On Quantified Negative Queries

Alessandra Di Pierro
Dipartimento di Matematica e Informatica,
Università di Udine,
Via delle Scienze 206, 33100 Udine, Italy
dipierro@dimi.uniud.it

Włodzimierz Drabent
IPIPAN, Polish Academy of Sciences,
Ordona 21, Pl – 01-237 Warszawa, Poland
wdr@ipipan.waw.pl

Abstract

Given a logic program P and a goal G, we introduce a notion which states when an SLD-tree for $P \cup \{G\}$ *instantiates* a set of variables V *with respect to* another one, W. We call this notion *weak instantiation*, as it is a generalization of the *instantiation* property introduced in [3]. A negation rule based on instantiation, the so-called Negation As Instantiation rule (NAI), allows for inferring existentially closed negative queries, that is formulas of the form $\exists\neg Q$, from logic programs. We show that, by using the new notion, we can infer a larger class of negative queries, namely the class of the queries of the form $\forall_W \exists_V \neg Q$ and of the form $\forall_W \exists_V \forall_U \neg Q$, where U is the set of the remaining variables of Q.

1 Introduction

In order to infer negative literals from a logic program, Clark introduced the Negation As Failure (NAF) rule and defined its declarative semantics in terms of the completion, Comp(P), of a program P ([2]). However, this rule allows us to infer only a small part of the negative information which could be drawn from a program, namely the universally closed negative literals (more generally, universally closed negated queries). NAF works as a test: For a program P and a query Q it is able to check whether Comp(P) $\models \neg Q$ (or, equivalently, Comp(P) $\models \forall\neg Q$).

One generalization of NAF is constructive negation: Methods of finding instances $Q\theta$ of a given query Q such that $\neg Q\theta$ is a logical consequence of Comp(P).

Recently another generalization of negation as failure was proposed. It is Negation As Instantiation (NAI) rule [3], which allows us to derive existentially closed negative literals (or negated queries). The semantical justification of this inference rule is still the completion of the program, but over an extended language L^*, containing infinitely many constant symbols, denoted by $\mathrm{Comp}_{L^*}(P)$. As shown in [3], if, for a program P and a query Q, there exists an SLD-tree that instantiates variables $\overline{x}$, then $\mathrm{Comp}_{L^*}(P) \models \exists\neg Q$ and, moreover, $\mathrm{Comp}_{L^*}(P) \models \exists\overline{x}\neg Q$.[1] Actually, as shown in Section 3.1, the existence of such a tree implies that $\mathrm{Comp}_{L^*}(P) \models \exists\overline{x}\forall\overline{y}\neg Q$ (where $\overline{y} = var(Q)\setminus\overline{x}$). Moreover, we will show that the reverse (completeness) holds too, in the case of definite programs and queries. However, if we are interested in deriving formulae of the form $\forall\exists\overline{x}\neg Q$, NAI is bound to be incomplete, even for positive programs and queries: There are programs and goals for which $\mathrm{Comp}_{L^*}(P) \models \exists\overline{x}\neg Q$, but there does not exist a tree for Q that instantiates $\overline{x}$.

In this paper we generalize the notion of instantiation introduced in [3]. With the new notion of weak instantiation we are able to derive conclusions of the form $\mathrm{Comp}_L(P) \models \exists\overline{x}\neg Q$ also in cases when $\mathrm{Comp}_L(P) \not\models \exists\overline{x}\forall\overline{y}\neg Q$. Thus, we define a negation rule based on the new notion of weak instantiation and prove both its soundness and completeness with respect to $\mathrm{Comp}_L(P)$. For the semantics to be correct, a different extension L of the language of the program is needed.

As a first step, in this paper we consider positive programs and queries only.

2 Preliminaries

We refer to [1, 9] for the basics of logic programming, including the negation as failure rule and the Clark completion semantics. For the purposes of this paper, a program P is a definite Horn clause program and a query Q is a conjunction of atoms.

A language L consists of the (well-formed) logical formulas built out of three disjoint sets of symbols: a set of *function symbols*, a set of *predicate symbols* and a set *Var* of *variable symbols*. Each function and predicate symbol is associated with a number representing its arity. Function symbols whose arity is 0 are also called *constant symbols*. We denote by *Term* the set of terms $t, u \ldots$ built of *Var* and the set of function symbols.

