Why the Occur-check is Not a Problem

Krzysztof R. Apt
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
and
Faculty of Mathematics and Computer Science, University of Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands

Alessandro Pellegrini
Dipartimento di Matematica Pura ed Applicata
Università di Padova, Via Belzoni 7, 35131 Padova, Italy

Abstract

In most Prolog implementations for the efficiency reasons so-called occur-check is omitted from the unification algorithm. We provide here natural syntactic conditions which allow the occur-check to be safely omitted. The established results apply to most well-known Prolog programs and seem to explain why this omission does not lead in practice to any complications.

Note. This research was done during the second author's stay at Centre for Mathematics and Computer Science, Amsterdam. His stay was supported by the 2060^{th} District of the Rotary Foundation, Italy.

1 Introduction

The occur-check is a special test used in the unification algorithm. In most Prolog implementations it is omitted for the efficiency reasons. This omission affects the unification algorithm and introduces a possibility of divergence or may yield incorrect results. This is obviously an undesired situation. This problem was studied in the literature under the name of the *occur-check problem* (see e.g. Plaisted [Pla84] and Deransart and Maluszynski [DM85b]).

The aim of this paper is to provide easy to check syntactic conditions which ensure that the occur-check can be safely omitted. We use here a recent result of Deransart, Ferrand and Téguia [DFT91] and build upon it within the context of moded programs. This allows us to extend the results of Deransart and Maluszynski [DM85b], to simplify the arguments of Chadha and Plaisted [CP91] and to offer a uniform presentation. Additionally, the results of the former paper needed here are proved directly, without resorting to the techniques of the attribute grammars theory, and the results of the latter paper are supplied with a needed justification. The established results apply to most well-known Prolog programs. In fact, we found in the book of Sterling and Shapiro [SS86] only two (sic!) programs to which these results cannot be directly applied.

In what follows we study logic programs executed by means of the *LD-resolution*, which consists of the SLD-resolution combined with the leftmost selection rule. An SLD-derivation in which the leftmost selection rule is used is called an *LD-derivation*. We allow in programs

various first-order built-in's, like $=$, $\neq$, $>$, etc, and assume that they are resolved in the way conforming to their interpretation.

Throughout the paper we use the standard notation of Lloyd [Llo87] and Apt [Apt90]. In particular, given a syntactic construct E (so for example, a term, an atom or a set of equations) we denote by $Var(E)$ the set of the variables appearing in E. Given a substitution $\theta = \{x_1/t_1, ..., x_n/t_n\}$ we denote by $Dom(\theta)$ the set of variables $\{x_1, \ldots, x_n\}$, by $Range(\theta)$ the set of terms $\{t_1, \ldots, t_n\}$, and by $Ran(\theta)$ the set of variables appearing in $\{t_1, \ldots, t_n\}$. Finally, we define $Var(\theta) = Dom(\theta) \cup Ran(\theta)$.

Recall that a substitution θ is called *grounding* if $Ran(\theta)$ is empty, and is called a *renaming* if it is a permutation of the variables in $Dom(\theta)$. Given a substitution θ and a set of variables V, we denote by $\theta|V$ the substitution obtained from θ by restricting its domain to V.

2 Occur-check Free Programs

We start our considerations by recalling a unification algorithm due to Martelli and Montanari [MM82]. We use below the notions of sets and of systems of equations interchangingly. Two atoms can unify only if they have the same relation symbol. With two atoms $p(s_1, ..., s_n)$ and $p(t_1, ..., t_n)$ to be unified we associate the set of equations $\{s_1 = t_1, ..., s_n = t_n\}$. In the applications we often refer to this set as $p(s_1, ..., s_n) = p(t_1, ..., t_n)$. The algorithm operates on such finite sets of equations. A substitution θ such that $s_1\theta = t_1\theta, \ldots, s_n\theta = t_n\theta$ is called a *unifier* of the set of equations $\{s_1 = t_1, ..., s_n = t_n\}$. Thus the set of equations $E = \{s_1 = t_1, ..., s_n = t_n\}$ has the same unifiers as the atoms $p(s_1, ..., s_n)$ and $p(t_1, ..., t_n)$.

A unifier θ of a set of equations E is called a *most general unifier* (in short *mgu*) of E if it is more general than all unifiers of E. An mgu θ of a set of equations E is called *relevant* if $Var(\theta) \subseteq Var(E)$.

A set of equations is called *solved* if it is of the form $\{x_1 = t_1, ..., x_n = t_n\}$ where the x_i's are distinct variables and none of them occurs in a term t_j.

The following unification algorithm will be used in the sequel.

MARTELLI-MONTANARI ALGORITHM

Nondeterministically choose from the set of equations an equation of a form below and perform the associated action.

(1) $f(s_1, ..., s_n) = f(t_1, ..., t_n)$	*replace by the equations* $s_1 = t_1, ..., s_n = t_n$,
(2) $f(s_1, ..., s_n) = g(t_1, ..., t_m)$ where $f \not\equiv g$	*halt with failure,*
(3) $x = x$	*delete the equation,*
(4) $t = x$ where t is not a variable	*replace by the equation* $x = t$,
(5) $x = t$ where $x \not\equiv t$, x does not occur in t and x occurs elsewhere	*perform the substitution* $\{x/t\}$ *in every other equation,*
(6) $x = t$ where $x \not\equiv t$ and x occurs in t	*halt with failure.*

The algorithm terminates when no action can be performed or when failure arises. To keep the formulation of the algorithm concise we identified here constants with 0-ary functions. Thus action (2) includes the case of two different constants.

The following theorem holds (see Martelli and Montanari [MM82]).

