Paraconsistent Semantics for Extended Logic Programs

Ofer Arieli
Department of Computer Science, The Academic College of Tel-Aviv
Antokolsky 4, P.O.Box 16131, Tel-Aviv 61161, Israel.
oarieli@mta.ac.il

Abstract *We introduce a declarative semantics for extended logic programs, and demonstrate its usefulness for reasoning with uncertainty. We show that this is a robust formalism that overcomes some drawbacks of related fixpoint semantics for incomplete or inconsistent logic programs.*

Keywords: logic programing, fixpoint semantics, paraconsistency, multiple-valued logics.

1 Introduction

It is well-known that the restricted syntactical structure of standard logic programs[1] limits their expressive power. This means, in particular, that it is not possible to properly represent uncertain information (e.g., contradictions or partial knowledge) by such programs. The standard way of dealing with this problem (in the context of logic programing) is to consider *extended logic programs*, in which two kinds of negation operators may appear in the clause bodies, and one of them may also appear in the clause heads. It is usual to intuitively refer to one of these operators (denoted here by $\neg$) as representing explicit negative information. The other negation operator (denoted here by `not`) is intuitively related to a more implicit negative data, and it is often associated with a "negation-as-failure" (to prove or verify the corresponding assertion on the basis of the available information).

Example 1 Let p, q, r be atomic formulae, and let `t` be a propositional constant that represents true assertions. Consider the following extended logic program:

$$\mathcal{P} = \{q \leftarrow \mathtt{t},\ p \leftarrow \mathtt{t},\ p \leftarrow r,\ \neg p \leftarrow \mathtt{not}\ \neg q\}$$

Intuitively, $\mathcal{P}$ may be understood such that both p and q are known to be true, p is defined in terms of r (where no information is available about r), and $\neg p$ holds provided that the negation of q cannot be shown. In this interpretation, $\mathcal{P}$ clearly lacks any information about r, and it contains inconsistent information regarding p. Thus, a plausible formalism for reasoning with $\mathcal{P}$ should not assume anything about r, and (unlike classical logic) should not give $\mathcal{P}$ a trivial semantics. That is, despite the contradictions in $\mathcal{P}$, not every formula may be inferred from it.[2]

Example 1 shows that an adequate formalism for giving semantics to extended logic programs must be *paraconsistent* [11], that is, inconsistent information should *not* entail every conclusion.[3] The next example shows that the underlying formalism should also be *nonmonotonic* (i.e., capable of changing the set of conclusions according to new data).

Example 2 Consider again the logic program $\mathcal{P}$ of Example 1, and suppose now that a new datum arrives, which indicates that if p holds then $\neg q$ must hold as well. The new program is therefore $\mathcal{P}' = \mathcal{P} \cup \{\neg q \leftarrow p\}$. Now, the

[1] I.e., set of clauses of the form $p \leftarrow l_1, \ldots, l_n$ where p is an atomic formula and $l_1, \ldots, l_n$ is a conjunction of literals.

[2] For instance, it is quite obvious that none of $\neg q$, r, or $\neg r$ should follow from $\mathcal{P}$.

[3] See [9] for a survey on paraconsistent systems.

information regarding p becomes consistent (as the condition for concluding $\neg p$ does not hold anymore), while the data regarding q turns to be inconsistent (and the data regarding r remains incomplete). A non-monotonic formalism should adapt itself to the new situation. In particular, while the query $\neg p$ should succeed w.r.t. $\mathcal{P}$, it should fail w.r.t. $\mathcal{P}'$.

In this paper we introduce a paraconsistent and non-monotonic declarative semantics for extended logic programs. For this we use Belnap's four-valued structure [6, 7], which is particularly useful for our purpose, since in addition to the "standard" classical values it also contains two other values for designating the two kinds of uncertainty mentioned above, namely: partial information and contradictory data. We show that the outcome is a fixpoint semantics for extended logic programs that is capable of pinpointing the incomplete and inconsistent parts of the data, while the remaining information may be regarded as classically consistent. [4]

