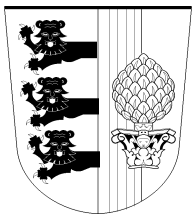
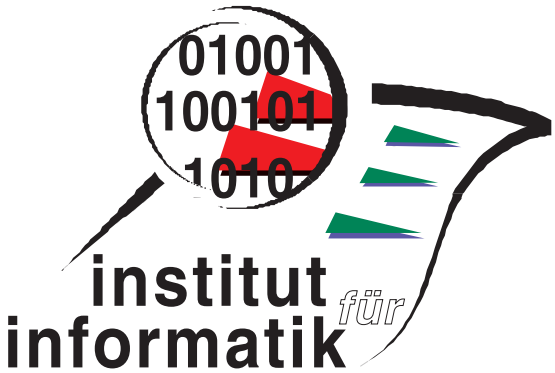
Universität Augsburg



Completeness and Termination of SLDNF-Resolution and Determination of a Selection function using Mode

Ebénézer Ntienjem

Report 1997-06 Dezember 1997



Institut für Informatik
D-86135 Augsburg

Completeness and Termination of SLDNF-Resolution and Determination of a Selection function using Mode

Ebénézer Ntienjem

December 17, 1997

Abstract

We consider a mode of an n-ary predicate symbol with respect to a logic program, which meets the aim of logic programming and captures the spirit of unification as arguments passing mechanism. We prove that the SLDNF-resolution which resolves a non-ground negative literal is complete for an interesting class of logic programs using this mode. To obviously do such a proof we do consider terms modulo variable renaming and map a logic program with a goal to an allowed logic program with an allowed goal, since it is well-known that the SLDNF-resolution is complete for the class of allowed logic programs with allowed goals [9]. The termination of the SLDNF-resolution is studied using a sophisticated selection function which only chooses those literals and clauses that are applicable in the sense that using such literals and clauses the SLDNF-resolution would not be infinite, if a finite SLDNF-resolution does exist.

Keywords: Logic Programming, Proof Theory, Model Theory, Semantics, Resolution, Mode

Contents

1 Introduction

We assume throughout this paper that the reader is acquainted with the basic notions of logic programming. If nothing else is noted, all notations used in the following are borrowed from Apt in [1] or Lloyd in [10]. We say logic program as a short hand for normal respectively general logic program.

Advances in improving the class of logic programs which is complete using the SLDNF-resolution have been made when considering the declaration of the conventional input/output dependencies [16, 18]. In addition, a permutation of the literals occurring in a goal is always assumed. To find such a permutation is as hard as to resolve the logic program with a goal. Hence, a good idea is to reduce the problem of finding a permutation to the case where the permutation is found while resolving the logic program with a goal.

The assumption made when selecting a negative literal is strong and hence restricts the class of logic programs which is complete using the SLDNF-resolution. It is also observed that this assumption on the selection of a negative literal is not used in the proof of the soundness of the SLDNF-resolution [16]. Since the definition of the SLDNF-resolution is top down [10] or 'pseudo' bottom up [9] this assumption is necessary. Besides the observation mentioned above, we observe that there does exist negative literals containing variables, which do finitely failed; for example

$$\neg\, \texttt{member}(v, [])$$

with [] denoting an empty list or

$$\neg\, \texttt{append}(vs, [1|ys], [2, 3, 4]).$$

Such examples are meaningful and correct as long as the SLDNF-resolution is really defined bottom up and the universe of discourse does contain variables modulo variable renamings.

Since logic programs are in general not augmented with an explicit respectively an implicit notion of mode, some problems related with the improvement of the efficiency of the evaluation, the determination of the selection function and the determination of sufficient conditions such that the evaluation might terminate are not easy to solve without restricting the class of logic programs. In the other hand, the absence of an explicit respectively an implicit notion of mode together with the spirit of SLDNF-resolution and the spirit of unification make a logic

program more expressive than the same program formulated in an imperative or a functional language; for example

$$\texttt{append}(x, y, z)$$

formulated as a logic program may also be used to determine all pairs (x, y) such that the concatenation of x and y is z.

We suppose that a mode of an n-ary predicate symbol defined in a logic program is inferred or is declared. We also consider a Herbrand universe which does contain variables. Since the representation of a term containing variables is not unique, we consider terms modulo variable renamings. We consider an extension of the SLDNF-resolution with respect to a logic program, say P, which might be characterized by the following two rules:

NaFF1 if the body of each clause in P fails or A and the head of each clause in P are not unifiable, then $\neg A$ succeeds;

NaFF2 if the body of some clause in P succeeds with the identity substitution and A and the head of this clause are unifiable, then $\neg A$ fails.

It is obvious that this extension of the SLDNF-resolution does include that in [16]. Hence, the selection of a non-ground negative literal is allowed. We then discuss the effect of mode on the evaluation and the determination of the selection function for an interesting and relevant class of logic programs for which the SLDNF-resolution is proven complete.

This paper is organized as follows: in section 2 we briefly fix the syntax of our language and some notational conventions. Section 3 is concerned with the semantics. In section 4 we define an extended SLDNF-resolution which allows the selection of non-ground negative literals. Section 5 briefly considers mode. In section 6 we discuss the completeness result using mode. We then in section 7 discuss the definition of a selection function and state the main theorem of this paper.

2 Syntax and basic notions

We assume that our language for predicate logic is fixed in advance, and does contain, for each $n \geq 0$ a countably infinite set $\mathcal{F}_n$ of function symbols, for each $n \geq 0$ a countably infinite set $\mathcal{P}_n$ of n-ary predicate symbols. Let $\mathcal{V}$ be a countably infinite set of variables. In addition, our language has particular predicate symbols, '$=$', for equality, and '$\cong$', for equality modulo variable renaming; the set $\mathcal{P}_{n\in\mathbb{N}}$ does not contain $=$ and $\cong$.

Let the syntactic categories $\mathcal{F}$ of function symbols, PRED of n-ary predicate symbols be given, $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ of terms, FOR of formulae be defined as usual. Terms are denoted by r, s, t, and atomic formulae by A, B, α, β. The *falsehood* $\bot$ denotes a formula that is false at all or finitely failed. A *literal* is an atomic formula or a

negated atomic formula. Literals are denoted by L, λ. A *program clause* or *clause* for short is a formula of the form

$$\alpha \leftarrow \lambda_1, \cdots, \lambda_n,$$

where α is an atomic formula and also called the *head*, $\lambda_1, \cdots, \lambda_n$ are literals and also called the *body*, and $n \geq 0$; we write $\alpha \leftarrow \epsilon$, if $n = 0$. Note that ',' in the body stands for '$\wedge$'. A *program goal* or *goal* for short is a clause of the form

$$\bot\leftarrow \lambda_1, \cdots, \lambda_n$$

where $n \geq 0$; we write ϵ if $n = 0$. If no confusion is feared, we also write a goal in the form $\lambda_1, \cdots, \lambda_n$. A *logic program* or *program* for short is a finite set of clauses. Instead of considering a logic program to be a set of clauses we let it be the union of the definitions of predicate symbols, where the definition of an n-ary predicate symbol is a set of clauses such that this n-ary predicate symbol does occur in the head of each clause of this set.

