# Negation as Instantiation

Alessandra Di Pierro[1], Maurizio Martelli[2], Catuscia Palamidessi[2]

[1]Dipartimento di Informatica,
Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy

[2]Dipartimento di Informatica e Scienze dell'Informazione,
Università di Genova,
via Benedetto XV 3, 16132 Genova, Italy

## Abstract

We propose a new negation rule for logic programming which derives existentially closed negative literals, and we define a version of completion for which this rule is sound and complete. The rule is called "Negation As Instantiation" (NAI). According to it, a negated atom succeeds whenever all the branches of the SLD-tree for the atom either fail or instantiate the atom. The set of the atoms whose negation is inferred by the NAI rule is proved equivalent to the complement of $T_C \downarrow \omega$, where $T_C$ is the immediate consequence operator extended to non-ground atoms (Falaschi et al., 1989). The NAI rule subsumes negation as failure on ground atoms, it excludes floundering and can be efficiently implemented. We formalize this way of handling negation in terms of SLDNI-resolution (SLD-resolution with Negation as Instantiation). Finally, we amalgamate SLDNI-resolution and SLDNF-resolution, thus obtaining a new resolution procedure which is able to treat negative literals with both existentially quantified variables and free variables, and we prove its correctness.

# 1 Introduction

SLD-resolution and the models á la Tarski of a definite logic program $P$, are used to characterize, respectively, the operational and the declarative meaning of $P$ with respect to the positive literals. In order to infer also negative literals, Clark (1978) introduced the Negation As Failure rule (NAF), which still nowadays represents the most widely used treatment of negation in logic programming. The declarative semantics of NAF is given in terms of the completion $Comp(P)$ of a definite program $P$. We recall the classical results:

$$\leftarrow A \overset{\theta}{\longmapsto}{}^{*} \Box \quad \Rightarrow \quad P \models \forall A\theta \qquad \text{(soundness of success)}$$

$$P \models A\theta \quad \Rightarrow \quad \exists\sigma \leq \theta \text{ s.t. } \leftarrow A \overset{\sigma}{\longmapsto}{}^{*} \Box \qquad \text{(completeness of success)}$$

$$A \text{ finitely fails} \quad \Leftrightarrow \quad Comp(P) \models \forall\neg A \qquad \text{(soundness and completeness of failure).}$$

These results are, however, somehow restrictive. There are essentially two limits:

1. One of the fundamental aspects of a logic program, i.e. the computed answer substitutions, are not characterized.
2. Only a small part of the negative information which could be drawn from a program is inferred, namely the universally closed negative literals.

The former fits into the more general problem of capturing, by means of suitable models, the so-called *observable properties* of a program (Falaschi et al., 1993) and has been tackled and solved by Falaschi et al. (1989). In the mentioned paper a new declarative semantics based on interpretations containing non-ground atoms was defined, and two different interpretation notions (the S-semantics and the C-semantics) were introduced. In particular it was shown that one of the two models (the S-semantics), allows for precisely characterizing the set of computed answer substitutions.

This paper improves the situation with respect to the second limitation, by presenting a new rule which allows us to derive existentially closed negative literals. This rule works as follows: $\exists\neg A$ is inferred when all derivations for $\leftarrow A$ either fail or instantiate some of the variables of $A$. The declarative justification of this inference is given in terms of an appropriate reference theory and the characterization of the set of existentially closed negative atoms is given in terms of the immediate consequence operator of the C-semantics.

The basic idea is the following. If all derivations for $\leftarrow A$ in a program $P$ either fail or instantiate some of the variables of $A$, then $\forall A$ is not a logical consequence of the Clark's completion of $P$, $Comp(P)$ (Clark, 1978). Therefore it is consistent to infer $\neg\forall A$, namely $\exists\neg A$. We want to extend now the theory $Comp(P)$ in order to *validly* infer $\exists\neg A$. To this purpose, note that if every branch of the SLD-tree for $P \cup \{\leftarrow A\}$ either fails or instantiates some of the variables of $A$, then for a grounding substitution $\eta$ instantiating all variables to distinct fresh constants, the 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, to obtain the appropriate reference theory, it is sufficient to extend the underlying language by infinitely many constant symbols, and consider the completion of $P$ over the extended language $L$, $Comp_L(P)$. Based on this concept of "finite instantiation" we define a new negation rule which we call *Negation As Instantiation* (NAI) and an operational semantics for negation, the Failure by Finite Instantiation set (the $FFI$ set), consisting of all the atoms whose existentially quantified negation can be inferred.

The fixpoint characterization of the $FFI$ set is obtained by using the C-semantics, and this leads to the following analogies with the standard semantics. Let $SS$ be the (ground) success set, $FF$ the (ground) finitely failure set and $\mathcal{B}$ the standard Herbrand base. Furthermore, let $T$ be the standard immediate consequence operator and $T_C$ the immediate consequence operator of the C-semantics, as defined in Falaschi et al. (1989). Then:

$$SS \;=\; \{A \mid A \text{ is ground and } \leftarrow A \overset{\varepsilon}{\longmapsto}{}^{*} \Box\} \tag{1}$$

$$\begin{aligned}
&= \{A \mid A \text{ is ground and } P \models A\} \\
&= M \text{ (the least Herbrand model)} \\
&= T\!\uparrow\omega,
\end{aligned}$$

and

$$\begin{aligned}
FF &= \{A \mid \ A \text{ is ground and there exists a finitely} \\
&\qquad\quad \text{failed SLD-tree for } P \cup \{\leftarrow A\}\ \} \\
&= \{A \mid Comp(P) \models \neg A\} \\
&= \mathcal{B} \setminus T\!\downarrow\omega.
\end{aligned} \tag{2}$$

The set of non-ground atoms which can be refuted with an empty computed answer substitution is shown in Falaschi et al. (1989,1993) to be equivalent to the set of (universally quantified) atomic consequences (called here *NESS* for Non-ground Empty computed answer substitutions Success Set), and to the least fixpoint of $T_C$. Thus we have

$$\begin{aligned}
NESS &= \{A \mid \leftarrow A \overset{\varepsilon}{\longmapsto}{}^{*} \Box\} \\
&= \{A \mid P \models \forall A\} \\
&= T_C\!\uparrow\omega.
\end{aligned}$$

Now, denoting by $\mathcal{B}_{\mathrm{v}}$ the Herbrand base extended with non-ground atoms, we will show that

$$\begin{aligned}
FFI &= \{A \mid \text{there exists a finitely instantiating SLD-tree for } P \cup \{\leftarrow A\}\ \} \\
&= \{A \mid Comp_L(P) \models \neg\forall A\} \\
&= \mathcal{B}_{\mathrm{v}} \setminus T_C\!\downarrow\omega.
\end{aligned}$$

Therefore *FFI* is the negative counterpart of *NESS* in the same way *FF* is the negative counterpart of *SS*.

We then consider the application of the NAI rule to programs containing existentially closed negative queries in the bodies of the clauses. We extend SLD-resolution by adding the NAI rule to solve these subgoals, thus obtaining what we call SLDNI-resolution, and we show that it is correct w.r.t. $Comp_L(P)$. Concerning completeness, SLDNI-resolution does not present the problem of floundering, but still the existence of non-terminating computations is an obstacle, as in the case of SLDNF-resolution. The two approaches to negation, NAF and NAI, are in a sense orthogonal, hence they can be combined smoothly. We propose an amalgamation of these notions which should combine the advantages of both of them. The resulting system, SLDNFI-resolution, is still correct. It is however not complete, as one might expect, since it results from the combination of two incomplete methods.

One drawback of the approaches mentioned so far (SLDNF, SLDNI and SLDNFI) is that they are not able to compute bindings for the variables in negative subgoals.

To solve the problem of getting bindings for negative literals, other approaches have been proposed, such as constructive negation (Chan, 1988). Our extension has the advantage that the implementation of the NAI rule is very simple and can be obtained by a small modification of the NAF rule.

## 1.1 Plan of the paper

The next section introduces the terminology and the basic results concerning the semantics of logic programs. In Section 3 the notion of *Failure by Instantiation* is introduced and characterized both in terms of $T_C$, and model-theoretically, in terms of an appropriate completion of the program. In section 4 we extend the NAI rule to a kind of general programs, called $\exists\neg$*general programs*, thus defining SLDNI-resolution, which is shown correct. Finally, in Section 5 we propose an amalgamation of SLDNF and SLDNI-resolution, and we show its correctness.

## 2 Preliminaries

In this section we recall the terminology and the basic results in the semantics of logic programs.

A language consists of three disjoint sets: a set *Con* of *function symbols* (*data constructors*), a set *Pred* of *predicate symbols* and a set *Var* of *variable symbols*. Each symbol in *Con* and *Pred* is associated with a number representing its arity.

Let *Term* be the set of terms $t, u \ldots$ built on *Con* and *Var*. The Herbrand universe $U$ is the set of all ground terms. A substitution is a mapping $\theta : Var \rightarrow Term$ such that the set $Dom(\theta) = \{x \mid \theta(x) \neq x\}$ (*domain* of $\theta$) is finite. The *codomain* of $\theta$ is the set $Cod(\theta) = \{t \mid t \in Term$ and $\exists x \in Dom(\theta)$ such that $t = \theta(x)\}$. For a set of variables $V$, the *restriction of* $\theta$ *to* $V$, denoted by $\theta_{|_V}$, is the substitution defined by $\theta_{|_V}(x) = \theta(x)$ if $x \in V$, and $\theta_{|_V}(x) = x$ otherwise. The empty substitution is denoted by $\varepsilon$. Given a term $t$ and a substitution $\theta$, $t\theta$ denotes the term obtained by replacing every variable $x$ in $t$ by $\theta(x)$. The composition $\theta\sigma$ of the substitutions $\theta$ and $\sigma$ is defined as $\theta\sigma(x) = x\theta\sigma$. The pre-ordering $\leq$ on substitutions is defined as $\theta \leq \sigma$ iff there exists $\theta'$ such that $\theta\theta' = \sigma$. The associate equivalence relation is called *renaming*.

The set *Atom* is a set of objects $A, B \ldots$ of the form $p(t_1, \ldots, t_n)$, where $p$ is a predicate with arity $n$, and $\forall i \in [1, n],\ t_i \in Term$. A *literal* is either an atom or the negation of an atom. The Herbrand Base $\mathcal{B}$ is the set of all ground atoms. The application of the substitution $\theta$ to the atom $A$ is denoted by $A\theta$. The relation $A \leq A'$ ($A$ is more general than $A'$) holds iff there exists $\theta$ such that $A\theta = A'$. The relation $\leq$ is a preorder, and the associated equivalence relation (still called *renaming*) will be denoted by $\approx$.

We use $\forall A$, $\exists A$ to denote respectively $\forall x_1, \ldots, \forall x_n.A$ and $\exists x_1, \ldots, \exists x_n.A$, where $x_1, \ldots, x_n$ are all the variables occurring in $A$.

