An argumentation-theoretic approach to logic program transformation

Francesca Toni and Robert A. Kowalski

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{ft,rak}@doc.ic.ac.uk

Abstract. We present a methodology for proving that any program transformation which preserves the least Herbrand model semantics when applied to sets of Horn clauses also preserves all semantics for normal logic programming that can be formulated in argumentation-theoretic terms [3, 4, 16]. These include stable model, partial stable model, preferred extension, stationary expansion, complete scenaria, stable theory, acceptability and well-founded semantics. We apply our methodology to prove that (some forms of) unfolding, folding and goal replacement preserve all these semantics. We also show the relationship of our methodology to that of Aravindan and Dung [1].

1 Introduction

The standard semantics for definite logic programs (i.e. the least Herbrand model semantics) is preserved by most of the program transformations studied in the literature, in particular by (some forms of) unfolding, folding and goal replacement (see [13] for a summary of these results). This paper provides a methodology for lifting these results from the definite logic program case to the normal logic program case, with respect to stable model [9], partial stable model [15], preferred extension [5], stationary expansion [14], complete scenaria [5], stable theory [10], acceptability [11] and well-founded semantics [18]. Most of the concrete cases obtained by applying our methodology [1] have been already shown elsewhere in the literature (see [1] for some of these results). Therefore, the main contribution of this paper lies in the general technique rather than in the concrete results.

All the semantics to which our methodology applies can be formulated in an argumentation framework [3, 4, 16], based upon a uniform notion of attack between sets of negative literals regarded as assumptions. We show that, to prove that any program transformation preserves all these semantics, it is sufficient to prove that there is a one-to-one correspondence between attacks before and after the transformation. This proof covers all of the semantics which can be defined in argumentation-theoretic terms, because the different semantics differ only in the way in which they build upon the same notion of attack. This technique, for proving soundness and completeness of a transformation, has already been

[1] with the exception that unfolding and folding preserve the acceptability semantics of [11]

used in [17] to show that abductive logic programs [16] can be transformed into normal logic programs preserving all argumentation-theoretic semantics. This general technique can be specialised to the case where the attacks before and after the transformation are exactly the same. The methodology we propose is a further specialisation of this technique, based upon the observation that all attacks are preserved if a transformation preserves the least Herbrand model semantics when it is applied to definite logic programs. The transformation must be such that it is applicable in the definite logic program case. In particular, unfolding inside negation is not allowed.

In [1], Aravindan and Dung show that (some forms of) folding and unfolding preserve the stable model, partial stable model, preferred extension, stationary expansion, complete scenaria, stable theory and well-founded semantics, due to the fact that normal logic programs before and after the application of folding and unfolding have the same semantic kernel [6] and that normal logic programs and their semantic kernels have the same semantics (for each of stable model, partial stable model, preferred extension, stationary expansion, complete scenaria, stable theory and well-founded semantics). We will show that this technique, of proving the correctness of a program transformation by proving that it preserves the semantic kernel, is equivalent to the technique of proving the correctness of a program transformation by proving that it preserves attacks. Therefore, our methodology can also be seen as a specialisation of Aravindan and Dung's. However, our methodology also generalises that of Aravindan and Dung in the sense that it applies directly to many program transformations. Whereas Aravindan and Dung consider only forlding and unfolding, ours is a general methodology for showing that, under certain conditions, any transformation that preserves the semantics of definite programs also preserves any semantics of normal logic programs that can be defined in argumentation terms.

The paper is organised as follows. First we apply the methodology to prove that the unfolding transformation, reviewed in section 2, preserves all semantics for normal logic programming which can be formulated in argumentation-theoretic terms of [3, 4, 16]. The unfolding transformation we consider is adapted from [13]. Section 3 reviews the argumentation framework and the formulation of various normal logic programming semantics in this framework. Section 4 proves that all semantics for normal logic programming formulated in the argumentation framework are preserved by unfolding. Section 5 presents the general methodology, by abstracting away from the proof in section 4. It also applies the general methodology to other transformations. Section 6 shows the relationships between our methodology and others, in particular the one in [1]. Finally, section 7 presents conclusions and discusses directions for further research.

2 Unfolding transformation

Normal logic programs are sets of clauses. A clause C is a formula of the form

$$H \leftarrow L_1, \ldots, L_n$$

where H is an atom, L_i is a literal, i.e. either an atom or the negation of an atom, for $i = 1, \ldots, n$, and all variables in $H, L_1, \ldots, L_n$ are implicitly universally quantified. H is the *head* and $L_1, \ldots, L_n$ the *body* of the clause C, referred to as $head(C)$ and $body(C)$ respectively. *Definite logic programs* are normal logic programs with no negative literals.

Definition 1. Let $\mathcal{P}$ be a normal logic program and let $C \in \mathcal{P}$ be a clause $H \leftarrow B_1, A, B_2$ where A is an atom and B_1, B_2 are (possibly empty) sets of literals. Suppose that

- $\{D_1, \ldots, D_m\}$, with $m > 0$, is the set of all clauses [2] in $\mathcal{P}$ such that A is unifiable with $head(D_1), \ldots, head(D_m)$ with most general unifiers $\theta_1, \ldots, \theta_m$, respectively, and
- U_j is the clause $[H \leftarrow B_1, body(D_j), B_2]\theta_j$ for $j = 1, \ldots m$.

Then, the result of *unfolding* C in $\mathcal{P}$ with respect to A is the program $(\mathcal{P} - \{C\}) \cup \{U_1, \ldots, U_m\}$.

