Logic Programming without Negation as Failure

Yannis Dimopoulos and Antonis Kakas
Department of Computer Science, University of Cyprus
75 Kallipoleos Str., CY-1678, Nicosia, Cyprus
{yannis, antonis}@turing.cs.ucy.ac.cy

Abstract

In this paper we study a new framework for normal and extended logic programming, called logic programming without negation as failure ($LPwNF$), which uses explicit negation but does not contain negation as failure (NAF) in its object-level language. The NAF principle is instead embodied at the metal-level in the argumentation based semantics of the framework.

A proof procedure for $LPwNF$ is presented whose form shows that this approach and the framework it provides for extended Logic Programming (ELP) remains within the basic backward reasoning computational paradigm of logic programming. We compare this computational property and the related issue of inconsistency and its possible resolution with other existing approaches for extended logic programming. We also show how $LPwNF$ can be used by providing a sound and complete translation of the A language for temporal reasoning. This translation provides an alternative equivalent understanding of the form of temporal reasoning carried out by the A language as a non-trivial interaction between abductive and default reasoning. It also gives a meaningful way of extending its semantics for some inconsistent domain descriptions.

1 Introduction

In a recent paper ([13]), a new framework for logic programming has been suggested, as a result of the argumentation-based approach ([2], [6], [11], [13]) for nonmonotonic reasoning. One of the main features of this framework, which we will call logic programming without negation as failure ($LPwNF$), is that it does not employ a NAF operator in the language but instead uses a limited form of classical (explicit) negation together with a (partial) ordering relation amongst the rules of the program. Its semantics, called the admissibility (or more generally the acceptability) semantics, is defined within an argumentation-based formalism and has been shown to be powerful enough to encompass and extend most of the earlier semantics for NAF in LP. The framework of $LPwNF$ retains, due to its argumentation semantics together with the ordering relation on the rules, the nonmonotonic feature of logic programming with NAF. In fact, in the case of normal logic programs it can be shown that these can be translated into a special type of programs in $LPwNF$.

For the case of extended logic programs (ELP) ([8], [15]) where we have explicit negation, we will show in this paper that $LPwNF$ offers an alternative to earlier approaches. One of the significant differences with these is the fact that it helps to avoid the possible ambiguity, at the representational level, that can arise with the existence of two types of negation. As a result, it offers a more direct representation of problems that is closer to their natural specification.

The purpose of this paper is to study this new $LPwNF$ framework primarily as an approach to ELP and present some of its main properties. In particular, we will examine its computational properties and show that it remains within the backward reasoning (resolution-based) computational paradigm of logic programming. This computational property together with other properties, such as the issue of inconsistency in an extended logic program and its resolution, will be compared with some other approaches for ELP. We also study an application of $LPwNF$ to temporal reasoning, by translating the A language ([9]) in this framework. This translation provides an alternative equivalent understanding of the reasoning carried out in consistent A language theories. Moreover, it offers a meaningful extension of the A language semantics for inconsistent theories.

2 The $LPwNF$ framework

In this section we present the $LPwNF$ framework and discuss some of its properties relevant to the rest of the paper. In the framework of $LPwNF$ logic programs are nonmonotonic theories where each logic program is viewed as pool of default sentences from which we must select a suitable subset, called extension, to reason with. Sentences in a logic program are written in the usual logic programming language with the addition of an explicit negation ([8]), but without the NAF operator. We will often refer to the sentences of a program as default rules or simply as rules. In addition these rules may be assigned locally a "relative strength" through a partial ordering relation. Hence, for example, we may have the following program

$$fly(x) \leftarrow bird(x)$$
$$\neg fly(x) \leftarrow penguin(x)$$
$$bird(x) \leftarrow penguin(x)$$
$$bird(Tweety)$$

with an ordering relation between the rules that assigns the second rule higher than the first. The above program captures the usual example of "flying birds" with its exceptions, without the use of NAF and abnormality predicates. We can conclude that Tweety flies because we can derive this from the first rule and there is no way to derive $\neg fly(Tweety)$ from the program. If we add the sentence $penguin(Tweety)$ then we can derive both $fly(Tweety)$ and $\neg fly(Tweety)$ from the rules of the program. But in its nonmonotonic semantics the second conclusion overrides the first since the

part of the program that derives $\neg fly(Tweety)$ contains the second rule which is designated higher than the first rule which belongs to the part of the program that derives the first conclusion $fly(Tweety)$. Note that if we want only to block the first rule without deriving the explicit negation of fly, then we would represent these rules using a new predicate symbol e.g. fly' in place of the predicate symbol fly, and add the extra rule $fly(x) \leftarrow fly'(x)$. In general, we can separate out a part of the program $P_0 \subset P$ (e.g. the last two rules of the program above) and consider this as a non-defeasible part that must be contained in any chosen subset of P. In this paper, unless otherwise stated, we assume that P_0 is empty.

This argumentation-based framework for non-monotonic Logic Programs is defined as follows.

Definition 2.1 (Background logic)
Formulae in the language $\mathcal{L}$ of the framework are defined as $L \leftarrow L_1, \ldots, L_n$, where $L, L_1, \ldots, L_n$ are positive or explicit negative literals. The only inference rule is the classical modus ponens rule

$$\frac{L \leftarrow L_1, \ldots, L_n \qquad\qquad L_1, \ldots, L_n}{L} \qquad (n \geq 0) \qquad \Box$$

We assume that, together with the set of sentences $\mathcal{T}$, we are given an ordering relation $<$ on these sentences (where $\phi < \psi$ means that ϕ has lower priority than ψ). The role of the priority relation is to encode locally the relative strength of rules in the theory, typically between contradictory rules. We will require that $<$ is irreflexive and antisymmetric.

