



**Centrum voor Wiskunde en Informatica**  
Centre for Mathematics and Computer Science

---

R.N. Bol

Generalizing completeness results for loop checks

Computer Science/Department of Software Technology

Report CS-R9025

June

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

# Generalizing Completeness Results for Loop Checks

Roland N. Bol

Centre for Mathematics and Computer Science  
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands  
Phone: (+31) - 20 - 592 4080. E-mail: bol@cw.nl.

## Abstract

Loop checking is a mechanism for pruning infinite SLD-derivations. In [BAK], *simple loop checks* were introduced and their soundness, completeness and relative strength was studied. Since no sound and complete simple loop check exists even in the absence of function symbols, subclasses of programs were determined for which the (sound) loop checks introduced in [BAK] are complete.

In this paper, the *Generalization Theorem* is proven. This theorem presents a method to extend (under certain conditions) a class of programs for which a given loop check is complete to a larger class, for which the loop check is still complete. Then this theorem is applied to the results of [BAK], giving rise to stronger completeness theorems.

It appears that unnecessary complications in the proof of the theorem can be avoided by introducing a *normal form* for SLD-derivations, allowing only certain most general unifiers. This normal form might have other applications than those in the area of loop checking.

*Key Words and Phrases:* deductive databases, logic programming, termination, loop checking, PROLOG interpreter.

*1985 Mathematical Subject Classification:* 68Q40, 68T15

*1987 CR Categories:* F.3.2, F.4.1, H.3.3, I.2.3.

*Note:* This research was partly supported by Esprit BRA-project 3020 Integration.

## 1. Introduction

Logic programming is advocated as a formalism for writing executable specifications. However, even when such specifications are correct in the logical sense, their execution by means of a PROLOG interpreter may lead to divergence. This problem motivated the study of loop checking mechanisms which are used to discover loops in SLD-derivations (see [B], [BW], [C], [vG], [KT], [PG], [SGG], [SI], [V]).

The use of a PROLOG interpreter augmented with a loop check renders a larger class of logic programs not only logically, but also operationally correct. Which class it is depends on the selected loop check. To study such problems in a rigorous way, Apt, Bol and Klop [ABK] introduced a number of natural concepts like soundness, completeness and relative strength of loop checks. Also the concept of a *simple loop check* was introduced, being a loop checking mechanism which does not depend on the analyzed logic program. It was shown that no sound and complete simple loop check exists even for programs without function symbols.

In [BAK], a number of natural simple loop checks was introduced. These loop checks were proven to be sound, but only complete for certain classes of programs. For each of these loop checks, one or more such classes were determined.

Here, the problem of finding classes of programs for which a simple loop check is complete is addressed in more generality. The main theorem of this paper is called the *Generalization Theorem*, since it allows us to generalize certain completeness results: given that a loop check  $L$  is complete for a class of programs  $\mathcal{G}$ , we may conclude that  $L$  is also complete (w.r.t. the leftmost selection rule) for a class of programs extending  $\mathcal{G}$ , provided that  $L$  and  $\mathcal{G}$  satisfy some natural conditions. Basically, the theorem is only applicable to a class of programs  $\mathcal{G}$  if  $\mathcal{G} = \{ P \mid \text{every clause in program } P \text{ satisfies } Pr \}$ , for some property  $Pr$  of clauses that is ‘local’ to clauses (that is, whether a clause satisfies  $Pr$  or not is independent of the rest of the program). We say that  $\mathcal{G}$  is the class of *Pr-programs*. By allowing the addition of atoms in clauses that cannot give rise to recursive calls to the head of the clause (so called *non-recursive* atoms), the class of *nr-extended Pr-programs* is obtained. The Generalization Theorem states that if the loop check  $L$  is complete for  $Pr$ -

programs, then  $L$  is also complete for nr-extended Pr-programs, provided that the non-recursive atoms are resolved before other atoms are selected (for simplicity, this is achieved by using the leftmost selection rule, and putting the non-recursive atoms left from the other atoms in the clause). Note that the property of being a non-recursive atom is *not* local to clauses; therefore the theorem cannot be applied repeatedly.

In the proof of the Generalization Theorem, we make use of certain properties of SLD-derivations that are in a normal form, tentatively called *normal* SLD-derivations. In normal SLD-derivations, only certain mgu's may be used. This normal form might well have other applications than those in the area of loop checking.

Once the proof of the Generalization Theorem is given, it is applied to two completeness results presented in [BAK] concerning a loop check that indeed satisfies the conditions of the Generalization Theorem. The extension of one of these completeness results is straightforward, whereas for the other one a more elaborate analysis is needed.

## 2. Basic notions

In this section we recall the basic notions concerning loop checking, as presented in [BAK]. Throughout this paper we assume familiarity with the concepts and notations of logic programming as described in [L]. For two substitutions  $\sigma$  and  $\tau$ , we write  $\sigma \leq \tau$  when  $\sigma$  is more general than  $\tau$  and for two expressions  $E$  and  $F$ , we write  $E \leq F$  if  $F$  is an instance of  $E$ . An SLD-derivation step from a goal  $G$ , using a clause  $C$  and an mgu  $\theta$ , to a goal  $H$  is denoted as  $G \Rightarrow_{C,\theta} H$ . By an SLD-derivation we mean an SLD-derivation in the sense of [L] *or an initial segment of it*.

### 2.1 Loop checks

The purpose of a loop check is to prune every infinite SLD-tree to a finite subtree of it containing the root. We define a loop check as a set of SLD-derivations: the derivations that are pruned exactly at their last node. Such a set of SLD-derivations  $L$  can be extended in a canonical way to a function  $f_L$  from SLD-trees to SLD-trees by pruning in an SLD-tree the nodes in  $\{ G \mid \text{the SLD-derivation from the root to } G \text{ is in } L \}$ . We shall usually make this conversion implicitly.

**DEFINITION 2.1.**

Let  $L$  be a set of SLD-derivations.

$\text{RemSub}(L) = \{ D \in L \mid L \text{ does not contain a proper subderivation of } D \}$ .

$L$  is *subderivation free* if  $L = \text{RemSub}(L)$ . □

In order to render the intuitive meaning of a loop check  $L$ : ‘every derivation  $D \in L$  is pruned *exactly* at its last node’, we need that  $L$  is subderivation free. Note that  $\text{RemSub}(\text{RemSub}(L)) = \text{RemSub}(L)$ .

In the following definition, by a *variant* of a derivation  $D$  we mean a derivation  $D'$  in which in every derivation step, atoms in the same positions are selected and the same program clauses are used.  $D'$  may differ from  $D$  in the renaming that is applied to these program clauses for reasons of standardizing apart and in the mgu used. It has been shown that in this case every goal in  $D'$  is a variant of the corresponding goal in  $D$  (see [LS]). Thus any variant of an SLD-refutation is also an SLD-refutation and yields the same computed answer substitution up to a renaming.

**DEFINITION 2.2.**

A *simple loop check* is a computable set  $L$  of finite SLD-derivations such that  $L$  is closed under variants and subderivation free. □

In [BAK], loop checks are treated in a more general way. There non-simple loop checks occur: their behaviour may depend on the program the interpreter is confronted with. In this paper, we shall only consider simple loop checks. Therefore we shall usually omit the adjective ‘simple’.

**DEFINITION 2.3.**

Let  $L$  be a loop check. An SLD-derivation  $D$  of  $P \cup \{G\}$  is *pruned by*  $L$  if  $L$  contains a subderivation  $D'$  of  $D$ . □

**2.2 Soundness and completeness**

Using a loop check should definitely not result in a loss of success. (The fact that even losing individual solutions may be undesirable is irrelevant for this paper.) On the other hand, the purpose of a loop check is to reduce the search space for top-down interpreters. We would like to end up with a finite search

space. This is the case when every infinite derivation is pruned. This leads to the following definitions.

**DEFINITION 2.4.**

A loop check  $L$  is *weakly sound* if for every program  $P$  and goal  $G$ , and SLD-tree  $T$  of  $P \cup \{G\}$ : if  $T$  contains a successful branch, then  $f_L(T)$  contains a successful branch.  $\square$

**DEFINITION 2.5.**

A loop check  $L$  is *complete w.r.t. a selection rule  $R$  for a class of programs  $\mathcal{C}$* , if for every program  $P \in \mathcal{C}$  and goal  $G$  in  $L_P$ , every infinite SLD-derivation of  $P \cup \{G\}$  via  $R$  is pruned by  $L$ .  $\square$

In general, comparing loop checks is difficult. The following relation comparing loop checks is not very general: most loop check will be incomparable with respect to it. Nevertheless it turns out to be very useful.

**DEFINITION 2.6.**

Let  $L_1$  and  $L_2$  be loop checks.  $L_1$  is *stronger than*  $L_2$  if every SLD-derivation  $D_2 \in L_2$  contains a subderivation  $D_1 \in L_1$ .  $\square$

In other words,  $L_1$  is stronger than  $L_2$  if every SLD-derivation that is pruned by  $L_2$  is also pruned by  $L_1$ . Note that the definition implies that every loop check is stronger than itself. The following theorem will enable us to obtain soundness and completeness results for loop checks which are related by the ‘stronger than’ relation, by proving soundness and completeness for only one of them.

**THEOREM 2.7 (Relative Strength).** *Let  $L_1$  and  $L_2$  be loop checks, and let  $L_1$  be stronger than  $L_2$ .*

- i) *If  $L_1$  is weakly sound, then  $L_2$  is weakly sound.*
- ii) *If  $L_2$  is complete then  $L_1$  is complete.*

PROOF. Straightforward.  $\square$

The undecidability of the halting problem implies that there cannot be a weakly sound and complete loop check for logic programs in general, as logic programming has the full power of recursion theory. So our first step is to rule out programs that compute over an infinite domain. We shall do so by restricting our attention to programs without function symbols, so called *function-free* programs, for which the Herbrand Universe is finite. However, it appears that even with this restriction, there is no weakly sound and complete loop check.

