Well-Founded semantics for Semi-Normal Extended Logic Programs

Martin Caminada*
Utrecht University

Abstract

In this paper we present a new approach for applying well-founded semantics to extended logic programs. The main idea is not to fundamentally change the definition of well-founded semantics (as others have attempted) but rather to define a few restrictions on the content of the extended logic program, that make it possible to apply "traditional" well-founded semantics in a very straightforward way.

Introduction

Well-founded semantics (van Gelder, Ross, & Schlipf 1991) has originally been stated as an alternative for stable model semantics in normal logic programs. Due to its skeptical nature, it has sometimes been regarded as an easily computable lower bound for the more credulous stable model semantics. At the same time, well-founded semantics avoids some of the problems of stable model semantics, in which relatively small pieces of information (like a rule $\mathtt{a} \leftarrow \mathtt{not\ a}$) can cause the total absence of stable models.

With the emergence of extended logic programming (Gelfond & Lifschitz 1991), several researchers have attempted to apply well-founded semantics to extended logic programs (Sakama 1992; Brewka 1996). The introduction of strong negation, however, introduces additional problems not present in normal (non-extended) logic programming. In this paper, we approach the issue of how to apply well-founded semantics for extended logic programs not by giving another complex and advanced specification of what well-founded semantics for extended logic programs should look like, but instead we state a few restrictions on the *content* of the extended logic programs. We then show that under these restrictions, a relatively simple and straightforward definition of well-founded semantics yields a decent and unproblematic well-founded model.

Basic Definitions

A *program* considered in this paper is en *extended logic program* (ELP) (Gelfond & Lifschitz 1991) containing rules with weak as well as strong negation.

*This work has been supported by the EU ASPIC project.

definition 1. *An* extended logic program P *is a finite set of clauses of the form:*

$$\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m} (n \geq 0,\ m \geq 0)$$

where each $\mathtt{c}$, $\mathtt{a_i}$ *and* $\mathtt{b_j}$ *is a positive/negative literal and* not *stands for negation as failure. In the above rule,* $\mathtt{b_j} (1 \leq j \leq m)$ *is called a* weakly negated literal. *The literal* $\mathtt{c}$ *is called the* head *of the rule, and the conjunction* $\mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m}$ *is called the* body *of the rule. A rule is called* strict *iff it contains no weakly negated literals (that is, if* $m = 0$*), otherwise, the rule is* defeasible.

Notice that the head of a rule is never empty, although the body can be empty. If $\mathtt{l}$ is a literal, then we identify $\neg\neg\mathtt{l}$ with $\mathtt{l}$. If P is an extended logic program, then $strict(P)$ stands for the set of strict rules in P, and $defeasible(P)$ stands for the set of defeasible rules in P.

The *closure* of a set of strict rules consists of all literals that can be derived with it, as is stated in the following definition.

definition 2. *Let* S *be a set of strict rules. We define* $Cl(S)$ *as the smallest set of literals such that if* S *contains a rule* $\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$ *and* $\mathtt{a_1}, \ldots, \mathtt{a_n} \in Cl(S)$ *then* $\mathtt{c} \in Cl(S)$.

If S is a set of strict rules and L a set of literals, then we write $Cl(S \cup L)$ as an abbreviation of $Cl(S \cup \{\mathtt{l} \leftarrow \mid\ \mathtt{l} \in L\})$.

definition 3. *We say that a set of literals* L *is* consistent *iff* L *does not contain a literal* $\mathtt{l}$ *and its negation* $\neg\mathtt{l}$. *We say that a set of strict rules* S *is* consistent *iff* $Cl(S)$ *is consistent.*

The idea of P^L (the Gelfond-Lifschitz reduct of a logic program P under a set of literals L) is to remove each rule from P that is "defeated" by L (that is, to remove each rule containing a weakly negated literal in L) and then from the remaining rules to remove all remaining occurrences of weak negation.

definition 4. *Let* P *be an extended logic program and let* L *be a set of literals. We define* P^L *as* $\{\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n} \mid \mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m} \in P(n, m \geq 0)$ *and* $\neg\exists \mathtt{b_j} (1 \leq j \leq m) : \mathtt{b_j} \in L\}$.

