal is false to some degree then it must be false for **all** higher degrees. In some cases strong coherency may be desirable, but not in others. We are currently working on a spectrum of annotated semantics where the intended degree of coherence can be tuned, e.g. by not requiring propagation of falsity to **all** higher degrees. In particular, one may want to allow for a undefinedness at a higher truth value not to be overriden by falsity at a weaker truth value. Thus, the introduction of coherence into annotated programs raises some non-trivial issues in the propagation of paraconsistency.

4 Embeddings

We next show how down coherent well-founded semantics extends several well-known semantics of logic and annotated programs. We assume the reader is acquainted with the syntax and definitions of the following semantics.

An embedding of the well-founded annotated semantics with a complete truth-value lattice having an explicit negation operator is given below. The rationale is to put two copies of the truth-value lattice side by side, merging their two bottom elements, and putting a new top element over both sub-lattices. The negation operator maps an element onto its corresponding element at the other lattice copy, and so provides the desired symmetry along the vertical axis.

Proposition 4 (Well-founded annotated semantics). *Let P be a normal annotated logic program over the complete lattice $(\mathcal{T}, \preccurlyeq)$. Construct the new lattice $(2\mathcal{T}, \preccurlyeq_2)$ as follows. Let*

$$2\mathcal{T} = \{(\bot, \bot)\} \cup \{(\mathbf{f}, \mu) \mid \mu \in \mathcal{T} - \{\bot\}\} \cup \{(\mathbf{t}, \mu) \mid \mu \in \mathcal{T} - \{\bot\}\} \cup \{(\top, \top)\}$$

and the ordering on $2\mathcal{T}$ and the explicit negation operator $\neg$ be defined by:

- *For every $(\mu, \vartheta) \in 2\mathcal{T}$ we have $(\bot, \bot) \preccurlyeq_2 (\mu, \vartheta)$, and $\neg(\bot, \bot) = (\bot, \bot)$;*
- *For every $(\mathbf{f}, \mu), (\mathbf{f}, \vartheta) \in 2\mathcal{T}$ we have $(\mathbf{f}, \mu) \preccurlyeq_2 (\mathbf{f}, \vartheta)$ iff $\mu \preccurlyeq \vartheta$. Furthermore, $\neg(\mathbf{f}, \mu) = (\mathbf{t}, \mu)$;*
- *For every $(\mathbf{t}, \mu), (\mathbf{t}, \vartheta) \in 2\mathcal{T}$ we have $(\mathbf{t}, \mu) \preccurlyeq_2 (\mathbf{t}, \vartheta)$ iff $\mu \preccurlyeq \vartheta$. Furthermore, $\neg(\mathbf{t}, \mu) = (\mathbf{f}, \mu)$;*
- *For every $(\mu, \vartheta) \in 2\mathcal{T}$ we have $(\mu, \vartheta) \preccurlyeq_2 (\top, \top)$, and $\neg(\top, \top) = (\top, \top)$.*

Construct program P^2 over the new lattice, from P, by substituting every occurrence of $\bot$ by $(\bot, \bot)$ and every other literal μ by $(\mathbf{t}, \mu)$. Then a literal is derived from program P over lattice $\mathcal{T}$ under the well-founded annotated semantics iff its corresponding literal – substituting its annotation either to $(\bot, \bot)$ or $(\mathbf{t}, \mu)$ – is derived from program P^2 over lattice $2\mathcal{T}$ under the down coherent well-founded semantics.

The technique of Prop. 4 should now be clear. Note that literals annotated with $(\top, \top)$ or $(\mathbf{f}, \mu)$ never appear in P^2, and therefore objective literals annotated with those truth-values are never derived from P^2. Thus the extra default annotated literals introduced in the down semi-normal program are never false. It is this fact that ensures that the fixpoint of $\Gamma_P^{\mathcal{T}}\Gamma_P^{\mathcal{T}}$ over the lattice $2\mathcal{T}$ coincides with that of $\Gamma_P^{\mathcal{T}}\Gamma_P^{\mathcal{T}}$ over $\mathcal{T}$, and so guarantees the validity of the embedding. The embedding requires the smallest addition of new truth values to the original lattice so that the embedding into down coherent well-founded semantics is valid. This is important since it is desirable to keep the lattice as simple and as close as possible to the original one.

