[McCHa69] J. McCarthy and P. Hayes, 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, (ed B. Meltzer & D. Michie), Edinburgh University Press.

[McDDo80] D. McDermott and J. Doyle, 1980. Non-monotonic logic I. *Artificial Intelligence* **13**, pp. 41-72.

[Moo84] R. Moore, 1984. Possible-worlds semantics for autoepistemic logic. *Proceedings of the 1984 Non-monotonic reasoning workshop*. AAAI, Menlo Park, California.

[Moo85] R. Moore, 1985. Semantical considerations on non-monotonic logic. *Artificial Intelligence* **25**, pp. 75-94.

[Nut88] D. Nute, 1988. Defeasible reasoning and decision support systems. *Decision Support Systems* **4**, pp. 97-110.

[Nut90] D. Nute, 1990. Defeasible logic and the frame problem. In *Knowledge Representation and Defeasible Reasoning.*, (eds. H. Kyburg, R Loui, and G. Carlson), Studies in Cognitive Systems. Kluwer Academic Publishers, Boston.

[NuLe86] D. Nute and M. Lewis, 1986. A users manual for d-Prolog. ACMC Research Report 01-016. The University of Georgia, Athens, Georgia.

[Pol87] J. Pollock, 1987. Defeasible reasoning. *Cognitive Science* **11**, pp. 481-518.

[Rei80] R. Reiter, 1980. A logic for default reasoning. *Artificial Intelligence* **13**, pp. 81-132.

Negation and linear completion *

Serenella Cerrito†

Abstract

We propose a declarative semantics for the class of allowed logic programs. Such a semantics is given by means of a logical theory, the linear completion of the program P, which differs from Clark's completion [CL 78] in that the underlying logic is linear [GI 87] rather than classical. With respect to such a semantics, we prove soundness and completeness of SLDNF resolution for the class of allowed programs and queries, that is, we prove that the computational notion of success of a query Q for a program P corresponds to the provability of Q in the linear completion of P and the notion of failure to the provability of the linear negation of Q. An 'extended abstract' of this paper is given in [CE90].

1 Introduction

The reasons motivating our choice of linear logic over classical or intuitionistic logic to study the problem of the semantics of logic programs may be explained as follows.

1. Classical logic is not always useful to describe the declarative content of a program because there are situations where we need a commutation of the connectives with provability which classical logic does not provide: typically, the classical provability of $A \vee B$ does not imply that either A is provable or B is provable. Below, we illustrate this by means of an example (Example 1).

2. Intuitionistic logic obviates the above difficulty but it is far from being free from defects. Unlike classical logic, it lacks an involutive negation (that is, the double negation of A is not equivalent to A).

*This research was partially supported by the Esprit Basic Research Action FIDE 3070.

†L.R.I., Bât.490, Université Paris XI, Centre d'Orsay, 91405 Orsay Cedex, France. E-mail: serena@lri.lri.fr.

Therefore it is not able to express the exchange of success and failure, namely the fact that the success of a negative query $NOT(A)$ is just the failure of A and that the failure of $NOT(A)$ is just the success of A. Moreover, like classical logic, intuitionistic logic leads to some undesired situations which are illustrated by Example 2 below.

3. In both classical and intuitionistic logic there is a hidden principle, namely Gentzen's 'contraction rule' ([GE 69]). Such an inference rule seems to be responsible for some of the difficulties that one meets when one tries to use Clark's completion to get completeness for SLDNF; more precisely, the contraction rule appears to be responsible for the difficulties encountered at the propositional level. When contraction and weakening are removed, we obtain linear logic ([GI 87]).

The following examples illustrate the above considerations.

Example 1 Let P_1 be the following program

1. $R :- P$
2. $R :- NOT(P)$
3. $P :- P$

Once it is evaluated according to the SLDNF method, the query R loops on this program. Now, the following formula is an axiom of Clark's completion of the program

$$R \leftrightarrow (P \vee \neg P)$$

Thus, R will be a theorem since $P \vee \neg P$ is classically valid. The point is that SLDNF 'internal logic' is such that a disjunction $A \vee B$ is 'provable' (succeeds) if and only if either A is 'provable' or B is 'provable' and classical disjunction does not behave like that. The reader familiar with Gentzen formalization of classical logic (that is, sequent calculus) will remark that any proof of the excluded-middle principle $A \vee \neg A$ does essentially use the contraction rule.

Example 2 Let P_2 be the trivial program whose only clause is $A :- NOT(A)$. The query A loops for this program. However $A \leftrightarrow \neg A$ is an axiom of Clark's completion, therefore if the underlying logic is classical or intuitionistic we get an inconsistent theory which trivially proves A. Notice that in the absence of the contraction rule the provability of $A \rightarrow \neg A$ and $\neg A \rightarrow A$ does not lead to the provability of all formulas.

The completeness result for SLDNF that we obtain extends the classical result which establishes completeness for the class of programs which are allowed and hierarchical [CL 78], [SH 85], [LT 86]. In the same way, it extends also the result of completeness for allowed 'strict' programs w.r.t. Clark's completion which appears in [KU 89]. Indeed, the real improvement is at the level of the analysis of the 'propositional behaviour' of programs; we have nothing really new to say at the level of the handling of variables (quantification) in logic programs. This is not surprising, because the essential novelty of linear logic with respect to classical logic lies in the analysis of propositional connectives.

The paper is organized as follows. Section 2 contains an introduction to linear logic and presents some notions and some notations used throughout the paper. In section 3 we explain how to modify Clark's completion so as to obtain the linear completion. In section 4 we define the notion of 'Realizability' of a linear sequent with respect to a program and we use this notion to prove the completeness of SLDNF w.r.t. the linear completion. In section 5 we prove soundness. Finally, in section 6 we compare our results with other related works. We assume some familiarity with logic programming and with first-order logic, in particular with Gentzen's sequent calculus [GE 69].

2 Preliminaries

2.1 An introduction to linear logic

We will present here a short survey of the basic ideas of linear logic and we will introduce the formal system which we will use. For a more precise account of linear logic, see [GI 87]. Linear sequent calculus — as well as the better known intuitionistic sequent calculus — is essentially a modification of Gentzen's classical sequent calculus ([GE 69]). A classical sequent is an expression of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sequences of formulas. A classical sequent

$$\Gamma_1, ..., \Gamma n \vdash \Delta_1, ..., \Delta m$$

is true when the implication

$$\Gamma_1 \wedge ... \wedge \Gamma_n \rightarrow \Delta_1 \vee ... \vee \Delta_m$$

is true. When the sequent is intuitionistic, Δ contains at most one formula; this restriction has immediately an effect on the inference rules; for example, in the intuitionistic calculus one does not have the right-contraction rule any longer. The modifications induced by the linear approach to the

classical calculus are more drastic: basically the sequents keep the same form as in the classical case, but the structural rules, weakening and contraction, are *eliminated* from the calculus and this seemingly local modification has far-reaching consequences on the structure of the whole calculus and on the very nature of the logical connectives. But let us go to the core of the matter. In presence of the structural rules of weakening:

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}(rW)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}(lW)$$

and contraction:

$$\frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}(rC)$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}(lC)$$

there are two equivalent formulations for the right-rule for conjunction, namely the *multiplicative* formulation:

$$\frac{\Gamma \vdash A, \Delta \quad \Lambda \vdash B, \Pi}{\Gamma, \Lambda \vdash A \wedge B, \Delta, \Pi}(r\wedge)$$

and the *additive* formulation :

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}(r \wedge *).$$

However, once weakening and contraction are dropped, $(r\wedge)$ and $(r\wedge *)$ are no longer equivalent; they actually correspond to two different connectives (the linear conjunctions), namely:

- $\otimes$ (read *times*)
- $\&$ (read *with*).

Linear logic is sensitive to how many times a formula is used as a hypothesis in a proof. The central idea is that the hypotheses used in a proof may be seen as *resources*, and generally only a limited amount of resources is available. Using C just once or using C a thousand times, in order to get A, does make a difference with respect to the price at which A is paid: in the second case A is obviously more expensive. (On the other hand, if there is a proof of A from the empty set of hypotheses, this means that A can be got for free and is always available). Using n times a formula C as a hypothesis in a linear proof of a formula A actually results in using n hypotheses of the form C; hence, a set of hypotheses Γ must be seen as a *multiset* of formulas rather than as a set of formulas.

Below, we describe the linear connectives by giving the corresponding inference rules and by explaining the meaning of the connectives implicitly defined by the rules.

2.1.1 Conjunctions

As suggested by the previous discussion, the right-rules for the two linear conjunctions are:

$$\frac{\Gamma \vdash A, \Delta \quad \Lambda \vdash B, \Pi}{\Gamma, \Lambda \vdash A \wedge B, \Delta, \Pi}(r\otimes)$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}(r\&).$$

A proof $\mathcal{D}$ of $A \otimes B$ from the multiset of hypotheses Π is the *juxtaposition* of a proof $\mathcal{D}_1$ of A and of a proof $\mathcal{D}_2$ of B; Π is the multiset of formulas which is the union of the multiset of hypotheses used in $\mathcal{D}_1$ and of the multiset of hypotheses used in $\mathcal{D}_2$. For example, if we have a proof of A which uses the hypothesis C exactly once and a proof of B which uses the hypothesis C exactly once:

$$\left.\begin{array}{c} C \\ \vdots \\ A \end{array}\right\} \mathcal{D}_1 \qquad\qquad \left.\begin{array}{c} C \\ \vdots \\ B \end{array}\right\} \mathcal{D}_2$$

we then have a proof of the sequent $C, C \vdash A \otimes B$ but we do not have a proof of the sequent $C \vdash A \otimes B$ because C is used twice in the couple $(\mathcal{D}_1, \mathcal{D}_2)$. On the other hand, in order to have a proof $\mathcal{D}$ of $A\&B$ from Π, we must have a proof $\mathcal{D}_1$ of A from Π and a proof $\mathcal{D}_2$ of B from Π; $\mathcal{D}$ 'proposes a choice' between $\mathcal{D}_1$ and $\mathcal{D}_2$. Thus, if we take $\mathcal{D}_1$ and $\mathcal{D}_2$ as in the above example, we have a proof of the sequent $C \vdash A\&B$ but we do not have a proof of the sequent $C, C \vdash A\&B$ because C is used *exactly* once in $\mathcal{D}_1$ and the same happens with $\mathcal{D}_2$. In other words, the sequent $C \vdash A\&B$ means 'using C once, you can get either A or B, as you wish'. Notice that $\&$ is not a disjunction: in order to prove $A\&B$, *both* a proof of A and a proof of B must be available and the following linear sequents are valid: $A\&B \vdash A$ and $A\&B \vdash B$.

2.1.2 Negation

Linear negation is noted as '$(.)^{\perp}$' (read *orthogonal*) and has the property that for any formula A, the formula $A^{\perp\perp}$ is provable exactly when A is provable. From this point of view, the linear negation has a rather classical flavour. We do not give inference rules for linear negation because such a connective is not primitive in the formal system that we use (see later on).

