Localizations of Unification Freedom through Matching Directions

Massimo Marchiori
Department of Pure and Applied Mathematics
University of Padova
Via Belzoni 7, 35131 Padova, Italy
max@hilbert.math.unipd.it

Abstract

In this paper it is performed a thorough theoretical study of unification free logic programs, that is programs in which unification can be replaced by (iterated) matching. We introduce a global framework to study unification freedom, based on the simple concept of matching direction. Furthermore, we develop some syntactical criterions to ensure a program is unification free. For the first time, also optimality of such criterions is analyzed, using the so-called localization tool, that properly formalizes the concept of clause-by-clause criterion. This allows a very precise analysis: two main classes of programs are defined which are shown to be maximal, and one of them is proved to be the greatest subsuming the previous work on the subject. Moreover, these classes result to exhibit surprising connections with functional programs as well.

Note: This work was done during an author's stay at CWI, Amsterdam.

1 Introduction

Unification is the leading feature that distinguishes logic programming from other programming paradigms. In this paper we perform a complete theoretical study of syntactical criterions ensuring that unification is not actually needed, and can be safely replaced by a simpler operation like (iterated) pattern matching. The motivations for such a study are not only of theoretical interest, but also of practical nature: indeed, being unification the prime inferential engine of logic programs, knowing that its full power (and complexity) is not needed offers the possibility of improving the program efficiency. This, in our case, is supported by the fact that matching is faster that unification, especially using parallel implementations (see e.g. [DKM84]).

So far, finding when in a program unification is really needed or not has been the subject of a number of works, e.g. [MK85, DM85b, AFZ89] and, more recently, [AE93]. The new contributions of this paper are the following.

First, we present a global framework to study unification freedom (briefly UF): it is studied under under what conditions on one of the two terms involved in a unification problem we can force a matching, and what *direction* it has, that is where the substitution has to be applied; we so introduce the framework of the matching directions, and a related notion of well directness, that offer a way to express various kinds of unification freedom in logic programs.

Furthermore, we present some criterions to ensure UF that improve previous works: the major novelty is that, for the first time, we also address the optimality of such criterions. The somehow vague concept of 'syntactical criterion' is here given

precise meaning using the so-called localization framework: those criterions are studied that are *local* in the sense that they act *clause-by-clause*. It is so shown that among the various *localizations* of UF (i.e. the clause-by-clause criterions ensuring a program is UF) there are two main classes of Flatly Well Moded (FWM) and co-Flatly Well Moded (coFWM) programs that are *maximal*. Moreover, a hierarchy of sub-classes is defined that allows to measure the 'degree' of unification freedom allowed. All these subclasses enjoy a maximality property in the framework of the matching directions, and one of them is just the main class of Simply and Well Moded programs (SWM) found in [AE93]. The analysis is then strengthened till to prove also relative uniqueness of all the presented classes: in particular, the class FWM is proved to be the *greatest* localization of unification freedom containing SWM.

It is also illustrated, by means of simple (non clause-by-clause) syntactical criterions, how the power of FWM and coFWM can be combined together.

Finally, we hint at the work in [Mar94], that shows how the FWM and coFWM classes here presented can also be transformed into the most paradigmatic of the functional languages, namely a Term Rewriting System. This has important practical consequences, since all the techniques developed to analyze properties of the TRSs (like termination for instance) can so be applied to the programs in FWM and coFWM.
Summing up, the final upshot can be roughly synthesized in the following slogan:

$$\textsf{Logic Programs} - \textsf{Unification} \subseteq \textsf{Term Rewriting Systems}$$

The paper is organized as follows: after giving the necessary preliminaries in Section 2, the localization tool is explained in Section 3. In Section 4 matching directions are studied, and in Section 5 Well Directness is introduced. Section 6 then illustrates the classes FWM and coFWM, together with some examples of their use. Section 7 states the uniqueness results, and Section 8 concludes showing how to combine FWM and coFWM, and mentioning the relationships with Term Rewriting Systems. Due to absolute lack of space, proofs are omitted: they are present in the full version of this paper.

2 Preliminaries

In this paper we deal with so-called *LD-derivations* (briefly derivations), that is to say SLD-derivations in which the leftmost selection rule is used.
The symbols TERM and GROUND stand for the set of all the terms of the given language and its subset of ground terms. Accordingly with [Llo87], given a clause $C = A \leftarrow B_1, \ldots, B_n$ then $C^+ \equiv A$ and $C^- \equiv B_1, \ldots, B_n$. Sequences of terms are written in vectorial notation (e.g. $\bar{t}$), and with $t^{(i)}$ we denote the i-th term in $\bar{t}$.

Given a family S of objects (terms, atoms, etc.), $\mathrm{Var}(S)$ is the set of all the variables contained in it; moreover, S is said *linear* if no variable occurs more than once in it; objects $O_1, \ldots, O_n$ are said *disjoint* if $1 \leq i < j \leq n \Rightarrow \mathrm{Var}(O_i) \cap \mathrm{Var}(O_j) = \emptyset$.

As far as substitutions are concerned, if $\vartheta = \{X_1/t_1, \ldots, X_n/t_n\}$ then $\mathrm{Dom}(\vartheta) \equiv \{X_1, \ldots, X_n\}$ and $\mathrm{Ran}(\vartheta) \equiv \{t_1, \ldots, t_n\}$. Substitution composition of ϑ_1 and ϑ_2 (and in general map composition) is denoted by $\vartheta_1 \cdot \vartheta_2$.
A substitution ϑ is said a *match* for two terms s and t if

$$(s = \vartheta t \wedge \mathrm{Dom}(\vartheta) \subseteq \mathrm{Var}(t)) \vee (t = s\vartheta \wedge \mathrm{Dom}(\vartheta) \subseteq \mathrm{Var}(s))$$

We write $s \leq t$ for $\exists\vartheta : s = t\vartheta$. $s\mathcal{U}t$ means that s and t are disjoint and unifiable.

