An Embedding of Calculi for Negation as Failure into Linear Logic

Giorgio Delzanno and Maurizio Martelli
Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova
via Dodecaneso 35, 16146 Genova, Italy
email: {giorgio,martelli}@disi.unige.it

Abstract

We propose a new *linear logic* completion with the aim of filling the gap between previous works relating linear logic and *negation as failure* [2, 4, 7] and sequent calculi used to encode the evaluation of normal programs [13, 9, 14]. More precisely, via an embedding of the SLDNF calculus of Lifschitz in Forum [12], we present a linear logic completion that provides us both a declarative (via the resulting Forum theory) and an operational (via the notion of goal-driven proofs peculiar of Forum) description of negation as failure.

Keywords. Negation as failure, sequent calculi, linear logic.

1 Introduction

The *negation as finite failure rule* of Clark [5] is defined as follows. Given a ground atom A, $\neg A$ succeeds if and only if A finitely fails, i.e., every SLD-derivation for A is finite and does not end with the empty clause. Symmetrically, $\neg A$ finitely fails if and only if A succeeds. The corresponding extension of SLD-resolution, namely SLDNF-resolution, has to deal with a formalization of derivation-trees much more complex with respect to the one for SLD-resolution. A precise definition of SLDNF-trees has been stated by Martelli and Tricomi [11] and Apt and Doets [1]. A different approach to the formalization of negation as failure is based on calculi à la Gentzen. Examples of sequent-calculi for SLD and SLDNF have been given by Mints [13], Sheperdson [14], and Lifschitz [9] among the others. These presentations are more intuitive than the original definition of [5, 10]. Furthermore, they simplify the proof of soundness and completeness results that can now be proved by induction on the structure of proof-trees.

In this paper we investigate the problem of finding general purpose logical systems in which to embed the above mentioned proof calculi for negations as failure. As case-study, we show that the Lifschitz's proof calculus [9] can be embedded in a proof-theoretical presentation of linear logic, namely Forum [12]. The reasons for choosing Forum are:

(1) the notion of formulas as resources peculiar of linear logic allows us to naturally encode several types of provability relations at the object level (e.g. two kind of sequents that depend on the *polarity* of goals as in Lifschitz [9]);

(2) Forum provides an interpretation of linear logic formulas in terms of *programs* and *goals* that is close to the usual notions we fnd in logic programming languages. This interpretation of linear logic theories simplifies the encoding of the operational aspects related to SLDNF resolution.

Our encoding of Lifschitza calculus in Forum shares some similarities with previous works relating linear logic and negation as failure [2, 4, 3], and, in particular, with the work of Jeavons in [7]. Specifically, in our encoding we use the linear logic connectives with a semantics close to that used in [7]. However, our approach tries to fill the gap between the logic formalization in [7] of negation as failure, and the operational description of the approaches such as [13, 14, 9]. Forum is a logical tool that seems suitable to achieve this goal. In fact, Forum provability is given via an extension of the notion of uniform proofs [12] to multi-conclusion sequent calculi. Intuitively, uniform proofs correspond to *cut-free*, *goal-driven* proofs, a notion at the basis of the operational interpretation of logic programming. The power of linear logic allows us to represent an SLDNF derivation by using a *single* Forum proof. In fact, the combination of *multiplicative dijsunction* and *additive disjunction* allows us to naturally represent special constructors (like the one used in [11]) to keep track of *alternating OR* and *AND* branches (needed to check whether a derivation can finitely fail) within a single SLD derivation. As a consequence the linear logic completion presented in this paper turns out to be both a declarative (the Forum theory encoding the completion) and an operational description (the uniform proofs associated to a given general goal) of negation as failure, with a formal correspondance with the SLDNF calculus of Lifschitz.

Plan of the paper. In Section 2 we will introduce some preliminary notions (negation as failure, linear logic programming and Forum) needed in the sequel of the paper. In Section 3 we will introduce the Lifschitz calculus for negationa as failure. In Section 4 we will present the embedding of Lifschitz calculus into Forum, and we will study its properties. In Section 5 we will address some conclusions.

2 Preliminaries

2.1 General programs

A *literal* is an atom or its negation. A *positive literal* is an atom and a *negative literal* is a negated atom. A *general goal* is a finite conjunction of literals. The empty goal is denoted by $\Box$. A *general clause* is an implication A:-G where G is a general goal. When the body is empty the clause is called a *fact.* A *general program* is a set of general clauses. A clause C defines a predicate p if $C = p(\bar{t})$:-G for some $\bar{t}$ and some G. Hence, a program P defines a predicate p if a clause of P defines p. In the sequel we will use $\bar{t}$ to denote vectors of terms $t_1, \ldots, t_n$, and $=$ to denote equality constraints between terms. The expression $\bar{x} = \bar{t}$ denotes the conjunction $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$.

2.2 Negation as failure

Many efforts have been spent in order to formalize the operational meaning of SLDNF-resolution. SLDNF-resolution is an extension of SLD-resolution to general programs and goals, stating that $\neg A$ suceeeds iff A finitely fails, and that $\neg A$ finitely fails iff A succeeds.

Differently from the case of SLD-derivation, where the search space is represented by a tree and a derivation by a path in the tree, in the case of SLDNF-resolution a derivation is a tree itself. Consider the program $P = \{A\text{:-}B, A\text{:-}C\}$ then, an SLD-derivation for $\neg A$ is given as follows:

$$\begin{tikzcd}
fail & & fail & \\
B \arrow[u] & & C \arrow[u] & \square \\
& A \arrow[ul] \arrow[ur] & & \neg A \arrow[u] \arrow[ll]
\end{tikzcd}$$

In fact, it is necessary to inspect the SLDNF-tree for A to be able to answer to the query $\neg A$. Thus, SLDNF-derivations and SLDNF-trees interleave in the derivation of a general goal. In the sequel of the paper we will restrict ourselves to ground SLDNF-resolution. In particular, we will adopt *instantiation* instead of *unification* in resolution steps. The reason is that we will mainly focus on the relationships between SLDNF-derivations and Linear Logic proofs.