Theorem 2.1 (Unification) *The Martelli-Montanari algorithm always terminates. If the original set of equations E has a unifier, then the algorithm successfully terminates and produces a solved set of equations determining a relevant mgu of E, and otherwise it terminates with failure.* □

The Martelli-Montanari algorithm does not generate all mgu's of a set of equations E but the following lemma, proved in Lassez, Marriot and Maher [LMM88], will allow us to cope with this peculiarity.

Lemma 2.2 *Let θ_1 and θ_2 be mgu's of a set of equations. Then for some renaming η we have $\theta_2 = \theta_1\eta$.* □

The test "x does not occur in t" in action (5) of the Martelli-Montanari algorithm is called the *occur-check*. In most Prolog implementations the occur-check is omitted. By omitting the occur-check in (5) and deleting action (6) from the Martelli-Montanari algorithm we are still left with two options depending on whether the substitution $\{x/t\}$ is performed in t itself. If it is, then divergence can result, because x occurs in t implies that x occurs in $t\{x/t\}$. If it is not (as in the case of the modified version of the algorithm just mentioned), then an incorrect result can be produced, as in the case of the single equation $x = f(x)$ which yields the substitution $\{x/f(x)\}$.

None of these alternatives is desirable. It is natural then to seek conditions which guarantee that, in absence of the occur-check, in all Prolog evaluations of a given goal w.r.t. a given program unification is correctly performed. This leads us to the following notion due to Deransart, Ferrand and Téguia [DFT91].

Definition 2.3 A set of equations E is called *not subject to occur-check* (NSTO in short) if in no execution of the Martelli-Montanari algorithm started with E action (6) can be performed. □

We now introduce the key definition of the paper.

Definition 2.4

- Let ξ be an LD-derivation. Let A be an atom selected in ξ and H the head of the input clause selected to resolve A in ξ. Suppose that A and H have the same relation symbol. Then we say that the system $A = H$ *is considered in* ξ.
- Suppose that all systems of equations considered in the LD-derivations of $P \cup \{G\}$ are NSTO. Then we say that $P \cup \{G\}$ is *occur-check free.* □

This definition assumes a specific unification algorithm but allows us to derive precise results. In contrast, no specific unification algorithm in the definition of the LD-resolution is assumed.

By Theorem 2.1 if a considered system of equations is unifiable, then it is NSTO, as well. Thus the property of being occur-check free rests exclusively upon those considered systems which are not unifiable. As in the definition of the occur-check freedom *all* LD-derivations of $P \cup \{G\}$ are considered, all systems of equations that can be considered in a possibly backtracking Prolog evaluation of a goal G w.r.t. the program P are taken into account.

In Deransart, Ferrand and Téguia [DFT91] a related concept of an NSTO program is studied which essentially states that, independently of the selection rule and the resolution strategy

chosen, all considered systems are NSTO. The definition of the occur-check freedom refers to the leftmost selection rule, so the results we obtain are usually incompatible with those dealing with NSTO programs.

The aim of this paper is to offer simple syntactic conditions which imply that $P \cup \{G\}$ is occur-check free. It is useful to note the following.

Lemma 2.5 *The problem whether a set of equations is NSTO, is decidable.* □

Lemma 2.5 provides a method to determine whether a given set of equations is NSTO. However, it is not easy to apply it. Instead, we shall use a result due to Deransart, Ferrand and Téguia [DFT91]. We need some preparatory definitions first.

Definition 2.6

- We call a family of terms (resp. an atom) *linear* if every variable occurs at most once in it.
- We call a set of equations *left linear* (resp. *right linear*) if the family of terms formed by their left-hand (resp. right-hand) sides is linear. □

Thus a family of terms is linear iff no variable has two distinct occurrences in any term and no two terms have a variable in common.

Definition 2.7 Let E be a set of equations. We denote by $\rightarrow_E$ the following relation defined on the elements of E:
$e_1 \rightarrow_E e_2$ iff the left-hand side of e_1 and the right-hand side of e_2 have a variable in common. □

In particular, if a variable occurs both in the left-hand and right-hand side of an equation e of E, then $e \rightarrow_E e$.

We can now state the result proved by Deransart, Ferrand and Téguia [DFT91].

Lemma 2.8 (NSTO) *Suppose that the equations in E can be oriented in such a way that the resulting system F is left linear and the relation $\rightarrow_F$ is cycle-free. Then E is NSTO.* □

The original formulation of this lemma is slightly stronger, but for our purposes the above version is sufficient.

3 Moded Programs

For a further analysis we introduce modes.

Definition 3.1 Consider an n-ary relation symbol p. By a *mode* for p we mean a function d_p from $\{1, \ldots, n\}$ to the set $\{+, -\}$. If $d_p(i) = '+'$, we call i an *input position* of p and if $d_p(i) = '-'$, we call i an *output position* of p (both w.r.t. d_p).

We write d_p in a more suggestive form $p(d_p(1), \ldots, d_p(n))$. □

Modes indicate how the arguments of a relation should be used. This definition assumes one mode per relation in a program. Multiple modes may be obtained by simply renaming the relations. From now on we assume that *every considered relation* has a mode associated with it. This will allow us to talk about input positions and output positions of an atom. Throughout the paper, given an atom A, we denote by $VarIn(A)$ (resp. $VarOut(A)$) the set of variables occurring in the input (resp. output) positions of A. Similar notation is used for sequences of atoms.

We now introduce the following concepts.

Definition 3.2

- An atom is called *input* (resp. *output*) *linear* if the family of terms occurring in its input (resp. output) positions is linear.
- An atom is called *input-output disjoint* if the family of terms occurring in its input positions has no variable in common with the family of terms occurring in its output positions. □

The following lemma is crucial.

Lemma 3.3 (NSTO via Modes) *Consider two atoms A and H with the same relation symbol. Suppose that*

- *they have no variable in common,*
- *one of them is input-output disjoint,*
- *one of them is input linear and the other is output linear.*

Then $A = H$ is NSTO.