As usual, we henceforth identify a program property with the class of programs that satisfy it, and write $\mathcal{P} \leq \mathcal{Q}$ for $\mathcal{P} \subseteq \mathcal{Q}$. Also, the following important property will reveal useful:

Definition 2.1 A property $\mathcal{P}$ is *persistent* if for very goal G and clause C that have a resolvent G', $G \in \mathcal{P}, C \in \mathcal{P} \Rightarrow G' \in \mathcal{P}$. □

All the properties hereafter mentioned will be persistent.

2.1 Programs and Regularity

We call (logic) program a finite set of clauses and goals. The class of these 'extended' logic programs will be denoted by PROG.
Let P be a program, P_1 be the greatest subset of P containing no goals, and P_2 the set of all the goals in P. We call *derivation of* P a derivation of $P_1 \cup \{G\}$, when G is in P_2. Analogously, we call *computed answer substitution of* P a computed answer substitution of $P_1 \cup \{G\}$, where G is in P_2.

This formalism is well suited when formulating program properties, since it allows not to treat separately the clauses and the goal. Moreover, it is a natural choice when using *regular* properties, that essentially do not distinguish between goals and clauses:

Definition 2.2 A property $\mathcal{P}$ is said *regular* if

$$P \cup \{\leftarrow B_1, \ldots, B_n\} \in \mathcal{P} \Leftrightarrow P \cup \{T \leftarrow B_1, \ldots, B_n\} \in \mathcal{P}$$

where T is a new nullary predicate symbol. □

Using regularity will prove useful to elegantly shorten the description of the properties we will use in this paper, like well modedness for example. Although this will not concern us, it is worthwhile to notice that regularity is not merely a 'syntactic sugaring' tool, but has an importance for its own (see the discussion in [Mar93]).

2.2 Well and Simply Moded Programs

Definition 2.3 A *mode* for a n-ary predicate p is a map from $\{1, \ldots, n\}$ to $\{\text{in}, \text{out}\}$. A *moding* is a map associating to every predicate p a mode for it. □

Multiple modings can be achieved simply renaming the predicates. An argument position of a moded predicate is said input (resp. output) if it is mapped by the mode into in (resp. out). For every predicate p, the number of arguments (i.e. the arity), input and output positions is denoted respectively with $\#p$, $\#_{\text{in}}p$ and $\#_{\text{out}}p$. From now on, m always denotes the fixed moding we are working with: often, especially when used as subscript, it will be omitted and considered understood. Moreover, we adopt the convention to write $p(\bar{s}, \bar{t})$ to denote a moded atom p having its input positions filled in by the sequence of terms $\bar{s}$, and its output positions filled in by $\bar{t}$.

Definition 2.4 A program has the (regular) property of *well modedness (WM)* if for every its clause $C = p_0(\bar{t}_0, \bar{s}_{n+1}) \leftarrow p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$ it holds

$$\forall i \in [1, n+1] : \text{Var}(\bar{s}_i) \subseteq \bigcup_{j=0}^{i-1} \text{Var}(\bar{t}_j)$$ □

WM is a persistent property too ([DM85a]).

Definition 2.5 A program is *data driven* if every time an atom is selected in a goal during a derivation, its input arguments are ground. □

The following well-known result holds:

Theorem 2.6 ([DM85a]) *Well Moded programs are data driven.*

Definition 2.7 A program has the (regular) property of *Simply Modedness (SM)* if for every its clause $C = p_0(\bar{t}_0, \bar{s}_{n+1}) \leftarrow p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$ it holds

1. The sequence $\bar{t}_1, \ldots, \bar{t}_n$ is linear, composed only by variables and disjoint with $\bar{t}_0$
2. $\forall i \in [1,n]\ \mathrm{Var}(\bar{s}_i) \cap \bigcup_{j=i}^{n} \mathrm{Var}(\bar{t}_j) = \emptyset$ □

SM is a persistent property (see [AE93]). We call *simply well modedness (SWM)* the property intersection of SM and WM. This is the main class proved in [AE93] to be unification free. The intuition besides this class is that when an atom is selected in a goal during a derivation, its input arguments are ground for Theorem 2.6, and the output arguments are forced to be 'fresh' variables by the SM condition. This, as noticed by the same authors of [AE93], obviously limits the applicability of this class, since in a number of natural programs the output positions are filled in with compound terms.

2.3 Unification Freedom

When two atoms $p(\bar{s})$ and $p(\bar{t})$ need to be unified during the resolution process, unification can be performed componentwise in the following sense (here $mgu(s,t)$ denotes a relevant mgu of s and t): if $\bar{s} = s^{(1)}, \ldots, s^{(n)}$ and $\bar{t} = t^{(1)}, \ldots, t^{(n)}$, then an *iterated unification* of $\bar{s}$ and $\bar{t}$ is an $n+1$-tuple $(\pi, \vartheta_1, \ldots, \vartheta_n)$ where π is a permutation of $\{1, \ldots, n\}$ and $\forall k \in [1,n]\ \vartheta_k = mgu(s^{(\pi(k))}\vartheta_1 \cdots \vartheta_{k-1}, t^{(\pi(k))}\vartheta_1 \cdots \vartheta_{k-1})$.
It is well known that $\vartheta_1 \cdots \vartheta_n$ is a relevant mgu for $\bar{s}$ and $\bar{t}$, hence this is a correct way to perform unification. The idea is so to replace in this process unification with matching: more precisely, an *iterated matching* for $\bar{s}$ and $\bar{t}$ is an $n+1$-tuple $(\pi, \vartheta_1, \ldots, \vartheta_n)$ where π is a permutation of $\{1, \ldots, n\}$ and $\forall k \in [1,n]\ \vartheta_k$ is a match for $s^{(\pi(k))}\vartheta_1 \cdots \vartheta_{k-1}$ and $t^{(\pi(k))}\vartheta_1 \cdots \vartheta_{k-1}$.
Since a match is also a relevant mgu, when an iterated matching is possible it can replace the unification process during the resolution steps of a derivation. This is precisely the sense of the following definition:

Definition 2.8 A program is *unification free* if in every its derivation whenever a clause with head $p(\bar{s})$ is selected to resolve the atom $p(\bar{t})$ in the goal then there is an iterated matching of $\bar{s}$ and $\bar{t}$. □

The property of Unification Freedom will be denoted by UF.