The single unfolding step defined above can be repeated. A sequence of programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ is an *unfolding sequence* if any program $\mathcal{P}_{k+1}$, with $1 \leq k < n$, is obtained from $\mathcal{P}_k$ by applying an unfolding step.

Proving the *correctness of the unfolding transformation* (i.e. its soundness and completeness) with respect to a given normal logic programming semantics $\mathcal{S}em$ amounts to proving that, for any unfolding sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and for any query [3] $\mathcal{Q}$ in the vocabulary of the original program $\mathcal{P}_1$: [4]

$$\mathcal{P}_1 \models_{\mathcal{S}em} \mathcal{Q} \text{ if and only if } \mathcal{P}_n \models_{\mathcal{S}em} \mathcal{Q}.$$

By induction, it suffices to show that, for any program $\mathcal{P}$ and for any program $\mathcal{P}'$ obtained by unfolding some clause in $\mathcal{P}$,

$$\mathcal{P} \models_{\mathcal{S}em} \mathcal{Q} \text{ if and only if } \mathcal{P}' \models_{\mathcal{S}em} \mathcal{Q}.$$

More generally, to prove the correctness of a sequence of transformation steps, not necessarily unfolding steps, with respect to a semantics $\mathcal{S}em$ it suffices to prove that any such step is correct with respect to $\mathcal{S}em$.

In section 4, we will show correctness of the unfolding transformation with respect to all semantics $\mathcal{S}em$ that can be formulated in the argumentation framework in terms of a single notion of *attack*.

---

[2] All variables in each of C, $D_1, \ldots, D_m$ are assumed to be standardised apart.

[3] In general, the correctness of unfolding as well as other program transformations is studied with respect to a given subset of all possible queries in the vocabulary of the original program. In this paper, for simplicity we will assume that the given subset of queries coincides with the set of all possible queries.

[4] Here and in the rest of the paper, for any program $\mathcal{P}rog$, $\mathcal{P}rog \models_{\mathcal{S}em} \mathcal{Q}$ stands for "$\mathcal{Q}$ holds in $\mathcal{P}rog$ with respect to the semantics $\mathcal{S}em$".

3 An argumentation framework

The abstract argumentation framework proposed in [3, 4, 16] can be used to define many exisiting semantics for non-monotonic reasoning in general and for normal logic programming in particular.

Definition 2. An argumentation framework is a tuple $\langle \mathcal{T}, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$ where

- $\mathcal{T}$ is a *theory* in some formal language,
- $\vdash$ is a notion of *monotonic derivability* for the given language,
- $\mathcal{AB}$ is a set of *assumptions*, which are sentences of the language, and
- $\mathcal{IC}$ is a set of *denial integrity constraints with retractibles*.

Normal logic programming can be formulated as an instance of such a framework. Given a normal logic program $\mathcal{P}$, the corresponding argumentation framework is $\langle \mathcal{T}, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$ where

- $\mathcal{T}$ is the set of all variable-free instances of clauses in $\mathcal{P}$;
- $\vdash$ is modus ponens for the clause implication symbol $\leftarrow$;
- $\mathcal{AB}$ is the set of all variable-free negative literals;
- $\mathcal{IC}$ is the set consisting of all denials of the form $\neg[A \wedge \mathit{not}\ A]$, where A is a variable-free atom and $\mathit{not}\ A$ is retractible.

Other non-monotonic logics, including default logic, autoepistemic logic, non-monotonic modal logic can also be understood as special cases of such a framework [3, 4, 16].

In any argumentation framework, a sentence is a non-monotonic consequence if it follows monotonically from $\mathcal{T}$ extended by means of an "acceptable" set of assumptions $\Delta \subseteq \mathcal{AB}$. Various notions of "acceptability" can be defined, based upon a single notion of "attack" between sets of assumptions.

Intuitively, one set of assumptions "attacks" another if the two sets together with the theory violate an integrity constraint (i.e. the two sets together with the theory derive, via $\vdash$, the sentence which is denied by the integrity constraint), and the set which is attacked is deemed responsible for the violation. Responsibility for violation of integrity can be assigned by explicitly indicating part of the integrity constraint as *retractible* (see [12]). A set of assumptions which leads to the derivation of a retractible is deemed responsible for the violation of integrity. Thus integrity can be restored by "retracting" such an assumption. Formally:

Definition 3. Given an argumentation framework $\langle \mathcal{T}, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$:
a set of assumptions $\Delta' \subseteq \mathcal{AB}$ *attacks* another set $\Delta \subseteq \mathcal{AB}$ if and only if for some integrity constraint $\neg[L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_n] \in \mathcal{IC}$ with L_i retractible,
(1) $\mathcal{T} \cup \Delta' \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, and
(2) $\mathcal{T} \cup \Delta \vdash L_i$.

In the simple and frequently occurring case where all retractibles in integrity constraints are assumptions and any assumption α can be derived from $\mathcal{T} \cup \Delta$

only if $\alpha \in \Delta$, for any $\Delta \subseteq \mathcal{AB}$, then condition (2) in the definition above becomes

(2′) $L_i \in \Delta$.

This simplification applies to the case of the argumentation framework corresponding to normal logic programming, where *not* A is retractible for every integrity constraint $\neg[A \wedge not\ A]$ in $\mathcal{IC}$. Therefore, the notion of attack in the argumentation framework $\langle \mathcal{T}, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$ corresponding to a normal logic program $\mathcal{P}$ reduces to the following:

- a set of variable-free negative literals Δ' *attacks* another set Δ if and only if $\mathcal{T} \cup \Delta' \vdash A$, for some literal *not* $A \in \Delta$.