Obviously, the lattice L_2 of Ex. 1 provides an embedding of Well-founded Semantics into the Well-founded Annotated Semantics. By resorting to the above result and letting $\mathcal{T} = L_2$ we obtain an embedding of well-founded semantics into down coherent well-founded semantics. Notice that

$$2L_2 = \{(\bot,\bot),(\mathbf{f},\top),(\mathbf{t},\top),(\top,\top)\}$$

Accordingly, the ordering relation is:

$$(\bot,\bot) \preccurlyeq_2 (\mathbf{f},\top) \quad (\bot,\bot) \preccurlyeq_2 (\mathbf{t},\top) \quad (\mathbf{f},\top) \preccurlyeq_2 (\top,\top) \quad (\mathbf{t},\top) \preccurlyeq_2 (\top,\top)$$

and the explicit negation operator behaves as follows:

$$\neg(\bot,\bot) = (\bot,\bot) \quad \neg(\mathbf{f},\top) = (\mathbf{t},\top) \quad \neg(\mathbf{t},\top) = (\mathbf{f},\top) \quad \neg(\top,\top) = (\top,\top)$$

The reader may verify that $(2L_2, \preccurlyeq_2)$ is isomorphic to Belnap's logic $\mathcal{FOUR}$. This justifies the following corollary:

Corollary 1 (Well-founded semantics). *Let P be a normal logic program, and P^{WFS} the following normal annotated logic program over the lattice $\mathcal{FOUR}$ with the usual explicit negation operator:*

$$\begin{aligned} P^{WFS} = \{&A_0 : \mathbf{t} \leftarrow A_1 : \mathbf{t}\& \ldots, A_n : \mathbf{t}\&not\ (B_1 : \mathbf{t})\& \ldots\&not\ (A_m : \mathbf{t}) \\ &\textit{such that } A_0 \leftarrow A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_m \textit{ belongs to } P\}\end{aligned}$$

Then A, respectively not A, belongs to the well-founded model of P iff $A : \mathbf{t}$, respectively not $(A : \mathbf{t})$, belongs to the down coherent well-founded model of P^{WFS}.

The transformation guarantees that all semi-normalization literals introduced in the semi-normal program transformation are of the form $not(A : \mathbf{f})$. All rules in P^{WFS} have a head annotated with $\mathbf{t}$, therefore neither $\top$ nor $\mathbf{f}$ are derivable, and therefore $not(A : \mathbf{f})$ is always true. Thus, the $\Gamma^{\mathcal{FOUR}}\Gamma_s^{\mathcal{FOUR}}$ alternating fixpoint construction coincides with the original $\Gamma\Gamma$ construction of [8].

More importantly, the same construction can be used to extend an arbitrary lattice with an explicit negation operator, where it is possible to categorically state the truth or falsity of literals and have coherence enforced. A similar effect can be obtained with the lattice operator $|$, where $\mathcal{T}|\mathcal{T}$ is the lattice $2\mathcal{T}$ without elements $(\mathbf{f},\top)$ and $(\mathbf{t},\top)$. This corresponds to merging the two top elements of the two lattice instances, as explained before. The lattice $\mathcal{T}|\mathcal{T}$ is normally used when there is no need to distinguish between the veracity and falsity of the top element of $\mathcal{T}$; in most situations this corresponds to interpreting $\top$ in $\mathcal{T}$ already as "contradiction".