Proof. Suppose first that A is input-output disjoint and input linear and H is output linear. Let $i_1^A, \ldots, i_m^A$ (resp. $i_1^H, \ldots, i_m^H$) be the terms filling in the input positions of A (resp. H) and $o_1^A, \ldots, o_n^A$ (resp. $o_1^H, \ldots, o_n^H$) the terms filling in the output positions of A (resp. H).

The system under consideration is

$$E = \{i_1^A = i_1^H, \ldots, i_m^A = i_m^H, o_1^A = o_1^H, \ldots, o_n^A = o_n^H\}.$$

Reorient it as follows:

$$F = \{i_1^A = i_1^H, \ldots, i_m^A = i_m^H, o_1^H = o_1^A, \ldots, o_n^H = o_n^A\}.$$

By assumption A and H have no variable in common. This implies that

- F is left-linear (because additionally A is input linear and H is output linear),
- the equations $i_j^A = i_j^H$ have no successor in the $\rightarrow_F$ relation and the equations $o_j^H = o_j^A$ have no predecessor (because additionally A is input-output disjoint).

Thus by the NSTO Lemma 2.8 $A = H$ is NSTO. The proofs for the remaining three cases are analogous and omitted. □

We now prove two results allowing us to conclude that $P \cup \{G\}$ is occur-check free. The first one uses the following notion due to Dembinski and Maluszynski [DM85a].

Definition 3.4 We call an LD-derivation *data driven* if all atoms selected in it are ground in their input positions. □

Theorem 3.5 *Suppose that*

- *the head of every clause of P is output linear,*
- *all LD-derivations of $P \cup \{G\}$ are data driven.*

Then $P \cup \{G\}$ is occur-check free.

Proof. By the NSTO via Modes Lemma 3.3. □

The second result uses the following notion.

Definition 3.6 We call an LD-derivation *output driven* if all atoms selected in it are output linear and input-output disjoint. □

Theorem 3.7 *Suppose that*

- *the head of every clause of P is input linear,*
- *all LD-derivations of $P \cup \{G\}$ are output driven.*

Then $P \cup \{G\}$ is occur-check free.

Proof. By the NSTO via Modes Lemma 3.3. □

This theorem is implicit in Chadha and Plaisted [CP91] (see the proof of their Theorem 2.2).

So far we isolated two properties of LD-derivations, each of which implies occur-check freedom. In both cases we had to impose some restrictions on the heads of the clauses. When we combine these two properties we get occur-check freedom directly.

Theorem 3.8 *Suppose that*

- *all LD-derivations of $P \cup \{G\}$ are both data and output driven.*

Then $P \cup \{G\}$ is occur-check free.

Proof. By the NSTO Lemma 2.8. □

4 Well-moded Programs

The obvious problem with Theorems 3.5, 3.7 and 3.8 is that is is not easy to check their conditions. In fact, one can show that in general it is undecidable whether for a given program P and goal G the conditions of Theorem 3.5, 3.7 or 3.8 hold.

The aim of this section is to propose some syntactic restrictions that imply the conditions of Theorems 3.5. We then show that these restrictions are satisfied by a number of well-known programs.

We use here the notion of a well-moded program. The concept is due to Dembinski and Maluszynski [DM85a]; we use here an elegant formulation due to Rosenblueth [Ros91] (which is equivalent to that of Drabent [Dra87] where well-moded programs are called simple). The definition of a well-moded program constrains the "flow of data" through the clauses of the programs. To simplify the notation, when writing an atom as $p(\mathbf{u}, \mathbf{v})$, we now assume that $\mathbf{u}$ is a sequence of terms filling in the input positions of p and that $\mathbf{v}$ is a sequence of terms filling in the output positions of p.

Definition 4.1

- A goal $\leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called *well-moded* if for $i \in [1, n]$

$$Var(\mathbf{s_i}) \subseteq \bigcup_{j=1}^{i-1} Var(\mathbf{t_j}).$$

- A clause $p_0(\mathbf{t_0}, \mathbf{s_{n+1}}) \leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called *well-moded* if for $i \in [1, n+1]$

$$Var(\mathbf{s_i}) \subseteq \bigcup_{j=0}^{i-1} Var(\mathbf{t_j}).$$

- A program is called *well-moded* if every clause of it is well-moded. □

Note that a goal with only one atom is well-moded iff this atom is ground in its input positions. The definition of a well-moded program is designed in such a way that the following theorem due to Dembinski and Maluszynski [DM85a] holds.

Theorem 4.2 *Let P and G be well-moded. Then all LD-derivations of $P \cup \{G\}$ are data driven.* □

This theorem brings us to the following conclusion.

Corollary 4.3 *Let P and G be well-moded. Suppose that*

- *the head of every clause of P is output linear.*

Then $P \cup \{G\}$ is occur-check free.

Proof. By Theorems 3.5 and 4.2. □

This corollary can be easily applied to a number of well-known Prolog programs.

Example 4.4 Below, when presenting the programs we adhere to the usual syntactic conventions of Prolog with the exception that Prolog's ":-" is replaced by the logic programming " ← ".
(i) Consider the program append:

```
app([X | Xs], Ys, [X | Zs]) ← app(Xs, Ys, Zs).
app([], Ys, Ys).
```

with the moding `app(+,+,-)`. It is easy to check that append is then well-moded and that the head of every clause is output linear. By Corollary 4.3 we conclude that for `s` and `t` ground, append ∪ { ← `app(s, t, u)`} is occur-check free.

(ii) Consider now the program append with the moding `app(-,-,+)`. Again, by Corollary 4.3, we conclude that for `u` ground, append ∪ { ← `app(s, t, u)`} is occur-check free.