Various notions of "acceptability" can be defined in terms of the same, uniform notion of attack and can be applied to any non-monotonic logic defined in argumentation-theoretic terms. Here we mention some of the notions presented in [3, 4, 16]: A set of assumptions which does not *attack* itself is called

- *stable*, if and only if it *attacks* all assumptions it does not contain, where Δ *attacks* δ if and only if Δ *attacks* $\{\delta\}$;
- *admissible*, if and only if it *attacks* all sets of assumptions that *attack* it, i.e. it *defends* itself against all attacks;
- *preferred*, if and only if it is maximally admissible (with respect to set inclusion);
- *weakly stable*, if and only if for each set of assumptions that *attacks* it, the set together with the attacking set *attacks* some assumption in the attacking set which does not occur in the given set;
- *stable theory*, if and only if it is maximally weakly stable (with respect to set inclusion);
- *acceptable* [5], if and only if it is *acceptable to* the empty set of assumptions, where a set of assumptions Δ is *acceptable to* another set Δ_0 if and only if
  - $\Delta \subseteq \Delta_0$, or
  - for each set of assumptions Δ' that *attacks* Δ - Δ_0, there exists a set of assumptions Δ'' such that Δ'' *attacks* Δ' - $(\Delta \cup \Delta_0)$ and Δ'' is *acceptable to* $\Delta \cup \Delta_0 \cup \Delta'$;
- *complete*, if and only if it is admissible and it contains all assumptions it defends, where Δ *defends* an assumption δ if and only if, for all sets of assumptions Δ', if Δ' *attacks* δ then Δ *attacks* Δ';
- *well-founded*, if and only if it is minimally complete (with respect to set inclusion).

Note that for all of these semantics, an "acceptable" set of assumptions satisfies all the denial integrity constraints, since any such set does not attack itself. Note, moreover, that the notion of integrity satisfaction is compatible with three-valued semantics for normal logic programming, since satisfaction of the integrity constraint $\neg[\underline{A} \wedge not\ A]$ does not imply that either A or *not* A must be derived.

[5] This notion should not be confused with the informal notion of "acceptable" set of assumptions used elsewhere in this paper.

Almost all existing semantics for normal logic programming can be expressed in argumentation-theoretic terms, as proved in [3, 4, 16]. In particular, stable models [9] correspond to stable sets of assumptions, partial stable models [15] and preferred extensions [5] correspond to preferred sets of assumptions, stable theories [10] correspond to stable theory sets of assumptions, acceptability [11] corresponds to acceptable sets of assumptions, stationary expansions [14] and complete scenaria [5] correspond to complete sets of assumptions and well-founded semantics [18] corresponds to the well-founded set of assumptions. As far as we know, these include all existing semantics except the (various versions of the) completion semantics. Moreover, the same notions of "acceptability" apply also to any other non-monotonic logic that can be defined in argumentation-theoretic terms.

In the remainder of this paper we will refer to any semantics for normal logic programming which can be expressed in argumentation-theoretic terms (see above) as "argumentation semantics".

4 Correctness of the unfolding transformation

Given a normal logic program $\mathcal{P}$ and the program $\mathcal{P}'$ obtained by unfolding some clause in $\mathcal{P}$, let $\mathcal{F}_{\mathcal{P}}$ and $\mathcal{F}_{\mathcal{P}'}$ be the argumentation frameworks corresponding to $\mathcal{P}$ and $\mathcal{P}'$, respectively.

In general, the Herbrand base of $\mathcal{P}$ may be larger than the Herbrand base of $\mathcal{P}'$, since the Herbrand universe of $\mathcal{P}$ may be larger than the Herbrand universe of $\mathcal{P}'$, as illustrated by the following example.

Example 1. Consider the program $\mathcal{P}$

$$p(X) \leftarrow q(a)$$
$$q(X) \leftarrow r(b)$$

and the program $\mathcal{P}'$ obtained by unfolding the first clause in $\mathcal{P}$:

$$p(X) \leftarrow r(b)$$
$$q(X) \leftarrow r(b)$$

The constant a belongs to the Herbrand universe of $\mathcal{P}$ but not to that of $\mathcal{P}'$.

As a consequence, the set of assumptions in $\mathcal{F}_{\mathcal{P}}$ may be larger than the set of assumptions in $\mathcal{F}_{\mathcal{P}'}$.

For simplicity, it is convenient to assume that, except for $\mathcal{P}$ and $\mathcal{P}'$, $\mathcal{F}_{\mathcal{P}}$ and $\mathcal{F}_{\mathcal{P}'}$ coincide, i.e. the two framework have the same monotonic derivability notion, $\vdash$, and the same sets of assumptions and integrity constraints. This assumption can be made, without loss of generality, by assuming that $\mathcal{P}$ and $\mathcal{P}'$ are formulated in the same vocabulary (Herbrand universe), which may, however, be larger than the vocabulary actually occurring in $\mathcal{P}$ and $\mathcal{P}'$ (this assumption is also adopted elsewhere, e.g. in [13]).

Theorem 4. *Given any two sets of assumptions Δ' and Δ in $\mathcal{F}_{\mathcal{P}}$ (and therefore in $\mathcal{F}_{\mathcal{P}'}$),*

Δ' attacks Δ in $\mathcal{F}_{\mathcal{P}}$ if and only if Δ' attacks Δ in $\mathcal{F}_{\mathcal{P}'}$.