2.1.3 Disjunctions

Corresponding to the pair of linear conjunctions, there is a pair of linear disjunctions, namely:

$\uparrow$ (read *par*)[1]

$\oplus$ (read *plus*)

The right-rule for the connective $\uparrow$ is:

$$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \uparrow B, \Delta}(r\uparrow).$$

Since the sequent $\vdash A, A^{\perp}$ is a logical axiom of linear sequent calculus, the rule above tells us that, for any formula A, the sequent $\vdash A \uparrow A^{\perp}$ is provable. Thus the 'par' disjunction satisfies an 'excluded middle principle' and has a rather classical flavour. A proof $\mathcal{D}$ of $A \uparrow B$ from the multiset of hypotheses Π is such that:

1. if a proof of $A^{\perp}$ from Π' is provided, then a proof of B from Π, Π' is obtained;
2. if a proof of $B^{\perp}$ from Π'' is provided, then a proof of A from Π, Π'' is obtained.

There are two right-rules for the 'plus' disjunction:

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta}(r \oplus 1)$$

$$\frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}(r \oplus 2).$$

It is easy to see from the above rules that $\oplus$ behaves as an intuitionistic disjunction, in the sense that a proof $\mathcal{D}$ of $A \oplus B$ from the empty set of

[1] Our notation $\uparrow$ replaces Girard's notation, inverted &

hypotheses is either a proof of A or a proof of B. The two linear disjunctions are related to the conjunctions by the two De Morgan principles:

$$A \uparrow B \text{ is logically equivalent to } (A^{\perp} \otimes B^{\perp})^{\perp}$$

$$A \oplus B \text{ is logically equivalent to } (A^{\perp} \& B^{\perp})^{\perp}.$$

The connectives $\otimes$ and $\uparrow$ are called *multiplicative* connectives, while $\oplus$ and & are called *additive*.

2.1.4 Implication

Linear implication is a defined connective in the calculus which we use:

$$A \multimap B = A^{\perp} \uparrow B \text{ (read } A \textit{ linearly implies } B).$$

The formula $A \multimap B$ says that one can get B by using A exactly once.

2.1.5 Quantifiers

The inference rules for the linear quantifiers are the same as for classical logic and the classical equivalence between $\forall x A(x)$ and the negation of $\exists x \neg A(x)$ holds also in the case of linear logic. However, thanks to the absence of the contraction rule, the linear existential quantifier enjoys the same constructive property as the intuitionistic existential quantifier, namely: if $\exists x A(x)$ is provable then there is a term t such that $A(t)$ is provable.

2.1.6 Exponential modalities

With linear logic one wants to be able to make finer distinctions than in classical or intuitionistic logic (for example between $\otimes$ and &); however, one does not want to lose the expressive power of intuitionistic calculus. This is why the so called *exponential* modalities

! (read *of course*)

? (read *why not?*)

are introduced: by their use it is possible to translate intuitionistic logic into linear logic (See [GI 87]). Intuitively, $!A$ means that we can have as many occurrences of type A as we want; ? is the dual of !. However, in this paper we will work just with the fragment of linear predicate calculus which does not contain exponentials.

2.1.7 Linear sequents

The intended meaning of a linear sequent

$$G_1, ..., G_n \vdash D_1, ..., D_m$$

is

$$G_1 \otimes ... \otimes G_n \multimap D_1 \uparrow ... \uparrow D_m.$$

In the paper we use a particular version of linear logic where *weakening is indeed available*. Such a choice is forced by the fact that otherwise we would not get a theory powerful enough to get the soundness result for SLDNF, but does not lead to an unnatural system. We can look at things in the following manner:

1. the provability in classical sequent calculus of the sequent

   $$\Gamma, B_1, .., B_n \vdash A$$

   where each B_i is an occurrence of the formula B, gives us no information about the number of utilizations of B which is necessary in order to prove A from Γ;

2. the provability of the above sequent in standard linear logic (the logic in [GI 87]) tells us that exactly n occurrences of B are necessary in order to prove A from Γ;

3. the provability of the above sequent in linear logic + weakening tells us that *at most* n occurrences of B are necessary in order to prove A from Γ.

The proof-system which we use is *linear sequent calculus with weakening* which is obtained by adding the weakening rule to standard linear sequent calculus. Below, we describe such a system. Since any linear sequent

$$G_1, ..., G_n \vdash D_1, ..., D_m$$

has the same meaning as the right-handed sequent

$$\vdash G_1^{\perp}, ..., G_n^{\perp}, D_1, ..., D_m$$

one can write just right-handed sequents; this allows to formulate the calculus by writing just the right rules.

2.1.8 Linear well-formed formulas

Let $\mathcal{F}$ be a set of function symbols. Let

$$\mathcal{P}^{pos} = \{P_1, ..., P_i, ...\},$$
$$\mathcal{P}^{neg} = \{P_1^{\perp}, .., P_i^{\perp}, ...\}$$

be ordered sets of predicates. A linear literal over the language $(\mathcal{F}, \mathcal{P}^{pos} \cup \mathcal{P}^{neg})$ is either:

1. an expression of the form $P_i(t_1, .., t_n)$, or
2. an expression of the form $P_i^{\perp}(t_1, .., t_n)$

where t_1,..,t_n are terms over $\mathcal{F}$ and n is the arity of the predicate P_i (resp. $P_i^{\perp}$).

The linear well-formed formulas over the language $(\mathcal{F}, \mathcal{P}^{pos} \cup \mathcal{P}^{neg})$ are the formulas built (in the ordinary manner) out of the linear literals by using the following binary connectives:

- multiplicative connectives $\otimes$, $\uparrow$,
- additive connectives $\&$, $\oplus$,

and the linear quantifiers:

- universal quantifier $\forall$,
- existential quantifier $\exists$.

Linear negation is defined in the following manner. There are two distinct kinds of predicates in the syntax, that is, P_i (positive predicate) and $P_i^{\perp}$ (its corresponding negative predicate). Since the linear disjunctions and the linear conjunctions are related by 'De Morgan rules', then the formula $(A^{\perp} \otimes B^{\perp})^{\perp}$ is taken to be the same thing as $A \uparrow B$, the formula $(A^{\perp} \& B^{\perp})^{\perp}$ is taken to be the same as $A \oplus B$, and so on. Moreover, $(\forall x F)^{\perp} = \exists x F^{\perp}$ and $(\exists x F)^{\perp} = \forall x F^{\perp}$.

2.1.9 The system LLW (linear logic with weakening)

A (right-handed) linear sequent is an expression of the form $\vdash \Gamma$, where Γ is a finite sequence of linear formulas $G_1, .., G_n$; the implicitly defined meaning of the linear sequent $\vdash \Gamma$ is $G_1 \uparrow G_2 \uparrow ... \uparrow G_{n-1} \uparrow G_n$.

- Logical axioms

  $$\vdash A, A^{\perp}$$

- Cut rule

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}(CUT)$$

- Exchange rule

$$\frac{\vdash \Gamma}{\vdash \Gamma'}(EXCH),$$

where Γ' is a permutation of Γ.

- Weakening rule

$$\frac{\vdash \Gamma}{\vdash \Gamma, A}(WEAK),$$

where A is any well formed linear formula.

- Additive rules

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A\&B}(\&)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}(\oplus 1)$$

$$\frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}(\oplus 2)$$

- Multiplicative rules

$$\frac{\vdash \Gamma, A \quad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}(\otimes)$$

$$\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, A \uparrow B, \Delta}(\uparrow)$$

- Quantifiers rules

$$\frac{\vdash A, \Gamma}{\vdash \forall x A, \Gamma}(\forall)$$

if x is not free in Γ,

$$\frac{\vdash A[x/t], \Gamma}{\vdash \exists x A, \Gamma}(\exists).$$

2.1.10 Properties of LLW

Theorem 1 (Cut-elimination theorem for LLW) *Let $\vdash \Delta$ be a linear sequent and H be a set of linear sequents. If the sequent $\vdash \Delta$ is LLW-provable from the set of non-logical axioms H then there is a proof $\mathcal{D}$ of the sequent $\vdash \Delta$ from H such that if*

$$\frac{\vdash \Gamma, A \quad \vdash \Sigma, A^{\perp}}{\vdash \Gamma, \Sigma}$$

is a cut in $\mathcal{D}$, then either A or $A^{\perp}$ is a formula of a sequent in H.

Proof. By a straightforward modification of the standard proof of cut-elimination for Gentzen classical calculus. ∎

Theorem 2 (Subformula-property for LLW) *Let $\vdash \Delta$ be a linear sequent and H be a set of linear sequents. If the sequent $\vdash \Delta$ is LLW-provable from the set of non-logical axioms H then there is a proof $\mathcal{D}$ of $\vdash \Delta$ from H such that each formula F occurring in $\mathcal{D}$ is either*

1. *a subformula of a formula in Δ, or*
2. *an instantiation of a formula in a sequent of H, or*
3. *the linear negation of a subformula of a formula in a sequent of H.*

Proof. As for Gentzen's calculus, the subformula property is a corollary of the cut elimination theorem. Let $\mathcal{D}$ be the proof of $\vdash \Delta$ from H given by the cut-elimination theorem. A rather straightforward induction on the height h of $\mathcal{D}$ shows that $\mathcal{D}$ satisfies the required conditions. ∎

Lemma 1 (Valid linear inferences) *In the presence of the weakening rule:*

1. *The following sequents are provable:*

$$\vdash (A \otimes B) \multimap (A\&B)$$

$$\vdash (A \oplus B) \multimap (A \uparrow B).$$

2. *If $\vdash A \oplus A^{\perp}$ is a theorem, then also the following sequents are provable:*

$$\vdash (A\&B) \multimap (A \otimes B)$$

$$\vdash (A \uparrow B) \multimap (A \oplus B).$$

3. The following inference is correct:

$$\frac{\vdash A^{\perp} \quad \vdash A \oplus B, \Gamma}{\vdash B, \Gamma}.$$

4. The following inference is correct:

$$\frac{\vdash (A\&B) \multimap C \quad \vdash D \oplus \forall y(A \oplus \forall z E) \quad \vdash D \oplus \forall y(B \oplus \forall z E)}{\vdash D \oplus \forall y(C \oplus \forall z E)}$$

Proof. The proof of this Lemma is just an exercise in linear deduction; here, we give only the proof of part (4) which is the hardest one. Suppose we have:

(a) $\vdash D \oplus \forall y(A \oplus \forall z E)$

(b) $\vdash D \oplus \forall y(B \oplus \forall z E)$

(c) $\vdash (A\&B) \multimap C$.

By applying the $\otimes$-rule to (a) and (b) and by distributing $\otimes$ over $\oplus$ (see [GI 87]) we get:

(c) $\vdash (D \otimes D) \oplus (D \otimes \forall y(B \oplus \forall z E)) \oplus (\forall y(A \oplus \forall z E) \otimes D) \oplus (\forall y(A \oplus \forall z E) \otimes \forall y(B \oplus \forall z E))$.

Since in the presence of weakening the sequent $\vdash (F \otimes G) \multimap F$ is provable for any formulas F and G, from (d) we can deduce:

(e $\vdash D \oplus D \oplus D \oplus (\forall y(A \oplus \forall z E) \otimes \forall y(B \oplus \forall z E))$.

By pushing the universal quantifiers out of $\oplus$ (this can always be done) we get:

(f) $\vdash D \oplus \forall y((A \oplus \forall z E) \otimes (B \oplus \forall z E))$.

Now, in the presence of weakening, the formula

$$(A \oplus \forall z E) \otimes (B \oplus \forall z E)$$

linearly implies

$$(A\&B) \oplus \forall z E,$$

because

(i) by distributing $\otimes$ over $\oplus$ we get:

$$(A \otimes B) \oplus (A \otimes \forall z E) \oplus (\forall z E \otimes B) \oplus (\forall z E \otimes \forall z E);$$

(ii) since in the presence of weakening $(F \otimes G) \multimap F$ is provable, from the formula above we get:

$$(A \otimes B) \oplus \forall z E \oplus \forall z E \oplus \forall z E;$$

(iii) hence we get $(A \otimes B) \oplus \forall z E$;

(iv) finally, by applying (1), we get $(A\&B) \oplus \forall z E$.

