Reduction of abductive logic programs to normal logic programs

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

Abstract

In this paper we study a form of abductive logic programming which combines default and non-default abducibles and employs retractibles in integrity constraints. We also present a transformation from abductive to normal logic programs, which is correct and complete with respect to many semantics. These are all the semantics that can be formulated in an argumentation framework. A simplified form of the event calculus is used as an illustration.

1 Introduction

Abductive logic programming (ALP) is the extension of normal logic programming (NLP) to incorporate abducibles and integrity constraints. Abducibles are atoms (or more generally literals) that represent incomplete information which can be added to programs, provided their addition does not violate the integrity constraints.

Various forms of ALP have been presented in the literature (see [8] for a survey). In this paper we present a form of ALP based on those proposed by Eshghi and Kowalski [5, 6] and Kakas and Mancarella [9]. We allow both default and non-default abducibles, as proposed by Poole [15]. Moreover, not only do we use integrity constraints to constrain abducibles, but we also indicate how satisfaction of integrity should be restored. We do this by identifying one or more literals in each integrity constraint as being retractible, as proposed in [11].

In this paper we transform abductive logic programs in which every retractible literal in an integrity constraint is either an abducible atom or its negation into normal logic programs. Hence, by virtue of this transformation for this form of ALP, any semantics for NLP provides a semantics for ALP; and any proof procedure for NLP provides a proof procedure for ALP.

The main result of this paper is that the transformation preserves almost all semantics for ALP. Using an argumentation framework, we define the semantics of ALP in such a way that NLP is a special case. It has been shown [1, 2] that for NLP almost all known semantics can be defined in a uniform way, in such a framework, based upon a single notion of attack between sets of negative literals regarded as assumptions. The result that the transformation preserves semantics is proved by demonstrating a one to one correspondence between attacks before and after the transformation.

2 Abductive logic programming

Our notion of ALP builds upon the following (by now conventional) notion of an **abductive logic program** as a triple $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, where

- $\mathcal{P}$ is a **logic program**, i.e. a set of clauses of the form

$$H \leftarrow L_1, \ldots, L_n \tag{1}$$

  where $n \geq 0$, H is an atom, $L_1, \ldots L_n$ are literals, i.e. atoms A or negations of atoms *not* A, and all the variables in H, $L_0, \ldots, L_n$ are universally quantified. H is the **conclusion** and $L_1, \ldots L_n$ the **conditions** of (1). If H is an atom $p(t)$, [1] the set of all clauses having H as a conclusion is a **definition** for the predicate p.

- Ab_0 is a set of predicate symbols, called **abducible predicates**. Literals of the form $a(t)$ or *not* $a(t)$ where a is an abducible predicate are called **abducible literals**.

- I_0 is a set of denial **integrity constraints**, i.e. formulas of the form $\neg[L_1 \wedge \ldots \wedge L_n]$ where $n \geq 1$, each L_i is a literal, and all variables in $L_1, \ldots, L_n$ are universally quantified before the formula. [2]

Without loss of generality [8], we can assume that abducible predicates in Ab_0 have no definitions in $\mathcal{P}$.

Example 2.1 A simplified version of the event calculus [13] can be expressed by means of an abductive logic program $\langle \mathcal{P}_{ec}, \mathcal{AB}_{ec}, \mathcal{I}_{ec} \rangle$, where $\mathcal{P}_{ec}$ contains a **persistence axiom** expressing that a property P persists from the time T_1 that it is initiated by an event E to a later time T_2:

$$holds\_at(P, T_2) \leftarrow happens(E, T_1), T_1 < T_2, initiates(E, P), persists(T_1, P, T_2)$$

The predicates *happens* and *persists* are both abducibles in Ab_{ec}. New information that a property holds at a particular time can be assimilated by adding an explanation in terms of the happening of some event that initiates this property at an earlier time together with an assumption that the property persists between the two time points. The predicate *persists*

[1] In this paper the following conventions are used: t is a tuple of terms and X is a tuple of variables.

[2] Note that two kinds of negation occur in the integrity constraints, namely $\neg$ and *not*. However, neither kind is actually needed. Indeed, negation as failure literals can be replaced by positive abducible atoms and integrity constraints, as in [6]. Moreover, $\neg$ is simply a shorthand indicating that literals are incompatible.

expresses the default nature of the persistence axiom and is used to predict information, while the predicate *happens* is used as a non-default abducible to explain observations. If no integrity constraint is violated, a variable-free atom $persists(t_1, p, t_2)$ must necessarily be assumed, while a variable-free atom $happens(e, t_1)$ need not be assumed.

The integrity constraints in I_{ec} include the denials

$$\neg[persists(T_1, P, T_2) \wedge happens(E, T) \wedge terminates(E, P) \wedge T_1 < T < T_2]$$
$$\neg[happens(E, T) \wedge precondition(E, T, P) \wedge not\ holds\_at(P, T)]$$

The first expresses that a property P cannot persist from a time T_1 to a later time T_2 if an event E that terminates P happens at a time T between T_1 and T_2. The second expresses that an event E cannot happen at a time T if a precondition P of E does not hold at time T.