Before proving this theorem, let us explore its consequences. Since the different argumentation semantics can all be formulated in terms of the same notion of attack and differ only in the way they use this notion (see the end of section 3), then

Theorem 5. *For any programs $\mathcal{P}rog$ and $\mathcal{P}rog'$, let $\mathcal{F}_{\mathcal{P}rog}$ and $\mathcal{F}_{\mathcal{P}rog'}$ be the corresponding argumentation frameworks. If*

(i) *there is a one-to-one onto (bijective) mapping, m, from sentences in the vocabulary of $\mathcal{P}rog$ onto sentences in the vocabulary of $\mathcal{P}rog'$, and*

(ii) *there is a one-to-one correspondence via m between attacks in $\mathcal{F}_{\mathcal{P}rog}$ and attacks in $\mathcal{F}_{\mathcal{P}rog'}$, i.e. for every sets of assumptions Δ', Δ in $\mathcal{F}_{\mathcal{P}rog}$, Δ' attacks Δ in $\mathcal{F}_{\mathcal{P}rog}$ if and only if $m(\Delta')$ attacks $m(\Delta)$in $\mathcal{F}_{\mathcal{P}rog'}$,* [6]

then, for every argumentation semantics $\mathcal{S}em$, there is a one-to-one correspondence via m between $\mathcal{S}em$ for $\mathcal{P}rog$ and $\mathcal{S}em$ for $\mathcal{P}rog'$, i.e. for all queries $\mathcal{Q}$ in the vocabulary of $\mathcal{P}rog$,

$$\mathcal{P}rog \models_{\mathcal{S}em} \mathcal{Q} \quad \text{if and only if} \quad \mathcal{P}rog' \models_{\mathcal{S}em} m(\mathcal{Q}).$$

If $\mathcal{P}rog'$ is obtained by applying a transformation to $\mathcal{P}rog$, then this theorem expresses a very general technique to prove correctness of the transformation. Such a general technique is needed when the given transformation changes the vocabulary of the original program, so that to establish the correspondence between attacks, it is necessary to establish first a mapping between the vocabulary of the original program $\mathcal{P}rog$ and that of the transformed program $\mathcal{P}rog'$. This technique has been used in [17] to show that abductive logic programs [16] can be transformed correctly into normal logic programs, by mapping abducible atoms onto negative literals.

When the transformation does not change the vocabulary of the original program, as in the case of unfolding, then the mapping m can be taken to be the identity function and condition (ii) reduces to the condition that the attacks in $\mathcal{F}_{\mathcal{P}rog}$ and $\mathcal{F}_{\mathcal{P}rog'}$ are exactly the same. Therefore, theorems 4 and 5 directly imply

Corollary 6. *Let $\mathcal{P}$ be a normal logic program and $\mathcal{P}'$ be obtained by unfolding some clause in $\mathcal{P}$. Then, for every argumentation semantics $\mathcal{S}em$ for normal logic programming, $\mathcal{S}em$ for $\mathcal{P}$ coincides with $\mathcal{S}em$ for $\mathcal{P}'$, i.e. for all queries $\mathcal{Q}$ in the vocabulary of $\mathcal{P}$,*

$$\mathcal{P} \models_{\mathcal{S}em} \mathcal{Q} \quad \text{if and only if} \quad \mathcal{P}' \models_{\mathcal{S}em} \mathcal{Q}.$$

[6] For every set of sentences S, $m(S) = \{m(\alpha) | \alpha \in S\}$.

Proof of theorem 4
First note that we can assume that $\mathcal{P}$ and $\mathcal{P}'$ are variable-free. If they are not, they can be replaced by all their variable-free instances over their common Herbrand universe. Then, directly from the definition of *attack*, theorem 4 is an immediate consequence of the following lemma.

Lemma 7. *For each atom A in the common Herbrand base of $\mathcal{P}$ and $\mathcal{P}'$ and for each set of assumptions Δ in $\mathcal{F}_{\mathcal{P}}$ (and therefore in $\mathcal{F}_{\mathcal{P}'}$),*
$\mathcal{P} \cup \Delta \vdash A$ if and only if $\mathcal{P}' \cup \Delta \vdash A$.

Lemma 7 follows from lemma 8. Here and in the rest of the paper, for any normal logic program $\mathcal{P}rog$, $\mathcal{P}rog_*$ stands for the definite logic program obtained by interpreting every negative literal, $not\, p$, in $\mathcal{P}rog$ syntactically as a new positive atom, p^* (see [8]). Similarly, for any set of assumptions Δ, Δ_* stands for the definite logic program obtained by interpreting every negative literal in Δ as a new positive atom.

Lemma 8. *For any normal logic program framework $\langle \mathcal{P}rog, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$, for any set of assumptions $\Delta \subseteq \mathcal{AB}$ and for any atom A in the Herbrand base of $\mathcal{P}rog$,*
$\mathcal{P}rog \cup \Delta \vdash A$ if and only if
$\mathcal{P}rog_* \cup \Delta_* \vdash A$ if and only if
A belongs to the least Herbrand model of $\mathcal{P}rog_* \cup \Delta_*$.

This lemma follows directly from the fact that a ground atom A belongs to the least Herbrand model of a definite program (e.g. $\mathcal{P}rog_* \cup \Delta_*$) if and only if it is derivable from the program, as proved in [7].

Proof of lemma 7
$\mathcal{P} \cup \Delta \vdash A$
if and only if (by lemma 8)
A belongs to the least Herbrand model of $\mathcal{P}_* \cup \Delta_*$
if and only if (by the result that the unfolding transformation for definite logic programs is correct with respect to the least Herbrand model semantics [13])
A belongs to the least Herbrand model of $(\mathcal{P}')_* \cup \Delta_*$
if and only if (by lemma 8)
$\mathcal{P}' \cup \Delta \vdash A$.

