INFORMATIK
BERICHTE

378 – 06/2019

On separation, conservation and unification

Marija Kulaš

Fakultät für Mathematik und Informatik
D-58084 Hagen

On separation, conservation and unification

Marija Kulaš
FernUniversität in Hagen, Wissensbasierte Systeme, 58084 Hagen, Germany
kulas.marija@online.de

Based on variable substitution, we discuss a practical aspect of Prolog computation, algorithms for unification and standardizing-apart. The focus is on simplicity and desirable mathematical properties. For unification, we start from a deterministic version of Martelli-Montanari's algorithmic scheme that also happens to correspond to Robinson's algorithm (hence, "the" classical unification algorithm) and discuss two of its adaptations, concerned with modularity or with ordering of variables. Standardizing-apart is defined as a special case of (relatively) fresh renaming. Examples for both kinds of algorithms are given, and some properties are discussed.

As a common ground, the issue of variable separation (i.e. variable-disjointness between parts of terms) is raised. Variable conservation, important for formal semantics, is also approached.

1 Introduction

In logic programming literature, one often reads about "the" most general unifier (*mgu*) of an equation E, even though there must be infinitely many of those, if any. This is usually justified by pointing out that all mgus of E are "equal modulo renaming"[1]. A justification of a more practical kind is brought about by implementation: in a Prolog interpreter, unification is performed by an algorithm, $\boldsymbol{U}$, so "the" mgu is the one produced by $\boldsymbol{U}$. Similarly, even though a program clause can (in theory) be renamed every which way, in practice this is also performed by an algorithm, $\boldsymbol{S}$.

The main concern of this paper is to present a few simple $\boldsymbol{U}$ and $\boldsymbol{S}$, and contribute toward discussion about their effect on Prolog computations and their desirable properties. Regarding effect, it is well known that restrictions imposed by $\boldsymbol{U}$ and $\boldsymbol{S}$ are not compromising soundness and completeness of SLD-resolution. Yet, there may be "lowlier" claims which more or less implicitly rely on free choice of mgus and renaming. One such claim states that an SLD-derivation can be renamed "wholesale" (the resolvents, the mgus, the input clauses). It does not hold in the presence of $\boldsymbol{S}$, as shall be shown. This is important for comparisons of SLD-derivations in the context of Prolog. The reason for impossibility of wholesale renaming is that no $\boldsymbol{S}$ can be *compatible with renaming*, unlike $\boldsymbol{U}$. Of attainable properties for $\boldsymbol{S}$, we advocate *structure-independence*, necessary for handling parts of derivation. For $\boldsymbol{U}$, apart from said *renaming-compatibility*, we found *relevance* to be necessary. Relevance of $\boldsymbol{U}$ means that the mgu produced has no extraneous variables, so $\left(\begin{smallmatrix}\mathtt{X} & \mathtt{Z}\\ \mathtt{Y} & \mathtt{X}\end{smallmatrix}\right)$ cannot be "the" mgu for `p(X)=p(Y)`. Apart from necessary qualities of $\boldsymbol{U}$, we also look into assets like *modularity* or *bias*.

As a basis, technical issues of *variable separation* and *variable conservation* shall be aproached. Variable separation can happen in SLD-derivations in many guises, like standardizing-apart of input clauses, idempotence of mgus or as (weaker or stronger) separation between mgus. Claims about variable conservation are necessary for correspondence proofs in detailed operational models of Prolog, where all variables need to be accounted for.

[1] A somewhat misleading phrase, suggesting that they "look the same but with other variables", which is not the case. More precisely, any two mgus of E are *equigeneral*, see Subsection 1.2 and Legacy 1.3. Example: $\left(\begin{smallmatrix}\mathtt{X} & \mathtt{Y} & \mathtt{Z}\\ \mathtt{Y} & \mathtt{Z} & \mathtt{X}\end{smallmatrix}\right)$ and $()$ are equigeneral and both are mgus for $t = t$, where t can be any term.

Put into a broader perspective, such questions are important for any detailed study of Prolog computation (our original motivation), and likewise for its further development. Together with a note on renaming [11] (currently undergoing revision, to include a more general kind of safety) and a note on substitution generality [12], we hope to contribute toward a sound yet gentle exchange between the occasionally puzzling mathematical realm of variable substitutions, Horn clause logic and its proof theory on the one hand, and the pragmatic issues of logic programming languages like Prolog on the other hand.

1.1 Overview of the paper

In Section 2 we handle *variable separation* within or between substitutions. Of the latter type, in Subsection 2.2 two kinds are considered. The stronger kind is named *cascading* and it makes its appearance in Martelli-Montanari-type unification algorithms and in SLD-derivations. The weaker kind is core-disjointness, and plays a rôle in *conserving the set of variables* when composing substitutions. A few possibilities for variable conservation are handled in Section 3. Both separation and conservation can occur in SLD-derivations, as shown later in Subsection 5.1.

In Section 4, we start from Martelli-Montanari's unification scheme and its deterministic version MM. Two adaptations are discussed: a modular rephrasing of the binding rule (Subsection 4.2), and addition of variable order (Subsection 4.3). The latter results in algorithm MMB, and the former in modular versions of respectively MM and MMB, denoted as RMM and RMMB. Some desirable properties of unification algorithms are discussed in Subsection 4.4.

In Section 5, the concept of algorithm for fresh renaming (relative to a set of variables) is defined, and the questions of its *structure-independence* and *renaming-compatibility* are discussed. Two structure-independent algorithms are shown, NT and NA.

1.2 Notation

As a visual aid, the first defining mention of a concept or symbol shall be shown in blue. In Prolog, everything is a term, and so shall *term* be here the topmost syntactic concept. *Term*s are built from *functor*s and *variable*s in the usual fashion; the (countably infinite) set of variables is denoted $\boldsymbol{V}$. The set of natural numbers (with 0) is denoted $\boldsymbol{N}$. The set of variables of t is $Vars(t)$. We sometimes treat a *sequence* of terms as a term in its own right, so e.g. $\boldsymbol{U}((E,E'))$ means that $\boldsymbol{U}$ is applied on the equation sequence E,E'.[2] A sequence enclosed in brackets $[t_1,...,t_n]$ is called a *list*. The list of all distinct variables of t, in order of appearance, shall be denoted as $VarList(t)$. If terms s and t are variable-disjoint, we write $s \rhd\lhd t$. If s is a subterm of t, we write $s \mathring{\in} t$. A variable occuring only once in a term is a *singleton variable* of that term. Limit cases of term: *empty* ($\square$), *anything* ($\_$) and *impossibility* ($\bot$).

We also allow for *undefined* terms, like an outcome of an ongoing (perhaps even non-terminating) computation.

Remark 1.1 (object vs. meta). Our object language is the language of Horn clause logic (HCL), depicted in Standard Prolog syntax (with the exception of indices, which we employ for better readability). Hence, object-variables are capitalized. We use `typewriter` font for object-level terms, and mathematical fonts for meta-level, so `X,Y` are "real" variables and x,y are meta-level variables.

Substitution For the most part traditional notation will be used, save for writing substitution application in the functional way as $\sigma(t)$, despite the meanwhile prevailing postfix notation $t\sigma$. Hence, composition of substitutions shall be applied from right to left.

[2] The double parentheses in $\boldsymbol{U}((E,E'))$ serve to distinguish a sequence *as* argument from a sequence *of* arguments.

A *substitution* σ is a mapping of variables to terms that is identity almost everywhere, i.e. its *(active) domain*, or *core*, defined as $Dom(\sigma) := \{x \in \boldsymbol{V} \mid \sigma(x) \neq x\}$, is finite. Hence, σ can be represented in a finitary way as $\sigma = \{x_1/\sigma(x_1), ..., x_n/\sigma(x_n)\}$, where $\{x_1, ..., x_n\} = Dom(\sigma)$. Each $x_i/\sigma(x_i)$ is called a *binding*. To aid readability, σ may be depicted as $\sigma = \binom{x_1 \ \ldots \ x_n}{\sigma(x_1) \ \ldots \ \sigma(x_n)}$.[3] The identity mapping () is also denoted ε. The *restriction* $\sigma\restriction_W$ of σ to[4] a set $W \subseteq \boldsymbol{V}$ means: if $x \in W$ then $\sigma\restriction_W(x) := \sigma(x)$, else $\sigma\restriction_W(x) := x$. For an arbitrary term t we put $\sigma\restriction_t := \sigma\restriction_{Vars(t)}$. Thus, as the core representation suggests, $\sigma \subseteq \theta$ means $\theta\restriction_{Dom(\sigma)} = \sigma$. Substitution definition is extended in the functor-preserving way on *all* terms, i.e. $\sigma(f(t_1, ..., t_n)) := f(\sigma(t_1), ..., \sigma(t_n))$, and $\sigma(t)$ is called an *instance* of t. The *(active) range* of σ is $Ran(\sigma) := \sigma(Dom(\sigma))$. A *renaming* ρ is a substitution represented by a permutation, and $\rho(t)$ is a *variant* of t, written as $t \cong \rho(t)$. *Composition* of σ and θ is defined by $(\sigma \cdot \theta)(x) := \sigma(\theta(x))$. If $\sigma \cdot \sigma = \sigma$, then σ is called *idempotent*. A substitution σ is *at least as*[5] *general* as θ, if there is a substitution δ with $\theta = \delta \cdot \sigma$. Two substitutions are *equigeneral*, if each is at least as general as the other one. A substitution σ can be renamed with ρ by $\rho(\sigma) := \{\rho(x)/\rho(\sigma(x)) \mid x \in Dom(\sigma)\}$. A set S of substitutions can be renamed with ρ by $\rho(S) := \{\rho(\sigma) \mid \sigma \in S\}$, and composed with a substitution δ by $S \cdot \delta := \{\sigma \cdot \delta \mid \sigma \in S\}$. *Union* of substitutions breaks down to *adding of binding*s to a substitution, which is defined as follows:

$$\sigma \cup \binom{x}{t} := \begin{cases} \bot, & \text{if } x \in Dom(\sigma) \text{ and } t \neq \sigma(x) \\ \sigma, & \text{if } x \in Dom(\sigma) \text{ and } t = \sigma(x) \\ \binom{x_1 \ \ldots \ x_n \ x}{\sigma(x_1) \ \ldots \ \sigma(x_n) \ t}, & \text{if } x \notin Dom(\sigma) = \{x_1, ..., x_n\} \end{cases}$$

Already known claims like the following shall be referred to as *legacy* claims.

Legacy 1.2 (finite permutation, [14]). *A substitution ρ is a renaming iff $\rho(Dom(\rho)) = Dom(\rho)$.*

Legacy 1.3 (equigenerality, [5, Lemma 2.10]). *Substitutions θ and θ' are equigeneral iff for some renaming ρ holds $\rho \cdot \theta = \theta'$.*

Legacy 1.4 (substitution renaming, [1]). *For any renaming ρ and any substitution σ, $\rho(\sigma) = \rho \cdot \sigma \cdot \rho^{-1}$.*

Unification Let s and t be terms. If there is a substitution θ such that $\theta(s) = \theta(t)$, then s and t (as well as the equation $s = t$) are said to be *unifiable*, and θ is their *unifier*, the set of all such being $UnifSet(s,t)$. A unifier of s,t is a *most general unifier (mgu)* of s,t if it is at least as general as any unifier of s,t. An mgu σ of $s = t$ is *relevant*, if $Vars(\sigma) \subseteq Vars((s,t))$. The set of all mgus for s,t shall be denoted as $MguSet(s,t)$. Since unifiability of the equation $f(s_1, ..., s_n) = f(t_1, ..., t_n)$ entails unifiability of the set of equations $\{s_1 = t_1, ..., s_n = t_n\}$ and vice versa, it is the same task. Hence, unary notation like $UnifSet(E)$, $MguSet(E)$ is also used, where E is a set of equations.[6] In case of only one equation, set-constructor braces may be dropped.

An equation of the form $x = t$ corresponds to a binding, hence we (somewhat sloppily) also call it a *binding*, and say that x is a *bound* variable of the current equation set. Bindings $x = t$ and $x' = t'$ are *additive* if $x \neq x'$ or $x = x', t = t'$. A set of pairwise additive bindings is said to be in *ab-form*. A set of

[3] Repeating core variables is not harmful but does not make much sense either, so for depicting a core representation we assume $x_1, ..., x_n$ to be mutually distinct.

[4] As remarked in [9], beware it only gives $Dom(\sigma) \subseteq W$ but not necessarily $Dom(\sigma) = W$, as would be expected.

[5] The phrase "at least as general as" is a mouthful, hence it is seldom used (apart from [21]). We can afford it here since generality of substitutions is not a central issue. Traditionally, "more general than" is used, which is somewhat misleading.