In many applications of the event calculus, the predicate *happens* is not abducible, but is defined by means of facts. In other applications, which combine narratives with hypothetical reasoning some instances of *happens* are defined by means of facts and others are abducible. In this latter case, the introduction of a new clause

$$happens(E, T) \leftarrow hyp\text{-}happens(E, T)$$

where the new predicate *hyp-happens* is abducible, but *happens* is not, makes abducible and non-abducible predicates disjoint.

In general, abducibles can be of two kinds, namely **default abducible predicates**, like *persists*, represented by $\mathcal{AB}_0^d$, and **non-default abducible predicates**, like *happens*, represented by $\mathcal{AB}_0^{nd}$. Therefore, Ab_0 is the union of $\mathcal{AB}_0^d$ and $\mathcal{AB}_0^{nd}$.

Some approaches to ALP [9, 16] consider only non-default abducibles, and use negation as failure to express default reasoning. The distinction between default reasoning and non-default abduction is made by Konolige [10], who, however, uses abduction for non-default hypothetical reasoning, but Reiter's default logic for default reasoning. Poole [15], on the other hand, uses an abductive framework where abducibles can be either default or non-default, and modifies Theorist to incorporate both kinds of abducibles.

Finally, in the form of ALP considered in this paper, in each integrity constraint at least one literal is specified as **retractible**. If the addition of abducible literals leads to a violation of integrity, then one of the retractibles is withdrawn to restore satisfaction. The use of retractibles in integrity constraints was proposed by Kowalski and Sadri in [11] for similar purposes.

The use of retractibles can be illustrated by means of the simplified event calculus example 2.1. Here, if an instance of the first integrity constraint is violated, it is natural to retract the corresponding instance of $persists(T_1, P, T_2)$. If an instance of the second integrity constraint is violated, it is natural to retract the corresponding instance of $happens(E, T)$.

In this, as in many other examples, it is natural to identify only some of the literals in an integrity constraint as retractible. If this is not possible, one can always nominate all literals as retractible.

The case where the retractibles are abducible literals is especially desirable, since it suffices not to abduce them in order to retract them. For this reason and because it simplifies the transformation, we shall assume that retractibles are always abducible literals. As shown in [17], in many cases abductive logic programs with non-abducible literals as retractibles can be transformed by "unfolding" into abductive logic programs where all retractibles are abducible literals. This transformation succeeds if the retractibles "depend upon" abducible literals; namely (and informally) the retractibles can be derived only by hypothesising abducible literals. The assumption that retractibles "depend upon" abducible literals, is justified by the fact that, if a retractible "depends upon" non-abducible literals only, then it might not be possible to retract it at all.

For the sake of readability, retractibles in integrity constraints will be underlined, e.g. the integrity constraints in I_{ec} will be written in the form:

$$\neg[\underline{persists(T_1,\ P,\ T_2)} \wedge happens(E,\ T) \wedge terminates(E,\ P) \wedge T_1 < T < T_2]$$

$$\neg[\underline{happens(E,T)} \wedge precondition(E,T,P) \wedge not\ holds\_at(P,T)] \qquad (2)$$

In applications which combine narratives with hypothetical reasoning the integrity constraint (2) needs to be modified to satisfy the restriction that retractibles are abducible literals. For this purpose, we can replace (2) by

$$\neg[\underline{hyp\text{-}happens(E,T)} \wedge precondition(E,T,P) \wedge not\ holds\_at(P,T)] \qquad (2')$$

assuming that those instances of *happens* which are defined by facts have been verified as satisfying (2) at time of input. We shall refer to this form of event calculus as the **modified event calculus**.

In most formulations of the persistence axiom, e.g. [4], a negative condition $not\ broken(T_1, P, T_2)$ is used instead of a positive default abducible condition $persists(T_1, P, T_2)$. Our transformation uses a variant of Satoh-Iwayama transformation [16] to justify the use of negation as failure to replace positive abducibles in general. It uses a variant of the technique of [12], to eliminate integrity constraints.

3 Transformation

The transformation is defined for abductive logic programs in which all retractibles are abducible literals (i.e. either abducible atoms or their negation). For simplicity, and without loss of generality, we will assume that exactly one abducible literal is retractible in each integrity constraint. Indeed, any integrity constraint with more than one retractible can be replaced by as many integrity constraints as the number of retractibles, every such integrity constraint having only one retractible.

Given an abductive logic program $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, for each abducible predicate a in $\mathcal{AB}_0$ let a' be a new predicate, which intuitively represents the "complement" of a.