Definition 2.2 (Logic Program or Non-Monotonic Theory)
A program $(\mathcal{T}, <)$ is a set of sentences $\mathcal{T}$ in $\mathcal{L}$ together with a priority relation $<$ on the sentences of $\mathcal{T}$.

We now proceed to define a notion of attack on these theories based on the possible conflicts that we can have in a theory $\mathcal{T}$ between a literal L and its explicit negation $\neg L$ and on the priority relation $<$ on $\mathcal{T}$.

Definition 2.3 (Attacks)
Let $(\mathcal{T}, <)$ be a program and $T, T' \subseteq \mathcal{T}$. Then T' attacks T iff there exist L, $T_1 \subseteq T'$ and $T_2 \subseteq T$ such that

(i) $T_1 \vdash_{min} L$ *and* $T_2 \vdash_{min} \neg L$

(ii) $(\exists r' \in T_1, r \in T_2 \text{ s.t. } r' < r) \Rightarrow (\exists r' \in T_1, r \in T_2 \text{ s.t. } r < r')$. $\Box$

$T \vdash_{min} L$ means that $T \vdash L$ under the background logic and that L can not be derived from any proper subset of T. Note that T_1 contains a unique rule r with head L. We then say that this attack T' is on the rule r on L. Using this we then define the basic notion of an admissible subset of a given program. A subset T of $\mathcal{T}$ is *closed* if it contains no rule whose conditions are not derived in T.

Definition 2.4 (Admissibility)
Let $(\mathcal{T}, <)$ be a program and T a closed subset of $\mathcal{T}$. Then T is admissible *iff T is consistent and for any $T' \subseteq \mathcal{T}$ if T' attacks T then T attacks T'.*

The credulous semantics of a logic program is given by its maximal (wrt set inclusion) admissible subsets. These are called *admissible extensions*. We can also define its skeptical semantics in terms of its admissible extensions.

Definition 2.5 *Let $(\mathcal{T}, <)$ be a program and L a ground literal. Then L is a* non-monotonic credulous (resp. skeptical) consequence *of the program iff L holds in a (resp. every) maximal admissible subset of T.*

Alternatively, the skeptical semantics can be defined in a form analogous to the well-founded model semantics by the least fixed point of the following recursive specification for a *strong form* of admissibility given by:
"$adm_{st}(\Delta)$ iff $\forall A$ s.t. A attacks Δ, $\neg adm_{st}(A)$".
It can be shown ([13]) that given a normal logic program P, with NAF in its object language we can define a corresponding equivalent theory $\mathcal{D}(P)$ in $LPwNF$. This transformation is motivated from the interpretation of *not p* as *unless p*. For example, if we have a rule "$p \leftarrow q, not\ r$" then this is transformed into two sentences "$p \leftarrow q$"and "$\neg p \leftarrow r$", and the second is assigned higher than the first.
Despite this equivalent transformation, it is sometimes natural to represent problems directly in the above $LPwNF$ framework. In this, we represent such exceptions to default rules closer to their natural specification, by assigning the exception rules higher than the default rules, rather than using some negative (NAF) condition that does not correspond to any negative information needed for the default to apply.

When explicit negation is present in extended logic programs the $LPwNF$ framework differs from other frameworks for ELP and can not easily be transformed into these. The comparison of $LPwNF$ with other frameworks for ELP will be studied in section 5.

3 A proof procedure for the framework

In this section we present a proof procedure for the $LPwNF$ framework. We will assume that the ordering relation will be restricted only between rules with contradictory heads (conclusions). The procedure consists of two kinds of interleaving derivations, called *Type A derivation* and *Type B derivation*, and constructs an admissible subset in which a given goal holds.
Every type A derivation generates part of the theory that we need. The outermost type A derivation generates a theory that is sufficient to derive some initial goal Q, whilst the others provide additional sentences to the theory in order to counterattack (defend) the attacks against the theory as it is constructed. These counterattacks are generated in type B derivations.

Within a type B derivation once an attack (or part of it) has been identified, a new type A derivation is initiated to find a counterattack to this.
Every type A derivation is essentially a SLD-resolution which also collects the rules that it uses. A type B derivation is given as input a rule r (with head l) which has been used in a type A derivation, and the purpose of the B derivation is to find and then block all possible proofs of the contrary of the conclusion of r, $\neg l$. In fact, the procedure only considers those proofs of $\neg l$ whose rule r' with $\neg l$ in the head is higher than the rule r that has initiated the type B derivation. A type B derivation operates on a *set of goals*, all of which are parts of proofs of the initial goal $\leftarrow \neg l$. At each step it considers all rules that can resolve with one of the current goals on a selected literal. If for such a rule r'' there is a type A derivation of the negation of the literal in the head of r'', then this way of proving $\neg l$ can be counterattacked. The rules used in this subsidiary A derivation are then added to the set that has been constructed so far. Hence, for the proofs that use such a rule r'', no further consideration is needed provided that the initial set of rules S grows with the rules of these subsidiary A derivations. Finally, if at some step of the B derivation the empty clause is derived, this signals a general failure and the original goal Q fails.