[6] At this point, it may prove worthwhile for a novice reader to establish just how general the identity substitution ε is, and what the values of $UnifSet(\emptyset)$ and $MguSet(\emptyset)$ are. The symbol $\emptyset$ denotes the empty set, as usual.

equations is in *solved form* (or *solved*), if it is in ab-form and all of its bound variables are singletons. If E is in ab-form, then its *associated* substitution is $Subst(E) := \{x/t \mid x = t \in E \text{ and } x \neq t\}$. Vice versa, every substitution σ determines an equation set in ab-form, $Eq(\sigma) := \{x = t \mid x/t \in \sigma\}$.

HCL Regarding HCL and its proof method of *SLD-resolution*, mostly we shall assume traditional concepts as given in [15] and [2], with the following changes. In a pragmatic deviation from the original meaning of clauses, we treat program clauses as *fixed terms*, having their variables chosen by the programmer, so a renamed program clause is not a program clause any more, but a *variant* of one. As a visual help, program clauses shall be written with a hat, like $\hat{\mathscr{K}}$. Assume an SLD-derivation $\mathscr{D}$ for G of the form $G \hookrightarrow_{\mathscr{K}_1:\sigma_1} G_1 \hookrightarrow_{\mathscr{K}_2:\sigma_2} \ldots \hookrightarrow_{\mathscr{K}_n:\sigma_n} G_n$. Here $\mathscr{K}_i$ is the *actually used* variant of a program clause (i.e., an *input clause*). Hence, the annotations (*score*s) $\mathscr{K}_i{:}\sigma_i$ can now be regarded as part of the derivation, so $Vars(\mathscr{D}) := Vars(G) \cup \ldots \cup Vars(G_n) \cup Vars(\mathscr{K}_1{:}\sigma_1) \cup \ldots \cup Vars(\mathscr{K}_n{:}\sigma_n)$. Since $Vars(G_n) \subseteq Vars(G_{n-1}) \cup Vars(\mathscr{K}_n{:}\sigma_n)$ and so forth, $Vars(\mathscr{D}) = Vars(G) \cup Vars(\mathscr{K}_1{:}\sigma_1) \cup \ldots \cup Vars(\mathscr{K}_n{:}\sigma_n)$. The substitution $\sigma_n \cdot \ldots \cdot \sigma_1$ shall be called the *partial answer* for G at the step n of the derivation $\mathscr{D}$.

2 Variable separation for substitutions

Idempotent substitutions exhibit a kind of variable separation within them. We recall some useful facts about idempotency (Subsection 2.1), and move on to separation between substitutions (Subsection 2.2).

2.1 Variable separation within a substitution: idempotency

Let us begin with the well-known visual criterion for idempotency, first reported by Eder: the core and the range may not have common variables.

Legacy 2.1 (idempotency, [5, Remark 3.2])**.** *A substitution σ is idempotent iff $Dom(\sigma) \not\bowtie Ran(\sigma)$.*

As a consequence, idempotent substitutions eliminate their core variables from their argument:

Corollary 2.2 (core release, [2, p. 37])**.** *If σ is idempotent and $x \in Dom(\sigma)$, then $x \notin \mathring{\sigma}(t)$ for any t.*

As shown in [5], composition of two idempotent substitutions does not have to be idempotent – not even equigeneral to an idempotent substitution. Example: $\sigma := \binom{\mathtt{X}}{\mathtt{f(Y)}}$ and $\theta := \binom{\mathtt{Y}}{\mathtt{f(Z)}}$ give $\sigma \cdot \theta = \binom{\mathtt{X}\ \ \mathtt{Y}}{\mathtt{f(Y)}\ \mathtt{f(Z)}}$. However, $\theta \cdot \sigma = \binom{\mathtt{X}\ \ \mathtt{Y}}{\mathtt{f(f(Z))}\ \mathtt{f(Z)}}$ is idempotent. This is an instance of a useful property: [7]

Legacy 2.3 (idempotent composition, [2])**.** *Let σ and θ be idempotent. If $Ran(\theta) \not\bowtie Dom(\sigma)$, then $\theta \cdot \sigma$ is also idempotent.*

In case of idempotent mgus, it is known that they cannot harbour extraneous variables:

Legacy 2.4 (relevance, [2])**.** *Every idempotent mgu is relevant.*

2.2 Variable separation across substitutions

Motivated by Legacy 2.3, we now consider compositions of substitutions that are not quite arbitrary, but in some way "variable-separated" from each other. We propose two kinds of separation for substitutions:

- weak variable separation, defined as core-disjointness
- strong variable separation, *cascading*

[7] For many of the claims in this text we give a justification, usually transferred to Appendix A and linked in.

2.2.1 Weak separation (core-disjointness)

Core-disjoint substitutions can extend each other (*sum of substitutions*, [11, Sec. 4]). Here and in Section 3 we show a few more of their qualities. For example, there is a shortcut for their composition:

Lemma 2.5. *If E is in ab-form and $Dom(\sigma) \not\bowtie Dom(Subst(E))$, then $\sigma \cdot Subst(E) = Subst(\sigma(E)) \cup \sigma$.*[8]

In general, composing substitutions is not monotone, i.e. compatible with the subset relation. Example: $\binom{\mathtt{X}}{\mathtt{Y}} \subseteq \binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{X}}$, but $\binom{\mathtt{Y}}{\mathtt{a}} \cdot \binom{\mathtt{X}}{\mathtt{Y}} \not\subseteq \binom{\mathtt{Y}}{\mathtt{a}} \cdot \binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{X}}$. An exception is composing with a core-disjoint substitution:

Lemma 2.6 (left monotonicity)**.** *If $\sigma \subseteq \theta$ and $Dom(\lambda) \not\bowtie Dom(\theta)$, then $\lambda \cdot \sigma \subseteq \lambda \cdot \theta$.*

As a consequence, core-disjoint substitutions compose in a layered way:

Corollary 2.7 (prefix refinement)**.** *For core-disjoint $\sigma_1, ..., \sigma_n$ holds $\sigma_1 \subseteq \sigma_1 \cdot \sigma_2 \subseteq \ldots \subseteq \sigma_1 \cdot ... \cdot \sigma_n$.*

2.2.2 Strong separation (cascading)

Now we shall require a bit more than core-disjointness. Motivation was the condition $Ran(\theta) \not\bowtie Dom(\sigma)$ from Legacy 2.3, which is satisfied by the sequence of bindings in Martelli-Montanari-type unification algorithms (Section 4), or by the sequence of idempotent mgus in a SLD-derivation (Subsection 5.1). In both of these cases, a stronger criterion is satisfied as well, so let us give it a name:

Definition 2.8 (cascading)**.** A sequence of substitutions $\sigma_1, \sigma_2, ...$ is *cascading*, if for every $i, k \geq 1$

$$\sigma_{i+k} \not\bowtie Dom(\sigma_i) \tag{1}$$

Further on in Example 4.16 we shall obtain the sequence $\binom{\mathtt{X}}{\mathtt{Z}}, \binom{\mathtt{Z}}{\mathtt{Y}}$ showing the nature of cascading: a later member of the sequence may have variables from the ranges, but not from the cores, of the former members.

As a special case of Legacy 2.3, composition of idempotent cascading substitutions is idempotent:

Lemma 2.9 (idempotent composition)**.** *If $\sigma_1, ..., \sigma_n$ are cascading and idempotent, then $\sigma_n \cdot ... \cdot \sigma_1$ is idempotent.*

3 Variable conservation during instantiation and composition

Here we discuss some possibilities for conserving the set of variables when applying a substitution, or when composing two substitutions. These are useful e.g. when proving some finer points about SLD-derivations like compositional deriving of conjunction [13, Sec. 4.2].

We start with an easy claim: through instantiation, a term may lose some or win some variables, but any changes are contained within the substitution.

Lemma 3.1. *For any term t and any substitution σ holds $Vars(t) \cup Vars(\sigma) = Vars(\sigma(t)) \cup Vars(\sigma)$.*

When composing two substitutions, no new variables can appear, but any number of variables may disappear from the core representation. A well-known example is $\binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{X}} \cdot \binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{X}} = () = \varepsilon$.

If one of the substitutions is idempotent and the other one a renaming, there is no such loss, as witnessed by $\binom{\mathtt{Y}\ \mathtt{Z}}{\mathtt{Z}\ \mathtt{Y}} \cdot \binom{\mathtt{Y}}{\mathtt{f(X)}} = \binom{\mathtt{Y}\ \ \mathtt{Z}}{\mathtt{f(X)}\ \mathtt{Y}}$ and $\binom{\mathtt{Y}}{\mathtt{f(X)}} \cdot \binom{\mathtt{Y}\ \mathtt{Z}}{\mathtt{Z}\ \mathtt{Y}} = \binom{\mathtt{Y}\ \ \mathtt{Z}}{\mathtt{Z}\ \mathtt{f(X)}}$.

[8] With an appropriate definition of *substitution instance* $\sigma(\theta)$, left to the reader, the claim can be more succinctly expressed by: If $Dom(\sigma) \not\bowtie Dom(\theta)$, then $\sigma \cdot \theta = \sigma(\theta) \cup \sigma$.

Lemma 3.2 (conservation I)**.** *For any idempotent substitution σ and any renaming ρ holds $Vars(\rho\cdot\sigma) = Vars(\sigma\cdot\rho) = Vars(\rho)\cup Vars(\sigma)$.*

Such a property holds also if both substitutions are idempotent, but with disjoint cores. Without core-disjointness, it does not hold, as witnessed by $\binom{\mathtt{X}}{\mathtt{Y}}\cdot\binom{\mathtt{X}}{\mathtt{Z}} = \binom{\mathtt{X}}{\mathtt{Z}}$.

Lemma 3.3 (conservation II)**.** *If σ,θ are idempotent and $Dom(\sigma) \not\bowtie Dom(\theta)$, then $Vars(\theta\cdot\sigma) = Vars(\theta)\cup Vars(\sigma)$.*

Hence, by composing idempotent cascading substitutions we cannot "get rid" of variables:

Corollary 3.4. *If $\sigma_1,\ldots,\sigma_n$ are cascading and idempotent, $Vars(\sigma_n\cdot\ldots\cdot\sigma_1) = Vars(\sigma_n)\cup\ldots\cup Vars(\sigma_1)$. So if at least one σ_j is not ε, then $\sigma_n\cdot\ldots\cdot\sigma_1 \neq \varepsilon$.*

4 Unification by algorithm

For any unifiable equation set E holds that $MguSet(E)$ is infinite. But in a sense, one mgu is enough, since any other can be obtained by composition with a renaming, as shown in Legacy 1.3.

Hence, it is no deprivation that any (deterministic[9]) unification algorithm $\boldsymbol{U}$ produces just one fixed value. For the sake of implementation, we assume the input of $\boldsymbol{U}$ to be not a set but a *sequence* of equations (so instead of $\emptyset$ we now have $\square$ as the limit case). The output of $\boldsymbol{U}$ shall be a substitution or a failure report. A *failure report* shall be a unary term with main functor *Failure*.

Definition 4.1 (unification algorithm)**.** A unary function $\boldsymbol{U}$ is a *unification algorithm*, if for every equation sequence E holds: in case E is unifiable, $\boldsymbol{U}(E)\in MguSet(E)$, otherwise $\boldsymbol{U}(E) = Failure(\_)$.

The first unification algorithm bearing that name was invented by Alan Robinson, who introduced and solved the task of unification in [20] (see also [2, p. 26 ff] for two alternative presentations). Another classical solution is Martelli-Montanari's nondeterministic unification scheme [16, p. 261].[9] [10] In the next subsection, we recall the scheme and choose a deterministic version of it as the basic unification algorithm of this paper. As shall be seen, it amalgamates both of the classical algorithms.

4.1 Martelli-Montanari scheme and its deterministic version, algorithm MM

Martelli-Montanari scheme, rephrased in Figure 1, operates by transforming a set of equations by means of rules. Each rule handles a certain type of equation, embodied in the *working equation*, freely chosen from the set.

Essential for the scheme is to preserve unifiers: in every step $E \rightsquigarrow E'$ must hold $UnifSet(E) = UnifSet(E')$. (For the failure steps, we must extend *UnifSet* with $UnifSet(Failure(\_)) := \emptyset$.) By definition of mgu, that would imply $MguSet(E) = MguSet(E')$. It is fairly obvious that the rules in Figure 1, save perhaps for the binding rule, do preserve unifiers. For the binding rule, an appropriate claim is given below (Lemma 4.4), formulated in two steps to suit the modified rule from Subsection 4.2 as well. But first we need a name for a particular kind of bindings:

Definition 4.2 (I-binding)**.** A binding $x = t$ with $x \mathring{\notin} t$ is called an *I-binding*[11].

Lemma 4.3 (I-binding)**.** *For every I-binding $x = t$ holds $\binom{x}{t}\in MguSet(x = t)$.*

[9] In this paper, we reserve the word "algorithm" solely for deterministic algorithmic schemes, to be able to use it as a mathematical function.

[10] Sometimes attributed to Herbrand, for being incipient in Herbrand's dissertation [7, p. 96-97], [8, p. 31].