2.3 Linear Logic Programming

Forum [12] is a presentation of Linear Logic based on a uniform multi-conclusion sequent calculus. Forum represents a well-founded environment in which studying Linear Logic extensions of Logic programming. Following [6], in the sequel of the paper we will employ a higher-order logic programming language having the following features. Diffently from standard logic programming, in Forum it is possible to define programs consisting of multi-conclusion clauses of the form:

$$\forall\ (G_1 \,\&\ldots\&\, G_k) \Rightarrow (A_1 \parr \ldots \parr A_n \multimapinv G)$$
or
$$\forall\ (G_1 \,\&\ldots\&\, G_k) \Rightarrow (A_1 \parr \ldots \parr A_n)$$

for $k \geq 0$ and $n > 0$, where A_i for $i : 1, \ldots, n$ is an atomic formula, G_j for $j : 1, \ldots k$ is a goal formula in which $\parr$, $\&$, $\forall$, $\top$, $D \multimap G$ and $D \Rightarrow G$ may occur (D should have clausal form and G is, in turn, a goal formula). In the following, $[D]$ denotes the set of instances of a formula D. Two zone-sequents are used to distinguish bewteen unbounded (i.e. program clauses) and bounded (clauses or goals) resources. Formally, a sequent has the form $P; \Gamma \longrightarrow \Delta$, where P is set of reusable program clauses, Γ is a multiset of clauses each one usable only once, and, finally, Δ is a multiset of goals. A Forum derivation consists of a proof of a Forum sequent based on the goal-driven uniform proof system described in [12]. Intuitively, a Forum proof is built by first decomposing all goals in the right-hand side, then by applying clauses from P or from the bounded context in the left-hand side Δ, and so on, until an axiom is encountered. Goal formulas consisting of $⅋$, $\&$, and $\forall$ can be reduced deterministically to the corresponding subgoals. Specifically, proving $G_1 ⅋ G_2, \Delta$ reduces to proving G_1, G_2, Δ; $G_1 \& G_2, \Delta$ reduces to the two subproofs G_1, Δ and G_2, Δ; finally, $\forall x.G, \Delta$ reduces to the goal $G[d/x], \Delta$ with the proviso that d is a *new* constant not occurring in the current sequent. For the left rules, we will consider here proofs in which it is possible to focus on one single program clause, as specified by the following derived *backchaining rule* (an extension of SLD resolution to our clauses):

$$\frac{P;\ \longrightarrow G_1 \quad \ldots P;\ \longrightarrow G_k \quad P;\Gamma \longrightarrow G, \Delta}{P;\Gamma \longrightarrow^{D} \mathcal{A}, \Delta}\ bc$$

where $(G_1 \& \ldots \& G_k) \Rightarrow (A_1 ⅋ \ldots ⅋ A_n \multimap\!\!\!\circ\ G) \in [D]$, and $\mathcal{A} = \{A_1, \ldots, A_n\}$. Note that $G_1, \ldots, G_k$ can be viewed as conditions under which the multiset rewriting rule *rewrite* $A_1, \ldots, A_n$ *into* G can be executed. The selection of the clause D on which we focus on is specified by the following rules:

$$\frac{D, P;\Gamma \longrightarrow^{D} \mathcal{A}}{D, P;\Gamma \longrightarrow \mathcal{A}}\ decide_1 \qquad\qquad \frac{P;\Gamma \longrightarrow^{D} \mathcal{A}}{P;\Gamma, D \longrightarrow \mathcal{A}}\ decide_2$$

provided that $\mathcal{A}$ is a multiset of atomic formulas. Finally, note that, in case the selected rule has no body, we have the rule:

$$\frac{P;\ \longrightarrow G}{P;\ \longrightarrow^{D} A_1, \ldots, A_n}\ bc$$

where $G \Rightarrow (A_1 ⅋ \ldots ⅋ A_n) \in [D]$. The last rule correspond to the application of a conditional *fact*.

3 The Lifschitz Calculus for Negation as Failure

Let $\mathcal{P}$ be a set of predicate symbols, $\mathcal{F}$ a set of function symbols. We will consider general programs as defined by the following grammar:

Clauses

$$C ::= A \text{ :- } G \mid \forall C.$$
$$G ::= \mathit{true} \mid \mathit{false} \mid A \mid \neg G \mid G_1 \wedge G_2.$$

Programs

$$P ::= C, P \mid C.$$

Let $\mathcal{L}$ be the resulting language. Given a program P, we aim at finding a proof of a goal G, in the following indicated by the sequent $\vdash G$, or of a goal $\neg G$, indicated by

Laws for $\vdash$

$$\frac{}{\vdash}\; ax^+ \qquad \frac{\Vdash G \quad \vdash \Delta}{\vdash \neg G, \Delta}\; \neg^+$$

$$\frac{\vdash G, \Delta}{\vdash A, \Delta}\; bc^+$$

if $\exists G$ *s.t.* $A\text{:-}G \in [P]$.

Laws for $\Vdash$

$$\frac{\vdash G}{\Vdash \neg G, \Delta}\; \neg^-$$

$$\frac{\Vdash G_1, \Delta \quad \dots \quad \Vdash G_n, \Delta}{\Vdash A, \Delta}\; bc^-$$

for all $A\text{:-}G_1, \dots, A\text{:-}G_n \in [P]$.