Well-founded semantics (van Gelder, Ross, & Schlipf 1991) is a concept originally proposed for non-extended logic programs. As its original description is quite complex, we will use the following definition instead (inspired by (Brewka 1996)).

definition 5. *Let P be an extended logic program and L be a set of literals. We define $\gamma(L)$ (the standard stable operator) as $Cl(P^L)$. We define $\Gamma(L)$ as $\gamma(\gamma(L))$. The* well-founded model *of P is the smallest fixpoint of Γ.*

The Problem

Well-founded semantics (WFS) has been applied successfully in non-extended logic programs (Dix 1995a; 1995b). Applying WFS for extended logic programs, however, introduces the problem that the well-founded model is not guaranteed to be consistent. Consider the following example, taken from (Caminada & Amgoud 2005).

example 1.
"John wears something that looks like a wedding ring."
"John parties with his friends until late."
"Someone wearing a wedding ring is usually married."
"A party-animal is usually a bachelor."
"A married person, by its definition, has a spouse."
"A bachelor, by definition, does not have a spouse."
These sentences are represented by the program P:

$$\begin{array}{llll} \mathtt{r} & \leftarrow & & \\ \mathtt{m} & \leftarrow & \mathtt{r}, \mathtt{not}\, \neg \mathtt{m} & \\ \mathtt{hs} & \leftarrow & \mathtt{m} & \\ \mathtt{p} & \leftarrow & & \\ \mathtt{b} & \leftarrow & \mathtt{p}, \mathtt{not}\, \neg \mathtt{b} & \\ \neg \mathtt{hs} & \leftarrow & \mathtt{b}. & \end{array}$$

For example 1, applying the unaltered version of WFS yields a well-founded model of $\{\mathtt{r}, \mathtt{p}, \mathtt{m}, \mathtt{b}, \mathtt{hs}, \neg\mathtt{hs}\}$, which is inconsistent.

To cope with this problem, many approaches have been stated. Brewka, for instance, proposes to define the function $\Gamma(L)$ not as $\gamma(\gamma(L))$ but as $\gamma(Cn(\gamma(L)))$, where $Cn(L)$ is L if L is consistent, or Lit if L is not consistent (Brewka 1996). Another approach would be to apply paraconsistent reasoning, as for instance has been done in (Sakama 1992).

An alternative approach would be not to redefine the *semantics* of an ELP, but instead to state some additional conditions on the *content* of the extended logic program. The above example, for instance, would yield a perfectly acceptable outcome if the rules $\neg\mathtt{m} \leftarrow \neg\mathtt{hs}$ and $\neg\mathtt{b} \leftarrow \mathtt{hs}$ were added (which are essentially the contraposed versions of $\mathtt{hs} \leftarrow \mathtt{m}$ and $\neg\mathtt{hs} \leftarrow \mathtt{b}$). In that case, the well-founded model would be $\{\mathtt{r}, \mathtt{p}\}$. This approach would be quite similar to the work that Caminada and Amgoud have done in the field of formal argumentation, where similar difficulties occur (Caminada & Amgoud 2005).

Logic Programming as Argumentation

In this section, we will state some theory that allows us to link logic programming to formal argumentation. Using this theory, we will be able to apply the solution of (Caminada & Amgoud 2005) in the context of extended logic programming.

The first thing to do is to define the set of arguments and the defeat relation, given an (extended) logic program P. We choose a form of arguments that is different from (Dung 1995) and better suited to our purpose.

definition 6. *Let P be an extended logic program.*

- *An* argument *A based on P is a finite tree of rules from P such that each node (of the form $\mathtt{c} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_n, \mathtt{not}\, \mathtt{b}_1, \ldots, \mathtt{not}\, \mathtt{b}_m$ with $n \geq 0$ and $m \geq 0$) has exactly n children, each having a different head $\mathtt{a}_i \in \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$. The* conclusion *of A ($\mathtt{Conc}(A)$) is the head of its root.*
- *We say that an argument A_1* defeats *an argument A_2 iff A_1 has conclusion $\mathtt{c}$ and A_2 has a rule containing $\mathtt{not}\, \mathtt{c}$.*

We define $Arguments_P$ as the set of arguments that can be constructed using P, and $Defeat_P$ as the defeat relation under P. Let $Args \subseteq Arguments_P$. We define $\mathtt{Concs}(Args)$ as $\{\mathtt{Conc}(A) \mid A \in Args\}$.