[11] "I" stands for idempotence. An I-binding is also a "simple pair" of [2, p. 27].

To find an mgu of a finite set of equations E_0, take $E := E_0$ and transform E according to the rules below. If no more rules apply, stop with success. If a failure rule applies, stop with failure. Recall that x denotes a variable (Remark 1.1), t, s are terms, and f, g are functors.

decomposition $E \cup \{f(s_1,...,s_n) = f(t_1,...,t_n)\} \rightsquigarrow E \cup \{s_1 = t_1,...,s_n = t_n\}$

failure: clash $E \cup \{f(s_1,...,s_n) = g(t_1,...,t_m)\} \rightsquigarrow \mathit{Failure}(\text{"clash"})$, if $f \neq g$ or $m \neq n$

cleaning $E \cup \{x = x\} \rightsquigarrow E$

orientation $E \cup \{t = x\} \rightsquigarrow E \cup \{x = t\}$, if $t \notin \mathbf{V}$

binding $E \cup \{x = t\} \rightsquigarrow \binom{x}{t}(E) \cup \{x = t\}$, if $x \hat{\in} E$ and $x \hat{\notin} t$

failure: occurs-check $E \cup \{x = t\} \rightsquigarrow \mathit{Failure}(\text{"OC"})$, if $x \hat{\in} t$ and $x \neq t$

Figure 1: Martelli-Montanari's non-deterministic unification scheme

Proof. Owing to the condition $x \hat{\notin} t$, every I-binding is unifiable, and $\binom{x}{t} \in \mathit{UnifSet}(x = t)$. Maximal generality is ensured by the *witness method* [12, Theorem IV.3].

To see this, let us take an arbitrary $\sigma \in \mathit{MguSet}(x = t)$ and build a diagram in Figure 2 starting with $\mathit{VarList}((x,t,\sigma)) = [x, y_1, ..., y_n]$. Due to $\sigma(x) = \sigma(t)$, the blue mapping δ exists, and $\delta = \sigma\restriction_{t,y_1,...,y_n}$. According to the witness method, the diagram commutes, giving[12] $\sigma = \sigma\restriction_{t,y_1,...,y_n} \cdot \binom{x}{t}$, which shows that $\binom{x}{t}$ is at least as general as σ. ♢

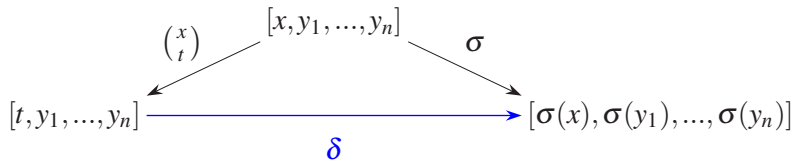

Figure 2: Proving the maximal generality of $\binom{x}{t}$

Lemma 4.4 (binding rule preserves mgus)**.** *For every equation set E and every I-binding $x = t$:*

1. $\mathit{UnifSet}(\{x = t\} \cup E) = \mathit{UnifSet}(\binom{x}{t}(E)) \cdot \binom{x}{t}$
2. $\mathit{UnifSet}(\binom{x}{t}(E)) \cdot \binom{x}{t} = \mathit{UnifSet}(\{x = t\} \cup \binom{x}{t}(E))$

Both claims also hold with MguSet instead of UnifSet, or with the argument being an equation sequence instead of a set.

By Lemma 4.4, if Martelli-Montanari scheme terminates, then it solves the unification task.

Legacy 4.5 (solved form, [16][2])**.** *For any E_0 and any choice of working equation, the transformation in Figure 1 is certain to stop. If the stop is due to failure, then E_0 is not unifiable. Otherwise, the final set E is in solved form and $\mathit{Subst}(E)$ is an idempotent mgu of E_0.*

[12] For a stronger claim, observe: If $x \notin \mathit{Dom}(\sigma)$, then $\sigma\restriction_{t,y_1,...,y_n} = \sigma$, otherwise $\sigma\restriction_{t,y_1,...,y_n} \cup \binom{x}{\sigma(x)} = \sigma$. Due to $x \hat{\notin} t$, in either case $\sigma\restriction_{t,y_1,...,y_n} \cdot \binom{x}{t} = \sigma \cdot \binom{x}{t}$. Summarily, $\sigma = \sigma \cdot \binom{x}{t}$. This proves "strongness" of $\binom{x}{t}$, see also [2, Th. 2.19].

Martelli-Montanari scheme can be turned into a (deterministic) algorithm in a number of ways, resulting in many of the known unification algorithms [16, p. 263]. Here we advocate a specific instance, emulating Robinson's unification algorithm, which makes it "the" classical unification algorithm. Correspondence with Robinson's algorithm shall be handled in Subsubsection 4.2.6.

Definition 4.6 (MM)**.** The algorithm MM is obtained from Martelli-Montanari scheme by using sequences instead of sets and picking the leftmost equation eligible for a rule application. One more adjustment shall be made, to suit Definition 4.1: The output of MM in case of success shall not be a solved form E, as usual, but its associated substitution $Subst(E)$.

By Legacy 4.5, MM is a unification algorithm.

4.2 Algorithm RMM, a modular adaptation of MM

Due to its simplicity, MM is widely used. Yet, it can be made even simpler, as observed in [3]. As a bonus, a kind of modularity can be acquired, which shall be useful in proving an iteration property (Subsubsection 4.4.3). The only change concerns the binding rule of MM:

$$\textbf{binding (MM)}\ \ x = t, E \rightsquigarrow x = t, \binom{x}{t}(E),\ \text{if } x \mathrel{\mathring{\notin}} t \text{ and } x \mathrel{\mathring{\in}} E \tag{2}$$

Let us take a look at its effects. After the i-th binding step, the used equation $x_i = t_i$ becomes an *ineligible* equation, i.e. it cannot be (re-) elected for transformation in MM. To see that, recall that applying $\binom{x_i}{t_i}$ with $x_i \mathrel{\mathring{\notin}} t_i$ on a term eliminates the variable x_i from that term (Corollary 2.2), so only one occurrence of x_i remains in the sequence: the one in $x_i = t_i$, which makes x_i a singleton variable of the sequence. Thus, any further bindings x_{i+k}/t_{i+k} must be variable-disjoint with x_i, i.e.

$$x_i \mathrel{\mathring{\notin}} x_{i+k}/t_{i+k},\ \text{for any } k \geq 1 \tag{3}$$

so they can only change the right-hand side of $x_i = t_i$. Therefore, the equation remains ineligible. Another possibility for an ineligible equation $x = t$ is that x was a singleton already; then $x = t$ shall never be elected by MM. Either way, an ineligible equation only stays underfoot, so it might as well be moved into a "result zone", which in [3] is a solved form and here it shall be a substitution right away (we dispense with solved form in favor of composing bindings, as in Robinson's algorithm).

At the same time, the binding rule of MM shall be freed from the dependence on the rest of the equations, i.e. the condition $x \mathrel{\mathring{\in}} E$ shall be dropped from (2). So any $x = t$ satisfying $x \mathrel{\mathring{\notin}} t$ (i.e., any I-binding) is now eligible. This gives more binding steps, but on the bonus side we now have *uniformity*: the working equation is now always the leftmost equation. For want of a better word we shall speak of *modular rephrasing* of the binding rule.

The algorithm so obtained is shown in Figure 3; it is a mix of Martelli-Montanari's transformation rules and Robinson's result building, therefore we denote it RMM. It differs from MM in two details:

- binding rule:
  - treatment of ineligible equations: MM ignores them, RMM removes them
  - context dependence (MM) vs. modularity (RMM)
- construction of the result: MM builds a solved form, RMM composes bindings

Despite those differences, do they compute the same result?

To find an mgu of a finite sequence of equations E_0, take $E := E_0$ and $\sigma := \varepsilon$ and transform E according to the rules below. If no more rules apply, stop with success, and $\mathsf{RMM}(E_0) := \sigma$. If a failure rule applies, stop with failure, and $\mathsf{RMM}(E_0) := E$.

decomposition $f(s_1,...,s_n) = f(t_1,...,t_n), E \rightsquigarrow s_1 = t_1,...,s_n = t_n, E$

failure: clash $f(s_1,...,s_n) = g(t_1,...,t_m), E \rightsquigarrow \mathit{Failure}(\text{"clash"})$, if $f \neq g$ or $m \neq n$

cleaning $x = x, E \rightsquigarrow E$

orientation $t = x, E \rightsquigarrow x = t, E$, if $t \notin \boldsymbol{V}$

binding (RMM) $x = t, E \rightsquigarrow \binom{x}{t}(E)$, if $x \mathrel{\mathring{\notin}} t$. Additionally, $\sigma := \binom{x}{t} \cdot \sigma$.

failure: occurs-check $x = t, E \rightsquigarrow \mathit{Failure}(\text{"OC"})$, if $x \mathrel{\mathring{\in}} t$ and $x \neq t$

Figure 3: RMM unification algorithm

4.2.1 Dissecting a termination proof

To answer this question, we shall employ two properties of the algorithms. These properties, or rather *tactics*, explain the (arguably somewhat contrived) termination proof for Martelli-Montanari algorithm [2, p. 32 ff], repeated here with Legacy 4.5. Together, they build a simple wave-like mechanism underlying not only RMM but, somewhat surprisingly, MM as well:

simplification: decompose, orient and clean until an I-binding is uncovered

reduction: if there is one, use it to eliminate its bound variable

Clearly, iterating these two tactics must come to a stop. This is the gist of an alternative termination proof for RMM, pieced together from the two tactics (Lemma 4.7, Lemma 4.9) in Theorem 4.10. Since the tactics hold for MM as well, its termination can also be proved in this manner.

4.2.2 Simplification: Finding a variable to eliminate (via common rules)

The first tactic holds for both algorithms and puts focus on I-bindings:

Lemma 4.7 (simplification)**.** *If E is not already in form $x = t, E'$ with $x \mathrel{\mathring{\notin}} t$, then the rules common to* MM *and* RMM *will either bring it in this form, or stop at $\square$, or stop at a failure. In each case, the unifiers will be preserved.*

4.2.3 Reduction: Eliminating the variable (via binding rule)

To express the second tactic, we need an auxiliary operator $*$ associating a possible output of unification with a substitution:

Definition 4.8 (maybe-composition)**.** Let A be a possible output of an $\boldsymbol{U}$, i.e. a substitution or a failure report, perhaps undefined. Let σ be a substitution. Their *maybe-composition* is

$$A * \sigma := \begin{cases} A \cdot \sigma, & \text{if } A \text{ is a substitution} \\ A, & \text{if } A \text{ is a failure report} \\ \text{undefined}, & \text{if } A \text{ is undefined} \end{cases}$$

In case of RMM, the second tactic is rather obvious:

Lemma 4.9 (reduction in RMM)**.** *For any sequence $x=t, E$ with $x \mathrel{\mathring{\notin}} t$ holds*

$$\mathsf{RMM}((x=t\,,E)) = \mathsf{RMM}(\tbinom{x}{t}(E)) * \tbinom{x}{t} \text{ with } \mathit{UnifSet}((x=t,E)) = \mathit{UnifSet}(\tbinom{x}{t}(E)) \cdot \tbinom{x}{t} \quad (4)$$

Proof. In RMM, the leftmost equation must be elected. Hence, computing $\mathsf{RMM}((x=t\,,E))$ is reduced to computing $\mathsf{RMM}(\binom{x}{t}(E))$, so if the latter fails or does not stop, the same holds for the former. The bindings are composed in reverse order. So the possible output of $\mathsf{RMM}((x=t\,,E))$ is built as in (4).

By Lemma 4.4, part 1, this is reflected in the set of unifiers. Hence, the new binding rule also preserves unifiers. ♢

4.2.4 Consequences for RMM

Theorem 4.10 (RMM)**.** *For any (even empty) equation sequences E and E' holds*

1. *if E is unifiable, then $\mathsf{RMM}((E\,,E')) = \mathsf{RMM}(\sigma(E')) * \sigma$, where $\sigma \in \mathit{UnifSet}(E)$*
2. *otherwise, $\mathsf{RMM}((E\,,E'))$ must fail.*

By taking $E' := \square$, we obtain that $\mathsf{RMM}(E)$ stops for any E and, in case of unifiability of E, produces an unifier, otherwise a failure report. Thus, RMM is a unification algorithm.