Common Laws

$$\frac{\blacktriangleright \Delta}{\blacktriangleright \mathit{true}, \Delta}\; red \qquad \frac{\blacktriangleright G_1, G_2, \Delta}{\blacktriangleright G_1 \wedge G_2, \Delta}\; \wedge^{+/-}$$

Figure 1: The $\mathcal{NAF}$ proof system

the sequent $\Vdash G$. In order to formalize this notion of provability we will introduce a set of rules which generalize ground SLD-resolution with negation as failure. More precisely, we will consider sequents having the form $\vdash \Delta$ and $\Vdash \Delta$, where Δ is a *multiset* of atomic formulas $p(\bar{t})$, or *true*. We will use $\blacktriangleright \Delta$ to denote properties which apply both to $\vdash \Delta$ and $\Vdash \Delta$. The proof system in Fig. 1 describes the operational meaning of the considered logic. By considering a goal $G_1 \wedge \ldots \wedge G_n$ as a multiset (see rule $\wedge^{+/-}$), it is possible to express a successful refutation of a positive goal as the simple axiom (ax^+), namely all the subgoals have been reduced to the empty clause, by including the rule (red). The rule (bc^+) allows to compute ground resolvents of a given goal. The rule (bc^-) represents the counterpart of (ax^+) and (bc^+). In fact, in case there are no clauses with head A the rule becomes the axiom for $\Vdash A, \Delta$, i.e., the goal finitely fails selecting A. Otherwise, the goal finitely fails if all the possible ground resolvents finitely fail. The rule ($\neg^+$) tries to prove G finitely failed, in this case the result of the proof depends on Δ, whereas the rule ($\neg^-$) tries to find a refutation for G in order for the whole goal to fail. Here are some properties of the above proof system.

Proposition 3.1 *The following properties hold:*

i) The sequents $\vdash \mathit{false}, \Delta$ and $\Vdash \mathit{true}$ are not provable in $\mathcal{NAF}$.

ii) If the sequent $\vdash G$ is provable, then $\Vdash G$ is not provable. The contrary does not hold.

iii) If the sequent $\Vdash G$ is provable, then $\vdash G$ is not provable. The contrary does not hold.

iv) If the sequent $\vdash \neg G$ is provable, then $\Vdash G$ is provable, and if the sequent $\Vdash \neg G$ is provable, then $\vdash G$ is provable.

Proof 3.1 *By an inspection of the proof system. The counter examples for the reverse implications (ii) and (iii) are given by the program p:-p and the sequents* $\Vdash \neg p$ *and* $\vdash p$*, respectively.*

3.1 A variation of the calculus

It is possible to further refine the previous calculus by including the connective $\vee$ in the logic and assuming some restrictions on program definitions. Let us consider programs generated by the following grammar:

Clauses

$$C ::= p(\bar{x}) \text{ :- } M \mid \forall C.$$
$$M ::= M_1 \vee M_2 \mid G.$$
$$G ::= \mathit{true} \mid \mathit{false} \mid A \mid \neg A \mid G_1 \wedge G_2.$$

Programs

$$P ::= C, P \mid C.$$

Furthermore, consider now the class of programs obtained by P-formulas with the following restrictions:

i) each predicate $q \in \mathcal{P}$ *is defined by at most one clause of* P*.*

ii) for each predicate $q \in \mathcal{P}$ *which is not defined in* P *we add a clause* $\forall p(\bar{x})$*:-false.*

iii) $\vee$ *can only occur in clauses having the form*

$$\forall \bar{x}.p_0(\bar{x})\text{:-}p_1(\bar{x}) \vee \ldots \vee p_k(\bar{x})$$

where p *and* p_i *have the same arity, and* $p_i \neq p_j$ *for* $i \neq j$*.*

Conditions *i)* and *ii)* imply that each predicate in $\mathcal{P}$ is defined exactly by one clause. Condition *iii)* ensures that $\vee$ is just used to re-introduce non-deterministic definitions of predicates as we will shown below. We will call the resulting language $\mathcal{L}'$. The aim of introducing $\mathcal{L}'$ is to refine the proof system in Fig. 1, moving the *universal quantification* (for all clauses ...) in the side condition of rule *bc* from the meta level to the object level (there exists a 'completed' clause ...). Actually, we need a further refinement to represent failure of *matching steps*. Given a clause defining the predicate p, $p(\bar{t})$:-G let $c_{p(\bar{t})}[\bar{x}]$ be the formula $\exists \bar{y}.\ x_1 = t_1 \wedge \ldots x_n = t_n$, where $\bar{y}$ are the variables in $\bar{t}$, $\bar{x}$ are new variables. Let $\neg c_{p(\bar{t})}$ be the negation of $c_{p(\bar{t})}$. Furthermore, assume that $\models \neg c_{p(\bar{t})}[\bar{s}]$ iff $\bar{s} = \bar{t}$ is not solvable. According to the new assumptions we can reformulate the proof system of Fig. 1 as depicted in Fig. 2. It is clear now from the new rule $(\vee^-)$ that allowing $\vee$ to occur in the body of clauses mitigates the effects of restriction *i*). In fact, a program in $\mathcal{L}$ can be encoded in $\mathcal{L}'$ and viceversa. This can be achieved as follows. A predicate p defined by n C-clauses:

$$p(\bar{t_1})\text{:-}G_1, \ldots, p(\bar{t_n})\text{:-}G_n$$

Laws for $\vdash$

$$\frac{}{\vdash}\ ax^+ \qquad \frac{\Vdash A \quad \vdash \Delta}{\vdash \neg A, \Delta}\ \neg^+ \qquad \frac{\vdash G_i, \Delta}{\vdash G_1 \vee G_2, \Delta}\ \vee^+ \quad i \in \{1,2\}$$

Laws for $\Vdash$

$$\frac{}{\Vdash \mathit{false}, \Delta}\ ax^- \qquad \frac{}{\Vdash p(\bar{s}), \Delta}\ \mathit{match}\text{ if } \models \neg c_{p(\bar{t})}[\bar{s}] \qquad \frac{\vdash A}{\Vdash \neg A, \Delta}\ \neg^-$$

$$\frac{\Vdash G_1, \Delta \quad \Vdash G_2, \Delta}{\Vdash G_1 \vee G_2, \Delta}\ \vee^-$$

Common Laws

$$\frac{\blacktriangleright \Delta}{\blacktriangleright \mathit{true}, \Delta}\ \mathit{red} \qquad \frac{\blacktriangleright G_1, G_2, \Delta}{\blacktriangleright G_1 \wedge G_2, \Delta}\ \wedge^{+/-} \qquad \frac{\blacktriangleright G, \Delta}{\blacktriangleright A, \Delta}\ \mathit{bc}\text{ if } \exists G \text{ s.t. } A\text{:-}G \in [P]$$

Figure 2: The $\mathcal{NAF}$ proof system

can be defined by a $C^\vee$ clause (and n auxiliary C-clauses) by splitting p into n subpredicates:

$$p(\bar{x})\text{:-}p_1(\bar{x}) \vee \ldots \vee p_n(\bar{x}),\ p_1(\bar{t_1})\text{:-}G_1,\ \ \ldots\ p_n(\bar{t_n})\text{:-}G_n.$$

The following proposition holds.