2 Four-valued semantics

As we have noted above, our formalism is based on four-valued semantics. Reasoning with four truth values may be traced back to the 1950's [8, 16]. Here we use Belnap's four-valued algebraic structure $\mathcal{FOUR}$, introduced in [6, 7]. This structure consists of four elements: two elements (t, f) that correspond to the classical truth values, an element $(\bot)$ that intuitively represents lack of information, and an element $(\top)$ that may intuitively be understood as representing contradictions. These elements are simultaneously arranged in two partial orders. In one of them (denoted here by $\leq_t$), f is the minimal element, t is the maximal one, and $\bot, \top$ are two intermediate values that are incomparable. This partial order may be intuitively understood as representing differences in the amount of *truth* of each element. We denote by $\wedge$ and $\vee$ the meet and join operations w.r.t $\leq_t$. In the other partial order (denoted here by $\leq_k$), $\bot$ is the minimal element, $\top$ is the maximal one, and t, f are two intermediate values. This partial order intuitively represents differences in the amount of *knowledge* (or information) that each element exhibits. We denote by $\otimes$ and $\oplus$ the meet and join operations w.r.t $\leq_k$. A negation operator $\neg$ on $\mathcal{FOUR}$ reverses the $\leq_t$-order and preserves the $\leq_k$-order, thus $\neg t = f$, $\neg f = t$, $\neg\bot = \bot$, and $\neg\top = \top$.

A double-Hasse diagram of $\mathcal{FOUR}$ is shown in Figure 1.

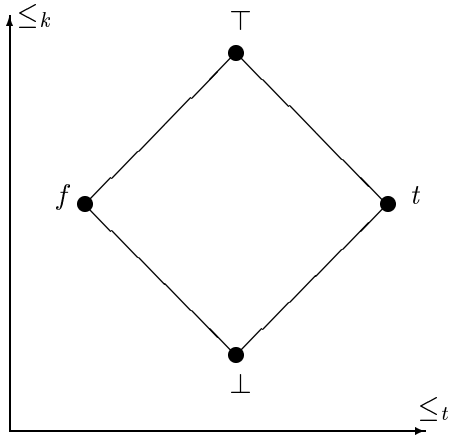

Figure 1: $\mathcal{FOUR}$

The various semantic notions are defined on $\mathcal{FOUR}$ as natural generalizations of similar classical ones: a *valuation* ν is a function that assigns a truth value in $\mathcal{FOUR}$ to each atomic formula. In what follows we shall sometimes write $\nu = \{p : x, q : y\}$ instead of $\nu(p) = x$, $\nu(q) = y$. Any valuation is extended to complex formulae in the obvious way. The set of the four-valued valuations is denoted by V^4.

The set of the *designated* truth values in $\mathcal{FOUR}$ (i.e., those elements in $\mathcal{FOUR}$ that represent true assertions) is $\mathcal{D} = \{t, \top\}$. A valuation ν *satisfies* a formula ψ iff $\nu(\psi) \in \mathcal{D}$. A valuation that assigns a designated value to every formula in a theory $\mathcal{P}$ is a *model* of $\mathcal{P}$. The set of all the models of $\mathcal{P}$ is denoted by $mod(\mathcal{P})$. The syntactical form of the formulae in $\mathcal{P}$ is the following:

[4] Due to a lack of space proofs are omitted here. Full proofs appear in [1].

Definition 3 In what follows p, q, r denote atomic formulae, $l, l_1, l_2, \ldots$ denote literals (i.e., atomic formulae that may be preceded by $\neg$), and $e, e_1, e_2, \ldots$ denote extended literals (i.e. literals that may be preceded by `not`). The complement of a literal l is denoted by $\overline{l}$. An *extended clause* is a formula $l \leftarrow e_1, \ldots, e_n$ where $n \geq 0$. A (possibly infinite) set $\mathcal{P}$ of extended clauses is called an *extended logic program*.[5] A clause (respectively, a set of clauses) without the operator `not` is called a *general clause* (respectively, a *general logic program*).

Let $\mathcal{P}$ be a general logic program. The meaning of conjunctions (,) and negations ($\neg$) is determined, respectively, by the $\leq_t$-meet and the negation operator on $\mathcal{FOUR}$.[6] This corresponds to the natural extensions for the multiple-valued case of the 2-valued interpretations of these connectives. However, this should not be the case with implication: as observed in [5, 15], in the context of multiple-valued semantics the material implication does *not* properly represent entailment. We therefore consider an alternative definition for the implication connective (see [1, 2] for a justification of this definition):

Definition 4 [2, 4] Let $x, y \in \mathcal{FOUR}$. Define: $x \leftarrow y = x$ if $y \in \mathcal{D}$, and $x \leftarrow y = t$ otherwise.