Example 3.1 *Consider the logic program P:*

$(r_1) : fly(x) \leftarrow bird(x)$ $\quad$ $(r_5) : bird(x) \leftarrow penguin(x)$
$(r_2) : \neg fly(x) \leftarrow penguin(x)$ $\quad$ $(r_6) : bird(T)$
$(r_3) : penguin(x) \leftarrow walkslikepeng(x)$ $\quad$ $(r_7) : walkslikepeng(T)$
$(r_4) : \neg penguin(x) \leftarrow \neg flatfeet(x)$ $\quad$ $(r_8) : \neg flatfeet(T)$

with the priorities $r_2 > r_1$, $r_4 > r_3$. □

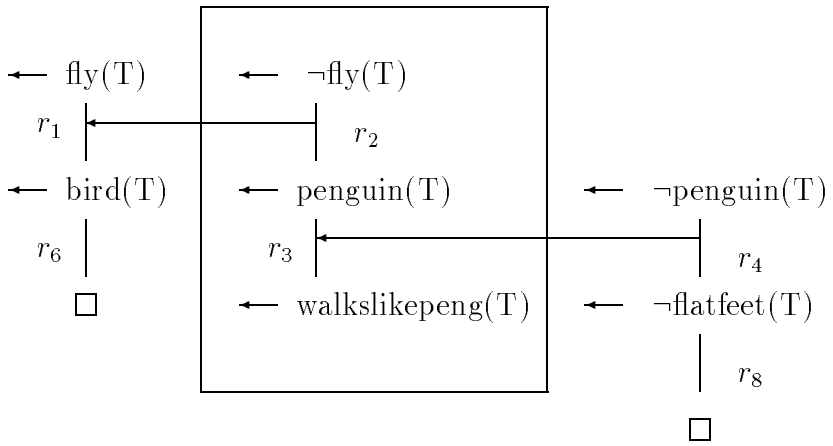

Figure 1

A proof for $fly(T)$ is shown in Figure 1, where the steps in the box belong to a type B derivation and the attacks are denoted by long arrows. Note that both the attacks are strong, in the sense that they contain a rule of higher priority than the rule in the set that they are attacking. Namely, $r_2 > r_1$ for

the first attack and $r_4 > r_3$ for the second.
Consider now a modified example where we remove the priority $r_4 > r_3$. For example if we replace r_4 with the rule $r_4' : \neg penguin(x) \leftarrow \neg singslikepeng(x)$, there is no reason why the two rules r_3 and r_4' should be ordered. Then with this replacement of r_4 by r_4' such that r_3 and r_4' are not ordered, and the replacement of r_8 with $singslikepeng(T)$, the above proof remains valid. This is because when generating a type A derivation form a type B, the top rule of the A derivation need not be higher than the generating rule in the B derivation, but only not smaller. Furthermore, after this change, a proof of $\neg fly(T)$ also exists. A formal description of the proof procedure follows.

Definition 3.2 *A* type A derivation *from* (G_1, R_1, r) *to* (G_n, R_n, r) *is a sequence*

$$(G_1, R_1, r), (G_2, R_2, r), \ldots, (G_n, R_n, r)$$

such that each R_i *is a set of rules,* r *is a rule, and each* G_i *has the form* $\leftarrow l, l'$*, where* l *is the literal selected by a selection rule* $S(G_i)$ *which selects literals the negation of which does not occur in the head of any rule in* R_i*. For* G_i*,* $i \geq 1$*, if there exists a rule* r_i *such that either*

1. $r_i \in R_i$ *and* r_i *resolves with* G_i *on* l*, or*
2. $i = 1$*,* r_i *is not lower than* r*,* r_i *resolves with* G_i *on* l*, and there is a type B derivation from* $(\{\leftarrow \neg l\}, R_i \cup \{r_i\}, r_i)$ *to* $(\{\}, R', r_i)$*, or*
3. $i \neq 1$*,* r_i *resolves with* G_i *on* l*, and there is a type B derivation from* $(\{\leftarrow \neg l\}, R_i \cup \{r_i\}, r_i)$ *to* $(\{\}, R', r_i)$

then $G_{i+1} = C$ *where* C *is the resolvent of* r_i *with* G_i *and for the first case* $R_{i+1} = R_i$ *while for the second and the third case* $R_{i+1} = R'$*.*

Definition 3.3 *A type B derivation from* (F_1, R_1, r) *to* (F_n, R_n, r) *is a sequence*

$$(F_1, R_1, r), (F_2, R_2, r), \ldots, (F_n, R_n, r)$$

where every F_i *is of the form* $F_i = \{\leftarrow l, l'\} \cup F_i'$ *with* F_i' *possibly empty,* l *a selected literal and* F_{i+1} *constructed from* F_i *as follows:*

1. *For* $i = 1$*,* $F_1 = \{\leftarrow l\}$*, let* P *be the set of pairs of the form* (l, r_i)*, where* r_i *is a rule which resolves with* $\leftarrow l$ *and* $r_i > r$*. Let* C *be the union of the resolvents of* $\leftarrow l$ *with the rules in* P*. If* $[] \notin C$ *then* $F_2 = C$ *and* $R_2 = R_1$*.*
2. *For* F_i*,* $i > 1$*, let* P *be the set of pairs of the form* (l, r_i)*, where* r_i *is a rule which resolves with* $\leftarrow l, l'$ *on* l*. Let* $P = P_1 \cup P_2$*, where* P_1 *is the sequence of pairs* $((l, r_1), (l, r_2), \ldots, (l, r_k))$ *such that for every* $1 \leq j \leq k$ *there is a type A derivation of from* $(\leftarrow \neg l, R_{r_j}, r_j)$ *to* $([], R_{r_{j+1}}, r_j)$*, with* $R_{r_1} = R_i$*. Let* C *be the union of the resolvents of the rules in the set* P_2 *with the rule* $\leftarrow l, l'$ *on* l*. If* $[] \notin C$ *then* $F_{i+1} = C \cup F_i'$ *and* $R_{i+1} = R_{r_{k+1}}$*.*

