A Note on Occur-Check

Włodzimierz Drabent
Institute of Computer Science, Polish Academy of Sciences
Department of Computer and Information Science, Linköping University, Sweden
drabent *at* ipipan *dot* waw *dot* pl

Most known results on avoiding the occur-check are based on the notion of "not subject to occur-check" (NSTO). It means that unification is performed only on such pairs of atoms for which the occur-check never succeeds in any run of a nondeterministic unification algorithm. Here we show that this requirement is too strong. We show how to weaken it, and present some related sufficient conditions under which the occur-check may be safely omitted. We show examples for which the proposed approach provides more general results than the approaches based on well-moded and nicely moded programs (this includes cases to which the latter approaches are inapplicable).

Keywords: occur-check, unification, modes, delays

1 Introduction

The programming language Prolog implements SLD-resolution employing an unsound implementation of unification without the occur-check. This usually creates no problems in practice. Programmers know that they do not need to care about it, unless they deal with something unusual like checking a difference list for emptiness.[1] Surprisingly, such attitude of programmers is often not justified by theory. The known criteria for occur-check freeness are applicable to restricted classes of cases. There seems to exist no further substantial work on avoiding the occur-check after that of Chadha and Plaisted [CP94], Apt and Pellegrini [AP94], reported in [Apt97], and the generalization in [AL95] to other selection rules than that of Prolog.

Even for LD-resolution (SLD-resolution with the Prolog selection rule) the proposed methods are inapplicable to some important cases. To deal with simple examples of programs employing difference lists, the methods of well-moded and nicely moded programs had to be refined [AP94] in a rather sophisticated way. (The refinement is not presented in the textbook [Apt97], one may suppose that it was considered too complicated.)

Here we are interested in sufficient conditions for safe execution of definite clause programs without the occur check. Approaches based on semantic analysis, like abstract interpretation, are left outside of the scope of this paper.

The existing approaches are based on the notion of NSTO (not subject to occur-check) [DFT91]. It means that unification is performed only on such pairs of atoms for which the occur-check never succeeds in *any* run of a nondeterministic unification algorithm.

It turns out that unification without the occur-check works correctly also for some cases which are not NSTO. In this paper we propose a generalization of NSTO. We show that it is sufficient that the occur-check does not succeed in *one* run of the unification algorithm for a given input (instead of *all*

[1] In the important Prolog textbook by Sterling and Shapiro [SS94], the occur-check is mentioned (in the context of actual programs) only when discussing difference lists (p. 298, p. 300, on p. 299 an error due to unsound Prolog unification is explained). The textbook of Bratko [Bra12] mentions the occur-check only once, when comparing matching in Prolog with unification in logic.

A. Formisano, Y.A Liu, et al. (Eds.): International Conference on
Logic Programming (Technical Communications) 2021 (ICLP 2021)
EPTCS 345, 2021, pp. 54–67, doi:10.4204/EPTCS.345.17

the runs). We discuss some related sufficient conditions for safely avoiding the occur-check. They are applicable to some examples to which the former approaches are inapplicable. For other examples, a wider class of initial queries is dealt with, or/and applying the proposed approach seems simpler than the former ones. We additionally present a sufficient condition, based on NSTO, for safely avoiding the occur-check under arbitrary selection rule (and provide a detailed proof of its correctness).

Preliminaries

We use the terminology, notation and many definitions from [Apt97] (and reintroduce here only some of them). The terminology is based on that of logic; in particular "atom" means an atomic formula.

By an *expression* we mean a term, an atom, or a tuple of terms (or atoms). An equation is a construct $s \doteq t$, where s,t are expressions. Given sequences of terms (or atoms) $s = s_1,\ldots,s_n$ and $t = t_1,\ldots,t_n$, the set $\{s_1 \doteq t_1,\ldots,s_n \doteq t_n\}$ will be sometimes denoted by $s \doteq t$. A syntactic object (expression, equation, substitution, etc) is *linear* when no variable occurs in it more than once. As in Prolog, each occurrence of _ in a syntactic object will stand for a distinct variable. Otherwise variable names begin with upper case letters. $Var(t)$ denotes the set of variables occurring in a syntactic object t. We say that s and t are *variable disjoint* if $Var(s) \cap Var(t) = \emptyset$.

For a substitution $\theta = \{X_1/t_1,\ldots,X_n/t_n\}$, we define $Dom(\theta) = \{X_1,\ldots,X_n\}$, $Ran(\theta) = Var(t_1,\ldots,t_n)$, $\theta|S = \{X/t \in \theta \mid X \in S\}$ (for a set S of variables), and $\theta|u = \theta|Var(u)$ (for an expression u).

We employ the Martelli-Montanari unification algorithm (MMA) (cf. [Apt97]). It unifies a set of equations, by iteratively applying one of the actions below to an equation from the current set, until no action is applicable. The equation is chosen nondeterministically.

(1) $f(s_1,\ldots,s_n) \doteq f(t_1,\ldots,t_n)$	$\longrightarrow$	replace by equations $s_1 \doteq t_1,\ldots,s_n \doteq t_n$
(2) $f(s_1,\ldots,s_n) \doteq g(t_1,\ldots,t_m)$ where $f \neq g$	$\longrightarrow$	halt with failure
(3) $X \doteq X$	$\longrightarrow$	delete the equation
(4) $t \doteq X$ where t is not a variable	$\longrightarrow$	replace by $X \doteq t$
(5) $X \doteq t$ where $X \notin Var(t)$ and X occurs elsewhere	$\longrightarrow$	apply substitution $\{X/t\}$ to all other equations
(6) $X \doteq t$ where $X \in Var(t)$ and $X \neq t$	$\longrightarrow$	halt with failure

By a *run* of MMA for an input E we mean a maximal sequence $E_0,\ldots,E_n$ of equation sets such that $E = E_0$ and E_i is obtained from E_{i-1} by one of the actions of the algorithm ($i = 1,\ldots,n$). See [Apt97] for the properties of MMA, in particular how the obtained mgu is represented by a final equation set.

An equation set E is said to be *NSTO* if action (6) is not performed in any execution of MMA starting with E. We often say "unification of s and t is NSTO" instead of "$\{s \doteq t\}$ is NSTO". In such case while unifying s,t the occur check never succeeds, and thus can be skipped.