We conclude this section by defining two useful order relations on the models of a program $\mathcal{P}$.

Definition 5 A valuation $\nu_1 \in mod(\mathcal{P})$ is *k-smaller* than another valuation $\nu_2 \in mod(\mathcal{P})$ if for every atomic formula p, $\nu_1(p) \leq_k \nu_2(p)$. $\nu \in mod(\mathcal{P})$ is a *k-minimal model* of $\mathcal{P}$ if there is no other model of $\mathcal{P}$ that is k-smaller than ν.

Definition 6 [2, 3] A valuation $\nu_1 \in mod(\mathcal{P})$ is *more consistent* than another valuation $\nu_2 \in mod(\mathcal{P})$ if $\{p \mid \nu_1(p) = \top\} \subset \{p \mid \nu_2(p) = \top\}$. $\nu \in mod(\mathcal{P})$ is a *most consistent model* of $\mathcal{P}$ if there is no other model of $\mathcal{P}$ that is more consistent than ν.

3 Paraconsistent fixpoint semantics

We are now ready to introduce our fixpoint semantics for logic programs. First, we treat general logic programs (i.e., programs without negation-as-failure), and then we consider extended logic programs.

3.1 General logic programs

Definition 7 Given a general logic program $\mathcal{P}$, define for every $i \geq 1$ and every literal l,

$\nu_0^{\mathcal{P}}(l) = \bot$.

$$\mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\mathsf{l}) = \begin{cases} t & \text{if there is a } l \leftarrow Body \in \mathcal{P} \\ & \text{s.t. } \nu_{i-1}^{\mathcal{P}}(Body) \in \mathcal{D}, \text{ [7]} \\ \bot & \text{otherwise.} \end{cases}$$

$\nu_i^{\mathcal{P}}(l) = \mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\mathsf{l}) \oplus \neg \mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\overline{\mathsf{l}})$.

For a limit ordinal λ we define

$\mathsf{val}_{\lambda}^{\mathcal{P}}(\mathsf{l}) = \mathsf{max}_{\leq_{\mathsf{k}}}\{\mathsf{val}_{\alpha}^{\mathcal{P}}(\mathsf{l}) \mid \alpha < \lambda\}$,

$\nu_{\lambda}^{\mathcal{P}}(l) = \mathsf{val}_{\lambda}^{\mathcal{P}}(\mathsf{l}) \oplus \neg \mathsf{val}_{\lambda}^{\mathcal{P}}(\overline{\mathsf{l}})$.

For a propositional constant $\mathtt{x} \in \{\mathtt{t}, \mathtt{f}, \mathtt{c}, \mathtt{u}\}$ that is respectively associated with an element $x \in \{t, f, \top, \bot\}$ in $\mathcal{FOUR}$, we define

$\nu_i^{\mathcal{P}}(\mathtt{x}) = \mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\mathtt{x}) = x \quad (i = 0, 1, \ldots)$.

Note that $\nu_i^{\mathcal{P}}$ behaves as expected w.r.t. negation: since $\neg(x \oplus y) = \neg x \oplus \neg y$ for every $x, y \in \mathcal{FOUR}$, we have that

$$\neg\nu_i^{\mathcal{P}}(l) = \neg(\mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\mathsf{l}) \oplus \neg\mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\overline{\mathsf{l}})) = \\ = \neg\mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\mathsf{l}) \oplus \mathsf{val}_{\mathsf{i}}^{\mathcal{P}}(\overline{\mathsf{l}}) = \nu_{\mathsf{i}}^{\mathcal{P}}(\overline{\mathsf{l}}).$$

Proposition 8 *Let $\mathcal{P}$ be a general logic program. Then the sequence $\nu_0^{\mathcal{P}}, \nu_1^{\mathcal{P}}, \ldots$ is $\leq_k$-monotonic in $\mathcal{V}$*[4].