Two atoms $A$ and $A'$ are *unifiable* iff there exists a substitution $\theta$ such that $A\theta = A'\theta$. Such substitution is called *unifier* of $A$ and $A'$. If $A$ and $A'$ are unifiable, then there exists a smallest unifier w.r.t. $\leq$, apart from renaming, which is called the *most general unifier* (*mgu*) of $A$ and $A'$, and denoted by $mgu(A, A')$. The notion of mgu generalizes to sequences of atoms or terms in the obvious way.

A *definite clause* is a formula $H \leftarrow B_1, \ldots, B_n$ ($n \geq 0$), where $H$ and the $B_i$'s are atoms, "$\leftarrow$" and "," denote logic implication and conjunction respectively, and all variables are universally quantified. $H$ is the *head* of the clause and $B_1, \ldots, B_n$ is the *body*. If the body is empty then the clause is a *unit clause*. A *definite Horn program*, (or simply *program*), is a finite set of definite clauses $P = \{C_1, \ldots, C_n\}$. A *goal* $G$ is a formula $\leftarrow A_1, \ldots, A_m$, where each $A_i$ is an atom. We will often refer to them as *positive* programs and goals.

Given a goal $G$ of the form $\leftarrow A_1, \ldots, A_m$, and a program $P$, a *derivation step* $G \overset{\theta}{\longmapsto} G'$ is possible if there exists a clause $H \leftarrow B_1, \ldots B_n$ which is obtained by renaming in an appropriate way (see later) one of the clauses in $P$, such that $\theta = mgu(A_i, H)$. The resulting goal $G'$ is $\leftarrow (A_1, \ldots, A_{i_1}, B_1, \ldots B_n, A_{i+1}, \ldots, A_m)\theta$. The atom $A_i$ is called *selected atom*. A SLD-derivation for $P \cup \{G\}$ is a sequence of derivation steps $G = G_0 \overset{\theta_0}{\longmapsto} G_1 \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_{n-1}}{\longmapsto} G_n$, with $n \geq 0$. We denote such a derivation by $G \overset{\theta}{\longmapsto}{}^{*} G_n$, with $\theta = \epsilon$ if $n = 0$, and $\theta = (\theta_0\theta_1 \ldots \theta_{n-1})_{|_{\bar{x}}}$ if $n \geq 0$, where $\bar{x}$ are the variables of $G$. In the following, the notation $\bar{x}$ will be used to indicate both a set $\{x_1, \ldots, x_n\}$ and a sequence $x_1, \ldots, x_n$. The rule which associates to each (occurrence of) a goal in a derivation the selected atom is called *selection rule*. When used in a derivation step, a clause has to be renamed so to contain no variables in common with any goal occurring in the prefix of the derivation until that step (*standardization apart*). Without loss of generality, we will assume that each mgu $\mu$ generated in a derivation is *idempotent* (i.e. $\mu\mu = \mu$) and *relevant* (i.e. $\mu$ affects only variables which occur in the atoms to be unified). If $G \overset{\theta}{\longmapsto}{}^{*} \Box$, where $\Box$ is the *empty goal*, the derivation is a *SLD-refutation*, and $\theta$ is the corresponding *computed answer substitution* (c.a.s.). An SLD-derivation is *failed* if it is finite, maximal (i.e. it is not the prefix of a longer derivation) and it is not a refutation. A derivation is *fair* if it is either finite or every atom which occurs in it is eventually selected.

An SLD-tree for $P \cup \{G\}$ is a way of representing all derivations for $G$ in $P$ via a fixed selection rule $R$. It is a maximal tree such that

**(i)** the nodes are goals, and the root is $G$,

**(ii)** the children of a node are all the goals obtained by performing a resolution step with a renamed clause of $P$ whose head is unifiable with the atom selected according to $R$.

Note that a branch of an SLD-tree corresponds to an SLD derivation. An SLD-tree is *successful* if it contains the empty clause, and it is *finitely failed* if it is finite and not successful. It is fair if all its branches are fair.

We refer to Lloyd (1987) or Apt (1990) for the standard notions of Herbrand base $\mathcal{B}$, Herbrand interpretations $I, J, \ldots$ and the immediate consequence operator $T$.

Following Falaschi et al. (1989,1993), we summarize now the notion of Herbrand structure enriched with variables, and the corresponding immediate consequence operator.

The *Herbrand base with variables* is the set of equivalence classes (quotient set) of the set of all atoms with respect to the equivalence relation $\approx$:

$$\mathcal{B}_{\mathrm{v}} = Atom_{/\approx}.$$

For the sake of simplicity, we denote the equivalence class of an atom with the atom itself.

We introduce the following operators on subsets $I_{\mathrm{v}}$ of $\mathcal{B}_{\mathrm{v}}$, which will be useful in the following.

- $Down(I_{\mathrm{v}}) = \{A \mid \text{there exists } A' \in I_{\mathrm{v}} \text{ s.t. } A \leq A'\}$
- $Ground(I_{\mathrm{v}}) = \{A \mid A \in I_{\mathrm{v}} \text{ and } A \text{ is ground}\}$.

$I_{\mathrm{v}}$ is upward closed if $A \in I_{\mathrm{v}}$ and $A \leq A'$ imply $A' \in I_{\mathrm{v}}$. The immediate consequence operator $T_C$ on Herbrand interpretations is

$$T_C(I_{\mathrm{v}}) = \{A \mid \text{there exist } H \leftarrow \bar{B} \in P_{\mathrm{ren}} \text{ and a substitution } \theta \text{ s.t. } \bar{B}\theta \in I_{\mathrm{v}} \text{ and } A = H\theta\}.$$

where $P_{\mathrm{ren}}$ is the closure of $P$ under renaming and $\bar{B}$ denotes a sequence of atoms. We define the following interpretations

$$\begin{aligned} T_C\uparrow 0 &= \emptyset \\ T_C\uparrow(n+1) &= T_C(T_C\uparrow n) \\ T_C\uparrow\omega &= \textstyle\bigcup_{n<\omega} T_C\uparrow n \end{aligned}$$

and

$$\begin{aligned} T_C\downarrow 0 &= \mathcal{B}_{\mathrm{v}} \\ T_C\downarrow(n+1) &= T_C(T_C\downarrow n) \\ T_C\downarrow\omega &= \textstyle\bigcap_{n<\omega} T_C\downarrow n \end{aligned} .$$

The set of non ground atoms which have a refutation with an empty computed answer substitution is

$$NESS = \{A \mid \leftarrow A \overset{\varepsilon}{\longmapsto}{}^{*} \Box\}.$$

We recall the model-theoretic and fixpoint characterization of $NESS$, which extends the results in (1).

$$\begin{aligned} NESS &= \{A \mid P \models \forall A\} \\ &= M_{\mathrm{v}} \text{ (the least Herbrand model with variables)} \\ &= T_C\uparrow\omega. \end{aligned}$$

No negative counterpart of $NESS$ has been studied until now. In Levi et al. (1990) the set of non-ground atoms which have a finitely failed computation ($NGFF$) was introduced and shown equivalent to the set of atoms $A$ such that $Comp(P) \models \neg\exists A$, where $Comp(P)$ is the Clark's completion of $P$ (Clark, 1978). However, Levi et al. (1990) showed that $NGFF$ is the counterpart of the set $NGSS = \{A \mid \leftarrow A \overset{\theta}{\longmapsto}{}^{*} \Box\}$, which is different from $NESS$.

# 3 Failure by instantiation

In this section we present a new notion of failure and, correspondingly, a new (non-ground) failure set to be interpreted as negative information. For this set we give a characterization similar to (2), using the $T_C$ operator. Finally we show that this set contains exactly the atoms $A$ for which $Comp_L(P) \models \exists\neg A$, holds, where $Comp_L(P)$ is the completion of $P$ with respect to the reference language $L$ (containing infinite constant symbols). This set represents the actual dual set of *NESS*.

The operator *var* returns the set of variables occurring in an expression (term, atom, goal,...). For a substitution $\theta$ and a set of variables $V$, we define the property $Inst(\theta, V)$ to hold iff $\theta$ strictly instantiates the variables of $V$.

**Definition 3.1** Let $V = \{x_1, \ldots, x_n\}$, $\theta$ be a substitution and $p$ a predicate symbol of arity $n$. We say that $Inst(\theta, V)$ holds iff $p(x_1, \ldots, x_n)\theta \not\leq p(x_1, \ldots, x_n)$.

**Definition 3.2 (Finitely Instantiating SLD-tree)** Let $P$ be a program, $G$ a goal and $V \subseteq var(G)$. Let $TR$ be an SLD-tree for $P \cup \{G\}$. Then

1. $TR$ instantiates $V$ at level $k$ iff for every branch $\xi$ of $TR$ one of the following holds:
   - $\xi$ fails at level[1] $k' \leq k$, or
   - $Inst(\theta_0 \cdots \theta_{k'-1}, V)$ is true, where $\theta_0, \ldots, \theta_{k'-1}$ are the substitutions labeling the edges of $\xi$ up to level $k' \leq k$.
2. $TR$ is a finitely instantiating SLD-tree for $P \cup \{G\}$ iff $\exists k \geq 1$ s.t. $TR$ instantiates $var(G)$ at level $k$.

**Definition 3.3 (Failure by Finite Instantiation)** Let $P$ be a program. The failure by finite instantiation set of $P$ is

$$FFI = \{A \in Atom \mid \text{ there exists a finitely instantiating SLD-tree for } P \cup \{\leftarrow A\}\}.$$

In the following we will show that *FFI* corresponds to the set of atoms not belonging to $T_C \downarrow \omega$.

## 3.1 The NAI rule

Clark (1978) defined the Negation As Failure rule, based on the concept of finite failure. We present here a new rule for inferring negative information based on the concept of finite instantiation. We call this rule *Negation As Instantiation* (NAI) and we define it as:

$$\frac{A \in FFI}{\exists\neg A}.$$

The NAI rule subsumes the NAF rule in the sense that for ground atoms it coincides with the NAF rule, and furthermore it has the advantage of an efficient implementation. To show that $\exists\neg A$ holds we perform an exhaustive search for a proof of $A$. If every possible proof fails or instantiates some variables of $A$ (that is if it is possible to construct a finitely instantiating SLD-tree for $P \cup \{\leftarrow A\}$), then $\exists\neg A$ is inferred.

### 3.1.1 Examples

We show some examples to clarify the possible use of the NAI rule. We consider here positive programs and goals consisting of only one existentially closed negative literal.

[1] By level of $\xi$ we mean the number of nodes in $\xi$.