We say that argument A is a *subargument* of argument B iff A is a subtree of B. We say that argument A is a *direct subargument* of argument B iff A is a subtree of B and there does not exist an argument C such that $C \neq A$, $C \neq B$, C is a subtree of B, and A is a subtree of C.

definition 7. *We say that:*

- *a set of arguments $Args$ is* conflict-free *iff $Args$ does not contain two arguments A and B such that A defeats B*
- *a set of arguments $Args$* defends *an argument A iff for each argument B that defeats A, $Args$ contains an argument C that defeats B.*

definition 8. *Let $Args$ be a set of arguments. We define $f(Args)$ as $\{A \mid Args$ does not contain an argument that defeats $A\}$ and $F(Args)$ as $f(f(Args))$.*

$F(Args)$ can be seen as the set of arguments that are defended by $Args$ (Dung 1995).

lemma 1. *Let P be an extended logic program and let E be the smallest fixpoint of F under P. E is conflict-free.*

Proof. As E is the smallest fixpoint of F under P, it holds that (Dung 1995) $E = \cup_{i=0}^{\infty} F^i(\emptyset)$. Suppose that E is not conflict-free. As F is a monotonic function and $F^0(\emptyset) = \emptyset$, there must be some smallest i ($i \geq 0$) such that $F^i(\emptyset)$ is conflict-free but $F^{i+1}(\emptyset)$ is not conflict-free. From definition 7 it then follows that $F^{i+1}(\emptyset)$ contains two arguments A and B such that A defeats B. The fact that A defeats B and $B \in F^{i+1}(\emptyset)$ means that there is an argument $C \in F^i(\emptyset)$ that defeats A. The fact that C defeats A and $A \in F^{i+1}(\emptyset)$ means that there is an argument $D \in F^i(\emptyset)$ that defeats C. But then $F^i(\emptyset)$ would not be conflict-free. Contradiction. □

The following property follows from definition 6 and 2.

property 1. *Let S be a set of strict rules and $\mathtt{l}$ be a literal. It holds that $\mathtt{l} \in Cl(S)$ iff there exists an argument A, based on S, such that $\mathtt{Conc}(A) = \mathtt{l}$.*

The following property follows from definition 4 and 6.

property 2. *Let P be an extended logic program and L be a set of literals. There exists an argument A, based on P^L, with $\mathtt{Conc}(A) = \mathtt{l}$ iff there exists an argument B, based on P, with $\mathtt{Conc}(B) = \mathtt{l}$, such that B does not contain a weakly negated literal $\mathtt{k} \in L$.*

The function γ is actually quite similar to the function f, as is stated in the following theorem.

theorem 1. *Let L be a set of literals and $Args$ be a set of arguments. If $L = \texttt{Concs}(Args)$ then $\gamma(L) = \texttt{Concs}(f(Args))$.*

Proof. We need to prove two things:

1. $\gamma(L) \subseteq \texttt{Concs}(f(Args))$
   Let $\texttt{l} \in \gamma(L)$. This, by definition 5, means that $\texttt{l} \in Cl(P^L)$. From property 1 it follows that there exists an argument (A), based on P^L, with $\texttt{Conc}(A) = \texttt{l}$. Then, according to property 2, there exists an argument (B), based on P, with $\texttt{Conc}(B) = \texttt{l}$, such that B does not contain a weakly negated literal $\texttt{k} \in L$. As $L = \texttt{Concs}(Args)$, the argument B is not defeated by $Args$. Therefore, $B \in f(Args)$. As B has conclusion $\texttt{l}$ it holds that $\texttt{l} \in \texttt{Concs}(f(Args))$
2. $\texttt{Concs}(f(Args)) \subseteq \gamma(L)$
   Let $\texttt{l} \in \texttt{Concs}(f(Args)) \subseteq \gamma(L)$. This means that $f(Args)$ contains some argument (say B) with conclusion $\texttt{l}$. That is, there exists an argument (B) with conclusion $\texttt{l}$ that is not defeated by $Args$. From property 2 it then follows that there exists an argument A, based on P^L (as $L = \texttt{Concs}(Args)$), with $\texttt{Conc}(A) = \texttt{l}$. This, by property 1, means that $\texttt{l} \in Cl(P^L)$, which by definition 5 means that $\texttt{l} \in \gamma(L)$.