We need to generalize some definitions from [Apt97], in order not to be limited to LD-resolution. We will say that *unification of A and H is available* in an SLD-derivation (or SLD-tree) for a program P, if A is the selected atom in a query of the derivation (tree), and H is a standardized apart head of a clause from P, such that A and H have the same predicate symbol. (A more formal phrasing is "equation set $\{A \doteq H\}$ is available".) If all the unifications available in an SLD-derivation (SLD-tree) are NSTO then the derivation (tree) is *occur-check free*. We say that a program P with a query Q is *occur-check free* if, under a given selection rule, the SLD-tree for P with Q is occur-check free.

We refer a few times to results of [AP94] reported in [Apt97]; in such cases only a reference to [Apt97] may be given.

Similarly to [AP94], we will employ modes. This means dividing the argument positions of predicates into two groups, by assigning a function $m_p: \{1,\ldots,n\} \to \{+,-\}$ (called a *mode*) to each predicate p of the considered program (where n is the arity of p). A program with a mode for each predicate is called a *moded program*, and the collection of modes is called *moding*. We follow the usual terminology, argument positions with $+$ assigned are called input, and those with $-$ are called output. We usually specify m_p by writing $p(m_p(1),\ldots,m_p(n))$. E.g. $p(+,-)$ states that the first argument of p is input and the second one is output. Note that moding does not need to correspond to any intuitive notion of data flow. It is to be chosen so that the moded program satisfies the conditions of interest.

We will write $p(s;t)$ to represent an atom $p(t_1,\ldots,t_n)$ and to state that s is the sequence of terms in its input positions, and t the sequence of terms in its output positions. An atom $p(s;t)$ is *input-output disjoint* if $Var(s) \cap Var(t) = \emptyset$. Let us define $VarIn(p(s;t)) = Var(s)$, $VarOut(p(s;t)) = Var(t)$. The input (resp. output) positions of a query Q are the input (output) positions of the atoms of Q. A query (or an atom) Q is *input linear* (resp. *output linear*) if the sequence of the terms occurring in the input (output) positions of Q is linear. We will refer to the following results.

Lemma 1 *1. Consider atoms A and H. If they are variable disjoint, one of them is input-output disjoint, one of them is input linear, and the other is output linear then $\{A \doteq H\}$ is NSTO [Apt97, Lemma 7.14].*

2. Let s and t be sequences of terms, such that the lengths of s and t are the same. If $Var(s) \cap Var(t) = \emptyset$ and s (or t) is linear then $s \doteqdot t$ is NSTO (a special case of [Apt97, Lemma 7.5]).

Obviously, $p(s) \doteq p(t)$ is NSTO iff $s \doteqdot t$ is NSTO. Based on Lemma 1.1, Apt and Pellegrini [AP94] introduced two sufficient conditions for occur-check freeness. One (*well-moded* programs [Apt97, Def. 7.8]) implies that the input positions of the atoms selected in LD-trees are ground. The other one (*nicely-moded* programs [Apt97, Def. 7.19]) implies that the selected atoms are output-linear.

2 NSTO and arbitrary selection rules

2.1 Sufficient condition

Here we propose a syntactic condition for occur-check freeness under arbitrary selection rules. We assume that the programs dealt with are moded.

Definition 2 *Let $Q = A_1,\ldots,A_n$ be a query. We define a relation $\to_Q$ on $\{A_1,\ldots,A_n\}$. Let $A_i \to_Q A_j$ when a variable occurs in an output position of A_i and an input position of A_j.*

Query Q is **tidy** *if it is output linear and $\to_Q$ is acyclic ($A_i \not\to_Q^+ A_i$ for $i = 1,\ldots,n$).*

Clause $H \leftarrow Q$ is **tidy** *if Q is tidy, and*

H is input linear,

no variable from an input position of H occurs in an output position of Q.

Note that each atom in a tidy query is input-output disjoint. Also, if a query Q is tidy then any permutation of Q is tidy too. A linear query is tidy under any moding.

Tidy programs are a generalization of nicely moded programs [CP94], [AP94] [Apt97] (the first reference uses different terminology). A nicely moded query $A_1,\ldots,A_n$ is tidy (as $A_i \to_Q A_j$ implies $i < j$), and a nicely moded clause with an input linear head is tidy. A tidy program can be converted into a nicely moded one by reordering the atoms in the clause bodies. The generalization proposed here may be understood as minor. However it still seems to be worth presenting, as the proof of the crucial lemma in [AL95] seems unavailable.

This is a basic property of tidy clauses and queries:

Lemma 3 *Let Q be a tidy query, and C a tidy clause variable disjoint from Q. An SLD-resolvent Q' of Q and C is tidy.*

Due to a space limit, the proof must be excluded from this paper and will be made available elsewhere. We only mention here two main lemmas involved in the proof.

Lemma 4 *Let θ be a substitution and X,V,V' variables. Assume that $X \in Var(V\theta)$ and $X \in Var(V'\theta)$. Then $X = V = V'$, or $X,V,V' \in Var(\theta)$ and moreover $X \in Ran(\theta)$.*

The next lemma employs the following notion: A substitution θ is **linear for** a set of variables $\mathscr{S}$ if for any two distinct variables $X,Y \in \mathscr{S}$ the pair $X\theta, Y\theta$ is linear. Note that if a $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ is linear for $Dom(\theta)$ then θ is linear in the sense of [AP94, Def. A.3] (i.e. $t_1, \ldots, t_n$ is linear).

Lemma 5 *Consider two variable disjoint expressions s,t where t is linear. Let $\mathscr{S}$ be a set of variables such that $\mathscr{S} \cap Var(t) = \emptyset$. If s,t are unifiable then there exists a relevant and idempotent mgu θ of s,t such that*

$$\theta|s \text{ is linear for } \mathscr{S} \qquad \text{and} \qquad Ran(\theta|s) \subseteq Var(t).$$

Now our sufficient condition for occur-check freeness is:

Corollary 6 *A tidy program with a tidy query is occur-check free, under any selection rule.*

PROOF By Lemma 3, in each SLD-derivation for a tidy program and query, each query is tidy. Assume A is an atom from a tidy query and H is the head of a standardized apart tidy clause. As A is input-output disjoint and output linear and H is input linear, $\{A \doteq H\}$ is NSTO, by Lemma 1. □.

2.2 Examples

Apt and Pellegrini [AP94] found that the approaches based on well-modedness or nice modedness are inapplicable to some programs, and introduced a more sophisticated approach. The programs are FLATTEN [SS94, Program 15.2], QUICKSORT_DL [SS94, Program 15.4], and NORMALIZE ([SS94, Program 15.7]); they employ difference lists. Here we focus on FLATTEN, which flattens a given list. (We split arguments of the form $t\backslash u$ or t++u into two argument positions; the argument positions of built-in predicates are considered input.)