This concludes the proof of theorem 4. Note that the proof of lemma 7 makes use of the property that $(\mathcal{P}')_*$ can also be obtained by applying unfolding to $\mathcal{P}_*$ (by applying the same unfolding step used to obtain $\mathcal{P}'$ from $\mathcal{P}$). In other words, the two operations $'$ and $*$ commute: $(\mathcal{P}')_*=(\mathcal{P}_*)'$.

5 General methodology

In proving the correctness of unfolding, we have used a technique which can be used more generally to prove that any transformation which preserves all attacks

also preserves all argumentation semantics. Note, however, that a precondition of this technique, which holds for the unfolding transformation, is that the argumentation frameworks corresponding to the programs before and after the transformation should have the same set of assumptions, or, equivalently, that the Herbrand bases before and after the transformation coincide. We will refer to this precondition as *Property 1*.

In this section we will generalise the proof of theorem 4, to obtain a more general methodology for proving the correctness of other program transformations satisfying *Property 1* as well as other properties we will discuss next. This methodology is less powerful than the technique expressed by theorem 5 but its preconditions are easier to check. For this purpose, let us analyse the proof of theorem 4.

Theorem 4 is an immediate consequence of lemma 7, which, in turn, follows directly from two properties. The first, expressed by lemma 8, is a general property, which holds for any normal logic program in general and for the programs $\mathcal{P}$ and $\mathcal{P}'$ before and after the unfolding transformation in particular. However, the second property, that unfolding preserves the least Herbrand model semantics of definite programs, is specific (in the sense that it is not true for every transformation). We will refer to this property as *Property 2*. Its applicability depends, in turn, upon the fact that unfolding a definite program produces a definite program (*Property 3*) and the fact that unfolding commutes with the operation $*$ of interpreting negative literals as positive atoms (*Property 4*). The unfolding transformation we have considered satisfies *Property 4* because it affects only atoms (we do not allow unfolding inside negation).

No other properties, besides *Properties 1, 2, 3* and *4*, are required in the proof of the correctness of unfolding. Therefore, the same proof demonstrates the correctness, with respect to all argumentation semantics, of any transformation which satisfies *Properties 1, 2, 3* and *4*. This is the basis of our methodology, which is expressed by the following theorem.

Theorem 9. *Given any transformation Transf from normal logic programs to normal logic programs, let $'$ be any specific application of this transformation producing a deterministic result. Then, if for all normal logic programs $\mathcal{P}$:*

Property 1: *$\mathcal{P}'$ and $\mathcal{P}$ have the same Herbrand base;*
Properties 2 *and* 3: *if $\mathcal{P}$ is a definite program then*
- *$\mathcal{P}'$ is a definite program (3),*
- *$\mathcal{P}'$ and $\mathcal{P}$ have the same least Herbrand model (2);*

Property 4: *$(\mathcal{P}')_* = (\mathcal{P}_*)'$;*

then Transf is correct with respect to all argumentation semantics, $\mathcal{S}em$, i.e. for all queries $\mathcal{Q}$,

$$\mathcal{P} \models_{\mathcal{S}em} \mathcal{Q} \quad \textit{if and only if} \quad \mathcal{P}' \models_{\mathcal{S}em} \mathcal{Q}.$$

Proof: The conclusion of the theorem holds for $\mathcal{P}$ and $\mathcal{P}'$ if theorem 4 holds for $\mathcal{P}$ and $\mathcal{P}'$. Theorem 4 makes sense because *Transf* satisfies *Property 1*. Theorem 4

holds for $\mathcal{P}$ and $\mathcal{P}'$ if lemma 7 holds for $\mathcal{P}$ and $\mathcal{P}'$. But:

$\mathcal{P} \cup \Delta \vdash A$

if and only if (by lemma 8)

A belongs to the least Herbrand model of $\mathcal{P}_* \cup \Delta_*$

if and only if (by *Properties 2, 3* and *4*)

A belongs to the least Herbrand model of $(\mathcal{P}')_* \cup \Delta_*$

if and only if (by lemma 8)

$\mathcal{P}' \cup \Delta \vdash A$.

This concludes the proof of theorem 9.

This theorem can be used to establish the correctness of all versions of the *folding* transformation which satisfy *Property 2* (see [13]), since every version of folding satisfies the other properties. Moreover, it can be used to establish correctness of any other transformation which satisfies all four properties. Consider, for example, the following version of goal replacement, adapted from [13].

Definition 10. Given

- a normal logic program $\mathcal{P}$,
- sets of atoms G_1, G_2, possibly empty, in the vocabulary of $\mathcal{P}$,
- $C \in \mathcal{P}$, a clause $H \leftarrow B_1, G_1, B_2$, where B_1, B_2 are (possibly empty) sets of literals,

let $\{X_1, \ldots, X_n\} = vars(G_1) \cap vars(G_2)$. [7] Suppose that

- $G_1 \equiv G_2$ is an *H-valid replacement rule with respect to* $\mathcal{P}$, i.e., for all $a_1, \ldots, a_n$ in the Herbrand universe of $\mathcal{P}$,
  - $\mathcal{P}_H \models_{LHM} \exists G_1[X_1/a_1, \ldots, X_n/a_n]$ if and only if $\mathcal{P}_H \models_{LHM} \exists G_2[X_1/a_1, \ldots, X_n/a_n]$ [8] and
  - for all argumentation semantics $\mathcal{S}em$, for $i = 1, 2$, $\mathcal{P} \models_{\mathcal{S}em} \exists G_i[X_1/a_1, \ldots, X_n/a_n]$ if and only if $\mathcal{P}_H \models_{LHM} \exists [G_i X_1/a_1, \ldots, X_n/a_n]$,

  where $\mathcal{P}_H$ is the subset of $\mathcal{P}$ consisting of all and only Horn clauses,
- $vars(H, B_1, B_2) \cap vars(G_2) = vars(H, B_1, B_2) \cap vars(G_1) = \{X_1, \ldots, X_n\}$
- C' is the clause $H \leftarrow B_1, G_2, B_2$.