3 Localizations

All the concepts in this section are part of a general framework developed in [Mar93]. Here we briefly expose only the specialized concepts that we need in the paper.

The first key definition formalizes the intuitive concept of 'clause-by-clause' property (and two weaker properties as well):

Definition 3.1 A property $\mathcal{P}$ is said *local* if

$$\forall P, P' :\ P \in \mathcal{P}, P' \in \mathcal{P} \Leftrightarrow P \cup P' \in \mathcal{P}$$

If '$\Leftrightarrow$' is replaced by $\Rightarrow$ (resp. $\Leftarrow$), then $\mathcal{P}$ is said *modular* (resp. *dense*). □

The following easy lemma states an important feature of local properties:

Lemma 3.2 *If $\mathcal{P}$ and $\mathcal{Q}$ are local, then $\mathcal{P} \cap \mathcal{Q}$ is local.*

Not all the properties we want to study are local, of course, but we may want to see what we can describe of a general property using only local analysis (that is, local properties):

Definition 3.3 $\mathcal{Q}$ is a *localization* of $\mathcal{P}$ if $\mathcal{Q}$ is local and $\mathcal{Q} \leq \mathcal{P}$. □

Moreover, we can perform local analysis also relatively to another local property:

Definition 3.4 If $\mathcal{R}$ is local, then $\mathcal{Q}$ is said a *localization of $\mathcal{P}$ inside $\mathcal{R}$* if $\mathcal{Q}$ is a localization of $\mathcal{P}$ and $\mathcal{Q} \leq \mathcal{R}$. □

Notice that WM and SM are local, and hence SWM is local too for Lemma 3.2.

The importance of the localization framework lies in its ability to formally state when a criterion is optimal, i.e. being a maximal localization it cannot be improved. Indeed, a paradigmatic example is the following

Theorem 3.5 ([Mar93]) WM *is a maximal localization of data driveness.*

Thus maximal localizations show to be quite a meaningful tool in analyzing local properties: as seen, they allow to give new meanings to the well modedness criterion formally proving that it enjoys an optimality property.

Now let us return to the UF property: it is easy to prove that UF is dense, but not local. Hence it makes sense to study UF from the point of view of local analysis, i.e. to search for (maximal) localizations of Unification Freedom: this is what we will do in this paper.

4 Matching Directions

Taken two sets of terms T_1 and T_2 then, for every couple of elements $s \in T_1$ and $t \in T_2$, we could have that the general case $s\mathcal{U}t$ is in fact $s \leq t$. We call *directional* matching the case in which unification reduces to matching and its direction is determined, that is $s\mathcal{U}t \Rightarrow s \leq t$ or $s\mathcal{U}t \Rightarrow t \leq s$.
An interesting phenomenon is that a *nondirectional* matching is possible too, that is to say it can be known in advance that unification will collapse into a matching but without knowing in what direction.
Formally, this means $s\mathcal{U}t \Rightarrow (s \leq t \ \vee \ t \leq s)$ but $s\mathcal{U}t \not\Rightarrow s \leq t$ and $s\mathcal{U}t \not\Rightarrow t \leq s$.

Trying to impose some conditions on logic programs such to obtain matching instead of unification, we are so lead to seek what conditions are needed on the two sets T_1 and T_2 (i.e. what properties the corresponding terms have to enjoy) to fall in one of the cases mentioned above. A choice is to start from a given T_1 and search for the corresponding T_2 to ensure the matching. As said, three situations are possible.

4.1 Three Kinds of Matchings

The following definition is from [AE93]:

Definition 4.1 Let T a set of terms; a term t is said to be a *generic expression* for T if for every $s \in T$ $s\mathcal{U}t$ implies $s \leq t$. □

The set of all the generic expressions for a given set T is indicated as GE(T).

Dually, the following concept can be defined:

Definition 4.2 Let T a set of terms; a term t is said to be a *particular expression* for T if for every $s \in T$ $s\mathcal{U}t$ implies $s \geq t$. □

The set of all the particular expressions for a given set T is indicated by $\mathrm{PE}(T)$.

The last case to be analyzed is the generalization of the previous two:

Definition 4.3 Let T a set of terms; a term t is said to be a *flattening expression* for T if for every $s \in T$ $s\mathcal{U}t$ implies $s \leq t$ or $s \geq t$. □

The set of all the flattening expressions for a given set T is indicated with $\mathrm{FE}(T)$.

Of course $\mathrm{FE}(T) \supseteq \mathrm{GE}(T) \cup \mathrm{PE}(T)$ but, as hinted above, the superset relation can be *proper*, as we'll see in the next subsection.

4.2 The General Case

Now we are going to calculate the previous sets in the most general case, that is when the set T is just the whole set of the terms (viz. TERM). In this case we often talk about generic/particular/flattening expressions only.

Theorem 4.4 $\mathrm{PE}(\text{TERM}) = \text{GROUND}$

Therefore the *groundness* condition for a term can be seen in a natural way as a matching directional property.

Theorem 4.5 $\mathrm{GE}(\text{TERM}) = \text{VAR}$

Thus even the *variable* information can be naturally seen as a directional property.

The last, and more interesting, case to consider is $\mathrm{FE}(\text{TERM})$: calculating what this set is will also show, as stated previously, that $\mathrm{FE}(T)$ is in general different from $\mathrm{GE}(T) \cup \mathrm{PE}(T)$.