Corollary 4.11 (iteration for RMM)**.** *If $\mathsf{RMM}(E') =: \sigma$ and $\mathsf{RMM}(\sigma(E'')) =: \theta$ both succeed, then so does $\mathsf{RMM}((E'\,,E''))$, and $\mathsf{RMM}((E'\,,E'')) = \theta \cdot \sigma$.*

Corollary 4.12 (idempotence for RMM)**.** *Assume $\mathsf{RMM}(E)$ succeeds, and $\sigma := \mathsf{RMM}(E)$. If the binding rule was not applied, then $\sigma = \varepsilon$, otherwise $\sigma = \sigma_n \cdot \ldots \cdot \sigma_1 \neq \varepsilon$, where $\sigma_1, \ldots, \sigma_n$ are the respective I-bindings. At any rate, σ is idempotent.*

4.2.5 Coincidence with MM

To conclude the comparison between MM and RMM, we need one last auxiliary result, namely that the second tactic for RMM (Lemma 4.9) holds for MM as well, even if less obviously:

Lemma 4.13 (reduction in MM)**.** *For any sequence $x=t, E$ with $x \mathrel{\mathring{\notin}} t$ holds*

$$\mathsf{MM}((x=t\,,E)) = \mathsf{MM}(\tbinom{x}{t}(E)) * \tbinom{x}{t} \text{ with } \mathit{UnifSet}((x=t,E)) = \mathit{UnifSet}(\tbinom{x}{t}(E)) \cdot \tbinom{x}{t} \quad (5)$$

Proof. According to the discussion of ineligible equations on page 8, computing $\mathsf{MM}((x=t\,,E))$ is reduced to computing $\mathsf{MM}(\binom{x}{t}(E))$, whether $x \mathrel{\mathring{\in}} E$ or not. Hence, if $\mathsf{MM}(\binom{x}{t}(E))$ fails or does not stop, the same holds for $\mathsf{MM}((x=t\,,E))$ as well.

Assume now $\mathsf{MM}(\binom{x}{t}(E))$ succeeds. The earlier discussion of ineligible equations gives

$$\mathsf{MM}((x=t\,,E)) = \mathit{Subst}\left(\mathsf{MM}(\tbinom{x}{t}(E))(x=t)\right) \cup \mathsf{MM}(\tbinom{x}{t}(E))$$

The right-hand side can be simplified by bearing in mind that x is eliminated from E by $\binom{x}{t}$, i.e. $x \mathrel{\mathring{\notin}} \binom{x}{t}(E)$, so by relevance $x \mathrel{\mathring{\notin}} \mathsf{MM}(\binom{x}{t}(E))$; now we can apply Lemma 2.5, obtaining $\mathsf{MM}(\binom{x}{t}(E)) \cdot \binom{x}{t}$.

Summarily, $\mathsf{MM}((x=t\,,E)) = \mathsf{MM}(\binom{x}{t}(E)) * \binom{x}{t}$. By Lemma 4.4, the set of unifiers mimicks this property. ♢

Theorem 4.14 (coincidence)**.** *For every equation sequence E holds $\mathsf{RMM}(E) = \mathsf{MM}(E)$.*

4.2.6 "The" classical unification algorithm

Since RMM is actually a rephrasing of Robinson's algorithm, from Theorem 4.14 follows

Corollary 4.15 (the classical $\boldsymbol{U}$)**.** MM *corresponds to Robinson's algorithm.*

So MM and RMM compute the same mgu, in slightly different ways. Should RMM be favoured over MM? The trade-off would be

- less search, more uniformity: the working equation is always the leftmost equation
- some unnecessary applications $\binom{x}{t}(E)$ despite $x \overset{?}{\notin} E$

Finally, let us compare the operation of MM and RMM on a typical unification task.

Example 4.16 (MM versus RMM)**.** In the computations below, the next eligible equation is underlined.

$$\begin{aligned}
&\mathsf{MM}(\underline{\texttt{f(X,Z,U)=f(Z,Y,U)}}) = \mathsf{MM}((\texttt{X=Z},\ \underline{\texttt{Z=Y}},\ \texttt{U=U})),\ \text{by decomposition}\\
&= \mathsf{MM}((\texttt{X=Y},\ \texttt{Z=Y},\ \underline{\texttt{U=U}})),\ \text{by binding } \texttt{Z}/\texttt{Y}\\
&= \mathsf{MM}((\texttt{X=Y},\ \texttt{Z=Y})) = \tbinom{\texttt{X}\ \texttt{Z}}{\texttt{Y}\ \texttt{Y}},\ \text{by cleaning}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{RMM}(\underline{\texttt{f(X,Z,U)=f(Z,Y,U)}}) = \mathsf{RMM}((\underline{\texttt{X=Z}},\ \texttt{Z=Y},\ \texttt{U=U})),\ \text{by decomposition}\\
&= \mathsf{RMM}((\underline{\texttt{Z=Y}},\ \texttt{U=U})) * \tbinom{\texttt{X}}{\texttt{Z}},\ \text{by binding } \texttt{X}/\texttt{Z}\\
&= \mathsf{RMM}(\underline{\texttt{U=U}}) * \tbinom{\texttt{Z}}{\texttt{Y}} * \tbinom{\texttt{X}}{\texttt{Z}},\ \text{by binding } \texttt{Z}/\texttt{Y}\\
&= \mathsf{RMM}(\square) * \tbinom{\texttt{Z}}{\texttt{Y}} * \tbinom{\texttt{X}}{\texttt{Z}} = \varepsilon \cdot \tbinom{\texttt{Z}}{\texttt{Y}} \cdot \tbinom{\texttt{X}}{\texttt{Z}} = \tbinom{\texttt{X}\ \texttt{Z}}{\texttt{Y}\ \texttt{Y}},\ \text{by cleaning}
\end{aligned}$$

Observe the cascading sequence of bindings[13] $\binom{\texttt{X}}{\texttt{Z}}, \binom{\texttt{Z}}{\texttt{Y}}$. This is no coincidence, since the condition (3) is a special case of (1).

4.3 Algorithm MMB, a biased derivative of MM

> *In fact, it appears to be convenient to restrict the choice of the mgu even more by disallowing the "needless renaming of variables in a derivation".*
>
> *[4], p. 13*

Even if we allow only idempotent mgus, a practical dilemma remains: in case of unifying variables x and x_1, where x is from the query and x_1 from the input clause, should we choose $\binom{x}{x_1}$ or $\binom{x_1}{x}$ as their mgu? In the first case, an "old" variable shall be renamed, and henceforth lost, since it cannot be re-introduced again (due to standardizing-apart). In the second case this fate befalls a "new" variable, which may be more acceptable for the user, who tends to prefer his own variables.

But naming continuity is not the only reason against renaming "older" variables (appearing earlier in the derivation) with "newer" ones (appearing later), i.e. against the bindings x_{old}/x_{new}. Bol [4] dispenses with such *needless renaming* in order to prove some claims.

To achieve this, he modifies Martelli-Montanari scheme toward producing a specific kind of mgu, compliant with an ordering of variables given by a function $Tag\colon \boldsymbol{V} \to \boldsymbol{N}$ that assigns natural numbers to variables.

[13] A variable binding x/y determines a substitution in its own right, $\binom{x}{y}$.

Bol's modification consists solely in replacing the orientation rule with a new one respecting *Tag*. Here we rephrase it in a deterministic way, using sequences:

$$\textbf{orientation (Bol)}\ \ t = x, E \ \rightsquigarrow\ x = t, E, \ \text{ if } \ t \notin \boldsymbol{V} \ \text{ or } \ \mathit{Tag}(t) < \mathit{Tag}(x) \tag{6}$$

This rule is obviously not disjoint with the binding rule of MM, namely for $t \in \boldsymbol{V}$ and $\mathit{Tag}(t) < \mathit{Tag}(x)$ and $x \stackrel{\circ}{\in} E$ both rules are applicable. Since we strive for a deterministic algorithm, hence mutually disjoint rules as in MM and RMM, we shall change the binding rule as well. To make the two changes more compact, let us extend the function *Tag* to *all* terms by

$$\mathit{Tag}(t) := -1, \ \text{ if } \ t \notin \boldsymbol{V} \tag{7}$$

The new rules are shown in Figure 4. Thus changed MM needs a new name, so let us take MMB, in acknowledgement of Bol's original idea. The algorithm has a parameter, the ordering function *Tag*, which can be made explicit by writing $\mathsf{MMB}_{\mathit{Tag}}$.

---

orientation (MMB) $s = t, E \ \rightsquigarrow\ t = s, E, \ \text{ if } \ \mathit{Tag}(s) < \mathit{Tag}(t)$

binding (MMB) $x = t, E \ \rightsquigarrow\ x = t, \binom{x}{t}(E), \ \text{ if } \ x \stackrel{\circ}{\in} E \text{ and } x \stackrel{\circ}{\notin} t \text{ and } \mathit{Tag}(x) \geq \mathit{Tag}(t)$

Figure 4: Changes toward MMB

Definition 4.17 (bias)**.** Let $\mathit{Tag}\colon \boldsymbol{V} \to \boldsymbol{N}$. A substitution σ is *Tag-biased*, if for every $x \in \boldsymbol{V}$ holds $\mathit{Tag}(x) \geq \mathit{Tag}(\sigma(x))$. A unification algorithm is *Tag-biased*, if it produces *Tag*-biased mgus.

It is easy to ascertain $\mathsf{MMB}_{\mathit{Tag}}$ to be a unification algorithm, and a *Tag*-biased one:

Theorem 4.18 (MMB)**.** *For any function* $\mathit{Tag}\colon \boldsymbol{V} \to \boldsymbol{N}$ *and any equation sequence* E_0, $\mathsf{MMB}_{\mathit{Tag}}(E_0)$ *stops and, in case of unifiable* E_0*, produces a Tag-biased mgu of* E_0*.*

Lemma 4.19 (bias is compositional)**.** *If* σ, θ *are Tag-biased, their composition is Tag-biased as well.*

Corollary 4.20 (bias)**.** *Assume* $\boldsymbol{U}$ *is Tag-biased for some* $\mathit{Tag}\colon \boldsymbol{V} \to \boldsymbol{N}$*. Then mgus* $\sigma_1, \ldots, \sigma_n$ *produced by* $\boldsymbol{U}$ *in an SLD-derivation are Tag-biased, as well as their composition* $\sigma_n \cdot \ldots \cdot \sigma_1$*.*

4.3.1 Modular adaptation, RMMB

MMB can be rewritten in a modular way, similarly to MM; let us name such an adaptation RMMB.

4.3.2 Seniority-biased SLD-derivations

Observe that the claims up to now hold for arbitrary $\mathit{Tag}\colon \boldsymbol{V} \to \boldsymbol{N}$. Now let us consider a particular choice. To express *variable seniority* in a SLD-derivation, Bol [4] assigns 0 to variables from the top-level query, 1 to variables from the first input clause, and so on; variables appearing later in the derivation ("newer" variables) get bigger number. For this choice of *Tag*, Theorem 4.18 and Corollary 4.20 ensure:

Corollary 4.21 (seniority)**.** *If* MMB *is seniority-biased, then older variables in SLD-derivations will never be renamed with newer ones.*

But can this perhaps be ensured easier, without modifications of MM resulting in MMB? Let us look into reasons for an old variable to be abandoned, or for new variables to appear. Assume a resolution of a query G with a clause $\mathscr{K}$. There can be patterns in the clause head, as with `nat(s(A)) :- nat(A)`. Also, there can be surplus variables in the clause body: `son(A) :- male(A), child(A,B)`. In those cases it is inevitable that some new variables must appear in the resolvent. Otherwise, this is not necessary, as with $G :=$ `p(X)` and $\mathscr{K} :=$ `p(A) :- q(A)`. Here `X` can be left unchanged (e.g. using $\binom{\texttt{A}}{\texttt{X}}$), or needlessly renamed (e.g. to `A` using $\binom{\texttt{X}}{\texttt{A}}$[14]). Also, there are cases of head-patterns like `p(Z,Z)` where MM may or may not perform needless renaming for a query like `p(X,Y)`: $\mathsf{MM}(\texttt{p(X,Y)=p(Z,Z)}) = \binom{\texttt{X Y}}{\texttt{Z Z}}$, but $\mathsf{MM}(\texttt{p(Z,Z)=p(X,Y)}) = \binom{\texttt{X Z}}{\texttt{Y Y}}$. This raises some hope that using MM with clause head on the left might be sufficient to ensure absence of bindings x_{old}/x_{new}, so no modification of MM would be needed.