Thus from (f) we can deduce:

(g) $\vdash D \oplus \forall y((A\&B) \oplus \forall z E)$.

Finally, (c) and (g) give us

(h) $\vdash D \oplus \forall y(C \oplus \forall z E)$.

■

2.2 Logic programs

In this section we introduce some concepts and notations that will be used in the rest of this paper. We also recall some results already existing in the literature on logic programming to which our own proofs will make reference.

Let $\mathcal{F}$ be a set of function symbols (which may contain 0-ary function symbols, that is, individual constants) and let $\mathcal{P}$ be a set of predicate symbols.

An *atom* over the language $\mathcal{L} = (\mathcal{F}, \mathcal{P})$ is an expression of the form

$$R(t_1, ..., t_n)$$

where $R \in \mathcal{P}$ and $t_1, ..., t_n$ are terms over $\mathcal{F}$.

A *program literal* over $\mathcal{L}$ is either an atom over $\mathcal{L}$ or an expression of the form $NOT(B)$ where B is an atom over $\mathcal{L}$.

A *program clause* over $\mathcal{L}$ is an expression of the form

$$A : -B_1 \; AND \; .... \; AND \; B_n \quad n \geq 0,$$

where A is an atom over $\mathcal{L}$ and each B_i is a program literal over $\mathcal{L}$. The atom A is called the *head* of the clause and the expression

$$B_1 \; AND \; .... \; AND \; B_n$$

is called the *body* of the clause; one says that a clause is *about* the predicate occurring in its head. When $n = 0$, that is, when the body is empty, the clause is called *unit clause*; we will write such a clause A rather than $A : -$.

A *logic program* P over L is a set of program clauses over L. The *language of the program* P is the language $\mathcal{L} = (\mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the set of functions occurring in P and $\mathcal{P}$ is the set of predicates occurring in P; we use the notation $\mathcal{L}ang(P)$ to denote such a language. (We may suppose that $\mathcal{L}ang(P)$ contains at least one constant: if this is not the case, we add a constant.)

A *query* Q over $\mathcal{L}$ is an expression of the form:

$$B_1 \ AND \ .... \ AND \ B_n \quad n \geq 0$$

where each B_i is a program literal over $\mathcal{L}$. When all the literals in Q are atoms, we say that Q is *positive*. In this paper, we will actually interpret the connectives $-:$, AND, NOT of logic programs as the connectives $\multimap$ (linear implication), & (additive linear conjunction) and $\perp$ (linear negation) of linear logic. Thus we will often abuse the notations and we identify a query

$$B_1 AND....AND B_n$$

with its transcription in terms of linear logic

$$B_1' \& .... \& B_n'$$

(where B_i' is obtained from B_i by replacing the operator NOT by the linear negation).

A *goal* G over $\mathcal{L}$ is an expression of the form

$$\leftarrow Q$$

where Q is a query.

Given a logic program P and a goal G, the notions of *SLDNF derivation*, *SLDNF successful derivation*, *SLDNF failed derivation*, *SLDNF tree* and *SLDNF finitely failed tree* for $P \cup \{G\}$ are defined as in [LL87]. We will say that a query Q *succeeds* on P if there is a SLDNF successful derivation for $P \cup \{\leftarrow Q\}$, and that Q *fails* on P if there is a SLDNF finitely failed tree for $P \cup \{\leftarrow Q\}$.

Also the notion of *computation rule* (which is also called selection rule) to be used in a SLDNF derivation is defined here as in [LL 87], however, we suppose that a computation rule used to derive a subgoal G' from a given goal G never selects a negative non-ground literal in G (that is, we take a computation rule to be always *safe* in the sense of [LL 87]).

A computation for Q w.r.t. P is an attempt to construct a SLDNF derivation for $P \cup \{\leftarrow Q\}$ which at a given point may be blocked by the fact that no (safe) computation rule can be activated. Given a program P and a query Q, a computation for Q w.r.t. P *flounders* if it ends with a

query whose literals are all negative and non-ground. A *query* Q *flounders* on P if there is at least a computation for Q w.r.t. P which flounders.

The possibility of floundering creates some rather undesirable phenomena, like, for instance, the fact that a query may not succeed even if some of its ground instantiations do, as in the example below.

Example 3 Let P_3 be a program whose only clauses are:

$$A(a)$$
$$B(x) : -NOT(C(x))$$

The query $B(a)$ succeeds but the query $B(x)$ neither succeeds nor fails: it flounders.

This kind of phenomenon contradicts the logical principle which says that the formula $\exists x B(x)$ is a logical consequence of the formula $B(a)$. However, one may impose conditions on programs and queries which prevent floundering. The following definitions give such conditions.

A query Q is said to be *allowed* if and only if any variable occurring in Q occurs in a positive literal. A program P is called *allowed* if and only if every variable occurring in a clause occurs in a positive literal of the body of the clause. The following result is known:

If P is an allowed program and Q is an allowed query then:

1. No computation for Q w.r.t. P flounders;

2. Every computed answer for Q w.r.t. P is a ground substitution for all the variables in Q ([SH 85], [LT 86]).

Thus allowedness of a program P and of a query Q is a sufficient (but not necessary) condition to prevent the floundering of Q.

It is well known that a query Q may fail under a given computation rule but loop under a different rule. However, there are rules which always lead to the failure of Q if there exists at least a way of making Q fail, for example the rule provided by the procedure SELECT defined below. The procedure SELECT is not recursive, because its definition involves the decidability of success of atomic goals. However, if we assume the existence of an oracle $\mathcal{O}$ which answers to the question 'Does the atom A succeed ?', SELECT is recursive in $\mathcal{O}$ and it may be described as follows.

Let $Q = B_1, ..., B_n$ be a query.

Case 1 The first literal B_1 of Q is positive. Then SELECT chooses B_1; if B_1 is unified via the substitution σ with the head of a clause

whose body is $A_1\ AND\ ....AND\ A_k$ then SELECT is recursively called on:

$$Q' = (B_2\ AND\ B_3\ AND\ ....\ AND\ B_n AND\ A_1\ AND\ ....\ AND\ A_k)\sigma.$$

Case 2 The first literal B_1 of Q falls in one of the following two sub-cases:

- it is negative and non-ground
- it is negative and ground but does not fail according to any computation rule (for no matter which rule R, either it succeeds or it loops).

Then SELECT is recursively called on

$$Q' = B_2\ AND\ B_3\ AND....AND\ B_n\ AND\ B_1.$$

Case 3 the first literal B_1 of Q is a negative ground literal $NOT(A)$ which fails according to some rule R. Then SELECT chooses B_1 and keeps applying R so to make $NOT(A)$ (and Q) eventually fail.

Moreover if the procedure SELECT, once called on the query $Q = B_1, ..., B_n$, executes n times the instruction of Case 2 and comes back to B_1, then the procedure goes on by simulating the computation rule which always chooses the left-most literal. (As a matter of fact, *any* rule would do, because we are in a situation where Q cannot fail.)

It is easy to see that the rule SELECT is such that if a query Q fails according to some computation rule, then Q fails also according to SELECT; that is, SELECT is 'maximal' with respect to failure.

Remark 1 If Q_τ is an instantiation of a query Q, the literal SELECT(Q_τ) may not be an instantiation of the literal SELECT(Q). For example, given the program whose clauses are:

```
A(a)
B(x) :- A(x)
```

and the query $Q = NOT\ A(x)\ AND\ B(x)$, SELECT(Q) is the literal $B(x)$ (which fails) while SELECT($Q[x/a]$) is $NOT(A(a))$ (which also fails). Notice that, for any query Q, if SELECT(Q_τ) is not an instantiation of SELECT(Q), then Q must begin with a negative unground literal L such that L_τ is ground and fails.

In our proof of soundness for SLDNF resolution we use a finite number of *projection functions* $\pi_1, ...\pi_m$, (where m is the maximum number of arguments of a function symbol in $\mathcal{L}ang(P)$). Intuitively, the meaning of the expression $\pi_i(f(t_1, ..., t_i, ...t_k))$ is t_i. We use such functions for technical reasons only.

3 Linear completion

In [CL 78] the 'completion of the program P' (notation : Comp(P)) is introduced and presented as the logical theory that one would 'naturally' associate to a logic program P. In what follows, we basically redefine Clark's completion by using linear connectives rather than classical ones, so as to get what we call the linear completion of a program P, denoted by **LComp(P)**. The next definition associates to a predicate R of of $\mathcal{L}ang(P)$ a set of sequents which play the same role as Clark's completed definition of R. Recall that a sequent $\vdash A, B$ means the same thing as $\vdash A \uparrow B$, that is the same as $\vdash A^\perp \multimap B$.

Definition 1 (Sequent definition of a predicate) Let R be a predicate symbol of $\mathcal{L}ang(P)$ and let $c_1, ..., c_k$ be all the clauses in P about R; clause c_i has the form:

$$R(t_1, ..., t_n) : -L_1, ..., L_m$$

Let $y_1, ..., y_p$ be the variables in c_i, $x_1, ..., x_n$ be new variables, and E_i be the formula

$$\exists y_1 ... \exists y_p (L_1\ \&...\ \& L_m\ \&\ x_1 = t_1\ \&...\&\ x_n = t_n).$$

The sequent definition of R is the following set of sequents:

1. $\vdash R^\perp(x_1, ..., x_n), E1 \oplus ... \oplus E_k$
2. $\vdash E_1^\perp, R(x_1, ..., x_n)$

   $\vdots$

   k. $\vdash E_k^\perp, R(x_1, ..., x_n)$

When there is no clause about R the sequent definition of R consists of the only sequent $\vdash R^\perp(x_1, ..., x_n)$.

We can now define the linear completion of the program P.

Definition 2 (Linear completion of a program) Let us add to $\mathcal{L}ang(P)$ m projection function symbols : $\pi_1, ...\pi_m$, where m is the maximum number of arguments of a function symbol in $\mathcal{L}ang(P)$. The non-logical axioms of the theory **LComp (P)** are obtained by taking the sequent definitions of each predicate R of $\mathcal{L}ang(P)$ and by adding the following axioms EQ for equality (we will write $\neq$ rather than $=^{\perp}$ for the inequality predicate) and for projection functions:

1. $\vdash x = x$
2. $\vdash x \neq y \oplus A[x]^{\perp}, A[x/y]$ for any formula $A[x]$
3. $\vdash x_1 \neq y_1 \oplus ... \oplus x_n \neq y_n, f(x_1, ..., x_n) = f(y_1, ..., y_n)$ for any function symbol f
4. $\vdash f(x_1, ..., x_n) \neq g(y_1, ..., y_m)$ for any distinct function symbols f, g other than a projection function π
5. $\vdash t[x] \neq x$ for any term $t[x]$ properly containing x and not containing a projection function π
6. $\vdash \pi(f(x_1, ..., x_n)) = x_i \; if \; i \leq n$ for any function symbol f other than a projection function π
7. $\vdash \pi(f(x_1, ..., x_n)) = f(x_1, ..., x_n) \; if \; i > n$ for any function symbol f.