Replacement of positive abducibles by negation as failure literals
For each abducible predicate a in $\mathcal{AB}_0$,

- all the positive conditions $a(t)$ in clauses of $\mathcal{P}$ are replaced by conditions $not\, a'(t)$. (Let $\mathcal{P}^{NAF}$ be the resulting program.)
- all the positive conditions $a(t)$ in integrity constraints in $\mathcal{I}_0$ are replaced by conditions $not\, a'(t)$. (Let $\mathcal{I}_0^{NAF}$ be the resulting set of integrity constraints.)

Note that the transformation does not affect negative occurrences of abducible predicates. Moreover, since all retractibles in $\mathcal{I}_0$ are either abducibles or their negation, each integrity constraint in $\mathcal{I}_0^{NAF}$ is of the form

$$\neg\, [L_1, \ldots, L_{i-1}, \underline{not\, \alpha(t)}, L_{i+1}, \ldots, L_n]$$

where $1 \leq i \leq n$, $n \geq 1$ and α is either an abducible predicate a or the "complement" a' of an abducible predicate a.

Simulation of abduction by negation as failure
Let $\mathcal{P}'_{\mathcal{AB}_0}$ be the set of clauses obtained as follows: For each non-default abducible predicate a in $\mathcal{AB}_0$, $\mathcal{P}'_{\mathcal{AB}_0}$ contains two clauses

$$a(X) \leftarrow not\ a'(X)$$
$$a'(X) \leftarrow not\ a(X)$$

For each default abducible predicate a in $\mathcal{AB}_0$, $\mathcal{P}'_{\mathcal{AB}_0}$ contains the clause

$$a(X) \leftarrow not\ a'(X)$$

Simulation of integrity constraints by program clauses
Let $\mathcal{P}'_{\mathcal{I}_0}$ be the set of clauses obtained as follows: For each integrity constraint

$$\neg[L_1, \ldots, L_{i-1}, \underline{not\, \alpha(t)}, L_{i+1}, \ldots, L_n]$$

in $\mathcal{I}_0^{NAF}$, $\mathcal{P}'_{\mathcal{I}_0}$ contains the clause

$$\alpha(t) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$$

Definition 3.1 Given an abductive logic program $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, the **corresponding program** is $\mathcal{P}' = \mathcal{P}^{NAF} \cup \mathcal{P}'_{\mathcal{AB}_0} \cup \mathcal{P}'_{\mathcal{I}_0}$.

Finally, any query to $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$ needs to be transformed into a corresponding query to $\mathcal{P}'$.

Definition 3.2 Given a query $\mathcal{Q}$ to $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, the **corresponding query**, $\mathcal{Q}'$, is obtained by replacing all positive conditions $a(t)$ in $\mathcal{Q}$, where $a \in \mathcal{AB}_0$, by $not\, a'(t)$.

Example 3.1 The program $\mathcal{P}'_{ec}$ resulting from the transformation is

$$
\begin{array}{lll}
holds\_at(P,\, T_2) & \leftarrow & not\, nohappens(E, T_1), T_1 < T_2, \\
 & & initiates(E,\, P), not\, broken(T_1,\, P,\, T_2) \\
\multicolumn{3}{l}{broken(T_1,\, P,\, T_2) \leftarrow not\ nohappens(E,\, T), terminates(E,\, P), T_1 < T < T_2} \\
\multicolumn{3}{l}{nohappens(E,\, T) \leftarrow preconditions(E,\, P),\ not\, holds\_at(P,\, T)} \\
\multicolumn{3}{l}{happens(E,\, T) \leftarrow not\, nohappens(E,\, T),} \\
\multicolumn{3}{l}{nohappens(E,\, T) \leftarrow not\ happens(E,\, T)} \\
\multicolumn{3}{l}{persists(T_1,\, P,\, T_2) \leftarrow not\, broken(T_1,\, P,\, T_2)}
\end{array}
$$

where *nohappens* and *broken* stand for the complements of *happens* and *persists*, respectively.

The predicate *persists* does not occur anywhere in the conditions of clauses in $\mathcal{P}'_{ec}$; and therefore, if no query to $\mathcal{P}'_{ec}$ contains a call to *persists*, then the clause defining *persists* will never be used. As a result, $\mathcal{P}'_{ec}$ can be simplified by deleting the clause defining *persists*.

Note that variants of the program $\mathcal{P}'_{ec}$ have been used by many authors, e.g. [4], as a formalisation of the event calculus in NLP. Here we have shown how to construct such a program in a systematic manner from a "higher level" specification. Moreover, (see section 5) we show that the transformation from $\langle \mathcal{P}_{ec}, \mathcal{AB}_{ec}, \mathcal{I}_{ec} \rangle$ to $\mathcal{P}'_{ec}$ is correct and complete.

4 An argumentation framework

The correctness and completeness of the transformation can be proved by using a variant of the abstract argumentation frameworks proposed in [1, 2, 17] as a semantics for non-monotonic reasoning in general. 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.*

In such a framework a sentence is a non-monotonic consequence if it follows monotonically from the theory extended by means of an "acceptable" set of assumptions. 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, and the second set is deemed responsible for the violation. Retractibles identify the set as responsible for the violation, as formalised by the following definition of "attack".