By an *expression* we mean a term or a formula. Let $vars(E)$ be the set of variables occurring in the expression E. $\forall E$ denotes the universal closure of E, and $\exists E$ the existential closure of E.

A substitution θ is a function from the set of variables to the set of terms. $dom(\theta)$ denotes the domain of the substitutions θ, its range is denoted by $ran(\theta)$, and $\theta\restriction_V$ denotes its restriction to the set of variables V.

The application of a substitution to an expression and the relation *more general than* between substitutions is defined in the usual way. A substitution θ is an *unifier* of expressions E and F if $E\theta = F\theta$, and is a *most general unifier* (in short: *mgu*) of E and F if it is an unifier which is more general than all other unifiers of E and F. We assume in the following that the properties of substitutions are stated as in [14]. In particular,

(i) if $\theta = \mathrm{mgu}(\lambda_1, \lambda_2)$ is idempotent, then

$$\mathrm{vars}(\theta) \subseteq \mathrm{vars}(\lambda_1) \cup \mathrm{vars}(\lambda_2).$$

(ii) if σ is idempotent,

$$\theta = \mathrm{mgu}(\lambda_1, \lambda_2\sigma) \quad \text{and} \quad \mathrm{vars}(\lambda_1) \cap (\mathrm{vars}(\lambda_2\sigma) \cup \mathrm{dom}(\sigma)) = \emptyset,$$

then $\sigma\theta$ is idempotent.

A clause is *allowed* iff every variable occurring in this clause does occur in at least one positive literal in the body of that clause. A program is *allowed* iff each clause of that program is.

We suppose that our language of discourse has sufficiently many terms. One get sufficiently many terms when assuming as in [8, 9] an infinite universal language in which all programs and goals occur.

Let P be a program and PRED(P) be the set of all predicate symbols occurring in P. Assume the relation "depend on", denoted by $\sqsupseteq$, be defined on the set

PRED(P) as by Kunen in [9]. The relation $\sqsupseteq$ defines an equivalence relation on PRED(P). We denote by $\sqsupset$ the transitive and irreflexive relation $\sqsupseteq$. Hence, $\mathrm{PRED}(P) = \bigcup_{j=1}^{k} \mathrm{PRED}(P_j)$, where PRED($P_j$) is an equivalence class.

Let us for simplicity write $\vec{x}$ for $x_1, \ldots, x_n$ with $n \geq 0$ and write $\tau[\vec{x}\,]$ for an expression τ with all its actual variables among $\vec{x}$. Let in the sequel u stands for a term t such that $\mathrm{vars}(t) = \emptyset$, v stands for a term t which is a variable and s stands for a term of the form $f(t)$ such that $\mathrm{vars}(f(t)) \neq \emptyset$. Let Π or Δ or Γ be a short hand for $\lambda_1, \cdots, \lambda_n$ with $n \geq 1$, and $\sim$ be a new logical symbol which denotes "finitely failed" and acts like $\neg$.

2.1 Partial completion of logic programs

Following Jäger in [7] and Stärk in [15] we form the partially completed program, which consists of the Clark's Equational Theory (CET for short) and a partially completed definition of each n-ary predicate symbol. Let $\bar{q}$ be a new n-ary predicate symbol, whenever q is an n-ary predicate symbol belonging to PRED. Then consider in the sequel the language

$$\bar{\mathcal{L}} = \mathcal{L} \cup \{\bar{q} \mid q \in \mathrm{PRED}\}.$$

A formula of the form $q(\vec{t}\,)$ is a *positive literal* and a formula of the form $\bar{q}(\vec{t}\,)$ is a *negative literal*. Note that for formulae $\sim$ is in general not $\neg$. If λ is a literal of the form $q(\vec{t}\,)$, then $\sim\lambda$ is $\bar{q}(\vec{t}\,)$. If λ is a literal of the form $\bar{q}(\vec{t}\,)$, then $\sim\lambda$ is $q(\vec{t}\,)$. The partially completed definition of each n-ary predicate symbol is briefly obtained as follows: let the definition of an n-ary predicate symbol q consists of m clauses of the form $q(\vec{t}\,) \leftarrow \Pi$ and $x_1, \ldots, x_n$ be new variables. We write

$$q(\vec{t_i}\,) \leftarrow \lambda_1 \wedge \cdots \wedge \lambda_{k(i)}$$

to refer to the i-th clause $1 \leq i \leq m$. Let us write $\mathrm{mgu}(E, F)$ as a shorthand for E and F are unifiable and $\sim\mathrm{mgu}(E, F)$ as a shorthand for the unification of E with F fails finitely. Then the partially completed definition of q is

$$q(\vec{x}\,) \leftarrow \bigvee_{i=1}^{m} (\mathrm{mgu}(\vec{x}, \vec{t_i}) \wedge \bigwedge_{l=1}^{k(i)} \lambda_l) \tag{1}$$

$$\bar{q}(\vec{x}\,) \leftarrow \bigwedge_{i=1}^{m} (\sim\mathrm{mgu}(\vec{x}, \vec{t_i}) \vee \bigvee_{l=1}^{k(i)} \sim\lambda_l). \tag{2}$$

Let P_λ resp. $\bar{P}_\lambda$ be a short hand for (1) resp. (2) in the sequel and P be a program. Then we denote the partial completion of the program P by *pcomp*(P).

3 Semantics

Let $\mathbb{B}$ be the set of boolean values. A *structure* $\mathcal{I}$ for the language consists of a nonempty set, the domain of discourse D, together with an assignment of a semantic object on D for each of the function symbols and for each of the predicate symbols of the language.

- $\mathcal{I}(=)$ is the true identity.
- $\mathcal{I}(\cong)$ is the true identity modulo variable renaming.
- Whenever f is an n-ary function symbol with $n \geq 1$, $\mathcal{I}(f)$ is a function from D^n into D; if $n = 0$, then $\mathcal{I}(f) \in D$.
- Whenever q is an n-ary predicate symbol other than '$=$' or '$\cong$' with $n \geq 1$, $\mathcal{I}(q)$ is a function from D^n into $\mathbb{B}$; if $n = 0$, then $\mathcal{I}(q) \in \mathbb{B}$.