First we introduce an important class of terms:

Definition 4.6 A term t is said to be *steady* if it is linear and $t = f_1 \cdots f_k(X_1, \ldots, X_n)$ with $k > 0$, $\#f_1 = \ldots = \#f_{k-1} = 1 \leq \#f_k$, $X_i \in \text{VAR}$ □

Example 4.7 $f(g(h(X,Y)))$ is steady, whereas $f(X,X)$, $f(g(X),Y)$ and $f(g(X),h(Y))$ are not. □

We can so state what the flattening expressions are:

Theorem 4.8 $\mathrm{FE}(\text{TERM}) = \text{VAR} \cup \{t : t \text{ is steady}\} \cup \text{GROUND}$

5 Well Directness

5.1 Argument Maps

We want to study property of predicates, that is to say, ultimately, of their term arguments. It is therefore natural to utilize a formal framework allowing to express what properties we are talking about. For this purpose, we introduce the notion of argument map ($\times$ denotes the cartesian product):

Definition 5.1 Given a poset $(S, \sqsubseteq_S)$, the set $am(S)$ of the S *argument maps* is the set of all the maps τ associating to every predicate symbol p an element $\tau(p)$ in $S^{\#p}(\equiv \times_{i=1}^{\#p} S)$. □

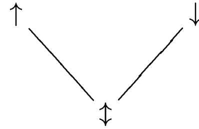
To be short, an S argument map is indicated simply as an S a-map.
Note how the a-maps interact well with the regular properties (see Def. 2.2) of logic programs, since the "new" predicate T, having null arity, must be automatically mapped in S^0, and so there is no ambiguity.

The argument maps can be clearly considered as posets: given two S a-maps τ_1 and τ_2, then define $\tau_1 \sqsupseteq \tau_2$ if for every predicate p $\tau_1(p) \sqsupseteq \tau_2(p)$.

Example 5.2 Taken the poset **IO** $= \{\text{in}, \text{out}\}$ (ordered as an antichain), then its **IO** a-maps are precisely the modings (moreover, it can be taken a preorder in $\sqsubseteq$ out, like in [Red86] for instance). □

5.2 Directions

So if we want to study the various kinds of matching, we can use an a-map over the poset of the *directions* $\mathbf{D} = \{\uparrow, \downarrow, \updownarrow\}$ with the following order:



where $\uparrow$, $\downarrow$ and $\updownarrow$ are labels indicating respectively particular, generic and flattening expressions.
Of course the **D** poset is not sufficient alone: to have a complete description of the matching directions in the (S)LD resolution we have to indicate what of the two involved terms the direction property is referred to: the one in the head of the clause or the one in the body of the goal.
To this purpose, the *head-body* poset $\mathbf{HB} = \{\mathrm{H}, \mathrm{B}\}$ is introduced, ordered as an antichain. Combining these two pieces of information is now an easy matter, simply taking the poset $\mathbf{HBD} = \mathbf{HB} \times \mathbf{D}$.

To be able to describe all logic programs, and not only the unification free ones, **HBD** can be lifted with a bottom element $\mathcal{U}$ (indicating unification). This way the complete description of the matching properties of a predicate can be given with a-maps over the lifted poset $\mathbf{HBD}_{\mathcal{U}}$ (see Figure 1). To simplify the notation,

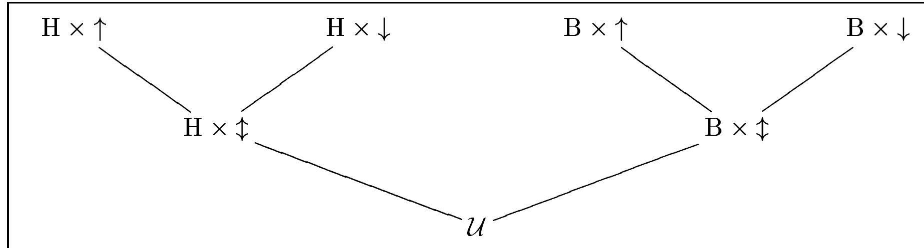


Figure 1: The $\mathbf{HBD}_{\mathcal{U}}$ poset.

elements like $\mathrm{H} \times \uparrow$ will be most of the times indicated as $\mathrm{H}\uparrow$.
Definition 5.3 The **HBD** a-maps are said to be *(head-body) matching directions*, or simply *directions*. The $\mathbf{HBD}_{\mathcal{U}}$ a-maps are said to be *partial (head-body) matching directions*, or *partial directions* for short. □

5.3 Semantics

We have already specified what the intended meaning of a direction is: it has come the moment to formally set out its semantics.

Let τ be a direction; its (intended) semantics is so defined (here π^{-1} is the inverse of π):

For every predicate symbol p, whenever a clause with head $p(\bar{s})$ is selected to resolve the atom $p(\bar{t})$ in the goal, there is an iterated unification $(\pi, \vartheta_1, \ldots, \vartheta_n)$ such that $\forall i \in [1,n]$

- $\tau(p)^{(i)} = \mathrm{H} \times \uparrow$ (resp. $\mathrm{H} \times \downarrow$, $\mathrm{H} \times \updownarrow$) implies $s^{(i)}\vartheta_1 \cdots \vartheta_{\pi^{-1}(i)-1} \in \mathrm{PE}(\textsc{Term})$ (resp. $\mathrm{GE}(\textsc{Term})$, $\mathrm{FE}(\textsc{Term})$).
- $\tau(p)^{(i)} = \mathrm{B} \times \uparrow$ (resp. $\mathrm{B} \times \downarrow$, $\mathrm{B} \times \updownarrow$) implies $t^{(i)}\vartheta_1 \cdots \vartheta_{\pi^{-1}(i)-1} \in \mathrm{PE}(\textsc{Term})$ (resp. $\mathrm{GE}(\textsc{Term})$, $\mathrm{FE}(\textsc{Term})$).