Note that the interleaving between the two types of derivations is done via contradictory rules. A transition from a type A derivation to a type B occurs only via (contradictory) rules which start the type B derivation that are of higher priority than the rules constructed by the type A derivation, whilst for a transition from a type B to a type A it is sufficient that the contradictory rule that start the type A derivation is not of lower priority. It is important to note that the structure of the computation is basically the same as that for normal logic programming despite the presence of explicit negation in the framework. Type B derivations can be seen analogous to subsidiary NAF derivations. The general structure of the computation is then the same as that of the backward reasoning SLDNF procedure.

We now prove the soundness of the procedure wrt the admissibility semantics. We will use the notion of *strong attack* of R on a rule $r \in R$ with $\text{head}(r) = l$. This is an attack that proves $\neg l$ whose rule for $\neg l$ is higher than r. Note that the set of attacks involved in the above procedure is restricted exactly to the set of strong attacks.

Proposition 3.4 *Let D be a successful type A derivation from $(Q, \{\}, r)$ to $([], R_n, r)$. Then R_n attacks every strong attack against R_n.*

Proof (sketch): Let K be a strong attack for R_n on a literal l and a rule $r \in R_n$. Let us represent K by $K = (r_1, \ldots, r_k)$. At the point where the rule r has been added to the set R_n, the procedure will start a type B derivation from $(\{\leftarrow \neg l\}, R, r)$ to $(\{\}, R', r)$. This means that for every possible SLD derivation which is a strong attack, and therefore for K, there will be a set of rules in R_n, denoted by $D_i = (d_1, \ldots, d_m)$, that will SLD-entail $\neg\text{head}(r_i)$, for some $1 \leq i < k$ and $r_i \in K$. The rules in D_i are exactly the rules used in the type A derivation $(G_1, A_1, m), (G_2, A_2, m), \ldots, (G_n, A_n, m)$ from $(\leftarrow \neg\text{head}(r_i), M, m)$ to $([], M', m)$. There are three possibilities for the relative priorities of the rules in K and D_i.
1) There exists a rule $d \in D_i$ greater than some rule in K.
2) There is no rule in K higher than some rule in D_i.
3) There is no rule in D_i with priority higher than some rule in K, and furthermore there is a rule r_j, $j > i$ used in K that has higher priority than some rule $\alpha \in D_i$.
In the first two cases D_i is an attack for K. If the third is the case, at the point where the rule α is used in the associated to D_i type A derivation, a type B derivation starts, from $(\{\leftarrow \neg\text{head}(r_j)\}, C, \alpha)$ to $(\{\}, C', \alpha)$. Notice that since $r_j > \alpha$, the rule r_j will be included in the possible derivations of $\neg\text{head}(r_j)$ encountered in the new B derivation.
Iterating the above arguments we obtain a strictly increasing subsequence of rules $(a_0, a_1, \ldots)$, from the sequence $(r_1, \ldots, r_k)$ of rules in K, where $a_0 = r_1 = \neg l$, and $a_1 = r_j$. Since K is a finite SLD derivation either one of the cases 1) or 2) must hold for some rule r_i in K, $i < k$, or the last rule in K, r_k has to be considered. If the first is the case we see that D_i and so R_n attacks K. In the second case, r_k is higher than some rule in R_n, and so

a type B derivation would start which at the first step leads to the empty clause, therefore the overall computation D fails, contradiction.

Proposition 3.5 *Let D be a successful type A derivation from $(Q, \{\}, r)$ to $([], R_n, r)$. Then R_n attacks every attack against R_n.*

Proof (sketch): Let $K = (S_1, r_1, S_2, r_2, \ldots, S_n)$ be an attack for R_n on a literal l and a rule r. The rule r is added to R_n during some type A derivation D_i. If there is a rule in D_i higher than a rule in K then D_i attacks K. On the other hand if there is no rule in K higher than a rule in D_i, then again D_i attacks K. Assume now that there is a pair of rules, $r_j \in K$ and $d_j \in D_i$ such that $r_j > d_j$ and there is no rule in D_i higher than some rule in K. Then we note that $K' = (\text{head}(r_j) = S_j, r_j, S_{j+1}, r_{j+1}, \ldots, S_n)$, $K' \subseteq K$, is a strong attack against R_n. By proposition 3.4 R_n attacks K', therefore attacks K. □

Theorem 3.6 *Let D be a successful type A derivation from $(Q, \{\}, r)$ to $([], R_n, r)$. Then R_n is an admissible set such that Q follows from R_n.*

Proof (sketch): It is easy to see that Q follows from the set of rules in R_n. By proposition 3.5 R_n attacks every attack, and by construction is consistent. Therefore, R_n is an admissible set. □

The above proof procedure can be easily modified so that successful A derivations imply that the initial goal holds in the skeptical admissibility semantics given by the strong form of admissibility. In this case the strong admissibility of a set must always reduce to its base case, namely that a set has no attack. The modifications required to the proof procedure are:
1. In type A (a) omit the first case where a rule r_i already in R_i can be used again with no further tests, and (b) in the second case when $i = 1$, r_1 must be higher than the rule r.
2. In type B in case (1) we must consider all rules r_i that resolve with l, such that r_i is not lower than r.