If $\tau[\vec{x}\,]$ is a term, then we define $\mathcal{I}(\tau)\colon D^n \to D$ in the obvious way. Likewise, if $\varphi[\vec{x}\,]$ is a formula, then we define $\mathcal{I}(\varphi)\colon D^n \to \mathbb{B}$ in the obvious way.

Let the Clark's Equational Theory, CET for short, be the equational axioms of the completed program. Note that CET does not depend on a program. We only do consider structures which satisfy CET. Such structures are characterized by the following three conditions:

cet1 $\mathcal{I}(f)$ is an injective function for each n-ary function symbol with $n \geq 1$.

cet2 $\mathcal{I}(f)$ and $\mathcal{I}(g)$ have disjoint ranges whenever f and g are distinct function symbols.

cet3 whenever x_i actually occurs in the term $\tau[\vec{x}\,]$ and $s_1, \ldots, s_n \in D$, it holds $s_i \neq \mathcal{I}(\tau)[\vec{s}\,]$.

The basic idea for the definition of our notion of Herbrand interpretation is to allow variables as elements in the domain of discourse. A term containing variables represents a set of elements whose structure is partially determined. Since in $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ there are different terms that represent the same set, for example $f(x, y)$ and $f(v, w)$, it is adequate to consider $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ modulo variable renaming. We define on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ an equivalence relation $\cong$ as follows: $t \cong r$ iff there exist variable renamings ρ and σ such that $t = r\rho$ and $r = t\sigma$. Let $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\!\cong$ be the set of equivalence classes of $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ with respect to the equivalence relation $\cong$. Assume $t \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$ and $r \in \mathcal{T}_{\mathcal{F},\mathcal{V}}$. Then the relation $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ such that $t \leq r$ if and only if there exists a substitution θ such that $r \cong t\theta$ holds defines a partial order on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$. It is obvious that the order $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ induces an order relation $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\!\cong$.

A *Herbrand structure* $\mathcal{H}$is a structure whose domain of discourse is the quotient set $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\!\cong$. The structure $\mathcal{I}$ is a model of the completed program iff all the sentences of the completed program have truth value true in $\mathcal{I}$. A *Herbrand base* $\mathcal{B}$ is the set all formulae $q(\vec{t})$, where $q \in \mathrm{PRED}$ and $\vec{t} \in (\mathcal{T}_{\mathcal{F},\mathcal{V}})^n/\!\cong$. It is obvious that the order $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\!\cong$ induces an order relation $\leq$ on $\mathcal{B}$. Usually the notion of truth coincides with the one of being an element of; Herbrand interpretations are subsets of the Herbrand base. Since our Herbrand base does contain variables, this notion of truth is no longer correct. With respect to the ordering $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ and hence on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\!\cong$ two definitions of the notion of truth arise:

truth1 a formula $q(\vec{t})$ with $q \in \mathrm{PRED}$ is *true* if there exists a formula $q(\vec{s})$ such that $q(\vec{s})$ is already true and $q(\vec{s}) \leq q(\vec{t})$ holds.

truth2 a formula $q(\vec{t})$ with $q \in \text{PRED}$ is *true* if there exists a variable renaming ρ such that $q(\vec{t})\rho$ is already true .

It is evident that the notion of truth with respect to **truth1** is more general than the notion of truth with respect to **truth2**. A detail discussion of these two notions of truth with respect to a domain of discourse containing variables is given in [6]. We are interested in applying an appropriate kind of these notions of truth to logic programming with negation. Without loss of generality, we do consider in the following the notion of truth according to **truth2**.

We now define the model relation $\models$ on a structure $\mathcal{M}$. We write $\not\models$ for the negation of $\models$. Let $|\mathcal{M}|$ denote the domain of discourse of the structure $\mathcal{M}$. We define $\mathcal{M} \models \varphi$ inductively on the structure of the formula φ as follows:

$$\mathcal{M} \models \epsilon. \tag{3}$$

$$\mathcal{M} \not\models \bot\ . \tag{4}$$

$$\mathcal{M} \models R(t_1, \ldots, t_n) \overset{\text{def}}{\Longleftrightarrow} (\, t_1\rho, \ldots, t_n\rho\,) \in |\mathcal{M}|^n \text{ for some } \rho \in \mathcal{R}. \tag{5}$$

$$\mathcal{M} \models {\sim}R(t_1, \ldots, t_n) \overset{\text{def}}{\Longleftrightarrow} (\, t_1\rho, \ldots, t_n\rho\,) \notin |\mathcal{M}|^n \text{ for all } \rho \in \mathcal{R}. \tag{6}$$

$$\mathcal{M} \models (\varphi \to \psi) \overset{\text{def}}{\Longleftrightarrow} \text{if } \mathcal{M} \models \varphi, \text{ then } \mathcal{M} \models \psi. \tag{7}$$

$$\mathcal{M} \models (\varphi \wedge \psi) \overset{\text{def}}{\Longleftrightarrow} \mathcal{M} \models \varphi \text{ and } \mathcal{M} \models \psi. \tag{8}$$

$$\mathcal{M} \models (\varphi \vee \psi) \overset{\text{def}}{\Longleftrightarrow} \mathcal{M} \models \varphi \text{ or } \mathcal{M} \models \psi. \tag{9}$$

$$\mathcal{M} \models (\forall x\varphi) \overset{\text{def}}{\Longleftrightarrow} \forall t \in |\mathcal{M}| \text{ it holds } \mathcal{M} \models \varphi\{\ t/x\ \}. \tag{10}$$

$$\mathcal{M} \models {\sim}\varphi \overset{\text{def}}{\Longleftrightarrow} \mathcal{M} \not\models \varphi. \tag{11}$$

Let φ be a formula, T be a theory and $\mathcal{U}$ be a structure. Then we write $\mathcal{U} \models \forall\varphi$ if $\forall\varphi$ is true in the structure $\mathcal{U}$; and $T \vdash \varphi$ if φ is derivable from T using the rules of classical first order predicate calculus with equality. Even if the domain of discourse of our model does contain variables which naturally simulate infinite elements, the following result is similar to theorem 6.3 in [8].

Theorem 3.1 *Let P be a program and Γ be a goal. $pcomp(P) \vdash \Gamma$ if and only if $pcomp(P)$ has a model $\mathcal{U}$ such that $\mathcal{U} \models \forall\Gamma$.*

Proof "$\Leftarrow$": by induction on the definition of $\models$.