Proposition 3.2 *Let P be an $\mathcal{L}$-program, G an $\mathcal{L}$-goal, and P', G' be their corresponding $\mathcal{L}'$ images. Then, $\vdash G$ ($\Vdash G$) is provable in the system of Fig.* 1 *if and only if $\vdash G'$ ($\Vdash G'$) is provable in the system of Fig.* 2.

Proof 3.2 The main point is to show the laws for $\Vdash$ are equivalent in the two systems. Intuitively, for each p there always exists one definition for p, and either the current goal $p(\bar{s})$ is not unifiable with the clause defining it, i.e. $\models \neg c_{p(\bar{t})}[\bar{s}]$, or the body of the definition is false, or, finally, the body is a goal G which must finitely fail, and then the rule *bc* must be applied.

4 Representing $\mathcal{NAF}$ in Linear Logic

The above introduced calculus can be encoded in a direct way in Linear Logic. The main points in the definition of the encoding are as follows:

- A *derivation tree* can be expressed in terms of a *Forum proof* by exploiting the sophisticate structure of Forum sequents and proofs. The encoding maps $\wedge$ into $\parr$, $\vee^-$ into $\&$, and $\vee^+$ into $\oplus$.

Encoding of programs : $[\![\mathbf{P}]\!]$

$$[\![C,P]\!] \;=\; [\![C]\!]\&\;[\![P]\!].$$

Encoding of clauses :

$$\begin{array}{rcl} [\![\forall C]\!] &=& \forall [\![C]\!] \\ [\![C]\!] &=& [\![C]\!]^{+} \;\&\; [\![C]\!]^{-} \\ [\![A\text{:-}M]\!]^{+} &=& (A \parr \boxplus) \;\circ\!-\; ([\![M]\!]^{+} \parr \boxplus) \\ [\![A\text{:-}M]\!]^{-} &=& [\![A\text{:-}M]\!]^{-}_{1} \;\&\; [\![A\text{:-}M]\!]^{-}_{2} \end{array}$$

where

$$\begin{array}{rcl} [\![p(\bar{t})\text{:-}M]\!]^{-}_{1} &=& (p(\bar{t}) \parr \boxminus) \;\;\circ\!-\;\; ([\![M]\!]^{-} \parr \boxminus). \\ [\![p(\bar{t})\text{:-}M]\!]^{-}_{2} &=& \forall \bar{x}.\; [\![\neg c_{p(\bar{t})}[\bar{x}]]\!] \;\;\Rightarrow\;\; ((p(\bar{x}) \parr \boxminus) \;\;\circ\!-\;\; \mathit{false} \parr \boxminus). \end{array}$$

$$[\![\neg c_{p(t_1,\ldots,t_n)}[x_1,\ldots,x_n]]\!] \;=\; \forall \bar{y}.(x_1 = t_1 \;\parr\; \ldots \;\parr\; x_n = t_n \;\parr\; \boxminus),$$

where $\bar{y}$ are the free variables in $t_1,\ldots,t_n$.

Encoding of goals :

$$\begin{array}{rcl} [\![M_1 \vee M_2]\!]^{+} &=& [\![M_1]\!]^{+} \;\oplus\; [\![M_2]\!]^{+}. \\ [\![M_1 \vee M_2]\!]^{-} &=& [\![M_1]\!]^{-} \;\&\; [\![M_2]\!]^{-}. \\ [\![G_1 \wedge G_2]\!]^{\blacksquare} &=& [\![G_1]\!]^{\blacksquare} \;\parr\; [\![G_2]\!]^{\blacksquare}. \\ [\![G]\!]^{\blacksquare} &=& G \end{array}$$

for G one of $A, \neg A, \mathit{true}, \mathit{false}$;

$\blacksquare$ one of $+, -$.

Figure 3: Encoding Programs in LL.

- The two types of sequents $\vdash A_1,\ldots,A_n$ and $\Vdash A_1,\ldots,A_n$ can be represented by a LL sequent in which the atomic formulas $\boxplus$ and $\boxminus$ denote the polarity of the sequent. Precisely, the encoding is as follows: $\mathcal{N}_{LL}; \longrightarrow \mathcal{A}_1 \parr \ldots \parr A_n \parr \boxplus$ and $\mathcal{N}_{LL}; \longrightarrow \mathcal{A}_1 \parr \ldots \parr A_n \parr \boxminus$, where $\mathcal{N}_{LL}$ will be introduced in the sequel.

- Switching the polarity of sequents from $\boxplus$ to $\boxminus$ and viceversa can be achieved by using a clause combining $\circ\!-$ and $\Rightarrow$.