A substitution is a mapping $\theta : Var \to Term$ such that the set $Dom(\theta) = \{x \mid \theta(x) \neq x\}$ (*domain* of θ) is finite. A substitution θ is also denoted by the set $\{x_1/t_1, \ldots, x_n/t_n\}$, where $\{x_1, \ldots, x_n\} = Dom(\theta)$ and $t_i = \theta(x_i)$, $i = 1, \ldots, n$. The empty substitution is denoted by ε. The notion of substitution can be naturally extended to terms. The composition $\theta\sigma$ of the substitutions θ and σ is defined as the functional composition. The pre-ordering $\leq$ on substitutions is such that $\theta \leq \sigma$ iff there exists θ' such that $\theta\theta' = \sigma$.

The application of the substitution θ to the atom A is denoted by $A\theta$. The relation $A \leq A'$ (A is less instantiated than A') holds iff there exists θ such that $A\theta = A'$. A is called a variant of A' if there exist θ and θ' such that $A\theta = A'$, and $A'\theta' = A$; θ (θ') is called a renaming for A (A').

For a formula F, we use $\forall F$, $\exists F$ to denote respectively $\forall x_1, \ldots, \forall x_n F$ and $\exists x_1, \ldots, \exists x_n F$, where $x_1, \ldots, x_n$ are all the free variables occurring in F.

[1] Deriving this formula is called in [3] "amalgamation of NAF and NAI". For the definition of an instantiating tree see section 3 below.

For a given language L, $\mathrm{Comp}_L(P)$ is the Clark completion of the program P over the language L, and CET_L the Clark equality theory for L, included in $\mathrm{Comp}_L(P)$. When the language is known, the Clark completion of P is indicated by $\mathrm{Comp}(P)$. Usually, it is the language whose function and predicate symbols are those occurring in the program P and the considered query Q. We will refer to it as the language of P and Q.

We will use the notation $|\overline{x}|$ to indicate the length of the tuple $\overline{x}$, and the symbol '$\equiv$' for syntactic identity (of terms, etc.).

3 Negation As Instantiation

The basic observation leading to the idea of *negation as instantiation* is that fresh constant symbols, that is constant symbols which do not belong to the language of the program and the goal under consideration, can play the role of existentially quantified variables. In fact, if every branch of an SLD-tree for $P \cup \{\leftarrow A\}$ either fails or instantiates some of the variables of A, then for a ground substitution η instantiating all variables to distinct fresh constants, the corresponding SLD-tree for $P \cup \{\leftarrow A\eta\}$ finitely fails. Thus, by the soundness of NAF, we can deduce $\neg A\eta$, and finally $\exists\neg A$. Therefore, it is sufficient to extend the underlying language by infinitely many constant symbols, to obtain the appropriate reference theory for validly inferring formulas like $\exists\neg A$. As shown in [3], the Negation As Instantiation (NAI) rule is, indeed, sound and complete with respect to the completion of P, $Comp_{L^*}(P)$, over the extended language L^*. The formal definition of the notion of *instantiation*, used in the above informal description of NAI, is given by means of the property *Inst*.

Definition 3.1 (Instantiation) *Let $V = \{x_1, \ldots, x_n\}$, θ be a substitution and p a predicate symbol of arity n.*

$$Inst(\theta, V) \Leftrightarrow p(x_1, \ldots, x_n)\theta \not\leq p(x_1, \ldots, x_n).$$

In words, the property $Inst(\theta, V)$ holds iff θ is not a renaming for $p(x_1, \ldots, x_n)$, ($p(x_1, \ldots, x_n)\theta$ is not a variant of $p(x_1, \ldots, x_n)$).

Note that if θ instantiates V, then it instantiates any $V' \supseteq V$. An equivalent definition that will come in handy in the proofs and definitions given further on, is as follows.

Proposition 3.2

$$Inst(\theta, V) \quad \textit{iff} \quad \begin{array}{l} -\ \textit{there exists } x \in V \textit{ such that } x\theta \notin Var, \textit{or} \\ -\ \textit{there exist } x, y \in V \textit{ such that } x\theta \equiv y\theta \in Var. \end{array}$$

Proof The (if) part is obvious. For the (only if) part, assume by contradiction that for all $x, y \in V$, $x\theta$ and $y\theta$ are two distinct variables. Then, $\neg Inst(\theta, V)$ holds.