**THEOREM 2.8.** *There is no weakly sound and complete simple loop check for function-free programs.*

PROOF. See [ABK]. □

It was shown in [ABK] that weakly sound and complete non-simple loop checks exist for function-free programs, but that they are in a sense too powerful. Therefore, we found it to be more useful to develop some simple loop checks, and to find classes of programs for which these loop checks are complete.

### 2.3 Some simple loop checks

In this section we introduce two groups of weakly sound simple loop checks. How we arrived at these loop checks and why we found them interesting was discussed in [BAK]. Here we restrict ourselves to giving the definitions and basic theorems (without proofs).

The first group of loop checks we consider consists of the so-called ‘equality checks’. In fact, each equality check should be defined separately. This would yield almost identical definitions. Therefore we compress them into two definitions, trusting that the reader is willing to understand our notation. The equality relation between goals (regarded as *lists*) is denoted by  $=_L$ . (In [BAK], also variants of these loop checks are considered, regarding goals as *multisets*.)

#### DEFINITION 2.9.

The *Equals Variant/Instance of Goal<sub>List</sub>* check is the set of SLD-derivations  $\text{EVG/EIG}_L = \text{RemSub}(\{ D \mid D = ( G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{k-1} \Rightarrow_{C_k, \theta_k} G_k )$   
such that for some  $i$ ,  $0 \leq i < k$ , there is a renaming/substitution  $\tau$  such that  $G_k =_L G_i \tau \}$ ). □

**DEFINITION 2.10.**

The *Equals Variant/Instance of Resultant<sub>List</sub>* check is the set of SLD-derivations  $EVR/EIR_L = \text{RemSub}(\{ D \mid D = ( G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{k-1} \Rightarrow_{C_k, \theta_k} G_k )$  such that for some  $i$ ,  $0 \leq i < k$ , there is a renaming/substitution  $\tau$  such that  $G_k =_L G_i \tau$  and  $G_0 \theta_1 \dots \theta_k = G_0 \theta_1 \dots \theta_i \tau$  ).  $\square$

**THEOREM 2.11 (Equality Soundness).** *The equality checks are weakly sound loop checks.*

PROOF. See [ABK].  $\square$

We now define a class of programs for which the equality checks are complete in the absence of function symbols (as was shown in [ABK]). This class of programs is closely related to the class of programs discussed in [ŠŠ]. For a formal definition, we use the notion of the *dependency graph*  $D_P$  of a program  $P$ .

**DEFINITION 2.12.**

The *dependency graph*  $D_P$  of a program  $P$  is a directed graph whose nodes are the predicate symbols appearing in  $P$  and

$(p, q) \in D_P$  iff there is a clause in  $P$  using  $p$  in its head and  $q$  in its body.

$D_P^*$  is the reflexive, transitive closure of  $D_P$ . When  $(p, q) \in D_P^*$ , we say that  $p$  *depends on*  $q$ . For a predicate symbol  $p$ , the *class of*  $p$  is the set of predicate symbols  $p$  ‘mutually depends’ on:  $clp(p) = \{ q \mid (p, q) \in D_P^* \text{ and } (q, p) \in D_P^* \}$ .  $\square$

**DEFINITION 2.13.**

Given an atom  $A$ , let  $rel(A)$  denote its predicate symbol. Let  $P$  be a program.

A clause  $A_0 \leftarrow A_1, \dots, A_n$  ( $n \geq 0$ ) is called *restricted w.r.t. P* if for  $i = 1, \dots, n-1$ ,  $rel(A_i)$  does not depend on  $rel(A_0)$  in  $P$ .

The atoms  $A_1, \dots, A_{n-1}$  are called *non-recursive atoms* of  $A_0 \leftarrow A_1, \dots, A_n$ .

A program  $P$  is called *restricted* if every clause in  $P$  is restricted w.r.t.  $P$ .  $\square$

**THEOREM 2.14 (Equality Completeness).** *All equality checks are complete w.r.t. the leftmost selection rule for function-free restricted programs.*

PROOF. See [ABK].  $\square$

The second group of loop checks we consider consists of the so-called ‘subsumption checks’. Again, we define them by means of two parametrized definitions. The inclusion relation between goals regarded as lists is denoted by  $\subseteq_L$ . Note:  $L_1 \subseteq_L L_2$  if all elements of  $L_1$  occur in the same order in  $L_2$ ; they need not to occur on adjacent positions. For example,  $(a,c) \subseteq_L (a,b,c)$ .

**DEFINITION 2.15.**

The *Subsumes Variant/Instance of Goal<sub>List</sub>* check is the set of SLD-derivations  $SVG/SIG_L = \text{RemSub}(\{ D \mid D = ( G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{k-1} \Rightarrow_{C_k, \theta_k} G_k )$   
such that for some  $i$ ,  $0 \leq i < k$ , there is a  
renaming/substitution  $\tau$  with  $G_k \supseteq_L G_i \tau \}$ ).  $\square$

**DEFINITION 2.16.**

The *Subsumes Variant/Instance of Resultant<sub>List</sub>* check is the set of SLD-derivations  $SVR/SIR_L = \text{RemSub}(\{ D \mid D = ( G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{k-1} \Rightarrow_{C_k, \theta_k} G_k )$  such that for some  $i$ ,  $0 \leq i < k$ ,  
there is a renaming/substitution  $\tau$  with  
 $G_k \supseteq_L G_i \tau$  and  $G_0 \theta_1 \dots \theta_k = G_0 \theta_1 \dots \theta_i \tau \}$ ).  $\square$

**THEOREM 2.17 (Subsumption Soundness).** *All subsumption checks are weakly sound loop checks.*

PROOF. See [BAK].  $\square$

We now show three classes of programs for which the subsumption checks are complete in the absence of function symbols. Since the subsumption checks are stronger than the ‘corresponding’ equality checks, the first result follows immediately.

**THEOREM 2.18 (Subsumption Completeness 1).** *All subsumption checks are complete w.r.t. the leftmost selection rule for function-free restricted programs.*

PROOF. By the Relative Strength Theorem 2.7 and the Equality Completeness Theorem 2.14.  $\square$

The remaining two classes of programs for which the subsumption checks are complete in the absence of function symbols are the following.

**DEFINITION 2.19.**

A clause  $C$  is *non-variable introducing* (in short *nvi*) if every variable that appears in the body of  $C$  also appears in the head of  $C$ .

A program  $P$  is *nvi* if every clause in  $P$  is *nvi*. □

**DEFINITION 2.20.**

A clause  $C$  has the *single variable occurrence* property (in short *svo*) if in the body of  $C$ , no variable occurs more than once.

A program  $P$  is *svo* if every clause in  $P$  is *svo*. □

**THEOREM 2.21 (Subsumption Completeness 2).** *All subsumption checks are complete for function-free nvi programs.*

PROOF. See [BAK]. □

**THEOREM 2.22 (Subsumption Completeness 3).** *All subsumption checks are complete for function-free svo programs.*

PROOF. See [BAK]. □

### 3. The choice of most general unifiers

We now divert for a moment from the subject of loop checking. It appears that, in order to prove the Generalization Theorem in section 4, we need some auxiliary results regarding SLD-derivations. These results can be obtained by putting extra requirements on the most general unifiers in those derivations. In this section we introduce these requirements and show why we consider them to be justifiable. Finally we prove the lemma's needed in section 4.

#### 3.1 Relevant and idempotent mgu's

The general feeling is that, in order to obtain mathematical elegance, the definition of an SLD-derivation must leave the choice of variables as free as possible. However, during the evolution of this definition, the allowable freedom was continuously overestimated. For example, in the first edition of

[L], the input clause was only standardized apart from the current goal, and not from the goals and clauses preceeding it. Thereby, the undesirable derivation of Figure 1 was allowed. In the second edition of [L], this has been corrected. However, yet another anomalous derivation is shown in Figure 2.

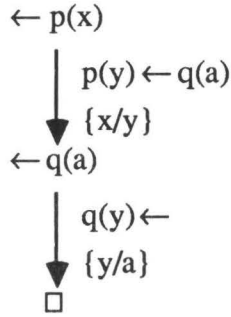


FIGURE 1

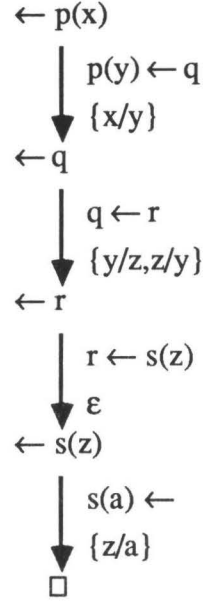


FIGURE 2

It is not clear whether or not this derivation is allowed by [L] (does  $z$  appear in the derivation before the goal  $\leftarrow r$  ?), but by [A] it definitely is, although later on in [A] it is assumed that all mgu's are relevant (a unifier of  $A$  and  $B$  is *relevant* if it acts only on variables in  $A$  and  $B$ ) and idempotent. It appears that the requirement that the mgu is relevant is redundant, as idempotent mgu's are always relevant. First of all, from now on we assume that only idempotent mgu's are used. Under this assumption we prove some properties of SLD-derivations. The first property we prove is that a variable cannot occur somewhere in the derivation, disappear and later reappear.

**LEMMA 3.1.** *Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be an SLD-derivation and let  $0 \leq i \leq j \leq k (< |D|)$ . Let  $C_0 = \emptyset$ . If  $x \in \text{var}(C_i) \cup \text{var}(G_i)$  and  $x \in \text{var}(G_k)$ , then  $x \in \text{var}(G_j)$ .*

PROOF. We use induction on  $j$  from  $k$  down to  $i$ . For  $j = k$  the claim is trivial. Now assume that for some  $j \geq i$ ,  $x \in \text{var}(G_{j+1})$ . We prove that  $x \in \text{var}(G_j)$ . Let  $G_j = \leftarrow(A, R)$ , where  $A$  is the selected atom in  $G_j$ . Let  $C_{j+1} = H \leftarrow B$ . Then  $\theta_{j+1}$  is an (idempotent) mgu of  $A$  and  $H$  and  $G_{j+1} = \leftarrow(B, R)\theta_{j+1}$ . So  $x \in \text{var}(B, R)\theta_{j+1}$ , hence for some  $y \in \text{var}(B, R)$ ,  $x \in \text{var}(y\theta_{j+1})$ . Two cases arise.