An application of the previous techniques and results gives a natural embedding of the Paraconsistent Extended Well-founded Semantics [3, 4] into Down Coherent Well-Founded Annotated programs. The Paraconsistent Extended Well-founded Semantics (denoted by WFM_p) is obtained from Well-founded Annotated Semantics by using the complete set of truth values of lattice $\mathcal{FOUR}$.

Proposition 5 (Paraconsistent extended well-founded semantics). *[3] Consider the extended logic program P. The normal annotated logic program $P^{\neg}$ over $\mathcal{FOUR}$ obtained from P by substituting every occurrence of $\neg A$ by $A : \mathbf{f}$, and of A by $A : \mathbf{t}$. Let $\mathcal{M}$ be its coherent well-founded model. Then*

- *A belongs to $WFM_p(P)$ iff $A : \mathbf{t}$ belongs to $\mathcal{M}$;*
- *$\neg A$ belongs to $WFM_p(P)$ iff $A : \mathbf{f}$ belongs to $\mathcal{M}$;*
- *not A belongs to $WFM_p(P)$ iff not $(A : \mathbf{t})$ belongs to $\mathcal{M}$;*
- *not $\neg A$ belongs to $WFM_p(P)$ iff not $(A : \mathbf{f})$ belongs to $\mathcal{M}$.*

If programs contain only literals annotated with $\perp$ or $\mathbf{t}$ we obtain the well-founded semantics (as expected from Prop. 4). Moreover, if programs contain no default negated literals then the objective literals labeled with $\mathbf{t}$ in the down coherent well-founded model coincide with the ones true in the minimal Herbrand Model of the corresponding definite program. We thus have shown how to move from generalized annotated logic programs or from extended logic programs to their natural paraconsistent and coherent well-founded based annotated semantics.

5 Conclusions

The theorems in the previous section show that coherent well-founded annotated programs incorporate both extended and annotated logic programs. The practical importance of such a combination has been indicated by several of the examples. Example 3 uses annotations to formulate probabilistic data mining rules; Example 4 uses explicit negation to represent a contradiction in information arising from different sources; and Example 5 uses default negation as an instance of default reasoning based on probabilistic rules. Thus, annotations, explicit negation, and default negation are all required together for this deductive database example. The generality and practical applicability of coherent well-founded annotated programs with strong coherency as so far described indicates that their efficient implementation is a worthwhile task; the simplicity of their fixpoint definition suggests that they can be implemented by extending a system, such as XSB, that computes the well-founded semantics, a task which is now underway.

Acknowledgements

We thank PRAXIS XXI project MENTAL (Mental Agents Architecture in Logic) and FLAD-NSF project REAP for their support. This work was also partially supported by NSF grants CCR-9702581, EIA-97-5998, and INT-96-00598. We also thank José Alferes for his helpful comments.

References

1. S. Adali and V. S. Subrahmanian. Amalgamating knowledge bases, III: Algorithms, data structures, and query processing. *J. of Logic Programming*, 28(1):45–88, 1996.
2. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*. LNAI volume 1111, Springer–Verlag, 1996.
3. Carlos Viegas Damásio. *Paraconsistent Extended Logic Programming with Constraints*. PhD thesis, Universidade Nova de Lisboa, October 1996.
4. Carlos Viegas Damásio and Luís Moniz Pereira. A survey of paraconsistent semantics for logic programas. In D. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer, 1998.
5. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *International Conference on Logic Programming 1997*, pages 391–495, 1997.
6. M. Van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.
7. J. Freire, P. Rao, K. Sagonas, T. Swift, and D. S. Warren. XSB: A system for efficiently computing the well-founded semantics. In *Fourth LPNMR*, pages 430–440, 1997.
8. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
9. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.
10. M. Kifer and E. Lozinskii. A logic for reasoning with inconsistency. *J. of Automated Reasoning*, 8:179–215, 1992.
11. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
12. C. Sakama and K. Inoue. Paraconsistent Stable Semantics for Extended Disjunctive Programs. *J. of Logic and Computation*, 5(3):265–285, 1995.
13. V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.