□

Definition 3.3 (Instantiating SLD-tree) *Let P be a program, Q a query, $V \subseteq var(Q)$. Let TR be an SLD-tree for P and Q. Then TR instantiates V iff for every branch ξ of TR one of the following holds:*

- *ξ finitely fails, or*
- *$Inst(\theta_1 \cdots \theta_k, V)$ holds, where $\theta_1 \cdots \theta_k$ are the substitutions labeling the first k edges of ξ, for some $k \geq 1$.*

This definition differs from (but is equivalent to) the original one. Note that if *TR* instantiates V then the set of those its nodes whose accumulated substitutions do not instantiate V is finite (by König's lemma). Thus in the definition above there exists an n such that, for all branches, $n > k \geq 1$. So it is sufficient to inspect the tree only to some restricted depth, in order to check that it instantiates V. This is why the original definition calls such tree "finitely instantiating".

The operational semantics for NAI is defined by the Failure by Finite Instantiation set (the *FFI* set), consisting of all atoms A, for which there exists an SLD-tree which instantiates the variables occurring in A in the the sense explained above. This has been shown to correspond to the set of atoms whose existentially quantified negation is a logical consequence of $Comp_{L^*}(P)$, where L^* is the language of P enriched with infinitely many new constant symbols.

Theorem 3.4 (*Soundness and completeness of the NAI rule,* [*3*])

$$Comp_{L^*}(P) \models \exists \neg A \text{ iff } A \in FFI.$$

3.1 Strong soundness and completeness of NAI

Actually, the existence of an instantiating tree for P and Q implies more than just $\exists \neg Q$. In [3, 4] it is shown that it implies $\exists \overline{x} \neg Q$ provided the tree instantiates $\overline{x}$. (This usage of NAI is called there amalgamation of NAI and NAF). We show something more:

Theorem 3.5 (Strong soundness of NAI) *Let P be a program, Q a query, $\overline{x}$ a tuple of variables occurring in Q. Let $\overline{y}$ be the remaining variables of Q. If there exists an SLD-tree for P and Q that instantiates $\overline{x}$ then*

$$Comp_{L^*}(P) \models \exists \overline{x} \forall \overline{y} \neg Q$$

For a proof of this theorem, see its generalization, Theorem 4.10. Clearly, $\exists \overline{x} \forall \overline{y} \neg Q$ implies $\forall \overline{y} \exists \overline{x} \neg Q$, but not vice-versa. Hence, the notion of an instantiating tree is not sufficient to infer formulas of the form $\forall \overline{y} \exists \overline{x} \neg Q$. The way of inferring such formulas will be presented in the next section.

Now, we show that the reverse of Theorem 3.5 holds for the NAI rule.

Theorem 3.6 (Strong completeness of NAI) *Let P be a program, Q a query, and $\overline{x} \subseteq var(Q)$. Let $\overline{z}$ be the remaining variables of Q.*
If $Comp_{L^*}(P) \models \exists \bar{x} \forall \overline{z} \neg Q$, then there exists an SLD-tree for $P \cup \{\leftarrow Q\}$ which instantiates $\bar{x}$.

For a proof of Theorem 3.6, observe that an SLD-tree for $P \cup \{\leftarrow A\}$ which instantiates $\bar{x}$ is an SLD-tree for $P \cup \{\leftarrow A\}$ which instantiates $\bar{x}$ w.r.t. the empty set, in the sense explained in Section 4.1. Then, Theorem 4.13 applies.

4 A new notion of instantiation

The notion of instantiation defined in [3] does not allow to derive as much negative information as possible. There exist queries Q for which $\exists\overline{x}\neg Q$ is true in the completion semantics (equivalently, $\forall\exists\overline{x}\neg Q$ is true) and yet there are no SLD-trees for Q that instantiate $\overline{x}$.

Example 4.1 *Consider the program*

$$P \quad = \quad \{ \quad r(z,z) \leftarrow \quad \}$$

Observe that $Comp_L(P) \models \exists x \neg r(x,y)$. However, $Comp_L(P) \not\models \exists x \forall y \neg r(x,y)$. Hence, by the soundness theorem (Theorem 3.5) there does not exist an SLD-tree for $P \cup \{\leftarrow r(x,y)\}$ that instantiates x. So, we are unable to infer $\exists x \neg r(x,y)$.

Example 4.2 *Consider the program*

$$P' \quad = \quad \{ \quad r(z,f(z)) \leftarrow \quad \}$$

Observe that $Comp_L(P) \models \exists x \neg r(x,y)$. Again, the amalgamation rule cannot infer the formula $\exists x \neg r(x,y)$, as the SLD-tree for $P \cup \{\leftarrow r(x,y)\}$ does not instantiate $\{x\}$, according to the definition of Inst.

