A Practical View on Renaming

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

We revisit variable renaming from a practitioner's point of view, presenting concepts we found useful in dealing with operational semantics of pure Prolog. A concept of *relaxed core representation* is introduced, upon which a concept of *prenaming* is built. Prenaming formalizes the intuitive practice of renaming terms by just considering the necessary bindings, where now some passive "bindings" x/x may be necessary as well. As an application, a constructive version of variant lemma for implemented Horn clause logic has been obtained. There, prenamings made it possible to incrementally handle new (*local*) variables.

1 Introduction

For logic program analysis or formal semantics, the issue of renaming variables and generally handling substitutions is inevitable. Yet the image of substitutions in logic programming research is a somewhat tainted one, at least since it has been pointed out by H.-P. Ko [16, p. 148] that the original claim of strong completeness of SLD-resolution needs to be amended, due to a counter-example using the fact that $\binom{x}{f(y,z)}$ is not more general than $\binom{x}{f(a,a)}$. The example may look counter-intuitive, but it complies with the definition of substitution generality. Also, by composing substitutions, properties like equivalence, idempotency or restriction are not preserved. Lastly, due to group structure of renamings, permuting any number of variables amounts to "doing nothing", as in $\binom{x\,y}{y\,x} \sim \varepsilon$, and such equivalences are also felt to be counter-intuitive. Hence the prevalent sentiments that substitutions are "a quite hard matter to deal with" ([13]) or "very tricky" ([16]). As a remedy, in the context of aggregating most general unifiers in a logic programming computation some helpful new concepts and operators were proposed, like *parallel composition* instead of traditional composition ([13]) and *resultant* instead of answer substitution ([12]). Still, for almost anyone embarking on a journey of logic program analysis or formal semantics, sooner or later the need for renaming variables and generally handling substitutions in a new context arises.

In case of this author, the need arose while trying to prove adequacy of an operational semantics for pure Prolog, S1:PP [9], and the context was one of *extensibility*: Given is a pair of queries that are alphabetic variants of each other. As their respective S1:PP derivations proceed to develop, new variables may crop up, due to standardization-apart (here called *local variables*, Subsection 6.1), but the status of being variant should hold. This setup is known from the classical *variant lemma* ([11]). Additionally, the corresponding variables need to be *collected*, obtaining at each step the temporary variance between the derivations. As an example, assume the first query is $p(z,u,x)$ and the second $p(y,z,x)$. There is only one relevant renaming, $\rho = \binom{z\,u\,y}{y\,z\,u}$. Now assume in the next step the first derivation acquires the variable y, and the second w. The relevant renaming this time would be $\rho' = \binom{z\,u\,y\,w}{y\,z\,w\,u}$. Clearly, ρ' is not an extension of ρ, which makes it seem unsafe to proceed: are some properties of the previous step now in danger? So the question is, how to "safely" extend a variable mapping. For this purpose, in Section 5 we introduce a slight generalization of renaming, called *prenaming*. It is a mathematical underpinning

S. Schwarz and J. Voigtländer (Eds.): 29th and 30th Workshops on (Constraint) Logic Programming and 24th International Workshop on Functional and (Constraint) Logic Programming (WLP'15/'16/WFLP'16).
EPTCS 234, 2017, pp. 27–41, doi:10.4204/EPTCS.234.3

of the intuitive practice of renaming terms by just considering the necessary bindings, and not worrying whether the result is a permutation. In the above example, renaming $p(z,u,x)$ to $p(y,z,x)$ means mapping $z \mapsto y$, $u \mapsto z$ and $x \mapsto x$. Intuitively, only $z \mapsto y$, $u \mapsto z$ are considered necessary bindings, giving the "renaming" $\binom{z\ u}{y\ z}$. For prenaming, $x \mapsto x$ is necessary as well. It is based on *relaxed core representation*, which is nothing else than allowing some $x \mapsto x$ pairs alongside "real" bindings, as placeholders.

Prenamings relate to and are inspired by previous work as follows. In [14], the concept of *translation* is defined, upon which alphabetic variance and standardization apart are built; this is the same as prenaming but for relaxed core (page 31). A *safe* prenaming is more general than *renaming for a term* from [11], and it maximizes W in the notion of *W-renaming* from [6] (page 33). Also, it generalizes *substitution renaming* from [1] (subsubsection 5.3.2).

In Section 6, prenamings are used to express and prove a propagation claim for implemented Horn clause logic, by means of local variable extension (Lemma 6.3). As a corollary, a variant lemma is obtained (Theorem 6.5). Underway, we touch on the discrepancy between the rather abundant theory of logic programming and a scarcity of mathematical claims for implemented logic programming systems. While there are some formal proofs of properties like nominal unification [17], for logic programming systems or their compilation such are still few and far between, a notable exception being [15]. New concepts like prenaming may be of help.