"$\Rightarrow$": The construction of the model is based on the construction of an universal search structure and the used of the well-known König's lemma. □

4 A bottom up definition of an extended SLDNF-resolution

Our aim in this section and with this extended definition is to eliminate the restrictive condition that a selected negative literal has to be closed in the well-known definition of the SLDNF-resolution as given in [2, 9, 10, 16]. Since this restrictive condition is not used in the proof of the soundness of the so defined

SLDNF-resolution, Stärk in [16] and Schwichtenberg in [13] have suggested to eliminate it. We argue that the elimination of this restrictive condition is best done by a really bottom up definition of the SLDNF-resolution. Note that the definition of the SLDNF-resolution given by Kunen in [9] is not really a bottom up definition. Let $\Gamma = \Delta \wedge \lambda$ be a goal, λ be a selected literal and $\beta \leftarrow \Pi$ be a clause such that the predicate symbol occurring in λ does also occur in β. In the definition of Kunen it is said that Γ holds if $(\Delta \wedge \Pi)\mathrm{mgu}(\lambda, \beta)$ does. This cannot be bottom up at all because of the subgoal Δ. When the SLDNF-resolution is defined bottom up, that means the goal $\Delta \wedge \lambda$ holds if Δ and λ already hold, the elimination of this restrictive condition is obvious and natural.

Let P be a logic program. Like Kunen in [9] we define $\mathbf{Q}(P)$ the set of all goals with respect to P and $\mathbf{RES}(P)$ the set of all pairs $(\Gamma, \theta\restriction_{\mathrm{vars}(\Gamma)})$, where $\Gamma \in \mathbf{Q}(P)$ and $\theta\restriction_{\mathrm{vars}(\Gamma)}$ is a substitution acting on the variables occurring in Γ. Let furthermore $\mathbf{N}(P) \subseteq \mathbf{Q}(P)$ be the set of all goals which fail. Since $\mathbf{N}(P)$ and $\mathbf{RES}(P)$ are related, we define the two subsets $\mathbf{R}(P)$ and $\mathbf{F}(P)$ by simultaneous induction. $\mathbf{F}(P)$ is a subset of $\mathbf{N}(P)$ of those goals that finitely fail. $\mathbf{R}(P)$ is a subset of $\mathbf{RES}(P)$ obtainable by SLDNF. We assume in the following simultaneous inductive definition that $(\beta \leftarrow \Pi) \in P$ is a variant of a clause and that $\mathrm{mgu}(\lambda, \beta)$ also denotes the fact that β and λ are not unifiable as well. In case β and λ are not unifiable it holds $\Pi\mathrm{mgu}(\beta, \lambda) = \bot$. We write in the following $\Delta\mathbf{R}(P)\theta$ for $(\Delta, \theta) \in \mathbf{R}(P)$. Then $\mathbf{R}(P)$ and $\mathbf{F}(P)$ are the least sets that satisfy the following closure properties:

(R0) $\epsilon\mathbf{R}(P)\varepsilon$.

(F0) $\bot \in \mathbf{F}(P)$.

(R1) If $\Delta\mathbf{R}(P)\theta$, λ is a positive literal, $(\beta \leftarrow \Pi) \in P_\lambda$, $\sigma = \mathrm{mgu}(\beta, \lambda\theta)$ and $\Pi\sigma\mathbf{R}(P)\vartheta$, then $(\Delta \wedge \lambda)\mathbf{R}(P)\theta\sigma\vartheta$.

(R2) If $\Delta\mathbf{R}(P)\theta$, λ is a negative literal and for each $(\beta \leftarrow \Pi) \in \bar{P}_\lambda$ it holds that $\sim\Pi\mathrm{mgu}(\beta, \lambda\theta) \in \mathbf{F}(P)$, then $(\Delta \wedge \lambda)\mathbf{R}(P)\theta$.

(F1) If $(\sim\Delta, \theta) \in \mathbf{RES}(P)$, λ is a negative literal and for each $(\beta \leftarrow \Pi) \in \bar{P}_\lambda$ it holds that $\sim\Pi\mathrm{mgu}(\beta, \lambda\theta) \in \mathbf{F}(P)$, then $(\sim\Delta \vee \lambda) \in \mathbf{F}(P)$.

(F2) If $(\sim\Delta, \theta) \in \mathbf{RES}(P)$, λ is a positive literal and for some $(\beta \leftarrow \Pi) \in P_\lambda$ it holds that $\Pi\mathrm{mgu}(\beta, \lambda\theta)\mathbf{R}(P)\vartheta$ with $\mathrm{mgu}(\beta, \lambda\theta)\vartheta$ a variable renaming of λ, then $(\sim\Delta \vee \lambda) \in \mathbf{F}(P)$.

Even if our definition of the SLDNF-resolution is somewhat different from the usually well-known one like [9], the soundness is straightforward and is obviously proven by simultaneous induction on the definition of $\mathbf{R}(P)$ and $\mathbf{F}(P)$ and using Lemma 4.2 below; cf. also Ntienjem in [11].

Theorem 4.1 (Soundness) *Let P be a program, Γ be a goal and θ be a substitution.*

(i) If $\Gamma\mathbf{R}(P)\theta$, then $pcomp(P) \vdash \Gamma\theta$.

(ii) If $\Gamma \in \mathbf{F}(P)$, *then* $pcomp(P) \vdash \sim\Gamma$.

To elegantly prove the completeness we consider two sets $\mathbf{Y}(P)$ and $\mathbf{N}(P)$, which we define by simultaneous induction. We first of all generalize the definition of $\mathbf{R}(P)$ and $\mathbf{F}(P)$. This generalisation leads to the set $\mathbf{Y}(P) \subseteq \mathbf{RES}(P)$ which is the set of all pairs $(\Gamma, \theta\restriction_{\mathrm{vars}(\Gamma)})$ such that $\Gamma\theta$ is true and to the set $\mathbf{N}(P) \subseteq \mathbf{Q}(P)$ which is the set of all goals which fails. The definition of the sets $\mathbf{Y}(P)$ and $\mathbf{N}(P)$ is similar to that of $\mathbf{YES}(P)$ and $\mathbf{NO}(P)$ given by Buchholz in [3] and Stärk in [15]. The sets $\mathbf{Y}(P)$ and $\mathbf{N}(P)$ are the least sets which are closed under the following rules:

(Y0) $(\epsilon, \varepsilon) \in \mathbf{Y}(P)$.

(N0) $\bot \in \mathbf{N}(P)$.

(Y1) If $(\Delta, \theta) \in \mathbf{Y}(P)$, λ is a positive literal, $(\beta \leftarrow \Pi) \in P_\lambda$, σ is a substitution such that $\beta\sigma \cong (\lambda\theta)\sigma$ and $(\Pi, \sigma\chi) \in \mathbf{Y}(P)$, then $(\Delta \wedge \lambda, \theta\sigma\chi) \in \mathbf{Y}(P)$.