4 An application to temporal reasoning

In this section we will study the A language, introduced in [9], by showing how theories written in this language can be translated into $LPwNF$. Other translations of the A language to logic programming have been already defined in [3], [5]. Our translation is related to that given in [3].
Let us first briefly review the A language. The A language is a syntactically simple language for representing actions. It presupposes two disjoint set of symbols, the *fluent names* and the *action names*. A *fluent expression* is a fluent name possibly preceded by the symbol $\neg$. A theory or *domain description* D, can contain two types of *propositions*, the *v-propositions* and the *e-propositions*. A v-proposition is an expression of the form F **after** $A_1; \ldots; A_n$, where F is a fluent expression, and $A_1 \ldots A_n$ are action names. If $m = 0$ then we write **initially** F. An e-proposition is an expression of the

form A **causes** F if $P_1, \ldots, P_n$, where A is an action name and $F, P_1, \ldots, P_n$ are fluent expressions.
A state is defined to be a set of fluents. Then the semantics of the the A language is defined through the models of D, each model being a structure $S = (\sigma_0, \Phi)$, where σ_0 is the initial state and Φ the transition function on the states (see [9]). A domain description is called *e-consistent* (the concept was introduced in [3]) if the set of e-propositions of D is consistent. It has been proved in [3] that a domain description D is e-consistent iff for every pair of rules A **causes** F if $P_1, \ldots, P_n$ and A **causes** $\neg F$ if $P'_1, \ldots, P'_m$ in D, there exists an i and j such that P_i is the complement of P'_j.
Let us now translate the A language into the framework of $LPwNF$.

Definition 4.1 *Let D be domain description. If we denote by $t(D) = (P, >)$ the associated program, then P contains the rules*

- *For every v-proposition* **Initially** F *the rule* $Holds(F, S_0) \leftarrow$.
- *For every fluent F such that neither* **Initially** F *nor* **Initially** $\neg F$ *belongs to D the rules* $Holds(F, S_0) \leftarrow$ *and* $\neg Holds(F, S_0) \leftarrow$
- *For every e-proposition* A **causes** F **if** $P_1, P_2, \ldots P_n$ *(resp.* A **causes** $\neg$ F **if** $P_1, P_2, \ldots P_n$*) the rule* $Holds(F, Result(A, s)) \leftarrow Holds(P_1, s), Holds(P_2, s), \ldots, Holds(P_n, s)$ *(resp.* $\neg Holds(F, Result(A, s)) \leftarrow Holds(P_1, s), Holds(P_2, s), \ldots, Holds(P_n, s)$*).*
- *The rules representing the frame axioms, namely,* $Holds(f, Result(a, s)) \leftarrow Holds(f, s)$ *and* $\neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s)$.[1]

The relation $>$ gives higher priority to any rule r resulting from an e-proposition, than to the corresponding ground rules of the frame axiom whose head is the negation of that r, for the same fluent, action and state.

We now define a notion of temporal admissible extension for $t(D)$.

Definition 4.2 *Let $G(D)$ denote the set of literals* $Holds(F, Result(A_n, \ldots Re\text{-}sult(A_1, S_0)\ldots)$ *(resp.* $\neg Holds(F, Result(A_n, \ldots Result(A_1, S_0)\ldots))$ *where* F **after** $A_1, A_2, \ldots A_n$ *(resp.* $\neg$ F **after** $A_1, A_2, \ldots A_n$*) is a v-proposition in D, with $n \geq 0$. We say that a set of rules E is a* **admissible temporal extension** *of $t(D)$ iff E is a (maximal) admissible extension that entails every literal of $G(D)$. Furthermore, a literal $Holds(F, S)$ is* **entailed** *by $t(D)$ if it is true in every admissible temporal extension of $t(D)$.*

We note here that the inclusion in $t(D)$ of the sentence $Holds(F, S_0)$ and $\neg Holds(F, S_0)$ for any fluent F which is not known initially in D will have the effect of choosing one of these two sentences in the (maximal) admissible extensions of the translated theory. Alternatively, we can separate these sentences out as a set of abducible hypotheses that can be added to the theory when constructing a (maximal) admissible extension of the rest of the translated theory $t(D)$. Note also that we get the same result by including

[1] Small letters such as f, a, s denote variables. As usual rules with variables represent the set of their ground instances over the Herbrand universe.

in the abducibles all sentences of the form $Holds(F, S_0)$ and $\neg Holds(F, S_0)$ irrespective if F is known in D or not and require that the set $G(D)$ contains corresponding sentences for all the v-propositions in D, including the ones which refer to the initial state.

We say that a model $M = (\sigma_0, \Phi)$ of D is *equivalent* to an admissible extension E of $t(D)$ if $F \in \Phi(A_n, \ldots, \Phi(A_1, \sigma_0)\ldots)$ iff $Holds(F, Result(A_n, \ldots, Result(A_1, S_0)\ldots) \in E$ and $F \notin \Phi(A_n, \ldots, \Phi(A_1, \sigma_0)\ldots)$ iff $\neg Holds(F, Result(A_n, \ldots, Result\ (A_1, S_0)\ldots) \in E$. Then, the following theorem implies that the admissible temporal extensions of the translation $t(D)$ of an e-consistent domain description D are in direct correspondence with the models of D. Due to lack of space the proofs in this section will be omitted and can be found in an extended version of the paper.