FLATTEN:

```
%flatten_dl(Xs,Ys,Zs) – difference list Ys\Zs represents the flattened list Xs
flatten_dl([X|Xs],Ys,Zs) ← flatten_dl(X,Ys,Ys1), flatten_dl(Xs,Ys1,Zs).
flatten_dl(X,[X|Xs],Xs) ← constant(X), X \== [].
flatten_dl([],Xs,Xs).
flatten(Xs,Ys) ← flatten_dl(Xs,Ys,[]).
```

We see that, for the program to be tidy, the second and the third arguments of *flatten_dl* cannot be both output (*Ys1* occurs in these positions in a clause body, which must be output linear). Also, the first and the second arguments of *flatten_dl* cannot be both input, the same for the second and the third one (as a clause head must be input linear). The reader is encouraged to check that FLATTEN is tidy under moding $M_1 = flatten(+,-), flatten\_dl(+,-,+)$, and under $M_2 = flatten(-,+), flatten\_dl(-,+,-)$. The relation

$\rightarrow_Q$ (where $Q = A_1, A_2$ is the body of the first clause) consists of one pair; for M_1 it is $A_2 \rightarrow_Q A_1$, for M_2 it is $A_1 \rightarrow_Q A_2$. In both cases the program remains tidy if the moding of *flatten* is replaced by $(-,-)$.

For any term t and variable $R \notin Var(t)$, query $Q_0 = flatten(t,R)$ is tidy under M_1. (To be tidy under M_2, Q_0 has to be linear.) By Corollary 6, FLATTEN with Q_0 is occur-check free, under any selection rule.

We only mention that QUICKSORT_DL and NORMALIZE are also tidy, and thus are occur-check free for a wide class of queries. NORMALIZE is similar to FLATTEN, and is tidy for similar modings. QUICKSORT_DL is tidy for instance for modings $quicksort(+,-)$, $quicksort\_dl(+,-,+)$, $partition(+,+,-,-)$, and $quicksort(-,+)$, $quicksort\_dl(-,+,-)$, $partition(-,+,+,+)$. Another example of a tidy program is DERIVATIVE (Example 22 below).

Surprisingly, the approach based on nice modedness *is* applicable to FLATTEN and NORMALIZE. FLATTEN is nicely moded under M_2, so is the query Q_0, provided it is linear. As the clause heads are input linear, it follows that FLATTEN with Q_0 is occur-check free for the Prolog selection rule (by [Apt97, Corollary 7.25]). Similarly, NORMALIZE is nicely moded (e.g. under $normalize\_ds(-,+,-)$); we skip further details. In both cases the modes may be seen as not natural; what is understood as input data appears in a position moded as output. This may explain why the nice modedness of the programs was not noticed in [AP94].

3 Weakening NSTO

The discussion on avoiding the occur-check above, and in all the work referred here, is based on the notion of NSTO. We show that NSTO is a too strong requirement. Unification without the occur-check produces correct results also for some pairs of atoms which are not NSTO. In such cases the algorithm may temporarily construct infinite terms, but eventually halt with failure. An example of such pair is $p(a, f(X), X)$, $p(b, Y, Y)$. For this pair, some runs of MMA halt due to selecting $a \doteq b$, some other ones due to a successful occur-check. Omitting the occur-check would result in failure on $a \doteq b$; this is a correct result.

NSTO requires that each run of MMA does not perform action (6). In this section we show that it is sufficient that there *exists* such run. For this we need to introduce a precise description of the algorithm without the occur-check, called MMA$^-$. We define WNSTO, a weaker version of NSTO, and show that MMA$^-$ produces correct results for expression pairs that are WNSTO. Then we show an example of a program with a query which is not occur-check free, but will be correctly executed without the occur-check, as all the atom pairs to be unified are WNSTO. Then we present sufficient conditions, based on WNSTO, for safely skipping the occur-check.

3.1 An algorithm without the occur-check

By abuse of terminology, we will write "unification algorithm without the occur-check", despite such algorithm does not correctly implement unification. We would not consider any actual unification algorithm of Prolog, this would require to deal with too many low level details. See for instance the algorithm of [Aït91, Section 2]. Instead, we use a more abstract algorithm, obtained from MMA. We cannot simply drop the occur-check from MMA (by removing action (6) and the condition $X \notin Var(t)$ in action (5)). The resulted algorithm may not terminate, as an equation $X \doteq t$ where $X \in Var(t)$ can be selected infinitely many times.

We obtain a reasonable algorithm in two steps. First, MMA is made closer to actual unification algorithms. The idea is to abandon action (5), except for t being a variable. (The action applies $\{X/t\}$ to

all equations except one). Instead, $\{X/t\}$ is applied only when needed, and only to one occurrence of X. This happens when the variable becomes the left hand side of two equations $X \doteq t$, $X \doteq u$ (where t, u are not variables, and $X \notin Var(t,u)$). Then $X \doteq u$ is replaced by $t \doteq u$. To achieve termination, t should not be larger than u. From such algorithm the occur-check may be dropped.

Without loss of generality we assume that we deal with unification of terms. Let $|t|$ be the number of occurrences in t of variables and function symbols, including constants.

Definition 7 ([Col82]) **MMA$^-$** *(MMA without the occur-check) is obtained from MMA by*

(a) removing action (6), and
(b) replacing action (5) by

(5a) $X \doteq Y$*, where* X, Y *are distinct variables and* X *occurs elsewhere*
$\longrightarrow$ *apply substitution* $\{X/Y\}$ *to all other equations,*

(5b) $X \doteq t$, $X \doteq u$ *where* t, u *are distinct non-variable terms; let* $\{s_1, s_2\} = \{t, u\}$ *and* $|s_1| \leq |s_2|$
$\longrightarrow$ *replace* $X \doteq s_2$ *by* $s_1 \doteq s_2$

A set E of equations is in a **semi-solved form** *if E is $\{X_1 \doteq t_1, \ldots, X_n \doteq t_n\}$, where $X_1, \ldots, X_n$ are distinct, each X_i is distinct from t_i, and if a t_i is a variable then X_i occurs only once in E (for $i = 1, ..., n$). E is in a* **solved form** *if, additionally, $X_i \notin Var(t_j)$ for $1 \leq i, j \leq n$.*

Note that inability of performing any action of MMA$^-$ means that the equation set is in a semi-solved form.