Definition 5.4 Given a partial direction τ, each program respecting its intended semantics is called *well directed* w.r.t. τ. □

The class of all the programs well directed w.r.t. τ is indicated with WD_τ.

Thus, for the property enjoyed by generic/particular/flattening expressions, well directness (w.r.t. a direction) implies Unification Freedom. Furthermore, just like UF, it is not difficult to prove via counterexamples the following:

Lemma 5.5 *For every direction τ, WD_τ is a dense but not local property.*

The ordering induced on $am(\mathbf{HBD})$ from the poset ordering in $\mathbf{HBD}$ and the corresponding classes of well directed programs are related by the following result (here $\wp$ is the usual powerset operator):

Lemma 5.6 *The map* $\mathcal{E} = \begin{cases} (am(\mathbf{HBD}), \sqsubseteq) & \rightarrow & (\wp(\textsc{Prog}), \subseteq) \\ \alpha & \mapsto & \mathrm{WD}_\alpha \end{cases}$ *is an anti-embedding, that is it is injective and $\alpha \sqsubseteq \beta \Leftrightarrow \mathcal{E}(\alpha) \supseteq \mathcal{E}(\beta)$.*

5.4 Well Moding vs. Well Directness

The Well Moding concept finds a natural representation in the well directness framework. Indeed, recall Theorem 2.6 stating that well moded programs are data driven. This means that, when dealing with well moded programs, the terms in input positions have their matching direction determined, and so they do not give problems for unification freedom: only output arguments are without 'matching' control.
Hence, we can define, for every element $\alpha \in \mathbf{HBD}_{\mathcal{U}}$, the following poset map that expresses our knowledge about input arguments:

$$\begin{array}{rcl} \zeta_\alpha : \mathbf{IO} & \rightarrow & \mathbf{HBD}_{\mathcal{U}} \\ \mathrm{in} & \mapsto & \mathrm{B}\uparrow \\ \mathrm{out} & \mapsto & \alpha \end{array}$$

Assumption: for simplicity, henceforth we write $\mathrm{WD}(\alpha)$ in place of $\mathrm{WD}_{\zeta_\alpha \cdot m}$ (recall m is the given moding).

In analogy with Theorem 3.5, we have:

Theorem 5.7 *Well Modedness is a maximal localization of* $\mathrm{WD}(\mathcal{U})$.

6 Localizations of UF

6.1 Variables Occurrences

We need to distinguish between different *occurrences* of the same variable in a sequence of atoms, and so implicitly in a clause too, since we can view a clause C as the sequence C^+, C^-. This can be achieved in many ways, for example numbering

the occurrences of a variable from left to right in writing order: here we keep the treatment at an informal level.

Taken a part σ of a sequence of atoms S, we write $\mathrm{VarOcc}_S(\sigma)$ for the set of variables occurrences present in σ. For instance, if $C = p(g(X), Y) \leftarrow q(f(X, g(X))), r(Z, h(X, X, g(Y)))$ and the part σ is 'the second argument of the third atom', then $\mathrm{VarOcc}_C(\sigma) = \{$the 4th occurrence of X in C, the 5th occurrence of X in C, the 2nd occurrence of Y in $C\}$.

If L is a family of terms and S a sequence of atoms, then $\mathrm{Occ}_S(L) \equiv \{$occurrences of Λ in $S \mid \Lambda \in \mathrm{Var}(L)\}$. Via the map Occ, we can define the intersection of two sets $R \subseteq \mathrm{VarOcc}_S(S)$ and $L \subseteq \mathrm{Var}(S)$ as $R \cap L \equiv R \cap \mathrm{Occ}_S(L)$.

Definition 6.1 Taken a well moded clause
$C = p_0(\bar{t}_0, \bar{s}_{n+1}) \leftarrow p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$, then
$G^+(C) \equiv \mathrm{VarOcc}_C(\bar{s}_{n+1}) \cap \mathrm{Var}(\bar{t}_0) \qquad G^-(C) \equiv \bigcup_{0 \le i < j \le n}(\mathrm{VarOcc}_C(\bar{t}_j) \cap \mathrm{Var}(\bar{t}_i))$
$I^+(C) \equiv \mathrm{VarOcc}_C(\bar{s}_{n+1}) \setminus G^+(C) \qquad I^-(C) \equiv \bigcup_{i=1}^{n} \mathrm{VarOcc}_C(\bar{t}_i) \setminus G^-(C)$
Elements in $G^+(C)$ (resp. $G^-(C)$) are said to be *head* (resp. *body*) *ground variables* of C, whereas elements in $I^+(C)$ (resp. $I^-(C)$) are said to be *head* (resp. *body*) *inner variables* of C. □

These four sets readily make up a partition of the occurrences of the variables in output positions of the clause. The intuition is that from the well modedness condition we can infer that some variables will be surely instantiated to ground terms (hence the name ground variables), whereas for the others (the inner variables) we cannot have this certainty. For instance, take the well-moded clause $C = p(X, f(X), Y) \leftarrow p(X, Y, Z), p(Y, Z, Z)$ with moding $p(\mathrm{in}, \mathrm{out}, \mathrm{out})$; then $G^+(C) = \{$2nd occ. of $X\}$, $I^+(C) = \{$1st occ. of $Y\}$, $G^-(C) = \{$2nd and 3rd occ. of $Z\}$, $I^-(C) = \{$1st occ. of Z, 2nd occ. of $Y\}$.