- $x = y$ . Then  $x \in \text{var}(R) \subseteq \text{var}(G_j)$ . ( $x \notin \text{var}(B)$  since  $x \in \text{var}(C_i) \cup \text{var}(G_i)$  and  $B$  is standardized apart.)
- $x \neq y$ . Then, since  $\theta_{j+1}$  is relevant,  $x \in \text{var}(A, H)$ . So  $x \in \text{var}(A) \subseteq \text{var}(G_j)$ . (Again  $x \notin \text{var}(H)$  since  $x \in \text{var}(C_i) \cup \text{var}(G_i)$  and  $H$  is standardized apart.)

So in both cases we have  $x \in \text{var}(G_j)$ .  $\square$

The following definition captures the notion that two variables in a goal are related, i.e. that they might be unified in an attempt to refute the goal. (Compare this notion with *connected (sets of) predicate instances* in [N].) We then prove that when two variables occur unrelated in a certain goal, they cannot be related in any goal later in the derivation.

### DEFINITION 3.2.

Let  $S$  be a set of atoms. We define the relation  $\sim_S$  on variables as:

$$x \sim_S y \text{ if there is an atom } A \text{ in } S \text{ such that } x, y \in \text{var}(A).$$

Obviously,  $\sim_S$  is a symmetrical relation. Now we define the relation  $\approx_S$  to be the transitive and reflexive closure of  $\sim_S$ . Then  $\approx_S$  is an equivalence relation.

An equivalence class of  $\approx_S$  is called a *chain (in  $S$ )*. For  $x \in \text{var}(S)$ , the chain of  $x$  is denoted by  $C_S(x)$ , or  $C(x)$  whenever  $S$  is clear from the context.  $\square$

**LEMMA 3.3.** *Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be an SLD-derivation and let  $0 < i (< |D|)$ . If  $x \approx_{G_i} y$  and  $x, y \in \text{var}(G_{i-1})$ , then  $x \approx_{G_{i-1}} y$ .*

PROOF. Let  $G_{i-1} = \leftarrow(A, R)$ , where  $A$  is the selected atom in  $G_{i-1}$ . Let  $C_i = H \leftarrow B$  and let  $\theta_i$  be an mgu of  $A$  and  $H$ . Assume  $x \neq y$  (for  $x = y$  the claim is trivial). Since  $x \approx_{G_i} y$ , there is a sequence of variables  $x = w_1, w_2, \dots, w_{2n} = y$  in  $G_i$  such that  $w_{2j-1} \approx_{B\theta_i} w_{2j}$  for  $1 \leq j \leq n$  and  $w_{2j} \sim_{R\theta_i} w_{2j+1}$  for  $1 \leq j < n$ .

Now, since every variable  $w_j \in \text{var}(R\theta_i)$ , we can fix for it a corresponding variable  $z_j \in \text{var}(G_{i-1})$  such that  $w_j \in \text{var}(z_j\theta_i)$  ( $1 \leq j \leq 2n$ ). Since  $\theta_i$  is idempotent, and  $x, y \in \text{var}(G_i)$ , it is easy to see that  $w_1 = x = x\theta_i$  and  $w_{2n} = y = y\theta_i$ . Therefore we can fix  $z_1 = x$  and  $z_{2n} = y$ . Now let  $1 \leq j < 2n$ .

We prove that  $z_j \approx_{G_{i-1}} z_{j+1}$ . Two cases arise.

-  $j$  is even, so  $w_j \sim_{R\theta_i} w_{j+1}$ .

Then there is an atom  $E$  in  $R$  such that  $w_j, w_{j+1} \in \text{var}(E\theta_i)$ . So we have variables  $v_j, v_{j+1} \in \text{var}(E)$  such that  $w_j \in \text{var}(v_j\theta_i)$  and  $w_{j+1} \in \text{var}(v_{j+1}\theta_i)$ . So  $v_j \sim_E v_{j+1}$ , and hence  $v_j \sim_R v_{j+1}$ . For  $v_j$  (and analogously for  $v_{j+1}$ ) two subcases arise.

-  $v_j = z_j$ . Then  $v_j \approx_A z_j$ .

-  $v_j \neq z_j$ . Then, since  $w_j \in \text{var}(v_j\theta_i) \cap \text{var}(z_j\theta_i)$  and  $\theta_i$  is relevant, we have  $v_j, z_j \in \text{var}(A)$ . Hence  $v_j \approx_A z_j$ .

Therefore  $z_j \approx_A v_j \sim_R v_{j+1} \approx_A z_{j+1}$ , so  $z_j \approx_{G_{i-1}} z_{j+1}$ .

-  $j$  is odd, so  $w_j \approx_{B\theta_i} w_{j+1}$ .

If  $w_j = w_{j+1}$ , then  $z_j = z_{j+1}$ , so  $z_j \approx_{G_{i-1}} z_{j+1}$ . Otherwise, we can prove that  $z_j \in \text{var}(A)$  (and analogously  $z_{j+1} \in \text{var}(A)$ ). Again two subcases arise.

-  $z_j\theta_i \neq z_j$ . Then  $z_j \in \text{var}(A)$ :  $\theta_i$  is relevant and  $z_j \in \text{var}(G_{i-1})$ , so  $z_j \notin \text{var}(H)$ .

-  $z_j\theta_i = z_j$ . Then  $w_j = z_j \in \text{var}(B\theta_i)$ , say  $v_j \in \text{var}(B)$  such that  $z_j \in \text{var}(v_j\theta_i)$ . Then  $v_j\theta_i \neq v_j$ , since  $v_j \in \text{var}(B)$ ,  $z_j \in \text{var}(G_{i-1})$  and  $B$  is standardized apart.

Therefore  $v_j \in \text{var}(H)$ , and hence  $z_j \in \text{var}(A)$ .

Now  $z_j \sim_A z_{j+1}$ , so  $z_j \approx_{G_{i-1}} z_{j+1}$ .

Therefore we have  $x = z_1 \approx_{G_{i-1}} z_2 \approx_{G_{i-1}} z_3 \approx_{G_{i-1}} \dots \approx_{G_{i-1}} z_{2n} = y$ .  $\square$

### 3.2. Normal SLD-derivations

In fact, it appears to be convenient to restrict the choice of the mgu even more by disallowing the ‘needless renaming of variables in a derivation’. We explain this now. When we have a variable  $x$  in the selected atom of the goal which is to be unified with a variable  $y$  in the input clause, then two idempotent mgu’s are available:  $\{x/y\}$  and  $\{y/x\}$ .

When  $\{x/y\}$  is chosen, it is likely that the variable  $y$  occurs further on in the derivation as a substitute for  $x$ , whereas  $x$  itself does not occur any more. On the other hand, if  $\{y/x\}$  is chosen, the variable  $x$  is retained and the variable  $y$  will not occur in any goal of the derivation. Therefore the renaming from  $x$  to  $y$  is considered to be a needless renaming. So we choose  $\{y/x\}$ , thereby retaining the ‘older’ variable  $x$  and adjusting the ‘newer’ variable  $y$ .

A more indirect instance of the same principle is shown in the derivation

$$\leftarrow A(x) \Rightarrow_{A(x') \leftarrow B(x',y), \{x'/x\}} \leftarrow B(x,y) \Rightarrow_{B(z,z) \leftarrow, \{y/x, z/x\}} \square.$$

In the first step  $\{x'/x\}$  is chosen for the reason described above. In the second step, the choice of  $\{x/z, y/z\}$  is out of the question for the same reason. However, this still leaves the choice between  $\{x/y, z/y\}$  and  $\{y/x, z/x\}$ . Although  $x$  and  $y$  occur both in  $B(x, y)$ ,  $x$  appears earlier in the derivation than  $y$ . Therefore we choose  $\{y/x, z/x\}$ , thereby again retaining the older variable  $x$  and adjusting the newer variable  $y$ .

It is important to note two things. Firstly, Lemma 3.1 says that a variable cannot be introduced, disappear and later on in the derivation reappear, which would complicate the decision criterion given above. Secondly, the choice of the mgu is still non-deterministic, as is shown in the derivation

$$\leftarrow A \Rightarrow_{A \leftarrow B(x, y), \varepsilon} \leftarrow B(x, y) \Rightarrow_{B(z, z) \leftarrow, \{y/x, z/x\}} \square.$$

Here the choice between  $\{y/x, z/x\}$  and  $\{x/y, z/y\}$  is arbitrary.

We now formalize these intuitions.

**DEFINITION 3.4.**

Let  $D = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots)$  be an SLD-derivation.

For every variable  $x$  occurring in  $D$ , we define

$$\text{tag}(x) = \min \{ i \mid x \in \text{var}(G_i) \cup \text{var}(C_i) \}.$$

$D$  is a *normal* SLD-derivation if for every  $i > 0$  ( and  $i < |D|$  when  $D$  is finite),

- $\theta_i$  is idempotent and
- for every variable  $x \in \text{var}(G_{i-1})$ : if  $x\theta_i$  is a variable, then  $\text{tag}(x) \geq \text{tag}(x\theta_i)$ .  $\square$

Intuitively, the lower the tag of a variable is, the ‘older’ it is. The following lemma shows that we may restrict our attention to normal SLD-derivations.

**LEMMA 3.5.** *Every SLD-derivation has a normal variant.*

PROOF. We introduce a slightly changed version of the unification algorithm of Martelli & Montanari ([MM]). Using this algorithm for computing the mgu yields a normal SLD-derivation.