Theorem 4.3 *Let D be a domain description and $t(D)$ its translation. Every model M of D induces an admissible temporal extension E for $t(D)$ such that E and M are equivalent. Conversely, if D is e-consistent then every admissible temporal extension E of $t(D)$ induces a model M for the domain description D such that M and E are equivalent.*

Let us illustrate this result by the following examples.

Example 4.4 *The Yale Shooting Problem, with the description D_0*

Initially alive	shoot **causes** $\neg alive$ **if** loaded
Initially $\neg loaded$	shoot **causes** $\neg loaded$
	load **causes** loaded

is translated to $t(D_0)$

$(r_1) : Holds(alive, S_0) \leftarrow$
$(r_2) : \neg Holds(loaded, S_0) \leftarrow$
$(r_3) : Holds(loaded, Result(load, s)) \leftarrow$
$(r_4) : \neg Holds(alive, Result(shoot, s)) \leftarrow Holds(loaded, s)$
$(r_5) : \neg Holds(loaded, Result(shoot, s)) \leftarrow$
$(r_6) : Holds(f, Result(A, s)) \leftarrow Holds(f, s)$
$(r_7) : \neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s)$

with the priority relation given by $\{r_3 > r_7, r_4 > r_6, r_5 > r_6\}$.

Then $G_0 = \{Holds(alive, S_0) \leftarrow, \neg Holds(loaded, S_0) \leftarrow\}$ and there is one admissible temporal extension that entails the literal $\neg Holds(alive, Result(shoot, Result(load, s_0)))$.

Example 4.5 *The Murder Mystery problem with description D_1*

Initially alive	load **causes** loaded
$\neg alive$ **after** shoot;wait	shoot **causes** $\neg alive$ **if** loaded
	shoot **causes** $\neg loaded$

Then $t(D_1)$ is

$(r_1) : Holds(alive, S_0) \leftarrow$ $\quad (r_3) : \neg Holds(loaded, S_0) \leftarrow$
$(r_2) : Holds(loaded, S_0) \leftarrow$ $\quad (r_4) : Holds(loaded, Result(load, s)) \leftarrow$
$(r_5) : \neg Holds(alive, Result(shoot, s)) \leftarrow Holds(loaded, s)$
$(r_6) : \neg Holds(loaded, Result(shoot, s)) \leftarrow$

$(r_7) : Holds(f, Result(a, s)) \leftarrow Holds(f, s)$
$(r_8) : \neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s)$
the priority relation is defined by $\{r_4 > r_8, r_5 > r_7, r_6 > r_7\}$.

There is only one admissible extension that contains $\neg Holds(alive, Result(shoot, Result(wait, S_0))$. This is also the only admissible temporal extension. Moreover $Holds(loaded, S_0)$ is true in this extension, therefore we conclude that the gun was initially loaded.

Hence, we see that we have a simple and natural translation of the A language (with $t(D)$ a syntactic variant of D) which provides a faithful representation in the computational framework of logic programming. This translation offers an alternative understanding of the temporal reasoning carried out in the A language. Let us separate out the initial predicate and regard its positive and negative ground instances as abducibles. Then the temporal reasoning of the A language can be understood via abduction to explain the given facts of the v-propositions where these explanations are defined wrt the nonmonotonic logic that captures the default reasoning with the frame axiom. In other words, the temporal reasoning can be seen as a non-trivial interaction between abduction and default reasoning.
It is interesting to note the inferences that $t(D)$ sanctions when D is e-inconsistent. Consider the following example.

Example 4.6 *Let D be*

Initially P_1	A **causes** F **if** P_1
Initially P_2	A **causes** $\neg F$ **if** P_2

and $t(D)$ its translation, consisting of frame axioms and the rules corresponding to the e-propositions

$(r_1) : Holds(P_1, S_0) \leftarrow$ $\quad$ $(r_2) : Holds(P_2, S_0) \leftarrow$
$(r_3) : Holds(F, S_0) \leftarrow$ $\quad$ $(r_4) : \neg Holds(F, S_0) \leftarrow$
$(r_5) : Holds(F, Result(A, s)) \leftarrow Holds(P_1, s)$
$(r_6) : \neg Holds(F, Result(A, s)) \leftarrow Holds(P_2, s)$

This program $t(D)$ has an infinite number of extensions, corresponding to the possible combinations of the rules $Holds(F, Result(A, s)) \leftarrow Holds(P_1, s)$ and $Holds(F, Result(A, s')) \leftarrow Holds(P_1, s')$ for any state s, s'. For example, $E_1 = \{Holds(P_1, S_0), Holds(P_2, S_0), Holds(F, S_0), Holds(F, Result(A, S_0)) \leftarrow Holds\ (P_1, S_0), \ldots\}$, and $E_2 = \{Holds(P_1, S_0), Holds(P_2, S_0), Holds(F, S_0), \neg Holds(F, Result(A, S_0)) \leftarrow Holds(P_2, S_0), \ldots\}$.

The non-determinism (in the sense of extension splitting) due to e-inconsistency indicated by the previous example, is a general feature of the $t(D)$ translation method. This can be stated formally as follows.