(Y2) If $(\Delta, \chi) \in \mathbf{Y}(P)$, λ is a negative literal, for all clauses $(\beta \leftarrow \Pi) \in \bar{P}_\lambda$, for all substitutions θ and σ it holds $\beta\sigma \cong (\lambda\chi)\theta$ and $\sim\Pi\sigma \in \mathbf{N}(P)$, then $(\Delta \wedge \lambda, \chi) \in \mathbf{Y}(P)$.

(N1) If $(\sim\Delta, \vartheta) \in \mathbf{RES}(P)$, λ is a negative literal, for all clauses $(\beta \leftarrow \Pi) \in \bar{P}_\lambda$, for all substitutions θ and σ it holds $\beta\sigma \cong \lambda\vartheta\theta$ and $\sim\Pi\sigma \in \mathbf{N}(P)$, then $(\sim\Delta \vee \lambda) \in \mathbf{N}(P)$.

(N2) If $(\sim\Delta, \vartheta) \in \mathbf{RES}(P)$, λ is a positive literal, for some clause $(\beta \leftarrow \Pi) \in P_\lambda$, for some substitutions σ and θ it holds $\beta\sigma \cong \lambda\vartheta\theta$ and $(\Pi, \sigma\chi) \in \mathbf{Y}(P)$, then $(\sim\Delta \vee \lambda) \in \mathbf{N}(P)$.

The following Lemma shows the relation between the sets $\mathbf{R}(P), \mathbf{F}(P)$ and $\mathbf{Y}(P), \mathbf{N}(P)$ when considering a goal Γ with respect to a program P. This Lemma is proven by simultaneous induction on the definition of $\mathbf{R}(P)$ and $\mathbf{F}(P)$.

Lemma 4.1 *Let* P *be a program,* Γ *be a goal and* θ *be a substitution.*

(i) If $\Gamma\mathbf{R}(P)\theta$, *then* $\Gamma\theta \in \mathbf{Y}(P)$.

(ii) If $\Gamma \in \mathbf{F}(P)$, *then* $\Gamma \in \mathbf{N}(P)$.

We first discuss an useful property of the sets $\mathbf{Y}(P)$ and $\mathbf{N}(P)$. Since it holds that $\mathbf{R}(P) \subseteq \mathbf{Y}(P)$ and $\mathbf{F}(P) \subseteq \mathbf{N}(P)$ the following lemma is a generalization of lemma when considering the sets $\mathbf{Y}(P)$ and $\mathbf{N}(P)$.

Lemma 4.2 *The sets* $\mathbf{Y}(P)$ *and* $\mathbf{N}(P)$ *are closed under substitutions.*

Proof by simultaneous induction on the definition of $\mathbf{Y}(P)$ and $\mathbf{N}(P)$. Cf. Ntienjem in [11] □

It is now interesting to look at the converse of (i) and (ii) in Lemma 4.1 for an interesting and practical class of programs. To reach this objective, we better consider the notion of mode in logic programming with negation.

5 Mode in logic programs

Our aim is to use a mode of an n-ary predicate symbol defined in a program to determine the selection function, to guide the SLDNF-resolution not to be infinite whenever the SLDNF-resolution would be infinite in some case, and to prove the completeness of the SLDNF-resolution for a relevant and interesting class of programs.

The mode is in general useful to both the compiler, for optimization, and the programmer, to help when verifying the correctness of the program. A mode of an n-ary predicate symbol defined in a program is a possible n-tuple of the instantiation of arguments of that predicate symbol in term of some domain. An element of such a domain says something about the degree of instantiation of an argument of an n-ary predicate symbol. Let us denote in the sequel such a domain by M.

In the context of imperative or functional languages, such a domain is the set $M = \{ \text{input}, \text{output} \}$. Note that in imperative or functional languages arguments are passed by pattern matching and a program is evaluated with respect to some fixed order of evaluation. Hence, a mode of an n-ary predicate symbol might be prescribed, that is declared. In this case a mode of an n-ary predicate symbol says how the arguments of this predicate symbol has to be according to the underlying domain M when this predicate symbol occurs and is selected in a goal.

In the context of logic programming languages, arguments are passed using the unification instead of pattern matching and a program is evaluated by the SLDNF-resolution which has no order of evaluation fixed in advance. Because of the unification partially instantiated terms are obvious and it is reasonable to say *closed term* instead of input term. Hence, it is not a good idea if a mode of an n-ary predicate symbol is prescribed. To keep the spirit of unification and that of SLDNF-resolution, logic programming languages are not augmented with the notion of mode. But a mode of an n-ary predicate symbol may be inferred from a logic program if it is said that a mode of an n-ary predicate symbol says how the arguments of this predicate symbol are instantiated according to the underlying domain M when this predicate symbol occurs and is selected in a goal.

Let us first of all find an adequate domain M for logic programming languages. Logic languages do allow partially instantiated terms as arguments. Hence, we classify arguments, that is terms, according to the degree of how they are instantiated. That is $M = \{\text{closed}, \text{partially instantiated}, \text{variable}\}$. Since modes in logic programs have been discussed by many researchers, the domain M for mode purpose is not unique. Warren in [19] uses the set $\{+, -, ?\}$, where "+, −, ?" denotes respectively bound, unbound, and unknown; Stroetmann in [18] uses $\{+, -\}$; Stärk in [16] uses $\{ \text{in}, \text{out}, \text{normal} \}$ where "in, out, normal" stands respectively for input, output and normal(logical) argument; Debray in [4] and Debray and Warren in [5] use the set $\{c, d, e, f, \text{nv}\}$ where "c, d, e, f, nv" denotes respectively the set of closed terms, the set of don't know terms, the empty set of terms, the set of uninstantiated variables and the set of non variable terms.

Let t be a term. Then the term resulting from the replacement of any constant term occurring in t by a symbolic closed term is called an *abstract* term. That

means, the real value of a constant term is not relevant for the purpose of mode determination. A substitution $\sigma = \{ t/v \}$ is an *abstract substitution* if the term t is an abstract term. A program P is an *abstract program* if every term occurring in P is an abstract term. We define in the same way an *abstract literal*, an *abstract clause* and an *abstract goal*.

A mode of an n-ary predicate symbol may then be automatically inferred or declared by the programmer. We say an *inferred mode* respectively a *declared mode* if a mode is inferred respectively declared. We simply say *mode* of an n-ary predicate symbol if the kind of mode of that predicate symbol is not relevant. Interesting is the inference of a mode of an n-ary predicate symbol with respect to a program. To automatically infer a mode of an n-ary predicate symbol with respect to a program, one need (i) an abstract unification as a transformation on abstract terms which is a little different from the well-known unification of terms, (ii) a terminating SLD-like resolution of an abstract program with an abstract goal since an inferred mode of an n-ary predicate symbol is a consequence of the abstract program. Since the negation of a literal does not affect the mode of the n-ary predicate symbol occurring in that literal, an SLD-like resolution is right. Such SLD-like resolution has to be terminating, since it is executed while compiling a program or just before running a program with a goal. Let us denote in the sequel the most general unificator of two abstract terms by *a-mgu*.