Counter-Example 4.22 (needless renaming)**.** By applying MM on equation $t_{new} = t_{old}$, where $t_{new} \not\bowtie t_{old}$, as in SLD-resolution, we usually obtain a mgu without bindings x_{old}/x_{new}. But alas, not in every case:

$$\begin{aligned}
&\mathsf{MM}(\texttt{p(f(Y}_1\texttt{),X}_1\texttt{,X}_1\texttt{)=p(X,f(Y),X)}) = \mathsf{MM}((\underline{\texttt{f(Y}_1\texttt{)=X}},\ \texttt{X}_1\texttt{=f(Y)},\ \texttt{X}_1\texttt{=X}))\\
&= \mathsf{MM}((\underline{\texttt{X=f(Y}_1\texttt{)}},\ \texttt{X}_1\texttt{=f(Y)},\ \texttt{X}_1\texttt{=X})) = \mathsf{MM}((\texttt{X=f(Y}_1\texttt{)},\ \underline{\texttt{X}_1\texttt{=f(Y)}},\ \texttt{X}_1\texttt{=f(Y}_1\texttt{)}))\\
&= \mathsf{MM}((\texttt{X=f(Y}_1\texttt{)},\ \texttt{X}_1\texttt{=f(Y)},\ \underline{\texttt{f(Y)=f(Y}_1\texttt{)}})) = \mathsf{MM}((\texttt{X=f(Y}_1\texttt{)},\ \texttt{X}_1\texttt{=f(Y)},\ \underline{\texttt{Y=Y}_1}))\\
&= \mathsf{MM}((\texttt{X=f(Y}_1\texttt{)},\ \texttt{X}_1\texttt{=f(Y}_1\texttt{)},\ \texttt{Y=Y}_1)) = \begin{pmatrix} \texttt{X} & \texttt{X}_1 & \texttt{Y} \\ \texttt{f(Y}_1\texttt{)} & \texttt{f(Y}_1\texttt{)} & \texttt{Y}_1 \end{pmatrix}
\end{aligned}$$

Here new variables are shown indexed. Clearly, `Y` is needlessly renamed, i.e. abandoned in favour of a newer variable, $\texttt{Y}_1$. Using MMB instead of MM would have prevented this.

4.4 Desirable properties of unification algorithms

As shall be established in the rest of this section, the algorithm MM as well as its *Tag*-biased (-B$_{Tag}$) and modular (R-) adaptations obey several properties, postulated below as *normalcy*. The first two we deem even necessary for claims about implemented logic programming.

Definition 4.23 (reasonable $\boldsymbol{U}$)**.** Unification algorithm $\boldsymbol{U}$ is *reasonable*, if for any equation sequence E and any renaming ρ holds

1. renaming-compatibility: $\boldsymbol{U}(\rho(E)) = \rho(\boldsymbol{U}(E))$
2. relevance: $\mathit{Vars}(\boldsymbol{U}(E)) \subseteq \mathit{Vars}(E)$

Definition 4.24 (normal $\boldsymbol{U}$)**.** $\boldsymbol{U}$ is *normal*, if it is reasonable and additionally for any equation sequences E, E' and any term t holds

3. idempotency: If $\boldsymbol{U}(E)$ succeeds, then $\boldsymbol{U}(E) \cdot \boldsymbol{U}(E) = \boldsymbol{U}(E)$.
4. anchor: $\boldsymbol{U}(t = t) = \varepsilon$
5. iteration: If $\sigma = \boldsymbol{U}(E)$ and $\theta = \boldsymbol{U}(\sigma(E'))$ both succeed, then $\theta \cdot \sigma = \boldsymbol{U}((E, E'))$.

4.4.1 Compatibility with renaming

The simple unification task `X=Y` has, among others, two equally attractive candidate mgus, $\binom{\texttt{X}}{\texttt{Y}}$ and $\binom{\texttt{Y}}{\texttt{X}}$. Assume our unification algorithm $\boldsymbol{U}$ decided upon $\binom{\texttt{X}}{\texttt{Y}}$. Assume further that we rename the equation

[14] Or to something else: in case $\boldsymbol{U}$ is not relevant, it could produce an mgu like $\begin{pmatrix} \texttt{X} & \texttt{A} & \texttt{Z} \\ \texttt{Z} & \texttt{Z} & \texttt{A} \end{pmatrix}$, renaming `X` to `Z`.

using $\rho = \binom{\mathtt{Y}\ \mathtt{Z}}{\mathtt{Z}\ \mathtt{Y}}$, obtaining `X=Z`. What mgu shall be chosen this time? In case $\boldsymbol{U}$ is renaming-compatible, we know that it chooses $\rho(\binom{\mathtt{X}}{\mathtt{Y}}) = \binom{\mathtt{X}}{\mathtt{Z}}$.

As observed in [1], the classical unification algorithms do not depend upon the actual names of variables, hence they are renaming-compatible (and prenaming[15]-compatible, for the same reason). This clearly holds for all algorithms shown above.

Counter-Example 4.25 (is every $\boldsymbol{U}$ renaming-compatible?)**.** Let $\boldsymbol{U}(E) := \binom{x\ y}{y\ x} \cdot \mathsf{MM}(E)$, where x is the rightmost variable in E, and y is x but indexed with the next available index. If no variables in E, set $\boldsymbol{U}(E) := \mathsf{MM}(E)$. Clearly, $\boldsymbol{U}(E)$ is an mgu of E. If $E := \mathtt{X=Y}_1$ and $\rho := \binom{\mathtt{Y}_1\ \mathtt{Z}}{\mathtt{Z}\ \mathtt{Y}_1}$, then $\boldsymbol{U}(E) = \binom{\mathtt{Y}_1\ \mathtt{Y}_2}{\mathtt{Y}_2\ \mathtt{Y}_1} \cdot \binom{\mathtt{X}}{\mathtt{Y}_1} = \binom{\mathtt{X}\ \mathtt{Y}_1\ \mathtt{Y}_2}{\mathtt{Y}_2\ \mathtt{Y}_2\ \mathtt{Y}_1}$ and $\boldsymbol{U}(\rho(E)) = \binom{\mathtt{X}\ \mathtt{Z}\ \mathtt{Z}_1}{\mathtt{Z}_1\ \mathtt{Z}_1\ \mathtt{Z}} \neq \rho(\boldsymbol{U}(E)) = \binom{\mathtt{X}\ \mathtt{Z}\ \mathtt{Y}_2}{\mathtt{Y}_2\ \mathtt{Y}_2\ \mathtt{Z}}$.

Note that, unlike $\boldsymbol{U}$, the set-valued *MguSet* must be renaming-compatible[16]:

Lemma 4.26 (*MguSet*)**.** *For any renaming ρ and equation sequence E: $\mathit{MguSet}(\rho(E)) = \rho(\mathit{MguSet}(E))$.*

4.4.2 Idempotency, relevance and anchor

The algorithm MM and its adaptations are not only renaming-compatible, they produce idempotent results as well. In case of non-modular algorithms MM or MMB, idempotency is guaranteed by the solved form. Idempotency also follows from cascading of bindings, as shown for RMM in Corollary 4.12.

From idempotency follows relevance (Legacy 2.4). Furthermore, idempotency of $\boldsymbol{U}$ prevents trivial equations $t = t$ from obtaining non-trivial mgus like $\binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{X}}$, even if they may be relevant:

Lemma 4.27 (anchor)**.** *For any t, the only idempotent mgu of $t = t$ is ε.*

Proof. Assume an idempotent $\sigma \in \mathit{MguSet}(t = t)$. Since $\varepsilon \in \mathit{MguSet}(t = t)$, ε and σ are equigeneral, so by Legacy 1.3 there is a renaming ρ with $\sigma = \rho \cdot \varepsilon = \rho$. Since σ is idempotent, by Legacy 2.1 holds that $\mathit{Dom}(\sigma) \not\bowtie \mathit{Ran}(\sigma)$. The only renaming with this property is ε, due to Legacy 1.2. ♢

4.4.3 Iteration

In [2, Lemma 2.24], it is claimed that an mgu ψ for a sequence E', E'' can be obtained in an iterative fashion, by first finding an mgu σ for E' and then an mgu θ for $\sigma(E'')$, which are then combined as $\psi = \theta \cdot \sigma$. If we have a unification algorithm, it would be good if those σ, θ are not just any mgus, but exactly the ones found by the algorithm.

As a consequence of Theorem 4.10, RMM satisfies this property, i.e. computes in a piecemeal fashion, compatible with LD-resolution (Corollary 4.11). By the coincidence claim (Theorem 4.14), it also holds for MM. Note that iteration is much easier to prove if the binding rule is modular. In a similar way the property can be proved for RMMB (and hence MMB).

5 "Fresh" renaming

Let us conclude with a separation issue. As is well-known [2], if we wish to extend an SLD-derivation $\mathscr{D}$ for a query G using a program clause $\hat{\mathscr{K}}$, but do not wish to risk missing some (logically correct)

[15] A *prenaming* is a variable-pure substitution with mutually distinct variables in range [11].

[16] But *MguSet* is not prenaming-compatible. For example, let $\alpha := \binom{\mathtt{X}}{\mathtt{W}}$ and $E := \mathtt{X=Y}$. Then $\alpha(E) = \mathtt{W=Y}$ and $\theta := \binom{\mathtt{X}\ \mathtt{Y}}{\mathtt{Y}\ \mathtt{W}} \in \mathit{MguSet}(\alpha(E))$, but clearly there is no way for θ to be obtained as $\alpha(\sigma)$ for some $\sigma \in \mathit{MguSet}(E)$.

answers for G, then we have to *standardize* $\hat{\mathscr{K}}$ *apart*[17] from $\mathscr{D}$. This means to regard the variables of $\mathscr{D}$ as "spent", and rename $\hat{\mathscr{K}}$ with some "fresh" variables.

5.1 Excursion: idempotent mgus in SLD-derivations

Now we have all the concepts necessary to establish cascading in SLD-derivations.

Lemma 5.1. *Idempotent mgus $\sigma_1, \ldots, \sigma_n$ in a SLD-derivation $G_0 \hookrightarrow_{\mathscr{K}_1:\sigma_1} G_1 \hookrightarrow_{\mathscr{K}_2:\sigma_2} \ldots$ are cascading.*

As a consequence, a further characterization of partial answers in a Prolog derivation is obtained. The first of its three parts is already known [2, p. 62]. The third part ensures that the partial answer can be ε only if all involved mgus are ε.

Theorem 5.2 (partial answer)**.** *Idempotent mgus $\sigma_1, \ldots, \sigma_n$ in a SLD-derivation satisfy*

1. $\sigma_n \cdot \ldots \cdot \sigma_1$ *is idempotent*
2. $\sigma_n \subseteq \sigma_n \cdot \sigma_{n-1} \subseteq \ldots \subseteq \sigma_n \cdot \ldots \cdot \sigma_1$
3. $Vars(\sigma_n \cdot \ldots \cdot \sigma_1) = Vars(\sigma_n) \cup \ldots \cup Vars(\sigma_1)$

5.2 Fresh renaming by algorithm and some properties

Standardizing-apart algorithms can be seen as a special case of algorithms renaming a given term t in a "fresh" way with respect to a given set of "spent" variables (represented by a term s):

Definition 5.3 (fresh renaming)**.** A binary function ***New*** is a *fresh-renaming algorithm*, if for any s,t holds $\boldsymbol{New}_s(t) \cong t$ and $\boldsymbol{New}_s(t) \not\bowtie s$. In other words, if $\boldsymbol{New}_s(t)$ is a variant of t variable-disjoint with s.

Standardizing-apart algorithms have not been given much attention in literature. In theoretical work, their existence is usually enough, established by the well-known renaming device of Lloyd [15, p. 41]: For the n-th derivation step, the original program clause variables are indexed with n, assuming that top-level queries may not contain indices. This assumption, however, rules out *resuming* of a derivation, i.e. starting from a resolvent, which is needed for proofs involving compositionality [13].

As a remedy, instead of the whole derivation $\mathscr{D}$ solely its variables $Vars(\mathscr{D})$ can be taken as the parameter of the algorithm.

Definition 5.4 (structure-independence)**.** A fresh-renaming algorithm ***New*** is *structure-independent* (or *flat*), if it depends only on the spent variables, i.e. $\boldsymbol{New}_s(t) = \boldsymbol{New}_{Vars(s)}(t)$.

5.2.1 Structure-independent fresh renaming: thrifty NT and prodigal NA

Here we show two structure-independent algorithms, our alternative to Lloyd's theoretical device and a traditional programming device. The former is based on the simple idea to rename only where necessary:

Definition 5.5 (thrifty fresh renaming, NT)**.** The algorithm NT renames any variable[18] x apart from s as follows. If x appears in s, together with its indexed versions $x_1, \ldots, x_k$, but x_{k+1} does not, then x_{k+1} shall replace x, i.e. $\mathsf{NT}_s(x) := x_{k+1}$. Otherwise, x remains unchanged.

[17] *Standardization apart* was introduced in [20, p. 31] and simplified in [18, p. 46]. Its subseqent evolution has led to more strictness, owing to loopholes [4, p. 11]. The current version is from [2]. The concept appears also under the name *variable separation* [10]. Most often the phrase *fresh renaming of program clauses* is encountered.

[18] Being a renaming, every ***New*** is functor-preserving. Hence, it suffices to define it on variables, if that is more comfortable.

In programming practice, usually the original variables from the query and from the program clauses are forsaken. Instead, variables are represented by special variable-free terms, like those obtained with the traditional utility `numbervars`/3 [19]: `'$VAR'(0)`, `'$VAR'(1)` and so forth. This practice has found a way into the ISO standard [8], and the terms `'$VAR'(`n`)` are output by default as variables $A, B, \ldots, Z, A1, B1, \ldots$. Fresh renaming consists here in advancing the counter of spent variable indices.