Proposition 4.7 *Let D be a domain description and D' the domain description that is identical to D but contains no v-propositions. Then D is e-inconsistent iff there is a complete initial state I such that $t(D') \cup t(I)$ has more than one admissible extensions.*

5 Comparison with other ELP approaches

In this section we will discuss some properties of the framework of $LPwNF$ presented above and compare it with other frameworks for ELP (e.g. [8], [14], [18], [16], [1], [7], [10], [19]). Together with the obvious difference in the existence or not of NAF in the object language another important difference is the fact that in $LPwNF$ a program is viewed as set of rules all of which (except a subset that might be separated out) are defeasible. Hence whereas before NAF was the defeasible part of the program now the rules themselves are defeasible. The significance of this difference stems from the fact that it allows different ways to resolve inconsistencies that arise due to the presence of explicit negation in the program.

This problem of inconsistency in ELP has been addressed in different ways. For example, in [7], [16], [1] and similarly in [18] for the well-founded semantics applied to ELP, consistency is restored by rejecting, whenever possible, the hypotheses of NAF literals. Hence in the program $\{p \leftarrow \text{not} q, \neg p \leftarrow \}$ we will reject the hypotheses "not$q$" by recognising that $P \cup \{ \text{not} q\}$ becomes inconsistent and putting the onus of the inconsistency on the sentence "not$q$" rather than on any sentence of P. On the other hand, in [14] consistency is restored for any program by choosing always to reject the positive atom over the explicit negative one when they can be both derived.

The $LPwNF$ framework with its defeasibility of rules, adopts a more neutral and flexible way of resolving inconsistency by allowing both possibilities of choosing either separate part of the program that derive the contradictory literals p and $\neg p$ respectively. An example of this was seen in the previous section where it allowed us to understand inconsistent domains in the A language (see proposition 4.7). To illustrate this further consider the following example:

$(r_1) : fly(x) \leftarrow bird(x)$ $\quad$ $(r_4) : \neg happy(x) \leftarrow hungry(x)$
$(r_2) : \neg fly(x) \leftarrow penguin(x)$ $\quad$ $(r_5) : bird(T)$
$(r_3) : happy(x) \leftarrow fly(x)$ $\quad$ $(r_6) : hungry(T)$

where the second rule is designated higher than the first but the two contradictory (default) rules for happy are not related by the ordering relation. Here we have two maximal admissible extensions one, containing the rule r_3 in which we can derive $\{bird(T), hungry(T), fly(T), happy(T)\}$ and another, containing the rule r_4 in which we can derive $\{bird(T), hungry(T), fly(T), \neg happy(T)\}$. Note that in both extensions we have $fly(T)$. Let us contrast this with an extended logic program with object-level NAF for this example where the first two rules with their link are replaced by the rules $\{fly(x) \leftarrow bird(x), \text{not } abnormalbird(x), abnormalbird(x) \leftarrow penguin(x)\}$.

In this case the hypothesis "not $abnormalbird(T)$" although is not directly contradicted it can not be accepted as it would lead to an inconsistency. In other words, we do not allow for the possibility that this inconsistency may be due to the default rule $\neg happy(x) \leftarrow hungry(x)$. In [14] "not $abnormalbird(T)$" will be accepted since this approach chooses a-priori

to reject the possibility that T may be happy as a consequence of $fly(T)$ when hungry holds. We thus see that these ELP semantics force a particular choice amongst the two rules for the relation "happy". Moreover, note when the last fact $hungry(T)$ is not present we can conclude $fly(T)$. Then the introduction of this fact, which is not directly related to the (default) rule for "fly" causes the loss of this conclusion. Note that some of these difficulties can be overcomed in ELP by adding the extra condition "not $\neg happy(x)$" in the body of r_3 and the condition "not $happy(x)$" in the body of r_4, as suggested by the methodology of [14]. These extended logic programs can then been seen as a way of implementing the higher level $LPwNF$ theories.

A similar behaviour can also be seen in the following program:
$drivinglicence(x) \leftarrow \text{not} convicted(x)$
$gunlicence(x) \leftarrow \text{not } convicted(x)$
where we can derive that John has both a driving and a gun licence. If later we learn that $\neg gunlicence(John)$ then this will result in John also loosing his driving licence as the hypotheses "not $convicted(John)$" is no longer consistent. This behaviour arises from the fact that the use of the object-level NAF to express exceptions to default rules identifies the exceptions between two rules and thus relates unnecessarily the applicability of one rule with that of the other.

Another approach which shares with $LPwNF$ the property that some of the rules of the program can be considered to be defeasible and thus can be rejected to avoid inconsistency is that of [10]. This is done by separating the program into two parts, a strict and a defeasible part. This separation though can only be used to express one level of exception. Thus in [10] we must retain NAF at the object level if we want more levels of exceptions and hence the discussion above also applies to this framework.

These differences in the treatment of inconsistency between $LPwNF$ and the other approaches described above has an important consequence in the computational model of the frameworks. The consistency requirement imposed by many approaches on NAF hypotheses of the form "not q" forces the need for reasoning forward from such hypotheses in the whole program to check that no inconsistency arises. On the other hand, as we saw in section 3 in the $LPwNF$ framework it is possible to develop a computational procedure that does not need to reason forward. Thus the $LPwNF$ framework remains within the same basic backward reasoning computational paradigm of normal logic programming when explicit negation is introduced in the framework. We point out though that if we separate a part of the program as a strict given part then we may also need to have a limited form of forward reasoning with this part of the program.