(iii) Consider the program `permutation` which consists of the clauses

```
perm(Xs, [X | Ys]) ←
   app(X1s, [X | X2s], Xs),
   app(X1s, X2s, Zs),
   perm(Zs, Ys).
perm([], []).
```

augmented by the append program.

We use here the following modings: perm(+,-), app(-,-,+) for the first call to append and app(+,+,-) for the second call to append.

It is easy to check that permutation is then well-moded and that the heads of all clauses are output linear. By Corollary 4.3 we get that for s ground, permutation $\cup$ { $\leftarrow$ perm(s, t)} is occur-check free.

(iv) Consider now the program quicksort which consists of the clauses

```
qs([X | Xs], Ys) ←
   partition(X, Xs, Littles, Bigs),
   qs(Littles, Ls),
   qs(Bigs, Bs),
   app(Ls, [X | Bs], Ys).
qs([], []).

partition(X, [Y | Xs], [Y | Ls], Bs) ← X > Y, partition(X, Xs, Ls, Bs).
partition(X, [Y | Xs], Ls, [Y | Bs]) ← X ≤ Y, partition(X, Xs, Ls, Bs).
partition(X, [], [], []).
```

augmented by the append program.

We mode it as follows: qs(+,-), partition(+,+,-,-), app(+,+,-). Again, it is easy to check that quicksort is then well-moded and that the heads of all clauses are output linear. By Corollary 4.3 we conclude that for s ground, quicksort $\cup$ { $\leftarrow$ qs(s, t)} is occur-check free.

(v) Finally, consider the program palindrome:

```
palindrome(Xs) ← reverse(Xs, Xs).
reverse(X1s, X2s) ← reverse(X1s, [], X2s).
reverse([X | X1s], X2s, Ys) ← reverse(X1s, [X | X2s], Ys).
reverse([], Xs, Xs).
```

We mode it as follows: palindrome(+), reverse(+,-), reverse(+,+,-). Then palindrome is well-moded and the heads of all clauses are output linear. By Corollary 4.3 we conclude that for s ground, palindrome $\cup$ { $\leftarrow$ palindrome(s)} is occur-check free. □

5 Nicely Moded Programs

The above conclusions are still of a restrictive kind, because in each case we had to assume that the input positions of the one atom goals are ground. To alleviate this restriction we now consider some syntactic restrictions that imply the conditions of Theorem 3.7.

The following notion was introduced in Chadha and Plaisted [CP91]. (We found essentially the same concept independently, though later; the name and formulation are ours.)

Definition 5.1

- A goal $\leftarrow p_1(s_1, t_1), \ldots, p_n(s_n, t_n)$ is called *nicely moded* if $t_1, \ldots, t_n$ is a linear family of terms and for $i \in [1, n]$

$$Var(s_i) \cap (\bigcup_{j=i}^{n} Var(t_j)) = \emptyset.$$

- A clause $p_0(\mathbf{s_0}, \mathbf{t_0}) \leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called *nicely moded* if $\leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is nicely moded and

$$Var(\mathbf{s_0}) \cap (\bigcup_{j=1}^{n} Var(\mathbf{t_j})) = \emptyset.$$

- A program is called *nicely moded* if every clause of it is nicely moded. □

Thus, assuming that in every atom the input positions occur first, a goal is nicely moded if every variable occurring in an output position of an atom does not occur earlier in the goal.

And a clause is nicely moded if every variable occurring in an output position of a body atom occurs neither earlier in the body nor in an input position of the head.

Note that a goal with only one atom is nicely moded iff it is output linear and input-output disjoint. The following theorem clarifies our interest in nicely moded programs.

Theorem 5.2 *Let P and G be nicely moded. Then all LD-derivations of $P \cup \{G\}$ are output driven.*

The proof is quite complicated and requires a number of lemmas. The first one allows us to search for mgu's in an iterative fashion.

Lemma 5.3 *Let E_1, E_2 be two sets of equations. Suppose that θ_1 is a relevant mgu of E_1 and θ_2 is a relevant mgu of $E_2\theta_1$. Then $\theta_1\theta_2$ is a relevant mgu of $E_1 \cup E_2$. Moreover, if $E_1 \cup E_2$ is unifiable then θ_1 exists and for any such θ_1 an appropriate θ_2 exists, as well.* □

Lemma 5.4 *Let θ be a substitution and* s *and* t *sequences of terms such that*

- $Var(\mathbf{s}) \cap Var(\mathbf{t}) = \emptyset$,
- $Ran(\theta | Var(\mathbf{s})) \cap Ran(\theta | Var(\mathbf{t})) = \emptyset$,
- $Var(\mathbf{s}) \cap Ran(\theta | Var(\mathbf{t})) = \emptyset$,
- $Var(\mathbf{t}) \cap Ran(\theta | Var(\mathbf{s})) = \emptyset$.

Then $Var(\mathbf{s}\theta) \cap Var(\mathbf{t}\theta) = \emptyset$. □

The next two lemmas use the following notion.

Definition 5.5 A substitution $\{x_1/t_1, \ldots, x_n/t_n\}$ is called *linear* if $t_1, \ldots, t_n$ is a linear family of terms. □

Lemma 5.6 *Let θ be a substitution and* t *a family of terms. Suppose that*

- *θ is linear,*
- t *is linear,*
- $Ran(\theta) \cap Var(\mathbf{t}) = \emptyset$.

Then $\mathbf{t}\theta$ is a linear family of terms, as well. □

The following lemma is stated in Deransart and Maluszynski [DM85b].

Lemma 5.7 *Consider two atoms A and H with the same relation symbol. Suppose that*

- *they have no variable in common,*
- *A is linear.*

Assume that A and H are unifiable. Then there exists a relevant mgu θ of A and H such that

- *$\theta|\,Var(H)$ is linear,*
- *$Ran(\theta|\,Var(H)) \subseteq Var(A)$.* □

Finally, we establish the following lemma.