Remark 2 We have the following facts

- The sequent $\vdash x = y \oplus x \neq y$ is provable.
- As a consequence of the projection axioms, any sequent of the form

  $$\vdash f(x_1, ..., x_n) \neq f(y_1, ..., y_n), x_1 = y_1 \; \& ... \; \& \; x_n = y_n$$

  (where f is any function symbol other than a projection π) is provable.
- The axioms EQ play the role of Clark's equality theory.

The essential results that we prove in this paper are as follows.

Theorem 3 (Completeness for SLDNF-resolution with respect to LComp(P) modulo allowedness) *Let P be an allowed logic program and let Q be an allowed query in the language of the program.*

1. *For any substitution τ of terms of $\mathcal{L}ang(P)$ for the variables in Q, if the sequent $\vdash Q_\tau$ is a theorem of* **LComp(P)** *then Q succeeds with the answer τ.*
2. *If the sequent $\vdash Q^{\perp}$ is a theorem of* **LComp(P)** *then Q fails.*

Theorem 4 (Soundness of SLDNF with respect to LComp(P)) *Let P be any logic program and let Q be any query in the language of the program. Let τ be a substitution of terms of $\mathcal{L}ang(P)$ for the variables in Q.*

1. *If Q SLDNF-succeeds with the answer τ on P then the sequent $\vdash Q_\tau$ is a theorem of* **LComp(P)**.
2. *If Q SLDNF-fails on P then the sequent $\vdash Q^{\perp}$ is a theorem of* **LComp(P)**.

4 Completeness of SLDNF and realizability

The method by which we are going to prove completeness (theorem 3) may be sketched as follows. Let P be any logic program.

1. First we define a property R(P) of sequents in such a way that if the sequent $\vdash G$ has the property R(P) then :
   (a) if G is an allowed query Q then Q succeeds on the program P;
   (b) if G is $Q^{\perp}$ and Q is an allowed query then Q fails on the program P.
2. Then we show that each axiom of **LComp(P)** has the property R(P) and that R(P) is stable under application of the rules of inference of **LComp(P)**.

The specific property R(P) which we use is called '*Realizability* w.r.t. P'. In order to properly define it, we first define a property '*realizable* w.r.t. P' (lower case 'r') on a given class of linear formulas, then we induce the property 'Realizable w.r.t. P' on sequents.

We start with an intuitive definition of realizability, which later on will be refined so as to handle some technical problems. Let us consider the language obtained by adding to $\mathcal{L}ang(P)$ the projection functions, the equality predicate and the inequality predicate. Let us note by $\mathcal{F}orm$ the set of additive linear formulas built up from literals of the extended language. We start by defining the realizability property for the subset of closed formulas in $\mathcal{F}orm$. (Since we suppose that $\mathcal{L}ang(P)$ contains at least one constant, such a subset is never empty.)

- $t_1 = t_2$ is realizable w.r.t. P iff t_1 and t_2 are equals;
- $t_1 \neq t_2$ is realizable w.r.t. P iff t_1 and t_2 are not equals;
- $R(t_1, ..., t_n)$ is realizable w.r.t. P iff $R(t_1, ..., t_n)$ succeeds on P;

- $R^{\perp}(t_1, ..., t_n)$ is realizable w.r.t. P iff $R(t_1, ..., t_n)$ fails on P;
- $F\&G$ is realizable w.r.t. P iff both F and G are realizable w.r.t. P;
- $F \oplus G$ is realizable w.r.t. P iff either F is realizable w.r.t. P or G is realizable w.r.t. P;
- $\forall x F$ is realizable w.r.t. P iff for any *ground* term t the formula $F[x/t]$ is realizable w.r.t. P;
- $\exists x F$ is realizable w.r.t. P if there is a *ground* term t such that the formula $F[x/t]$ is realizable w.r.t. P.

We extend the definition of realizability to open formulas in *Form* by saying that an open formula F is realizable w.r.t. P iff the universal closure of F is realizable w.r.t. P.

Recall that our goal is to prove, using the notion of realizability, that if a query Q is provable then Q succeeds and if its negation is provable then Q fails. Now, by definition, a literal $A(x)^{\perp}$ is realizable when its universal closure is realizable, hence when every ground instantiation $A(t)^{\perp}$ is so. Thus, we need to prove that if every ground instantiation $A(t)$ fails, then the open formula $A(x)$ fails. Now, if we limit ourselves to the ordinary terms of $\mathcal{L}ang(P)$, it is false that the failure of all ground instantiations of $A(x)$ entails the failure of $A(x)$, as the following example shows.

Example 4 Let P_4 be the following program:

1. $A(s(y)) : -A(y)$
2. $B(0)$

The query $A(x)$ loops, however, if t is a ground term of the language of the program, $A(t)$ fails.

Now consider the 'infinite ground term' $s^{\omega} = ssss...$ The query $A(s^{\omega})$ loops for the program above, because the 'term' s^{ω} 'simulates the behaviour' of the variable x. Thus, we would like to have infinite ground terms among our ground terms. But infinite terms cause problems; for example, the formula in the axiom 5 of the group EQ is not realizable if we allow s^{ω} as a term. However, we can 'describe' such an infinitary object by using infinitely many new constants $a_0...a_i....$ together with the following infinite set of equations : $a_0 = s(a_1), a_1 = s(a_2), a_2 = s(a_3)$ .... The new constants $a_0...a_i....$ which we use so to be able to 'speak' about infinite terms are not terms of **LComp(P)** : they are just a technical tool for doing proofs. Questions of equality and unifiability between terms which contain

the new constants may be answered by making reference to a correct and complete set of equations between ground terms which, in particular, will contain all the equations which are derivable by instantiating the equality axioms and the projection axioms of **LComp(P)**. Let us be more formal.

Definition 3 Let P be a logic program. By $\mathcal{EL}ang(P)$ we mean a language obtained by adding to $\mathcal{L}ang(P)$:

1. m projection function symbols: $\pi_1, ...\pi_m$, where m is the maximum number of arguments of a function symbol in $\mathcal{L}ang(P)$;
2. at most a countable infinity of new constants $a_0, \ldots, a_i$
3. the equality and inequality predicates.

Definition 4 (Reference system) Let T be the set of ground terms over $\mathcal{EL}ang(P)$. A set of equations and inequations E between terms in T is said to be sound and complete iff:

1. for every couple of terms t_1, t_2 in T, exactly one formula between $t_1 = t_2$ and $t_1 \neq t_2$ belongs to E;
2. E is closed under any derivation of equations and inequations which uses instantiations via terms in T of the equality axioms and the projection axioms EQ.

When E is sound and complete we say that E is a reference system over $\mathcal{EL}ang(P)$. In other words, a reference system is an equational theory over $\mathcal{EL}ang(P)$ which contains only closed formulas and which is a sound and complete extension of the equality theory EQ of **LComp(P)**.

Definition 5 (Equality modulo a reference system E) Let t_1, t_2 be terms of $\mathcal{EL}ang(P)$ such that if an expression $\pi_j(p)$ is a subterm, then p is a ground term of $\mathcal{EL}ang(P)$. Let E be a term system over $\mathcal{EL}ang(P)$. The terms t_1, t_2 are equal modulo E when one of the two following conditions is satisfied:

1. Both t_1 and t_2 are ground and the equation $t_1 = t_2$ belongs to E;
2. Neither t_1 nor t_2 is ground and :
   - if t_1 is a variable x, then also t_2 is x
   - if t_1 is $f(p_1, ..., p_m)$, then t_2 is $f(q_1, ..., q_m)$ and for every i, $1 \leq i \leq m$, then p_i is equal modulo E to q_i.

Definition 6 (Unification modulo a term system E) Let $P(t_1, ..., t_n)$ and $P(q_1, ..., q_n)$ be two atoms over $\mathcal{EL}ang(P)$ such that if an expression $\pi_j(p)$ appears as a subterm, then p is a ground term of $\mathcal{EL}ang(P)$. The atoms $P(t_1, ..., t_n)$ and $P(q_1, ..., q_n)$ are unifiable modulo E via the substitution τ (which replaces variables with terms of $\mathcal{EL}ang(P)$) if for every $i, 1 \leq i \leq n$, $(t_i)_\tau$ is equal modulo E to $(q_i)_\tau$.

Remark 3 Given any two atoms over $\mathcal{L}ang(P)$, if they are unifiable in the ordinary sense, then they are also unifiable modulo E, for any term system E over $\mathcal{EL}ang(P)$.

Definition 7 (Resolution w.r.t. a term system E) Let P be a logic program and let Q be a query over $\mathcal{EL}ang(P)$ such that if an expression $\pi_j(p)$ appears as a subterm, then p is a ground term of $\mathcal{L}ang(P)$. The notions of success modulo E, failure modulo E and loop modulo E for $P \cup \{\leftarrow Q\}$ are obtained by replacing the notion of unification with the unification modulo E in SLDNF resolution.

Example 5 Let us consider again the program P_4 of Example 4. Let $\mathcal{EL}ang(P)$ be such that the set of new constants is $\{a_i : i \in N\}$, so that the corresponding set of ground terms is:

$$T = \{s^i(c) : i \in N, c \in \{0, a_0 ... a_i ...\}\}.$$

Let E be a reference system which contains all the equations: $a_i = s(a_{i+1})$ where $i \in N$ and all the inequations: $a_i \neq t$ where t is a ground term in $\mathcal{L}ang(P)$. The term a_0 is unifiable modulo E with $s(y)$ via the substitution $\tau(y) = a_1$. Also a_1 is unifiable with $s(y)$, and so on. The query $A(a_0)$ loops modulo E.

Definition 8 (Realizability of additive formulas in $\mathcal{EL}ang(P)$ w.r.t. a program P and a reference system E) Let P be a logic program, let $\mathcal{EF}orm$ be the set of additive linear formulas built up from literals of $\mathcal{EL}ang(P)$ and let E be a reference system over $\mathcal{EL}ang(P)$. The definition realizability for formulas in $\mathcal{EF}$orm with respect to P and E is obtained by taking the 'intuitive definition' of realizability given above and by replacing:

'$\mathcal{F}orm$' by '$\mathcal{EF}orm$',

'equal' by 'equal modulo E',

'succeeds' by 'succeeds modulo E',

'fails' by 'fails modulo E'.

We can now define the notion of *realizability* for sequents. The intuitive idea is that the atomic sequent $\vdash A^\perp, B$ is realizable if the success of A implies the success of B and the failure of B implies the failure of A. The formal definition is a bit more complicated because:

1. the formulas occurring in the sequent may contain variables, connectives and quantifiers;
2. the sequent may contain more than two formulas because of weakening.

Definition 9 (Realizability for sequents in in $\mathcal{EL}ang(P)$ w.r.t. a program P and a term system E) Let P be a logic program, let E be a reference system over $\mathcal{EL}ang(P)$, and let let $F_1, ..., F_n$ be elements of $\mathcal{EF}orm$ (the set of additive linear formulas built up from literals of $\mathcal{EL}ang(P)$). The sequent $\vdash F_1, ..., F_n$ is realizable with respect to P and E iff for any substitution τ of ground terms of $\mathcal{EL}ang(P)$ for the free variables $x_1, ..., x_q$ of $F_1, ..., F_n$ and for any i between 1 and n we have:

if $(F_1^\perp)_\tau, ..., (F_{i-1}^\perp)_\tau, (F_{i+1}^\perp)_\tau, ..., (F_n^\perp)_\tau$ are all realizable w.r.t. P and E **then** also $(F_i)_\tau$ is realizable w.r.t. P and E.