The two atomic formulas $\boxplus$ and $\boxminus$ denoting the polarity of sequents, must not be confused with the boolean constants *true* and *false*. The encoding of programs as in Section 3 is defined formally in Fig. 3. The LL theory that assigns a meaning to the non-logical constants *true*, *false*, $=$ of Fig. 3 is given in Fig. 4. The encoding of $\mathcal{NAF}$-sequents into Forum sequents is defined in Fig. 4, too. Let us describe the intuition behind the encoding defined in Fig. 3. We first recall that in Section 3 we have considered a restricted class of the programs built using the connective $\vee$. Let $p(\bar{t})$:-G be a clause defining p. Furthermore, suppose that $\vee$ does not occur in G. Then, $[\![\forall p(\bar{t})\text{:-}G]\!]$ is a &-conjunction (i.e. a Forum program) consisting of the following three clauses:

- $\forall\ (p(\bar{t}) \parr \boxminus) \quad \circ\!\!- \quad [\![G]\!]^- \parr \boxminus.$
- $\forall\ (p(\bar{t}) \parr \boxplus) \quad \circ\!\!- \quad [\![G]\!]^+ \parr \boxplus.$
- $\forall\ [\![\neg c_{p(\bar{t})}[\bar{x}]]\!] \ \Rightarrow\ p(\bar{x}) \parr \boxminus \circ\!\!- \mathit{false} \parr \boxminus.$

Since $\vee$ does not occur in G, it follows that $[\![G]\!]^- = [\![G]\!]^+$. Thus, the previous program is equivalent to:

- $\forall\ p(\bar{t}) \circ\!\!- [\![G]\!]^+.$
- $\forall\ [\![\neg c_{p(\bar{t})}[\bar{x}]]\!] \quad \Rightarrow \quad (p(\bar{x}) \parr \boxminus) \circ\!\!- \mathit{false}.$

The former clause allows us to simulate the $\mathcal{NAF}$-rule *bc*. The latter one instead allows us to explicitly express failure due to non-matching between the selected atomic goal and the head of the considered clause, i.e., the condition $[\![\neg c_{p(\bar{t})}[\bar{x}]]\!]$ states that (the encoding of) the conjunction $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ should fail in order for the latter rule to be applied. (The Forum laws defining the predicate $=$ are given in Fig. 4).
More in general, if $\vee$ occurs in G, then $\vee$-subformulas are mapped to $\oplus$- or $\&$-formulas depending on the *polarity* of the considered sequent.

Given a program P, let us denote with $\mathcal{N}_{LL}$ the Forum theory $[\![P]\!], \mathcal{L}^+, \mathcal{L}^-, \mathcal{L}^=$ obtained by applying the encoding of Fig. 3 to P, and by enriching $[\![P]\!]$ with the laws of Fig. 4. Furthermore, given a multiset of M-formulas $\Delta = M_1, \ldots, M_n$, let $[\![\Delta]\!]^\blacksquare$ be the multiset $[\![M_1]\!]^\blacksquare, \ldots, [\![M_n]\!]^\blacksquare$ for $\blacksquare$ one of $+, -$. As shown in Fig. 5 and Fig. 6, the $\mathcal{NAF}$-rules can be derived in Forum by exploiting the LL theory $\mathcal{N}_{LL}$. The counterpart of rule *bc* derives from the above mentioned property of the encoding of clauses without $\vee$ in the body (they can be used both in negative and positive sequents). A particular attention must be paid to the encoding of the *match* rule that models the failure due to the non-matching of arguments passed to a predicate: for a sequent with negative polarity, it is always possible to apply the clause $[\![p(\bar{t}) \text{:-} G]\!]^-$ to an atomic formula $p(\bar{s})$. In this case we have the following sequence of rules:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{N}_{LL}; \longrightarrow s_1 = t_1', \ldots, s_n = t_n', \boxminus \\ \vdots\ \forall_r + \parr_r \\ \mathcal{N}_{LL}; \longrightarrow [\![\neg c_{p(\bar{s})}[\bar{t}]]\!], \boxminus\end{array} \qquad \dfrac{}{\mathcal{N}_{LL}; \longrightarrow \mathit{false}, [\![\Delta]\!], \boxminus}\ ax^-}{\mathcal{N}_{LL}; \xrightarrow{[\![C]\!]^-} p(\bar{s}), [\![\Delta]\!], \boxminus}\ bc}{\mathcal{N}_{LL}; \longrightarrow p(\bar{s}), [\![\Delta]\!], \boxminus}\ decide_1}{}$$

where t_1' is obtained from t_1 by substituting each universally quantified variables with a newly introduced constant according to the right-introduction rule $\forall_r$ of the universal quantifier in Forum. To complete the reconstruction of the *match* rule, we need the following lemma.

Lemma 4.1 *Let $\bar{s}$ be a tuple of ground terms and $\bar{t}$ be a tuple of terms with free variables $\bar{y}$. Let $\bar{d}$ be a tuple of new constants not occurring in $\mathcal{L}^=$ and let $\bar{t'} = \bar{t}\ [\bar{d}/\bar{y}]$. Then, $\mathcal{L}^=; \longrightarrow s_1 = t_1' \parr \ldots \parr s_n = t_n', \boxminus$ is provable in Forum if and only if $\bar{s}$ does not match $\bar{t}$.*

Laws for $\vdash$: $\mathcal{L}^+$

$$\begin{array}{ll}
\boxplus. & (ax^+)\\
\mathit{true} \multimapinv \bot. & (red)\\
(A \parr \boxminus) \Rightarrow (\neg A \parr \boxplus \ \multimapinv\ \boxplus). & (\neg^+)
\end{array}$$

Laws for $\Vdash$: $\mathcal{L}^-$

$$\begin{array}{ll}
\mathit{false} \parr \boxminus \multimapinv \top. & (ax^-)\\
(A \parr \boxplus) \Rightarrow (\neg A \parr \boxminus). & (\neg^-)
\end{array}$$

Laws for equality : $\mathcal{L}^=$

$$\begin{array}{c}
(c = c) \multimapinv \mathit{true}, \text{ for each constant } c.\\
\forall X_1 \ldots X_n, Y_1, \ldots, Y_k.\\
(f(X_1, \ldots, X_n) = g(Y_1, \ldots, Y_k)) \ \multimapinv\ \mathit{false}. \text{ for } f \neq g.\\
\forall X_1 \ldots X_n, Y_1, \ldots, Y_k.\\
(f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n)) \ \multimapinv\ (X_1 = Y_1) \parr \ldots \parr (X_n = Y_n).\\
\forall X, Y, C.\ (X = C) \parr (Y = C) \ \multimapinv\ (X = C) \parr (Y = X).
\end{array}$$