2 Substitution

First we need a bit of notation. Assume two disjoint sets: a countably infinite set $\boldsymbol{V}$ of *variable*s and a set ***Fun*** of *shape*s. If $W \subseteq \boldsymbol{V}$, any mapping F with $F(W) \subseteq \boldsymbol{V}$ shall be called *variable-pure on* W. A mapping variable-pure on the whole set of variables $\boldsymbol{V}$ shall be simply called *variable-pure*. If $\boldsymbol{V} \setminus W$ is finite, W is said to be *co-finite*. A mapping F is *injective on* W, if whenever $F(x) = F(y)$ for $x, y \in W$ also holds $x = y$. Each $f/n \in \boldsymbol{Fun}$ consists of a *functor* f and an associated number of arguments n, called *arity*. Functors of arity 0 are called *constant*s. Starting from $\boldsymbol{V}$ and ***Fun***, data objects or *terms*[1] are built: Any variable $x \in \boldsymbol{V}$ is a term. If $t_1, \ldots, t_n$ are terms and $f/n \in \boldsymbol{Fun}$, then $f(t_1, \ldots, t_n)$ is a term with *shape* f/n and *constructor* f. In case of $f/0$, the term shall be written without parentheses. If a term s occurs within a term t, we write $s \in t$. The *ordered pair* of terms h and t is written as $[h|t]$, where h is called the *head* and t the *tail* of the pair. A special case is a *non-empty list*, distinguished by its tail being a special term $[]$ called the *empty list*, or a non-empty list itself. A *list of n elements* is the term $[t_1|[t_2|[\ldots[t_n|[]]]]]$, conveniently written as $[t_1, \ldots, t_n]$. Let $Vars(t)$ be the set of variables in the term t. If the terms s and t share a variable, that shall be written $s \bowtie t$. Otherwise, we say s, t are *variable-disjoint*, written as $s \not\bowtie t$.

A recurrent theme in this paper shall be "relevance", meaning "no extraneous variables" relative to some term or terms. It was used in [2, p.38] in unary sense, i.e. no extraneous variables relative to (one) term. This usage shall be reflected in the text as follows: A renaming ρ embedding a prenaming α is a *relevant* embedding, if $Vars(\rho) \subseteq Vars(\alpha)$ (Figure 1). Additionally, relevance in a binary sense, concerning two terms, shall also be needed: A mapping F is *relevant for t_1 to t_2*, if $Dom(F) \subseteq Vars(t_1)$ and $Range(F) \subseteq Vars(t_2)$ (Figure 2, Lemma 6.3).

Definition 2.1 (substitution)**.** A *substitution* θ is a function mapping variables to terms, which is identity almost everywhere. In other words, it is a function θ with domain $Dom(\theta) = \boldsymbol{V}$ such that the set $Core(\theta) := \{x \in \boldsymbol{V} \mid \theta(x) \neq x\}$ is finite.[2]

[1] In Prolog, everything is a term, and so shall *term* be here the topmost syntactic concept.

[2] [7] speaks of *finite support*.

The set $Core(\theta)$ shall be called the *active domain*[3] or *core* of θ, and its elements *active variables*[4] of θ. The set $Ran(\theta) := \theta(Core(\theta))$ is the *active range* of θ. A variable x such that $\theta(x) = x$ shall be called a *passive variable*, or a *fixpoint*, for θ. Also, we say that θ is *active* on the variables from $Core(\theta)$, and *passive* on all the other variables. If $Core(\theta) = \{x_1,\ldots,x_k\}$, where $x_1,\ldots,x_k$ are pairwise distinct variables, and θ maps each x_i to t_i, then θ shall have the *core representation* $\{x_1/t_1,\ldots,x_k/t_k\}$, or the perhaps more visual $\left(\begin{smallmatrix} x_1 & \ldots & x_k \\ t_1 & \ldots & t_k \end{smallmatrix}\right)$. Each pair x_i, t_i is called the *binding* for x_i in θ, denoted by $x_i/t_i \in \theta$. Often we identify a substitution with its core representation, and thus regard it as a syntactical object, a term representing a finite set. So the set of variables of a substitution is defined as $Vars(\theta) := Core(\theta) \cup Vars(Ran(\theta))$.

The notions of restriction and extension of a mapping shall also be transported to core representation: if $\theta \subseteq \sigma$, we say θ is a *restriction* of σ, and σ is an *extension* of θ. The restriction $\theta\restriction_W$ of a substitution θ on a set of variables $W \subseteq \boldsymbol{V}$ is defined as follows: if $x \in W$ then $\theta\restriction_W(x) := \theta(x)$, otherwise $\theta\restriction_W(x) := x$. The restriction of θ upon the variables of t is abbreviated as $\theta\restriction_t := \theta\restriction_{Vars(t)}$.