**Example 3.4** Consider the following programs:

$$P_1 = \{\ p(a) \leftarrow,\quad r(b) \leftarrow\ \}$$

$$P_2 = \{\ p(x) \leftarrow,\quad r(b) \leftarrow\ \}.$$

We have:

$$Comp(P_1) \models \neg p(b),\ \text{hence}\ Comp(P_1) \models \exists\neg p(x),$$

and

$$Comp(P_2) \models \forall p(x),\ \text{hence}\ Comp(P_2) \models \neg\exists\neg p(x).$$

There exists a finitely instantiating SLD-tree for $P_1 \cup \{\leftarrow p(x)\}$, but not for $P_2 \cup \{\leftarrow p(x)\}$. Hence the NAI rule will allow us to derive $\exists\neg p(x)$ in $P_1$, and not in $P_2$. Note that neither $P_1 \cup \{\leftarrow p(x)\}$ nor $P_2 \cup \{\leftarrow p(x)\}$ have a finitely failed SLD-tree, so no conclusion can be drawn by the NAF rule, even when extended to deal with non-ground negative literals as it is done in Apt (1990), Section 6. This is in accordance with the fact that $Comp(P_i) \not\models \forall\neg p(x)$ for both $i = 1, 2$.

**Example 3.5** The following program defines a predicate whose third argument is the sum of the others.

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

We have, for instance, that $Comp(P) \models \exists\neg plus(x, s^2(0), y)$. From the NAI rule, we can derive $\exists\neg plus(x, s^2(0), y)$; in fact, $P \cup \{\leftarrow plus(x, s^2(0), y)\}$ has a finitely instantiating SLD-tree where the goal variable $y$ results instantiated to $s^2(x)$. On the other hand, we have that for all $n \in \omega$

$$Comp(P) \models \neg\exists\neg plus(x, s^n(0), s^n(x))$$

holds, and the NAI rule fails to derive $\exists\neg plus(x, s^n(0), s^n(x))$, since $P \cup \{\leftarrow plus(x, s^n(0), s^n(x))\}$ has a non-instantiating SLD-refutation.

Let's now consider an example of application of the NAI rule in the case of a finitely instantiating, but infinite, SLD-tree.

**Example 3.6** Consider the following program

$$P = \{\ p(a) \leftarrow q(x),\quad q(x) \leftarrow q(x)\ \}$$

We have that the SLD-tree for $P \cup \{\leftarrow p(y)\}$ is infinite, but finitely instantiating. In fact $y$ is instantiated to $a$ already at the second level. Hence the NAI rule allows us to derive $\exists\neg p(y)$, while no conclusion can be derived from the NAF rule, in accordance with the fact that $Comp(P) \not\models \forall\neg p(y)$.

On the other hand, the SLD-tree for $P \cup \{\leftarrow q(y)\}$ is infinite, but not finitely instantiating. Hence no conclusion for $q$ can be derived either from the NAI or from the NAF rule.

In these examples it is not clear the relation between the NAI rule and the completion of a program; and more in general it is not clear what is the semantical justification of such a rule. We will investigate this aspect in Section 3.3, whereas now we will focus on a characterization of *FFI* in terms of the immediate consequence operator $T_C$.

## 3.2 Fixpoint characterization of failure by instantiation

Let us recall the characterizations of finite failure given in Apt (1990).

**Theorem 3.7** *Let $P$ be a program and $A$ a ground atom. Then the following are equivalent:*

**a)** $A \notin T \downarrow \omega$

**b)** $A \in FF$

**c)** *Every fair SLD-tree for $P \cup \{\leftarrow A\}$ is finitely failed.*

We show that the set $FFI$ enjoys an analogous characterization. In fact, there is a close correspondence between failure by instantiation and finite failure, as we show in this section. The basic idea is that a finitely instantiating tree becomes a finitely failed tree when the variables of the initial goal are replaced by constant symbols not occurring in the program. In fact if the unification of $A$ and $B$ requires to instantiate the variables of $A$, then no unification is possible if we replace the variables of $A$ by constant symbols not occurring in $B$. In order to exploit this relation in the proofs, technically, we have to consider Herbrand structures, $T$, $FF$ etc. defined on a language extended with the additional constant symbols.

Given an expression $E$, a sequence of variables $\bar{x}$ and a sequence of terms $\bar{t}$ of the same length, $E[\bar{t}/\bar{x}]$ denotes the expression obtained from $E$ by replacing verbatim each variable of $\bar{x}$ by the corresponding term of $\bar{t}$. Furthermore, $\theta[\bar{t}/\bar{x}]$ is the substitution defined by $\theta[\bar{t}/\bar{x}](y) = \theta(y)[\bar{t}/\bar{x}]$.

**Proposition 3.8** *Let $A \in Atom$, let $\bar{x}$ be the sequence of all variables occurring in $A$, and let $\bar{d}$ be a sequence of fresh constant symbols. Assume $A \in FFI$. Then $A[\bar{d}/\bar{x}] \in FF$.*

**Proof** Note that if $P \cup \{\leftarrow A[\bar{d}/\bar{x}]\}$ has a non-failed SLD-derivation, then we obtain a non-failed SLD-derivation for $P \cup \{\leftarrow A\}$ by replacing verbatim the occurrences of symbols of $\bar{d}$ by the corresponding symbols of $\bar{x}$ in the mgu's and in the goals. The resulting mgu's do not bind $\bar{x}$, hence the derivation does not instantiate $\bar{x}$. Therefore,

$$\begin{array}{rcl} A[\bar{d}/\bar{x}] \notin FF & \Rightarrow & \text{every SLD-tree } TR \text{ for } P \cup \{\leftarrow A[\bar{d}/\bar{x}]\} \\ & & \text{has a non-failed branch} \\ & & \\ & \Rightarrow & \text{every SLD-tree } TR' \text{ for } P \cup \{\leftarrow A(\bar{x})\} \\ & & \text{has a (finite or infinite) non-failed branch with} \\ & & \text{mgu's } \theta_0, \theta_1, \ldots \text{ such that } \forall n.\ \neg Inst(\theta_0 \cdots \theta_n, \bar{x}) \\ & & \\ & \Rightarrow & A \notin FFI. \end{array}$$

□

Let us recall the relation between the operators $T$ and $T_C$ given by Levi et al. (1990).

**Lemma 3.9** *For every $n$, $T \downarrow n = Ground(T_C \downarrow n)$ holds.*

**Lemma 3.10** $T \downarrow \omega = Ground(T_C \downarrow \omega)$.

**Proof**

$A \in T\downarrow\omega$

$\Leftrightarrow$

$A \in \bigcap_{n<\omega} T\downarrow n$

$\Leftrightarrow$

$\forall n.\ A \in T\downarrow n$

$\Leftrightarrow$ $\{$Lemma 3.9$\}$

$\forall n.\ A \in Ground(T_C\downarrow n)$

$\Leftrightarrow$ $\{$upward closeness of $T_C\downarrow n$, see Definition 6.2 in Falaschi et al. (1989)$\}$

$A$ is ground and $\forall n.\ A \in T_C\downarrow n$

$\Leftrightarrow$

$A$ is ground and $A \in T_C\downarrow\omega$

$\Leftrightarrow$

$A \in Ground(T_C\downarrow\omega)$.

$\square$

**Proposition 3.11** $A \in FFI \Rightarrow A \notin T_C\downarrow\omega$.

**Proof** Let $\bar{d}$ be a sequence of new constant symbols. We have:

$A \in FFI$

$\Rightarrow$ $\{$Proposition 3.8$\}$

$A[\bar{d}/\bar{x}] \in FF$

$\Rightarrow$ $\{$Theorem 3.7$\}$

$A[\bar{d}/\bar{x}] \notin T\downarrow\omega$

$\Rightarrow$ $\{$Lemma 3.10$\}$

$A[\bar{d}/\bar{x}] \notin Ground(T_C\downarrow\omega)$

$\Rightarrow$ $\{$upward closedness of $T_C\downarrow\omega\}$

$A \notin T_C\downarrow\omega$.

$\square$

Observe that the definition of $FFI$ is existential, in the sense that $A \in FFI$ iff there exists at least a selection rule which gives a finitely instantiating SLD-tree. Since we don't know a-priori which selection rule will have this property, it might be very expensive to check whether $A \in FFI$. Fortunately, as in the case of $FF$, it is possible to give a universal characterization of $FFI$ in terms of fair SLD-trees. We will see, in fact, that if $A \in FFI$ then every fair SLD-tree for $P \cup \{\leftarrow A\}$ is finitely instantiating. To prove this result we need the following lemma which generalizes the Lemma 5.10 of Apt (1990).

**Lemma 3.12** *Suppose there exists an infinite fair SLD-derivation, $G_0, G_1, \ldots$, for $P \cup \{G\}$, with $G = G_0 = \leftarrow A_1, \ldots, A_m$ and substitutions $\theta_0, \theta_1, \ldots$. Then for every $k \geq 0$ there exists $n \geq 0$ s.t. $\forall i \in [1, m].\ A_i\theta_0 \cdots \theta_{n-1} \in T_C\downarrow k$.*

**Proof** Lemma 5.10 of Apt (1990) states a similar results, the difference is that it concludes $[A_i\theta_0 \cdots \theta_n] \subseteq T \downarrow k$, where $[A]$ stands for the set of the ground instances of $A$. The proof is analogous, and proceeds by induction on $k$. The claim is clearly true for $k = 0$. Suppose it is true for $k-1$. Let $i \in [1, m]$. Since the derivation is fair, there exists $p \geq 0$ such that the atom $A_i\theta_0 \cdots \theta_{p-1}$ is selected in the goal $G_p$. Let $G_{p+1}$ be the goal $\leftarrow B_1, \ldots, B_q$. By inductive hypothesis, for some $s \geq 0$

$$\{B_j\theta_{p+1} \cdots \theta_{p+s} \mid j \in [1, q]\} \subseteq T_C \downarrow (k-1)$$

holds. From the definition of $T_C$ we have

$$A_i\theta_0 \cdots \theta_{p+s} \in T_C(\{B_j\theta_{p+1} \cdots \theta_{p+s} \mid j \in [1, q]\}).$$

Therefore, by monotonicity of $T_C$, we conclude

$$A_i\theta_0 \cdots \theta_{p+s} \in T_C \downarrow k.$$

□

**Proposition 3.13** *Let $P$ be a program and let $A \in \mathcal{B}_{\mathrm{V}}$. If $A \notin T_C \downarrow \omega$, then every fair SLD-tree for $P \cup \{\leftarrow A\}$ is finitely instantiating.*

**Proof** Suppose, by contradiction, that there is a fair SLD-tree $TR$ for $P \cup \{\leftarrow A\}$ which is not instantiating at level $k$ for any $k$. Then there exists a branch of $TR$ which for any $k$ does not fail at level $k$, and its mgu's do not instantiate $var(A)$ up to level $k$. This means that such a derivation is

- a refutation with computed answer substitution $\varepsilon$, or
- an infinite fair SLD-derivation which does not instantiate $var(A)$.