Definition 5.6 (all-new fresh renaming, NA)**.** Assume variables to be enumerated, $\boldsymbol{V} = \{\mathtt{V}_1, \mathtt{V}_2, \ldots\}$. The algorithm NA renames t apart from s as follows. If t has k variables and $n = \max_i(\mathtt{V}_i \mathrel{\hat{\in}} s)$, then $\mathsf{NT}_s(t)$ is obtained from t by replacing variables in order of appearance by $\mathtt{V}_{n+1}, ..., \mathtt{V}_{n+k}$.

5.2.2 A missing property

Encouraged by all the nice properties of "the" classical unification algorithm and its adaptations, we might wonder whether a fresh-renaming algorithm ***New*** can also be renaming-compatible, i.e. whether for any renaming ρ and terms s, t may hold $\boldsymbol{New}_{\rho(s)}(\rho(t)) = \rho(\boldsymbol{New}_s(t))$.

Counter-Example 5.7 (can ***New*** be renaming-compatible?)**.** Let s be an arbitrary term. Assume ***New*** to be renaming-compatible, and let $\boldsymbol{New}_s(\mathtt{[X,Y]}) = \mathtt{[U,W]}$. Now let $\rho := \left(\begin{smallmatrix}\mathtt{U}&\mathtt{W}\\ \mathtt{W}&\mathtt{U}\end{smallmatrix}\right)$. Clearly, $\rho(\mathtt{[X,Y]}) = \mathtt{[X,Y]}$. Due to $\mathtt{[U,W]} \not\bowtie s$, we also have $\rho(s) = s$. Hence, $\boldsymbol{New}_{\rho(s)}(\rho(\mathtt{[X,Y]})) = \boldsymbol{New}_s(\mathtt{[X,Y]}) \neq \mathtt{[W,U]} = \rho(\boldsymbol{New}_s(\mathtt{[X,Y]}))$.

In hindsight, this negative result appears trivial, yet it causes a claim that holds in HCL to break if fresh renaming is done by an algorithm:

Lemma 5.8 (HCL only)**.** *Assume free choice of mgu and program clause renaming. If ρ is a renaming and $G \hookrightarrow\!\triangleright_{\mathtt{s}_1} G_1 \ldots \hookrightarrow\!\triangleright_{\mathtt{s}_n} G_n$, then $\rho(G) \hookrightarrow\!\triangleright_{\rho(\mathtt{s}_1)} \rho(G_1) \ldots \hookrightarrow\!\triangleright_{\rho(\mathtt{s}_n)} \rho(G_n)$.*

In the presence of $\boldsymbol{S}$, the given proof does not work. More importantly, the claim does not hold. To see this, let $\hat{\mathscr{K}}$ be `a(X,Y) :- b(X,Y)`. We take $\boldsymbol{S} := \mathsf{NT}$, hence $\mathscr{K} := \boldsymbol{S}_{\mathtt{a(X,Y)}}(\hat{\mathscr{K}}) = \mathtt{a(X_1,Y_1) :- b(X_1,Y_1)}$. Unification shall be done by $\boldsymbol{U} := \mathsf{MM}$ but used in the traditional way, with clause head on the right, giving $\boldsymbol{U}(\mathtt{a(X,Y)} = \mathtt{a(X_1,Y_1)}) = \left(\begin{smallmatrix}\mathtt{X}&\mathtt{Y}\\ \mathtt{X_1}&\mathtt{Y_1}\end{smallmatrix}\right)$. Finally, set $\rho := \left(\begin{smallmatrix}\mathtt{X_1}&\mathtt{Y_1}\\ \mathtt{Y_1}&\mathtt{X_1}\end{smallmatrix}\right)$. For the query `a(X,Y)` we obtain $\mathscr{D} : \mathtt{a(X,Y)} \hookrightarrow\!\triangleright_{\mathscr{K}:\left(\begin{smallmatrix}\mathtt{X}&\mathtt{Y}\\ \mathtt{X_1}&\mathtt{Y_1}\end{smallmatrix}\right)} \mathtt{b(X_1,Y_1)}$, and for the query $\rho(\mathtt{a(X,Y)})$ the same, instead of $\rho(\mathscr{D})$.

Comparing Prolog derivations of mutually variant queries requires a bit of bookkeeping [11, Sec. 6].

6 Related work

In [10], *variable separation* is synonymous with standardizing-apart in SLD-derivations. Here we regard it in a broader sense, as variable-disjointness between (parts of) arbitrary terms. The themes of *variable separation* and *variable conservation* in the context of substitutions appear to be novel.

The main idea of *modular rephrasing* of the binding rule, i.e. to put away ineligible equations and drop the context dependence, is not new: In [3, p. 449] a variant of Martelli-Montanari's algorithmic scheme was shown, with equations divided in a "to-do" and a "result" zone, and with the binding rule

$$\textbf{variable elimination}\ \ \{x = t\} \cup E;\ S \rightsquigarrow \binom{x}{t}(E);\ \binom{x}{t}(S) \cup \{x = t\},\ \text{if } x \mathrel{\hat{\notin}} t$$

where S is the emergent solved form. This is equivalent to the binding rule of RMM, which can be seen from $\mathit{Subst}(\sigma_n(S_{n-1})) \cup \sigma_n = \sigma_n \cdot \mathit{Subst}(S_{n-1}) = \sigma_n \cdot \ldots \cdot \sigma_1$, owing to $\sigma_n \not\bowtie \mathit{Dom}(\mathit{Subst}(S_{n-1}))$ (Lemma 2.5). Where the two approaches differ is in result building. The explicit composition favoured in

our rephrasing enabled an easy discernment (and proof) of a "reduction" property inherent to RMM rules (Lemma 4.9), which then led to recognizing this property in MM as well, and so forth (Subsection 4.2).

Ordering of variables by mapping to natural numbers was proposed by Bol in [4], incorporated in the orientation rule. We modified the binding rule as well, thus re-establishing rule disjointness.

7 Summary

A somewhat neglected aspect of Prolog derivations is addressed: algorithms for unification and "fresh" renaming, with focus on simplicity and desirable mathematical properties. For fresh renaming, *structure-independent* algorithms NT and NA are presented.

For unification, our starting point was the left-to-right deterministic version of Martelli-Montanari's algorithmic scheme, here denoted MM. Two kinds of adaptations are discussed:

- addition of well-founded variable orderings as advocated in [4], to control variable renaming; this resulted in the algorithm MMB
- a modular rephrasing of the binding rule similarly to [3], to obtain uniform rules; this resulted in the algorithms RMM and RMMB (the former is coincident with MM and Robinson's algorithm, the latter with MMB)

Owing to the modular rephrasing, two "tactics" underlying the operation of Martelli-Montanari-type unification algorithms, *simplification* and *reduction*, were uncovered, which led to a hitherto unreported *iteration* property (compatibility with LD-resolution).

Underway, we address *variable separation* of substitutions, including *cascading* (appears in unification and SLD-derivations), and *variable conservation* when applying or composing substitutions (necessary for detailed operational models of Prolog).

Acknowledgement

Many thanks to Ch. Beierle for his help in debugging the terminology of an earlier draft.

References

[1] G. Amato & F. Scozzari (2009): *Optimality in goal-dependent analysis of sharing*. *Theory and Practice of Logic Programming* 9(5), pp. 617–689, doi:10.1017/S1471068409990111.

[2] K. R. Apt (1997): *From logic programming to Prolog*. Prentice Hall.

[3] F. Baader & W. Snyder (2001): *Unification theory*. In J. A. Robinson & A. Voronkov, editors: *Handbook of automated reasoning*, Elsevier.

[4] R. N. Bol (1992): *Generalizing completeness results for loop checks in logic programming*. *Theor. Comp. Sci.* 104(1), pp. 3–28, doi:10.1016/0304-3975(92)90164-B.

[5] E. Eder (1985): *Properties of substitutions and unifications*. *J. Symbolic Computation* 1(1), pp. 31–46, doi:10.1016/S0747-7171(85)80027-4.

[6] J. Gallier (2011): *Discrete mathematics*. Springer-Verlag, doi:10.1007/978-1-4419-8047-2.

[7] J. Herbrand (1930): *Recherches sur la Théorie de la Démonstration*. Ph.D. thesis, Université de Paris. Available from `http://www.numdam.org/item/THESE_1930__110__1_0`.

[8] ISO/IEC JTC 1/SC 22 (1995): *ISO/IEC 13211-1-1995. Information technology - Programming languages - Prolog - Part 1: General core*. `https://www.iso.org/standard/21413.html`.

[9] J. W. Klop & R. de Vrijer et al., editors (2003): *TeReSe: Term Rewriting Systems*, chapter First-order term rewriting systems. Cambridge University Press. Excerpt on `http://www.cs.vu.nl/~tcs/trs`.

[10] H.-P. Ko & M. E. Nadel (1991): *Substitution and refutation revisited*. In: *Proc. of ICLP*, pp. 679–692.

[11] M. Kulaš (2017): *A practical view on renaming*. In S. Schwarz & J. Voigtländer, editors: *Proc. WLP'15/'16 and WFLP'16*, *EPTCS* 234, pp. 27–41, doi:10.4204/EPTCS.234.3.

[12] M. Kulaš (2017): *A term matching algorithm and substitution generality*. Technical Report IB 376-11/2017, FernUniversität in Hagen. `http://nbn-resolving.de/urn:nbn:de:hbz:708-dh5549`.

[13] M. Kulaš (2019): *Toward a concept of derivation for Prolog*. To appear.

[14] J. L. Lassez, M. J. Maher & K. Marriott (1988): *Unification revisited*. In M. Boscarol et al., editors: *Foundations of Logic and Functional Programming*, *LNCS* 306, Springer-Verlag, pp. 67–113, doi:10.1007/3-540-19129-1_4.

[15] J. W. Lloyd (1987): *Foundations of logic programming*, 2. edition. Springer-Verlag, doi:10.1007/978-3-642-83189-8.

[16] A. Martelli & U. Montanari (1982): *An efficient unification algorithm*. *ACM Trans. on Prog. Lang. and Systems* 4(2), pp. 258–282, doi:10.1145/357162.357169.

[17] C. Palamidessi (1990): *Algebraic properties of idempotent substitutions*. In: *Proc. 17th ICALP*, *LNCS* 443, Springer-Verlag, pp. 386–399, doi:10.1007/BFb0032046.

[18] G. D. Plotkin (1971): *Automatic methods of inductive inference*. Ph.D. thesis, U. of Edinburgh. Available from `http://homepages.inf.ed.ac.uk/gdp`.

[19] Quintus Corp., Palo Alto, CA (1991): *Quintus Prolog language and library*. Release 3.1. Also on `http://quintus.sics.se/isl/quintus/html/quintus`.

[20] J. A. Robinson (1965): *A machine-oriented logic based on the resolution principle*. *J. of ACM* 12(1), pp. 23–41, doi:10.1145/321250.321253.

[21] Dept. of Comp. Sci. VU Amsterdam (2013): *Study materials on term-rewriting systems*. `http://www.cs.vu.nl/~tcs/trs`.