(In the particular case where $n = 1$, $\vdash F$ is Realizable iff F is realizable.)

Lemma 2 *Let P be an allowed program, let A be an atom of $\mathcal{EL}ang(P)$ and let E be a reference system over $\mathcal{EL}ang(P)$. If A succeeds modulo E on P, then for any ground term t occurring in A there is a ground term t' of $\mathcal{L}ang(P)$ such that the equation $t = t'$ belongs to E.*

Proof. This is straightforward, because thanks to the fact that P is allowed, any variable which appears in the head of a clause must appear also in a positive literal of its body and any term which occurs in a unitary clause must be a ground term of $\mathcal{L}ang(P)$. ∎

Lemma 3 *Let P be a logic program and let E be any reference system over $\mathcal{L}ang(P)$. The axioms of* **LComp(P)** *are realizable with respect to P and E.*

Proof.

(a) Logical axioms. This case is trivial.

(b) Axioms of the form $\vdash R^\perp(x_1, ..., x_n)$. In this case P does not contain any clause about R. Thus, given any substitution τ of ground terms

of $\mathcal{EL}ang(P)$ to the variables of $R^{\perp}(x1,...,x_n)$, $R(x_1,...,x_n)_{\tau}$ fails. Thus $R^{\perp}(x_1,...,x_n)_{\tau}$ is always realizable and the universal closure of $R^{\perp}(x_1,...,x_n)$ is realizable.

(c) Axioms of the form

$$\vdash (\exists y_1...\exists y_p(L_1 \ \&...\& \ L_m \ \& \ x_1 = t_1 \ \&...\& \ x_n = t_n))^{\perp}, R(x_1,...,x_n)$$

with $x_1,...,x_n$ being the only free variables.

Let τ be a substitution of ground terms of $\mathcal{EL}ang(P)$ to $x_1,...,x_n$, such that $\tau(x_i) = q_i$. We must prove the following.

(i) if the formula

$$\exists y_1...\exists y_p(L_1 \ \&...\& \ L_m \ \& \ q_1 = t_1 \ \&...\& \ q_n = t_n)$$

is realizable w.r.t. P and E, then $R(q_1,...,q_n)$ is realizable w.r.t. P and E.

(ii) if $R(q_1,...,q_n)^{\perp}$ is realizable w.r.t. P and E, then also the formula

$$(\exists y_1...\exists y_p(L_1 \ \&...\& \ L_m \ \& \ q1 = t_1 \ \&...\& \ q_n = t_n))^{\perp}$$

is realizable w.r.t. P and E.

Proof of (i) Suppose that the formula

$$\exists y_1...\exists y_p(L_1 \ \&...\& \ L_m \ \& \ q_1 = t_1 \ \&...\& \ q_n = t_n)$$

is realizable w.r.t. P and E. Then there are ground terms r_1,..., r_p of $\mathcal{EL}ang(P)$ such that the following formula is realized:

$$(L_1 \ \&...\& \ L_m \ \& \ q_1 = t_1 \ \&...\& \ q_n = t_n)[y_1/r_1,...,y_p/r_p]$$

Thus:

1. $(L_1 \ \&...\& \ L_m)[y_1/r_1,...,y_p/r_p]$

is realized, hence the ground query

$$(L_1 \ \&...\& \ L_m)[y_1/r_1,...,y_p/r_p]$$

succeeds w.r.t. P and E (by definition of realizability),

2. for each i, $1 \leq i \leq n$, q_i and $t_i[y_1/r_1,...,y_p/r_p]$ are the same term (modulo E).

Thus q_i is unifiable with t_i via the unifier ρ where $\rho(y_i) = r_i$. Now, since

$$R(t_1,...,t_n) : -L_1 \ \&...\& \ L_m$$

is a clause of P, we get that $R(q_1,...,q_n)$ succeeds modulo E if the query $(L_1 \ \&...\& \ L_m)[y_1/r_1,...,y_p/r_p]$ succeeds modulo E. But this is the case by (1). Therefore the formula $R(q_1,...,q_n)$ is realizable w.r.t. P and E. This ends the proof of (i).

Proof of (ii). Suppose that $R^{\perp}(q_1,...,q_n)$ is realizable w.r.t. P and E. To prove that the formula

$$(\exists y_1...\exists y_p(L_1 \ \&...\& \ L_m \& \ q_1 = t_1 \ \&...\& \ q_n = t_n))^{\perp}$$

is realizable w.r.t. P and E we must prove that the formula

3. $\forall y_1...\forall y_p(L_1^{\perp} \ \oplus ... \oplus \ L_m^{\perp} \ \oplus \ q_1 \neq t_1 \ \oplus ... \oplus \ q_n \neq t_n)$

is realizable.

This means that given any substitution ρ of ground terms $r_1,...,r_p$ of $\mathcal{EL}ang(P)$ to the variables $y_1,...,y_p$, the formula

4. $(L_1^{\perp} \ \oplus ... \oplus \ L_m^{\perp} \ \oplus \ q_1 \neq t_1 \ \oplus ... \oplus \ q_n \neq t_n)_{\rho}$

must be realizable.

Now, let $\rho(yi) = r_i$. Since the ground formula $R^{\perp}(q_1,...,q_n)$ is realizable, $R(q_1,...,q_n)$ must fail w.r.t. P and E. Assume that there exists an i, $1 \leq i \leq n$ such that q_i is not the same term as $t_i[y_1/r_1,...,y_p/r_p]$ (modulo E). Then the formula $q_i \neq t_i[y_1/r_1,...,y_p/r_p]$ is realizable w.r.t. P and E, which implies that the formula (4) is realizable.

Assume now that the equalities

$$\begin{aligned} q_1 &= t_1[y_1/r_1,...,y_p/r_p] \\ &\vdots \\ q_n &= t_n[y_1/r_1,...,y_p/r_p] \end{aligned}$$

all hold (in E). Since $R(t_1,...,t_n) : -L_1 \ \&...\& \ L_m$ is a clause of P and $R(q_1,...,q_n)$ fails w.r.t. P and E, it follows that for any unifier σ of $q_1,...,q_n$ and $t_1,...,t_n$, the query $(L_1 \ \&...\& \ L_m)_{\sigma}$ fails. But since for each i, we have $q_i = t_i[y_1/r_1,...,y_p/r_p]$ and q_i is ground, ρ is an unifier of $q_1,...,q_n$ and $t_1,...,t_n$. Thus the ground query $(L_1 \ \&...\& \ L_m)_{\rho}$ fails modulo E, therefore by definition the formula

$$(L_1^{\perp} \ \oplus ... \oplus \ L_m^{\perp})_{\rho}$$

is realizable w.r.t. P and E, which implies that the formula (4) is realizable.

This completes the proof of **(c)**.

(d) Axioms of the form

$$\vdash R^{\perp}(x_1, ..., x_n), E_1 \oplus ... \oplus E_k$$

where each E_i has the form $(\exists y_1 ... \exists y_p (L_1 \,\&...\&\, L_m \,\& x_1 = t_1 \,\&...\&\, x_n = t_n))$. Proof is similar to the case **(c)**.

(e) Equality and projection axioms (EQ). A reference system E has been defined so to always 'respect' ground instantiations of such axioms. ∎

Lemma 4 *Let P be a logic program and let E be any reference system over $\mathcal{EL}ang(P)$. The exchange rule, the weakening rule, the additive rules, the cut rule and the quantifiers rules of the calculus LLW preserve the realizability property w.r.t. P and E.*

Proof of the above lemma is straightforward. It is worthwhile emphasizing that the contraction rule does **not** preserve realizability. For a trivial example, take the program whose only clause is $A(a) : -NOT(A(a))$. The sequent $\vdash A(a), A(a)$ is realizable (for any system of terms); however, its contraction $\vdash A(a)$ is not so. Indeed, *it is the absence of contraction that makes the difference between the behaviour of LComp(P) and Clark's completion.*

Lemma 5 *Let P be an allowed program, let Q be an allowed query over $\mathcal{L}ang(P)$ and let τ be a substitution of ground terms of $\mathcal{L}ang(P)$ to the variables in Q. If the query Q_τ succeeds, then the query Q succeeds with the answer τ.*

Notice here that by 'success' we mean success in the ordinary sense. The proof of the above lemma is an easy induction on the length of the successful SLDNF derivation of Q_τ. Of course the hypothesis of allowedness plays an essential role; in fact, the example 3 shows that the lemma is false in the general case.

Proposition 1 *Let P be an allowed logic program, let Q be an allowed query over $\mathcal{L}ang(P)$, let τ be a substitution of ground terms of $\mathcal{L}ang(P)$ to the variables in Q. If the sequent $\vdash Q_\tau$ is provable in* **LComp(P)**, *then Q succeeds on P (in the ordinary sense) with the answer τ.*

Proof. Suppose that $\vdash Q_\tau$ is a theorem of **LComp(P)**; by the subformula property for LLW there is a proof $\mathcal{D}$ of $\vdash Q_\tau$ from the axioms of **LComp(P)** such that any formula occurring in $\mathcal{D}$ is either a subformula of Q_τ, or an instantiation of a formula in a sequent of **LComp(P)** or the linear negation of a subformula of a formula in a sequent of **LComp(P)**. As a consequence, the only inference rules used in $\mathcal{D}$ are exchange, weakening, cut, additive rules, quantifier rules. Now, let $\mathcal{EL}ang(P)$ be such that only a new constant c has been added to the language of the program and consider the reference system over $\mathcal{EL}ang(P)$ where the only equation concerning c is the trivial equation $c = c$. By Lemma 3 the sequents in **LComp(P)** are realizable w.r.t. P and E^{min}. By Lemma 4 exchange, weakening, cut, additive rules and quantifier rules preserve realizability w.r.t. P and E^{min}. It follows that $\vdash Q_\tau$ is realizable w.r.t. P and E^{min} so that the formula Q_τ is realizable w.r.t. P and E^{min}. We can prove that Q_τ is necessarily ground by an easy *reductio ad absurdum* argument. Suppose that it is not so; from its realizability w.r.t. E^{min} it follows that also $(Q_\tau)_\sigma$ is realizable w.r.t. E^{min}, where σ is a substitution of the new constant c to the variables in Q_τ. Since Q is allowed, also the query Q_τ is allowed. Thus any variable occurring in Q_τ must occur in a conjunct of Q_τ which is positive, say L_τ. By definition of realizability we will have that $(L_\tau)_\sigma$ is a positive ground literal which succeeds modulo E^{min}. By Lemma 2 this implies that the new constant c of $(L_\tau)_\sigma$ is equal modulo E^{min} to a ground term in the language of the program, but this is impossible. Therefore Q_τ is necessarily ground. Since Q_τ is realizable w.r.t. E^{min} and ground, by the very definition of realizability Q_τ succeeds modulo E^{min}. But this just means that Q_τ succeeds in the ordinary sense. Thus we may apply Lemma 5 to conclude that Q succeeds with the answer τ . ∎

Proposition 2 *Let P be an allowed logic program, let Q be an allowed query over $\mathcal{L}ang(P)$. If $\vdash Q^{\perp}$ is a theorem of* **LComp(P)** *then Q fails.*

Proof. Let T be a SLDNF-tree for $P \cup \{\leftarrow Q\}$ built up according to the computation rule SELECT (see section 1). Each branch of T is a SLDNF-derivation for $P \cup \{\leftarrow Q\}$ (no floundering occurs). Let β be a branch of T. Below, we show that β is necessarily a failure branch. First, observe that if β where a successful branch, then Q should succeed with a closed answer τ. Thus Q_τ should be realizable w.r.t. P and any reference system E. Now, by Lemmas 3 and 4, $Q^{\perp}$ is realizable w.r.t. P and E, thus $Q^{\perp}_\tau$ is indeed realizable w.r.t. P and E. Therefore we can rule out this case.