Proof procedures for ELP have also been developed in [19], [1]. In [19], extended logic programs are transformed into equivalent normal logic programs under a suitably chosen semantics for ELP. Then we can compute with the usual normal LP procedures. Like in our case this chosen semantics for ELP does not impose a global consistency requirement but is based

on the notion of conservative derivability in [20]. It is a skeptical form of semantics that under inconsistency, as in the discussion above, does not allow any of the possible choices. Also in the framework of [14], it is possible to transform extended logic programs into equivalent normal logic programs and thus compute in the usual way for normal logic programming. Another method for computing a skeptical semantics for ELP, namely the WFSX semantics of [16] which is an extension of the well-founded model semantics ([18]) for ELP, has been given in [1]. This is though applied mainly to non-contradictory programs. To deal more fully with inconsistencies the authors propose to use other methods for contradiction removal in [17].

6 Conclusions

We have studied the framework of $LPwNF$ as a new framework for LP and have compared its properties with other approaches to normal and extended LP. This framework, which is based on an argumentation semantics, does not contain negation as failure in its object level language but uses only explicit negation, thus removing any possible ambiguity in the use of two types o negation. A proof procedure has been developed for this framework whose form shows that the $LPwNF$ framework for ELP remains within the basic backward reasoning computational paradigm of LP with no (or limited) forward reasoning. This property is related to the assumption that the rules in a logic program are defeasible and a "liberal" view of contradiction resolution through non-determinism in the number of possible extensions. More work is needed to understand further the significance of this property and more generally the relationship of $LPwNF$ with other approaches to ELP. In particular, it is interesting to investigate how some classes of $LPwNF$ programs can be implemented by translations into ELP programs exploiting the possibility of using conditions of the form "not $\neg p$".

We have also used the framework of $LPwNF$ to study the A language ([9]) framework for temporal reasoning by providing a faithful translation in the computational framework of logic programming and have seen how this can enhance our understanding of the A language. The temporal reasoning of the A language can be seen as a combination of abductive and default reasoning. This translation can also be used to give a meaningful extension of the A language semantics for e-inconsistent domain descriptions.

Another application of the $LPwNF$ framework in the area of Machine Learning has been studied in [4], where it has been used as a suitable representation framework for extending inductive logic programming to learn nonmonotonic programs. In the future we will study other applications of the framework in legal reasoning and reasoning in multi-agent systems.

Further study is also needed to understand the connections of the $LPwNF$ framework and its argumentation-based semantics with other approaches of nonmonotonic reasoning that also use some notion of ordering or preference. It is also interesting to study the possibility, offered by the *explicit* (local) representation of the ordering amongst the rules in $LPwNF$, of reasoning

about this ordering at an extra level over and above that of admissibility.

Acknowledgements The authors thank P. Mancarela, R. A. Kowalski, F. Toni for many useful discussions. This work was partly supported by the ESPRIT BRA project Compulog 2 no 6810.

References

[1] J. J. Alferes, C. V. Damasio and L. M. Pereira. Top-down query evaluation for well-founded semantics with explicit negation. *Proc. ECAI'94*, Amsterdam, 1994.

[2] A. Bondarenko, F. Toni and R.A. Kowalski. An assumption-based Framework for Non-monotonic Reasoning. *Proc. 2nd Int. Workshop on Logic Programming and NonMonotonic Reasoning*, Lisbon, 1993.

[3] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. *Proc. ILPS'93*, Vancouver, 1993.

[4] Y. Dimopoulos and A. Kakas. Learning Non-Monotonic Logic Programs: Learning Exceptions. *Machine Learning: ECML-95* , (Proc. European Conf. on Machine Learning), LNCS 914, Springer Verlag.

[5] P. M. Dung. Representing actions in logic programming and its applications in database updates. *Proc. ICLP'93*, 1993.

[6] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. *Proc. IJCAI'93*.

[7] P. M. Dung. An argumentation semantics for logic programming with explicit negation. *Proc. ICLP'93*, Budapest, 1993.

[8] M. Gelfond and V. Lifschitz. Logic programs with classical negation. *Proc. ICLP'90*, Jerusalem, 1990.

[9] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. *Proc. ICSLP'92*, 1992.

[10] K. Inoue. Extended logic programs with default assumptions. *Proc. ICLP'91*, Paris, 1991.

[11] A.C. Kakas. Default Reasoning via Negation as Failure. In *The theoretical foundations of Knowledge Representation and Reasoning*, LNAI 810, Springer Verlag, 1994.

[12] A.C. Kakas, R.A. Kowalski and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719-770, 1993.

[13] A. C. Kakas, P. Mancarella and P.M. Dung. The Acceptability Semantics for Logic Programs. *Proc. ICLP'94*, Santa Margherita, Italy, 1994.

[14] R.A. Kowalski and F. Sadri. Logic Programs with Exceptions. *Proc. ICLP'90*, Jerusalem, 1990.

[15] D. Pearce and G. Wagner. Reasoning with Negative Information I: Strong Negation in Logic Programs. In *Language, Knowledge and Intensionality*, L. Haaparanta, M. Kusch and I. Niiniluoto (eds), 1990.

[16] L.M. Pereira and J.J. Alferes. Well-founded semantics for logic programs with explicit negation. *Proc. ECAI'92*, Vienna, 1992.

[17] L.M. Pereira, J.J. Alferes and J.N. Aparicio. Contradiction removal with explicit negation. *Proc. Applied Logic Conference*, Amsterdam, 1992.

[18] T.C. Przymusinski. Extended stable semantics for normal and disjunctive programs. *Proc. ICLP'90*, Jerusalem, 1990.

[19] F. Teusink. A proof procedure for extended logic programs. *Proc. ILPS'93*.

[20] G. Wagner. Ex contradictione nihil sequitur. *Proc. IJCAI'91*, 1991.