Definition 4.1 Given an argumentation framework $\langle \mathcal{T}, \vdash, \mathcal{AB}, \mathcal{IC} \rangle$:

- a set of assumptions $\mathcal{A} \subseteq \mathcal{AB}$ *attacks* an assumption $\delta \in \mathcal{AB}$, if and only if for some integrity constraint $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots \wedge L_n] \in \mathcal{IC}$

  (1) $\mathcal{T} \cup \mathcal{A} \vdash \;\; L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, and

  (2) $\mathcal{T} \cup \{\delta\} \vdash L_i$.

- a set of assumptions $\mathcal{A} \subseteq \mathcal{AB}$ *attacks* another set $\Delta \subseteq \mathcal{AB}$ if and only if $\mathcal{A}$ *attacks* δ, for some assumption δ in Δ.

Note that a set of assumptions *violates* a denial integrity constraint if all the conjuncts in the denial can be derived from the theory together with the set of assumptions; and a set of assumptions *satisfies* a denial integrity constraint if it does not violate it.

If all retractibles in integrity constraints in $\mathcal{IC}$ 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 of $\mathcal{A}$ *attacks* Δ becomes

$2'$) $L_i \in \Delta$.

As we will see below, this is the case for the argumentation frameworks corresponding both to ALP and NLP.

Various notions of "acceptability" can be defined in terms of the same notion of *attack*. Here we mention some of the notions presented in [1, 2]: A set of assumptions which does not *attack* itself is called

- *stable*, if and only if it *attacks* all assumptions it does not contain;

- *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 (with respect to set inclusion) admissible;

- *complete*, if and only if it is admissible and it contains all assumptions it defends against all attacks; (Δ *defends* an assumption δ against an attack $\mathcal{A}$ if and only if $\mathcal{A}$ *attacks* δ and Δ *attacks* $\mathcal{A}$.)

- *grounded*, if and only if it is minimally (with respect to set inclusion) complete.

Note that any stable, admissible, preferred, complete or grounded set of assumptions satisfies all the denial integrity constraints, since any such set is necessarily conflict-free.

Both ALP and NLP can be given a semantics by appropriately applying any of the abstract semantics given above, treating NLP as a special case of ALP. Given an abductive logic program $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, 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 together with all the domain-specific abducibles in $\mathcal{AB}_0$;
- $\mathcal{IC}$ is the set consisting of
  (1) all denials of the form $\neg[A \wedge \underline{not\ A}]$ where A is a variable-free atom,
  (2) all denials of the form $\neg[\underline{A} \wedge not\ A]$ where A is a variable-free non-default abducible atom, and
  (3) all domain-specific denial integrity constraints in $\mathcal{I}_0$.

NLP is the special case of ALP where the assumptions are all variable-free negative literals alone and the integrity constraints are the denials of kind (1) alone. Note that the notion of integrity satisfaction which is implicit in the notion of attack, is compatible with three-valued semantics for NLP, since an integrity constraint $\neg[\underline{A} \wedge not\ A]$ is satisfied by a program extended with a set of assumptions if neither A nor $not\ A$ can be derived.

Many existing semantics for NLP can expressed in argumentation-theoretic terms, as proved in [1, 2]. In particular, stable models correspond to stable sets of assumptions, partial stable models and preferred extensions correspond to preferred sets of assumptions, stationary expansions and complete scenaria correspond to complete sets of assumptions and well-founded semantics corresponds to the grounded set of assumptions. Moreover, various new semantics for the form of ALP we use in this paper are obtained as instances of the abstract notions. Note that, in the well-founded semantics for ALP, all non-default abducibles are undefined, and consequently serve no purpose. An alternative, less sceptical semantics for non-default abducibles in ALP has been defined in [14].

The following example illustrates the notions of attack and of admissibility in the ALP case. The example is the Kautz stolen car problem.

Example 4.1 The problem is to explain that a car is not in a car park at a time t, after having been parked there at an earlier time t_0. Namely, we want to explain $not\ holds\text{-}at(in, t)$ given that

$$\begin{array}{l} happens(park, t_0) \\ t_0 < t \end{array}$$

belong to $\mathcal{P}_{ec}$. Moreover, in addition to these two facts and the persistence axiom, $\mathcal{P}_{ec}$ also contains the facts

$$\begin{array}{l} initiates(park, in) \\ terminates(steal, in) \\ precondition(steal, in). \end{array}$$

This problem illustrates the combination of a narrative with eypothetical reasoning, since the predicate *happens* is both defined by a fact in $\mathcal{P}_{ec}$ and is an abducible in Ab_{ec}. In this case, we can use the modified event calculus.