When  $p(s_1, \dots, s_n)$  and  $p(t_1, \dots, t_n)$  are to be unified, first the set of equations  $\{s_1 = t_1, \dots, s_n = t_n\}$  is constructed. This set is then transformed according to the following six rules:

- (a)  $E \dot{\cup} \{t = x\} \Rightarrow E \cup \{x = t\}$  if  $t \notin \text{VAR}$  or  $\text{tag}(t) < \text{tag}(x)$ ,
- (b)  $E \dot{\cup} \{x = x\} \Rightarrow E$ ,
- (c1)  $E \dot{\cup} \{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)\} \Rightarrow E \cup \{s_1 = t_1, \dots, s_n = t_n\}$  ( $n \geq 0$ ),
- (c2)  $E \dot{\cup} \{f(s_1, \dots, s_n) = g(t_1, \dots, t_m)\} \Rightarrow \text{failure}$  if  $f \neq g$ ,
- (d1)  $E \dot{\cup} \{x = t\} \Rightarrow E\{x/t\} \cup \{x = t\}$  if  $x \notin \text{var}(t)$  and  $x \in \text{var}(E)$ ,
- (d2)  $E \dot{\cup} \{x = t\} \Rightarrow \text{failure}$  if  $x \neq t$  and  $x \in \text{var}(t)$ ,

until none of these rules is applicable. (Here  $\dot{\cup}$  denotes the disjoint union.) Now we take  $\theta = \{x/t \mid (x = t) \in E\}$ .

The change w.r.t. the original algorithm is in rule (a), where now tags are taken into account. Whenever  $x\theta = y \neq x$ , we have that  $(x = y) \in E$  and no rules are applicable on  $E$ , hence  $\text{tag}(x) \geq \text{tag}(y)$  (otherwise rule (a) would be applicable). Showing that the algorithm terminates and that a resulting substitution is indeed an idempotent mgu of  $p(s_1, \dots, s_n)$  and  $p(t_1, \dots, t_n)$  is straightforward.  $\square$

### 3.3. Properties of normal SLD-derivations

In this section we prove some properties of normal SLD-derivations that appear to be needed in the next section. The reader who is not interested in such technical details is encouraged to skip this section.

**LEMMA 3.6.** *Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be a normal SLD-derivation and let  $0 \leq j < k (< |D|)$ . Let  $C$  be a chain in  $G_j$ .*

*Then  $C\theta_k \cap \text{VAR} \subseteq C$ .*

**PROOF.** Let  $x \in C$  and assume that  $x\theta_k$  is a variable. We prove that  $x\theta_k \in C$ .

If  $x\theta_k = x$  then clearly  $x\theta_k \in C$ .

Otherwise,  $x \in \text{var}(G_{k-1})$ , since  $\theta_k$  is relevant and by standardizing apart,  $x \notin \text{var}(C_k)$ .  $D$  is normal,  $x \in \text{var}(G_{k-1})$  and  $x\theta_k$  is a variable, so  $\text{tag}(x) \geq \text{tag}(x\theta_k)$ . So, with  $C_0 = \emptyset$ , for some  $i$ ,  $0 \leq i = \text{tag}(x\theta_k) \leq \text{tag}(x) \leq j < k$ ,  $x\theta_k \in \text{var}(C_i) \cup \text{var}(G_i)$ .  $x\theta_k \neq x$  and  $\theta_k$  is relevant, so  $x\theta_k \in \text{var}(G_{k-1})$  ( $x\theta_k \notin \text{var}(C_k)$  because  $x\theta_k \in \text{var}(C_i) \cup \text{var}(G_i)$ ,  $i < k$  and  $C_k$  is standardized apart). Now by Lemma 3.1,  $x\theta_k \in \text{var}(G_j)$ , since  $i \leq j < k$ . Again by Lemma 3.1, for every  $l$  such that  $i \leq l < k$ ,  $x \in \text{var}(G_l)$  and  $x\theta_k \in \text{var}(G_l)$ .  $x$  and  $x\theta_k$  occur both in the selected atom of  $G_{k-1}$ . Therefore  $x \approx_{G_{k-1}} x\theta_k$ . Applying Lemma 3.3  $k-1-j$  times yields that  $x \approx_{G_j} x\theta_k$ . Hence  $x\theta_k \in C$ .  $\square$

**COROLLARY 3.7.** Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be a normal SLD-derivation of a function-free program  $P$  and  $G_0$  and let  $0 \leq j < k (< |D|)$ . Then  $\text{var}(G_j \theta_k) \subseteq \text{var}(G_j)$ .

PROOF. Let  $x \in \text{var}(G_j \theta_k)$ .  $P$  is function-free, so for some  $y \in \text{var}(G_j)$ ,  $x = y \theta_k$ . Now by Lemma 3.6,  $x = y \theta_k \in C_{G_j}(y) \theta_k \cap \text{VAR} \subseteq C_{G_j}(y) \subseteq \text{var}(G_j)$ .  $\square$

**COROLLARY 3.8.** Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be a normal SLD-derivation of a function-free program  $P$  and  $G_0$  and let  $0 \leq j < k (< |D|)$ . Then  $\text{var}(G_j \theta_{j+1} \dots \theta_k) \subseteq \text{var}(G_j)$ .

PROOF. Repeatedly using Corollary 3.7, we have  $\text{var}((G_j \theta_{j+1}) \theta_{j+2} \dots \theta_k) \subseteq \text{var}(G_j \theta_{j+2} \dots \theta_k) \subseteq \dots \subseteq \text{var}(G_j \theta_k) \subseteq \text{var}(G_j)$ .  $\square$

**COROLLARY 3.9.** Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_{i-1} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be a normal SLD-derivation and let  $0 \leq j < k (< |D|)$ . Let  $C$  be a chain in  $G_j$ . Then  $C \theta_{j+1} \theta_k \cap \text{VAR} \subseteq C \theta_{j+1}$  and  $C \theta_{j+1} \dots \theta_k \cap \text{VAR} \subseteq C \theta_{j+1}$ .

PROOF. If  $j+1 = k$ , then the claim is trivial. So assume  $j+1 < k$ .

Let  $x \in C \theta_{j+1}$  and assume that  $x \theta_k$  is a variable. We prove that  $x \theta_k \in C \theta_{j+1}$ . By Lemma 3.6,  $x \in C \theta_{j+1} \cap \text{VAR}$  implies  $x \in C$ . Therefore, again by Lemma 3.6,  $x \theta_k \in C \theta_k \cap \text{VAR} \subseteq C$ . Two cases arise.

- $x \theta_k \theta_{j+1} = x \theta_k$ . Then  $x \theta_k \in C$  implies  $x \theta_k = x \theta_k \theta_{j+1} \in C \theta_{j+1}$ .
- $x \theta_k \theta_{j+1} \neq x \theta_k$ . Then  $x \theta_k \notin \text{var}(G_{j+1})$ , since  $\theta_{j+1}$  is idempotent. If  $x \theta_k = x$  then  $x \theta_k \in C \theta_{j+1}$  follows immediately. Now assume  $x \theta_k \neq x$ . Then  $x \theta_k \in \text{var}(G_{k-1})$  (by standardizing apart,  $x \theta_k \notin \text{var}(C_k)$ ). So we have  $x \theta_k \in C \subseteq \text{var}(G_j)$ ,  $x \theta_k \in \text{var}(G_{k-1})$ , but  $x \theta_k \notin \text{var}(G_{j+1})$ . This contradicts Lemma 3.1.

Now  $((C \theta_{j+1}) \theta_{j+2}) \dots \theta_k \cap \text{VAR} \subseteq (C \theta_{j+1}) \theta_{j+3} \dots \theta_k \cap \text{VAR} \subseteq \dots \subseteq (C \theta_{j+1}) \theta_k \cap \text{VAR} \subseteq C \theta_{j+1}$ .  $\square$

In order to formulate the final property of normal derivations we prove in this section, we need the following definition.

**DEFINITION 3.10.**

(This definition is equivalent to the definition of local selection functions in [V].) A selection rule  $R$  is *local* if every SLD-derivation  $D = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots)$  via  $R$  satisfies the following property. If in a goal  $G_i$ , an atom  $A$  is selected and in a

goal  $G_j$  ( $j > i$ ) the further instantiated version  $B\theta_{i+1}\dots\theta_j$  of the atom  $B$  in  $G_i$  is selected, then  $A$  is resolved completely between  $G_i$  and  $G_j$ .  $\square$

It is easy to see that the leftmost selection rule and the rightmost selection rule are examples of local selection rules.

**COROLLARY 3.11.** *Let  $D = G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow_{G_{i-1}} \Rightarrow_{C_i, \theta_i} G_i \Rightarrow \dots$  be a normal SLD-derivation of a function-free program  $P$  and  $G_0$  and let  $0 \leq j < k$  ( $< |D|$ ). Let  $A$  be the selected atom in  $G_j$ . Suppose a local selection rule is used between  $G_j$  and  $G_k$  and  $A$  is not completely resolved before  $G_k$ . Then  $\text{var}(A\theta_k) \subseteq \text{var}(A)$  and  $\text{var}(A\theta_{j+1}\dots\theta_k) \subseteq \text{var}(A)$ .*

PROOF. Let  $x \in \text{var}(A)$  and assume that  $x\theta_k$  is a variable. We prove that  $x\theta_k \in \text{var}(A)$ . Let  $G_j = (A, R)$  and regard the derivation  $\leftarrow A = H_j \Rightarrow_{C_{j+1}, \theta_{j+1}} H_{j+1} \Rightarrow \dots \Rightarrow_{C_k, \theta_k} H_k$  (hence for  $j \leq i \leq k$ ,  $G_i = (H_i, R\theta_{j+1}\dots\theta_i)$ ). Note that this derivation exists, since a local selection rule is used and  $A$  is not completely resolved before  $G_k$ , and note that the derivation is normal. Now  $x \in \text{var}(A) = \text{var}(H_j)$  implies  $x\theta_k \in \text{var}(H_j) = \text{var}(A)$  by Corollary 3.7.