Let us first discuss termination of MMA$^-$. Let $k > 1$ be an integer greater than the arity of each function symbol appearing in the equation set E which is the input of the algorithm. We define a function $||\ ||$ assigning natural numbers to equation sets and equations:

$$||\{t_1 \doteq u_1, \ldots, t_n \doteq u_n\}|| = \sum_{i=1}^{n} ||t_i \doteq u_i||, \qquad \text{where} \qquad ||t \doteq u|| = k^{\max(|t|,|u|)}.$$

Assume that an action of MMA$^-$ is applied to a set E of equations, resulting in E'. Then $||E|| = ||E'||$ if the action is (4), (5a), or (5b), and $||E|| > ||E'||$ if it is (3). By the lemma below, $||E|| > ||E'||$ for action (1).

Lemma 8 *For any terms (or atoms) $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$,* $||s \doteq t|| > \sum_{i=1}^{n} ||s_i \doteq t_i||$.

PROOF[2] The inequality obviously holds for $n = 0$. Let $n > 0$, and l, r be, respectively, the left and the right hand side of the inequality. Without loss of generality, assume that $|s| \geq |t|$. Now $l = k \cdot k^{|s_1|} \cdots k^{|s_n|} \geq k \cdot k^{|t_1|} \cdots k^{|t_n|}$. Hence $l/k \geq k^{|u|}$ for any $u \in \{s_1, \ldots, s_n, t_1, \ldots, t_n\}$. Thus $l/k \geq ||s_i \doteq t_i||$ for $i = 1, \ldots, n$, and then $n \cdot l/k \geq r$. As $n/k < 1$, we obtain $l > r$. □

Let $f_{45a}(E)$ be the number of those equations from E to which action (4) or (5a) applies. Let $f_{5b}(E)$ be the number of equations of the form $X \doteq u$ in E, where u is not a variable. Note that applying (4) or (5a) decreases $f_{45a}(E)$, and applying (5b) decreases $f_{5b}(E)$ without changing $f_{45a}(E)$,

[2] Function $||\ ||$ is proposed and the lemma is stated without proof in [Col82]. There, however, k is the maximal arity of symbols from E, which does not make sense when it is 0 or 1. The lemma also holds (with a slightly longer proof) for k being the maximal number out of 2 and the arities of the symbols.

Now consider the lexicographic ordering $\prec_3$ on $\mathbb{N}^3$ (cf. for instance [Apt97, p. 33]). If E' is obtained from E by applying one action of the algorithm, it holds that

$$(||E'||, f_{45a}(E'), f_{5b}(E')) \prec_3 (||E||, f_{45a}(E), f_{5b}(E)).$$

Thus, as $\prec_3$ is well-founded, MMA$^-$ terminates for any input set of equations E.

In discussing further properties of the algorithm, we will consider possibly infinite terms (**i-terms**) over the given alphabet. We require that the set of variables occurring in an i-term is finite. The corresponding generalization of the notion of substitution is called **i-substitution**.

Definition 9 *A substitution (respectively i-substitution) θ is a* **solution** *(***i-solution***) of an equation $t \doteq u$ if $t\theta = u\theta$; θ is a* solution *(*i-solution*) of a set E of equations, if θ is a solution (i-solution) of each equation from E. Two sets of equations are* **equivalent** *(respectively* **i-equivalent***) if they have the same set of solutions (i-solutions).*

Lemma 10 *Each action of MMA or of MMA$^-$ replaces an equation set by an i-equivalent one.*

PROOF For any i-substitution θ, $f(s_1,\ldots,s_n)\theta = f(t_1,\ldots,t_n)\theta$ iff $s_i = t_i$ for all $i \in \{1,\ldots,n\}$. Thus the claim holds for action (1). For actions (3), (4) the claim is obvious; the same for (2), (6), Actions (5) and (5a) replace an equation set $E = E_X \cup E_1$ by $E' = E_X \cup E_1\{X/t\}$, where $E_X = \{X \doteq t\}$. Consider an i-solution θ of E_X. So $X\theta = t\theta$. Hence $(V\{X/t\})\theta = V\theta$ for any variable V, and thus $t\{X/t\}\theta = t\theta$ for any expression t. So θ is a solution of E_1 iff θ is a solution of $E_1\{X/t\}$. For (5b), equivalence of $\{X \doteq t, X \doteq u\} \cup E_1$ and $\{X \doteq t, t \doteq u\} \cup E_1$ follows immediately from Def. 9 □

Lemma 11 *Any set of equations E in a semi-solved form has an i-solution.*

PROOF If an equation of the form $X_i \doteq Y$ occurs in E then E has an i-solution iff $E \setminus \{X_j \doteq Y\}$ has an i-solution (as such X_j occurs in E only once), Hence we can assume that E does not contain any equation of this form. Now the result follows from Th. 4.3.1 of [Cou83]. □

It remains to discuss the results of MMA$^-$. Note that if E and E' are i-equivalent then they are equivalent. Consider a run R of MMA$^-$ starting from an equation set E. If R halts with failure (due to action (2)) then, by Lemma 10, E has no solutions (is not unifiable). If it halts with equation set E' in semi-solved form, then by Lemma 10, E is unifiable iff E' is. So applying MMA to E', which boils down to applying actions (5) and (6), either halts with failure, or produces a solved form E'', representing an mgu of E. Prolog does not perform the occur-check, and treats the semi-solved form as the result of unification. Prolog implementations present the result to the user in various ways. For instance the answer to query $g(X,X) = g(Y,f(Y))$ is displayed as $X{=}Y, Y{=}f(Y)$ by SWI, and as $X{=}f(f(f(\ldots))), Y{=}f(f(f(\ldots)))$ by SICStus (predicate =/2 is defined by clause $=(Z,Z)$).

3.2 WNSTO

Let us say that a run of MMA is *occur-check free* if the run does not perform action (6). (In other words, no equation $X = t$ is selected where $X \in Var(t)$ and $X \neq t$; simply – the occur-check does not succeed in the run). An equation set E is **WNSTO** (weakly NSTO) when there exists an occur-check free run of MMA for E. When E is $s \doteq t$ we also say that the unification of s and t is WNSTO. A program P with a query Q is **weakly occur-check free** if, under a given selection rule, all the unifications available in the SLD-tree for P with Q are WNSTO. A run of MMA$^-$ on an equation set E is **correct** if it produces correct results i.e. the run halts with failure if E is not unifiable, and produces a unifiable equation set E'

in a semi-solved form otherwise. The latter means that applying to E' action (5) iteratively produces an mgu of E, in a form of an equation set in a solved form. We say that MMA$^-$ is **sound** for E if all the runs of MMA$^-$ on E are correct.