In both cases $A$ would belong to $T_C \downarrow \omega$, against the hypothesis. In fact, in the former case, by strong soundness w.r.t. the C-semantics (Corollary 7.1 of Falaschi et al. (1989)),

$$A \in T_C \uparrow \omega \subseteq T_C \downarrow \omega$$

holds, and in the latter case, by Lemma 3.12 we have

$$\forall k.\ A \in T_C \downarrow k \ \Rightarrow\ A \in \bigcap_{k \in \omega} T_C \downarrow k = T_C \downarrow \omega.$$

□

**Remark 3.14** *If every fair SLD-tree with $\leftarrow A$ as a root is finitely instantiating, then there exists a finitely instantiating SLD-tree for $P \cup \{\leftarrow A\}$, i.e. $A \in FFI$.*

The next theorem collects the results of Proposition 3.11, Proposition 3.13 and Remark 3.14 and gives the fixpoint characterization of the set *FFI*.

**Theorem 3.15** *Consider a program $P$ and let $A \in \mathcal{B}_{\mathrm{V}}$. Then the following are equivalent:*

**a)** $A \notin T_C \downarrow \omega$

**b)** $A \in FFI$

**c)** *Every fair SLD-tree for $P \cup \{\leftarrow A\}$ is finitely instantiating.*

**Proof**

$(c) \Rightarrow (b)$ {Remark 3.14}

$(b) \Rightarrow (a)$ {Proposition 3.11}

$(a) \Rightarrow (c)$ {Proposition 3.13}.

□

Note that, analogously to (2), we have

$FFI = \mathcal{B}_{\rm v} \setminus T_C \downarrow \omega.$

Although in the proof we have used Herbrand structures on a language enriched with additional constant symbols, Theorem 3.15 holds also for a $T_C$ defined only on the vocabulary of the program. In fact it is easy to see that the relations between $T$ and $T_C$ expressed by Lemmata 3.9 and 3.10 still hold when the domain of $T_C$ is restricted to the vocabulary of the program, provided that we replace $Ground(T_C \downarrow n)$ by $Ground(Up(T_C \downarrow n))$, and $Ground(T_C \downarrow \omega)$ by $Ground(Up(T_C \downarrow \omega))$, with $Up$ defined as follows:

$Up(I_{\rm v}) = \{A \mid \exists B \in I_{\rm v} \text{ such that } B \leq A\}$

where $A$ is an atom in the extended language. The rest of the proofs carries out without modification.

By an obvious generalization of Proposition 3.11 and Proposition 3.13, it is possible to extend the previous theorem to the case of a goal consisting of several atoms.

**Theorem 3.16** *Consider a program $P$ and a goal $G = \leftarrow A_1, \ldots, A_n$. Then the following are equivalent:*

**a)** $\exists i \in [1, n]$ *s.t.* $A_i \notin T_C \downarrow \omega$,

**b)** $P \cup \{G\}$ *has a fair finitely instantiating SLD-tree,*

**c)** *Every fair SLD-tree for $P \cup \{G\}$ is finitely instantiating.*

The *FFI* set is greater than the standard *NGFF* set. In fact, Levi et al. (1990) proved that the *NGFF* set is the complement (w.r.t. $\mathcal{B}_{\rm v}$) of the set

$$\tilde{I}_{\rm v} = \bigcap_{n \in \omega} Down(T_C \downarrow n)$$

and, by definition, $T_C \downarrow n \subseteq Down(T_C \downarrow n)$. Therefore $T_C \downarrow \omega \subseteq \tilde{I}_{\rm v}$ holds. Hence we have:

$NGFF = \mathcal{B}_{\rm v} \setminus \tilde{I}_{\rm v} \subseteq \mathcal{B}_{\rm v} \setminus T_C \downarrow \omega = FFI.$

## 3.3 Model-theoretic characterization of *FFI*

We give here a validation of the NAI rule in terms of the model-theoretic semantics. First of all, we show that we need to consider a language extended by infinitely many symbols. The reason is that $Comp(P)$ is not always adequate, in fact in some cases there are models of $Comp(P)$ which do not contain enough elements.

**Example 3.17** Consider the program

$$P = \{\ p(0) \leftarrow, \quad p(s(x)) \leftarrow p(x)\ \}.$$

The atom $p(x)$ belongs to *FFI* (hence the NAI rule would infer $\exists\neg p(x)$), but in some models of $Comp(P)$ the formula $\exists\neg p(x)$ is false. More precisely, it is false in those models which contain only the elements corresponding to the terms $0, s(0), s(s(0)), \ldots$.

An obvious solution is to "force" in the models additional elements. In the above example we would need just one, but in general we might need more of them.

**Example 3.18** Consider the program

$$
\begin{aligned}
P = \{\ & p(a, x) \leftarrow, \\
& p(x, a) \leftarrow, \\
& p(x, x) \leftarrow \quad \}.
\end{aligned}
$$

The atom $p(x, y)$ belongs to $FFI$, but the formula $\exists\neg p(x, y)$ is true only in those models of $Comp(P)$ which contain at least three distinct elements.

As the previous example shows we must consider a reference language which contains, besides all symbols which might occur in the programs, as many new constant symbols as the number of variables which might occur in the formulae we want to deal with. Since we are interested to extend the results to conjuncts of arbitrary length, it is convenient to consider a language containing infinitely many additional constants symbols $d_1, d_2, \ldots, d_n, \ldots$. We call $L$ this language. An alternative approach would be to consider a language containing at least one constant symbol and one additional unary function symbol.

**Definition 3.19** The extended completion of $P$, $Comp_L(P)$, is defined as $IFF(P) \cup CET_L$, where $IFF(P)$ is the collection of completed definitions of predicates in $P$ (see Apt, 1990) and $CET_L$ is the set of the equality and freeness axioms ($EA \cup FA$) for $L$, as defined, for instance, in Shepherdson (1988).

$Comp_L(P)$ is an extension of $Comp(P)$, as shown by the following proposition.

**Proposition 3.20** *Let $F$ be a (first order) formula. Then*

*if $Comp(P) \models F$ then $Comp_L(P) \models F$.*

**Proof** $L$ contains all symbols which occur in $P$, hence the axioms of $Comp(P)$ are a subset of the axioms of $Comp_L(P)$. Therefore, every model of $Comp_L(P)$ is also a model of $Comp(P)$. □

We show now that the assertion $Comp_L(P) \models \exists\neg A$ can be appended to the chain of equivalences in Theorem 3.15. We start with the soundness of the NAI rule.

**Proposition 3.21** *If $A \in FFI$ then $Comp_L(P) \models \exists\neg A$.*

**Proof** Let $A \in FFI$ and let $\bar{x}$ be the sequence of all variables in $A$. Let $\bar{d}$ be a sequence of symbols not occurring in $P$. Then

$$
\begin{aligned}
& A \in FFI \\
\Rightarrow \quad & \{\text{Proposition 3.8}\} \\
& A[\bar{d}/\bar{x}] \in FF \\
\Rightarrow \quad & \{\text{Theorem 5.32 in Apt (1990)}\} \\
& Comp(P) \models \neg A[\bar{d}/\bar{x}] \\
\Rightarrow \quad & \\
& Comp(P) \models \exists\neg A \\
\Rightarrow \quad & \{\text{Proposition 3.20}\} \\
& Comp_L(P) \models \exists\neg A.
\end{aligned}
$$

□

To prove the completeness of the NAI rule we use some results from the literature on the three-valued logic and the Kunen/Fitting immediate consequence operator $\Phi_P$. In the following, we indicate by $\models_3$ the truth in a three-valued model. For the basic definitions of these notions see, fot instance, Fitting (1985) or Shepherdson (1988). These results are formulated within a framework which assumes an underlying language containing countably infinite sets of $n$-adic function symbols and of $n$-adic predicate symbols, for each $n \geq 0$. Anyway, for our purposes, it is not necessary to consider such an extension, since the only properties we need are valid for every kind of language.

We use also the following relation between *FFI* and *FF*, which is the reverse of Proposition 3.8. Actually it is more general than the reverse, because the variables here can be replaced with ground terms whatsoever.

**Lemma 3.22** *Let $A \in Atom$, let $\bar{x}$ be the sequence of all variables occurring in $A$, and let $\bar{t}$ be a sequence of ground terms. Assume $A[\bar{t}/\bar{x}] \in FF$. Then $A \in FFI$ holds.*

**Proof** Assume that $P \cup \{\leftarrow A\}$ has a (finite or infinite) non-failed SLD-derivation with mgu's $\theta_0, \theta_1, \ldots$ such that $\forall n.\ \neg Inst(\theta_0 \cdots \theta_n, \bar{x})$. Then we can replace in the goals and in the mgu's of the derivation all occurrences of variables of $\bar{x}$ by the corresponding terms of $\bar{t}$, thus obtaining a non-failed SLD-derivation for $P \cup \{\leftarrow A[\bar{t}/\bar{x}]\}$. Therefore,

$A \notin FFI \Rightarrow$ every SLD-tree $TR$ for $P \cup \{\leftarrow A\}$ has a (finite or infinite) non-failed branch with mgu's $\theta_0, \theta_1, \ldots$ such that $\forall n.\ \neg Inst(\theta_0 \cdots \theta_n, \bar{x})$

$\Rightarrow$ every SLD-tree $TR'$ for $P \cup \{\leftarrow A[\bar{t}/\bar{x}]\}$ has a non-failed branch

$\Rightarrow A[\bar{t}/\bar{x}] \notin FF$.

□

We can now prove the completeness of *FFI*.

**Proposition 3.23** *If $Comp_L(P) \models \exists \neg A$ then $A \in FFI$.*

**Proof** Let $\bar{x}$ be the sequence of variables occurring in $A$.

$$
\begin{array}{ll}
 & Comp_L(P) \models \exists\neg A \\
\Rightarrow & \{\text{Theorem 38 in Shepherdson (1988)}\} \\
 & Comp_L(P) \models_3 \exists\neg A \\
\Rightarrow & \{\text{Theorem 4.1 in Kunen (1989)}\} \\
 & \exists\neg A \text{ is true in } \Phi_P \uparrow n, \text{ for some } n \\
\Rightarrow & \\
 & \neg A[\bar{t}/\bar{x}] \text{ is true in } \Phi_P \uparrow n \text{ for some ground terms } \bar{t} \\
\Rightarrow & \\
 & A[\bar{t}/\bar{x}] \text{ is false in } \Phi_P \uparrow n \\
\Rightarrow & \{\text{Lemma 6 in Shepherdson (1988)}\} \\
 & A[\bar{t}/\bar{x}] \notin T_P \downarrow n \\
\Rightarrow & \\
 & A[\bar{t}/\bar{x}] \notin T_P \downarrow \omega \\
\Rightarrow & \{\text{Theorem 3.7}\} \\
 & A[\bar{t}/\bar{x}] \in FF \\
\Rightarrow & \{\text{Lemma 3.22}\} \\
 & A \in FFI.
\end{array}
$$

□

From Propositions 3.21 and 3.23 and from Theorem 3.15, we have the following complete characterization of the NAI rule:

**Theorem 3.24** *Consider a program $P$ and let $A \in \mathcal{B}_V$. Then the following are equivalent:*

**a)** $A \notin T_C \downarrow \omega$

**b)** $A \in FFI$

**c)** *Every fair SLD-tree for $P \cup \{\leftarrow A\}$ is finitely instantiating*

**d)** $Comp_L(P) \models \exists\neg A$.

Theorem 3.24 generalizes naturally to the case of a goal consisting of several atoms, i.e., $A$ can be substituted by a conjunction $A_1, \ldots, A_n$.