Now  $\text{var}((A\theta_{j+1})\theta_{j+2}\dots\theta_k) \subseteq \text{var}(A\theta_{j+2}\dots\theta_k) \subseteq \dots \subseteq \text{var}(A\theta_k) \subseteq \text{var}(A)$ .  $\square$

## 4. Generalizing completeness results

The rest of this paper discusses the completeness of loop checks. *Therefore we assume from now on the absence of function symbols.* In this section we shall prepare, formulate and prove the Generalization Theorem, the main theorem of this paper. This theorem states that, given a loop check, and given a class of programs for which this loop check is complete, the loop check is (under certain conditions) also complete w.r.t. the leftmost selection rule for a larger class of programs.

### 4.1. Preparation

The formulation of the Generalization Theorem requires the formalization of the classes of programs for which it is applicable. Roughly, these classes of programs are characterized by the condition that all clauses in the program satisfy some (preferably decidable) property. We do not go into details about these

properties: we assume that the notion ‘a clause  $C$  satisfies a property  $Pr$ ’ is given.

**DEFINITION 4.1.**

Let  $Pr$  be a property of clauses. A program  $P$  *satisfies*  $Pr$  ( $P$  is a *Pr-program*) if every clause in  $P$  satisfies  $Pr$ .  $\square$

**DEFINITION 4.2.**

A property of clauses  $Pr$  is *closed under instantiation* if for every clause  $C$  that satisfies  $Pr$  and for every substitution  $\sigma$ ,  $C\sigma$  satisfies  $Pr$ .  $\square$

Note that  $C\sigma$  is not necessarily a ground instance of  $C$ . The Generalization Theorem is only valid for properties that are closed under instantiation. However, in the next section, where we shall give some examples of the use of the Generalization Theorem, we shall also consider a property that is *not* closed under instantiation. A detailed inspection of the proof of the Generalization Theorem enables us to derive useful results for this property as well.

The Generalization Theorem is only valid for loop checks satisfying certain conditions. These conditions are formalized here. The first condition is that the loop check is ‘safe for goal extension’. Informally, this means that when we have a derivation that is pruned by the loop check, adding some atoms to the initial goal that are never selected (before the derivation is pruned), yields a pruned derivation again.

**DEFINITION 4.3.**

A loop check  $L$  is *safe for goal extension* if for every SLD-derivation  $D$  of  $P \cup \{\leftarrow G_0\}$  that is pruned by  $L$ , an SLD-derivation of  $P \cup \{\leftarrow (G_0, H_0)\}$  which selects the same atoms, and uses the same input clauses and mgu’s as  $D$  is also pruned by  $L$ .  $\square$

The second condition is that the loop check is ‘safe for initialization’. Informally, this means that when we have a derivation that is pruned by the loop check, adding some derivation steps in front of it (‘initialization steps’), yields a pruned derivation again.

**DEFINITION 4.4.**

A loop check  $L$  is *safe for initialization* if for every SLD-derivation  $D = (G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1} \Rightarrow_{C_{i+2}, \theta_{i+2}} G_{i+2} \Rightarrow \dots)$  that is pruned by  $L$  ( $i > 0$ ), every derivation  $(G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots \Rightarrow G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1} \Rightarrow_{C_{i+2}, \theta_{i+2}} G_{i+2} \Rightarrow \dots)$  in which in  $G_i, G_{i+1}, \dots$  the same atoms are selected as in  $D$ , is pruned by  $L$ .  $\square$

The third condition is that the loop check is ‘safe for detailing’. Informally, this means that when we have a derivation that is pruned by the loop check, replacing every derivation step by one or more steps giving the same computed answer (‘showing the details of one step in several steps’), yields a pruned derivation again.

**DEFINITION 4.5.**

A loop check  $L$  is *safe for detailing* if for every SLD-derivation  $D = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots)$  that is pruned by  $L$ , every derivation of the form  $(G_0 \Rightarrow_{C_1^1, \tau_1^1} H_1^1 \Rightarrow \dots \Rightarrow H_{n_1-1}^1 \Rightarrow_{C_{n_1}^1, \tau_{n_1}^1} G_1 \Rightarrow_{C_1^2, \tau_1^2} H_1^2 \Rightarrow \dots \Rightarrow H_{n_2-1}^2 \Rightarrow_{C_{n_2}^2, \tau_{n_2}^2} G_2 \Rightarrow \dots)$  with for every  $i \geq 0$ :  $\tau_1^i \dots \tau_{n_i}^i \text{ lvar}(G_0, G_1, \dots, G_i) = \theta_i \text{ lvar}(G_0, G_1, \dots, G_i)$  and in which in  $G_0, G_1, \dots$  the same atoms are selected as in  $D$ , is pruned by  $L$ .  $\square$

Finally, for a certain property  $\text{Pr}$ , we describe the larger class of programs for which the loop check is complete according to the Generalization Theorem (so-called *nr-extended Pr* programs), given that the loop check is complete for *Pr*-programs. In section 5 it will appear that the resemblance between the following definition and Definition 2.13 is not a coincidence.

**DEFINITION 4.6.**

Let  $P$  be a program. A clause  $C = (H \leftarrow \text{NR}, R)$  is *nr-extended Pr w.r.t. P* if the clause  $H \leftarrow R$  satisfies  $\text{Pr}$  and for every atom  $A$  in  $\text{NR}$ ,  $\text{rel}(A)$  does not depend on  $\text{rel}(H)$  in  $P$ .  $\text{NR}$  is called the *non-recursive part* of  $C$  and  $R$  is called the *Pr-part*. A program  $P$  is *nr-extended Pr* if every clause in  $P$  is *nr-extended Pr w.r.t. P*.  $\square$

**4.2. The Generalization Theorem**

We can now formulate the Generalization Theorem.

**THEOREM 4.7 (GENERALIZATION THEOREM).** *Let  $Pr$  be a property of clauses that is closed under instantiation. Let  $L$  be a loop check such that*

- *$L$  is complete for  $Pr$ -programs,*
- *$L$  is safe for goal extension,*
- *$L$  is safe for initialization,*
- *$L$  is safe for detailing.*

*Then  $L$  is complete w.r.t. the leftmost selection rule for nr-extended  $Pr$  programs.*

In the rest of this section, we shall assume that  $Pr$  is a property and  $L$  is a loop check satisfying the above conditions. For proving this theorem, we use the following lemma.

**LEMMA 4.8.** *Let  $P$  be a nr-extended  $Pr$ -program and let  $G_0$  be a goal in  $L_P$ . Let  $D$  be an infinite SLD-derivation of  $P \cup \{G_0\}$  via the leftmost selection rule. Suppose that*

(\*) *for no goal  $G_i = (G, H)$  in  $D$  ( $i \geq 0$ ), the derivation of  $P \cup \{G\}$  (using the same input clauses, mgu's and selection rule as  $D$ ) is pruned by  $L$ .*

*Then  $D$  is pruned by  $L$ .*

Before proving this lemma, we show that the Generalization Theorem is an immediate consequence of it.

**PROOF OF THE GENERALIZATION THEOREM.** Let  $P$  be an nr-extended  $Pr$ -program,  $G_0$  a goal in  $L_P$  and  $D$  an infinite SLD-derivation of  $P \cup \{G_0\}$ . Two cases arise.

- i) For no goal  $(G, H)$  in  $D$ , the derivation of  $G$  (using the same input clauses, mgu's and selection rule as  $D$ ) is pruned by  $L$ . Then by Lemma 4.8,  $D$  is pruned by  $L$ .
- ii) Otherwise, there is a goal  $(G, H)$  in  $D$  for which the derivation of  $G$  (using the same input clauses, mgu's and selection rule as  $D$ ) is pruned by  $L$ . Then the tail of  $D$  starting at this goal  $(G, H)$  is pruned, since  $L$  is safe for goal extension. So  $D$  is pruned by  $L$  too, since  $L$  is also safe for initialization.  $\square$

PROOF OF LEMMA 4.8. The dependency graph  $D_p$  defines a (well founded) partial ordering  $\leq$  of the set  $\{clp(p) \mid p \text{ is a predicate symbol in } L_P\}$ . Therefore we may assume as induction hypothesis (by a complete induction on  $\leq$ ), that this lemma has been proved for every derivation of  $P \cup \{G\}$  where  $G$  contains only strict  $\leq$ -smaller predicate symbols than the  $\leq$ -largest predicate symbol in  $G_0$ .

CLAIM 1.  $D$  is of the form  $(G_0 \Rightarrow_{C_1, \tau_1^1} H_1^1 \Rightarrow \dots \Rightarrow H_{n_1-1}^1 \Rightarrow_{C_{n_1}, \tau_{n_1}^1} G_1 \Rightarrow_{C_1^2, \tau_1^2} H_1^2 \Rightarrow \dots \Rightarrow H_{n_2-1}^2 \Rightarrow_{C_{n_2}, \tau_{n_2}^2} G_2 \Rightarrow \dots)$  for some derivation  $D' = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow_{C_2, \theta_2} G_2 \Rightarrow \dots)$ , with for every  $i \geq 0$ :  $\tau_1^i \dots \tau_{n_i}^i \text{ lvar}(G_0, G_1, \dots, G_i) = \theta_i \text{ lvar}(G_0, G_1, \dots, G_i)$ , and where  $C_1, C_2, \dots$  all satisfy Pr. Moreover, in the goals  $G_0, G_1, \dots$ , the same atoms are selected in  $D$  and  $D'$ .

The lemma follows from Claim 1:  $D'$  is a derivation of  $\{G_0, C_1, C_2, \dots\}$ ,  $\{C_1, C_2, \dots\}$  is an Pr-program, and  $L$  is complete for Pr-programs, therefore  $D'$  is pruned by  $L$ . Hence  $D$  is pruned by  $L$ , since  $L$  is safe for detailing.

PROOF OF CLAIM 1. We prove the claim by induction. Suppose we have constructed  $D'$  and proved the claim up to the goal  $G_i$ . (Up to  $G_0$ , the claim is trivial.)