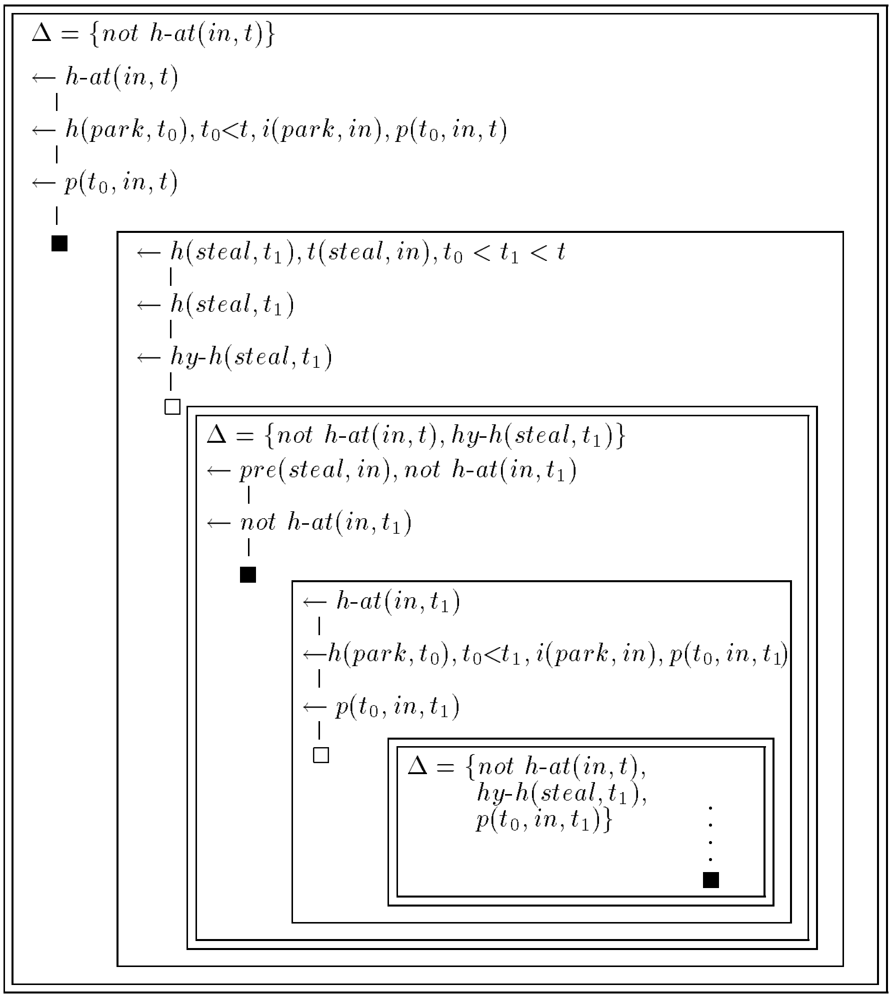

Figure 1: Kautz's stolen car problem, in example 4.1

To simplify the description below, we will assume that the Herbrand universe of $\langle \mathcal{P}_{ec}, \mathcal{AB}_{ec}, \mathcal{I}_{ec} \rangle$ contains a term t_1 and that $t_0 < t_1 < t$.

Figure 1 illustrates the construction of an admissible set of assumptions $\{not\ holds\text{-}at(in, t), hyp\text{-}happens(steal, t_1), persists(t_0, in, t_1)\}$, starting from the initially given assumption *not holds-at*(in, t). The double boxes in the figure represent the construction of attacks using SLD resolution. The single boxes represent the construction of defences against attacks. These extensions are also constructed by SLD resolution. The transition from one box to the next box inside it is done by resolving an assumption against a retractible literal in an integrity constraint.

5 Correctness and completeness

In this section we assume that an abductive logic program $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$ and the corresponding transformed program $\mathcal{P}'$ are given. Note that $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$ and $\mathcal{P}'$ have the same Herbrand universe, but different Herbrand bases, since $\mathcal{P}'$ contains more predicates than $\mathcal{P}$. Let Lit be the set of variable-free literals over the Herbrand base of $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$, and let Lit' be the set of variable-free literals over the Herbrand base of $\mathcal{P}'$. First let us define the correspondence between sets of assumptions for $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$ and sets of assumptions for $\mathcal{P}'$.

Definition 5.1 Given L in Lit and L' in Lit', L and L' are **corresponding literals** (L **corresponds to** L' and L' **corresponds to** L) if and only if

- $L{=}a(t)$ and $L' = not\ a'(t)$, if a is an abducible predicate in $\mathcal{AB}_0$ and a' represents the complement of a, or
- $L' = L$, otherwise.

Given a set of literals $\mathcal{S}$ contained in Lit and a set of literals $\mathcal{S}'$ contained in Lit', $\mathcal{S}$ and $\mathcal{S}'$ are **corresponding sets** ($\mathcal{S}$ **corresponds to** $\mathcal{S}'$ and $\mathcal{S}'$ **corresponds to** $\mathcal{S}$) if and only if for each literal L in $\mathcal{S}$ there is a literal L' in $\mathcal{S}'$ such that L and L' are corresponding literals, and vice versa.