Encoding of sequents : $\llbracket \blacktriangleright \Delta \rrbracket$

$$\begin{array}{l}
\llbracket \vdash \Delta \rrbracket = \llbracket P \rrbracket, \mathcal{L}^+, \mathcal{L}^-, \mathcal{L}^=; \ \longrightarrow \llbracket \Delta \rrbracket^+, \boxplus.\\
\llbracket \Vdash \Delta \rrbracket = \llbracket P \rrbracket, \mathcal{L}^+, \mathcal{L}^-, \mathcal{L}^=; \ \longrightarrow \llbracket \Delta \rrbracket^-, \boxminus.
\end{array}$$

Figure 4: The LL theory for $\mathcal{NAF}$.

Proof 4.1 (sketch) Let us consider the multiset $s_1 = t'_1, \ldots, s_n = t'_n$ Applying the rules of $\mathcal{L}^=$, each equation $s_i = t'_i$ can be reduced in a finite number of steps to a multiset $c_1, \ldots, c_k$ s.t.: c_i is *false* or c_i is the atomic formula $w = d$ where d is one of the constants in $\bar{d}$. At this point, we can apply the last laws of $\mathcal{L}^=$ which allows us to substitute w for d in all other equations having form $v = d$, yielding the equation $v = w$. Repeating this algorithm, we end up either in *false* ($\bar{s}$ and $\bar{t}$ do not match), in the empty multiset (i.e., they unify), or in the multiset $w_1 = d_1, \ldots, w_n = d_n$ with $d_i \neq d_j$, i.e., the sequent is not provable (we cannot prove that they do not unify).

The above lemma allows us to check whether the actual parameters match the arguments of a predicate by checking the provability of the encoding of $\neg c_{p(\bar{t})}[\bar{s}]$ against the theory $\mathcal{L}^=$ on a separate branch of a Forum proof. More precisely, we can state the following theorem.

Theorem 4.1 (Completeness of $\mathcal{N}_{LL}$) *Let P be a program as in Section 3, Δ a* multiset *of goals. Then, $\blacktriangleright\Delta$ has a $\mathcal{NAF}$-proof iff $\llbracket\blacktriangleright\Delta\rrbracket$ has a Forum+$\mathcal{N}_{LL}$-proof.*

Proof 4.2 First we prove that if $\blacktriangleright\Delta$ is provable then $\llbracket\blacktriangleright\Delta\rrbracket$ is provable. The proof is by induction on the length of the proof of $\blacktriangleright\Delta$. The basis of the induction has two subcases, namely (ax^+) and (ax^-), which have a counterpart in $\mathcal{N}_{LL}$as shown in Fig. 5 and Fig. 6. The inductive step easily follows by applying, case by case,

$\mathcal{NAF}$ **rules**	$\mathcal{N}_{LL}$ **rules**
(ax^+)	$\dfrac{\dfrac{}{\mathcal{N}_{LL}; \overset{(ax^+)}{\longrightarrow} \boxplus}\ bc}{\mathcal{N}_{LL}; \longrightarrow \boxplus}\ decide_1$
(red)	$\dfrac{\dfrac{\dfrac{\mathcal{N}_{LL}; \longrightarrow \llbracket\Delta\rrbracket, \blacksquare}{\mathcal{N}_{LL}; \longrightarrow \bot, \llbracket\Delta\rrbracket, \blacksquare}\ \bot_r}{\mathcal{N}_{LL}; \overset{(red)}{\longrightarrow} true, \llbracket\Delta\rrbracket, \blacksquare}\ bc}{\mathcal{N}_{LL}; \longrightarrow true, \llbracket\Delta\rrbracket, \blacksquare}\ decide_1$ *where* $\blacksquare \in \{\boxplus, \boxminus\}$
$(\vee^+)$	$\dfrac{\mathcal{N}_{LL}; \longrightarrow \llbracket G_i\rrbracket, \llbracket\Delta\rrbracket, \boxplus}{\mathcal{N}_{LL}; \longrightarrow \llbracket G_1\rrbracket \oplus \llbracket G_2\rrbracket, \llbracket\Delta\rrbracket, \boxplus}\ \oplus_r \quad i \in \{1,2\}$
$(\neg^+)$	$\dfrac{\dfrac{\dfrac{\mathcal{N}_{LL}; \longrightarrow A, \boxminus}{\mathcal{N}_{LL}; \longrightarrow A \parr \boxminus}\ \parr_r \qquad \mathcal{N}_{LL}; \longrightarrow \llbracket\Delta\rrbracket, \boxplus}{\mathcal{N}_{LL}; \overset{(\neg^+)}{\longrightarrow} \neg A, \llbracket\Delta\rrbracket, \boxplus}\ bc}{\mathcal{N}_{LL}; \longrightarrow \neg A, \llbracket\Delta\rrbracket, \boxplus}\ decide_1$

Figure 5: Deriving the $\mathcal{NAF}$ rules in Forum+$\mathcal{N}_{LL}(1)$.