[5] Some formalisms also allow the appearance of implicit negations in the clause heads of extended logic programs; see e.g. [12] for a discussion on possible ways to understand default negation in the clause heads.

[6] We shall discuss the meaning of the negation as failure operator `not` in what follows.

[7] Note that $\nu_j^{\mathcal{P}}(Body) = \bigwedge_{l_i \in \mathcal{L}(Body)} \nu_j^{\mathcal{P}}(l_i)$, thus $\nu_j^{\mathcal{P}}(Body) \in \mathcal{D}$ iff $\forall l_i \in \mathcal{L}(Body)\ \nu_j^{\mathcal{P}}(l_i) \in \mathcal{D}$.

By Knaster-Tarski theorem [23], it follows from Proposition 8 that the sequence $\{\nu_i^{\mathcal{P}}\}$ has a $\leq_k$-least fixpoint. Denote this fixpoint by $\nu^{\mathcal{P}}$. An induced consequence relation $|\sim_\nu$ may now be defined as follows: $\mathcal{P} \,|\sim_\nu\, \psi$ iff $\nu^{\mathcal{P}}(\psi) \in \mathcal{D}$ (Thus, a formula ψ *follows* from a logic program $\mathcal{P}$, if $\nu^{\mathcal{P}}(\psi)$ is designated).

Proposition 9 *Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-minimal four-valued model of $\mathcal{P}$. Moreover, it is at least as consistent as any other model of $\mathcal{P}$, and the consequence relation $|\sim_\nu$ that is induced by it is non-monotonic and paraconsistent.*

Corollary 10 *Let $\mathcal{P}$ be a general logic program. Then:*
a) *$\nu^{\mathcal{P}}$ is the k-least model of $\mathcal{P}$,*
b) *$\nu^{\mathcal{P}}$ is a most consistent model of $\mathcal{P}$,*
c) *$\nu^{\mathcal{P}}$ is the k-minimal element among the most consistent models of $\mathcal{P}$.*

Corollary 10 implies that $\nu^{\mathcal{P}}$ minimizes the amount of knowledge that is pre-supposed, i.e. it does not assume anything that is not *really* known. The same corollary also shows that $\nu^{\mathcal{P}}$ is a most consistent model of $\mathcal{P}$. As such, it minimizes the amount of inconsistent belief in the set of clauses. This is in accordance with the intuition that while one has to deal with conflicts in a nontrivial way, contradictory data corresponds to inadequate information about the real world, and therefore it should be minimized.[8]

3.2 Extended logic programs

In this section we extend the fixpoint semantics for general logic programs, considered in the previous section, to extended logic programs. So now, in addition to the explicit negation $\neg$, the negation-as-failure operator (`not`) may also appear in the clauses bodies.

One way of understanding `not` in the four-valued setting is the following: If we don't know anything about p, i.e. we cannot prove either p or $\neg p$, then we cannot say anything about `not` p as well. Otherwise, if p has a designated value in the intended semantics (i.e., p is provable), then `not` p does not hold, and if p does not have a designated value (i.e., it is not provable), then `not` p holds. It follows, then, that `not` $t = f$, `not` $\top = f$, `not` $f = t$, and `not` $\bot = \bot$.

In what follows we use a transformation, similar to that of the well-founded semantics [25], for reducing extended logic programs to general logic programs. Then we use the formalism of the previous section for giving semantics to the general logic programs that are obtained.

Definition 11 Let ν be a four-valued valuation. The set S_ν that is *associated* with ν is the smallest set of literals that satisfies the following conditions: [9]

if $\nu(l) = t$ then $l \in S_\nu$,
if $\nu(l) = f$ then $\overline{l} \in S_\nu$,
if $\nu(l) = \top$ then $\{l, \overline{l}\} \subseteq S_\nu$.

Definition 12 Let $\mathcal{P}$ be an extended program and let S be a set of literals. The *reduction* of $\mathcal{P}$ w.r.t. S is the general logic program $\mathcal{P}\downarrow S$, obtained from $\mathcal{P}$ as follows:

1. Each clause that has a condition of the form `not` l, where $l \in S$, is deleted from $\mathcal{P}$.
2. Every occurrence of `not` l, where $\overline{l} \in S$, is eliminated from the (bodies of the) remaining clauses.[10]
3. Every occurrence of `not` l in the remaining clauses is replaced by the propositional constant `u`.