□

The following theorem states that the well-founded model of a program P coincides with the conclusions of the grounded extension (Dung 1995) of the argument-interpretation of P.

theorem 2. *Let P be an extended logic program. The grounded extension GE of $\langle Arguments_P, Defeat_P \rangle$ coincides with the smallest fixpoint (WFM) of Γ. That is: $concs(GE) = WFM$.*

Proof. From theorem 1 it follows that, if $L = \texttt{Concs}(Args)$, then $\gamma(\gamma(L)) = \texttt{Concs}(f(f(Args)))$, so $\Gamma(L) = \texttt{Concs}(F(L))$. Therefore, the smallest fixpoint of Γ is equal to the conclusions of the smallest fixpoint of F, which is the grounded extension. □

Semi-Normal Extended Logic Programs

In this section, we define some restrictions on an extended logic program. An extended logic program that satisfies these restrictions is called a *semi-normal* extended logic program (a term inspired by semi-normal default theories). We then show that a semi-normal extended logic program avoids problems like illustrated in example 1 by always having a consistent well-founded model.

definition 9. *Let s_1 and s_2 be strict rules. We say that s_2 is a* transposition *of s_1 iff:*
$s_1 = \texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_n$ *and*
$s_2 = \neg\texttt{a}_i \leftarrow \texttt{a}_1, \ldots, \texttt{a}_{i-1}, \neg\texttt{c}, \texttt{a}_{i+1}, \ldots, \texttt{a}_n$ *for some* $1 \leq i \leq n$.

The intuition behind transposition can be illustrated by translating a strict rule $\texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_n$ to a material implication $\texttt{c} \subset \texttt{a}_1 \wedge \cdots \wedge \texttt{a}_n$. This implication is logically equivalent to $\neg\texttt{a}_i \subset \texttt{a}_1 \wedge \cdots \wedge \texttt{a}_{i-1} \wedge \neg\texttt{c} \wedge \texttt{a}_{i+1} \wedge \cdots \wedge \texttt{a}_n$, which is again translated to $\neg\texttt{a}_i \leftarrow \texttt{a}_1, \ldots, \texttt{a}_{i-1}, \neg\texttt{c}, \texttt{a}_{i+1}, \ldots, \texttt{a}_n$. Notice that, when $n = 1$, transposition coincides with classical contraposition.

definition 10. *A defeasible rule is* semi-normal *iff it is of the form*
$\texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_n, \texttt{not}\ \texttt{b}_1, \ldots, \texttt{not}\ \texttt{b}_m, \texttt{not}\ \neg\texttt{c}$.

definition 11. *An extended logic program P is called* semi-normal *iff:*

1. *$strict(P)$ is consistent,*
2. *$strict(P)$ is closed under transposition, and*
3. *$defeasible(P)$ consists of semi-normal rules only*

If A is an argument, then the *depth* of A is the number of nodes on the longest root-originated path in A. If A is an argument and r is a rule in A then the *depth of r in A* is the number of nodes on the shortest path from the root to a node labeled with r.

lemma 2. *Let P be a semi-normal extended logic program, Ass (the assumptions) be a nonempty set of strict rules with empty antecedents $\{\texttt{a}_1 \leftarrow, \ldots, \texttt{a}_n \leftarrow\}$ and A an argument with conclusion $\texttt{c}$ based on $strict(P) \cup Ass$, such that A contains an assumption $\texttt{a}_i \leftarrow$ ($1 \leq i \leq n$) that does not occur in P. There exists an argument B, based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$ such that B has a conclusion $\neg\texttt{a}_i$.*

Proof. We prove this by induction on the depth of A.

basis Let's assume that the depth of A is 1. In that case, A consists of a single rule, which must then have an empty antecedent. Therefore, the root of A must be $\texttt{c} \leftarrow$. It then follows that $\texttt{c} = \texttt{a}_i$. Therefore, there exists an argument (A itself) based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$ that has conclusion $\neg\texttt{a}_i$.