We still need to rule out the possibility that β involves a loop. Now, let $\mathcal{EL}ang(P)$ be such that infinitely many new constants $a_0, ..., a_n$ have been added and let E* be a reference system over $\mathcal{EL}ang(P)$ which contains any

equation of the form $a_i = t[a_j, ..., a_{j+n}]$, whenever in β a variable x_i is affected to the term $t[x_j, ..., x_{j+n}]$, and any inequation of the form $a_i \neq t$, for any ground term t of $\mathcal{L}ang(P)$, whenever in β a variable x_i is affected to nothing else but a variable. Let $x_1, ..., x_n$ be the variables in Q. By Lemmas 3 and 4 $Q^{\perp}$ is realizable w.r.t. P and E*. Thus also $Q^{\perp}[x_1/a_1..., x_n/a_n]$ is realizable w.r.t. P and E*. Hence the query $Q[x_1/a_1..., x_n/a_n]$ must fail modulo E*. Now, let β' be an 'object' obtained from β by replacing each variable x_i in β by the new constant a_i. β' is a SLDNF derivation for $P \cup \{\leftarrow Q[x_1/a_1..., x_n/a_n]\}$, where unification is done modulo E*. Since the derivation β' 'follows' β, to show that β does not loop is equivalent to show that β' does not loop. Unfortunately, we cannot immediately deduce that β' is a finitely failed derivation from the fact that $Q[x_1/a_1..., x_n/a_n]$ must fail modulo E*, for the following reason. The rule underlying β is SELECT and is indeed maximal with respect to the failure of Q. However, it might happen that the rule $R^{\beta'}$ underlying the derivation β' of $P \cup \{\leftarrow Q[x_1/a_1..., x_n/a_n]\}$, which 'imitates' the behaviour of SELECT in β, treats subgoals differently from what SELECT would once called on $Q[x_1/a_1..., x_n/a_n]$. (See Remark 1 in section 2). We cannot immediately exclude the possibility that $R^{\beta'}$ is not maximal and 'misses' the failure of $Q[x_1/a_1..., x_n/a_n]$.

Nevertheless, we can indeed prove that β' is necessarily a failed derivation by making the following *reductio ad absurdum* argument. Suppose that β' loops; since the derivation γ built up by using the maximal computation rule SELECT is necessarily finitely failed, there is a subquery in the evaluation of $Q[x_1/a_1..., x_n/a_n]$ such that the rule underlying β' 'behaves differently' from our maximal rule. Let Q' be the first of such subqueries. By the remark of section 1, Q' must have the form

$$NOT(A[x_i/a_i, ..., x_n/a_{i+m}])\ AND\ B_1\ AND ... AND\ B_n$$

where $NOT(A[xi/a_i, ..., x_n/a_{i+m}])$ is ground and fails (modulo E*) according to some computation rule R'. (The rule underlying β' 'imitates' the behaviour of SELECT on Q and 'misses' the fact that $NOT(A[x_i/a_i, ..., x_n/a_{i+m}])$ is ground). Thus we get the following fact:

1. the atom $A[x_i/a_i, ..., x_n/a_{i+m}]$ is ground and succeeds modulo E* with respect to some computation rule.

Now suppose that each new constant a_j in $A[x_i/a_i, ..., x_n/a_{i+m}]$ is equal modulo E* to a ground term t_j of the language of the program. Then the literal

$$NOT(A[x_i/a_i, ..., x_n/a_{i+m}])$$

eventually appears as a component of a subgoal in β and is necessarily evaluated to failure in β. But in that case β is a finite failed derivation and so is β', in contradiction with our hypothesis that β' loops. Thus we get also a second fact:

2. The atom $A[x_i/a_i, ..., x_n/a_{i+m}]$ contains at least one new constant c such that for no ground term t of the language of the program the equation $c = t$ belongs to E*.

The above two facts together with Lemma 2 give a contradiction. Thus β and β' are necessarily finitely failed. ∎

Our completeness result (Theorem 3) follows from Propositions 1 and 2.

5 Soundness of SLDNF-resolution

The proof of theorem 4 is a modification of the proof of soundness for Clark's completion. Some care is indeed needed to make sure that the logical inferences actually used in the proof are linearly correct and the presence of weakening appears necessary to make **LComp(P)** strong enough to get the result. We give in detail only the proofs of those lemmas where the specificity of linear logic plays an essential role.

The next lemma is essential to obtain the soundness result. Its precise formulation is quite complex, but the intuitive idea is rather easy. Roughly, the lemma says that if $Q = A\ AND\ B$ is a query for a program P whose only clauses about A are $A : -C$ and $A : -E$ and if the sequents $\vdash (A\ \&\ C)^{\perp}, \vdash (A\ \&\ E)^{\perp}$ are theorems of **LComp(P)**, then also the sequent which expresses the failure of Q, namely $\vdash (A\ \&\ B)^{\perp}$, is provable. The proof involves some difficulties due to the specific behaviour of linear quantifiers. For example, we cannot deduce $\exists x(A(x)\ \&\ B(a))$ from $\exists x A(x)\ \&\ B(a))$; however, we do need to make this kind of inference. We have chosen to overcome this difficulty by using the projection functions to 'eliminate' existential quantifiers as much as possible. For example, the formula

$$\exists w \exists u \{f(x, y) = f(g(w), f(w, u))\ \&\ R1(w, u)\}$$

can be rewritten as

$$f(x, y) = f[g(f(x, y)_{11}), f(f(x, y)_{11}, f(x, y)_{22})] \& R1(f(x, y)_{11}, f(x, y)_{22})$$

where t_n is an abbreviation for $\pi_n(t)$. We also make an essential use of the fact that the sequent $\vdash x = y \oplus x \neq y$ is a theorem of **LComp(P)**. In fact,

this allows to make some exchanges between the 'par' disjunction and the 'plus' disjunction. (See Lemma 1). We start with a definition.

Definition 10 Let $A_1, ..., A_n$ be linear literals. The set of formulas

$$Fail\{A_1, ..., A_n\}$$

is defined as follows: $F \in Fail\{A_1, ..., A_n\}$ iff F is a closed formula obtained:

- using $A_i^\perp$ exactly once, for $1 \leq i \leq n$ and no other literal,
- using $\oplus$ exactly $n-1$ times,
- using $\forall x_i$ exactly once, for $1 \leq i \leq p$, where $x_1, ..., x_p$ are all the variables in F.

Remark 4 The universal closure of $A_1^\perp \oplus ... \oplus A_n^\perp$ is an element of $Fail\{A_1, ..., A_n\}$. For example, $Fail\{R(x,y), S(x,z)\}$ has 4 elements (modulo trivial exchanges of order):

1. $\forall x(\forall y R^\perp(x,y) \oplus \forall z S^\perp(x,z))$
2. $\forall x \forall y(R^\perp(x,y) \oplus \forall z S^\perp(x,z))$
3. $\forall x \forall z(\forall y R^\perp(x,y) \oplus S^\perp(x,z))$
4. $\forall x \forall z \forall y(R^\perp(x,y) \oplus S^\perp(x,z))$.

Lemma 6 (Provability of failure) *Let P be a program, let $Q = A_1 \& \ldots \& A_n$ be a query over $\mathcal{L}ang(P)$, and let the atom $A_i = R(s_1, ..., s_m)$ be the selected literal. Let $\vec{t_i}$ be a shorthand for a sequence $t_{i_1}, ..., t_{i_m}$ of terms. If the following two conditions are satisfied:*

1. *$R(s_1, ..., s_m)$ unifies exactly with the heads of the clauses :*
   $R(t_1) : -G_1$
   $\vdots$
   $R(t_k) : -G_k$
   via the most general unifiers $\sigma_1, ..., \sigma_k$;
2. *for any j between 1 and k and for any formula F which belongs to the set*

$$Fail\{(A_1)_{\sigma_j}, ..., (Ai-1)_{\sigma_j}, (G_j)_{\sigma_j}, (A_{i+1})_{\sigma_j}, ...(A_n)_{\sigma_j}\},$$

   the sequent $\vdash F$ is a theorem of **LComp(P)**;

then for any formula H which belongs to $Fail\{A_1, ..., A_n\}$, the sequent $\vdash H$ is a theorem of **LComp(P)**. *In particular, the universal closure of $(A_1 \& ... \& A_n)^\perp$ is provable.*

Proof. For the sake of clarity, we develop our argument on a concrete example which has been carefully chosen so to embody all the essential difficulties. Thus, let Q be $A(x)$ & $B(f(x,y)$ & $C(x,y,z)$ and let $B(f(x,y))$ be the selected atom. Let the following clauses be all the clauses of P about the predicate B:

$$B(f(g(w), f(w,u))) : -R1(w,u,v)$$
$$B(f(w',w')) : -R2(w',u')$$
$$B(g(w'')) : -R2(w'',u').$$

Thus $\sigma_1 = \{x/g(w), y/f(w,u)\}, \sigma_2 = \{x/w', y/w'\}$, while $f(x,y)$ and $g(w'')$ are not unifiable.
By hypothesis we have:

1. if $G \in Fail\{A(g(w)), R_1(w,u,v), C(g(w), f(w,u), z)\}$
   or $G \in Fail\{A(w'), R_2(w',u'), C(w',w',z)\}$,
   then the sequent $\vdash G$ is a theorem of **LComp(P)**.

We want to prove that if $F \in Fail\{A(x), B(f(x,y), C(x,y,z)\}$ then $\vdash F$ is a theorem of **LComp(P)**; for this it suffices to prove:

$$\vdash A(x)^\perp \oplus \forall y(B^\perp(f(x,y)) \oplus \forall w(C(x,y,z))$$

because universal quantifiers can always be 'pushed out' of $\oplus$. By (1) we have the provability of the following sequent:

2. $\vdash \forall w(A(g(w))^\perp \oplus \forall u(\forall v R_1^\perp(w,u,v) \oplus \forall z C^\perp(g(w), f(w,u), z))$.

Hence also:

3. $\vdash (A(g(\pi_1(x)))^\perp \oplus \forall u(\forall v R1^\perp(p1(x), u, v) \oplus \forall z C^\perp(g(\pi_1(x)),$
   $f(\pi_1(x), u), z))$.

By the equality axiom 2 we have that the following sequent is provable:

4. $\vdash (A(g(\pi_1(x)))^\perp \multimap x \neq (g(\pi_1(x)) \oplus A^\perp(x)$

Thus from 3. and 4. we can deduce:

5. $\vdash x \neq (g(\pi_1(x)) \oplus A^\perp(x) \oplus \forall u(\forall v R_1^\perp(\pi_1(x), u, v)$
   $\oplus \forall z C^\perp(x, f(\pi_1(x), u), z))$.

Now we can push the universal quantifiers outwards to get:

6. $\vdash A^\perp(x) \oplus \forall u\{\forall v(x \neq (g(\pi_1(x)) \oplus (R_1^\perp(\pi_1(x), u, v))$
   $\oplus \forall z C^\perp(x, f(\pi_1(x), u), z)\}$.