From Theorem 3.24 and Proposition 3.20, it follows that the NAI rule is complete also with respect to $Comp(P)$.

**Corollary 3.25**

*If $Comp(P) \models \exists\neg A$ then $A \in FFI$.*

Under a stronger hypothesis we can prove soundness too. Intuitively, if $P \cup \{\leftarrow A\}$ has a finitely instantiating SLD-tree and the instances on $A$ given by this tree do not cover all possible terms of the Herbrand universe of the program, then for $\theta$ instantiating $var(A)$ to some missing terms, $P \cup \{\leftarrow A\theta\}$ should fail. Hence $\exists\neg A$ should be captured by the standard completion. The next definition formalizes this notion of "non-covering tree".

**Definition 3.26** Let $A \in \mathcal{B}_V$, $P$ be a program, $TR$ an SLD-tree for $P \cup \{\leftarrow A\}$ and $\bar{x}$ the sequence of variables occurring in $A$. We say that $TR$ is a *non-covering tree* for $A$ iff there exists a grounding substitution $\sigma$ (on the standard Herbrand universe associated to $P$) for $A$, such that for every maximal branch $\xi$ in $TR$, either

- $\xi$ finitely fails, or
- there exists $k$ such that $A\sigma \notin [A\theta_0 \cdots \theta_k]$, where $\theta_0, \ldots, \theta_k$ are the substitutions labeling the edges of $\xi$ up to level $k$.

**Proposition 3.27** *If there exists a non-covering finitely instantiating SLD-tree for $P \cup \{\leftarrow A\}$, then $Comp(P) \models \exists\neg A$.*

**Proof** Consider a non-covering finitely instantiating SLD-tree $TR$ for $P \cup \{\leftarrow A\}$. Let $\sigma = \{\bar{x}/\bar{t}\}$ be the ground substitution in Definition 3.26. By substituting verbatim $\bar{t}$ for $\bar{x}$ in $TR$ we obtain a finitely failed SLD-tree for $P \cup \{\leftarrow A\sigma\}$. By the soundness of NAF we have $Comp(P) \models \neg A\sigma$, hence we conclude $Comp(P) \models \exists\neg A$. □

We give some examples to make clear the sense of the above proposition.

**Example 3.28** Consider the program in Example 3.17 and consider the goal $\leftarrow p(x)$. We have that the (only) SLD-tree for $P \cup \{\leftarrow p(x)\}$ is finitely instantiating, and covering (i.e. not non-covering). In fact, the substitution $\theta = \{x/a\}$, which is the only possible grounding substitution for $p(x)$ in the language of $P$, labels a non-failing branch of the tree. Therefore, $Comp(P) \not\models \exists\neg p(x)$, whereas $Comp_L(P) \models \exists\neg p(x)$.

**Example 3.29** Consider the program $P$ of Example 3.5 and consider the goal $\leftarrow plus(x, s^2(0), y)$.

We have that the (only) SLD-tree for $P \cup \{\leftarrow plus(x, s^2(0), y)\}$ is finitely instantiating and non-covering; in fact the edges of the (only) branch of the tree are labeled by $\theta_0 = \{y/s(y_1)\}, \theta_1 = \{y_1/s(y_2)\}, \theta_2 = \{y_2/x\}$. Thus we have, for instance,

$$plus(0, s^2(0), 0) \notin Ground(plus(x, s^2(0), y)\theta_0\theta_1\theta_2).$$

Therefore $Comp(P) \models \neg plus(0, s^2(0), 0)$, hence $Comp(P) \models \exists\neg plus(x, s^2(0), y)$.

# 4 SLDNI-resolution

In this section we define an extension of SLD-resolution based on the treatment of negative information via the NAI rule. We consider programs possibly containing clauses with existentially quantified negative queries in the body, which we call $\exists\neg$*general programs*.

The syntax for $\exists\neg$*general* queries, goals and programs is described by the following grammar:

| | |
|---|---|
| Queries | $Q ::= A \mid \exists\neg(Q) \mid Q, Q$ |
| Goals | $G ::= \leftarrow Q$ |
| Clauses | $C ::= A \leftarrow Q \mid A \leftarrow .$ |

where $A \in Atom$. In the rest of the paper, when there is no risk of ambiguity, we write $\exists\neg Q$ for $\exists\neg(Q)$.

The notions of literal and selection rule are extended as follows.

**Definition 4.1** Let $Q$ be a query. The *conjuncts* of $Q$, *Conj*(Q), is the smallest set such that

- if $Q \equiv A$ then *Conj*(Q)={A}
- if $Q \equiv \exists\neg Q'$ then $Conj(Q) = \{\exists\neg Q'\}$
- if $Q \equiv Q_1, Q_2$ then $Conj(Q) = Conj(Q_1) \cup Conj(Q_2)$.

**Definition 4.2** A *selection rule* specifies, for every goal $\leftarrow Q$, one conjunct of $Q$ (the selected conjunct).

Roughly, the notion of derivation extends as follows. Let $P$ be a $\exists\neg general$ program, $R$ a selection rule and $\leftarrow Q$ the current goal. Let $Q'$ be the selected conjunct. If $Q'$ is an atom, then we choose a clause $A \leftarrow Q''$ of $P_{\text{ren}}$ such that $A$ unifies with $Q'$, and replace $Q'$ by $Q''$, in $Q$. Then, we apply the mgu $\theta$ to all variables which are not in the scope of an existential quantifier. We will use the notation $\leftarrow ((Q \setminus Q'), Q'')\theta$ to indicate the resulting goal. If such a clause does not exist, then the derivation fails.

If $Q'$ is of the form $\exists\neg Q''$, then we check whether the derivation tree for $P \cup \{\leftarrow Q''\}$ instantiates the free variables of $Q''$. If this is the case, then $Q'$ is removed from $\leftarrow Q$. The resulting goal will be indicated with $\leftarrow Q \setminus Q'$. If there exists a refutation for $P \cup \{\leftarrow Q''\}$ which does not instantiate the free variables of $Q''$, then the derivation fails.

The derivation succeeds if it ends with the empty goal.

Note that there are derivations which neither succeed nor fail. They can be infinite, like in $\{p \leftarrow p\} \cup \{\leftarrow p\}$, or loop forever in the attempt to solve a negative conjunct, like in $\{p \leftarrow p\} \cup \{\leftarrow \exists\neg p\}$ and in $\{p \leftarrow \exists\neg p\} \cup \{\leftarrow p\}$.

In the following, we denote by $freevar(Q)$ the free variables in $Q$, i.e.

- if $Q \equiv A$, then $freevar(Q) = var(A)$,
- if $Q \equiv \exists\neg Q'$, then $freevar(Q) = \emptyset$,
- if $Q \equiv Q_1, Q_2$, then $freevar(Q) = freevar(Q_1) \cup freevar(Q_2)$.

Following Shepherdson (1989), we formalize the extended notion of refutation and tree. We call them *SLDNI-refutation* and *SLDNI-tree*, for SLD with Negation As Instantiation. First, let's make precise the notion of application of a substitution to a $\exists\neg general$ query.

**Definition 4.3** Let $Q$ be a $\exists\neg general$ query and $\theta$ be a substitution. Then

$$Q\theta = \begin{cases} A\theta & \text{if } Q = A \\ Q & \text{if } Q = \exists\neg Q' \\ Q_1\theta, Q_2\theta & \text{if } Q = Q_1, Q_2. \end{cases}$$

Analogously, the notation $Q[\bar{d}/\bar{x}]$ indicates the replacement of $\bar{d}$ for the free occurrences of $\bar{x}$ in $Q$.

**Definition 4.4 (SLDNI-resolution)** Let $P$ be a $\exists\neg general$ program, $R$ a selection rule and $G$ a $\exists\neg general$ goal.

- An *SLDNI-tree of rank k* for $P \cup \{G\}$, via $R$, is a tree $TR$ defined as follows:
  - $(a)$ Every node is a $\exists\neg general$ goal and every edge is labeled by a substitution,
  - $(b)$ the root node is $G$,
  - $(c)$ for every node $\leftarrow Q$
    - $(i)$ if the selected conjunct is an atom $A$, then for each clause $H \leftarrow Q'$ in $P$ (standardized apart), if $H$ and $A$ are unifiable with mgu $\theta$, then the goal $\leftarrow ((Q \setminus A), Q')\theta$ is a child of $\leftarrow Q$, and the edge is labeled by $\theta$. If there are no such children the node is a (failed) leaf node;

(*ii*) if the selected conjunct is $\exists\neg Q'$ and there exists a finitely instantiating SLDNI-tree for $P \cup \{\leftarrow Q'\}$ of rank $k' < k$, then $\leftarrow Q \setminus \exists\neg Q'$ is the only child of $\leftarrow Q$, and the edge is labeled by $\varepsilon$. If there exists a non-instantiating SLDNI-refutation of rank $k' < k$ for $P \cup \{\leftarrow Q'\}$, then $\leftarrow Q$ is a (failed) leaf node.

- *TR* is *finitely instantiating* at level $h$ iff it instantiates $freevar(Q)$ at level $h$ (see Definition 3.2).
- An *SLDNI-refutation* $\xi$ *of rank* $k$ for $P \cup \{G\}$ is a sequence of goals $\xi = G_0, G_1, \ldots, G_n$, with $G_0 = G$ and $G_n = \Box$, and an associated sequence of substitutions $\theta_0, \ldots, \theta_{n-1}$ such that for each $i \in [0, n-1]$, either

  (*i*) the selected conjunct in $G_i$ is an atom $A$, there exists a clause $H \leftarrow Q$ in $P$ (standardized apart) such that $H$ is unifiable with $A$ with mgu $\theta_i$ and $G_{i+1} = \leftarrow ((G_i \setminus A), Q)\theta_i$, or

  (*ii*) the selected conjunct in $G_i$ is $\exists\neg Q$, there exists a finitely instantiating SLDNI-tree of rank $k' < k$ for $P \cup \{\leftarrow Q\}$, $G_{i+1} = G_i \setminus \exists\neg Q$ and $\theta_i = \varepsilon$.

  The substitution $(\theta_0 \cdots \theta_{n-1})_{|_{(G)}}$ is the *computed answer substitution*.
- $\xi$ is *non-instantiating* if $Inst(\theta_0 \cdots \theta_{n-1}, freevar(G))$ is false.