step Suppose the above lemma holds for all strict arguments of depth $\leq j$. We now prove that it also holds for all strict arguments of depth $j+1$. Let A be an argument of depth $j+1$, based on $strict(P) \cup Ass$, with conclusion $\texttt{c}$. Let $\texttt{c} \leftarrow \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_m)$ be the root of A. Let A_i be a direct subargument of A that contains the assumption $a_i \leftarrow$. Because the set of strict rules in P is closed under transposition, there exists a rule $\neg\texttt{Conc}(A_i) \leftarrow \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_{i-1}), \neg c, \texttt{Conc}(A_{i+1}), \ldots, \texttt{Conc}(A_m)$. The fact that A_i has a depth $\leq j$ means that we can apply the induction hypothesis. That is, there exists an argument (say B'), based on $strict(P) \cup Ass \cup \{\neg\texttt{Conc}(A_i) \leftarrow\}$, with conclusion $\neg\texttt{a}_i$. Now, in B', substitute $\neg\texttt{Conc}(A_i) \leftarrow$ by the subargument $\neg\texttt{Conc}(A_i) \leftarrow A_1, \ldots, A_{i-1}, \neg c, A_{i+1}, \ldots A_m$. The resulting argument (call it B) is a strict argument, based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$, with conclusion $\neg\texttt{a}_i$.

□

theorem 3. *Let $\langle Arguments_P, Defeat_P \rangle$ be an argumentation framework built from a semi-normal extended logic program P, and let E be the smallest fixpoint of F. It holds that $\texttt{Concs}(E)$ is consistent.*

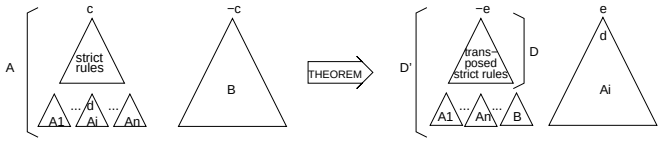

Figure 1: The working of theorem 3

Proof. Let E be the grounded extension of $\langle Arguments_P, Defeat_P \rangle$. Suppose the conclusions of E are not consistent. Then E contains an argument (say A) with conclusion $\mathtt{c}$ and an argument (say B) with conclusion $\neg\mathtt{c}$. As $strict(P)$ is consistent, at least one of these two arguments must contain a defeasible rule. Let us, without loss of generality, assume that A contains at least one defeasible rule. Let d be a defeasible rule in A that has minimal depth. Notice that the depth of d must be at least 1, for if d were the top-rule of A, then B would defeat A and E would not be conflict-free (which conflicts with lemma 1). It now holds that every rule in A with a smaller depth than d is a strict rule (see also figure 1). Let A_i be a subargument of A that has d as its top-rule. We will now prove that there exists an argument (D') in E that defeats A_i. Let $A_1, \ldots, A_n$ be the subarguments of A that are at the same level as A_i in A. Lemma 2 tells us that with the conclusions of $A_1, \ldots, A_n, B$ it is possible to construct an argument with a conclusion that is the opposite of the conclusion of A_i. Call this argument D. Now, let D' be equal to D, but with the assumptions $\mathtt{Conc}(A_1) \leftarrow, \ \ldots, \mathtt{Conc}(A_n) \leftarrow, \ \mathtt{Conc}(B) \leftarrow$ substituted by the underlying arguments $A_1, \ldots, A_n, B$. It holds that $D' \in E$ (this is because each defeater of D' is also a defeater of $A_1, \ldots, A_n, B \in E$, and the fact that E is a fixpoint of F means it defends itself against this defeater, which means that $D' \in E$). D', however, defeats A_i on d, so the fact that $D', A_i \in E$ means that E is not conflict-free, and (lemma 1) also no fixpoint of F. Contradiction. □

theorem 4. *Let P be a semi-normal extended logic program. The smallest fixpoint WFM (the well-founded model) of Γ is consistent.*

Proof. This follows directly from theorem 2 and theorem 3. □

Discussion

Many scholars in the field of defeasible reasoning distinguish two types of abstract rules: *strict rules* and *defeasible rules* (Pollock 1992; Nute 1994; Prakken & Sartor 1997; García & Simari 2004). A strict rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \rightarrow \mathtt{b}$ basically means that if $\mathtt{a_1}, \ldots, \mathtt{a_n}$ hold, then it is *without any possible exception* also the case that $\mathtt{b}$ holds. A defeasible rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \Rightarrow \mathtt{b}$ basically means that if $\mathtt{a_1}, \ldots, \mathtt{a_n}$ hold, then it is *usually* (or *normally*) the case that $\mathtt{b}$ holds.