We will either investigate the automatic inference of mode of n-ary prerdicate symbols with respect to a program or consider the declaration of the mode by the programmer since doing that we go out of the scope of this paper. A discussion of the automatic inference of inferred mode of n-ary prerdicate symbols with respect to a program is given in [12]. Note that the declaration of a mode of an n-ary predicate symbol with respect to a program is simple but does suffer from the fact that

(i) a partially instantiated term cannot be correctly declared, especially when repeated variables do occur in the head of a clause of the definition of an n-ary predicate symbol (see for example `append`);

(ii) all possible n-tuples of modes of an n-ary predicate symbol, this is k^n where k is the number of elements of the domain M, have to be declared if the aim of logic programming is of interest.

Let us in the sequel write λ for a literal and mean the n-ary predicate symbol occurring in the literal λ as well. In the context of mode we then write $M(\lambda)$ to denote the mode of the n-ary predicate symbol occurring in the literal λ. In the sequel we simply suppose an inferred respectively a declared mode of an n-ary predicate symbol occurring in a program be given.

6 Completeness result for goal moded programs

Let a clause of a program be given. Then a variable occurring only in the head of the clause is called a *head variable*; we denote the set of the head variables of

a clause κ by $hvars(\kappa)$. Let $\Gamma = \lambda_1, \cdots, \lambda_n$ be a goal and

$$M(\Gamma) = \{\ M(\lambda_1), \ldots, M(\lambda_n)\ \}$$

be the set of modes of the predicate symbols occurring in the goal Γ, where $M(\lambda_i)$ is a mode of the predicate symbol occurring in the literal λ_i with $1 \leq i \leq n$. We write for simplicity $xM(\lambda)$ to denote that the mode of x under $M(\lambda)$ is an element of $\{$ closed, variable, partially instantiated $\}$.

Let $\lambda[\vec{t}]$ be a literal, $M(\lambda)$ be a mode of $\lambda[\vec{t}]$ with

$$\mathrm{dom}(M(\lambda)) = \{x_1, \ldots, x_n\}.$$

Since $M(\lambda)$ is an abstract substitution we write it in the form

$$\{r_1/x_1, \ldots, r_n/x_n\}.$$

Then we write $\lambda[\vec{t}] = \lambda[\vec{x}]\sigma$, where

$$\sigma = \{\ t_1/x_1, \ldots, t_n/x_n\ \};$$

we also write $\lambda[\vec{t}]M(\lambda) = \lambda[\vec{x}]\theta$, where

$$\theta = \text{a-mgu}(\{\ \langle t'_1, r_1\rangle, \ldots, \langle t'_n, r_n\rangle\ \}$$

with t'_i an abstract term of the term t_i for $1 \leq i \leq n$. We write simply $\Gamma M(\Gamma)$ for

$$\{\ \lambda_1 M(\lambda_1), \ldots, \lambda_n M(\lambda_n)\ \}$$

whenever a goal Γ is of the form $\bot \leftarrow \lambda_1, \cdots, \lambda_n$ and $M(\Gamma)$ is a set of modes of Γ.

The set of closed variables of λ is

$$cvars(\lambda) = \{\ x\ \mid\ x \in \mathrm{vars}(\lambda) \wedge\ xM(\lambda) = \text{closed}\ \}.$$

The set of variables of λ is

$$ovars(\lambda) = \{\ x\ \mid\ x \in \mathrm{vars}(\lambda) \wedge\ xM(\lambda) \neq \text{closed}\ \}.$$

The following Lemma follows immediately from the definition of the sets cvars(λ) and ovars(λ).

Lemma 6.1 *Let $\kappa = \alpha \leftarrow \lambda_1, \cdots, \lambda_n$ be a clause, $\Gamma = \Delta \wedge \lambda$ be a goal, $M(\lambda)$ be a mode of λ and $\theta = a\text{-}mgu(\alpha, \lambda)$. Then for $1 \leq i \leq n$ it holds*

$$\begin{aligned} cvars(\lambda_i) &= \{\ x\ \mid\ x \in vars(\lambda_i) \wedge x \in cvars(\lambda\theta)\ \} \cup \\ &\quad \{\ x\ \mid\ x \in vars(\lambda_i\theta)\ \wedge\ xM(\lambda_i) = closed\ \}. \qquad (12) \\ ovars(\lambda_i) &= vars(\lambda_i) \setminus cvars(\lambda_i). \qquad (13) \end{aligned}$$

From this Lemma we have

$$\mathrm{cvars}(\kappa) = \mathrm{cvars}(\alpha) \cup \bigcup_{i=1}^{n} \mathrm{cvars}(\lambda_i)\ \text{ and }\ \mathrm{ovars}(\kappa) = \mathrm{ovars}(\alpha) \cup \bigcup_{i=1}^{n} \mathrm{ovars}(\lambda_i),$$

where $\kappa = \alpha \leftarrow \lambda_1, \cdots, \lambda_n$ is a clause.

A program containing head variables is not allowed. Since it is well-known that allowed programs are complete [9], we map an arbitrary program using mode to allowed program. The completeness of a goal moded programs is then straightforward.

A goal Γ is *allowed with respect to* $M(\Gamma)$ if $\Gamma M(\Gamma)$ is allowed.

Definition 6.1 *Let λ be a literal, $\kappa = \alpha \leftarrow \Pi$ be a clause, $M(\lambda)$ and $\theta = mgu(\alpha, \lambda)$ be given. The clause κ is then* allowed with respect to $M(\lambda)$ *if it holds that either*

(i) $hvars(\kappa) \cap ovars(\kappa) = \emptyset$ *and* $\Pi\theta$ *is allowed with respect to* $M(\Pi\theta)$ *or*

(ii) $hvars(\kappa) \cap ovars(\kappa) \neq \emptyset$ *and* $(\bigcup_{L \in \Pi} PRED(P_L)) \subset PRED(P_\alpha)$ *and* $\Pi\theta$ *is allowed with respect to* $M(\Pi\theta)$.

The definition P_α is allowed with respect to $M(\lambda)$ *if each clause occurring in P_α is allowed with respect to $M(\lambda)$.*

In the above definition $\subset$ means proper subset and not simply subset. It follows from (i) in this definition that every allowed program is allowed with respect to any mode.