From 6. by the equality axiom 2 and Lemma 1(1):

7. $\vdash A^{\perp}(x) \oplus \forall u \forall y\{y = f(\pi_1(x), u)$
$\multimap \{\forall v(x \neq (g(\pi_1(x)) \oplus (R_1^{\perp}(\pi_1(x), u, v)) \oplus \forall z C^{\perp}(x, y, z)\}\}$.

Hence by Lemma 1(2), since $\vdash x = y \oplus x \neq y$ is a theorem:

8. $\vdash A^{\perp}(x) \oplus \forall y\{y \neq f(\pi_1(x), p2(y)) \oplus$
$\{\forall v(x \neq (g(\pi_1(x)) \oplus (R_1^{\perp}(\pi_1(x), p2(y), v)) \oplus \forall z C^{\perp}(x, y, z)\}\}$.

Finally moving outwards the universal quantifiers:

9. $\vdash A^{\perp}(x) \oplus \forall y\{\forall v[y \neq f(\pi_1(x), p2(y)) \oplus x \neq (g(\pi_1(x))$
$\oplus (R_1^{\perp}(\pi_1(x), \pi_2(y), v)] \oplus \forall z C^{\perp}(x, y, z)\}$.

By a similar reasoning we get from the second part of (1):

10. $\vdash A^{\perp}(x) \oplus \forall y\{\forall u'[x \neq y \oplus R2^{\perp}(x, u')] \oplus \forall z C^{\perp}(x, y, z)\}$.

Now, by construction of **LComp(P)**, we have the theorem:

11. $\vdash B^{\perp}(f(x, y)), \exists w \exists u \exists v\{f(x, y) = f(g(w), f(w, u)) \ \& \ R_1(w, u, v)\}$
$\oplus \exists w' \exists u'\{f(x, y) = f(w', w') \ \& \ R_2(w', u')\}$
$\oplus \exists w'' \exists u''\{f(x, y) = g(w'') \ \& \ R_2(w'', u'')\}$.

Let us abbreviate the expression $\pi_i(\pi_j(t))$ as t_{ij}. By the projection axioms we can prove:
$\vdash f(x, y) = f(g(w), f(w, u)) \multimap w = f(x, y)_{11} \ \& \ u = f(x, y)_{22}$
$\vdash f(x, y) = f(w', w') \multimap w' = f(x, y)_1$.
Thus from (11) by using the equality axiom 2 we can get:

12. $\vdash B^{\perp}(f(x, y)), \exists v\{f(x, y) = f[g(f(x, y)_{11}), f(f(x, y)_{11}, f(x, y)_{22})]$
$\& \ R_1(f(x, y)_{11}, f(x, y)_{22}, v)\} \oplus \exists u'\{f(x, y) = f(f(x, y)_1, f(x, y)_1)$
$\& \ R_2(f(x, y)_1, u')\} \oplus \exists w'' \exists u''\{f(x, y) = g(w'') \ \& \ R_2(w'', u'')\}$

Using the equality axioms we can prove (as in the classical case), the sequent

$$\vdash f(x, y) \neq g(w'').$$

Thus by the inference of Lemma 1 (3) we can deduce from 12.:

13. $\vdash B^{\perp}(f(x, y)), \exists v\{f(x, y) = f[g(f(x, y)_{11}), f(f(x, y)_{11}, f(x, y)_{22})]$
$\& \ R_1(f(x, y)_{11}, f(x, y)_{22}, v)\} \oplus \exists u'\{f(x, y) = f(f(x, y)_1, f(x, y)_1)$
$\& \ R_2(f(x, y)_1, u')\}$.

Hence by using again the projection axioms:

14. $\vdash B^{\perp}(f(x, y)), \exists v\{x = g(\pi_1(x))$
$\& y = f(\pi_1(x), \pi_2(y)) \& R_1(\pi_1(x), \pi_2(y), v)\} \oplus \exists u'\{y = x \ \& \ R_2(x, u')\}$.

Now if one takes:
$A = [\exists v\{x = g(\pi_1(x)) \ \& \ y = (\pi_1(x), \pi_2(y)) \ \& \ R_1(\pi_1(x), \pi_2(y), v)\}]^{\perp}$;
$B = [\exists u'\{y = x \& R_2(x, u')\}]^{\perp}$;
$C = B^{\perp}(f(x, y))$;
$D = A^{\perp}(x)$;
$E = \forall z C^{\perp}(x, y, z)$
by applying Lemma 1 (4) to 9., 10. and 14. one gets:
$\vdash A(x)^{\perp} \oplus \forall y(B^{\perp}(f(x, y)) \oplus \forall w(C(x, y, z))$. ∎

The next lemma is much easier to establish.

Lemma 7 (Provability of success) *Let P be a program, and let Q be a query A_1 &...& A_n. Let the selected literal A_i be an atom $P(s_1, ..., s_n)$ which unifies with the head of a clause:*

$$P(t_1, ..., t_n) : -B$$

via the m.g.unifier θ. Given any substitution γ subsumed by θ, if the sequent

$$\vdash (A_1 \ \& ... \& \ A_{i-1} \ \& \ B \ \& \ A_{i+1} \ \& \ A_m)_{\gamma}$$

is a theorem of **LComp(P)**, *then the sequent* $\vdash Q_{\gamma}$ *is a theorem of* **LComp(P)**.

Proof. For the sake of clarity, we consider the case where the predicate P is unary and Q has just two conjuncts A_1 and A_2. Let $A_1 = P(s)$ be the selected literal in Q and let $P(s)$ be unifiable with the head of the clause $P(t) : -B$ via the most general unifier θ. Suppose we have as a theorem:

1. $\vdash B_{\gamma} \ \& \ (A_2)_{\gamma}$

where γ is a substitution subsumed by θ. Since s and t are unifiable via γ, by using the equality axioms we can prove (as in the classical case):

2. $\vdash s_{\gamma} = t_{\gamma}$.

Now by construction of **LComp(P)** the following is a theorem:

3. $\vdash (\exists \vec{z}(B \ \& \ s_{\gamma} = t))^{\perp}, P(s_{\gamma})$

where $\vec{z}$ is the list of all the variables in $P(t)$ and B. Thus we have also:

4. $\vdash (B_{\gamma} \ \& \ s_{\gamma} = t_{\gamma})^{\perp}, P(s_{\gamma})$.

By applying the rule of Lemma 1(3) to (2) and (4) we get:

5. $\vdash B_{\gamma}^{\perp}, P(s_{\gamma})$

From (1) and (5) we get

(6) $\vdash P(s_\gamma)$ & $(A_2)_\gamma$

that is, $\vdash Q_\gamma$. ∎

The rest of the proof of the soundness theorem is just a straightforward induction on the complexity of SLDNF derivations which uses Lemmas 6 and 7. Since the inductive argument is essentially the same as for the proof for the classical Clark's completion (the logic used does not play any essential role), we do not repeat it here.

6 Comparison with related work

In this section we compare our completeness result with other completeness results already existing in the literature. (In the formulation of these last results we will alter the notations since we identify a query L_1 *AND* ... *AND* L_m with its logic transcription $L'_1 \wedge ... \wedge L'_m$, where L'_i is obtained from L_i by replacing NOT with $\neg$).

A logic program is called *hierarchic* if its predicates can be assigned to levels so that, given any predicate R of level i and any clause about R:

$$R(t_1, ..., t_n) : -L_1 \ AND ... AND \ L_m$$

the predicates occurring in L_1 *AND*...*AND* L_m are of levels strictly less than i. The following result holds.

Theorem 5 (Completeness for hierarchic programs w.r.t. Comp(P)) *If a logic program P is allowed and hierarchic, then given any allowed query Q and any substitution τ, we have the following:*

1. **Completeness for success:** *if Q_τ is a theorem of Comp(P) then Q succeeds on P with an answer more general than τ;*
2. **Completeness for failure:** *if $\neg Q$ is a theorem of Comp(P) then Q fails on P.*

(Versions of this result appear in [CL 78], [SH 85], [LT 86]).

Our completeness theorem (Theorem 3) shows that such a result may be improved by eliminating the hypothesis 'P is hierarchic' if one takes the logic underlying the completion of the program to be linear rather than classical. The class of programs for which we obtain completeness is still somewhat restrictive (unit clauses of the form $R(x)$ or $R(f(x))$ are still forbidden because of the allowedness condition) but it is indeed larger than the class of programs for which the above theorem holds. For example, the program P_1 of Example 1 (see Section 2) is not hierarchic and we get that R is a classical theorem of Clark's completion of P_1 although the query R loops on P_1. However, R is not a linear theorem of **LComp**(P_1). For another example of a non-hierarchic program to which our completeness result applies, take the program P_2 of Section 2. The theory Comp(P_2) is inconsistent and trivially entails A; however, the query A loops on P_2. The non-logical axioms of **LComp**(P_2) are $\vdash A, A$ and $\vdash A^\perp, A^\perp$ and neither $\vdash A$ nor $\vdash A^\perp$ are linear theorems of such a theory (contraction is forbidden!). As a matter of fact, **LComp**(P_2) is consistent (not every formula is provable) and it admits 'models' in the sense of [GI 87], that is, '*phase-models*'.

Clearly our completeness theorems extends also (modulo the shift of logic) the result of completeness w.r.t. (classical) Clark's completion for allowed '*strict*' programs which appears in [KU 89].

In [SH 89] a semantics for logic programs somewhat different from Clark's completion is proposed: the classical theory T^Ω(P); such a theory is actually a weak version of Comp(P) based on an iterative construction. The following completeness results for SLDNF is proven.

Theorem 6 (Partial completeness for allowed programs w.r.t. T^Ω(P).) *Let P be an allowed logic program, Q be an allowed query and τ be a substitution.*

1. **Full completeness for success:** *if Q_τ is a classical logical consequence of $T^\Omega(P)$ then Q succeeds on P with an answer more general than τ.*
2. **Partial completeness for failure:** *if Q is a* **positive** *query and $\neg Q$ is a classical logical consequence of $T^\Omega(P)$, then Q fails on P.*

(See [SH 89])

The reason why we call such a result *partial* completeness is because (2) holds only for *positive* queries rather than for the more general class of allowed ones. As Shepherdson points out, this restriction is due to the fact that SLDNF resolution is not capable of using the law of contradiction which says that $A \wedge \neg A$ is always false, while $\neg(A \wedge \neg A)$ will always hold in any classical declarative semantics proposed for P. To focus the problem, consider the following example.

Example 6 Let P_6 be the program whose only clause is $R : -R$. We have that $\neg(R \wedge \neg R)$ is a classical logical consequence of $T^\Omega(P_6)$ (as well as of Comp(P_6) indeed) but the query $R \ AND \ NOT(R)$ loops on P_6.

We have seen that if we formulate the completion of the program by the means of linear connectives, we can get rid of the restriction 'Q is positive'. Indeed, $(A \ \& \ A^{\perp})^{\perp}$ is not a general theorem of linear logic, even when weakening is available. To see this, remember that the formula $(A \ \& \ A^{\perp})^{\perp}$ is just the same thing as the formula $A^{\perp} \oplus A$; the latter is provable iff either $A^{\perp}$ is provable or A is provable.

Moreover, given the iterative way in which the theory $T^{\Omega}(P)$ is actually built up, proofs in $T^{\Omega}(P)$ are not objects that one can easily investigate and manipulate; this is not the case for LComp(P).