Note that a particular case of finitely instantiating SLDNI tree is a tree in which all branches fail, which we call finitely failed SLDNI tree.

## 4.1 Correctness of SLDNI-resolution

In this section we define the completion of a $\exists\neg$*general* program and we show that the SLDNI-resolution is sound with respect to it, thus extending the results of Section 3 to $\exists\neg$*general* programs.

The completion of a $\exists\neg$*general* program is just the straightforward extension of the standard notion of completion recalled in Definition 3.19. However we need here to give more details about its definition, because we are going to use it in the proofs.

**Definition 4.5** Let $P$ be a $\exists\neg$*general* program. Its completion with respect to the extended language $L$ is defined in the usual way, namely $Comp_L(P) = IFF(P) \cup CET_L$. The set $CET_L$ is the set of the Clark's equality axioms for the language $L$ (Clark, 1978). The set $IFF(P)$ is the collection of completed definitions of predicates in $P$, defined as follows.
Let $p$ be a predicate occurring in the program and $x_1, \ldots, x_n$ fresh variables. Assume there are $s \geq 0$ clauses in $P$ defining $p$ (i.e. with head predicate $p$). For $i \in [1, s]$, let $p(\bar{t_i}) \leftarrow Q_i$ be the $i$-th clause in $P$ defining $p$. Let $E_i$ be the formula $\exists \bar{y_i}.\ \bar{x} = \bar{t_i} \wedge Q_i$, where $\bar{y_i}$ denotes all the (free) variables in $p(\bar{t_i}) \leftarrow Q_i$. The completed definition of $p$ in $P$ is

$$p(\bar{x}) \leftrightarrow \bigvee_{i \in [1,s]} E_i.$$

We remind that the symbol "," in the queries has to be interpreted as the logical conjunction "$\wedge$". In the following, when the program $P$ is clear from the context, we write $\models F$ for $Comp_L(P) \models F$.

**Proposition 4.6** *Let $P$ be a $\exists\neg$general program, and $\leftarrow Q$ a $\exists\neg$general goal. Consider an SLDNI-tree for $P \cup \{\leftarrow Q\}$. Assume the selected conjunct in $\leftarrow Q$ to be an atom $A$. Let $\bar{x} = \{x_1, \ldots, x_m\} \subseteq freevar(Q)$ $(m \geq 0)$.*

1. *If $\leftarrow Q'$ is a child of $\leftarrow Q$, with associated substitution $\theta$, then*

$$Comp_L(P) \models Q\theta \leftarrow Q'.$$

2. *If* $\leftarrow Q_1, \ldots, \leftarrow Q_n$ $(n \geq 0)$ *are all the children of* $\leftarrow Q$ *whose associated substitutions* $\theta_1, \ldots, \theta_n$ *satisfy* $\forall i \in [1, n]$ $\neg Inst(\theta_i, \bar{x})$*, then*

$$Comp_L(P) \models \neg Q[\bar{d}/\bar{x}] \leftarrow \bigwedge_{i \in [1,n]} \forall \bar{y}_i .\, \neg Q_i[\bar{d}/\bar{x}\theta_i],$$

*where* $\bar{d} = d_1, \ldots, d_m$ *are (some of) the additional constant symbols of* $L$*, and the* $\bar{y}_i$*'s are all the local variables of the clauses used in the derivation steps, after renaming.*

**Proof**

1. This part extends to $\exists\neg general$ programs the result stated (without proof) in Apt (1990), Lemma 3.1.

   Let $H \leftarrow Q''$ be the selected clause. Then

$$\begin{array}{l} \models H \leftarrow Q'' \\ \Rightarrow \\ \models H\theta \leftarrow Q''\theta \\ \Rightarrow \quad \{\text{since } A\theta = H\theta\} \\ \models A\theta \leftarrow Q''\theta \\ \Rightarrow \\ \models (Q \setminus A)\theta, A\theta \leftarrow (Q \setminus A)\theta, Q''\theta \\ \Rightarrow \quad \{\text{since } Q' = ((Q \setminus A), Q'')\theta = (Q \setminus A)\theta, Q''\theta\} \\ \models Q\theta \leftarrow Q'. \end{array}$$

2. Assume $A = p(\bar{t})$. Let $p(\bar{y}) \leftrightarrow \bigvee_{j \in [1,s]} \exists \bar{z}_j .\bar{y} = \bar{u}_j \wedge Q'_j$ be the completed definition of $p$, where all variables are renamed according to the standardization apart. Then

$$\begin{array}{l} \models A \leftrightarrow \bigvee_{j \in [1,s]} \exists \bar{z}_j .\, \bar{t} = \bar{u}_j \wedge Q'_j \\ \Rightarrow \\ \models A[\bar{d}/\bar{x}] \leftrightarrow \bigvee_{j \in [1,s]} \exists \bar{z}_j .\, \bar{t}[\bar{d}/\bar{x}] = \bar{u}_j \wedge Q'_j. \end{array}$$

   Observe that, for $j \in [1, s]$, if $\bar{t}$ and $\bar{u}_j$ are not unifiable, then the component $F_j = \exists \bar{z}_j .\, \bar{t}[\bar{d}/\bar{x}] = \bar{u}_j \wedge Q'_j$ is false in $Comp_L(P)$, hence we can discard it from the disjunction in the right hand side.

   Analogously, if there exists $\mu_j = mgu(\bar{t}, \bar{u}_j)$, but $Inst(\mu_j, \bar{x})$ holds, then $\bar{t}[\bar{d}/\bar{x}]$ and $\bar{u}_j$ are not unifiable, hence we can discard the component $F_j$.

   Finally, if $\mu_j = mgu(\bar{t}, \bar{u}_j)$ and $\neg Inst(\mu_j, \bar{x})$, then $mgu(\bar{t}[\bar{d}/\bar{x}], \bar{u}_j) = \mu_j[\bar{d}/\bar{x}\mu_j]$, hence

$$\models F_j \rightarrow \exists \bar{z}_j .\, Q'_j \mu_j [\bar{d}/\bar{x}\mu_j].$$

   Let $j_1, \ldots, j_n \in [1, s]$ be the indexes such that $\exists$ $mgu(\bar{t}, \bar{u_{j_k}}) = \mu_{j_k}$ and $\neg Inst(\mu_{j_k}, \bar{x})$. We have

$$
\begin{aligned}
& \models A[\bar{d}/\bar{x}] \rightarrow \textstyle\bigvee_{k\in[1,n]} \exists \bar{z}_{j_k}.\ Q'_{j_k}\mu_{j_k}[\bar{d}/\bar{x}\mu_{j_k}] \\
\Rightarrow \quad & \{\text{since } Q = (Q \setminus A), A\} \\
& \models Q[\bar{d}/\bar{x}] \rightarrow \textstyle\bigvee_{k\in[1,n]} \exists \bar{z}_{j_k}.\ ((Q \setminus A), Q'_{j_k})\mu_{j_k}[\bar{d}/\bar{x}\mu_{j_k}] \\
\Leftrightarrow \quad & \{\text{definition of the } Q_i\text{'s, } \bar{y}_i\text{'s, and } \theta_i\text{'s}\} \\
& \models Q[\bar{d}/\bar{x}] \rightarrow \textstyle\bigvee_{i\in[1,n]} \exists \bar{y}_i.\ Q_i[\bar{d}/\bar{x}\theta_i] \\
\Leftrightarrow \quad & \\
& \models \neg Q[\bar{d}/\bar{x}] \leftarrow \textstyle\bigwedge_{i\in[1,n]} \forall \bar{y}_i.\ \neg Q_i[\bar{d}/\bar{x}\theta_i].
\end{aligned}
$$

□

Note that for $m = 0$ (i.e. $\bar{x} = \emptyset$) Proposition 4.6(2) corresponds to Lemma 3.15 of Lloyd (1987), and for $n = 0$ (i.e. when all resolvents instantiate $\bar{x}$) it reduces to $Comp_L(P) \models \neg Q[\bar{d}/\bar{x}]$.

**Theorem 4.7 (Correctness of SLDNI-resolution)** *Let $P$ be a $\exists\neg$general program and $\leftarrow Q$ a $\exists\neg$general goal. Then*

1. *if $P \cup \{\leftarrow Q\}$ has an SLDNI-refutation with c.a.s. $\theta$, then $Comp_L(P) \models Q\theta$,*
2. *if $P \cup \{\leftarrow Q\}$ has a finitely instantiating SLDNI-tree, then $Comp_L(P) \models \exists\neg Q$.*

**Proof** By mutual generalized induction on the rank $k$.

1. For this part we reason by induction w.r.t. both the rank $k$ and the length $n$ of the refutation. We use the lexicographic ordering on pairs $\langle k, n\rangle$ of natural numbers. In this ordering

   $$\langle k_1, n_1\rangle < \langle k_2, n_2\rangle \text{ iff } k_1 < k_2 \text{ or } (k_1 = k_2 \text{ and } n_1 < n_2).$$

   If $n = 0$, then $\leftarrow Q$ is the empty goal.

   Assume now $n > 0$. We distinguish the two cases.

   ($i$) The selected conjunct is an atom. Let $\mu$ be the mgu and $\leftarrow Q'$ the resolvent of $\leftarrow Q$. Then $P \cup \{\leftarrow Q'\}$ has an SLDNI-refutation of length $n-1$ with a c.a.s. $\theta'$ such that $Q\mu\theta' = Q\theta$. By inductive hypothesis, $\models Q'\theta'$ holds. Hence, by Proposition 4.6(1), we have $\models Q\mu\theta'$, i.e. $\models Q\theta$.

   ($ii$) The selected conjunct is a negative query $\exists\neg Q'$, and $P \cup \{\leftarrow Q'\}$ has a finitely instantiating SLDNI-tree of rank $k' < k$. Furthermore, $P \cup \{\leftarrow Q \setminus \exists\neg Q'\}$ has an SLDNI-refutation of length $n-1$ with a c.a.s. $\theta$. Then, by inductive hypothesis, we have $\models \exists\neg Q'$ and $\models (Q \setminus \exists\neg Q')\theta$. Finally observe that $(Q \setminus \exists\neg Q')\theta, \exists\neg Q' = Q\theta$ holds, since, by Definition 4.3, $(\exists\neg Q')\theta = \exists\neg Q'$. Therefore we have $\models Q\theta$.

2. For this part we reason by induction w.r.t. both the rank $k$ and the instantiation level $h$. We use the lexicographic ordering on pairs $\langle k, h\rangle$ of natural numbers. Note that changing the ordering w.r.t. Part 1 is harmless, since the mutual inductive hypothesis is done on $k$ strictly smaller, and $k$ is the first component of the pairs in both the lexicographic orderings.