The correctness and completeness of the transformation is a corollary of the following theorem, which establishes a one-to-one correspondence between attacks in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ and attacks in $\mathcal{F}_{\mathcal{P}'}$. The proof is in the appendix.

Theorem 5.1 *Let $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ be the argumentation framework corresponding to $\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle$ and let $\mathcal{F}_{\mathcal{P}'}$ be the argumentation framework corresponding to $\mathcal{P}'$. Given sets of assumptions $\mathcal{A}$ and Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ and $\mathcal{A}'$ and Δ' in $\mathcal{F}_{\mathcal{P}'}$ such that $\mathcal{A}$, $\mathcal{A}'$ and Δ, Δ' are corresponding sets, then $\mathcal{A}$ attacks Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ if and only if $\mathcal{A}'$ attacks Δ' in $\mathcal{F}_{\mathcal{P}'}$.*

Corollary 5.1 Given corresponding sets of assumptions Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ and Δ' in $\mathcal{F}_{\mathcal{P}'}$ Δ is stable (admissible, preferred, complete, grounded) in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ if and only if Δ' is stable (admissible, preferred, complete, grounded, respectively) in $\mathcal{F}_{\mathcal{P}'}$.

The following theorem establishes a one to one correspondence between abductive explanations for queries and abductive explanations for corresponding queries. The proof is in the appendix.

Theorem 5.2 *Given a query $\mathcal{Q}$, corresponding query $\mathcal{Q}'$, set of assumptions Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$, and corresponding set of assumptions Δ' in $\mathcal{F}_{\mathcal{P}'}$,*

$$\mathcal{P} \cup \Delta \vdash \mathcal{Q} \text{ if and only if } \mathcal{P}' \cup \Delta' \vdash \mathcal{Q}'.$$

An important consequence of this theorem and corollary 5.1 is that any proof procedure which answers queries with respect to one of the above semantics for normal logic programs answers queries with respect to the same semantics for the corresponding abductive logic programs, and, less interestingly, vice versa. Therefore, instead of defining new proof procedures for ALP, we can use proof procedures for NLP instead.

6 Comparisons

We use the transformation of Satoh and Iwayama [16] to simulate non-default abduction via negation as failure. Equivalent techniques are used also by Inoue [7] and Pereira, Aparicio and Alferes [14] to eliminate non-default abducibles from extended logic programs augmented with abduction.

Kowalski and Sadri [12] use exceptions, formulated as extended clauses of the form

$$\neg p \leftarrow L_1, \ldots, L_n$$

to override general rules, of the form

$$p \leftarrow K_1, \ldots, K_m$$

They transform rules with exceptions into extended logic programs, by replacing general rules of the form above by clauses of the form

$$p \leftarrow K_1, \ldots, K_m, not\ \neg p$$

In the special case of ALP, where we have an abducible p which is similar to a general rule with conclusion p, the transformation gives $p \leftarrow not\ \neg p$ which treats p as a default abducible. This is similar to our rewriting of p as $not\ p'$.

In an earlier paper [11], Kowalski and Sadri investigated a similar transformation, where, however, exceptions were represented as integrity constraints with retractibles, i.e.

$$\neg[L_1 \wedge \ldots \wedge L_n \wedge \underline{p}].$$

Our simulation of integrity constraints can be viewed as the rewriting of integrity constraints in [11] as exceptions in [12], where explicitly negated literals $\neg p$ are replaced by atoms p'.

In [12] it was shown that the Kowalski-Sadri transformation preserves a variant of the answer set semantics. In this paper we show that all argumentation-theoretic semantics are preserved.

In [4], Denecker and De Schreye present a transformation from abductive logic programs with integrity constraints to abductive logic programs without integrity constraints. Integrity constraints of the form

$$\neg [L_1 \wedge \ldots \wedge L_n]$$

are rewritten as clauses of the form

$$false \leftarrow L_1, \ldots, L_n$$

where $false$ is a new predicate. The transformation allows query evaluation to be performed by using SLDNFA, an abductive procedure for ALP without integrity constraints. For this purpose, queries of the form $\leftarrow \mathcal{Q}$ are transformed into queries of the form $\leftarrow \mathcal{Q}$, *not* $false$ to the transformed program. The original abductive logic program and query are shown to be equivalent to the transformed ones under the completion semantics.

In [3], Chakravarthy, Grant and Minker propose a transformation which also uses integrity constraints to transform programs, but for query optimisation rather than for query evaluation.

7 Conclusions

It can be argued [8, 5, 6, 9, 7] that abductive logic programs are at a higher level of specification than normal logic programs. In this paper we have shown that abductive logic programs can be transformed into normal logic programs. The advantage of the transformation is that proof procedures for NLP can be used for ALP.