Then, the result of *H-valid replacing the goal* G_1 in $C \in \mathcal{P}$ by the goal G_2 is $(\mathcal{P} - \{C\}) \cup \{C'\}$.

This definition of H-valid goal replacement is equivalent to the definition of goal replacement in [13] except for the notion of H-valid replacement rule, which is a variation of the notion of valid replacement rule in [13].

[7] For any set of expressions $E_1, \ldots, E_m$, $vars(E_1, \ldots, E_m)$ stands for the set of all free variables in $E_1, \ldots, E_m$.

[8] LHM stands for least Herbrand model semantics, and, for any sentence G, $\exists G$ stands for the existential closure of G with respect to all variables occurring in G.

Let $\mathcal{P}'$ be the result of H-valid replacing the goal G_1 in $C \in \mathcal{P}$ by the goal G_2. If $G_2 \equiv G_1$ is a H-valid replacement rule with respect to $\mathcal{P}'$, then definition 10 defines a version of *reversible goal replacement*, which preserves the least Herbrand model semantics when applied to definite programs (see [13]). Moreover, it satisfies *Property 3* and it does not modify the set of predicates, by definition, i.e. it satisfies *Property 1*. Finally, since it only affects atoms in clauses, it satisfies *Property 4*. As a consequence, by theorem 9, this form of reversible goal replacement preserves all argumentation semantics.

6 Comparisons

The general methodology (theorem 9) is a special case of the general technique of proving the correctness of a transformation by proving that attacks are preserved by the transformation. This general technique, in turn, is a special case of the even more general technique (theorem 5) for proving correctness of a program transformation by proving that there is a one-to-one correspondence between attacks before and after the transformations. Theorem 5 generalises the method used in [17] to show that abductive logic programs [16] can be transformed correctly into normal logic programs.

In the remainder of this section we compare our methodology with the technique used by Aravindan and Dung in [1] to prove that (some forms of) unfolding and folding are correct with respect to a number of semantics for normal logic programming. We will see that their technique is equivalent to the method, used in section 4 (for unfolding), of proving that the attack relations before and after the transformation are identical.

The method in [1] uses the notion of semantic kernel, given by the following definition which is adapted from [6].

Definition 11. Given a normal logic program $\mathcal{P}rog$, let $\mathcal{S}_{\mathcal{P}rog}$ be the operator on sets of variable-free clauses of the form

$H \leftarrow \Delta$ with Δ a (possibly empty) set of negative literals

defined as follows:

$\mathcal{S}_{\mathcal{P}rog}(I) = \{ H \leftarrow \Delta', B_1, \ldots, B_m \mid \Delta'$ is possibly empty, $m \geq 0$,
$H \leftarrow \Delta', H_1, \ldots, H_m$ is a variable-free instance of a clause in $\mathcal{P}rog$,
$H_i \leftarrow B_i \in I$, for $i = 1, \ldots, m$ $\}$.

Then, the *semantic kernel* of $\mathcal{P}rog$, $SK(\mathcal{P}rog)$ in short, is the least fix point of $\mathcal{S}_{\mathcal{P}rog}$, i.e. (since $\mathcal{S}_{\mathcal{P}rog}$ is continuous) $SK(\mathcal{P}rog) = \bigcup_{i \geq 1} \mathcal{S}^i_{\mathcal{P}rog}(\emptyset)$.

Note that, in the construction of $SK(\mathcal{P}rog)$, one application of the operator $\mathcal{S}_{\mathcal{P}rog}$ amounts to performing a step of monotonic reasoning, leaving the non-monotonic part (the negative literals) untouched.

In [1], it is proved that unfolding and folding preserve the semantic kernel of programs, i.e. for any logic program $\mathcal{P}$ and for any logic program $\mathcal{P}'$ obtained by applying folding or unfolding to $\mathcal{P}$, $SK(\mathcal{P}) = SK(\mathcal{P}')$. As a consequence, unfolding and folding preserve all semantics $\mathcal{S}em$ for normal logic programming for which it can be shown that, for any logic program $\mathcal{P}rog$ and query $\mathcal{Q}$, $\mathcal{P}rog$

$\models_{Sem} \mathcal{Q}$ if and only if $SK(\mathcal{P}rog) \models_{Sem} \mathcal{Q}$. These semantics include all the argumentation semantics for normal logic programming we have considered in this paper.

Theorems 13 and 14 below imply that, for the purpose of proving the correctness of a transformation with respect to any argumentation semantics for normal logic programming, the two techniques of showing that the transformation preserves the semantic kernel and of showing that it preserves all attacks are equivalent. These theorems are stated in terms of the following definition which is adapted from [2]:

Definition 12. For any normal program $\mathcal{P}rog$ and sets of assumptions Δ', Δ in the argumentation framework corresponding to $\mathcal{P}rog$

Δ' is a *locally minimal attack against* Δ if and only if
there exists a subset $\mathcal{P}rog'$ of the set of all variable-free instances of clauses in $\mathcal{P}rog$ such that

- $\mathcal{P}rog' \cup \Delta' \vdash A$ for some $not\, A \in \Delta$, and
- there exists no $\Delta'' \subset \Delta'$ such that $\mathcal{P}rog' \cup \Delta'' \vdash A$.