Lemma 5.8 *Consider two atoms A and H with the same relation symbol. Suppose that*

- *they have no variable in common,*
- *A is input-output disjoint and output linear.*

Assume that A and H are unifiable. Then there exists a relevant mgu θ of A and H such that for $V = VarOut(H) - VarIn(H)$, $\eta_1 = \theta|V$ and $\eta_2 = \theta|\,VarIn(H)$

(i) *η_1 is linear,*

(ii) *$Ran(\eta_1) \subseteq Var(A)$,*

(iii) *$Ran(\eta_2) \cap (Ran(\eta_1) \cup V) = \emptyset$.*

Proof. Let $i_1^A, \ldots, i_m^A$ (resp. $i_1^H, \ldots, i_m^H$) be the terms filling in the input positions of A (resp. H) and $o_1^A, \ldots, o_n^A$ (resp. $o_1^H, \ldots, o_n^H$) the terms filling in the output positions of A (resp. H). Let θ_1 be the relevant mgu of $\{o_1^A = o_1^H, \ldots, o_n^A = o_n^A\}$ constructed in the proof of Lemma 5.7. By the disjointness of A and H we have $\theta_1|\,Var(H) = \theta_1|\,VarOut(H)$, so by Lemma 5.7

$$\theta_1|\,Var(H) \text{ is linear} \tag{1}$$

and

$$Ran(\theta_1|\,Var(H)) \subseteq VarOut(A). \tag{2}$$

Let θ_2 be a relevant mgu of $\{i_1^A = i_1^H, \ldots, i_m^A = i_m^H\}\theta_1$. By Lemma 5.3 θ_2 exists and $\theta = \theta_1\theta_2$ is a relevant mgu of $A = H$.

By the relevance of θ_1 we have $Dom(\theta_1) \subseteq VarOut(A) \cup VarOut(H)$, so by the input-output disjointness of A and the disjointness of A and H we get $\{i_1^A = i_1^H, \ldots, i_m^A = i_m^H\}\theta_1 = \{i_1^A = i_1^H\theta_1, \ldots, i_m^A = i_m^H\theta_1\}$.

By the relevance of θ_2 we have $Var(\theta_2) \subseteq Var(\{i_1^A = i_1^H\theta_1, \ldots, i_m^A = i_m^H\theta_1\}) \subseteq VarIn(A) \cup VarIn(H) \cup Ran(\theta_1|\,VarIn(H))$.

Thus, by the disjointness of A and H and (2),

$$Var(\theta_2) \cap V = \emptyset. \tag{3}$$

For the same reasons and additionally by the input-output disjointness of A and (1)

$$Var(\theta_2) \cap Ran(\theta_1|V) = \emptyset. \tag{4}$$

Now, (3) and (4) imply that

$$\eta_1 = \theta_1 | V. \tag{5}$$

Thus $\eta_1 \subseteq \theta_1 | \mathit{Var}(H)$, so by (1) we conclude (i) and by (2) we conclude (ii).

Consider now η_2. Note that $\eta_2 \subseteq (\theta_1 | \mathit{VarIn}(H))\,\theta_2$, so

$$\mathit{Ran}(\eta_2) \subseteq \mathit{Ran}(\theta_1 | \mathit{VarIn}(H)) \cup \mathit{Var}(\theta_2). \tag{6}$$

But by (1), (4), (2), disjointness of A and H, and (3)

$$(\mathit{Ran}(\theta_1 | \mathit{VarIn}(H)) \cup \mathit{Var}(\theta_2)) \cap (\mathit{Ran}(\theta_1 | V) \cup V) = \emptyset,$$

so by (6) and (5) we conclude (iii). □

Note that the first atom of a nicely moded goal is output linear and input-output disjoint, and a variant of a nicely moded clause is nicely moded. Thus to prove Theorem 5.2 it suffices to prove the following lemma which shows the "persistence" of the notion of being nicely moded.

Lemma 5.9 *An LD-resolvent of a nicely moded goal and a disjoint with it nicely moded clause is nicely moded.*

Proof. We start by proving three claims.

Claim 1 *Suppose that A and H satisfy the assumptions of Lemma 5.8 and assume that θ is a relevant mgu of $A = H$ which satisfies conditions $(i) - (iii)$ of Lemma 5.8. Let $H \leftarrow \mathbf{B}$ be a nicely moded clause with no variables in common with A. Then $\leftarrow \mathbf{B}\theta$ is nicely moded.*

Proof. Below, by the standardization apart we mean the assumption that $H \leftarrow \mathbf{B}$ and A have no variables in common. Let V, η_1 and η_2 be as in the formulation of Lemma 5.8.

Let $\theta_1 = \theta | \mathit{VarOut}(\mathbf{B})$ and $\theta_2 = \theta | (\mathit{VarIn}(\mathbf{B}) - \mathit{VarOut}(\mathbf{B}))$. We first establish some claims about θ_1 and θ_2. By the standardization apart and the definition of a nicely moded clause

$$\mathit{VarOut}(\mathbf{B}) \cap (\mathit{Var}(A) \cup \mathit{Var}(H)) \subseteq V, \tag{7}$$

so by the fact that θ is relevant

$$\theta_1 \subseteq \eta_1. \tag{8}$$

Thus by the linearity of η_1 (condition (i) of Lemma 5.8)

$$\theta_1 \text{ is linear.} \tag{9}$$

Moreover, by (8), (ii) of Lemma 5.8 and standardization apart

$$\mathit{Ran}(\theta_1) \cap \mathit{Var}(\mathbf{B}) = \emptyset. \tag{10}$$

Now, let $\theta_2' = \theta_2 | V$ and $\theta_2'' = \theta_2 | \mathit{VarIn}(H)$. We have