We have proved that the transformation is correct and complete for many different semantics, by using a novel technique of showing that there is a one-to-one correspondence between attacks before and after the transformation. To the best of our knowledge, this is the only technique which has been used to prove such a result for many different semantics using only a single proof. The same technique can be used to show that the unfolding transformation also preserves many different semantics (see [17]). Hopefully, other applications of this technique will also be useful in the future.

Acknowledgements

This research was supported by the Fujitsu Research Laboratories. The authors are grateful to Noboru Iwayama, Fariba Sadri and Ken Satoh for helpful discussions.

References

Appendix

As we remarked in section 4, in the argumentation frameworks for both ALP and NLP all retractibles in integrity constraints are assumptions. Therefore, without loss of generality we can use the following notion of *attack*:

- For any sets of assumptions $\mathcal{A}$ and Δ, $\mathcal{A}$ *attacks* Δ if and only if there exists an integrity constraint $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots \wedge L_n]$ such that

  1) $\mathcal{T} \cup \mathcal{A} \vdash\ L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, and

  2$'$) $L_i \in \Delta$.

Note also that we can assume that $\mathcal{P}$, $\mathcal{I}_0$ and $\mathcal{P}'$ are variable-free. If they are not, they can be replaced by all their variable-free instances over their Herbrand universe.

The following lemmas are used to prove the theorem.

Lemma A.1 *Given corresponding sets of assumptions Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ and Δ' in $\mathcal{F}_{\mathcal{P}'}$ and corresponding variable-free literals L in Lit and L' in Lit',*

$$\mathcal{P} \cup \Delta \vdash L \text{ if and only if } \mathcal{P}' \cup \Delta' \vdash L'.$$

Proof:
We prove the only-if half by induction on the length n of the proof of L.
$n = 1$ if and only if either

(i) $L = not\ p(t)$, where p is any predicate, and $L \in \Delta$, or
(ii) $L = a(t)$, where a is an abducible predicate, and $L \in \Delta$, or
(iii) $L = p(t)$, where p is a non-abducible predicate, and $p(t) \in \mathcal{P}$.

In cases (i) and (ii), $\mathcal{P}' \cup \Delta' \vdash L'$, since $L \in \Delta$ if and only if $L' \in \Delta'$, by definition of corresponding literals and sets of literals. In case (iii), $\mathcal{P}' \cup \Delta' \vdash L'$ since, by definition of $\mathcal{P}'$, $p(t) \in \mathcal{P}'$, and, by definition of corresponding literals, $L = L'$.
$n > 1$ if and only if $L = p(t)$, where p is a non-abducible predicate and there exists a clause $L \leftarrow K_1, \ldots, K_m$ in $\mathcal{P}$, with $m \geq 1$, such that $\mathcal{P} \cup \Delta \vdash K_i$ for each $1 \leq i \leq m$, and each such proof requires less than n steps. By definition of $\mathcal{P}'$ there exists a clause $L' \leftarrow K'_1, \ldots, K'_m$ in $\mathcal{P}'$ and $L = L'$. By induction hypothesis $\mathcal{P}' \cup \Delta' \vdash K'_i$ for each $1 \leq i \leq m$. Therefore, $\mathcal{P}' \cup \Delta' \vdash L'$, and the only-if half is proved.
The if half can be proved similarly, by induction.

Lemma A.2 *Given corresponding sets of assumptions Δ in $\mathcal{F}_{\langle \mathcal{P}, \mathcal{AB}_0, \mathcal{I}_0 \rangle}$ and Δ' in $\mathcal{F}_{\mathcal{P}'}$, For each variable-free abducible atom $a \in \mathcal{AB}_0$ and variable-free term t:*

1. *If $\mathcal{P} \cup \Delta \vdash a(t)$ then $\mathcal{P}' \cup \Delta' \vdash a(t)$;*

2. *If a is a non-default abducible, i.e. $a \in \mathcal{AB}_0^{nd}$ and $\mathcal{P} \cup \Delta \vdash not\ a(t)$ then $\mathcal{P}' \cup \Delta' \vdash a'(t)$;*

3. *If $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots L_n]$ is in $\mathcal{I}_0$ and $L_i = a(t)$ ($L_i = not\ a(t)$, resp.) and $\mathcal{P} \cup \Delta \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, then $\mathcal{P}' \cup \Delta' \vdash a'(t)$ ($a(t)$, resp.);*

4. *If* $\mathcal{P}' \cup \Delta' \vdash a(t)$ *then either* $\mathcal{P} \cup \Delta \vdash a(t)$ *or there exists* $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots L_n]$ *in* $\mathcal{I}_0$ *such that* $L_i = \mathit{not}\ a(t)$ *and* $\mathcal{P} \cup \Delta \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$.

5. *If* $\mathcal{P}' \cup \Delta' \vdash a'(t)$ *then* *either* a *is a non-default abducible and* $\mathcal{P} \cup \Delta \vdash \mathit{not}\ a(t)$ *or there exists* $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots L_n]$ *in* $\mathcal{I}_0$ *such that* $L_i = a(t)$ *and* $\mathcal{P} \cup \Delta \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$.