One possible application of strict rules is to describe things that hold by definition (like ontologies). For instance, a cow is by definition a mammal and someone who is married by definition has a spouse. For this kind of rules, it appears that transposition is quite naturally applicable. If from $\mathtt{a_1}, \ldots, \mathtt{a_n}$ it follows without any possible exception that $\mathtt{b}$, then it also holds that from $\mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{b}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$ it follows without any possible exception that $\neg\mathtt{a_i}$.

In essence, one could say that the problems of example 1 are caused by the fact that two conclusions ($\mathtt{m}$ and $\mathtt{b}$) are conflicting (as $\mathtt{m}$ implies $\mathtt{hs}$, and $\mathtt{b}$ implies $\neg\mathtt{hs}$) but the standard entailment of ELP is too weak to discover this conflict. Transposition (for strict rules) can thus be seen as a way of strengthening the entailment, so that this kinds of hidden conflicts become explicit, and therefore manageable.

Some formalisms for defeasible reasoning, like (Pollock 1992; 1995), have strict rules that coincide with classical (propositional or first order) reasoning. That is, there exists a strict rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \rightarrow \mathtt{b}$ iff $\mathtt{a_1}, \ldots, \mathtt{a_n} \vdash \mathtt{b}$. In such a formalism, example 1 could be represented by the defeasible rules $\mathtt{r} \Rightarrow \mathtt{m}$ and $\mathtt{p} \Rightarrow \mathtt{b}$ and by the propositions $\mathtt{r}$, $\mathtt{p}$, $\mathtt{m} \supset \mathtt{hs}$ and $\mathtt{b} \supset \neg\mathtt{hs}$. Using these propositions one can then construct the strict rules $\mathtt{m}, (\mathtt{m} \supset \mathtt{hs}) \rightarrow \mathtt{hs}$ and $\mathtt{b}, (\mathtt{b} \supset \neg\mathtt{hs}) \rightarrow \neg\mathtt{hs}$, as well as the strict rules $\neg\mathtt{hs}, (\mathtt{m} \supset \mathtt{hs}) \rightarrow \neg\mathtt{m}$ and $\mathtt{hs}, (\mathtt{b} \supset \neg\mathtt{hs}) \rightarrow \neg\mathtt{b}$. These rules can be used not only to construct arguments for $\mathtt{m}$ and $\mathtt{b}$ but also to construct the much needed counterarguments deriving $\neg\mathtt{m}$ and $\neg\mathtt{b}$. By basing strict rules on classical entailment, Pollock is able to specify a formalism that avoids many of the difficult issues that have been plaguing the field of extended logic programming.

It is not difficult to see that transposition is a valid principle in classical logic (from $\mathtt{a_1}, \ldots, \mathtt{a_n} \vdash \mathtt{b}$ it follows that $\mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{b}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n} \vdash \neg\mathtt{a_i}$). In general, the set of strict rules generated by classical entailment satisfies many interesting properties. With transposition we have isolated the specific property of classical logic that is actually needed to avoid problems like illustrated by example 1. We simply apply the part of classical logic that we actually need, without having to go through the complexities of having to implement a full-blown classical logic theorem prover to generate the set of strict rules, as is for instance done in (Pollock 1995). The main cost of our approach is in generating the transpositions of the strict rules. For each strict rule, n transpositions are generated, where n is the number of literals in the body of the rule.

As for the defeasible rules, Pollock distinguishes two ways in which these can be argued against: rebutting and undercutting (Pollock 1992; 1995). Rebutting essentially means deriving the opposite consequent (head) of the rule, whereas undercutting basically means that there is some additional information under which the antecedent (body) of the rule is no longer a reason for the consequent (head) of the rule. For instance, suppose that we have the defeasible rule that an object that looks red usually is red. A rebutter would be that the object is not red, because it is known to be blue. An undercutter would be that the object is illuminated by a red light. This is not a reason for it *not* being red, but merely means that the fact that it looks red can no longer be regarded as a valid reason for it actually being red. Thus, rebutting attacks the consequent (head) of a rule, whereas undercutting attacks merely the connection between the antecedent (body) and the consequent (head) of a rule. Pollock claims, based on his philosophical work regarding episte-

mology, all forms of defeat can be reduced to rebutting and undercutting (Pollock 1992). This observation is important, as both of these forms of defeat can be modeled using semi-normal defeasible rules in extended logic programs.