$$\theta_2 = \theta_2' \dot{\cup}\, \theta_2'', \tag{11}$$

$$\theta_2' \subseteq \eta_1, \tag{12}$$

and

$$\theta_2'' \subseteq \eta_2. \tag{13}$$

Consider now θ_2'. We have $Dom(\theta_1) \cap Dom(\theta_2) = \emptyset$, so $Dom(\theta_1) \cap Dom(\theta_2') = \emptyset$. Thus, by (8), (12) and the linearity of η_1

$$Ran(\theta_2') \cap Ran(\theta_1) = \emptyset \tag{14}$$

Moreover, by (12), (ii) of Lemma 5.8 and the standardization apart

$$Ran(\theta_2') \cap VarOut(\mathbf{B}) = \emptyset. \tag{15}$$

Consider now θ_2''. By (8), (13) and (iii) of Lemma 5.8 we get

$$Ran(\theta_2'') \cap Ran(\theta_1) = \emptyset. \tag{16}$$

Also, by the fact that θ is relevant $Ran(\theta_2'') \subseteq Var(A) \cup Var(H)$, so by (7) $Ran(\theta_2'') \cap VarOut(\mathbf{B}) \subseteq V$. Thus by (13) and (iii) of Lemma 5.8

$$Ran(\theta_2'') \cap VarOut(\mathbf{B}) = \emptyset. \tag{17}$$

Combining (14) with (16) and (15) with (17) we get by virtue of (11)

$$Ran(\theta_2) \cap (Ran(\theta_1) \cup VarOut(\mathbf{B})) = \emptyset. \tag{18}$$

Now, let us consider $\mathbf{B}$ more in detail. Suppose $\mathbf{B} = p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$. By assumption $\mathbf{t_1}, \ldots, \mathbf{t_n}$ is a linear family of terms and for $i \in [1, n]$ $\mathbf{t_i}\theta \equiv \mathbf{t_i}\theta_1$. So by (9), (10) and Lemma 5.6 $\mathbf{t_1}\theta, \ldots, \mathbf{t_n}\theta$ is a linear family of terms, as well.

Fix now $i \in [1, n]$ and $j \in [i, n]$. We have

$$Ran(\theta | Var(\mathbf{s_i})) \subseteq Ran(\theta_1 | Var(\mathbf{s_i})) \cup Ran(\theta_2 | Var(\mathbf{s_i})) \tag{19}$$

and

$$Ran(\theta | Var(\mathbf{t_j})) = Ran(\theta_1 | Var(\mathbf{t_j})). \tag{20}$$

$\leftarrow \mathbf{B}$ is nicely moded, so

$$Var(\mathbf{s_i}) \cap Var(\mathbf{t_j}) = \emptyset. \tag{21}$$

Thus by the linearity of θ_1 $Ran(\theta_1 | Var(\mathbf{s_i})) \cap Ran(\theta_1 | Var(\mathbf{t_j})) = \emptyset$, and consequently by (19), (20) and (18)

$$Ran(\theta | Var(\mathbf{s_i})) \cap Ran(\theta | Var(\mathbf{t_j})) = \emptyset. \tag{22}$$

Next, by (20) and (10)

$$Var(\mathbf{s_i}) \cap Ran(\theta | Var(\mathbf{t_j})) = \emptyset. \tag{23}$$

Finally, by (19), (10) and (18)

$$Var(\mathbf{t_j}) \cap Ran(\theta | Var(\mathbf{s_i})) = \emptyset. \tag{24}$$

Now, by (21), (22), (23), (24) and Lemma 5.4 we conclude that $Var(\mathbf{s_i}\theta) \cap Var(\mathbf{t_j}\theta) = \emptyset$. This proves that $\leftarrow \mathbf{B}\theta$ is nicely moded. □

Claim 2 *Let θ be a substitution and $\leftarrow \mathbf{A}$ a nicely moded goal such that $Var(\theta) \cap VarOut(\mathbf{A}) = \emptyset$. Then $\leftarrow \mathbf{A}\theta$ is nicely moded, as well.*

Proof. For any term s and a substitution σ we have $Var(s\sigma) \subseteq Var(s) \cup Var(\sigma)$. Moreover, for any term t occurring at an output position of A by the assumption about θ we have $t\theta = t$. The claim now follows by the definition of a nicely moded goal. □

Claim 3 *Suppose* $\leftarrow \mathbf{A}$ *and* $\leftarrow \mathbf{B}$ *are nicely moded goals such that* $VarOut(\mathbf{A}) \cap Var(\mathbf{B}) = \emptyset$. *Then* $\leftarrow \mathbf{B}, \mathbf{A}$ *is a nicely moded goal, as well.*

Proof. Immediate by the definition of a nicely moded goal. □

Consider now a nicely moded goal $\leftarrow A, \mathbf{A}$ and a disjoint with it nicely moded clause $H \leftarrow \mathbf{B}$, such that A and H unify. Observe that A and H satisfy the assumptions of Lemma 5.8. Assume now that θ is a relevant mgu of $A = H$ which satisfies conditions $(i) - (iii)$ of Lemma 5.8. By Claim 1 $\leftarrow \mathbf{B}\theta$ is nicely moded.

θ is relevant and $Var(A) \cap VarOut(\mathbf{A}) = \emptyset$, so by the standardization apart

$$Var(\theta) \cap VarOut(\mathbf{A}) = \emptyset. \tag{25}$$

By Claim 2 $\leftarrow \mathbf{A}\theta$ is nicely moded.