the inductive hypothesis to the premises of the schemes in Fig. 5. and in Fig. 6. Let us prove the other implication. The proof is now by induction on the length of the Forum-proof for $\llbracket \blacktriangleright \Delta \rrbracket = \mathcal{N}_{LL} \longrightarrow \llbracket\Delta\rrbracket^{\blacksquare}, \blacksquare$. First notice that $\llbracket\Delta\rrbracket$ is a multiset of formulas in which only the connectives $\oplus$, $\&$, $\parr$, $\top$ and the non-logical symbols $p \in \mathcal{P}$, *true*, *false*, $\neg$, $=$, can occur. Thus, in a Forum-proof for $\llbracket \blacktriangleright \Delta \rrbracket$ only the right-rules $\oplus_r$, $\&_r$, $\parr_r$, $\top_r$ can occur. Furthermore, the only left-rules applicable are *bc* (over formulas in $\mathcal{N}_{LL}$) and *(decide$_1$)*. To prove the base case we must consider two subcases, one of length one, namely the counterpart of the rule *(ax$^+$)*, one of length two, namely the counterpart of rule *(ax$^-$)*, since there are no formulas s.t. $\llbracket F \rrbracket = \top$. In the first case $\llbracket\Delta\rrbracket$ must be equal to $\mathcal{N}_{LL} \longrightarrow \boxplus$, since the side condition of the Forum *bc* rule requires the right hand side to match the fact $\boxplus \in \mathcal{N}_{LL}$ to be correctly applied. Thus, $\Delta = \emptyset$ and $\vdash$ is provable. In the second case, Δ contains at least *false*, since by hypothesis the proof results by a *bc* step over the rule *false* $\parr \boxminus \multimapinv \top$. Thus, $\Vdash$ *false*$, \Delta$ is provable by *(ax$^-$)*. The inductive step is by cases on the last rule. If the last rule is a right-introduction rule, then it must be one of $\oplus_r$, $\&_r$, $\parr_r$. For instance, if the last rule is $\oplus_r$, then, by definition, $[\![\blacktriangleright\Delta]\!] = [\![\vdash G_1 \vee G_2, \Gamma]\!]$, i.e., $\mathcal{N}_{LL}; \longrightarrow [\![G_1]\!]^+ \oplus [\![G_2]\!]^+, [\![\Gamma]\!]^+, \boxplus$. The thesis now follows by applying the inductive hypothesis to the premises of the counterpart of *($\vee^+$)* in Fig. 5, which ensures the existence of a proof for $\vdash G_i, \Gamma$ for $i \in \{1,2\}$ and thus a proof for $\vdash G_1 \vee G_2, \Gamma$.

Finally, if the last rule is a *bc* over a clause in $\mathcal{N}_{LL}$, we first notice that it must be one of the scheme in Fig. 6. Now observe that in each scheme once a rule has been selected, i.e., a *decide*$_1$ has been applied, the sequence of rule is determined as far as the premises are introduced. For instance, let us consider the more complex one, i.e., an application of *($\neg^+$)*, when the flag is set to $\boxplus$. By construction, $[\![\blacktriangleright\Delta]\!]$ must be equal to $[\![\vdash \neg A, \Gamma]\!]$, i.e., $\mathcal{N}_{LL}; \longrightarrow \neg A, [\![\Gamma]\!]^+, \boxplus$. Now after the application of *decide*$_1$ over the rule *($\neg^+$)* we are forced, by the *focusing* property of Forum, to apply a *bc* step. This yields two premises, namely $\mathcal{N}_{LL}; \longrightarrow A \parr \boxminus$ and $\mathcal{N}_{LL}; \longrightarrow [\![\Gamma]\!]^+, \boxplus$. Now, by the uniformity of Forum proofs, we are forced to apply the right-rules to the left premise obtaining $\mathcal{N}_{LL}; \longrightarrow A, \boxminus$ and $\mathcal{N}_{LL}; \longrightarrow [\![\Gamma]\!]^+, \boxplus$. These, in turn, correspond, by definition, to $[\![\Vdash A]\!]$ and $[\![\vdash \Gamma]\!]$, respectively. Thus, we can apply the inductive hypothesis, and the $\mathcal{NAF}$-rule *($\neg^+$)* to obtain the thesis.

The other cases can be proved in a similar way.

The relationships between the two calculi become more evident if we collapse the Forum-schemes of Fig. 5 and Fig. 6 as shown in Fig. 7. Here P is the program which results by collapsing clauses with complex body (i.e., containing & and $\oplus$) and clauses with simple goals (i.e., whose goals contains at most the connective $\parr$), similarly to the step perfomed in the Horn-Clause example.

5 Conclusions

Differently from previous works on negation as failure in linear logic, we have given here a linear logic completion which has the following operational properties.

- The completion can be represented as a theory in Forum [12];
- an SLDNF derivation is represented via a Forum proof, that enjoys the property of being uniform (goal-driven);
- thanks to the interpretation of Forum proofs as computations it comes natural to give an operational interpretation of a general goal G w.r.t. a program P, by using the corresponding Forum proof of its encoding $[\![G]\!]$ w.r.t. $[\![P]\!]$.

The approaches in [2, 4, 3] successively revised by Jeavons in [7] are based on a different representation of goal and clauses. Goals are tensor of literals and the negation is the involutive negation of linear logic. Our encoding follows instead the methodology suggested in [6], in order to exploit multiplicative disjunction to represent a multiset of resources, & to create proofs with several independent branches.

As future work, it would be interesting to investigate the applicability of more general proof calculi like the *Calculus of Structures* of [8] as possible reference logic for negation as failure.

References

[1] K. R. Apt and K. Doets. A New Definition of SLDNF-resolution. *Journal of Logic Programming*, 18(2):177–190, 1994.

[2] S. Cerrito. A Linear Semantics for Allowed Logic Programs. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 219–227, Philadelphia, Pennsylvania, June 1990. IEEE Computer Society Press.

[3] S. Cerrito. A Linear Axiomatization of Negation as Failure. *Journal of Logic Programming*, 12:1–24, 1992.

[4] S. Cerrito. Negation and Linear Completion. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logic for Programming*, pages 155–194. Clarendon Press, 1992.

[5] K. L. Clark. Negation as Failure. In H.Gallaire and J.Minker, editors, *Logic and Database*, pages 293–232. Plenum, New York, 1978.

[6] G. Delzanno and M. Martelli. Proofs as Computations in Linear Logic. *Theoretical Computer Science*, 258(1-2):269–297, 2001.