The point is that a substitution which links the existentially quantified variables $\overline{x}$ and the free variables $\overline{y}$, or the free variables with some terms containing $\overline{x}$, should be considered as instantiating $\overline{x}$. In other words, we want to define a notion of "relative instantiation" such that the substitution $\vartheta = \{x = z, y = z\}$ of Example 4.1, as well as the substitution $\vartheta' = \{x = z, y = f(z)\}$ of Example 4.2, turn out to be instantiating $\{x\}$ with respect to {y}.

4.1 Weak instantiation

Definition 4.3 (Weak instantiation) *Let $V = \{x_1, \dots, x_n\}$, $W = \{y_1, \dots, y_m\}$ be two disjoint sets of variables, and let θ be a substitution. We say that θ instantiates V w.r.t. W, in symbols $Winst(\theta, V, W)$, iff*

- *$Inst(\theta, V)$, or*

- *$V\theta$ and $W\theta$ have a common variable.*

Note that for $W = \emptyset$, $Winst(\theta, V, W)$ is equivalent to $Inst(\theta, V)$, (weak instantiation subsumes instantiation).

The following proposition provides an alternative definition of weak instantiation.

Proposition 4.4 *A substitution θ instantiates V w.r.t. W iff*

- *there exists $x \in V$ such that $x\theta$ is not a variable, or*

- *there exist $x, y \in V$ such that $x\theta \equiv y\theta$ is a variable, or*

- *there exist $x \in V$ and $y \in W$ such that $x\theta$ is a variable occurring in $y\theta$.*

Definition 4.5 *Let P be a program, G a goal, $V, W \subseteq Var$ two disjoint sets of variables. Let TR be an SLD-tree for $P \cup \{G\}$. Then TR instantiates V w.r.t. W iff for every branch ξ of TR one of the following holds:*

- *ξ finitely fails, or*
- *$Winst(\theta_1 \cdots \theta_k, V, W)$ is true, where $\theta_1, \ldots, \theta_k$ are the substitutions labeling the first k edges of ξ for some $k \geq 1$.*

TR is called a *weakly instantiating* tree for $P \cup \{G\}$.
Observe that this definition subsumes both that of a finitely failed SLD-tree, ([1]) and that of an instantiating SLD-tree ([3]). In fact, if $V \cup W$ is the set of the variables occurring in Q, then the case $V = \emptyset$ corresponds to the case of a finitely failed tree (the predicate *Winst* is *false*); on the other hand, when $W = \emptyset$ a weakly instantiating tree is just a finitely instantiating tree under the Definition 3.2 of [3].

4.2 Negation by Weak Instantiation

The notion of weak instantiation can be used to derive negative information from programs. Namely, for a program P, a query Q and disjoint tuples $\overline{x}$, $\overline{y}$ of variables of Q, if an SLD-tree for P and Q instantiates $\overline{x}$ w.r.t. $\overline{y}$ then we can infer $\exists\overline{x}\neg Q$. This is justified by the fact that the formula $\exists\overline{x}\neg Q$ is true in the completion semantics of P. For the details see Theorem 4.10. We call this rule the Negation by Weak Instantiation (NWI) rule. We show some examples to clarify its possible use.

Example 4.6 *Let plus be a predicate whose third argument is the sum of the others. It can be defined by the following program*

$$PLUS = \{ \; plus(x, 0, x) \leftarrow, \quad plus(x, s(y), s(z)) \leftarrow plus(x, y, z) \quad \}.$$

$PLUS \cup \{\leftarrow plus(x, s^2(0), y)\}$ has an SLD-tree where the goal variable x is instantiated w.r.t. y (y results instantiated to $s^2(x)$). From the NWI rule, we can derive $\exists x \neg plus(x, s^2(0), y)$. Indeed, $Comp(PLUS) \models \exists x \neg plus(x, s^2(0), y)$.

Note that x is not instantiated according to the original notion of instantiation. In fact, in this case, the formula $\exists x \forall y \neg plus(x, s^2(0), y)$ is not true.

Now we show an example where the NWI rule behaves like the amalgamation rule.

Example 4.7 *Consider the program $PLUS$ of Example 4.6.*
We have that $Comp(PLUS) \models \exists x \neg plus(x, y, s^n(0))$. From the NWI rule, we can derive $\exists x \neg plus(x, y, s^n(0))$; in fact, $PLUS \cup \{\leftarrow plus(x, y, s^n(0))\}$ has an SLD-tree where the goal variable x results instantiated w.r.t. y.

Note that in the latter example, $Inst(\theta_i, \{x\})$ holds for all $i = 1, \dots, n$, where θ_i is the composition of the substitutions labeling the first $i + 1$ edges of the i-th branch. Indeed the formula

$$\exists x \forall y \neg plus(x, y, s^n(0))$$

(hence $\forall y \exists x \neg plus(x, y, s^n(0))$) is true.