Proof:

1. If $\mathcal{P} \cup \Delta \vdash a(t)$ then, by lemma A.1, $\mathcal{P}' \cup \Delta' \vdash \mathit{not}\ a'(t)$. $\mathcal{P}'$ necessarily contains the clause $a(t) \leftarrow \mathit{not}\ a'(t)$ Therefore $\mathcal{P}' \cup \Delta' \vdash a(t)$;

2. If $\mathcal{P} \cup \Delta \vdash \mathit{not}\ a(t)$ then, by lemma A.1, $\mathcal{P}' \cup \Delta' \vdash \mathit{not}\ a(t)$. Since a is a non-default abducible, $\mathcal{P}'$ necessarily contains the clause $a'(t) \leftarrow \mathit{not}\ a(t)$. Therefore, $\mathcal{P}' \cup \Delta' \vdash a'(t)$;

3. If $\mathcal{P} \cup \Delta \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, then, by lemma A.1, $\mathcal{P}' \cup \Delta' \vdash L'_1, \ldots, L'_{i-1}, L'_{i+1}, \ldots, L'_n$. If $L_i = a(t)$, then $\mathcal{P}'$ contains the clause $a'(t) \leftarrow L'_1, \ldots, L'_{i-1}, L'_{i+1}, \ldots, L'_n$. Therefore, $\mathcal{P}' \cup \Delta' \vdash a'(t)$. If $L_i = \mathit{not}\ a(t)$, then $\mathcal{P}'$ contains the clause $a(t) \leftarrow L'_1, \ldots, L'_{i-1}, L'_{i+1}, \ldots, L'_n$. Therefore, $\mathcal{P}' \cup \Delta' \vdash a(t)$.

4. and 5. Directly by definition of $\mathcal{P}'$.

Proof of theorem 5.1: We prove the theorem by case analysis.

By lemma A.1, $\mathcal{A}$ *attacks* Δ via $\neg[p(t) \wedge \overline{\mathit{not}\ p(t)}]$, where p is an ordinary, non-abducible predicate, if and only if $\mathcal{A}'$ *attacks* Δ' via the same integrity constraint.

By lemmas A.1 and A.2, part (1), if $\mathcal{A}$ *attacks* Δ via $\neg[a(t) \wedge \overline{\mathit{not}\ a(t)}]$, where a is an abducible predicate, then $\mathcal{A}'$ *attacks* Δ' via $\neg[a(t) \wedge \overline{\mathit{not}\ a(t)}]$.

By lemmas A.1 and A.2, part (2), if $\mathcal{A}$ *attacks* Δ via $\neg[\underline{a(t)} \wedge \overline{\mathit{not}\ a(t)}]$, where a is a (non-default) abducible predicate, then $\mathcal{A}'$ *attacks* Δ' via $\neg[a'(t) \wedge \underline{\mathit{not}\ a'(t)}]$.

By lemmas A.1 and A.2, part (3), if $\mathcal{A}$ *attacks* Δ via $\neg[L_1 \wedge \ldots \wedge \underline{L_i} \wedge \ldots \wedge L_n]$, where $L_i = a(t)$ ($L_i = \mathit{not}\ a(t)$, resp.) and a is an abducible predicate, then $\mathcal{A}'$ *attacks* Δ' via $\neg[a'(t) \wedge \underline{\mathit{not}\ a'(t)}]$ (via $\neg[a(t) \wedge \overline{\mathit{not}\ a(t)}]$, resp.).

By lemmas A.1 and A.2, part (4), if $\mathcal{A}'$ *attacks* Δ' via $\neg[a(t) \wedge \overline{\mathit{not}\ a(t)}]$, where a is an abducible predicate, then $\mathcal{A}$ *attacks* Δ via the same integrity constraint or $\mathcal{A}$ *attacks* Δ via $\neg[L_1 \wedge \ldots \wedge \overline{\mathit{not}\ a(t)} \wedge \ldots \wedge L_n]$.

By lemmas A.1 and A.2, part (5), if $\mathcal{A}'$ *attacks* Δ' via $\neg[a'(t) \wedge \overline{\mathit{not}\ a'(t)}]$, where a is an abducible predicate, then $\mathcal{A}$ *attacks* Δ via $\neg[\underline{a(t)} \wedge \overline{\mathit{not}\ a(t)}]$, or $\mathcal{A}$ *attacks* Δ via $\neg[L_1 \wedge \ldots \wedge \underline{a(t)} \wedge \ldots \wedge L_n]$.

Since all possible cases in which $\mathcal{A}$ can *attack* Δ and $\mathcal{A}'$ can *attack* Δ' are considered above, theorem 5.1 is proved.

Proof of theorem 5.2: Directly from lemma A.1.