Let  $G_i = \leftarrow A_1, \dots, A_n$ , let  $C = C_1^{i+1} = (A \leftarrow NR, R)$  and let  $\theta = \theta_1^{i+1}$ . Suppose

that  $NR$  is the non-recursive part of the body of  $C$  and that  $R$  is the Pr-part. The next step in  $D$  is  $G_i \Rightarrow_{C, \theta} \leftarrow (NR, R, A_2, \dots, A_n) \theta$ . Let  $D_1$  be the SLD-derivation of  $P \cup \{\leftarrow NR \theta\}$  that uses the same input clauses, mgu's and selection rule as the tail of  $D$  starting at  $\leftarrow (NR, R, A_2, \dots, A_n) \theta$ . Four cases arise.

1)  $NR$  is empty.

This is a special case of case 4:  $P \cup \{\leftarrow NR \theta\}$  is immediately successfully refuted. (If  $G_0$  is  $\leq$ -minimal, then this is the only possible case, since then  $\text{rel}(A_1) = \text{rel}(A)$  is  $\leq$ -minimal and by definition every predicate symbol in  $NR$  is strict  $\leq$ -smaller than  $\text{rel}(A)$ .)

2)  $D_1$  is failed.

Then  $D$  is failed too, which contradicts the assumption that  $D$  is infinite.

3)  $D_1$  is infinite.

By definition, every predicate symbol in  $NR$  is strict  $\leq$ -smaller than  $\text{rel}(A_1)$ , which is  $\leq$ -smaller than the  $\leq$ -largest predicate symbol in  $G_i$  (hence in  $G_0$ ), so we may assume that Lemma 4.8 holds for  $D = D_1$ . Now it follows that the Generalization Theorem can be applied on  $D = D_1$ . Hence  $D_1$  should be pruned by  $L$ . However, this contradicts the assumption (\*), for  $G = NR\theta$  and  $H = (R, A_2, \dots, A_n)\theta$ .

4)  $D_1$  is successful, yielding a computed answer substitution  $\sigma$  (if  $NR$  is empty then  $\sigma = \epsilon$ ).

This is the only remaining case. In this case we have in  $D$  the goal  $G_{i+1} = \leftarrow (R, A_2, \dots, A_n)\theta\sigma$ , immediately after  $NR$  is completely resolved.

**CLAIM 2.** The sequence of resolution steps between  $G_i$  and  $G_{i+1}$  in  $D$  can be mimicked by one resolution step  $G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1}$  in  $D'$ , where  $C_{i+1}$  is an instance of  $A \leftarrow R$  and  $\theta\sigma_{\text{lvar}(G_0, G_1, \dots, G_i)} = \theta_{i+1} \text{lvar}(G_0, G_1, \dots, G_i)$ .

Claim 1 follows from Claim 2: since  $\text{Pr}$  is closed under instantiation,  $C_{i+1}$  satisfies  $\text{Pr}$ . So we have constructed  $D'$  and proved Claim 1 up to the goal  $G_{i+1}$ .

Now the construction of the resolution step  $G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1}$  remains.

**PROOF OF CLAIM 2.** First, we define  $C_{i+1}$  and  $\theta_{i+1}$ , then we prove that  $G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1}$  is indeed a derivation step. Finally, we check the other requirements on  $C_{i+1}$  and  $\theta_{i+1}$ . By Lemma 3.5 we may assume that  $D$  is normal.

For every chain  $C$  in  $NR$ , we fix a substitution  $\rho_C$  such that for every  $x \in C\theta$ ,  $x\rho_C \in C$  and  $x\rho_C\theta = x$ . Moreover, if  $x \in (\text{var}(R) \cap C)\theta$ , then  $x\rho_C \in \text{var}(R)$ . For every chain, such a substitution exists: if  $x \in C\theta$ , then  $\{y \in C \mid y\theta = x\} \neq \emptyset$ . If  $\{y \in \text{var}(R) \cap C \mid y\theta = x\} \neq \emptyset$ , then  $x\rho_C$  must be chosen from the latter set, otherwise any element of the former set will do.

Now we can define  $\psi$  by: 
$$x\psi = \begin{cases} x & \text{if } x \notin \text{var}(NR) \\ x\theta\sigma\rho_{C(x)} & \text{if } x \in C \subseteq \text{var}(NR) \end{cases}.$$

(Note that  $x\theta\sigma \in C(x)\theta\sigma \subseteq C(x)\theta$  by Corollary 3.9, since  $D$  is normal.)

Finally, we define  $C_{i+1} = (A \leftarrow R)\psi$  and  $\theta_{i+1} = \theta\sigma_{\text{lvar}(A_1, A\psi)}$ .

Now we must prove that  $G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1}$  is indeed a resolution step.  
That is:

CLAIM 3.  $(A \leftarrow R)\psi$  is properly standardized apart.

CLAIM 4.  $\theta_{i+1}$  is an idempotent mgu of  $A\psi$  and  $A_1$ .

CLAIM 5.  $(R\psi, A_2, \dots, A_n)\theta_{i+1} = (R, A_2, \dots, A_n)\theta\sigma$ .

In the proofs of these claims, we take  $C(x) = C_{NR}(x)$ .

PROOF OF CLAIM 3. We prove that  $\text{var}((A \leftarrow R)\psi) \subseteq \text{var}(A \leftarrow NR, R)$ .

Let  $x \in \text{var}(A \leftarrow R)$ . Then:

if  $x\psi = x$ , then  $x\psi \in \text{var}(A \leftarrow R)$ ;

if  $x\psi \neq x$ , then  $x \in C(x) \subseteq \text{var}(NR)$ , so  $x\psi = x\theta\sigma\rho_{C(x)} \in C(x) \subseteq \text{var}(NR)$ .

Before proving Claim 4, we prove an additional claim.

CLAIM 6.  $\psi$  is idempotent.

PROOF. Let  $x$  be a variable. If  $x\psi = x$ , then  $x\psi\psi = x\psi$ .

Otherwise,  $x\psi\psi = x\theta\sigma\rho_{C(x)}\psi = ( \text{ since } x\theta\sigma\rho_{C(x)} \in C(x) \subseteq \text{var}(NR) ) = x\theta\sigma\rho_{C(x)}\theta\sigma\rho_{C(x)} = x\theta\sigma\sigma\rho_{C(x)} = ( \text{ as } \sigma \text{ is idempotent} ) = x\theta\sigma\rho_{C(x)} = x\psi$ .

PROOF OF CLAIM 4. We prove that for every unifier  $\eta$  of  $A_1$  and  $A\psi$ :  $\eta = \theta_{i+1}\eta$ . Let  $\eta$  be a unifier of  $A_1$  and  $A\psi$ :  $A_1\eta = A\psi\eta$ .

By standardizing apart,  $\text{var}(A_1) \cap \text{var}(NR) = \emptyset$ , so we have  $A_1 = A_1\psi$ . Therefore,  $\psi\eta$  is a unifier of  $A_1$  and  $A$ . Since  $\theta$  is an idempotent mgu of  $A_1$  and  $A$ , we have  $\psi\eta = \theta\omega = \theta\theta\omega = \theta\psi\eta$  (  $\theta \leq \psi\eta$ , so for some  $\omega$ :  $\theta\omega = \psi\eta$  ).

Let  $x$  be a variable. If  $x \notin \text{var}(A_1, A\psi)$ , then  $x = x\theta_{i+1}$ , so  $x\eta = x\theta_{i+1}\eta$ .

If  $x \in \text{var}(A_1)$ , then at the corresponding position in  $A$ , we find a term (constant or variable)  $t$  such that  $x\eta = t\psi\eta$  and  $x\theta = t\theta$ . Two cases arise.

-  $x\theta = x\theta\sigma$ . Then  $x\eta = t\psi\eta = t\theta\psi\eta = x\theta\psi\eta =^* x\theta\eta = x\theta\sigma\eta = x\theta_{i+1}\eta$ .

\* :  $x\theta \notin \text{var}(NR)$ , since either  $x\theta$  is ground, or  $x\theta \in \text{var}(A_1\theta) \subseteq \text{var}(A_1)$  (the latter inclusion by Corollary 3.11, since  $D$  is normal).

-  $x\theta \neq x\theta\sigma$ . Then  $x\theta \in \text{var}(NR\theta)$ , so for some  $v \in \text{var}(NR)$ :  $v\theta = x\theta$  and  $v\psi = v\theta\sigma\rho_{C(v)}$ . Now  $x\eta = t\psi\eta = t\theta\psi\eta = x\theta\psi\eta = v\theta\psi\eta = v\psi\eta = (\text{by Claim 6}) = v\psi\psi\eta = v\psi\theta\psi\eta = v\theta\sigma\rho_{C(v)}\theta\psi\eta = v\theta\sigma\psi\eta = x\theta\sigma\psi\eta =^* x\theta\sigma\eta = x\theta_{i+1}\eta$ .

\* :  $x\theta\sigma \notin \text{var}(\text{NR})$ , since either  $x\theta\sigma$  is ground, or  $x\theta\sigma \in \text{var}(A_1\theta\sigma) \subseteq \text{var}(A_1)$ . (the latter inclusion by Corollary 3.11, since  $D$  is normal).

If  $x \in \text{var}(A\psi)$ , then for some  $y \in \text{var}(A)$  we have  $y\psi = x$ . At the corresponding position in  $A_1$ , we find a term  $t$  such that  $x\eta = t\eta$  and  $y\theta = t\theta$ . Again two cases arise.

-  $y \notin \text{var}(\text{NR})$ . Then  $y\psi = y$  and  $y\theta\sigma = y\theta$ . Therefore we have  $x\eta = y\psi\eta = y\theta\psi\eta =^* y\theta\eta = y\theta\sigma\eta = y\psi\theta\sigma\eta = x\theta\sigma\eta = x\theta_{i+1}\eta$ .

\* :  $y\theta \notin \text{var}(\text{NR})$ , since either  $y\theta$  is ground, or  $y\theta \in \text{var}(A\theta) = \text{var}(A_1\theta) \subseteq \text{var}(A_1)$ .