Now we are ready to define our fixpoint semantics for extended logic programs. Recall that $\nu^{\mathcal{P}}$ denotes the fixpoint semantics for a general logic program $\mathcal{P}$.

[8] See also [3].

[9] Such sets are sometimes called *answer sets* (for ν). We shall not use this terminology here, since inconsistent answer sets contain *every* literal, and this is not the case here.

[10] If a clause body becomes empty by this transformation, it is treated as consisting of the propositional constant `t`.

Definition 13 A valuation $\mu \in V^4$ is an *adequate solution* for an extended logic program $\mathcal{P}$, if it coincides with the fixpoint semantics of the general logic program obtained by reducing $\mathcal{P}$ w.r.t. the set that is associated with μ. In other words, μ is an adequate solution for $\mathcal{P}$ iff the following equation holds:

$$\mu = \nu^{\mathcal{P}\downarrow S_\mu}$$

Note 14 If the only negation operator that appears in $\mathcal{P}$ is $\neg$, then $\mathcal{P}$ is a general logic program, and so its unique adequate solution is $\nu^{\mathcal{P}}$. It follows, in particular, that the notion of adequate solutions of extended logic programs is a generalization of the fixpoint semantics for general logic programs.

Proposition 15 *Any adequate solution for $\mathcal{P}$ is a model of $\mathcal{P}$, and the consequence relation that is induced by it is non-monotonic and paraconsistent.*

As it is shown in Example 17 below, an extended logic program may have more than one adequate solution, and so one may use different preference criteria for choosing the best solutions among the adequate ones. In the case of general logic programs we have chosen $\leq_k$-minimization as the criterion for preferring the "best" model among the fixpoint valuations. This was justified by the fact that general logic programs may contain contradictory data, and so we want to minimize the redundant information as much as possible. In the present case we rather use the opposite methodology: since the negation-as-failure operator corresponds to incomplete information, we are dealing here with a lack of data, so this time we should try to restrict the effect of the negation-as-failure operator only to those cases in which indeed there is not enough data available. It follows, therefore, that now we should seek for a *maximal knowledge* (among the adequate solutions). [11]

[11] Informally, we use here a "min/max strategy": knowledge minimization of the contradictory components of the program, and knowledge maximization of its incomplete components.

Definition 16 μ is a *most adequate model* of $\mathcal{P}$ if it is a $\leq_k$-maximal adequate solution for $\mathcal{P}$. [12]

Example 17 Below we consider our semantics for some inconsistent and/or incomplete logic programs.

1. $\mathcal{P} = \{\neg p \leftarrow \texttt{not } p\}$.
   Intuitively, $\mathcal{P}$ represents a closed word assumption (CWA, [22]) regarding p: In the absence of any evidence for p, assume that $\neg p$ holds. $\mathcal{P}$ has two adequate solutions $\mu_1 = \{p : \bot\}$ and $\mu_2 = \{p : f\}$. But $\mu_2 >_k \mu_1$, so μ_2 is the most adequate model of $\mathcal{P}$.

2. $\mathcal{P} = \{p \leftarrow \texttt{not } p,\ q \leftarrow \texttt{t}\}$.
   The most adequate model here is $\{p : \bot,\ q : t\}$. This indeed seems to be the only reasonable interpretation in this case, since it distinguishes between the meaningful data in $\mathcal{P}$ ($\{q \leftarrow \texttt{t}\}$), and the meaningless data ($\{p \leftarrow \texttt{not } p\}$). Note also that the most adequate solution here coincides with the well-founded model [25] (for standard logic programs) of $\mathcal{P}$. Two-valued semantics, such as Gelfond-Lifschitz stable model semantics [14], do not provide any model for $\mathcal{P}$.

3. (Examples 1 and 2, revisited)
   $\mathcal{P} = \{q \leftarrow \texttt{t},\ p \leftarrow \texttt{t},\ p \leftarrow r,\ \neg p \leftarrow \texttt{not } \neg q\}$.
   The most adequate model here is $\{p : \top,\ q : t,\ r : \bot\}$. It reflects our expectation that no information about r is available, and since $\neg q$ does not follow from $\mathcal{P}$, the knowledge about p is contradictory. Note that according to the semantics given in [15, 19], $\mathcal{P}$ does not have any model, since it is not classically consistent.