[7] J. Jeavons. An Alternative Linear Semantics for Allowed Logic Programs. *Annals of Pure and Applied Logic*, 84:3–16, 1997.

[8] A. Guglielmi and L. Strassburger Non-commutativity and MELL in the Calculus of Structures In *Proc. CSL 01*, 2001.

[9] V. Lifschitz. Foundations of Logic Programming. Technical report, Dept. of Computer and Information Science, University of Texas, Austin, TX 78712, 1995.

[10] J. W. Lloyd. *Foundation of Logic Programming.* Symbolic Computation - Artificial Intelligence. Springer Verlag, 1987.

[11] M. Martelli and C. Tricomi. A New SLDNF-tree. *Information Processing Letters*, 43(2):57–62, 1992.

[12] D. Miller. Forum: A Multiple-Conclusion Specification Logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[13] G. Mints. Several Formal Systems of Logic Programming. *Computers and Artificial Intelligence*, 9(1):19–41, 1990.

[14] J. C. Sheperdson. Mints Type Deductive Calculi for Logic Programming. *Annals of Pure and Applied Logic*, 56:7–17, 1992.

$\mathcal{NAF}$ **rules**	$\mathcal{N}_{LL}$ **rules**
(ax^-)	$\dfrac{\dfrac{\dfrac{}{\mathcal{N}_{LL};\longrightarrow \top, [\![\Delta]\!], \boxminus}\ \top_r}{\mathcal{N}_{LL};\xrightarrow{(ax^-)} \mathit{false}, [\![\Delta]\!], \boxminus}\ bc}{\mathcal{N}_{LL};\longrightarrow \mathit{false}, [\![\Delta]\!], \boxminus}\ decide_1$
$(\vee^-)$	$\dfrac{\mathcal{N}_{LL};\longrightarrow [\![G_1]\!], [\![\Delta]\!], \boxminus \quad \mathcal{N}_{LL};\longrightarrow [\![G_2]\!], [\![\Delta]\!], \boxminus}{\mathcal{N}_{LL};\longrightarrow [\![G_1]\!] \& [\![G_2]\!], [\![\Delta]\!], \boxminus}\ \&_r$
$(\neg^-)$	$\dfrac{\dfrac{\mathcal{N}_{LL};\longrightarrow A, \boxplus}{\mathcal{N}_{LL};\xrightarrow{(\neg^-)} \neg A, [\![\Delta]\!], \boxminus}\ bc}{\mathcal{N}_{LL};\longrightarrow \neg A, [\![\Delta]\!], \boxminus}\ decide_1$
$(\wedge^{+/-})$	$\dfrac{\mathcal{N}_{LL};\longrightarrow [\![G_1]\!], [\![G_2]\!], [\![\Delta]\!], \blacksquare}{\mathcal{N}_{LL};\longrightarrow [\![G_1]\!] ⅋ [\![G_2]\!], [\![\Delta]\!], \blacksquare}\ \&_r$
(bc)	$\dfrac{\dfrac{\dfrac{\mathcal{N}_{LL};\longrightarrow [\![M]\!]^{\blacksquare}, [\![\Delta]\!], \blacksquare}{\mathcal{N}_{LL};\longrightarrow [\![M]\!]^{\blacksquare} ⅋ \blacksquare, [\![\Delta]\!]}\ ⅋}{\mathcal{N}_{LL};\xrightarrow{[\![C]\!]} A, [\![\Delta]\!], \blacksquare}\ bc}{\mathcal{N}_{LL};\longrightarrow A, [\![\Delta]\!], \blacksquare}\ decide_1$ where $A ⅋ \blacksquare \circ\!- [\![M]\!]^{\blacksquare} ⅋ \blacksquare \in [\ [\![C]\!]\ ]$ and $\blacksquare$ is one of $\boxplus, \boxminus$.

Figure 6: Deriving the $\mathcal{NAF}$ rules in Forum+$\mathcal{N}_{LL}(2)$.

Positive Polarity

$$\frac{}{\mathcal{N}_{LL};\longrightarrow \boxplus}\ ax^{+} \qquad \frac{\mathcal{N}_{LL};\longrightarrow A, \boxminus \quad \mathcal{N}_{LL};\longrightarrow \Delta, \boxplus}{\mathcal{N}_{LL};\longrightarrow \neg A, \Delta, \boxplus}\ \neg^{+}$$

$$\frac{\mathcal{N}_{LL};\longrightarrow G, \Delta, \boxplus}{\mathcal{N}_{LL};\longrightarrow A, \Delta, \boxplus}\ bc^{+}$$

where A :- $G \in [P]$.

Negative Polarity

$$\frac{}{\mathcal{N}_{LL};\longrightarrow \mathit{false}, \Delta, \boxminus}\ ax^{-} \qquad \frac{\mathcal{N}_{LL};\longrightarrow A, \boxplus}{\mathcal{N}_{LL};\longrightarrow \neg A, \Delta, \boxminus}\ \neg^{-}$$

$$\frac{\mathcal{N}_{LL};\longrightarrow G_1, \Delta, \boxminus \quad \ldots \quad \mathcal{N}_{LL};\longrightarrow G_n, \Delta, \boxminus}{\mathcal{N}_{LL};\longrightarrow A, \Delta, \boxminus}\ bc^{-}$$

where A :- $G_1, \ldots, A$:- $\circ\!-$ G_n are all the clauses in $[P]$ with head A.

Neutral Polarity

$$\frac{\mathcal{N}_{LL};\longrightarrow G_1, G_2, \Delta, \blacksquare}{\mathcal{N}_{LL};\longrightarrow G_1 \parr G_2, \Delta, \blacksquare}\ \wedge^{+/-} \qquad \frac{\mathcal{N}_{LL};\longrightarrow \Delta, \blacksquare}{\mathcal{N}_{LL};\longrightarrow \mathit{true}, \Delta, \blacksquare}\ \mathit{red}$$

Figure 7: The proof system $\mathcal{N}_{LL}$.