Next step is to try to determine which argument will be (surely) instantiated to a ground term:

Definition 6.2 Taken a clause C, if an argument is ground or has all ground variables (resp. if it has all inner variables), it is said a *$\mathcal{G}$-argument* (resp. *$\mathcal{I}$-argument*). □

In general an argument cannot be classified as $\mathcal{G}$ or $\mathcal{I}$; to avoid this undesirable case the following definition (whose meaning will be clarified later) is introduced:

Definition 6.3 Taken a sequence $S = p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$ of atoms, and $L \subseteq \cup_{i=1..n} \mathrm{VarOcc}_S(\bar{t}_i)$, L is said to be *tough w.r.t.* S if

$$\forall i \in [1, n], j \in [1, \#_{\mathrm{out}} p_i] : \ L \cap \mathrm{VarOcc}_S(t_i^{(j)}) \neq \emptyset \Rightarrow L \supseteq \mathrm{VarOcc}_S(t_i^{(j)}) \qquad \square$$

6.2 The class coFWM

Recall the discussion in Subsection 5.4: well modedness ensures that, to have UF via matching directions, it suffices to constrain the output arguments only. A possible choice is to impose a direction on the heads of the clauses, that is a direction having as image in the poset $\mathbf{HBD}_{\mathcal{U}}$ elements of the form $\mathrm{H} \times \ldots$ (review Figure 1). The least restrictive condition we can require is hence, by Theorem 5.7 and Lemma 5.6, $\mathrm{WD}(\mathrm{H}\updownarrow)$. So let us define:

Definition 6.4 The (local and regular) property of *co-Flatly Well Modedness* (coFWM) for a clause C is

1. $C \in \mathrm{WM}$

2. $G^+(C)$ is tough w.r.t. C^+
3. Every $\mathcal{I}$-argument of the head is a flattening expression
4. The $\mathcal{I}$-arguments of the head are disjoint $\square$

Notice how saying that a clause $C \in$ WM satisfies point 2 is the same than saying we can separate without ambiguities the output arguments of the predicates appearing in its head between $\mathcal{G}$ or $\mathcal{I}$-ones: indeed, if an output argument of C^+ has at least one head ground variable then by toughness of $G^+(C)$ w.r.t. C^+ it must have all of its variables in $G^+(C)$ (hence being a $\mathcal{G}$-argument). Then, point 3 ensures that all the output arguments whose 'groundness' cannot be inferred by the WM condition (i.e. the $\mathcal{I}$-arguments) belong to FE(TERM), and point 4 that they do not interfere among them.

Example 6.5 Consider the following program that finds the elements common to two lists:

$$\begin{array}{l} intersection(Xs, Ys, Z) \leftarrow member(Xs, Z), member(Ys, Z) \\ member([X], X) \leftarrow \\ member([X \mid Ys], Z) \leftarrow member(Ys, Z) \end{array}$$

with moding $intersection(\mathrm{in}, \mathrm{in}, \mathrm{out}), member(\mathrm{in}, \mathrm{out})$.
It belongs to coFWM, as it is easy to see, but it does not to SWM, since the body of the first clause has a double occurrence of Z in output positions. $\square$

Example 6.6 Take the following program to halve a number:

$$halve(X,Y) \leftarrow plus(Y,Y,X) \qquad \begin{array}{l} plus(0,X,X) \leftarrow \\ plus(s(X),Y,s(Z)) \leftarrow plus(X,Y,Z) \end{array}$$

with moding $halve(\mathrm{in}, \mathrm{out}), plus(\mathrm{out}, \mathrm{out}, \mathrm{in})$. It belongs to coFWM but not to SWM for the double occurrence of Y in the body of the first clause. $\square$

Example 6.7 Consider the program member defined using append ([SS86, p. 48]):

$$\begin{array}{l} member(X, Ys) \leftarrow append(As, [X|Xs], Ys) \\ append([\,], Ys, Ys) \leftarrow \\ append([X|Xs], Ys, [X|Zs]) \leftarrow append(Xs, Ys, Zs) \end{array}$$

with moding $member(\mathrm{in}, \mathrm{in}), append(\mathrm{in}, \mathrm{out}, \mathrm{in})$. It is in coFWM, but not in SWM due to the argument $[X|Xs]$ in the first clause. $\square$

Lemma 6.8 coFWM *is persistent.*

Then the following holds:

Theorem 6.9 *The class* coFWM *belongs to* WD(H$\updownarrow$).

This gives right away (since WD(H$\updownarrow$) $\leq$UF)

Corollary 6.10 coFWM *is a localization of Unification Freedom.*

But far more is provable:

Theorem 6.11 *The class* coFWM *is a maximal localization of* UF *inside* WM

6.3 The class FWM

The 'dual' choice to have UF via matching directions is to constrain the body in place of the head, that is to require WD(B$\updownarrow$). Now let us define the main class:

Definition 6.12 The (local and regular) property of *Flatly Well Modedness* (FWM) for a clause $C = p_0(\bar{t}_0, \bar{s}_{n+1}) \leftarrow p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$ with $\bar{s}_{n+1} = s^{(1)}_{n+1}, \ldots, s^{(k)}_{n+1}$ is

1. $C \in$ WM
2. $G^-(C)$ is tough w.r.t. C^-
3. Every $\mathcal{I}$-argument of the body is a flattening expression
4. The $\mathcal{I}$-arguments of the body are disjoint
5. $\forall i \in [1, k]$ $\mathrm{Occ}_{C^-}(s^{(i)}_{n+1})$ is tough w.r.t. C^- □

Points 1–4 are the 'dual' analogous to points 1–4 in the Definition 6.4 of coFWM (e.g. point 2 ensures we can separate without ambiguities the output arguments of the predicates appearing in its body between $\mathcal{G}$ or $\mathcal{I}$-ones). Moreover, point 5 is needed to constrict interferences that may come from the output arguments of the head via the resolution process: if an output argument in C^- has at least one variable in $s^{(i)}_{n+1}$, then all of its variables must stay in $s^{(i)}_{n+1}$.