Many problems in logic programming are caused by specific logic programs containing anomalous information (a rule like $\mathtt{a} \leftarrow \mathtt{not\ a}$ could for instance cause the absence of stable models). If one wants to apply standard and relatively straightforward semantics then one needs to make sure that a logic program does not contain such anomalies. If one provides anomalous input (like stating that a married person always has a spouse, without stating that someone who does not have a spouse is not married, using a formalism (ELP) that is not powerful enough to make this inference itself) then one should not be surprised that the outcome (the well-founded model) is anomalous as well. For reasons described above, we think that that the concept of semi-normal extended logic programs can serve as a quite natural and reasonable restriction of which programs can be regarded to be free of anomalies.

Quality Postulates

One way to evaluate the different approaches for providing a suitable semantics for ELP is by providing quality postulates (Caminada & Amgoud 2005). The idea is to state a number of general properties that should be satisfied by any formalism for defeasible reasoning, including ELP. In (Caminada & Amgoud 2005; ASPIC-consortium 2005) the following quality postulates have been stated:

- direct consistency. Let P be an extended logic program such that $strict(P)$ is consistent, and let M be a model of P (under some specified semantics). It must hold that M is consistent.
- closedness. Let P be an extended logic program and let M be a model under P (under some specified semantics). It must hold that $Cl(strict(P) \cup M) = M$.
- indirect consistency. Let P be an extended logic program such that $strict(P)$ is consistent, and let M be a model of P (under some specified semantics). It must hold that $Cl(strict(P) \cup M)$ is consistent.

The quality postulate of direct consistency is quite straightforward and is satisfied by most formalisms that we know of. The quality postulate of closedness basically states that, as far as the strict rules are concerned, the model is "complete". The quality postulate of indirect consistency does by itself not require that the model is closed under the strict rules, but instead requires the more modest property that if one would compute the closure of the model under the strict rules, the result would at least not contain any inconsistencies.

The above three quality postulates are not completely independent. Indirect consistency, for instance, implies direct consistency. Similarly, closedness and direct consistency imply indirect consistency.

To illustrate the value of the above three quality postulates, consider a person who knows a set of strict and defeasible rules, encodes these as a semi-normal extended logic program and then examines a model generated by an ELP inference engine. If the ELP inference engine would (in example 1) provide a model containing $\mathtt{m}$ but not containing $\mathtt{hs}$ (thus violating closedness) then the user may conclude that the ELP inference engine apparently "forgot" something. Worse yet, if the ELP inference engine provides a model containing $\mathtt{m}$ and $\mathtt{b}$ (thus violating indirect consistency) then the user may reason like: "My inference engine says that $\mathtt{m}$, and I know that from $\mathtt{m}$ it always follows that $\mathtt{hs}$, therefore $\mathtt{hs}$. My inference engine also says that $\mathtt{b}$ and I know that from $\mathtt{b}$ it always follows that $\neg\mathtt{hs}$, therefore $\neg\mathtt{hs}$." It is our view that, from an agent perspective, a formalism that does not satisfy indirect consistency cannot be used to generate the beliefs of an agent, as we think that an agent should never run into inconsistencies once it starts to do additional reasoning on its own beliefs.

Although ELP-models should ideally be closed under the strict rules of P, they should not necessarily be closed under the defeasible rules of P. If $\mathtt{a}$ is given and there exists a rule "if $\mathtt{a}$ then normally $\mathtt{b}$", then one cannot simply derive $\mathtt{b}$ since the situation may not be normal. The quality postulate of closedness is thus only relevant with respect to strict rules.

A fourth quality postulate that has, as far as we know, not been published earlier is that of crash-resistancy:

- crash-resistancy. There should not exist an extended logic program P, with $strict(P)$ consistent, such that for *any* extended logic program P', with $strict(P')$ consistent, that does not share any atoms with P, it holds that P has the same models (under some specific semantics) as $P \cup P'$.

Crash-resistancy basically states that it should not be possible for an extended logic program to contain some pieces of information (P) that makes other totally unrelated pieces of information (P') totally irrelevant when added.