A Proofs

Legacy 2.3 (idempotent composition, [2]). *Let σ and θ be idempotent. If $Ran(\theta) \not\bowtie Dom(\sigma)$, then $\theta \cdot \sigma$ is also idempotent.*

Proof. If $x \,\hat{\in}\, Ran(\theta \cdot \sigma)$, then either $x \,\hat{\in}\, Ran(\theta)$, or $x \,\hat{\in}\, Ran(\sigma), x \notin Dom(\theta)$. In the latter case idempotency of σ additionally gives $x \notin Dom(\sigma)$, so summarily $x \notin Dom(\theta \cdot \sigma)$. Let us now consider the former case, $x \,\hat{\in}\, Ran(\theta)$. Due to the additional assumption, $x \notin Dom(\sigma)$, and due to idempotency of θ holds $x \notin Dom(\theta)$, so here also $x \notin Dom(\theta \cdot \sigma)$. ♢

Lemma 2.6 (left monotonicity). *If $\sigma \subseteq \theta$ and $Dom(\lambda) \not\bowtie Dom(\theta)$, then $\lambda \cdot \sigma \subseteq \lambda \cdot \theta$.*

Proof. The claim is clearly a corollary of Lemma 2.5. But it can also be proved more directly. The property $\sigma \subseteq \theta$ means that for $x \in Dom(\sigma)$ holds $\sigma(x) = \theta(x)$. Hence,

$$\lambda(\sigma(x)) = \begin{cases} \lambda(\theta(x)), & \text{if } x \in Dom(\lambda \cdot \sigma) \cap Dom(\sigma) \\ \lambda(x), & \text{if } x \in Dom(\lambda \cdot \sigma) \setminus Dom(\sigma) \subseteq Dom(\lambda) \end{cases}$$

Due to $Dom(\lambda) \not\bowtie Dom(\theta)$, the latter value is equal to $\lambda(\theta(x))$, so indeed $\lambda \cdot \sigma \subseteq \lambda \cdot \theta$. ♢

Lemma 2.9 (idempotent composition). *If $\sigma_1, ..., \sigma_n$ are cascading and idempotent, then $\sigma_n \cdot ... \cdot \sigma_1$ is idempotent.*

Proof. It holds $Dom(\sigma_i \cdot ... \cdot \sigma_1) \subseteq Dom(\sigma_i) \cup ... \cup Dom(\sigma_1)$. This and the cascading property (1) give $Ran(\sigma_n) \not\bowtie Dom(\sigma_{n-1} \cdot ... \cdot \sigma_1)$. By induction on n and Legacy 2.3, $\sigma_n \cdot ... \cdot \sigma_1$ is idempotent. ♢

Lemma 3.1. *For any term t and any substitution σ holds $Vars(t) \cup Vars(\sigma) = Vars(\sigma(t)) \cup Vars(\sigma)$.*

Proof. Let $x \mathrel{\mathring{\in}} t$. If $x \notin Dom(\sigma)$, then $x = \sigma(x) \mathrel{\mathring{\in}} \sigma(t)$. By contraposition, any variables from t that are missing in $\sigma(t)$ can be found in $Dom(\sigma)$. Analogously for the possible win. ♢

Lemma 3.2 (conservation I). *For any idempotent substitution σ and any renaming ρ holds $Vars(\rho \cdot \sigma) = Vars(\sigma \cdot \rho) = Vars(\rho) \cup Vars(\sigma)$.*

Proof. The nontrivial half is to prove that any variable x from σ or ρ appears in both $\rho \cdot \sigma$ and $\sigma \cdot \rho$. Here we employ $Dom(\rho) = Ran(\rho) = Vars(\rho)$ (by Legacy 1.2) and $Dom(\sigma) \not\bowtie Ran(\sigma)$ (by Legacy 2.1).

Case $\rho \cdot \sigma$: If $\rho(\sigma(x)) \neq x$, then $x \in Dom(\rho \cdot \sigma)$. Otherwise, $\sigma(x)$ must be a variable, of two kinds:
First, $\sigma(x) = x$; then also $\rho(x) = x$, i.e. $x \mathrel{\mathring{\notin}} \rho$, $x \notin Dom(\sigma)$. This leaves $x \in Ran(\sigma)$, i.e. for some $z \neq x$ holds $x \mathrel{\mathring{\in}} \sigma(z)$, so $x = \rho(x) \mathrel{\mathring{\in}} \rho(\sigma(z))$, hence $x \mathrel{\mathring{\in}} Ran(\rho \cdot \sigma)$.
Second, $\sigma(x) = y \neq x$ and $\rho(y) = x$. Since $y \mathrel{\mathring{\in}} Ran(\sigma)$, by idempotency $y \notin Dom(\sigma)$, hence $y/x \in \rho \cdot \sigma$, so $x \mathrel{\mathring{\in}} Ran(\rho \cdot \sigma)$.

Case $\sigma \cdot \rho$: If $\sigma(\rho(x)) \neq x$, then $x \in Dom(\rho \cdot \sigma)$. Otherwise, there are two subcases for $\rho(x)$:
First, $\rho(x) = x$; then also $\sigma(x) = x$, i.e. $x \mathrel{\mathring{\notin}} \rho$, $x \notin Dom(\sigma)$. This leaves $x \in Ran(\sigma)$, i.e. for some $z \neq x$ holds $x \mathrel{\mathring{\in}} \sigma(z)$. If $z \mathrel{\mathring{\notin}} \rho$, then $z/\sigma(z) \in \sigma \cdot \rho$, QED. If $z \mathrel{\mathring{\in}} \rho$, there is y with $y/z \in \rho$, $y \neq x$, so $y \neq \sigma(z)$ and $y/\sigma(z) \in \sigma \cdot \rho$, QED.
Second, $\rho(x) = y \neq x$ and $\sigma(y) = x$; here $x \mathrel{\mathring{\in}} Ran(\rho)$, $x \notin Dom(\sigma)$, therefore $x \in Dom(\sigma \cdot \rho)$. ♢

Lemma 3.3 (conservation II). *If σ, θ are idempotent and $Dom(\sigma) \not\bowtie Dom(\theta)$, then $Vars(\theta \cdot \sigma) = Vars(\theta) \cup Vars(\sigma)$.*

Proof. It suffices to prove that any variable $x \mathrel{\mathring{\in}} \sigma, \theta$ appears in $\theta \cdot \sigma$. Since $Dom(\theta) \not\bowtie Dom(\sigma)$, by Lemma 2.6 holds $\theta \subseteq \theta \cdot \sigma$. Thus it remains to consider the case $x \mathrel{\mathring{\in}} \sigma$, $x \mathrel{\mathring{\notin}} \theta$.
If $x \in Dom(\sigma)$: It is not possible that $x/y \in \sigma$ and $y/x \in \theta$, so $x \in Dom(\theta \cdot \sigma)$.
If $x \mathrel{\mathring{\in}} Ran(\sigma)$: Since $x \notin Dom(\theta)$, clearly $x \mathrel{\mathring{\in}} Ran(\theta \cdot \sigma)$. ♢

Lemma 4.4 (binding rule preserves mgus). *For every equation set E and every I-binding $x = t$:*

1. $UnifSet(\{x = t\} \cup E) = UnifSet(\binom{x}{t}(E)) \cdot \binom{x}{t}$
2. $UnifSet(\binom{x}{t}(E)) \cdot \binom{x}{t} = UnifSet(\{x = t\} \cup \binom{x}{t}(E))$

Both claims also hold with MguSet instead of UnifSet, or with the argument being an equation sequence instead of a set.

Proof. *Part 1*: Subset claim can be proved by generality of mgus, similarly to [2, Lemma 2.24]: Assume $\theta \in \mathit{UnifSet}(\{x=t\} \cup E)$, so moreover $\theta \in \mathit{UnifSet}(x=t)$. Hence, by Lemma 4.3 and Legacy 1.3 there is λ with $\theta = \lambda \cdot \binom{x}{t}$. By choice of θ then $\lambda \cdot \binom{x}{t} \in \mathit{UnifSet}(E)$, so $\lambda \in \mathit{UnifSet}(\binom{x}{t}(E))$. Superset claim is trivial: if λ unifies $\binom{x}{t}(E)$, by Lemma 4.3 we know $\lambda \cdot \binom{x}{t}$ unifies $\{x=t\} \cup E$.

Part 2: For subset claim, assume λ unifies $\binom{x}{t}(E)$. Then $\lambda \cdot \binom{x}{t}$ unifies $\{x=t\} \cup \binom{x}{t}(E)$, by Lemma 4.3 and idempotency of $\binom{x}{t}$. For superset claim, assume $\theta \in \mathit{UnifSet}(\{x=t\} \cup \binom{x}{t}(E))$, so moreover $\theta \in \mathit{UnifSet}(x=t)$. Hence, there is again λ with $\theta = \lambda \cdot \binom{x}{t}$. By choice of θ then $\lambda \cdot \binom{x}{t} \in \mathit{UnifSet}(\binom{x}{t}(E))$, so $\lambda \in \mathit{UnifSet}(\binom{x}{t} \cdot \binom{x}{t}(E)) = \mathit{UnifSet}(\binom{x}{t}(E))$, again by idempotency. ♢

Legacy 4.5 (solved form, [16][2]). *For any E_0 and any choice of working equation, the transformation in Figure 1 is certain to stop. If the stop is due to failure, then E_0 is not unifiable. Otherwise, the final set E is in solved form and $\mathit{Subst}(E)$ is an idempotent mgu of E_0.*

Proof. Termination can be proved as in [2, p. 34], using lexicographic order $\succ$ on triples of natural numbers, built by a *measure* of an equation set $\mathscr{M}(E) := (\#\neg bs(E), \#lfun(E), \#(E))$. Here $\#\neg bs(E)$ is the number of variables in E which are not bound singletons[19]. Next, $\#lfun(E)$ is the number of functor occurrences on the left sides in E, and $\#(E)$ is the number of equations in E.

Clearly, transformation of $\emptyset$ stops. If a step results in failure, transformation stops as well. Otherwise, the step leads from an equation set E to an equation set E', while decreasing at least one component of the measure, without increasing[20] the earlier components (see Figure 5). Hence $\mathscr{M}(E) \succ \mathscr{M}(E')$.

Since the measure cannot decrease indefinitely, transformation must come to a stop.

If the stop was due to failure, then E_0 was not unifiable, due to Lemma 4.4. Otherwise, the final E must be in solved form. One (clearly idempotent) mgu of such E is its associated substitution $\mathit{Subst}(E)$. Again by Lemma 4.4, $\mathit{Subst}(E) \in \mathit{MguSet}(E_0)$. ♢

rule	$\#\neg bs(E)$	$\#lfun(E)$	$\#(E)$
binding	$<$	-	-
decomposition	$\leq$	$<$	-
orientation	$\leq$	$<$	-
cleaning	$\leq$	$=$	$<$

Figure 5: Measure decrease by Martelli-Montanari rules

Lemma 4.7 (simplification). *If E is not already in form $x=t, E'$ with $x \notin t$, then the rules common to* MM *and* RMM *will either bring it in this form, or stop at $\square$, or stop at a failure. In each case, the unifiers will be preserved.*

Proof. The common rules obviously preserve unifiers. The main claim shall be proved by induction on lexicographically ordered pairs of natural numbers, in a similar vein as in the termination proof above. Let $\mathscr{L}(n,k)$ mean the claim holds for equation sequences with n functor occurrences on the left and k equations. Clearly, $\mathscr{L}(0,0)$ holds. Assume now a non-empty equation sequence with parameters n,k.

[19] In place of our "bound singleton", [2, p. 34] uses the notion "solved variable". While it does sound less complicated, it could be misunderstood as suggesting that in a solved form all variables need to be solved.

[20] In particular, no bound singleton can be lost by applying Martelli-Montanari rules, so their number cannot decrease. Since the total number of variables in E cannot increase, $\#\neg bs(E)$ cannot increase.

If the leftmost equation is a binding, either the claim $\mathscr{L}(n,k)$ immediately holds, or it will be reduced to $\mathscr{L}(n,k-1)$. Namely, a binding $x=t$ either satisfies $x \not\in t$ (QED), or $x \mathring{\in} t, x \neq t$ (**failure**: occurs-check, QED), or it is $x=x$ (to be removed by **cleaning**).

Otherwise, the leftmost equation has functors on the left, so if we don't get **failure**: clash (QED), then **decomposition** or **orientation** will reduce the claim $\mathscr{L}(n,k)$ to $\mathscr{L}(m,\_)$ with $0 \leq m < n$.

By well-founded induction [6, Ch. 5.3], $\mathscr{L}(n,k)$ holds for any n,k, so the claim holds. Observe, however, that not every n,k makes sense for the claim: in case $k=0$ must hold $n=0$. This, of course, is due to *dependence* of arguments, namely n,k depend on the "real" argument, the equation sequence E, and provide a *measure* of E, akin to the one in the termination proof. ♢

Theorem 4.10 (RMM). *For any (even empty) equation sequences E and E' holds*

1. *if E is unifiable, then $\mathsf{RMM}((E\,,E')) = \mathsf{RMM}(\sigma(E')) * \sigma$, where $\sigma \in \mathit{UnifSet}(E)$*
2. *otherwise, $\mathsf{RMM}((E\,,E'))$ must fail.*

Proof. Since RMM transforms an equation sequence from left to right, and the new binding rule is independent of the rest sequence, we know that RMM shall choose the same[21] equations to transform in E as in the concatenation E,E', as long as E is not "spent".

For both parts of the claim we shall use induction on the number of variables in E.

Part 1, E is unifiable: For no variables, applying Lemma 4.7 on $\mathsf{RMM}(E)$ must give $\mathsf{RMM}((E,E')) = \mathsf{RMM}(E')$, since failure cannot be reached (common rules preserve unifiers) and neither can an I-binding, obviously. Assume the claim holds for at most n variables, and let E be a sequence with $n+1$ variables. Again by applying Lemma 4.7 on $\mathsf{RMM}(E)$, either $\mathsf{RMM}((E,E')) = \mathsf{RMM}(E')$, or for some x,t,E'' holds $\mathsf{RMM}((E,E')) = \mathsf{RMM}((x=t,E'',E'))$.

In the former case, $\mathsf{RMM}(E) = \varepsilon$ so since unifiers are preserved $\varepsilon \in \mathit{UnifSet}(E)$, QED.