   Assume that $P \cup \{\leftarrow Q\}$ has an SLDNI-tree which finitely instantiates at level $h$ some variables $\bar{x} = x_1, \ldots, x_m \in \mathit{freevar}(Q)$. We prove a stronger property, namely that $Comp_L(P) \models \neg Q[\bar{d}/\bar{x}]$ holds, where $\bar{d} = d_1, \ldots, d_n$ are some of the additional constant symbols of $L$. Consider the two cases:

$(i)$ The selected conjunct is an atom.

Consider all the children $\leftarrow Q_1, \ldots, \leftarrow Q_n$ $(n \geq 0)$ of $\leftarrow Q$, whose associated substitutions $\theta_1, \ldots, \theta_n$ satisfy $\neg Inst(\theta_i, \bar{x})$. By definition, for each $i \in [1, n]$, $P \cup \{\leftarrow Q_i\}$ has an SLDNI-tree which finitely instantiates $\bar{x}\theta_i$ at level $h_i < h$ (note that $\bar{x}\theta_i$ is a sequence of variables). By inductive hypothesis, $\forall i \in [1, n]$, $\models \neg Q_i[\bar{d}/\bar{x}\theta_i]$, hence $\models \bigwedge_{i\in[1,n]} \forall \bar{y}_i.\neg Q_i[\bar{d}/\bar{x}\theta_i]$ holds, where the $\bar{y}_i$ are the local variables of the clauses used in the derivation steps, after renaming. Finally, apply Proposition 4.6(2).

$(ii)$ The selected conjunct is a negative query $\exists\neg Q'$. Consider the two cases:

$(a)$ $P \cup \{\leftarrow Q'\}$ has an SLDNI-refutation of rank $k' < k$ with a c.a.s. $\theta$ such that $\neg Inst(\theta, freevar(Q'))$, namely there exists $\theta'$ such that $Q'\theta\theta' = Q'$. By inductive hypothesis, $\models Q'\theta$ holds, hence we deduce $\models Q'\theta\theta'$, i.e. $\models Q'$. By simple formula manipulation we have:

$$\models Q'$$

$$\Leftrightarrow$$

$$\models \neg\exists\neg Q'$$

$$\Rightarrow$$

$$\models \neg(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}] \vee \neg\exists\neg Q'$$

$$\Rightarrow$$

$$\models \neg((Q \setminus \exists\neg Q')[\bar{d}/\bar{x}], \exists\neg Q').$$

Finally, observe that $(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}], \exists\neg Q' = Q[\bar{d}/\bar{x}]$.

$(b)$ $P \cup \{\leftarrow Q'\}$ has a finitely instantiating SLDNI-tree of rank $k' < k$, so the (only) child of $\leftarrow Q$ is $\leftarrow Q \setminus \exists\neg Q'$ and $\models \exists\neg Q'$. By hypothesis, $P \cup \{\leftarrow Q \setminus \exists\neg Q'\}$ has an SLDNI-tree which finitely instantiates $\bar{x}$ at level $h' < h$. By inductive hypothesis we have $\models \neg(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}]$. By simple formula manipulation:

$$\models \neg(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}]$$

$$\Rightarrow$$

$$\models \neg(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}] \vee \neg\exists\neg Q'$$

$$\Rightarrow$$

$$\models \neg((Q \setminus \exists\neg Q')[\bar{d}/\bar{x}], \exists\neg Q').$$

Finally, use again the equality $(Q \setminus \exists\neg Q')[\bar{d}/\bar{x}], \exists\neg Q' = Q[\bar{d}/\bar{x}]$.

□

## 4.2 Incompleteness of SLDNI-resolution

The SLDNF-resolution is known to have two major drawbacks from which incompleteness arises. The first is what Clark (1978) called a *floundering*, which occurs when a derivation ends in a goal containing only non-ground negative literals. The more restricted use of NAF imposes that in this situation the computation arrests without any conclusion (neither success nor failure). The more liberal version (see for instance Lloyd (1987), § 15) allows to go on in some cases: $\leftarrow \neg A$ succeeds if $\leftarrow A$ has a failed SLD-tree, and it fails if $\leftarrow A$ has a refutation which does not instantiate the variables of $A$. However, the computation still arrests if none of these possibilities occurs. In particular, this happens when $\leftarrow A$ has a refutation which instantiates the variables of $A$.

The second problem is what Shepherdson (1985) calls a *dead end*, and it is related to loops. In case there is an infinite fair computation for $\leftarrow A$, then the mainstream evaluation of $\leftarrow \neg A$ is stuck because no answer is received about the subsidiary evaluation.

The SLDNI-resolution excludes floundering, but it suffers from the dead end problem.

**Example 4.8**

$$
\begin{aligned}
P \quad = \quad \{ \quad & p \leftarrow \exists\neg q(x), \\
& p \leftarrow q(x), \\
& q(x) \leftarrow q(x) \quad \}
\end{aligned}
$$

As one can easily verify, $Comp_L(P) \models p$ holds, but $P \cup \{\leftarrow p\}$ has no refutations. This is due to a dead end: neither SLDNI-refutations, nor finitely instantiating SLDNI-trees can be constructed for the subgoal $\leftarrow q(x)$.

Of course, if the program and the query are positive (i.e. they do not contain existential negative conjuncts), the dead end problem cannot occur because the NAI rule is never invoked during the computation. So, in this case we have for the NAI-rule a completeness result (see Proposition 3.23 and its extension to the non-atomic goals) which, paraphrasing Shepherdson's terminology, we call $\exists\neg$*completeness* of SLDNI-resolution.

Observe that $\exists\neg$*completeness* is a further way to intend completeness in addition to the three ones considered by Shepherdson (1988) for SLDNF-resolution:

**($\theta$-completeness)** If $Comp(P) \models Q\theta$, then $P \cup \{\leftarrow Q\}$ succeeds with a c.a.s. $\theta'$ s.t. $\theta' \leq \theta$,

**($\neg$-completeness)** If $Comp(P) \models \neg\exists Q$, then $P \cup \{\leftarrow Q\}$ fails,

**($\exists$-completeness)** If $Comp(P) \models \exists Q$ then $P \cup \{\leftarrow Q\}$ succeeds.

We can consider $\theta$-completeness as the dual concept of $\neg$-completeness, both from a logical and from an operational point of view.

The $\exists\neg$*completeness* can be thought of as the dual concept of $\exists$-completeness and stated as follows:

**($\exists\neg$completeness)** If $Comp_L(P) \models \exists\neg Q$ then $P \cup \{\leftarrow Q\}$ instantiates *freevar*$(Q)$.

# 5 Amalgamating NAF and NAI

A major limitation of the language presented in Section 4 is that the negative components of a query cannot share variables with the positive ones. The situation is in a sense opposite to what happens in SLDNF-resolution, where, in order to avoid floundering, all the variables in negative literals must occur in some positive atom too (in the clause or in the query). It comes then natural to try an amalgamation of NAF and NAI in order to reduce the limitations of the two methods. This can be done by allowing the presence of expressions like $\exists\bar{x}\neg Q$, where $\bar{x}$ is a (possibly empty) sequence of variables, and by modifying the NAI rule so that $\leftarrow \exists\bar{x}\neg Q$ is evaluated successfully in $P$ if there exists a tree which finitely instantiates $\bar{x}$. The NAF rule and the NAI rule are particular instances of this rule, obtained with $\bar{x} = \emptyset$ and $\bar{x} = freevar(Q)$ respectively.

Concerning the treatment of failure, it is still correct to infer the failure of $P \cup \{\leftarrow \exists\bar{x}\neg Q\}$ when there exists a refutation for $P \cup \{\leftarrow Q\}$ which does not instantiate the variables of $Q$. Note that, for correctness, it is necessary to consider all variables of $Q$, not only $\bar{x}$. This is similar to the NAF case.

**Example 5.1** Consider the program

$$
\begin{aligned}
P \quad = \quad \{ \quad & p \leftarrow \exists\neg q, \\
& q \leftarrow \exists x\neg r(x, y), \\
& r(x, a) \leftarrow \qquad \}
\end{aligned}
$$

A rule allowing to derive the falsity of $q$ from the refutation of $\leftarrow \neg r(x, y)$ would be incorrect, because $Comp_L(P) \not\models p$. It would also be inconsistent, because actually $Comp_L(P) \models \neg p$ Note that such a refutation instantiates $y$.

The arrest of the computation (with no conclusion) is limited to the situation in which there exists a refutation instantiating $freevar(Q) \setminus \bar{x}$. Note that our treatment of negation (for $\bar{x} = \emptyset$) is more general than the standard SLDNF-resolution as formalized, for instance, by Lloyd (1987). Usually SLDNF-resolution requires a selected negative literal to be ground, otherwise it flounders. We allow the selected conjunct to be not ground, and stop the computation (with no conclusion) only if there exists a refutation instantiating its free variables.

The language amalgamating NAF and NAI is defined as follows:

| | |
|---|---|
| Queries | $Q ::= A \mid \exists\bar{x}\neg(Q) \mid Q, Q$ |
| Goals | $G ::= \leftarrow Q$ |
| Clauses | $C ::= A \leftarrow Q \mid A \leftarrow .$ |

where $A \in Atom$ and $\bar{x} \in Var$. As usual, we omit the parentheses when there is no risk of ambiguity.

Note that this language allows us to express various kinds of general formula in the body of clauses. For instance, $\neg Q$ can be expressed as $\exists\emptyset\neg Q$; $\forall\bar{x}Q$ can be expressed as $\neg\exists\bar{x}\neg Q$; $\exists\bar{x}(Q_1 \rightarrow Q_2)$ can be expressed as $\exists\bar{x}\neg(Q_1, \neg Q_2)$; etc.

The notion of conjunct is modified according to the new grammar in the obvious way. So, a conjunct is either an atom or an expression of the form $\exists\bar{x}\neg Q$. As one would expect, the application of a substitution to a query generalizes as follows:

$$Q\theta = \begin{cases} A\theta & \text{if } Q = A \\ \exists\bar{x}(\neg Q'\theta|_{\setminus\bar{x}}) & \text{if } Q = \exists\bar{x}\neg Q' \\ Q_1\theta, Q_2\theta & \text{if } Q = Q_1, Q_2. \end{cases}$$

From this definition it appears clear that quantified negative conjuncts can now communicate, i.e. share variables, with the other conjuncts.

We present now the resolution mechanism for the amalgamated language, which we call SLDNFI-resolution. This is obtained by modifying Definition 4.4 according to the above described treatment of formulas like $\exists\bar{x}\neg Q$.

**Definition 5.2 (SLDNFI-resolution)**

- The notion of SLDNFI-tree is obtained by replacing the point c(ii) in Definition 4.4 by the following condition:

  (*ii*) if the selected conjunct is $\exists\bar{x}\neg Q'$, then

  - if there exists an SLDNFI-tree for $P \cup \{\leftarrow Q'\}$ of rank $k' < k$ which finitely instantiates $\bar{x}$, then $\leftarrow Q \setminus \exists\bar{x}\neg Q'$ is the only child of $\leftarrow Q$, and the edge is labeled by $\varepsilon$.
  - if there exists an SLDNFI-refutation for $P \cup \{\leftarrow Q'\}$ of rank $k' < k$ which does not instantiate $freevar(Q')$, then $\leftarrow Q$ is a (failed) leaf node.
    (Note that the restriction concerning instantiation involves all the variables of $Q'$, not only $\bar{x}$).