The NWI rule can be efficiently implemented by suitably extending the implementation of the NAF rule. In Prolog this is typically obtained by means of a *cut–fail* combination. The well-known program is:

$$\begin{array}{lll} \texttt{not}(G) & \leftarrow, & \\ \texttt{not}(G) & \leftarrow & G, !, fail. \end{array}$$

In order to implement NWI, the idea is as follows. For a goal $G(\overline{x}, \overline{y})$, generate $|\overline{x}|$ new function symbols $\overline{f}$ of arity $|\overline{y}|$, and then apply `not` to the goal G' obtained from G by replacing all occurrences of the variables $\overline{x}$ by $\overline{f(\overline{y})}$.

Note that, as in the NAF case, this implementation works correctly provided that the *cut–fail* combination is applied to *ground* goals. This requirement is always satisfied in the case of the NAI rule, as *all* variables of the goal are replaced by new constant symbols.

4.3 Correctness of NWI

A natural way to prove soundness of negation as weak instantiation seems as follows. From an SLD-tree for A instantiating $\overline{x}$ w.r.t. $\overline{y}$ construct a finitely failed tree for $A\{\overline{x}/\overline{f}(\overline{y})\}$, where $\overline{f}$ are $|\overline{x}|$ new function symbols of arity $|\overline{y}|$. However, we found the technical details of the proof difficult and here we present another proof.

Lemma 4.8 *Let Q be a node of an SLD-tree, and let $Q_1, \dots, Q_n$, $(n \geq 0)$ be its children. Then*

$$Comp(P) \models Q \leftrightarrow \bigvee_{i=1}^{n} \exists \overline{w} \; (\overline{v} = \overline{v}\theta_i \wedge Q_i)$$

where θ_i are the mgu's corresponding to Q_i, $\overline{w}$ are the "new" variables and $\overline{v}$ are "all the variables".
(Here $\overline{v} = \overline{v}\theta_i$ is a conjunction of equalities).

This is a version of Lemma 15.3 of [9] and of Lemma 4.1 of [5], and the proof is similar.

The soundness of negation as weak instantiation is implied by the following lemma.

Lemma 4.9 *Let P be a program, Q a query, $\overline{x}, \overline{y}$ a sequence of distinct variables. Let language L_0 contain exactly the functors of P and Q. Assume that there exists an SLD-tree for P and Q weakly instantiating $\overline{x}$ w.r.t. $\overline{y}$ such that variables of $\overline{x}, \overline{y}$*

are not used in the resolution[2]*. Then in any model of Comp*(P) *formula* $\neg Q$ *is true in any valuation* ν *in which the values of* $\overline{x}$ *are*
1. distinct,
2. "new", i.e. not in the range of the interpretations of the functors from L_0*,*
3. for every $x \in \overline{x}$ *and* $y \in \overline{y}$ *and for any term* $t[x]$ *of language* L_0 *and containing* x*, the value* $\nu(y)$ *differs from* $\mu(t[x])$ *for any valuation* μ *such that* $\mu(x) = \nu(x)$*.*

Proof Without lack of generality we may assume that all the variables are ordered and if an mgu contains a binding v/w then w precedes v in the ordering. We will also assume that the variables of $\bar{x}$ precede those of $\bar{y}$ in this ordering and those of $\bar{y}$ precede the remaining ones.

The proof is by induction on the number of nodes in the tree for which the corresponding accumulated substitution does not instantiate $\overline{x}$ w.r.t. $\overline{y}$. (Let us call such nodes non-instantiating). It follows from König's lemma that this number is finite.

Consider an arbitrary model of Comp(P). (We will write $\models_\nu \neg Q$, not mentioning the model). Consider the root Q of the SLD-tree, its children $Q_1, \ldots, Q_n$ $(n \geq 0)$ and the corresponding mgu's $\theta_1, \ldots, \theta_n$. Assume that the lemma holds for all the trees with the number of non-instantiating nodes less than the number of such nodes in the tree under consideration. Then $\models_\nu \neg Q$ iff

$$\models_\nu \bigwedge_i \forall \overline{w}(\neg(\overline{v} = \overline{v}\theta_i) \vee \neg Q_i) \tag{1}$$

where $\overline{w}$ and $\overline{v}$ are as in the previous lemma. For $i = 1, \ldots, n$ we have two cases:

1. If θ_i instantiates $\overline{x}$ w.r.t. $\overline{y}$, we show that $\models_\nu \forall \overline{w} \neg(\overline{v} = \overline{v}\theta_i)$. We have three cases:

   (a) For some x from $\overline{x}$, term $x\theta_i$ is not a variable. Equation $x = x\theta_i$ is a conjunct from the conjunction of equations $\overline{v} = \overline{v}\theta_i$. As the main functor of $x\theta_i$ occurs in P or Q, the equation is false in any valuation in which the value of x is the same as in ν.