Intuitively, a locally minimal attack is a minimal attack with respect to some subset of the program. A locally minimal attack can be thought of as the set of negative literals that occur in clauses directly involved in the derivation of the complement A of an assumption $not\, A$ in the attacked set. Since programs can have redundant clauses, including clauses which are subsumed by other clauses, the set of all negative literals in a derivation is not guaranteed to be minimal in an absolute sense.

Locally minimal attacks are important because they subsume all other attacks, as expressed by the following

Theorem 13. *For any normal logic program $\mathcal{P}rog$ and sets of assumptions Δ', Δ in the argumentation framework corresponding to $\mathcal{P}rog$,*

Δ' attacks Δ if and only if
there exists $\Delta'' \subseteq \Delta'$ such that Δ'' is a locally minimal attack against Δ.

This theorem follows directly from the monotonicity of $\vdash$. (See [16] for the full proof.) An important consequence of this theorem is that a transformation preserves all argumentation semantics if and only if it preserves all locally minimal attacks. In fact, via theorem 13, all notions of "acceptable" set of assumptions in section 3 can be reformulated in terms of locally minimal attacks and $\subseteq$. For example, a set of assumptions Δ is *stable* if and only if none of its subsets is a locally minimal attack against Δ and some subset of Δ is a locally minimal attack against every assumption Δ does not contain.

The following theorem establishes the equivalence between semantic kernels and locally minimal attacks.

Theorem 14. *Given a normal logic program $\mathcal{P}rog$, let $\langle \mathcal{P}rog, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$ be the corresponding argumentation framework. Then, for every set of assumptions Δ in $\langle \mathcal{P}rog, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$ and atom H in the vocabulary of $\mathcal{P}rog$:*

$(H \leftarrow \Delta) \in SK(\mathcal{P}rog)$ *if and only if*
Δ *is a locally minimal attack against* $\{not\, H\}$.

The proof of this theorem can be found in the appendix.

7 Conclusions and future work

We have presented a methodology for proving that some program transformations, e.g. unfolding, folding and (a form of) goal replacement, preserve many semantics for normal logic programming, namely all argumentation semantics for normal logic programming (these include all known semantics except the completion semantics). This methodology is a special case of the more general technique, introduced in [17], of showing that there is a one-to-one correspondence between attacks before and after a transformation.

We are investigating the application of the proposed methodology to show the correctness of other program transformations. Clause elimination and introduction do not satisfy *Property 1* of theorem 9. Therefore, the methodology cannot be applied to these transformations. Note, however, that these transformations do not preserve all argumentation semantics. For example, given the normal logic program $\mathcal{P}$

$$p \leftarrow not\, q$$

the program $\mathcal{P}'$ obtained by introducing the extra clause

$$r \leftarrow not\, r$$

is not equivalent to $\mathcal{P}$ under the stable model semantics. In fact, $\mathcal{P}$ has a single stable model $\{p\}$ while $\mathcal{P}'$ has no stable model. Whether some restricted forms of clause elimination and introduction might preserve all argumentation semantics (including the stable model semantics), and whether our methodology can be generalised to prove this are open issues that require further investigation.

Finally, another interesting topic for future research is the generalisation of our methodology to prove the correctness of transformations for other non-monotonic logics including default, autoepistemic and non-monotonic modal logic, which can be formalised in the argumentation framework of section 3.

Acknowledgements

This research was supported by the Fujitsu Research Laboratories. The authors are grateful to Alberto Pettorossi, Maurizio Proietti and each other for helpful discussions, and to the anonymous referees for helpful suggestions.

References

1. C. Aravindan, P.M. Dung, On the correctness of unfold/fold transformation of normal and extended logic programs. *Journal of Logic Programming* 24(3):201–217 (1995)
2. C. Aravindan, P.M. Dung. Belief dynamics, abduction and databases. *Proc. 4th European Workshop on Logic in AI* (1994)
3. A. Bondarenko, F. Toni, R. A. Kowalski, An assumption-based framework for non-monotonic reasoning. *LPNMR'93* (A. Nerode and L. Pereira eds.) MIT Press, 171–189
4. A. Bondarenko, P. M. Dung, R. A. Kowalski, F. Toni, An abstract, argumentation-theoretic framework for default reasoning. Technical Report (1995)
5. P.M. Dung, Negation as hypothesis: an abductive foundation for logic programming. *ICLP'91* (K. Furukawa ed.) MIT Press, 3–17
6. P.M. Dung, K. Kanchanasut, A fixpoint approach to declarative semantics of logic programs. *NACLP'89* 1:604–625
7. M.H. van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language. *ACM* 23(4):733–742 (1976)
8. K. Eshghi, R.A. Kowalski, Abduction compared with negation as failure. *ICLP'89* (G. Levi and M. Martelli eds.) MIT Press, 234–255
9. M. Gelfond, V. Lifschitz, The stable model semantics for logic programs. *ICLP'88* (K. Bowen and R.A. Kowalski eds.) MIT Press, 1070–1080
10. A.C. Kakas, P. Mancarella, Stable theories for logic programs. *ILPS'91* (V. Saraswat and K. Ueda eds.) MIT Press, 85–100
11. A.C. Kakas, P. Mancarella, P.M. Dung, The Acceptability Semantics for Logic Programs. *ICLP'94* (P. Van Hentenryck ed.) MIT Press, 504–519
12. R.A. Kowalski, F. Sadri, Knowledge representation without integrity constraints. Imperial College Technical Report (1988)
13. A. Pettorossi, M. Proietti, Transformation of logic programs. *Journal of Logic Programming* 19/20:261–320 (1994)
14. T.C. Przymusinski, Semantics of disjunctive logic programs and deductive databases. *DOOD'91* (C. Delobel, M. Kifer, and Y. Masunaga eds.) 85–107
15. D. Saccà, C. Zaniolo, Stable models and non determinism for logic programs with negation. *ACM Symposium on Principles of Database Systems*, ACM Press, 205–217 (1990)
16. F. Toni, Abductive logic programming, PhD Thesis, Imperial College (1995)
17. F. Toni, R.A. Kowalski. Reduction of abductive logic programs to normal logic programs. *ICLP'95* (L. Sterling ed.) MIT Press, 367–381
18. A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650 (1991)

Appendix

Proof of theorem 14. As in the proof of theorem 4, we will assume (without loss of generality) that $\mathcal{P}rog$ is variable-free.