Since the selection function we are determining finds an ordering of the literals occurring in a goal, the SLDNF-resolution may terminate provided the program does. Hence, we do not discuss the flow dependency of the variables occurring in a goal.

Definition 6.2 *Let P be a program, $\Gamma = \lambda_1, \cdots, \lambda_n$ be a goal, $M(\Gamma)$ be a set of modes of Γ. We say that Γ is* strong resolvable *if for each literal λ_i the definition P_{λ_i} of the predicate symbol occurring in this literal is allowed with respect to $M(\lambda_i)$ for $1 \leq i \leq n$.*

Suppose the goal Γ is not empty, that is $\Gamma \neq \epsilon$. If no literal does exist such that the definition of that literal is allowed with respect to a mode of that literal, then Γ is *strong non-resolvable.*

Lemma 6.2 *Let P be a program and Γ be a goal.*

(i) If $\Gamma = \epsilon$, then Γ is strong resolvable.

(ii) If Γ is strong resolvable and θ is a substitution, then $\Gamma\theta$ is strong resolvable.

(iii) If Γ is strong resolvable and λ is a positive literal such that for some clause $\kappa = \alpha \leftarrow \Pi$ it holds that $\Pi mgu(\lambda, \alpha)$ is strong resolvable, then $\Gamma \wedge \lambda$ is strong resolvable.

(iv) If Γ is strong resolvable and λ is a negative literal such that for all clauses $\kappa = \alpha \leftarrow \Pi$ it holds that $\Pi mgu(\lambda, \alpha)$ is strong resolvable, then $\Gamma \wedge \lambda$ is strong resolvable.

Proof (ii) by induction on the definition of $\mathbf{R}(P)$ and $\mathbf{F}(P)$. (i) and (iii) and (iv) by induction on the structure of a goal. □

Since the definition of each literal occurring in a goal has to be allowed with respect to a mode of that literal, definition 6.2 is very restrictive to be applied in practice. Considering the sample program `append` it is clear that this program is strong resolvable with a goal having a mode

$$(u/xs, u/ys, v/zs) \quad \text{or} \quad (v/xs, v'/ys, u/zs),$$

and strong non-resolvable with a goal having a mode

$$(v/xs, u/ys, v'/zs) \quad \text{or} \quad (v/xs, s/ys, v'/zs).$$

Hence, we weaken definition 6.2 such that the allowedness of the whole definition of a literal occurring in a goal is anymore assumed, whenever that literal is positive.

Definition 6.3 *Let P be a program, $\Gamma = \lambda_1, \cdots, \lambda_n$ be a goal, $M(\Gamma)$ be a set of modes of Γ. We say that Γ is* resolvable *if either*

(i) for each positive literal λ_i there exists a clause κ which is allowed with respect to $M(\lambda_i)$ for $1 \leq i \leq n$ or

(ii) for each negative literal λ_i the definition P_{λ_i} of the predicate symbol occurring in this literal is allowed with respect to $M(\lambda_i)$ for $1 \leq i \leq n$.

If for all $1 \leq i \leq n$ neither (i) nor (ii) holds, then Γ is *non-resolvable*. Note that Lemma 6.2 also holds in the context of resolvable goals if strong resolvable in that Lemma is replaced by resolvable. Reconsidering the sample program `append` it is clear that this program is resolvable with a goal having any mode; for a mode

$$(v/xs, s/ys, v'/zs),$$

the clause

$$\texttt{append}([x|xs], ys, [x|zs]) \leftarrow \texttt{append}(xs, ys, zs)$$

is not allowed. It is obvious that any strong resolvable goal is also resolvable. We now formulate the converse of lemma 4.1.

Lemma 6.3 *Let P be a program, Γ be a resolvable goal and θ be a substitution.*

(i) If $\Gamma\theta \in \mathbf{Y}(P)$, then $\Gamma\mathbf{R}(P)\theta\sigma$ for some substitution σ.

(ii) If $\Gamma \in \mathbf{N}(P)$, then $\Gamma \in \mathbf{F}(P)$.

Proof by simultaneous induction on the definition of $\mathbf{Y}(P)$ and $\mathbf{N}(P)$. □

7 Determination of an admissible selection function

The determination of an admissible selection function is not widely discussed in the literature. In most cases it is supposed that a selection function exists. When implementing the SLDNF-resolution the samplest selection function, namely choice of the first literal in the list, is used and clauses are chosen according to the order in which they are declared. Note that reordering the list does not meet the objective of SLDNF-resolution and furthermore the reordering of the list required that the literal which is next to be selected be known in advance if it is not assumed that the user has to care of the order of the literals in a goal. We say selection function, but it is really a selection map or application, since it may be the case that for example two literals have the same image.

7.1 A measure function and an admissible selection function

To reach our aim, we first of all define a measure function μ from a set of goals into $\mathbb{N}\times\mathbb{N}$ as follows:

$$\begin{array}{rccc} \mu: & \Gamma & \longmapsto & \mathbb{N}\times\mathbb{N} \\ & \lambda & \longrightarrow & \langle |cvars(\lambda)|, |ovars(\lambda)| \rangle \end{array}$$

where $|M|$ is the cardinality of the set M. It is well-known that $(\mathbb{N}\times\mathbb{N}, \leq)$ is well-founded; then the minimal element, the greatest lower bound respectively least upper bound are well-known as well.

A literal λ is now selected if the image $\mu(\lambda)$ is a minimal element with respect to $\leq$. It may be the case that for a goal the number of minimal elements with respect to $\leq$ is greater than one. Since this situation may frequently occur, some further considerations besides the measure function are needed to permit an admissible selection.

7.2 An admissible selection function

The measure function is required, but it does not suffice for a reasonable selection which lets the SLDNF-resolution terminate. Two methods will be useful to reach the aim of making the SLDNF-resolution terminate while using an admissible selection function. The idea is to only consider those literals which are resolvable in the sense of section 6.

Definition 7.1 *Let Γ be a goal, $\lambda \in \Gamma$ be a literal and $M(\lambda)$ be an instantiated mode of λ. The literal λ is* strong admissible *if*

(i) $\mu(\lambda)$ is minimal and

(ii) the definition of the predicate symbol occurring in that literal is allowed with respect to $M(\lambda)$.

This definition is still restrictive for an interesting class of programs. Let us next weaken it to get a somewhat relevant class of programs.