   (b) For some x, x' from $\overline{x}$, $x\theta_i$ and $x'\theta_i$ are the same variable. In any valuation that differs from ν only on variables from $\overline{w}$, the values of x and x' are distinct (by condition 1). So $x = x\theta_i \wedge x' = x'\theta_i$ is false is false in any such valuation.

   (c) For some x from $\overline{x}$ and y from $\overline{y}$, $x\theta_i$ is a variable occurring in term $y\theta_i$. We may assume that $x\theta_i \equiv x$, otherwise $x\theta_i$ is another variable from $\overline{x}$ and the previous case applies. Now by condition 3, in any valuation that differs from ν only on variables from $\overline{w}$, the value of y differs from the value of $y\theta_i$. So the equation $y = y\theta_i$ is false.

[2] No variable of $\overline{x}, \overline{y}$ occurs in the renamed clauses used in the SLD-derivations that are the branches of the tree.

2. θ_i does not instantiate $\overline{x}$ w.r.t. $\overline{y}$. Note that $\overline{x}\theta_i \equiv \overline{x}$. First we show that the subtree rooted at Q_i instantiates $\overline{x}$ w.r.t. $vars(\overline{y}\theta_i)$.

   Consider any non failing branch of the subtree. It has a node with the accumulated substitution φ such that $\theta_i\varphi$ instantiates $\overline{x}$ w.r.t. $\overline{y}$. So there exists a variable x from $\overline{x}$ such that either (a) $x\theta_i\varphi \equiv x\varphi$ is not a variable or (b) $x\theta_i\varphi \equiv x'\theta_i\varphi$ is a variable for some $x' \in \overline{x}$, so $x\varphi \equiv x'\varphi$ is a variable; or (c) $x\theta_i\varphi \equiv x\varphi$ is a variable that occurs in $y\theta_i\varphi$, for some $y \in \overline{y}$. Hence φ instantiates $\overline{x}$ w.r.t. $vars(\overline{y}\theta_i)$.

   Consider a valuation π that differs from ν only on the variables from $\overline{w}$. It remains to show that
   $$\models_\pi \neg(\overline{v} = \overline{v}\theta_i) \vee \neg Q_i.$$
   Assume that $\models_\pi \overline{v} = \overline{v}\theta_i$ (otherwise the conclusion is immediate). Obviously, π satisfies conditions 1 and 2 of the lemma.

   Assume that π does not satisfy condition 3. So $\pi(z) = \mu(t[x])$ for some x from $\overline{x}$, some variable z occurring in $\overline{y}\theta_i$, some term $t[x]$ and some valuation μ such that $\mu(x) = \pi(x) = \nu(x)$. Let $s[z]$ be a term from the tuple $\overline{y}\theta_i$, such that z occurs in $s[z]$. $s[z] \equiv y\theta_i$, for some $y \in \overline{y}$.

   We may assume that $vars(s[z]) \cap vars(t[x]) \subseteq \{x\}$. (Otherwise we can rename the variables of $t[x]$ and modify μ accordingly). Let μ' be a valuation such that $\mu'(s[z]) = \pi(s[z])$ and $\mu'(t[x]) = \mu(t[x])$. Such a valuation exists as $\pi(x) = \mu(x)$. Now we have $\mu'(x) = \nu(x)$ and $\mu'(z) = \pi(z) = \mu(t[x]) = \mu'(t[x])$. Then $\nu(y) = \pi(y) = \pi(s[z]) = \mu'(s[z]) = \mu'(s[t[x]])$; contradiction with condition 3 for ν.

   So π satisfies conditions 3 of the lemma. By the inductive assumption, $\models_\pi \neg Q_i$.

Thus, we showed that each conjunct of the conjunction in (1) is true in ν. This completes the proof. □

Theorem 4.10 (Strong soundness) *Let P be a program, Q a query, $\overline{x}$,$\overline{y}$ a sequence of distinct variables occurring in Q. Let $\overline{z}$ be the remaining variables of Q. If there exists an SLD-tree for P and Q weakly instantiating $\overline{x}$ w.r.t. $\overline{y}$, then*

$$Comp_L(P) \models \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$$

where the set of functors of L contains the functors of P and of Q and $|\overline{x}|$ distinct $|\overline{y}|$-ary functors that do occur neither in P nor in Q.