In [KU 87] and [KU 89] a variant of Kleene's three-valued logic [KL 52] is used to provide a semantics to logic programs. In Kleene's three-valued logic there are three truth-values, namely **t** (true), **f** (false) and **u** (undetermined). A three-valued interpretation $\mathcal{I}$ for a set S of first-order formulas may be seen as a partial interpretation which gives the value **t** or **f** to some formulas in S, but may leave undetermined the truth value of some other formula, to which $\mathcal{I}$ will assign **u**. Notice that in such a logic neither $A \rightarrow A$ nor $A \leftrightarrow A$ are tautologies because if $\mathcal{I}(A) = \mathbf{u}$ then such formulas are assigned the value **u** by $\mathcal{I}$. Kunen's approach consists in building up Comp(P) exactly as Clark does, but to interpret $\neg$, $\vee$ and $\wedge$ in Kleene's three-valued fashion. Moreover the equivalence $\leftrightarrow$ between formulas introduced by the completion when the definition of a predicate is built up:

$$\forall x_1 ... \forall x_n (R(x_1, ..., x_n) \leftrightarrow E_1 \ \vee ... \vee \ E_k)$$

is actually interpreted by an ad hoc three-valued operator $\cong$ which is not definable in term of $\wedge$, $\neg$ and $\vee$ (the so called 'Lukasiewicz's equivalence'). The truth-table of $A \ \cong \ B$ outputs the value **t** when both A and B are assigned the same truth values and **f** otherwise. Thus, $A \ \cong \ A$ is indeed a tautology but $A \ \cong \ A$ is not always false. Let us call Kunen's modification of Clark's completion 3Comp(P). The following result holds.

Theorem 7 (Three-valued completeness for SLDNF modulo allowedness) *If P is an allowed logic program, Q is an allowed query, and τ is a substitution, then*

1. **Completeness for success:** *if Q_{τ} gets the value* **t** *in all (3-valued) models of 3Comp(P) then Q succeeds on P with answer τ.*
2. **Completeness for failure:** *if (the universal closure of) the negation of Q gets the value t in all (3-valued) models of 3Comp(P) then Q fails on P.*

Thus, by means of three-valued logic, Kunen gets a SLDNF-completeness result under the same hypothesis on programs and queries as us, namely

allowedness. From a model-theoretic point of view the two completeness results are indeed strongly interrelated because any three-valued model $\mathcal{M}$ of Comp(P) may be 'simulated' by a linear model $\mathcal{M}'$ (a 'phase-space' [GI 87]), so that being a linear consequence of **LComp**(P) implies being a three-valued consequence of 3Comp(P) (modulo the appropriate translation of logical constants; see the Appendix for more details). Therefore our completeness result is entailed by Kunen's result. For the same reason, our soundness result entails an analogous soundness result w.r.t. 3Comp(P). The important difference between our approach and Kunen's is proof-theoretical. Kunen provides only a *semantical* definition of the three-valued logic which he uses (without proposing any proof system). On the other hand, we do use the proof theory of linear logic and **LComp(P)** may be indeed seen as a *logical axiomatization* of the behaviour of the program P.

Acknowledgements

The author would like to thank J. Y. Girard, N. Bidoit and H. Comon.

References

[CE 88] S. Cerrito, 1988. A linear axiomatization of negation as failure. *Journal of Logic Programming*, to appear.

[CE 90] S. Cerrito, 1990. A linear semantics for allowed logic programs. In *Proceedings of fifth annual IEEE symposium on logic in computer science*, pp. 219-227.

[CL 78] K. L. Clark, 1978. Negation as failure. In *Logic and dabases* (eds. H. Gallaire and J. Minker), pp. 293-322. Plenum, New York.

[GA 86] J. H. Gallier, 1986. *Logic for computer science*, Harper & Row.

[GE 69] G. Gentzen, 1969. *Collected works*, (ed. M. Szabo), North Holland, Amsterdam .

[GI 87] J-Y. Girard, 1987. Linear logic, *Theoretical Computer Science*, **50**, pp. 1-108.

[GI 89] J-Y. Girard , 1989. Towards a theory of geometry of interaction. In *Categories in logic and computer science*, (eds. J. Gray and A. Scedrov), Contemporary Mathematics, Vol. 92, pp. 69-108 AMS, Providence Rhode Island.

[GL 89] J-Y. Girard , Y. Lafont and P. Taylor, 1989. Proofs and types, *Cambridge tracts in theoretical computer science*, Cambridge University Press, Cambridge.

[KL 52] S. C. Kleene, 1952. *Introduction to metamathematics*, Van Nostrand, Princeton .

[KU 87] K. Kunen, 1987. Negation in logic programming, *Journal of Logic Programming*, **4**, pp. 289-308.

[KU 89] K. Kunen, 1989. Signed data dependencies in logic programs, *Journal of Logic Programming*, **7** (3),pp. 231-245.

[LL 87] J. W. Lloyd, 1987. *Foundations of logic programming*, (second edn). Springer, Berlin.

[LT 85] J. W. Lloyd and R. W. Topor, 1985. A basis for deductive database systems, *Journal of Logic Programming* **2** (2), pp. 93-109.

[LT 86] J. W. Lloyd and R. W. Topor, 1986. A basis for deductive database systems II, *Journal of Logic Programming* **3** (1), pp. 55-67.

[SH 84] J. C. Shepherdson, 1984. Negation as failure: a comparison of Clark's completed data base and Reiter's closed world assumption, *Journal of Logic Programming* **1** (1), pp. 51-81.

[SH 85] J. C. Shepherdson, 1985. Negation as failure II, *Journal of Logic Programming* **2** (3), pp. 185-202.

[SH 88] J. C. Shepherdson, 1988. Negation in logic programming. In *Foundations of deductive databases and logic programming*, (ed. J. Minker), pp. 19-87. Morgan Kauffmann, New York,

[SH 89] J. C. Shepherdson, 1989. A sound and complete semantics for a version of negation as Failure, *Theoretical Computer Science*, **65**(3), pp. 343-71.

A Three Valued Models and Phase Semantics

In this Appendix we discuss more closely the relation between linear models of **LComp(P)** and 'Kunen's like' three-valued models of Comp(P) [KU 89]. The notions of phase-space semantics for linear logic which we use are defined in [GI 87]. First of all, notice that Kunen interprets the connective

$\leftrightarrow$ occurring in the axioms of Comp(P) by Lukasiewicz's operator $\cong$ rather than by Kleene's three-valued equivalence simply because he needs to have a logical operation which gives the result **t** when the two arguments have the same truth value (Kleene's three-valued equivalence does not behave like that). As a matter of fact, nothing would change in Kunen's approach if $\cong$ were replaced by the operator $\Leftrightarrow$ defined by the following truth-table:

$\Leftrightarrow$	**t**	**f**	**u**
t	**t**	**f**	**u**
f	**f**	**t**	**u**
u	**u**	**u**	**t**

because given any Kunen-like three-valued interpretation $\mathcal{I}$ of **LComp(P)**, $\mathcal{I}$ is a model of **LComp(P)** iff the three-valued interpretation $\mathcal{I}^\circ$, which differs from $\mathcal{I}$ just because the connective $\leftrightarrow$ is interpreted by $\Leftrightarrow$, is a model of **LComp(P)**. Thus, in the sequel we can do 'as if' Kunen's three-valued interpretation of the connective $\leftrightarrow$ actually were $\Leftrightarrow$.

Proposition 3 *Given any first-order formula F which contains just the connectives $\neg$, $\wedge$, $\vee$ and the quantifiers, let F^l be the linear formula obtained from F by replacing $\neg$ by linear negation, $\wedge$ by &, $\vee$ by $\oplus$, $\leftrightarrow$ by linear equivalence (the $\otimes$ of two linear implications), quantifiers by linear quantifiers. Let $\mathcal{I}$ be a 'Kunen-like' three-valued interpretation for F. There is a linear interpretation $\mathcal{I}^\circ$ such that $\mathcal{I}(F) = \mathbf{t}$ if and only if F^l is valid with respect to $\mathcal{I}^\circ$.*

Proof. Let S be the set $\{0,1,2\}$, and let $*$ be the operation on S defined by: $n*m = inf(n+m, 2)$. The operation $*$ is commutative, associative and admits the neutral element 0. The structure $\mathcal{M} =< S, *, 0 >$ is therefore a commutative monoid; if we choose the set $\{2\}$ as the *anti-phases* set $\perp$ we get a *phase-space* $\mathcal{P} =< \mathcal{M}, \{2\} >$. The following subsets of S are *facts* of $\mathcal{P}$: $\{0,1,2\}$, $\{1,2\}$ and $\{2\}$, because they are the duals respectively of $\{2\}$, $\{1,2\}$, $\{0,1,2\}$. It is also easy to see that they are the only facts, because if a subset X of S is a fact and $x \in X$ then also the successor of $x \in X$ (here the successor of 2 is 2); this because a fact X of P is always the dual of a subset of S. Let h be the bijection between the set of truth-values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ and the set of facts of $\mathcal{P}$ defined by:

$$h(\mathbf{t}) = \{0,1,2\}$$
$$h(\mathbf{u}) = \{1,2\}$$
$$h(\mathbf{f}) = \{2\}$$

A simple calculus shows that for any x, y, z in $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$:

$$\neg x = y \text{ iff } h(x)^{\perp} = h(y)$$
$$x \wedge y = z \text{ iff } h(x) \ \& \ h(y) = h(z)$$
$$x \vee y = z \text{ iff } h(x) \oplus h(y) = h(z)$$
$$x \Leftrightarrow y = z \text{ iff } h(x) \circ\!\!-\!\!\circ h(y) = h(z)$$

(where $h(x) \circ\!\!-\!\!\circ h(y)$ is $(h(x) \multimap h(y)) \otimes (h(y) \multimap h(x))$). Thus, given any 'Kunen-like' three-valued interpretation $\mathcal{I}$ for F, one can build up the linear interpretation $\mathcal{I}^{\circ}$ by taking the domain of $\mathcal{I}^{\circ}$ to be the same domain of $\mathcal{I}$, by interpreting predicate and function symbols of F so to simulate $\mathcal{I}$ and by choosing $\mathcal{P}$ as the associate phase-space. ∎

Corollary 1 *If F^l is a linear consequence of* **LComp(P)**, *then F is a three-valued consequence of $3Comp(P)$.*

A promenade from provability to consistency

Philippe Balbiani *

Abstract

The semantics of logic programming with negation is addressed by means of modal logic. Inspired by the well-known three-valued semantics, a modal completion formula of logic programs is defined with the help of modal operators. Depending on the logical behaviour of these operators, the modal formula completion will alternatively possess the soundness and completeness properties with respect to 'SLDNF-refutability' or the soundness and completeness properties with respect to 'non-SLDNF-refutability'. It will allow a logical characterization of some properties possessed by SLDNF-proof trees as well. Lastly, since one of these properties is closely related to the closed-world assumption rule, the completion formula will give an almost complete proof procedure for this non-monotonic rule of inference.

Outline

We review in Section 1 the basic results on the logical foundations of the negation as failure. Section 2 consists of preliminary definitions of logic programming. In particular, it spells out three declarative semantics of logic programming with or without negation. It is Section 3 which defines the tools which are used in Section 4 to formalize in modal logic some properties of the proof trees of logic programming. Section 5 presents an application of these formalizations as an almost complete proof procedure for the closed world assumption rule.

*Institut de Recherche en Informatique de Toulouse, Toulouse,France