   $\mathcal{P}' = \mathcal{P} \cup \{\neg q \leftarrow p\}$.
   As noted in Example 2 above, the new information that is added to $\mathcal{P}$ should cause a complete revision in the reasoner's belief about p and q. The most adequate model of $\mathcal{P}'$, $\{p : t,\ q : \top,\ r : \bot\}$, indeed reflects the expected result of such a revision.

[12] Note that by Proposition 15, μ is indeed a model of $\mathcal{P}$.

4 Concluding remarks

One of the main drawbacks of some related fixpoint semantics (such as those introduced in [15] and [19]) is that they become trivial in the presence of contradictions, and so these formalisms are not paraconsistent. We do believe that since inconsistent knowledge can and may be represented in extended logic programs, a plausible semantics for such programs should be able to draw meaningful conclusions (and reject others) despite the inconsistency. The fixpoint semantics considered here has such capabilities: it pinpoints the inconsistent and the incomplete parts of the data, and regards the rest of the information as classically consistent.

Another major difference between the semantics introduced here and some other semantics for extended logic programs (e.g, [13, 15, 17, 21]) concerns with the way negative data is related to its positive counterpart. While the formalisms of [13, 15, 17, 21] treat p and $\neg p$ as two different *atomic formulae*, we preserve the relation between an atomic formula and its negated atom. To see the importance of this, consider the following program (also considered in [5, Example 3.3.6] and [19, Example 1]):

$$\mathcal{P} = \{p \leftarrow \texttt{not}\ q,\ q \leftarrow \texttt{not}\ p,\ \neg p \leftarrow \texttt{t}\}$$

According to the approaches that treat $\neg p$ as (a strange way of writing) an atomic formula, the well-founded semantics would assign here t to $\neg p$, $\bot$ to p, and $\bot$ to q. So even though $\mathcal{P}$ is classically consistent, the distinction between p and $\neg p$ causes a counter-intuitive result here: since there is no way to refute p without relating it to $\neg p$, it is not possible to conclude q. In contrast, our semantics reflects the intuitive expectations in this case, and the unique adequate solution for $\mathcal{P}$ is $\{p{:}f, q{:}t\}$.

For another example, consider the following logic program [19, Example 6]:

$$\mathcal{P} = \{r \leftarrow \texttt{not}\ q,\ q \leftarrow \texttt{not}\ p,\ p \leftarrow \texttt{not}\ p,\ \neg q \leftarrow \texttt{t}\}$$

If $\neg q$ is considered as an atomic formula, this program has a single extended stable model, in which $\neg q$ is true and all the other atomic formulae (p, q, r) are unknown. This seems to be a counter-intuitive result, since in this case one expects that r would follow from $\mathcal{P}$. The unique adequate solution for $\mathcal{P}$ (and so its most adequate model) is $\{p{:}\bot,\ q{:}f,\ r{:}t\}$. According to this semantics r indeed follows from $\mathcal{P}$, as expected.[13]

Finally, we note that our approach may be incorporated with other techniques for improving the way knowledge is represented in the underlying program. For instance, by using the methodology proposed by Pereira et al. in [20], it is possible to represent preferences among different program rules by associating a different 'label' to each program rule, and then adding these labels as new conditions to the bodies of the rules. This enables an easy way to represent a hierarchy of rules in the language itself. For instance, the fact that under the conditions specified in $Body$ one should apply a rule labeled by l_1 instead of a rule labeled by l_2, is encoded by a preference rule like $\neg l_2 \leftarrow Body,\ l_1$.

The same paper also suggests a method for exception handling that may also be encoded in our framework. For instance, a rule like

$$\texttt{fly}(x) \leftarrow \texttt{bird}(x)$$

that states that *every* bird can fly, may be replaced by more cautious rules, such as

$$\texttt{fly}(x) \leftarrow \texttt{bird}(x), \texttt{not abnormal\_bird}(x),$$
$$\texttt{abnormal\_bird}(x) \leftarrow \texttt{bird}(x), \neg\texttt{fly}(x),$$

which imply that flying ability is only a default property of birds.