But (25) implies that $VarOut(\mathbf{A}\theta) = VarOut(\mathbf{A})$. Moreover, $Var(\mathbf{B}\theta) \subseteq Var(\mathbf{B}) \cup Var(\theta)$ and by the standardization apart $VarOut(\mathbf{A}) \cap Var(\mathbf{B}) = \emptyset$, so, again by (25),

$$VarOut(\mathbf{A}\theta) \cap Var(\mathbf{B}\theta) = \emptyset. \tag{26}$$

Now (26) establishes the last assumption of Claim 3 with $\leftarrow \mathbf{A}$ replaced by $\leftarrow \mathbf{A}\theta$ and $\leftarrow \mathbf{B}$ replaced by $\leftarrow \mathbf{B}\theta$. We conclude by Claim 3 that the LD-resolvent $\leftarrow (\mathbf{B}, \mathbf{A})\theta$ of the goal $\leftarrow A, \mathbf{A}$ and the clause $H \leftarrow \mathbf{B}$ is nicely moded.

θ is just one specific mgu of $A = H$. By Lemma 2.2 every other mgu of $A = H$ is of the form $\theta\eta$ for a renaming η. But a renaming of a nicely moded goal is nicely moded, so we conclude that every LD-resolvent of $\leftarrow A, \mathbf{A}$ and $H \leftarrow \mathbf{B}$ is nicely moded. □

This brings us to the following conclusion.

Corollary 5.10 *Let P and G be nicely moded. Suppose that*

- *the head of every clause of P is input linear.*

Then $P \cup \{G\}$ is occur-check free.

Proof. By Theorems 3.7 and 5.2. □

This corollary is stated in Chadha and Plaisted [CP91] as a direct consequence of Theorem 3.7 without mentioning Theorem 5.2. In our opinion the latter theorem is necessary to draw the above conclusion. Pierre Deransart (private communication) pointed out to us that this corollary is a consequence of Theorem 4.1 in Deransart, Ferrand and Téguia [DFT91] whose conditions are satisfied for a nicely moded program P and a nicely moded goal G. This actually suggests a stronger result, namely that such a P and G is NSTO.

It is worthwhile to note that to prove Corollary 5.10 it is actually sufficient to prove Lemma 5.9 under the assumption that the head of every clause of P is input linear. The proof is considerably simpler than that of Lemma 5.9.

This corollary can be easily applied to the previously studied programs.

Example 5.11
(i) Consider again the program append with the moding app(+,+,-). Clearly, append is nicely moded and that the head of every clause is input linear. By Corollary 5.10 we conclude that when u is linear and $Var(u) \cap Var(s,t) = \emptyset$, append $\cup$ { $\leftarrow$ app(s, t, u)} is occur-check free.

(ii) With the moding app(-,-,+) the program append is nicely moded, as well, and the head of every clause is input linear. Again, by Corollary 5.10 we conclude that when s,t is a linear family of terms and $Var(u) \cap Var(\{s,t\}) = \emptyset$, append $\cup$ { $\leftarrow$ app(s, t, u)} is occur-check free.

(iii) Reconsider now the program permutation with the modings as before. Again, it is easy to check that permutation is nicely moded and that the heads of all clauses are input linear. By Corollary 5.10 we get that when t is linear and $Var(s) \cap Var(t) = \emptyset$, permutation $\cup$ { $\leftarrow$ perm(s, t)} is occur-check free.

(iv) Consider again the program quicksort with the modings as before. Again, Corollary 5.10 applies and we conclude that when t is linear and $Var(s) \cap Var(t) = \emptyset$, quicksort $\cup$ { $\leftarrow$ qs(s, t)} is occur-check free.

(v) So far it seems that Corollary 5.10 allows us to draw more useful conclusions that Corollary 4.3. However, reconsider the program palindrome. In Chadha and Plaisted [CP91] it is shown that no moding exists in which palindrome is nicely moded with the heads of all clauses being input linear. Thus Corollary 5.10 cannot be applied to this program. □

Finally, let us mention that Chadha and Plaisted [CP91] proposed two efficient algorithms for generating modings with the minimal number of input positions, for which the program is nicely moded. These algorithms were implemented and applied to a number of well-known Prolog programs.

6 Strictly Moded Programs

Finally, we consider syntactic restrictions that imply the condition of Theorem 3.8. To this end it is sufficient to combine the properties of being well-moded and nicely moded.

Definition 6.1

- A goal $\leftarrow p_1(s_1,t_1),\ldots,p_n(s_n,t_n)$ is called *strict* if $t_1,\ldots,t_n$ is a linear family of terms.
- A clause $H \leftarrow B$ is called *strict* if $\leftarrow B$ is strict.
- A program is called *strict* if every clause of it is strict.
- A goal (clause) (program) is called *strictly moded* if it is both strict and well-moded. □

Theorem 6.2 *Let P and G be strictly moded. Then all LD-derivations of $P \cup \{G\}$ are both data and output driven.*

Proof. Omitted. □

Corollary 6.3 *Let P and G be strictly moded. Then $P \cup \{G\}$ is occur-check free.*

Proof. By Theorems 6.2 and 3.8. □

7 Conclusions

The aim of this paper was to provide simple syntactic conditions which imply that for a given program P and goal G, $P \cup \{G\}$ is occur-check free. To apply the established results one needs to find appropriate modings for the considered relations such that the conditions of one of the established Corollaries (4.3, 5.10 or 6.3) are satisfied. In the table below several programs taken from the book of Sterling and Shapiro [SS86] are listed. (A similar analysis of the notion of a well-moded program was carried in Drabent [Dra87]). For each program it is indicated which of the relevant conditions for a given moding are satisfied. All built-in's are moded completely input.

In programs which use difference lists we replaced "\" by ",", thus splitting a position filled in by a difference list into two positions. Because of this change in some relations additional arguments are introduced, and so certain clauses have to be modified in an obvious way. For example, in the parsing program on page 258 each clause of the form `p(X)` $\leftarrow$ `r(X)` has to be replaced by `p(X,Y)` $\leftarrow$ `r(X,Y)`. Such changes are purely syntactic and they allow us to draw conclusions about the occur-check freedom of the original program.