Now we show that if unification of E can be split in two parts and each of them is NSTO, then E is WNSTO. (For a proof see Appendix A.)

Lemma 12 *Let $E_1 \cup E_2$ be an equation set.*
If E_1 is not unifiable and is NSTO then $E_1 \cup E_2$ is WNSTO.
If θ_1 is an mgu of E_1, and each E_1 and $E_2\theta_1$ is NSTO then $E_1 \cup E_2$ is WNSTO.

Corollary 13 *Consider a moding and atoms $p(s;t)$ and $p(s';t')$, where $s \doteqdot s'$ is NSTO.*
If $s \doteqdot s'$ is not unifiable then $p(s;t) \doteq p(s';t')$ is WNSTO.
If θ is an mgu of $s \doteqdot s'$, and $(t \doteqdot t')\theta$ is NSTO then $p(s;t) \doteq p(s';t')$ is WNSTO.

PROOF Equation $p(s;t) \doteq p(s';t')$ is WNSTO iff equation set $s \doteqdot s' \cup t \doteqdot t'$ is WNSTO. Now Lemma 12 applies. □

WNSTO is sufficient for the unification without the occur-check to work correctly:

Theorem 14 *Consider an equation set E. Assume that there exists an occur-check free run of MMA on E. Then MMA$^-$ is sound for E.*

PROOF Let R_1 be an occur-check free run of MMA on E, and R_2 be a run of MMA$^-$ on E. We show that R_2 is correct. Let S be the set of the i-solutions of E, and thus of every equation set E' appearing in R_1 or R_2 (by Lemma 10).

If R_1 succeeds then S contains unifiers of E, and of every E' appearing in R_2. Hence action (2) is not performed in R_2, and R_2 halts with success producing a unifiable equation set E_2 in a semi-solved form.

If R_1 halts with failure then the last performed action is (2), thus $S = \emptyset$. This implies that R_2 does not produce a semi-solved form (by Lemma 11). Hence R_2 terminates with failure, due to action (2). □

It immediately follows that a weakly occur-check free program can be safely executed without the occur check:

Corollary 15 *Assume a selection rule. If a program P with a query Q is weakly occur-check free then algorithm MMA$^-$ is sound for each unification available in the SLD-tree for P with Q.*

In other words, P with Q may be correctly executed without the occur-check.

3.3 Example – a weakly occur-check free program

The core fragment of the *n* queens program [Frü91] will be now used as an example. We call it NQUEENS, see [Dra21a] for explanations.

```
pqs(0,_,_,_).                                   (1)
pqs(s(I),Cs,Us,[_|Ds]):-
        pqs(I,Cs,[_|Us],Ds),                    (2)
        pq(s(I),Cs,Us,Ds).

pq(I,[I|_],[I|_],[I|_]).                        (3)
pq(I,[_|Cs],[_|Us],[_|Ds]):-
        pq(I,Cs,Us,Ds).                         (4)
```

A typical initial query is $Q_{\text{in}} = pqs(n, q_0, \_, \_)$, where q_0 is a list of distinct variables, and n a natural number represented as $s^i(0)$. The program works on non-ground data.

We now show that the standard syntactic approaches to deal with avoiding the occur-check are inapplicable to NQUEENS. Under no moding the program is well-moded with Q_{in} because its answers are non-ground. To be tidy (or nicely moded with input linear clause heads), at most one position of pq is input (as (3) must be input linear). Thus at least three positions of pqs have to be output (as a variable from an input position of the head of (2) cannot appear in an output position of body atom $pq(s(I), Cs, Us, Ds)$). This makes the body not output linear, contradiction.

It can be shown that NQUEENS with Q_{in} is occur-check free under any selection rule, by showing that in all SLD-derivations each atom in each query is linear [Dra21b]. This is however rather tedious. The program is not occur-check free for some non linear queries, for instance for $A_{\text{STO}} = pq(m, L, [L|\_], \_)$ (where m is ground). This is because unifying A_{STO} with the unit clause (3) is not NSTO.

We now show that NQUEENS can be correctly executed without the occur-check, for a wider class of initial queries, including each $pqs(m, t_1, t_2, t_3)$ where m is ground. Let us say that a query Q is *1-ground* if the first argument of the predicate symbol in each atom of Q is ground. We show that:

Proposition 16 NQUEENS *is weakly occur-check free, under any selection rule, for any 1-ground query.*

PROOF Note first that each query in each SLD-derivation is 1-ground. Let $A = p(s_1, \ldots, s_4)$ be a 1-ground atom, and $H = pq(I, [I|\_], [I|\_], [I|\_])$ be the head of (3), standardized apart. Equation $s_1 \doteq I$ is NSTO and $\theta = \{I/s_1\}$ is its mgu. Let $s = (s_2, s_3, s_4)$ and $t = ([I|\_], [I|\_], [I|\_])$. As s_1 is ground, $t\theta$ is linear. Thus $s\theta \ddot{=} t\theta$ is NSTO by Lemma 1. Hence by Lemma 12, $s_1, s \ddot{=} I, t$ is WNSTO. So $A \doteq H$ is WNSTO. The cases of the remaining clause heads of NQUEENS are obvious, as the heads are linear. For another proof, see Examples 19, 21. □

By Corollary 15, NQUEENS with with any 1-ground query is correctly executed without the occur-check, under any selection rule.

NQUEENS may be considered a somehow unusual program. However similar issues appear with rather typical programs dealing with ground data. Assume, for instance, that data items from a ground data structure are to be copied into two data structures. Program

$$\text{USE2:} \quad p([X|Xs], f(X, Xs1), [g(X, \_)|Xs2]) \leftarrow p(Xs, Xs1, Xs2). \qquad p([], g, []).$$

provides a concise example. Similarly as for NQUEENS, it can be shown that USE2 is not occur-check free for some 1-ground queries, but is weakly occur-check free for all such queries.[3] For a quick proof see Ex. 19 or 21.

3.4 Sufficient conditions for WNSTO

Now we discuss sufficient conditions for safely avoiding the occur-check due to WNSTO. We assume that the programs dealt with are moded.

We say that a selection rule is **compatible with moding** (for a program P with a query Q) if (i) the input positions are ground in each selected atom in the SLD-tree for P with Q ([AL95] calls this "delay declarations imply the moding"), and (ii) some atom is selected in a query whenever the query contains

[3] USE2 is well-moded under $p(+, -, -)$, but the approach for well-moded programs does not apply, as the clause head is not output linear. In contrast to NQUEENS, USE2 can be treated as tidy, or nicely moded. The program is tidy under any moding with at most one position $+$. Hence it is occur-check free for tidy queries (they are a proper subset of 1-ground queries, and include all linear queries).

an atom with its input positions ground. Note that (i) implies that the selection rule is partial, in the sense that there exist nonempty queries in which no atom is selected. For such a query no resolvent exists, this is called *floundering* (or deadlock).