-  $y \in \text{var}(\text{NR})$ . Then  $y\psi = y\theta\sigma\rho_{C(y)}$ , so (see Claim 6),  $y\psi = y\psi\psi = y\psi\theta\sigma\rho_{C(y\psi)} = x\theta\sigma\rho_{C(x)}$ . Therefore we have  $x\eta = y\psi\eta =$  (by Claim 6)  $y\psi\psi\eta = y\psi\theta\psi\eta = x\theta\sigma\rho_{C(x)}\theta\psi\eta = x\theta\sigma\psi\eta =^* x\theta\sigma\eta = x\theta_{i+1}\eta$ . \* : again,  $x\theta\sigma \notin \text{var}(\text{NR})$ .

#### PROOF OF CLAIM 5.

If  $x \in \text{var}(A_i)$  ( $2 \leq i \leq n$ ), then

if  $x \notin \text{var}(A_1)$  then  $x\theta_{i+1} = x = x\theta\sigma$ ;

if  $x \in \text{var}(A_1)$  then by definition  $x\theta_{i+1} = x\theta\sigma$ .

If  $x \in \text{var}(R)$ , then two cases arise.

-  $x\psi \in \text{var}(A\psi)$ . Then  $x\psi\theta_{i+1} = x\psi\theta\sigma =$

(if  $x \notin \text{var}(\text{NR})$ ) :  $x\theta\sigma$ .

(if  $x \in \text{var}(\text{NR})$ ) :  $x\theta\sigma\rho_{C(x)}\theta\sigma = x\theta\sigma\sigma = x\theta\sigma$ .

-  $x\psi \notin \text{var}(A\psi)$ . Then either  $x\psi$  is ground or for no  $y \in \text{var}(A)$ :  $y\psi = x\psi$ .

If  $x\psi$  is ground, then  $x\theta\sigma$  is ground, so  $x\psi = x\theta\sigma\rho_{C(x)} = x\theta\sigma$ .

If for no  $y \in \text{var}(A)$ :  $y\psi = x\psi$ , then in particular,  $x \notin \text{var}(A)$ , so  $x\theta = x$ .

Now, if  $x \notin \text{var}(\text{NR})$ , then  $x\psi = x = x\theta = x\theta\sigma$ .

If  $x \in \text{var}(\text{NR})$ , then  $x\psi = x\theta\sigma\rho_{C(x)}$ . Also,  $x\theta\sigma \in C(x)\theta\sigma \subseteq C(x)\theta$  (by Corollary 3.9, since  $D$  is normal), so for some  $z \in C(x)$ :  $z\theta = x\theta\sigma$  ( and  $C(z) = C(x)$  ).

Then  $z\theta\sigma = x\theta\sigma\sigma = x\theta\sigma$ , so  $z\psi = x\psi$ . Hence  $z \notin \text{var}(A)$ , so  $z\theta = z = z\rho_{C(z)}\theta$ , so  $z = z\rho_{C(z)} = z\rho_{C(x)}$ . Therefore  $x\theta\sigma\rho_{C(x)} = z\theta\rho_{C(x)} = z\rho_{C(x)} = z = z\theta = x\theta\sigma$ .

Obviously,  $C_{i+1}$  is an instance of  $A \leftarrow R$ . Also,  $\theta_{i+1}|_{\text{var}(G_0, G_1, \dots, G_i)} = \theta\sigma|_{\text{var}(A_1, A\psi) \cap \text{var}(G_0, G_1, \dots, G_i)} = \theta\sigma|_{\text{var}(A_1)} = \theta\sigma|_{\text{var}(G_0, G_1, \dots, G_i)}$ , by Corollary 3.11, since  $D$  is normal and a local selection rule is used. This concludes the proof of Claim 2 and thereby the proof of Lemma 4.8.  $\square$

## 5. Applications of the Generalization Theorem

A simplest example of the application of the Generalization Theorem is the following.

**COROLLARY 5.1.** *If  $P$  is a hierarchical program, then every SLD-derivation of  $P \cup \{G\}$  via the leftmost selection rule is finite.*

PROOF. We prove an equivalent proposition, namely that the empty loop check is complete w.r.t. the leftmost selection rule for hierarchical programs. This follows from the Generalization Theorem and the following observations.

- The empty loop check is complete for ‘unit-programs’, programs that consist solely of unit clauses.
- The ‘unit’ property is closed under instantiation.
- The empty loop check is safe for goal extension, initialization and detailing.
- Nr-extended unit-programs are known as hierarchical programs. □

Of course, this result is well known, even for arbitrary selection rules. More interesting results can be obtained by using the Generalization Theorem to extend the completeness results presented in section 2. The first result presented there is the completeness of equality checks (and subsumption checks) for function-free restricted programs w.r.t. the leftmost selection rule. The Generalization Theorem cannot be applied on this proposition. In contrast, the Generalization Theorem provides an easier proof of this proposition than the one given in [ABK], based on the lemma ‘the equality checks are complete for function-free programs in which the body of each clause contains at most one atom’.

The other results of section 2 are only valid for the subsumption checks. Therefore we shall now prove that the weakest of the subsumption checks, the  $SVR_L$  check, satisfies the conditions of the Generalization Theorem, i.e. that it is safe for goal extension, initialization and detailing.

**LEMMA 5.2.** *The  $SVR_L$  check is safe for goal extension.*

PROOF. Let  $D$  be an SLD-derivation of  $P \cup \{G_0\}$ . Let  $D'$  be an SLD-derivation of  $P \cup \{(G_0, H_0)\}$ , in which the same atoms are selected and the same input clauses and mgu's are used as in  $D$ . Denote by  $\theta_n$  the mgu used in the  $n$ -th resolution step of  $D$  and  $D'$  ( $n \geq 1$ ). If  $D$  is pruned, then we have for some

renaming  $\tau$  two goals  $G_i$  and  $G_k$  in  $D$  with  $G_k \supseteq_L G_i\tau$  and  $G_0\theta_1\ldots\theta_k = G_0\theta_1\ldots\theta_i\tau$ . Since  $D'$  uses the same input clauses and mgu's as  $D$ ,  $D$  cannot contain any variable occurring in  $H_0$  but not in  $G_0$ . Assuming that  $\tau$  acts only on the variables in  $D$ , we have that  $\theta_1\ldots\theta_k$  and  $\theta_1\ldots\theta_i\tau$  coincide on *all* variables of  $H_0$ . So  $(G_0, H_0)\theta_1\ldots\theta_k = (G_0, H_0)\theta_1\ldots\theta_i\tau$  and  $(G_k, H_0\theta_1\ldots\theta_k) \supseteq_L (G_i, H_0\theta_1\ldots\theta_i)\tau$ . This means that  $D'$  is pruned by  $SVR_L$ , as well.  $\square$

**LEMMA 5.3.** *The  $SVR_L$  check is safe for initialization.*

PROOF. Let  $D' = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow_{C_2, \theta_2} G_2 \Rightarrow \dots)$  be an SLD-derivation. Suppose that the derivation  $D = (G_i \Rightarrow_{C_{i+1}, \theta_{i+1}} G_{i+1} \Rightarrow_{C_{i+2}, \theta_{i+2}} G_{i+2} \Rightarrow \dots)$  is pruned by  $SVR_L$ , for some  $i > 0$ . Clearly for some  $j, k > j$  and renaming  $\tau$  (acting only on variables in  $D$ ) we have  $G_k \supseteq_L G_j\tau$  and  $G_i\theta_{i+1}\ldots\theta_k = G_i\theta_{i+1}\ldots\theta_j\tau$ . So it remains to prove that  $G_0\theta_1\ldots\theta_k = G_0\theta_1\ldots\theta_j\tau$ .

Let  $x \in \text{var}(G_0\theta_1\ldots\theta_i)$ . Two cases arise.

- i)  $x \notin \text{var}(G_i)$ . Then  $x$  does not occur in  $D$ , hence  $x\theta_{i+1}\ldots\theta_k = x\theta_{i+1}\ldots\theta_j\tau = x$ .
- ii)  $x \in \text{var}(G_i)$ . Then we have directly that  $x\theta_{i+1}\ldots\theta_k = x\theta_{i+1}\ldots\theta_j\tau$ .

Hence  $D'$  is pruned by  $SVR_L$  as well.  $\square$

**LEMMA 5.4.** *The  $SVR_L$  check is safe for detailing.*

PROOF. Let  $D = (G_0 \Rightarrow_{C_1, \theta_1} G_1 \Rightarrow \dots)$  be an SLD-derivation that is pruned by  $L$  and let  $D'$  be an SLD-derivation of the form  $(G_0 \Rightarrow_{C_1^1, \tau_1^1} H_1^1 \Rightarrow \dots \Rightarrow H_{n_1-1}^1 \Rightarrow_{C_{n_1}^1, \tau_{n_1}^1} G_1 \Rightarrow_{C_1^2, \tau_1^2} H_1^2 \Rightarrow \dots \Rightarrow H_{n_2-1}^2 \Rightarrow_{C_{n_2}^2, \tau_{n_2}^2} G_2 \Rightarrow \dots)$  with for every  $i \geq 0$ :  $\tau_1^i \dots \tau_{n_i}^i \text{lvar}(G_0, G_1, \dots, G_i) = \theta_i \text{lvar}(G_0, G_1, \dots, G_i)$  in which in  $G_0, G_1, \dots$

the same atoms are selected as in  $D$ . Since  $D$  is pruned by  $L$ , we have for some  $j, k > j$  and renaming  $\tau$ ,  $G_k \supseteq_L G_j\tau$  and  $G_0\theta_1\ldots\theta_k = G_0\theta_1\ldots\theta_j\tau$ . Therefore we also have  $G_0\tau_1^1 \dots \tau_{n_1}^1 \tau_1^2 \dots \tau_{n_2}^2 \dots \tau_1^k \dots \tau_{n_k}^k = G_0\tau_1^1 \dots \tau_{n_1}^1 \tau_1^2 \dots \tau_{n_2}^2 \dots \tau_1^j \dots \tau_{n_j}^j \tau$ , hence

$D'$  is pruned by  $SVR_L$  as well.  $\square$

Now we can use the Generalization Theorem together with the fact that the subsumption checks are complete for function-free nvi programs.