- The notion of SLDNFI-refutation is obtained by replacing the corresponding point (*ii*) in Definition 4.4 by the condition:

($ii$) the selected conjunct in $G_i$ is $\exists\bar{x}\neg Q$, there exists an SLDNFI-tree of rank $k' < k$ for $P \cup \{\leftarrow Q\}$ which finitely instantiates $\bar{x}$, $G_{i+1} = G_i \setminus \exists\bar{x}\neg Q$ and $\theta_i = \varepsilon$.

The following result states the correctness of SLDNFI-resolution, thus generalizing Theorem 4.7.

**Theorem 5.3 (Correctness of SLDNFI-resolution)** *Let $P$ be an amalgamated program, $\leftarrow Q$ an amalgamated goal and $\bar{x} \subseteq freevar(Q)$. Then*

1. *if $P \cup \{\leftarrow Q\}$ has an SLDNFI-refutation with c.a.s. $\theta$, then $Comp_L(P) \models Q\theta$,*

2. *if $P \cup \{\leftarrow Q\}$ has an SLDNFI-tree finitely instantiating $\bar{x}$, then $Comp_L(P) \models \exists\bar{x}\neg Q$.*

**Proof** Slight modification of the proof of Theorem 4.7. In 1($ii$) replace $\exists\neg Q'$ by $\exists\bar{z}\neg Q'$, and observe that $\models \exists\bar{z}\neg Q' \Rightarrow \models (\exists\bar{z}\neg Q')\theta$, and that $(Q \setminus \exists\bar{z}\neg Q')\theta, (\exists\bar{z}\neg Q')\theta = Q\theta$.

In 2($ii$)($a$) and 2($ii$)($b$) replace $\exists\neg Q'$ by $\exists\bar{z}\neg Q'$, and observe that $(Q \setminus \exists\bar{z}\neg Q')[\bar{d}/\bar{z}], \exists\bar{z}\neg Q' = Q[\bar{d}/\bar{z}]$. Furthermore, in 2($ii$)($a$) observe that $\models Q' \Rightarrow \models \forall\bar{z}Q' \Leftrightarrow \models \neg\exists\bar{z}\neg Q'$.

The rest of the proof remains the same. □

As a particular case of Theorem 5.3, when $\bar{x} = \emptyset$ we obtain:

**Remark 5.4** *If $P \cup \{\leftarrow Q\}$ has a finitely failed SLDNFI-tree, then $Comp_L(P) \models \neg Q$, i.e. $Comp_L(P) \models \forall\neg Q$.*

This can be used as a basis for the introduction of a "direct" universal quantifier in the language, thus improving the efficiency of the interpreter. We do not investigate further this possibility here, but we give an example of how this construct could be used.

**Example 5.5** In logic programming sets are usually represented as lists, and the "membership" relation is defined as

$$\begin{aligned} Member = \{ \; & member(x, [x|y]) \leftarrow, \\ & member(x, [z|y]) \leftarrow member(x, y) \quad \}. \end{aligned}$$

Consider now the subset relation, formally expressed by

$$\begin{aligned} Y \subseteq Z \quad & \Leftrightarrow \quad \forall x(x \in Y \Rightarrow x \in Z) \\ & \text{i.e.} \\ Y \subseteq Z \quad & \Leftrightarrow \quad \forall x \neg(x \in Y \wedge x \notin Z) \end{aligned}$$

In our language this relation can be defined by the program

$$P = \{subset(y, z) \leftarrow \forall x \neg(member(x, y), \neg member(x, z))\} \cup Member$$

where $\forall x Q$ is an abbreviation for $\neg\exists x \neg Q$ and $\neg Q$ is an abbreviation for $\exists\emptyset\neg Q$. When we try to refute $\leftarrow subset(t, u)$, we have to evaluate three nested negations before evaluating $\leftarrow member(x, t), \neg member(x, u)$. By using Remark 5.4 we could instead directly infer

$$\forall x \neg(member(x, t), \neg member(x, u))$$

from the finite failure of the tree for $P \cup \{\leftarrow member(x, t), \neg member(x, u)\}$.

Note that the program for *subset* in Example 5.5 is "complete" w.r.t. the positive use, in the sense that, given two ground lists $t, u$ which represent two sets $T, U$ with $T \subseteq U$, then $P \cup \{\leftarrow subset(t, u)\}$ has an SLDNFI-refutation.

On the other hand, it is not complete w.r.t. the negative use, in the sense that if $T \not\subseteq U$, then $P \cup \{\leftarrow \neg subset(t, u)\}$ gives no answer, because the SLDNFI-resolution only generates SLDNFI-refutations for $P \cup \{\leftarrow member(x, t), \neg member(x, u)\}$ which instantiate $x$ to elements of $T \setminus U$.

The negative counterpart of the relation subset could better be defined by

$$P \quad = \quad \{ \quad not\_subset(y, z) \leftarrow member(x, y), \neg member(x, z) \quad \}.$$

In fact, $P \cup \{\leftarrow \neg subset(t,u)\}$ succeeds if $T \not\subseteq U$. Concerning completeness, SLDNFI-resolution suffers from the floundering problem as well as SLDNF-resolution. This problem is however limited to the case in which there is a refutation instantiating the free variables of a selected negative conjunct. Obviously, to the purpose of approximating completeness as much as possible, it is better to delay the selection of a negative conjunct until all the positive literals have been resolved, and all possible bindings generated.

Thanks to the capability to deal with existentially quantified conjuncts, hence with local variables, our treatment of negation fits better than negation as failure to be the operational counterpart of the completion semantics.

**Example 5.6** Consider the program

$$\begin{array}{rcll} P & = & \{ & p(x) \leftarrow \neg q(x,y), \\ & & & q(x,a) \leftarrow, \\ & & & r(b) \leftarrow \qquad\qquad \} \end{array}$$

We have $Comp(P) \models p(x)$, but, due to the local variable $y$, the standard SLDNF-resolution would not derive the success of $P \cup \{\leftarrow p(x)\}$, not even when modified according to Definition 5.2.

On the other hand, in our language $P$ would be naturally written as

$$\begin{array}{rcll} P' & = & \{ & p(x) \leftarrow \exists y \neg q(x,y), \\ & & & q(x,a) \leftarrow, \\ & & & r(b) \leftarrow \qquad\qquad \} \end{array}$$

and $P' \cup \{\leftarrow p(x)\}$ has an SLDNFI-refutation with empty substitution.

We end this section with another example.

**Example 5.7** Consider a binary operation $\underline{op}$, defined in Logic Programming by a ternary relation $p_{op}$ (i.e. $p_{op}(x,y,z) \Leftrightarrow op(x,y) = z$). The *neutral elements*, formally defined as

$$neutral(x) \Leftrightarrow \forall y (op(x,y) = y) \wedge (op(y,x) = y),$$

can be defined in our language by

$$neutral(x) \leftarrow \neg \exists y \neg (p_{op}(x,y,y), p_{op}(y,x,y)).$$

The opposite can be defined as

$$not\_neutral(x) \leftarrow \neg neutral(x)$$

or, more simply

$$not\_neutral(x) \leftarrow \exists y \neg p_{op}(x,y,y),$$
$$not\_neutral(x) \leftarrow \exists y \neg p_{op}(y,x,y).$$

# 6 Conclusions and future work

We have defined a failure set (the *FFI* set) which is the negative counterpart of the atomic consequences set of a program. This set is operationally characterized as the set of atoms for which every SLD-derivation either fails or instantiates some variables. Then, we have shown that *FFI* is the set of atoms whose existentially quantified negation is a logical consequence of the completion of the program $P$. This model-theoretic characterization has given the foundations of a new rule for the treatment of negated atoms, the Negation As Instantiation rule, which infers $\exists \neg A$ if $P \cup \{\leftarrow A\}$ has a "finitely instantiating" SLD-tree. The NAI rule is as easy to implement as the

NAF rule and it is orthogonal to it. We have combined the two rules into an interpreter called SLDNFI-resolution, of which we have proved the correctness.

The obstacles to completeness are the loops and the uninstantiated variables. Our idea is to try the approaches of Drabent and Martelli (1991) and of Kunen (1989) in order to overcome the problem of the loops, and to find a restriction similar to allowdness in order to avoid the problem of the variables.


# Acknowledgments

We acknowledge prof. H. Blair for his helpful comments. We also thank the anonymous referees, whose criticism and suggestions greatly contributed to the improvement of this paper.


# References


Apt, K. R. (1990), Logic Programming, *in* "Handbook of Theoretical Computer Science, Vol. B, Formal Models and Semantics" (J. Van Leeuwen, Ed.), pp. 493–574, Elsevier, Amsterdam.

Chan, D. (1988), Constructive Negation Based on the Completed Database, *in* "Proceedings, Fifth International Conference on Logic Programming" (R. A. Kowalski and K. A. Bowen, Eds.), pp. 111–125, The MIT Press, Cambridge, MA.

Clark, K. L. (1978), Negation as failure, *in* "Logic and Data Bases" ( H. Gallaire and J. Minker, Eds.), pp. 293–322, Plenum Press, New York.

Drabent, W., and Martelli, M. (1991), Strict Completion of Logic Programs, *New Generation Computing* **9** (1), 69–79.

Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. (1989), Declarative Modeling of the Operational Behavior of Logic Languages, *Theoret. Comput. Sci.* **69** (3), 289–318.

Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. (1993), A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs, *Inform. and Comput.* **103** (1), 86–113.

Fitting, M. (1985), A Kripke-Kleene semantics for logic programs, *J. Logic Programming* **2** (4), 295–312.

Kunen, K. (1989), Signed Data Dependencies in Logic Programs, *J. Logic Programming* **7** (3), 231–245.

Levi, G., Martelli, M., and Palamidessi, C. (1990), Failure and success made symmetric, *in* "Proceedings, North American Conference on Logic Programming" (S. Debray and M. Hermenegildo, Eds.), pp. 3–22, The MIT Press, Cambridge, MA.

Lloyd, J. W. (1987), "Foundations of Logic Programming", 2nd Ed., Springer-Verlag, Berlin.

Shepherdson, J. C. (1985), Negation as Failure II, *J. Logic Programming* **2** (3), 185–202.

Shepherdson, J. C. (1988), Negation in logic programming, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 19–88, Morgan Kaufmann, Los Altos, CA.

Shepherdson, J. C. (1989), A sound and complete semantics for a version of negation as failure, *Theoret. Comput. Sci.* **65** (3), 343–371.