An atom A is **weakly linear** if any variable X which occurs more than once in A occurs in an input position of A. (Speaking informally, grounding the variables in the input positions of A results in a linear atom.)

Lemma 17 *Consider variable disjoint atoms A and H, such that the input positions of A are ground, and H is weakly linear. The unification of A and H is WNSTO.*

PROOF Let $A = p(s;t)$, where s is ground, and $H = p(s';t')$. Equation set $s \doteqdot s'$ is NSTO (by Lemma 1). Assume that $s \doteqdot s'$ is unifiable and that θ is an mgu of $s \doteqdot s'$. Thus $X\theta$ is ground for each variable $X \in Var(s')$. Hence $t'\theta$ is linear, and $(t \doteqdot t')\theta$ is NSTO (by Lemma 1). Now by Corollary 13, $A \doteqdot H$ is WNSTO. □

It immediately follows:

Corollary 18 *Let P be a program in which each clause head is weakly linear. If the selection rule is compatible with moding then P (with any query) is weakly occur-check free.*

Example 19 The heads of the clauses of NQUEENS are weakly linear under moding $pqs(+,-,-,-)$, $pq(+,-,-,-)$. By Corollary 18, the program (with any query) is weakly occur-check free under any selection rule compatible with moding. Consider a query Q which is 1-ground (cf. Section 3.3, p. 62). A simple check shows that in any SLD-derivation for NQUEENS and Q all queries are 1-ground. So each selection rule is compatible with moding (for NQUEENS with Q). Thus NQUEENS with any 1-ground query is weakly occur-check free for any selection rule. The same reasoning applies to USE2, with $p(+,-,-)$.

Now we provide a syntactic sufficient condition for a program to be weakly occur-check free. It employs a generalized notion of moding, in which some argument positions may be neither $+$ (input) nor $-$ (output), to such positions we assign $\perp$ (neutral). We will call it **3-moding** when it is necessary to distinguish it from a standard moding. We write $p(s;t;u)$ to represent an atom $p(t_1,\ldots,t_n)$ and to state that s (respectively t, u) is the sequence of terms in its $+$ ($-$, $\perp$) positions. The idea is to distinguish (as $+$ or $-$) some argument positions which, roughly speaking, deal with ground data. A syntactic sufficient condition will imply for LD-derivations that in each selected atom the input positions are ground.

By a **well-3-moded** program (or query) we mean one which becomes well-moded after removing the $\perp$ argument positions. For a direct definition, let a *defining occurrence* of a variable V in a clause $C = H \leftarrow Q$ be an occurrence of V in an input position of H, or in an output position in Q. Now C is *well-3-moded* when each variable V in an output position of H has its defining occurrence in C, and each occurrence of a V in an input position in Q is preceded by a defining occurrence of V in another literal of C [Dra87]. A query Q is well-3-moded when clause $p \leftarrow Q$ is. An equivalent definition can be obtained by an obvious adaptation of [Apt97, Def. 7.8]. Note that any query with its input positions ground is well-3-moded.

We now use the fact that well-3-moded programs/queries inherit the main properties of well-moded ones.

Lemma 20 *Let P and Q be well-3-moded.*

1. All queries in SLD-derivations of P with Q are well-3-moded.

2. For P with Q the Prolog selection rule is compatible with moding.

3. If each clause head in P is weakly linear then P with Q is weakly occur-check free under the Prolog selection rule (and any selection rule compatible with moding). Moreover, if no argument position is moded as output then P with Q is weakly occur-check free under any selection rule.

4. P with Q does not flounder under any selection rule compatible with moding.

PROOF 1. An SLD-resolvent of a well-3-moded query and a well-3-moded clause is well-3-moded. The proof is the same as that of the analogical property of well-moded queries and clauses [Apt97, Lemma 7.9]. 2. and 4. From 1. and the fact that the input positions of the first atom of a well-3-moded query are ground. 3. From 2. by Corollary 18. Additionally, under a 3-moding without $-$, all input positions in a well-3-moded query are ground. Thus each selection rule is compatible with moding and Corollary 18 applies. □

Example 21 Programs NQUEENS and USE2 are well-3-moded under $pqs(+,\bot,\bot,\bot)$, $pq(+,\bot,\bot,\bot)$, and $p(+,\bot,\bot)$; so is any 1-ground query. Their clause heads are weakly linear. (The same holds under the modings from Ex. 19.) Thus by Lemma 20.3, the programs are weakly occur-check free for 1-ground queries, under any selection rule. So we obtained by syntactic means the results of Ex. 19.

Example 22 Apt and Pellegrini [AP94] use program DERIVATIVE [SS94, Program 3.30] as an example for an approach combining those for well-moded and nicely moded programs. Here are representative clauses of the program (infix operators $\uparrow, *$ are used).

DERIVATIVE:
$$d(X,X,s(0)).$$
$$d(X\uparrow s(N), X, s(N)*X\uparrow N).$$
$$d(F*G,\ X, F*DG+DF*G) \leftarrow d(F,X,DF), d(G,X,DG).$$

A typical query is $d(e,x,t)$, where e,x are ground (e represents an expression, and x a variable), t is often a variable. Here moding $d(+,\bot,\bot)$ will be sufficient. Consider a query $Q = d(e_1,x_1,t_1),\ldots,d(e_n,x_n,t_n)$ where $e_1,\ldots,e_n$ are ground. DERIVATIVE and Q are well-3-moded moded under $d(+,\bot,\bot)$. Also, the clause heads are weakly linear. By Lemma 20.3, the program with Q is weakly occur-check free under any selection rule.

Alternatively, DERIVATIVE is tidy under $d(-,+,-)$. Hence, by Corollary 6, under any selection rule the program is occur-check free for tidy queries, including any linear queries.

[AP94] applied a combination of methods of well-moding and nice moding to show that DERIVATIVE is occur-check free for an atomic Q ($n = 1$) with ground e_1,x_1 and linear t_1, under LD-resolution. That result is subsumed by each of our two conclusions above. Surprisingly, a more general result can be obtained by a simpler approach from that work. Under $d(-,+,-)$ DERIVATIVE is nicely-moded and its clause heads are input linear. Thus it is occur-check free under LD-resolution for any nicely moded queries, this includes any linear queries.