First, we define the notion of *minimal derivability*. For any normal program $\mathcal{P}rog$, set of assumptions Δ in the argumentation framework corresponding to $\mathcal{P}rog$ and atom A in the vocabulary of $\mathcal{P}rog$,

$\mathcal{P}rog \cup \Delta \vdash_{min} A$ ($\mathcal{P}rog \cup \Delta$ *minimally derives* A)
if and only if there exists $\mathcal{P}rog' \subseteq \mathcal{P}rog$ such that
$\mathcal{P}rog' \cup \Delta \vdash A$ and there exists no $\Delta' \subset \Delta$ such that $\mathcal{P}rog' \cup \Delta' \vdash A$.
By definition, Δ is a locally minimal attack (with respect to $\mathcal{P}rog$) against Δ' if and only if $\mathcal{P}rog \cup \Delta \vdash_{min} A$ for some *not* $A \in \Delta'$. Therefore, theorem 14 directly follows from the following lemma:

Lemma 15. *Given a normal logic program* $\mathcal{P}rog$,
$H \leftarrow \Delta \in SK(\mathcal{P}rog)$ *if and only if* $\mathcal{P}rog \cup \Delta \vdash_{min} H$.

In the proof of this lemma $\vdash^i$ will indicate derivability by applying the modus ponens inference rule i times and $\vdash^i_{min}$ will indicate $\vdash_{min}$ as defined above but with $\vdash$ replaced for by $\vdash^i$.

Proof of lemma 15
$H \leftarrow \Delta \in SK(\mathcal{P}rog)$ if and only if $H \leftarrow \Delta \in \mathcal{S}^j_{\mathcal{P}rog}(\emptyset)$, for some $j \geq 1$. By induction on j, we prove that, for all $j \geq 1$,
$H \leftarrow \Delta \in \mathcal{S}^j_{\mathcal{P}rog}(\emptyset)$ if and only if $\mathcal{P}rog \cup \Delta \vdash^j_{min} H$,
which directly proves the lemma.

• If $j = 1$ then $H \leftarrow \Delta \in \mathcal{S}^j_{\mathcal{P}rog}(\emptyset)$ if and only if $H \leftarrow \Delta \in \mathcal{P}rog$ if and only if $\{H \leftarrow \Delta\} \cup \Delta \vdash^1 H$ if and only if $\mathcal{P}rog \cup \Delta \vdash^1_{min} H$.

• If $j > 1$ then let us assume the inductive hypothesis that, for each $1 \leq k < j$, $H' \leftarrow \Delta' \in \mathcal{S}^k_{\mathcal{P}rog}(\emptyset)$ if and only if $\mathcal{P}rog \cup \Delta' \vdash^k_{min} H'$.
Then, $H \leftarrow \Delta \in \mathcal{S}^j_{\mathcal{P}rog}(\emptyset)$
if and only if (by definition)
$H \leftarrow \Delta', H_1, \ldots, H_m \in \mathcal{P}rog$, with $m \geq 0$, and for all $i = 1, \ldots, m$ there exists some k_i with $1 \leq k_i \leq j-1$ such that $H_i \leftarrow B_i \in \mathcal{S}^{k_i}_{\mathcal{P}rog}(\emptyset)$, and $\Delta = \Delta' \cup B_1 \cup \ldots \cup B_m$
if and only if (by inductive hypothesis)
$H \leftarrow \Delta', H_1, \ldots, H_m \in \mathcal{P}rog$, with $m \geq 0$, and for all $i = 1, \ldots, m$ there exists some k_i with $1 \leq k_i \leq j-1$ such that $\mathcal{P}rog \cup B_i \vdash^{k_i}_{min} H_i$, i.e. there exists $\mathcal{P}rog_i \subseteq \mathcal{P}rog$ such that $\mathcal{P}rog_i \cup B_i \vdash^{k_i} H_i$ and B_i is minimal with respect to $\mathcal{P}rog_i$
if and only if
$\bigcup_{1 \leq i \leq m} \mathcal{P}rog_i \cup \{H \leftarrow \Delta', H_1, \ldots, H_m\} \cup \Delta \vdash^{k+1} H$, with k the maximum of the k_i and B_i is minimal with respect to $\mathcal{P}rog_i$
if and only if (since at least one of the k_i is necessarily $j-1$),
$\bigcup_{1 \leq i \leq m} \mathcal{P}rog_i \cup \{H \leftarrow \Delta', H_1, \ldots, H_m\} \cup \Delta \vdash^j H$ and B_i is minimal with respect to $\mathcal{P}rog_i$
if and only if (due to the minimality of the B_i with respect to $\mathcal{P}rog_i$)
$\mathcal{P}rog \cup \Delta \vdash^j_{min} H$. This concludes the proof.

This article was processed using the LaTeX macro package with LLNCS style