Obviously, $\mathrm{Comp}_L(P) \models \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$ implies $\mathrm{Comp}_L(P) \models \exists\overline{x}\neg Q$.

Proof
From Lemma 4.9. Note that any model of $\mathrm{Comp}_L(P)$ is a model of $\mathrm{Comp}(P)$. We show that for any model of CET_L, for any valuation ν_0 there exists a valuation ν_1 that agrees with ν_0 on the variables outside of $\overline{x}$, such that any valuation ν that agrees with ν_1 on the variables outside of $\overline{z}$ satisfies the conditions of the lemma.

Let $\overline{x} = x_1, \ldots, x_k$, let $f_1, \ldots, f_k$ be the "new" functors. Consider an arbitrary valuation ν_0. Let $\nu(x_i) = \nu_0(f_i(\overline{y}))$, for $i = 1, \ldots, k$; let $\nu(z)$ be arbitrary for any z from $\overline{z}$ and let $\nu(v) = \nu_0(v)$ for any other variable v.

Conditions 1 and 2 of the lemma obviously hold. Consider an x_i, a variable y from $\overline{y}$ and a term $t[x_i]$. Let μ be a valuation such that $\mu(x_i) = \nu(x_i)$. We have to show that $\nu(y) \neq \mu(t[x_i])$.

Consider a renaming ρ such that $\overline{y}\rho$ is a tuple of variables not occurring elsewhere. Take a valuation μ' that agrees with μ on the variables of $t[x_i]$ and such that $\mu'(v\rho) = \nu(v)$ for any $v \in \overline{y}$. So $\mu'(x_i) = \mu(x_i) = \nu(x_i)$ and $\mu'(f_i(\overline{y}\rho)) = \nu(f_i(\overline{y}))$. From this we obtain $\mu'(x_i) = \mu'(f_i(\overline{y}\rho))$, as $\nu(x_i) = \nu(f_i(\overline{y}))$. Now we have $\mu(t[x_i]) = \mu'(t[x_i]) = \mu'(t[f_i(\overline{y}\rho)]) \neq \mu'(y\rho)$ (from CET_L). As $\mu'(y\rho) = \nu(y)$, we obtain $\mu(t[x_i]) \neq \nu(y)$.

From the lemma $\neg Q$ is true for valuation ν. $\square$

Note that the requirement that the variables of $\overline{x}, \overline{y}$ occur in Q can be weakened. It is sufficient to require that these variables are not used in the resolution in the tree. Also the requirement on the language can be modified. For instance, instead of $|\overline{x}|$ new $|\overline{y}|$-ary functors we may use $|\overline{x}|$ new constants and a new binary functor.

Note also that Theorem 3.5 (strong soundness of negation as instantiation) follows immediately from Theorem 4.10 with $\overline{y} = \emptyset$. In this case, the language L contains $|\overline{x}|$ distinct constant symbols. Thus L is indeed a subset of the language L^* used in Section 3.

4.4 Completeness of NWI

In this section we show that negation by weak instantiation is complete w.r.t. $\mathrm{Comp}_L(P)$, for any fair selection rule. In contrast with the correctness theorem, this result holds for any L containing the symbols of the program and the query. In our proof we use the Lloyd-Topor transformation described in [10] and a constructive negation method (the SLDFA-resolution of [5]), which is sound and complete for the Kunen semantics.

Definition 4.11 *Let P be a program , Q a query, $F = \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$, and p, q, r new predicate symbols. Consider the following general logic program*

$$P' = P \cup \{\ p \leftarrow \neg q(\overline{y}),\quad q(\overline{y}) \leftarrow \neg r(\overline{x}, \overline{y}),\quad r(\overline{x}, \overline{y}) \leftarrow Q\ \}$$

We call P' the Lloyd-Topor transformation of P with respect to F.

Clearly, by the definition of the completion we have that $\neg p \leftrightarrow \neg\exists\overline{y}\neg\exists\overline{x}\neg\exists\overline{z}Q \leftrightarrow \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$ holds w.r.t. the new program (it is a logical consequence of $\mathrm{Comp}(P')$).

The completeness of the negation by weak instantiation follows by the next lemma, whose proof is omitted due to space limitation.