In this section we dealt with clause heads whose certain instances are linear. Appendix B employs clauses whose certain instances are tidy, to construct another sufficient condition for weak occur-check freeness.

4 Comments

Let us first discuss briefly the limits of applicability of the presented results. The approaches discussed here are based on conditions imposed on clauses and queries. The conditions treat any predicate argument as a single entity, and refer to groundness or to placement of variables within certain argument positions.

This may not be sufficient when the occur-check depends on other features of the terms in argument positions. For instance, in the SAT-solver of Howe and King [HK12] an argument is a non-linear list of lists of pairs, and for occur-check freeness the first element of each pair should be ground [Dra18]. In such case our methods fail, and some semantic analysis is needed instead. One may expect that introducing a suitable type system could be useful.

Introduction of WNSTO has two consequences. Some cases where unification is not NSTO can actually be safely executed without the occur check. Also, reasoning based on WNSTO is sometimes simpler. For instance, showing that program NQUEENS is occur-check free was substantially more complicated than showing it to be weakly occur-check free for a wider class of queries.

Most of the employed sufficient conditions are based on the notion of modes. Examples show that modes (except for well-moded programs) do not need to correspond to any intuitive understanding of data flow. Instead, they deal with how variables are placed in argument positions. An output argument may well be used for input data. Neglecting this fact may be the reason why in some examples of [AP94] unnecessarily complicated methods were applied, or more general results could have been obtained. (For examples and explanations see the comments on FLATTEN and NORMALIZE in Section 2.2, and on DERIVATIVE in Ex. 22.)

Conclusions The main contribution of this paper is weakening the notion of NSTO (not subject to occur-check) used in the previous work on avoiding the occur-check. We generalize NSTO to WNSTO (weakly NSTO). This leads to a generalization of the notion of occur-check free programs/queries (based on NSTO) to *weakly occur-check free* ones (based on WNSTO). We proved that unification without the occur-check is sound for any input which is WNSTO. We presented a few sufficient conditions for WNSTO, and for a program/query being weakly occur-check free. Some conditions are syntactic, like Lemma 20, some refer to semantic notions, like Corollary 18 which explicitly refers to details of SLD-derivations. Additionally, we presented a sufficient condition based on NSTO, generalizing the approach based on nicely moded programs. Examples show that the proposed approach makes it possible to omit the occur-check in cases, to which the approaches based on NSTO are inapplicable. In some other cases, it leads to simpler proofs.

A Appendix. Proof of Lemma 12

The proof employs a technical lemma.

Lemma 23 *Consider a run R of MMA producing an mgu θ. Let $X_1 \doteq t_1, \ldots, X_k \doteq t_k$ (in this order) be the equations selected in action (5) in R, and $\gamma_i = \{X_1/t_1\}$ for $i = 1, \ldots, k$. Then $\theta = \gamma_1 \cdots \gamma_k$. Moreover, the last equation set of R is $\{X_i \doteq t_i\gamma_{i+1} \cdots \gamma_k \mid 0 < i \leq k\}$.*

PROOF Action (5) means applying γ_i to all the equations except for $X_i \doteq t_i$. Moreover, (a current instance of) $X_i \doteq t_i$ is not selected anymore in R. So θ contains a pair $X_i/t_i\gamma_{i+1} \cdots \gamma_k$. Let $\psi_j = \{X_i/t_i\gamma_{i+1} \cdots \gamma_j \mid 0 < i \leq j\}$ and $\varphi_j = \gamma_1 \cdots \gamma_j$, for $j = 0, \ldots, k$. Now $\psi_j = \varphi_j$, by induction on j (as $\{X_i/t_i\gamma_{i+1} \cdots \gamma_j \mid 0 < i \leq j\}\gamma_{j+1} = \{X_i/t_i\gamma_{i+1} \cdots \gamma_{j+1} \mid 0 < i \leq j\} \cup \gamma_{j+1}$). Thus $\theta = \varphi_k$. □

PROOF (of Lemma 12) Assume that E_1 is not unifiable. Then from a run R of MMA on E_1 one can construct in an obvious way a run R' of MMA on $E_1 \cup E_2$, performing the same actions and selecting the same equations. Thus action (6) is not performed in R'.

Now assume that θ_1 is an mgu of E_1. Consider a run R_1 of MMA on E_1, and a run R_2 of MMA on $E_2\theta_1$. Without loss of generality we may assume that θ_1 is the result of R_1. (Otherwise, R_1 produces θ such that $E_2\theta_1$ is a variant of $E_2\theta$.)

Let step (5) be applied in R_1 to equations $X_1 \doteq t_1, \ldots, X_k \doteq t_k$ (in this order), and in R_2 to $X_{k+1} \doteq t_{k+1}, \ldots, X_m \doteq t_m$ (in this order). Let $\gamma_i = \{X_i/t_i\}$, for $i = 1, \ldots, m$. Let F be the last equation set in R_1; by Lemma 23, $F = \{X_1 \doteq t_1\gamma_2 \cdots \gamma_k, \ldots, X_k \doteq t_k\}$.

To construct a run R of MMA on $E_1 \cup E_2$, for each equation set from R_1 or R_2 we construct a corresponding equation set from R.

Consider an equation set E in R_1. Let $X_1 \doteq t_1, \ldots, X_i \doteq t_i$ be the equations on which action (5) has been performed in R_1 until obtaining E ($i \in \{0, \ldots, k\}$). The equation set corresponding to E is $E' = E \cup E_2\gamma_1 \cdots \gamma_i$. In particular (by Lemma 23), $F \cup E_2\theta_1$ corresponds to the last equation F of R_1.

For $i \in \{k, \ldots, m\}$ let us define $F_i = \{X_l \doteq t_l\gamma_{l+1} \cdots \gamma_i \mid 0 < l \leq k\}$. Note that $F_k = F$. Consider an equation set E in R_2. Let $X_{k+1} \doteq t_{k+1}, \ldots, X_i \doteq t_i$ be the equations on which action (5) has been performed in R_2 until obtaining E ($i \in \{k, \ldots, m\}$). The equation set corresponding to E is $E' = F_i \cup E$.

Consider now the sequence consisting of the equation sets corresponding to those of R_1 and then of the equation sets corresponding to those of R_2 (without the first one, to avoid a repetition of $F \cup E_2\theta_1$). The sequence is a run of MMA on $E_1 \cup E_2$. As the run does not involve action (6), $E_1 \cup E_2$ is WNSTO. □

B Appendix. Another syntactic sufficient condition

Here we present a sufficient condition for avoiding the occur-check, related to WNSTO and based on the syntactic conditions for tidy programs.