Example 6.13 The program *intersection* of Example 6.5 is in FWM but, as said there, not in SWM. □

Example 6.14 Reconsider the program member defined using append (Ex. 6.7), but with moding *member*(out, in), *append*(out, out, in): this time it belongs to FWM but neither to coFWM (for the third clause) nor to SWM (for the first clause). □

Example 6.15 Take the program to flatten a list after [SS86, page 243]:

$$
\begin{array}{l}
\mathit{flatten}(Xs, Ys) \leftarrow \mathit{flatten}(Xs, [\,], Ys) \\
\mathit{flatten}([X|Xs], Zs, Ys) \leftarrow \mathit{flatten}(Xs, Zs, Ys1), \mathit{flatten}(X, Ys1, Ys) \\
\mathit{flatten}(X, Xs, [X|Xs]) \leftarrow \mathit{constant}(X), X \neq [\,] \\
\mathit{flatten}([\,], Xs, Xs) \leftarrow
\end{array}
$$

with moding *flatten*(in, in), *flatten*(in, in, out), *constant*(in), $\neq$(in, in). It belongs to FWM $\cap$ coFWM but not to SWM because of the first clause. In all similar is the program to flatten a list using difference-lists ([SS86, page 241]), with moding *flatten*(in, in). □

Lemma 6.16 FWM *is persistent.*

Then it holds:

Theorem 6.17 *The class* FWM *belongs to* WD(B$\updownarrow$).

Corollary 6.18 FWM *is a localization of Unification Freedom.*

As before for coFWM, much more is provable since FWM is maximal too:

Theorem 6.19 *The class* FWM *is a maximal localization of* UF *inside* WM

The great importance of the FWM class lies in the fact that it is the *best* (unification free and local) class that subsumes SWM: this will be better explained in the next section.

7 Uniqueness

7.1 A Hierarchy of Classes

The two classes coFWM and FWM previously presented resulted to be maximal localizations of WD(H$\updownarrow$) and WD(B$\updownarrow$) respectively.
What if we ask for more restrictive direction requirements, that is (following the **HBD** poset framework) for maximal localizations of, say, WD(H$\uparrow$), WD(B$\downarrow$) and so on? The answer is related to Definition 6.2, distinguishing between $\mathcal{G}$ and $\mathcal{I}$ (output) arguments for a clause: we could require the absence of one of the two kinds of arguments, $\mathcal{G}$ and $\mathcal{I}$. Thus we define two properties *no-ground-arguments in the head (*resp. *body)* NGA$^+$ (resp. NGA$^-$) and *no-inner-arguments in the head* (resp. *body*) NIA$^+$ (resp. NIA$^-$) which are still local. Now consider the intersection properties given by

IFWM	$\equiv$ NGA$^-$	$\cap$ FWM	coGFWM $\equiv$	NIA$^+$	$\cap$ coFWM
GFWM	$\equiv$ NIA$^-$	$\cap$ FWM	coIFWM $\equiv$	NGA$^+$	$\cap$ coFWM
NOUT	$\equiv$ GFWM	$\cap$ IFWM	coNOUT $\equiv$	coGFWM	$\cap$ coIFWM

(NOUT stands for No OUTput, since NOUT (resp. coNOUT) is easily seen to be the class having bodies (resp. heads) with no $\mathcal{G}$ or $\mathcal{I}$ output arguments). They are all local by Lemma 3.2, and so provide a *hierarchy* of local classes satisfying the following theorem:

Theorem 7.1 IFWM *(resp. GFWM) is a maximal localization of* WD(B$\downarrow$) *(resp.* WD(B$\uparrow$)*), whereas* coIFWM *(resp.* coGFWM*) is a maximal localization of* WD(H$\downarrow$) *(resp.* WD(H$\uparrow$)*).*

Inside this hierarchy, the class of simply well moded programs (SWM) of [AE93] can be given new significance, thanks to the following

Lemma 7.2 SWM = IFWM

Hence, the class of simply well moded programs is a maximal localization of WD(B$\downarrow$).

7.2 Relative Uniqueness

So far we managed to state maximality properties for localizations of UF. With the structure provided by the above seen hierarchy, even relative uniqueness for the major classes FWM and coFWM can be proved:

Theorem 7.3 FWM *is the greatest localization of UF inside* WM *containing* IFWM*(=*SWM*).*

Theorem 7.4 coFWM *is the greatest localization of UF inside* WM *containing* coGFWM

We will better explain the relevance of these theorems (especially of Theorem 7.3) in the next subsection, since both of them can still be sharpened.

These relative uniqueness results can be extended to the whole hierarchy:

Theorem 7.5 IFWM *(resp.* GFWM*) is the greatest localization of* WD(B$\downarrow$) *(resp.* WD(B$\uparrow$)*) containing* NOUT*, whereas* coIFWM *(resp.* coGFWM*) is the greatest localization of* WD(H$\downarrow$) *(resp.* WD(H$\uparrow$)*) containing* coNOUT.

Thus, using the equivalence IFWM = SWM of Lemma 7.2, we have got a meaningful characterization for the class of simply well moded programs.

7.3 Beyond Well Modedness

Theorems 7.3 and 7.4 express relative uniqueness of FWM and coFWM *inside the class of well moded programs*. However, these two classes are 'big enough' that much more holds since, surprisingly, the above restriction can be lifted:

Theorem 7.6 FWM *is the greatest localization of* UF *containing* IFWM *(*=SWM*).*

Theorem 7.7 coFWM *is the greatest localization of* UF *containing* coGFWM*.*

These results are tightly linked with the fact, expressed by Theorem 3.5, that Well Modedness is a maximal localization of WD($\mathcal{U}$).
The significance of the two above theorems is that optimality (i.e. maximality) and relative uniqueness of FWM and coFWM still hold even if we try to use non well-moded clauses. In particular, Theorem 7.6 states that if we want an UF local class (viz. a clause-by-clause criterion to ensure Unification Freedom) that contains all the programs in SWM, the best we can obtain is just FWM.