Definition 7.2 *Let Γ be a goal, $\lambda \in \Gamma$ be a literal and $M(\lambda)$ be an instantiated mode of λ. The literal λ is* admissible *if*

(i) *$\mu(\lambda)$ is minimal and either*

(ii) *λ is a positive literal and there exists a clause which is allowed with respect to $M(\lambda)$ or λ is a negative literal and the definition of the predicate symbol occurring in that literal is allowed with respect to $M(\lambda)$.*

Lemma 7.1 *If a literal occurring in a goal is strong admissible, then that literal is also admissible.*

Theorem 7.1 (Completeness) *Let P be a program, Γ be a goal and θ be a substitution. Suppose that the goal Γ is resolvable.*

(i) *If $pcomp(P) \vdash \Gamma\theta$, then $\Gamma\mathbf{R}(P)\theta$.*

(ii) *If $pcomp(P) \vdash \sim\Gamma$, then $\Gamma \in \mathbf{F}(P)$.*

Proof follows with Lemma 6.3 together with theorem 3.1. □

Let us consider the programs `plus, times` and `factorial`. s denotes the successor function. Our definition of these predicate symbols is a little different from that defined in [17] pages 36-39. This formulation ensures the termination of the SLDNF-resolution of this program with any goal, when the selection is defined as above.

$$
\begin{array}{lll}
C_1: & \texttt{plus}(0, y, y,) & \leftarrow \\
C_2: & \texttt{plus}(s(x), 0, s(x),) & \leftarrow \\
C_3: & \texttt{plus}(s(x), s(y), s(z)) & \leftarrow \texttt{plus}(x, s(y), z) \\
C_4: & \texttt{times}(0, y, 0) & \leftarrow \\
C_5: & \texttt{times}(s(x), 0, 0) & \leftarrow \\
C_6: & \texttt{times}(s(x), s(y), s(z)) & \leftarrow \texttt{times}(x, s(y), w), \texttt{plus}(s(y), w, s(z)) \\
C_7: & \texttt{factorial}(0, s(0)) & \leftarrow \\
C_8: & \texttt{factorial}(s(x), s(z)) & \leftarrow \texttt{factorial}(x, y), \texttt{times}(s(x), y, s(z))
\end{array}
$$

Then considering this program with the goal

$$\Gamma = \bot \leftarrow \sim\texttt{factorial}(v, s(s(s(0)))\,)$$

we have $\Gamma\mathbf{R}(P)\varepsilon$. To prove this fact let $s^n(0)$ denotes $\underbrace{s(s(\cdots(s(0))\cdots))}_{n-\text{times}}$. For any $n \geq 0$ it holds that $\Delta = \texttt{plus}(v, w, s^n(0))$ is resolvable with $M(\Delta) = \{v/x, w/y, s^n(0)/z\}$ since P_{plus} is allowed with respect to $M(\Delta)$. This fact also holds for any goal $\Delta = \texttt{plus}(t_1, t_2, t_3)$ such that t_i is $s^n(0)$ for $1 \leq i \leq 3$. A similar conclusion is made for the goals

$$\texttt{times}(t_1, t_2, t_3) \quad \text{and} \quad \texttt{factorial}(t_1, t_2).$$

Hence,

$$\texttt{times}(t_1, t_2, t_3) \wedge \texttt{plus}(t_2, t_3, t_3')$$

is resolvable and also

$$\texttt{factorial}(t_1', t_2) \wedge \texttt{times}(t_1, t_2, t_3)$$

with $t_3 = s^n(0)$.

Let us consider the well known program append.

$$\begin{array}{lll} C_1: & \texttt{append}([], y, y,) & \leftarrow \\ C_2: & \texttt{append}([x|xs], ys, [x|zs]),) & \leftarrow \texttt{append}(xs, ys, zs,) \end{array}$$

One prove in the same way as above that the example goal

$$\bot \leftarrow \sim \texttt{append}(vs, [1|ys], [2, 3, 4])$$

given in the introduction is resolvable. Therefore, it holds that

$$\sim \texttt{append}(vs, [1|ys], [2, 3, 4]) \mathbf{R}(P)\varepsilon.$$

8 Conclusion and future works

Putting all together, the domain of discourse which contains variables modulo renaming, a mode of an n-ary predicate symbol with respect to a logic program which captures the spirit of the unification when passing the arguments, an extended SLDNF-resolution which does select non-ground negative literals, and an admissible selection function, an interesting and practical and relevant class of logic programs is then defined and its theoretical background proven.

The partial completion of a program does not simply meet the way the SLDNF-resolution handles negative literals. Since the problem of inconsistency of the program completion has already been solved by Jäger in [7] and Stärk in [15], future works may be investigated on a program completion which is consistent and does meet the way the SLDNF-resolution handles negative literals.

References

[1] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 10, pages 495–574. Elsevier, 1990.

[2] K. R. Apt and K. Doets. A new definition of sldnf-resolution. *Journal of Logic Programming*, 18:177–190, 1994.

[3] W. Buchholz. Negation-as-failure-kalkül. Oberseminarvortrag, Ludwig-Maximilians-Universität München, November 1992.

[4] S. K. Debray. Static inference of modes and data dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):418–450, 1989.

[5] S. K. Debray and D. S. Warren. Automatic mode inference for logic programs. *Journal of Logic Programming*, 5:207–229, 1988.

[6] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Journal of Theoretical Computer Science*, 69:289–318, 1989.

[7] G. Jäger. Some proof-theoretic aspects of logic programming. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *logic and algebra of specification*, pages 113–142. Springer, Berlin, 1993.

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

[9] K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.

[10] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1987.

[11] E. Ntienjem. Completeness Result of SLDNF-Resolution for a relevant Class of Logic Programs. Technical Report 1997-4, Institut für Informatik, Universität Augsburg, Germany, December 1997. submitted to JICSLP 1998.

[12] E. Ntienjem. A descriptive mode inference for logic programs. Technical Report 1997-5, Institut für Informatik, Universität Augsburg, Germany, December 1997. submitted to Journal of Logic Programming.

[13] H. Schwichtenberg. Logikprogrammierung. Vorlesungsausarbeitung, Ludwig-Maximilians-Universität München, Wintersemester, 1993.

[14] W. Snyder and J. H. Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.

[15] R. F. Stärk. From logic programs to inductive definitions. Technical report, Mathematisches Institut, Universität München, 1993. for Logic Colloquium 1993, Keele, Great Britain.

[16] R. F. Stärk. Input/output dependencies of normal logic programs. *Journal of Logic and Computation*, 4(3):249–262, 1994.

[17] L. Sterling and E. Shapiro. *The Art of Prolog Advanced Programming Techniques*. The MIT Press, Cambridge, Massachusetts, 1986.

[18] K. Stroetmann. A completeness result for sldnf–resolution. *Journal of Logic Programming*, 15(4):337–355, 1993.

[19] D. H. D. Warren. Implementing prolog – compiling predicate logic programs. Research Report 39,40, Department of Artificial Intelligence, University of Edinburgh, 1977.