Consider a 3-moding M and an additional moding M', for the latter we use symbols $+', -'$. Consider transforming each clause C of a program P, by grounding the variables that occur in the $+$ positions in the head. Let P' be the resulting program. Let us say that P is *weakly tidy* (under M, M') if P' is tidy under M'.

For an example, consider a program P containing a clause C with body $B = q(X,Y), q(Y,Z), q(Z,X)$. Assume also that P contains a clause head $H = q(t,u)$ with t, u containing a common variable. For P to be tidy, the argument positions of q cannot be both input (due to H), and cannot be both output (due to B). However, if they are $(+,-)$, or $(-,+)$ then $\rightarrow_B$ is cyclic. Thus P is not tidy (and not nicely moded) under any moding. Assume now that C is $p(X) \leftarrow B$. Under $p(+)$, and $q(+', -')$ the clause is weakly tidy. (A corresponding clause of P' is $p(s) \leftarrow q(s,Y), q(Y,Z), q(Z,s)$, for a ground term s; note that $q(+', -')$ may be replaced by $q(-', +')$.) By the lemma below, if $p(s)$ is selected in a tidy query Q then the resolvent of Q and C is tidy.

Lemma 24 *Let P be a weakly tidy program under M, M', and Q a query tidy under M'. Under any selection rule compatible with M*

each query in any SLD-derivation for P with Q is tidy under M', and

P with Q is weakly occur-check free.

PROOF Let A be the selected atom of a tidy (under M') query Q, and H be the head of a standardized apart clause C of P. Let $A = p(s;t;u)$ and $H = p(s';t';u')$ (under M). Unifying A with H can be divided in two steps: [Apt97, Lemma 2.24].

1. Unifying s with s'. As s is ground, $s \doteq\doteq s'$ is NSTO.

2. Unifying $(t, u \doteq\doteq t', u')\theta$ provided that $s \doteq\doteq s'$ is unifiable with an mgu θ. This is the same as unifying $A\theta \doteq H\theta$ (as $s\theta = s'\theta$ and is ground). Note that $Q\theta = Q$ (thus $A\theta = A$), and that $C\theta$ is tidy under M'. So $A\theta$ is an atom from a tidy query, and $H\theta$ is the head of a standardized apart tidy clause $C\theta$. By Corollary 6, $A\theta \doteq H\theta$ is NSTO.

Now by Corollary 13, $A \doteq H$ is WNSTO. If $A \doteq H$ is unifiable then, by Lemma 3, the resolvent Q' of $Q\theta$ and $C\theta$ is tidy. Q' is also the resolvent of Q and C.

We showed, for a tidy query Q and a standardized apart clause C of P, that unification of the selected atom of Q with the head of C is WNSTO, and that the resolvent (if it exists) of Q with C is tidy. The Lemma follows by simple induction. □

Consider P, Q satisfying the conditions of the Lemma. If P and Q are well-3-moded under M then P with Q is weakly occur-check free under Prolog selection rule (and under any selection rule compatible with M, under such rule it does not flounder).

References

[Aït91] H. Aït-Kaci (1991): *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, doi:10.7551/mitpress/7160.001.0001.

[AL95] K. R. Apt & I. Luitjes (1995): *Verification of Logic Programs with Delay Declarations*. In V. S. Alagar & M. Nivat, editors: *Algebraic Methodology and Software Technology, AMAST '95, Proceedings*, *Lecture Notes in Computer Science* 936, Springer, pp. 66–90, doi:10.1007/3-540-60043-4_47.

[AP94] K. R. Apt & A. Pellegrini (1994): *On the Occur-Check-Free Prolog Programs*. *ACM Trans. Program. Lang. Syst.* 16(3), pp. 687–726, doi:10.1145/177492.177673.

[Apt97] K. R. Apt (1997): *From Logic Programming to Prolog*. International Series in Computer Science, Prentice-Hall.

[Bra12] I. Bratko (2012): *PROLOG Programming for Artificial Intelligence*, 4th edition. Addison-Wesley.

[Col82] A. Colmerauer (1982): *Prolog and Infinite Trees*. In K. L. Clark & S.-Å. Tärnlund, editors: *Logic programming*, Academic Press, pp. 231–251.

[Cou83] B. Courcelle (1983): *Fundamental Properties of Infinite Trees*. *Theor. Comput. Sci.* 25, pp. 95–169, doi:10.1016/0304-3975(83)90059-2.

[CP94] R. Chadha & D. A. Plaisted (1994): *Correctness of Unification Without Occur Check in Prolog*. *J. Log. Program.* 18(2), pp. 99–122, doi:10.1016/0743-1066(94)90048-5.

[DFT91] P. Deransart, G. Ferrand & M. Téguia (1991): *NSTO Programs (Not Subject to Occur-Check)*. In V. A. Saraswat & K. Ueda, editors: *Logic Programming, Proceedings of the 1991 International Symposium*, MIT Press, pp. 533–547.

[Dra87] W. Drabent (1987): *Do Logic Programs Resemble Programs in Conventional Languages?* In: *Proceedings of 1987 Symposium on Logic Programming*, IEEE Computer Society Press, pp. 389–396. ISBN 0-8186-0799-8.

[Dra18] W. Drabent (2018): *Logic + control: On program construction and verification*. *Theory and Practice of Logic Programming* 18(1), pp. 1–29, doi:10.1017/S1471068417000047.

[Dra21a] W. Drabent (2021): *On correctness and completeness of an n queens program*. *Theory and Practice of Logic Programming*. To appear. Former version available at http://arxiv.org/abs/1909.07479.

[Dra21b] W. Drabent (2021): *SLD-resolution without occur-check, an example*. *CoRR* abs/2103.01911. Available at https://arxiv.org/abs/2103.01911.

[Frü91] Thom Frühwirth (1991): nqueens. A post in comp.lang.prolog. Available at https://groups.google.com/d/msg/comp.lang.prolog/qiyibDALhTE/uk6f6AQzOCAJ. 1991-03-08. Also in [SS94, Section 4.1, Exercise (v)].

[HK12] J. M. Howe & A. King (2012): *A Pearl on SAT and SMT Solving in Prolog*. *Theor. Comput. Sci.* 435, pp. 43–55. Available at http://dx.doi.org/10.1016/j.tcs.2012.02.024.

[SS94] L. Sterling & E. Shapiro (1994): *The Art of Prolog*, 2 edition. The MIT Press.