Lemma 4.12 *Let P be a program, Q a conjunction of atoms, $\overline{x}, \overline{y}$ a sequence of distinct variables occurring in Q, and $\overline{z}$ the remaining variables of Q. Let F be $\forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$ and P' the Lloyd-Topor transformation of P with respect to F.*

If there exists an SLDFA finitely failed tree for P' and query p then there exists an SLD-tree for P and Q (via the same selection rule) that weakly instantiates $\overline{x}$ w.r.t. $\overline{y}$.

Theorem 4.13 (Completeness of NWI) *Let P be a program, Q a query, $\overline{x}, \overline{y}$ a sequence of distinct variables occurring in Q, and $\overline{z}$ the remaining variables of Q. Let L be any language whose set of function symbols contains those of the language of P and Q, and has at least two elements.*
If $Comp_L(P) \models \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$, then for any fair selection rule there exists an SLD-tree for P and Q weakly instantiating $\overline{x}$ w.r.t. $\overline{y}$.

Proof
$\mathrm{Comp}_L(P) \models \forall\overline{y}\exists\overline{x}\forall\overline{z}\neg Q$ iff $\mathrm{Comp}(P') \models \neg p$ iff $\mathrm{Comp}(P') \models_3 \neg p$ (as P' is call consistent and query $\neg p$ is strict with respect to P').

By the completeness of the SLDFA-resolution (cf. [5, Theorem 5.1]), if $\mathrm{Comp}(P') \models_3 \neg p$ then for any fair selection rule there exists an SLDFA finitely failed tree for p. Then, by Lemma 4.12, for the same selection rule there exists an SLD-tree for P and Q that weakly instantiates $\overline{x}$ w.r.t. $\overline{y}$.

□

5 Conclusions and future work

In this paper we studied what it means that in an SLD-tree certain variables are instantiated in a certain way. This work is a continuation of [3, 4]. First we showed that if an SLD-tree with the root Q instantiates (in the sense of [3]) some variables $\overline{x}$, then this implies not only $\exists\overline{x}\neg Q$, but also $\exists\overline{x}\forall\overline{z}\neg Q$ (where $\overline{z}$ are the remaining variables of Q; cf. Theorem 3.5). Then, we introduced a new notion of weak instantiation that makes it possible to derive formulae of the form $\exists\overline{x}\neg Q$ even when $\exists\overline{x}\forall\overline{z}\neg Q$ does not hold. The semantics of reference is given by Clark's completion over a certain extended language. We proved both soundness and completeness of negation as weak instantiation.

In line with the approach of [6, 7], the Failure by Weak Instantiation can be seen as one of the so-called *observable properties* of a program P. The set of atoms A for which the $\forall\exists\forall$-closure of $\neg A$ can be inferred from P represents yet another *failure set* for the program P, along with the standard finite failure set *FF*, the failure by instantiation set *FFI* of [3], etc. (see [4] for a classification of the semantical characterizations of the various operational properties of a logic program). The correctness and the completeness results shown in this paper give a model-theoretic characterization of the new observable. We plan to provide it with an equivalent fixpoint characterization. The fixpoint operator will be (a suitable modification of) the immediate consequence operator, T_c, of the C-semantics ([6, 7]).

In this paper we dealt with definite programs only. In the context of this work it is natural to consider general programs and goals in which subformulae of the form $\exists \overline{x} \forall \overline{z} \neg Q$ may occur, in addition to positive and negative literals. We believe that the results presented in this paper could be easily extended to such general programs and goals. The semantics of interest would be the three-valued completion semantics of Kunen ([8]).

6 Acknowledgments

The research of Alessandra Di Pierro was carried out during a post-doc stay at the Department of Computer and Information Science, Linköping University, in the context of the HCM project "Logic Program Synthesis and Transformation" (CHRX-CT93-0414).

The research of Włodzimierz Drabent was partly supported by Polish Academy of Sciences and by Swedish Research Council for Engineering Sciences (dnr 221-93-942).

References

[1] K. R. Apt. Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[3] A. Di Pierro, M. Martelli, and C. Palamidessi. Negation as Instantiation. *Information and Computation*, 120(2):263–278, August 1995.

[4] A. Di Pierro. Negation and Infinite Computations in Logic Programming. PhD thesis, Universitá di Pisa, 1994. Technical Report 3/94.

[5] W. Drabent. What is failure? an approach to constructive negation. *Acta Informatica*, 32:27–59, 1995.

[6] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[7] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 103(1):86–113, 1993.

[8] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

[9] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.

[10] J. W. Lloyd and R. W. Topor. Making Prolog more Expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.