8 Remarks

8.1 Combining FWM and coFWM

The classes FWM and coFWM are complementary each other, since the first constrains the body of the (goal-)clause and the second the head. Thus it would be interesting to try to combine their power. However, being maximal localizations, they are already optimal, and so we cannot find a local class improving them. Hence it would seem that to infer unification freedom we must choose separately only one of the two classes. Nevertheless, there is a simple *global* (i.e. non local, see also [Mar93]) criterion that allows to extend the class FWM (resp. coFWM) utilizing resorts of the corresponding dual class coFWM (resp. FWM).

Call a program L a *library* for a program P if every predicate symbol appearing in L (briefly: library predicate) does not appear in the head of a clause in P. This means that P can make 'subroutine calls' to programs in L using library predicates in the bodies of the clauses, but not modify the definition of predicates in L. Then, under certain conditions, programs in FWM can utilize libraries of programs in coFWM, and vice versa:

Theorem 8.1 *Let L a library for P, and $L \in$ FWM, $P \in$ coFWM. Suppose every clause $C = p_0(\bar{t}_0, \bar{s}_{n+1}) \leftarrow p_1(\bar{s}_1, \bar{t}_1), \ldots, p_n(\bar{s}_n, \bar{t}_n)$ satisfies the following condition: if p_i $(1 \leq i \leq n)$ is a library predicate, 1) $G^-(C)$ and $I^-(C)$ are tough w.r.t. $\bar{t}_i$ 2) Every $\mathcal{I}$-argument of $\bar{t}$ is a flattening expression 3) The $\mathcal{I}$-arguments of $\bar{t}_i$ are disjoint 4) $\bar{s}_{n+1}$ is disjoint with $\bar{t}_i$. Then $P \cup L$ is unification free.*

Notice how the condition imposed is, to a certain extent, the FWM property applied only to the output arguments of p_i (namely, points 2–5 of Definition 6.12).

Example 8.2 Consider the program quicksort using difference-lists after [SS86, page 244], with moding *quicksort_dl*(in, in, out). That program is neither in FWM nor in coFWM. However, regard the *partition* program as a library for the program consisting of the other clauses: they satisfy the above Theorem 8.1, and hence the whole program is unification free. □

The second dual case is even more interesting:

Theorem 8.3 *Let L be a library for P, and $L \in$ coFWM, $P \in$ FWM. Then $P \cup L$ is unification free.*

Hence coFWM programs are particularly nice in that they can be freely used, without any restriction, as libraries for FWM programs.

Example 8.4 Member defined using append (see Example 6.7) can be safely used in FWM programs: this because its two possible modings, *member*(in, in) and *member*(out, in), give programs in coFWM (cf. Example 6.7) and FWM (cf. Example 6.14) respectively. □

Observe that the above theorems could also be easily extended to allow nested libraries (i.e. libraries containing libraries, etc.).

8.2 Term Rewriting Systems

Finally, let us conclude mentioning a surprising result that shows another important aspect of the two found maximal classes. As unification is the prime inferential engine for logic programming, matching is it for functional programming, in particular for the paradigm of Term Rewriting Systems (see [Klo92]).
In [Mar94] a theory of transforms from logic programs to TRSs is presented. Of particular interest are transform that are *complete*, in the sense that: 1) there is a one-to-one correspondence between computed answer substitutions of the logic programs and normal forms of the corresponding TRS, and 2) the transform preserves termination (i.e. a logic program terminates iff the corresponding TRS terminates). It is there shown that the classes FWM and coFWM have a transform which is *complete* and also *compositional*, and hence programs in these classes can be considered as TRSs in disguise. This result, joined with Theorems 7.6 and 7.7, can be roughly summarized in the slogan, already seen in the introduction,

$$\text{Logic Programs} - \text{Unification} \subseteq \text{Term Rewriting Systems}$$

This is important not only from a theoretical point of view, but also practically: a complete transform allows to infer properties of a logic program from properties of the corresponding TRS. Indeed, the main stimulus in the study of complete transforms was the study of termination (see [Mar94] for more references on this topic): checking whether the transformed TRS terminates provides a *sufficient* and *necessary* criterion for termination of logic programs. This approach is quite attractive because TRSs enjoy powerful techniques to prove termination, like path orderings for instance (see [Der87]), whereas for logic programs the situation is far more complicated.

Acknowledgements

I wish to heartly thank Krzysztof R. Apt and Jan Willem Klop for their support.

References

[AE93] K.R. Apt and S. Etalle. On the unification free Prolog programs. In S. Sokołowski, editor, *Proceedings of Conference on Mathematical Foundations of Computer Science (MFCS)*, LNCS, pages 1–19. Springer–Verlag, 1993.

[AFZ89] I. Attali and P. Franchi-Zannettacci. Unification-free execution of TYPOL programs by semantic attribute evaluation. In R.A. Kowalski and K.A. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 160–177. The MIT Press, 1989.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[DKM84] C. Dwork, P.C. Kanellakis, and J.C. Miller. On the sequential nature of unification. *Journal of Logic Programming*, 1(1):35–50, 1984.

[DM85a] P. Dembinski and J. Małuszyński. 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. Małuszyński. Relating logic programs and attribute grammars. *Journal of Logic Programming*, 2:119–156, 1985.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Clarendon Press, Oxford, 1992.

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

[Mar93] M. Marchiori. Local analysis and localizations. Manuscript, June 1993. Extended and revised version to appear as Technical Report, Dep. of Pure and Applied Mathematics, University of Padova, 1994.

[Mar94] M. Marchiori. Logic programs as term rewriting systems. In *Proceedings of the Third International Conference on Algebraic and Logic Programming*, LNCS. Springer–Verlag, 1994. Extended version forthcoming as technical report.

[MK85] J. Małuszyński and H.J. Komorowski. Unification-free execution of logic programs. In *Proceedings of the 1985 IEEE Symposium on Logic Programming*, pages 78–86, Boston, 1985. IEEE Computer Society Press.

[Red86] U.S. Reddy. On the relationships between logic and functional programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 3–36. Prentice–Hall, Englewood Cliffs, New Jersey, 1986.

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