The above four quality postulates are violated by various approaches that aim to provide extended logic programs with a suitable semantics. Indirect consistency, for instance, is problematic in approaches that are based on paraconsistent reasoning. When the approach of, for instance, (Sakama 1992) is applied to example 1, it produces a well-founded model $\langle\{\mathtt{r}, \mathtt{p}, \mathtt{m}, \mathtt{b}, \mathtt{hs}, \neg\mathtt{hs}\}, \{\neg\mathtt{r}, \neg\mathtt{p}, \neg\mathtt{m}, \neg\mathtt{b}\}\rangle$. Using Ginsberg's 7-valued default bilattice, this means that only $\mathtt{r}$, $\mathtt{p}$, $\mathtt{m}$ and $\mathtt{b}$ (but not $\mathtt{hs}$ or $\neg\mathtt{hs}$) are considered true, thus violating closedness and indirect consistency.

Brewka's approach to well-founded semantics (Brewka 1996), on the other hand, violates direct consistency as well as crash-resistancy. In example 1, $strict(P)$ is consistent, but Brewka's approach nevertheless yields the inconsistent set Lit, which violates direct consistency. As the outcome of Lit is obtained even when one adds syntactically totally unrelated rules to P, crash-resistancy is violated as well.

The quality postulate of crash-resistancy is violated by the stable model semantics of answer set programming, where a simple rule like $\mathtt{a} \leftarrow \mathtt{not\ a}$ yields no stable models at all, regardless of what additional (unrelated) information is contained in the logic program. A common opinion in the ELP-research community is that programs that have no stable models are by definition anomalous and unnatural. We

hereby would like to argue against this view. Consider a situation in where persons are usually believed in what they say, unless information of the contrary is available (rebut) or the person is known to be unreliable (undercut). Now consider the following three persons, who give the following statements:

- Bert: "Ernie is unreliable."
- Ernie: "Elmo is unreliable."
- Elmo: "Bert is unreliable."

This would correspond with the following extended logic program:

- bert_says_u_ernie $\leftarrow$
- u_ernie $\leftarrow$ bert_says_u_ernie, not $\neg$u_ernie, not u_bert
- ernie_says_u_elmo $\leftarrow$
- u_elmo $\leftarrow$ ernie_says_u_elmo, not $\neg$u_elmo, not u_ernie
- elmo_says_u_bert $\leftarrow$
- u_bert $\leftarrow$ elmo_says_u_bert, not $\neg$u_bert, not u_elmo

It is perfectly possible for a situation to occur in which three persons, sitting in a circle, claim their direct neighbour is unreliable. How this conflict should be dealt with is an issue open for discussion, but it should at least not cause the hearer to enter a state of total ignorence in which also all other entailment is completely blocked. It is our opinion, also for reasons described in (Dung 1995) that the problems of stable model semantics are very often caused by the nature of the semantics itself, and not by an "anomalous" extended logic program.

Summary and Conclusions

One of the advantages of the approach as sketched in the current paper is that it satisfies each of the quality postulates direct consistency, indirect consistency, closedness and crash-resistancy. Furthermore, it does so without the need of an advanced semantics that is complex and potentially difficult to understand. Although the approach only works for the somewhat restricted notion of semi-normal extended logic programs, we believe that these restrictions are in essence quite natural and can be given a decent philosophical justification.

References

ASPIC-consortium. 2005. Deliverable D2.5: Draft formal semantics for ASPIC system.

Brewka, G. 1996. Well-founded semantics for extended logic programs with dynamic preferences. *J. Artif. Intell. Res. (JAIR)* 4:19–36.

Caminada, M., and Amgoud, L. 2005. An axiomatic account of formal argumentation. In *Proceedings of the AAAI-2005*, 608–613.

Dix, J. 1995a. A classification theory of semantics of normal logic programs: I. strong properties. *Fundam. Inform.* 22(3):227–255.

Dix, J. 1995b. A classification theory of semantics of normal logic programs: Ii. weak properties. *Fundam. Inform.* 22(3):257–288.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77:321–357.

García, A., and Simari, G. 2004. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4):365–385.

Nute, D. 1994. Defeasible logic. In Gabbay, D.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford: Clarendon Press. 253–395.

Pollock, J. L. 1992. How to reason defeasibly. *Artificial Intelligence* 57:1–42.

Pollock, J. L. 1995. *Cognitive Carpentry. A Blueprint for How to Build a Person*. Cambridge, MA: MIT Press.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7:25–75.

Sakama, C. 1992. Extended well-founded semantics for paraconsistent logic programs. In *FGCS*, 592–599.

van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.