References

[1] O.Arieli. Paraconsistent Semantics for Extended Logic Programs. *Technical Report No. CW299,* Department of Computer Science, University of Leuven, 2000.

[13] Indeed, our results in the last two examples are in accordance with those of [19], and follow the same intuition.

[2] O.Arieli and A.Avron. Reasoning with logical bilattices. *Journal of Logic, Language, and Information* 5(1):25–63, 1996.

[3] O.Arieli and A.Avron. The value of the four values. *Artificial Intelligence* 102(1):97–141, 1998.

[4] A.Avron. Simple consequence relations. *Information and Computation* 92:105–139, 1991.

[5] C.Baral and M.Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming* 19–20:73–148, 1994.

[6] N.D.Belnap. A useful four-valued logic. *Modern Uses of Multiple-Valued Logic,* pp.7–37, editors: G.Epstein and J.M.Dunn, Reidel Publishing Company, 1977.

[7] N.D.Belnap. How computer should think. *Contemporary Aspects of Philosophy,* pp.30–56, editor: G.Ryle, Oriel Press, 1977.

[8] A.Bialynicki-Birula and H.Rasiowa. On the representation of quasi-boolean algebras. Bulletin of the Acad. Polonaise des Sciences Cl.III, Vol.V(3):259–261, 1957.

[9] W.A.Carnielli and J.Marcos. A Taxomony of C-Systems. preprint: CLE e-Prints 1(5), 2001. http://www.cle.unicamp.br/e-prints/articles

[10] K.L.Clark. Negation as failure. *Logic and Databases,* pp.293–322, editors: H.Gallaire and J.Minker, Plenum Press, 1978.

[11] N.C.A.da-Costa. On the theory of inconsistent formal systems. *Notre Damm Journal of Formal Logic* 15:497–510, 1974.

[12] C.V.Damásio and L.M.Pereira. Default negation in the heads: why not? *Proc. 5th International Workshop on Extensions of Logic Programming* (ELP'96), pp.103–118, editors: R.Dyckhoff et. al., Lecture Notes in Artificial Intelligence No.1050, Springer Verlag, 1996.

[13] A.J.García, G.R.Simari, and C.I.Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. *ECAI'98 Workshop on Practical Reasoning and Rationality,* 1998.

[14] M.Gelfond and V.Lifschitz. The stable model semantics for logic programming. *Proc. 5th Logic Programming Symposium,* pp.1070–1080, editors: R.Kowalski and K.Bowen, MIT Press, 1988.

[15] M.Gelfond and V.Lifschitz. Logic programs with classical negation. *Proc. 7th International Conference on Logic Programming* (ICLP'90), pp.579–597, editors: D.Warren and P.Szeredi, 1990.

[16] J.A.Kalman. Lattices with involution. *Transactions of the American Mathematical Society* 87:485–491, 1958.

[17] R.A.Kowalski and F.Sadri. Logic programs with exceptions. *Proc. 7th International Conference on Logic Programming* (ICLP'90), pp. 598–613, editors: D.Warren and P.Szeredi, 1990.

[18] J.W.Lloyd. *Foundations of logic programming.* Springer Verlag, 1987.

[19] L.M.Pereira and J.J.Alferes. Well-founded semantics for logic programs with explicit negation. *Proc. 10th European Conference on Artificial Intelligence* (ECAI'92), pp.102–106, editor: B.Neumann, 1992.

[20] L.M.Pereira, J.N.Aparício and J.J.Alferes. Nonmonotonic reasoning with well founded semantics. *Proc. 8th International Conference on Logic Programming* (ICLP'91), pp.475–489, editor: K.Furukawa, 1991.

[21] T.Przymusinski. Extended stable semantics for normal and disjunctive programs. *Proc. 7th International Conference on Logic Programming* (ICLP'90), pp.459–477, editors: D.Warren and P.Szeredi, 1990.

[22] R.Reiter. On closed word data bases. *Logic and Data Bases,* pp. 119–140, editors: H.Gallaire and J.Minker, Plenum Press, 1978.

[23] A.Tarski. Lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics* 5:285–309, 1955.

[24] M.van-Emden and R.A.Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM* 23(4):733–742, 1976.

[25] 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.