The modings considered are usually intuitive and at least one of the Corollaries 4.3, 5.10 or 6.3 applies. This indicates that the established results are widely applicable and thus justifies the title of this paper.

program	page	moding	well-moded	heads out. lin.	nicely moded	heads in. lin.	strictly moded
member	45	(−,+)	yes	yes	yes	yes	yes
member	45	(+,+)	yes	yes	yes	no	yes
prefix	45	(−,+)	yes	yes	yes	yes	yes
prefix	45	(+,+)	yes	yes	yes	no	yes
suffix	45	(−,+)	yes	yes	yes	yes	yes
suffix	45	(+,+)	yes	yes	yes	no	yes
naive reverse	48	r(+,−) a(+,+,−)	yes	yes	yes	yes	yes
reverse-accum.	48	r(+,−) r(+,+,-)	yes	yes	yes	yes	yes
delete	53	(+,+,−)	yes	yes	yes	no	yes
select	53	(+,+,−)	yes	yes	yes	no	yes
insertion sort	55	s(+,−) i(+,+,−)	yes	yes	yes	yes	yes
tree-member	58	(−,+)	yes	yes	yes	yes	yes
tree-member	58	(+,+)	yes	yes	yes	no	yes

isotree	58	(+,+)	yes	yes	yes	no	yes
substitute	60	(+,+,+,-)	yes	yes	yes	no	yes
pre-order	60	p(+,-) a(+,+,-)	yes	yes	yes	yes	yes
in-order	60	i(+,-) a(+,+,-)	yes	yes	yes	yes	yes
post-order	60	p(+,-) a(+,+,-)	yes	yes	yes	yes	yes
polynomial	62	(+,+)	yes	yes	yes	no	yes
derivative	63	(+,+,-)	yes	no	yes	no	yes
hanoi	64	h(+,+,+,-) a(+,+,-)	yes	yes	yes	yes	yes
append_dl	241	(+,-,+,+,-,-)	yes	yes	yes	yes	yes
append_dl	241	(+,-,+,-,-,-)	no	no	yes	yes	no
flatten_dl	241	f(+,+) f_dl(+,+,-)	yes	yes	yes	no	yes
flatten	243	f(+,-) f(+,+,-)	yes	yes	yes	yes	yes
reverse_dl	244	r(+,-) r_dl(+,-,+)	yes	yes	yes	yes	yes
quicksort dl	244	q(+,+) q_dl(+,+,-) p(+,+,-,-)	yes	yes	no	yes	yes
dutch	246	dutch(+,-) di(+,-,-,-)	yes	yes	yes	yes	yes
dutch_dl	246	dutch(+,-) di(+,-,+,-,+,-,+)	yes	yes	yes	yes	yes
parsing	258	all (+,-)	yes	yes	yes	yes	yes

P.S. Of course, you would like to know to which two programs from Sterling and Shapiro [SS86] we could not apply the results of this paper. These are `flatten_dl` (program 15.2 on page 241): and `quicksort_dl` (program 15.4 on page 244).

The appropriate entry in the table above indicates that, after replacing "\" by ",", in the mode `flatten(+,+)` and `flatten_dl(+,+,-)`, `flatten_dl` is well-moded and the heads of the clauses are output linear. Thus by virtue of Corollary 4.3 for `s` and `t` ground, all LD-derivations of `flatten_dl` $\cup$ { $\leftarrow$ `flatten(s,t)`} are occur-check free. Similar conclusion can be drawn about `quicksort_dl` moded `qs(+,+)` and `qs_dl(+,+,-)`.

However, *no* conclusion can be drawn for the modes `flatten(+,-)` and `qs(+,-)` in which these two programs are customarily used. Indeed, it is easy to check that for both programs no completion of the moding exists for which the program is well-moded, or nicely moded and with the heads of all clauses being input linear.

A solution to this problem is proposed in Pellegrini [Pel92] and for the space reasons omitted here.

Acknowledgement

We thank Pierre Deransart for constructive remarks on the subject of this paper.

References

[Apt90] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.

[CP91] R. Chadha and D.A. Plaisted. Correctness of unification without occur check in Prolog. Technical report, Department of Computer Science, University of North Carolina, Chapel Hill, N.C., 1991.

[DFT91] P. Deransart, G. Ferrand, and M. Téguia. NSTO programs (not subject to occur-check). In V. Saraswat and K. Ueda, editors, *Proceedings of the International Logic Symposium*, pages 533–547. The MIT Press, 1991.

[DM85a] P. Dembinski and J. Maluszynski. AND-parallelism with intelligent backtracking for annotated logic programs. In *Proceedings of the International Symposium on Logic Programming*, pages 29–38, Boston, 1985.

[DM85b] P. Deransart and J. Maluszynski. Relating Logic Programs and Attribute Grammars. *Journal of Logic Programming*, 2:119–156, 1985.

[Dra87] W. Drabent. Do Logic Programs Resemble Programs in Conventional Languages? In *International Symposium on Logic Programming*, pages 389–396. San Francisco, IEEE Computer Society, August 1987.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[LMM88] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[Pel92] A. Pellegrini. Sul problema dell' "occur check" in Prolog. Technical report, Department of Computer Science, University of Padova, Padova, Italy, 1992. Tesi di Laurea, in Italian, to appear.

[Pla84] D.A. Plaisted. The occur-check problem in Prolog. In *Proc. International Conference on Logic Programming*, pages 272–280. IEEE Computer Science Press, 1984.

[Ros91] D.A. Rosenblueth. Using program transformation to obtain methods for eliminating backtracking in fixed-mode logic programs. Technical Report 7, Universidad Nacional Autonoma de Mexico, Instituto de Investigaciones en Matematicas Aplicadas y en Sistemas, 1991.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1986.