In the latter case, by Lemma 4.9 we get $\mathsf{RMM}((E,E')) = \mathsf{RMM}((\binom{x}{t}(E''), \binom{x}{t}(E'))) * \binom{x}{t}$, so by preserving of unifiers and by inductive hypothesis there is $\theta \in \mathit{UnifSet}(\binom{x}{t}(E''))$ such that $\mathsf{RMM}((E,E')) = \mathsf{RMM}(\theta(\binom{x}{t}(E'))) * \theta * \binom{x}{t}$. By Lemma 4.4, $\theta \cdot \binom{x}{t} \in \mathit{UnifSet}(x=t,E'')$, QED.

Part 2, E is not unifiable: By Lemma 4.7, E must reach one of the three forms, but due to preservation of unifiers it cannot be $\square$. In case of no variables that means failure (QED). In the inductive case, either we reach failure (QED), or for some x,t,E'' holds $\mathsf{RMM}((E,E')) = \mathsf{RMM}((x=t,E'',E')) = \mathsf{RMM}((\binom{x}{t}(E''), \binom{x}{t}(E'))) * \binom{x}{t}$. By Lemma 4.4, $x=t,E''$ and $\binom{x}{t}(E'')$ are equi-unifiable, but the latter has less variables, so by inductive hypothesis it must reach failure, so $\mathsf{RMM}((E,E'))$ fails as well. ♢

Corollary 4.12 (idempotence for RMM). *Assume $\mathsf{RMM}(E)$ succeeds, and $\sigma := \mathsf{RMM}(E)$. If the binding rule was not applied, then $\sigma = \varepsilon$, otherwise $\sigma = \sigma_n \cdot \ldots \cdot \sigma_1 \neq \varepsilon$, where $\sigma_1, \ldots, \sigma_n$ are the respective I-bindings. At any rate, σ is idempotent.*

Proof. I-bindings satisfy (3), which is a special case of (1), so the binding sequence $\sigma_1, \ldots, \sigma_n$ is cascading. Thus, by Corollary 3.4 $\sigma \neq \varepsilon$. By Lemma 2.9, σ is idempotent as well. ♢

Theorem 4.14 (coincidence). *For every equation sequence E holds $\mathsf{RMM}(E) = \mathsf{MM}(E)$.*

[21] This clearly does not hold for MM.

Proof. By induction on the number of variables in E. In case of no variables, using Lemma 4.7 we obtain $\mathsf{RMM}(E) = \varepsilon$, or $\mathsf{RMM}(E) = \mathit{Failure}(\mathit{Info})$, and the same must hold for $\mathsf{MM}(E)$, due to the same common rules applied. Assume the coincidence holds for sequences of up to n variables, and consider a sequence E of $n+1$ variables. By Lemma 4.7, we either again have a halting case $\mathsf{RMM}(E) = \varepsilon$ or $\mathsf{RMM}(E) = \mathit{Failure}(\mathit{Info})$, with sure coincidence to $\mathsf{MM}(E)$, or we have a binding case $\mathsf{RMM}(E) = \mathsf{RMM}((x \mathbin{=} t \mathbin{,} E'))$ with $x \mathbin{\mathring{\notin}} t$. In the latter case, we also have $\mathsf{MM}(E) = \mathsf{MM}((x \mathbin{=} t \mathbin{,} E'))$ with $x \mathbin{\mathring{\notin}} t$. Using Lemma 4.9 and its pendant Lemma 4.13 eliminates x and completes the inductive step. ♢

Theorem 4.18 (MMB). *For any function $\mathit{Tag}\colon \mathbf{V} \to \mathbf{N}$ and any equation sequence E_0, $\mathsf{MMB}_{\mathit{Tag}}(E_0)$ stops and, in case of unifiable E_0, produces a Tag-biased mgu of E_0.*

Proof. Termination can be proved with an equations' measure that also takes care of bias, $\mathscr{M}_{\mathsf{MMB}}(E) := (\#\neg \mathit{bbs}(E), \#\mathit{lfun}(E), \#\neg b(E), \#(E))$, see Figure 6.

rule	$\#\neg \mathit{bbs}(E)$	$\#\mathit{lfun}(E)$	$\#\neg b(E)$	$\#(E)$
binding (MMB)	$<$	-	-	-
decomposition	$\leq$	$<$	-	-
orientation (MMB)	$\leq$	$\leq$	$<$	-
cleaning	$\leq$	$=$	$=$	$<$

Figure 6: Measure decrease by rules of MMB

Here $\#\neg \mathit{bbs}(E)$ is the number of variables in E that are not "biased" bound singletons (i.e., x that occurs just once, in $x = t$ with $\mathit{Tag}(x) \geq \mathit{Tag}(t)$) and $\#\neg b(E)$ is the number of "non-biased" equations in E (i.e., $s = t$ with $\mathit{Tag}(s) < \mathit{Tag}(t)$).

MMB unifies, which can be seen as in the proof of Legacy 4.5: upon termination, if the stop was due to failure, then E_0 was not unifiable, due to Lemma 4.4. Otherwise, the final E must be in solved form. Clearly, any equation $x = t$ in the solved form must satisfy $\mathit{Tag}(x) \geq \mathit{Tag}(t)$. ♢

Lemma 4.19 (bias is compositional). *If σ, θ are Tag-biased, their composition is Tag-biased as well.*

Proof. For each $x \in \mathit{Dom}(\theta \cdot \sigma)$ we need to prove $\mathit{Tag}(x) \geq \mathit{Tag}(\theta(\sigma(x)))$.

First consider the case $x \in \mathit{Dom}(\sigma)$. If $\sigma(x) \in \mathit{Dom}(\theta)$, by bias of σ and bias of θ clearly $\mathit{Tag}(x) \geq \mathit{Tag}(\sigma(x)) \geq \mathit{Tag}(\theta(\sigma(x)))$, i.e. the claim holds. Otherwise, $\sigma(x) \notin \mathit{Dom}(\theta)$, so $\theta(\sigma(x)) = \sigma(x)$, hence by bias of σ the claim again holds.

In the remaining case $x \in \mathit{Dom}(\theta) \setminus \mathit{Dom}(\sigma)$, the claim holds by $\theta(\sigma(x)) = \theta(x)$ and bias of θ. ♢

Lemma 4.26 (*MguSet*). *For any renaming ρ and equation sequence E: $\mathit{MguSet}(\rho(E)) = \rho(\mathit{MguSet}(E))$.*

Proof. By definition of substitution renaming, for any substitution σ

$$\rho(\sigma)(\rho(E)) = \rho(\sigma(E)) \tag{A.8}$$

Hence, if $\sigma \in \mathit{UnifSet}(E)$, then $\rho(\sigma) \in \mathit{UnifSet}(\rho(E))$.

For the other direction, assume $\theta \in \mathit{UnifSet}(\rho(E))$. Since we need a substitution of the form $\rho(\_)$, and by Legacy 1.4 holds $\rho(\sigma) = \rho \cdot \sigma \cdot \rho^{-1}$, we set $\sigma := \rho^{-1} \cdot \theta \cdot \rho$. Then $\theta = \rho(\sigma)$, so by (A.8) holds that $\rho \cdot \sigma$ unifies E, thus $\sigma \in \mathit{UnifSet}(E)$ (recall that equality is compatible with renaming).

Summarily, $\rho(\mathit{UnifSet}(E)) = \mathit{UnifSet}(\rho(E))$, so their maximally general subsets are the same. ♢

Lemma 5.1. *Idempotent mgus $\sigma_1,...,\sigma_n$ in a SLD-derivation $G_0 \hookrightarrow_{\mathscr{K}_1:\sigma_1} G_1 \hookrightarrow_{\mathscr{K}_2:\sigma_2} ...$ are cascading.*

Proof. Cascading is due to idempotence (and thus relevance) of mgus, and to standardizing-apart. To see this, let us pick an index i and one of its valid increments k and show $\sigma_{i+k} \not\bowtie Dom(\sigma_i)$.

By definition of SLD-resolution, $Vars(G_m) \subseteq Vars(G_{m-1}) \cup Vars(\mathscr{K}_m) \cup Vars(\sigma_m)$ for any $m \geq 1$. Due to relevance of idempotent mgus (Legacy 2.4), an mgu σ_m can only have variables from the current query G_{m-1} and the current input clause $\mathscr{K}_m$, i.e. $Vars(\sigma_m) \subseteq Vars(G_{m-1}) \cup Vars(\mathscr{K}_m)$ for any $m \geq 1$. Hence, $Vars(G_m) \subseteq Vars(G_{m-1}) \cup Vars(\mathscr{K}_m)$, so by iteration $Vars(G_{i+k-1}) \subseteq Vars((G_i, \mathscr{K}_{i+1}, ..., \mathscr{K}_{i+k-1}))$.

Combining it with the previosly obtained $Vars(\sigma_{i+k}) \subseteq Vars(G_{i+k-1}) \cup Vars(\mathscr{K}_{i+k})$ gives

$$Vars(\sigma_{i+k}) \subseteq Vars((G_i, \mathscr{K}_{i+1}, ..., \mathscr{K}_{i+k})) \tag{A.9}$$

Applying an idempotent mgu eliminates its core variables (Corollary 2.2), so

$$Dom(\sigma_i) \not\bowtie G_i \tag{A.10}$$

Due to standardizing-apart of input clauses,

$$\sigma_i \not\bowtie \mathscr{K}_{i+m} \text{ for any } m \geq 1 \tag{A.11}$$

On the strength of (A.9) – (A.11) we obtain $Dom(\sigma_i) \not\bowtie \sigma_{i+k}$. ♢

Theorem 5.2 (partial answer)**.** *Idempotent mgus $\sigma_1,...,\sigma_n$ in a SLD-derivation satisfy*

1. *$\sigma_n \cdot ... \cdot \sigma_1$ is idempotent*
2. *$\sigma_n \subseteq \sigma_n \cdot \sigma_{n-1} \subseteq ... \subseteq \sigma_n \cdot ... \cdot \sigma_1$*
3. *$Vars(\sigma_n \cdot ... \cdot \sigma_1) = Vars(\sigma_n) \cup ... \cup Vars(\sigma_1)$*

Proof. Owing to Lemma 5.1, $\sigma_1,...,\sigma_n$ are cascading, so respectively Lemma 2.9, Corollary 2.7 and Corollary 3.4 can be applied. ♢

Lemma 5.8 (HCL only)**.** *Assume free choice of mgu and program clause renaming. If ρ is a renaming and $G \hookrightarrow_{\mathsf{s}_1} G_1 \ ... \ \hookrightarrow_{\mathsf{s}_n} G_n$, then $\rho(G) \hookrightarrow_{\rho(\mathsf{s}_1)} \rho(G_1) \ ... \ \hookrightarrow_{\rho(\mathsf{s}_n)} \rho(G_n)$.*

Proof. Assume M,A,N was resolved with input clause $\mathscr{K} := (H :- B)$ and mgu $\sigma \in MguSet(H = A)$, i.e. with score $\mathsf{s} = \mathscr{K}{:}\sigma$, giving resolvent $\sigma((M,B,N))$. By Lemma 4.26, $\rho(\sigma) \in \rho(MguSet(H = A)) = MguSet(\rho(H = A))$ so $\rho((M,A,N))$ can be resolved with input clause $\rho(\mathscr{K})$ and mgu $\rho(\sigma)$, giving resolvent $\rho(\sigma((M,B,N)))$, due to $\rho(\sigma)(\rho(x)) = \rho(\sigma(x))$.

Standardizing-apart is satisfied as well, since the relation "variable-disjoint" is compatible with renaming. ♢

Verzeichnis der zuletzt erschienenen Informatik-Berichte

[367] Hoyrup, M., Ko, K., Rettinger, R., Zhong, N.:
CCA 2013 Tenth International Conference on Computability and Complexity in Analysis (extended abstracts), 7/2013

[368] Beierle, C., Kern-Isberner, G.:
4th Workshop on Dynamics of Knowledge and Belief (DKB-2013), 9/2013

[369] Güting, R.H., Valdés, F., Damiani, M.L.:
Symbolic Trajectories, 12/2013

[370] Bortfeldt, A., Hahn, T., Männel, D., Mönch, L.:
Metaheuristics for the Vehicle Routing Problem with Clustered Backhauls and 3D Loading Constraints, 8/2014

[371] Güting, R. H., Nidzwetzki, J. K.:
DISTRIBUTED SECONDO: An extensible highly available and scalable database management system, 5/2016

[372] M. Kulaš
A practical view on substitutions, 7/2016

[373] Valdés, F., Güting, R.H.:
Index-supported Pattern Matching on Tuples of Time-dependent Values, 7/2016

[374] Sebastian Reil, Andreas Bortfeldt, Lars Mönch:
Heuristics for vehicle routing problems with backhauls, time windows, and 3D loading constraints, 10/2016

[375] Ralf Hartmut Güting and Thomas Behr:
Distributed Query Processing in Secondo, *12/2016*

[376] Marija Kulaš:
A term matching algorithm and substitution generality, *11/2017*

[377] Jan Kristof Nidzwetzki, Ralf Hartmut Güting:
BBoxDB - A Distributed and Highly Available Key-Bounding-Box-Value Store, *5/2018*