**COROLLARY 5.5.** *The subsumption checks are complete w.r.t. the leftmost selection rule for function-free nr-extended nvi programs.*

PROOF. The nvi property is obviously closed under instantiation. Therefore by Subsumption Completeness Theorem 2.21, the Generalization Theorem, and the Lemma's 5.2, 5.3 and 5.4, the  $SVR_L$  loop check is complete w.r.t. the leftmost selection rule for function-free nr-extended nvi programs. Since the  $SVR_L$  check is the weakest of the subsumption checks, by the Relative Strength Theorem 2.7, the same holds for the other subsumption checks.  $\square$

Finally, in section 2 it was mentioned that the subsumption checks are also complete for function-free svo programs. However, the property 'svo' is *not* closed under instantiation, so we cannot immediately use the Generalization Theorem. In fact, this should not come as a surprise, since *every* program can be converted into a 'computationally equivalent' nr-extended svo program. This can be done by replacing the  $k > 1$  occurrences of a variable  $x$  in the body of a clause by  $x_1, \dots, x_k$  and adding the non-recursive atoms  $eq(x, x_1), \dots, eq(x, x_k)$  in the body of the clause. Finally the clause  $eq(x, x)$  is added to the program (assuming that  $eq$  is a new predicate symbol in  $P$ ).

In the proof of Lemma 4.8, we need that the clause  $C_{i+1} = (A \leftarrow R)\psi$  satisfies the property of clauses considered, given that the clause  $A \leftarrow R$  satisfies the property. Up till now, this was derived immediately from the assumption that the property should be closed under instantiation. Since for the svo property this is not true, we shall derive conditions that ensure directly that  $C_{i+1}$  satisfies the svo property, i.e. that every variable in  $R\psi$  occurs only once (provided that every variable in  $R$  occurs only once).

Formally, let  $x, y \in \text{var}(R)$  such that  $x \neq y$  and  $x\psi, y\psi \in \text{VAR}$ . We shall derive conditions on the program ensuring that  $x\psi \neq y\psi$ .

If  $x \notin \text{var}(NR)$ , then  $x\psi = x$ .

Then, if  $y \notin \text{var}(NR)$ ,  $y\psi = y \neq x$ , and

if  $y \in \text{var}(NR)$ ,  $y\psi = y\theta\sigma\rho_{C(y)} \in C(y) \subseteq \text{var}(NR)$ , so  $y\psi \neq x$ .

The same argument holds if  $y \notin \text{var}(NR)$ . So a problem can only arise in the case that  $x, y \in \text{var}(NR)$ . Then we have  $x\psi = x\theta\sigma\rho_{C(x)} \in C(x)$  and  $y\psi = y\theta\sigma\rho_{C(y)} \in C(y)$ .

One solution is demanding that for every pair of distinct variables  $x, y \in \text{var}(R) \cap \text{var}(NR)$ ,  $C(x) \neq C(y)$ . Then  $C(x) \cap C(y) = \emptyset$ , so  $x\psi \neq y\psi$ . This disallows the addition of the eq-atoms in the construction above.

Another solution is to avoid that different variables in a (sub)goal are unified while the (sub)goal is refuted. (That is: to ensure that for every variable  $x$  in a goal, and for every unifier  $\sigma$  in the derivation, either  $x\sigma = x$  or  $x\sigma$  is a constant.) This condition can be met (for normal derivations) by the demand that variables do not occur more than once in the *head* of a clause. This disallows the addition of the clause  $\text{eq}(x, x) \leftarrow$ .

In this case such a condition yields  $x\psi = x\theta\rho_{C(x)}$  ( $x\theta\sigma$  cannot be a constant, since  $x\psi$  is a variable). Then  $x\theta = x\theta\rho_{C(x)}\theta = x\psi\theta$ . Using the condition again (but now w.r.t.  $\theta$ ), we obtain  $x = x\psi$  (still,  $x\theta$  cannot be a constant). Similarly we obtain  $y = y\psi$ , so  $x\psi \neq y\psi$ .

These two solutions give rise to two classes of programs for which the subsumption checks are complete w.r.t. the leftmost selection rule (in the absence of function symbols).

#### DEFINITION 5.6.

Let  $P$  be a program. A clause  $C = (A \leftarrow NR, R)$  is *chain-restricted svo w.r.t.  $P$*  if  $C$  is nr-extended svo w.r.t.  $P$ , where  $NR$  is the non-recursive part and  $R$  is the svo-part of  $C$ , and for every pair of distinct variables  $x, y \in \text{var}(R)$ ,  $C_{NR}(x) \neq C_{NR}(y)$ . A program  $P$  is *chain-restricted svo* if every clause in  $P$  is chain-restricted svo w.r.t.  $P$ .  $\square$

#### DEFINITION 5.7.

Let  $P$  be a program. A clause  $C$  is *head-restricted svo w.r.t.  $P$*  if  $C$  is nr-extended svo w.r.t.  $P$  and in the head of  $C$ , no variable occurs more than once. A program  $P$  is *head-restricted svo* if every clause in  $P$  is head-restricted svo w.r.t.  $P$ .  $\square$

**COROLLARY 5.8.** *The subsumption checks are complete w.r.t. the leftmost selection rule for function-free chain-restricted svo programs.*

**PROOF.** By Subsumption Completeness Theorem 2.22, the Generalization Theorem, the Lemma's 5.2, 5.3 and 5.4 and the considerations above, the  $\text{SVRL}$  loop check is complete w.r.t. the leftmost selection rule for function-free

chain-restricted svo programs. Since the  $SVR_L$  check is the weakest of the subsumption checks, by the Relative Strength Theorem 2.7, the same holds for the other subsumption checks.  $\square$

**COROLLARY 5.9.** *The subsumption checks are complete w.r.t. the leftmost selection rule for function-free head-restricted svo programs.*

PROOF. By Subsumption Completeness Theorem 2.22, the Generalization Theorem, the Lemma's 5.2, 5.3 and 5.4 and the considerations above, the  $SVR_L$  loop check is complete w.r.t. the leftmost selection rule for function-free head-restricted svo programs. Since the  $SVR_L$  check is the weakest of the subsumption checks, by the Relative Strength Theorem 2.7, the same holds for the other subsumption checks.  $\square$

## References

- [ABK] K.R. APT, R.N. BOL and J.W. KLOP, *On the Safe Termination of PROLOG Programs*, in: Proceedings of the Sixth International Conference on Logic Programming, (G. Levi and M. Martelli eds.), MIT Press, Cambridge Massachusetts, 1989, 353-368.
- [B] Ph. BESNARD, *On Infinite Loops in Logic Programming*, Internal Report 488, IRISA, Rennes, 1989.
- [BAK] R.N. BOL, K.R. APT and J.W. KLOP, *An Analysis of Loop Checking Mechanisms for Logic Programs*, Technical Report CS-R8942, Centre of Mathematics and Computer Science, Amsterdam; Technical Report TR-89-32, University of Texas at Austin, 1989. To appear in Theoretical Computer Science.
- [BW] D.R. BROUGH and A. WALKER, *Some Practical Properties of Logic Programming Interpreters*, in: Proceedings of the International Conference on Fifth Generation Computer Systems, (ICOT eds), 1984, 149-156.
- [Co] M.A. COVINGTON, *Eliminating Unwanted Loops in PROLOG*, SIGPLAN Notices, Vol. 20, No. 1, 1985, 20-26.
- [vG] A. VAN GELDER, *Efficient Loop Detection in PROLOG using the Tortoise-and-Hare Technique*, J. Logic Programming 4, 1987, 23-31.

- [KT] D.B. KEMP and R.W. TOPOR, *Completeness of a Top-Down Query Evaluation Procedure for Stratified Databases*, in: Proceedings of the Fifth International Conference on Logic Programming, MIT Press, Cambridge Massachusetts, 1988, 178-194.
- [L] J.W. LLOYD, *Foundations of Logic Programming*, Second Edition, Springer-Verlag, Berlin, 1987.
- [LS] J.W. LLOYD and J.C. SHEPHERDSON, *Partial Evaluation in Logic Programming*, Technical Report CS-87-09, Dept. of Computer Science, University of Bristol, 1987.
- [MM] A. MARTELLI and U. MONTANARI, *An Efficient Unification Algorithm*, in: ACM Transactions on Programming Languages and Systems, Vol. 4, No. 1, 1982, 258-282.
- [N] J.F. NAUGHTON, *One-Sided Recursion*, in: Proceedings of the Sixth ACM Symposium on Principles of Database Systems, ACM New York, 1987, pp. 340-348.
- [PG] D. POOLE and R. GOEBEL, *On Eliminating Loops in PROLOG*, SIGPLAN Notices, Vol. 20, No. 8, 1985, 38-40.
- [SGG] D.E. SMITH, M.R. GENESERETH and M.L. GINSBERG, *Controlling Recursive Inference*, Artificial Intelligence 30, 1986, 343-389.
- [SI] H. SEKI and H. ITOH, *A Query Evaluation Method for Stratified Programs under the Extended CWA*, in: Proceedings of the Fifth International Conference on Logic Programming, MIT Press, Cambridge Massachusetts, 1988, 195-211.
- [ŠŠ] O. ŠTĚPÁNKOVÁ and P. ŠTĚPÁNEK, *A Complete Class of Restricted Logic Programs*, in: Logic Colloquium '86, (F.R. Drake and J.K. Truss eds.), North Holland, Amsterdam, 1988, 319-324.
- [V] L. VIEILLE, *Recursive Query Processing: The Power of Logic*, Theoretical Computer Science 69, No. 1, 1989, 1-53.