The *composition* $\theta \cdot \sigma$ of substitutions θ and σ is defined by $(\theta \cdot \sigma)(x) := \theta(\sigma(x))$. Composition may be iterated, written as $\sigma^n := \sigma \cdot \sigma^{n-1}$ for $n \geq 1$, and $\sigma^0 := \varepsilon$. Here $\varepsilon := ()$ is the identity function on $\boldsymbol{V}$. In case a variable-pure substitution ρ is bijective, its inverse shall be denoted as ρ^{-1}. A substitution θ satisfying the equality $\theta \cdot \theta = \theta$ is called *idempotent*.

Definition of substitution is enhanced from variables to arbitrary terms in a structure-preserving way by $\theta(f(t_1,\ldots,t_n)) := f(\theta(t_1),\ldots,\theta(t_n))$. If t is a term, then $\theta(t)$ is an *instance* of t via θ.

Example 2.2. $\left(\begin{smallmatrix} x & w & u & v \\ u & v & x & w \end{smallmatrix}\right) \cdot \begin{pmatrix} u & v & x & y & z & w \\ x & w & y & u & v & z \end{pmatrix} = \begin{pmatrix} \cancel{u} & \cancel{v} & x & y & z & w & \cancel{x} & \cancel{w} & \cancel{u} & \cancel{v} \\ \cancel{u} & \cancel{v} & y & x & w & z & \cancel{u} & \cancel{v} & \cancel{x} & \cancel{w} \end{pmatrix} = \begin{pmatrix} x & y & z & w \\ y & x & w & z \end{pmatrix}$.

3 Renaming

Definition 3.1 (renaming)**.** A *renaming* of variables is a bijective variable-pure substitution.

In [6], it is synonymously called "permutation". We shall reserve the word for the general case where movement of infinitely many variables is possible. Here we synonymously speak of *finite permutation* due to the fact that, being a substitution, any renaming has a finite core, and Legacy 3.3 holds.

Due to structure preserving, if $s \in t$ then $\sigma(s) \in \sigma(t)$. For bijective substitutions (i.e. renamings), the converse property holds as well, giving

Lemma 3.2 (renaming stability of "$=$", "$\in$", "$\not\bowtie$")**.** *Let ρ be a renaming and s,t be terms. Then $s = t$ iff $\rho(s) = \rho(t)$, and also $s \in t$ iff $\rho(s) \in \rho(t)$. As a consequence, $s \not\bowtie t$ iff $\rho(s) \not\bowtie \rho(t)$.*

Legacy 3.3 ([10])**.** *A substitution ρ is a renaming iff $\rho(Core(\rho)) = Core(\rho)$.*

Legacy 3.4 ([6])**.** *Every injective variable-pure substitution is a renaming.*

So composition of renamings is a renaming. The next property is about cycle decomposition of a finite permutation.

Lemma 3.5 (cycles)**.** *Let σ be a variable-pure substitution. It is injective iff for every $x \in \boldsymbol{V}$ there is $n \in \boldsymbol{N}$ such that $\sigma^n(x) = x$.*

Proof. Assume σ injective, and choose $x_0 \in \boldsymbol{V}$. If $\sigma(x_0) = x_0$, we are done. Otherwise, $\sigma^i(x_0) \neq \sigma^{i-1}(x_0)$ for all $i \geq 1$, due to injectivity. Hence, $\sigma^{i-1}(x_0) \in Core(\sigma)$ for every $i \geq 1$. Because of the finiteness of $Core(\sigma)$, there is $m > k \geq 1$ such that $\sigma^m(x_0) = \sigma^k(x_0)$. Due to injectivity, $\sigma^{m-1}(x_0) = \sigma^{k-1}(x_0)$. By

[3] Traditionally called just *domain*. This may be confusing, since in the usual mathematical sense it is always the whole $\boldsymbol{V}$ that is the domain of any substitution.

[4] The name *active variable* appears in [8].

iteration we get $n := m - k$. For the other direction, assume $\sigma(x) = \sigma(y)$, and minimal m, n such that $\sigma^n(x) = x$, $\sigma^m(y) = y$. Consider the case $m \neq n$, say $m > n$. Then $\sigma^{m-n}(y) = \sigma^{m-n}(x) = \sigma^{m-n}(\sigma^n(x)) = \sigma^{m-n}(\sigma^n(y)) = \sigma^m(y) = y$, contradicting minimality of m. Hence $m = n$, so $x = \sigma^n(x) = \sigma^n(y) = y$. $\diamondsuit$

4 Relaxed core representation

If there is a substitution σ mapping a term s on a term t, then it is mapping each variable in s on a subterm of t. It is possible that a variable stays the same, so if we want our mapping to explicitely cover *all* variables in s, as in the promised application (Section 6), then necessarily x/x would have to be tolerated as a ”binding”.

To cater for such wishes, the core of the substitution σ can be relaxed to contain some passive variables, raising those above the rest, as it were. This simple technique is useful beyond the context of renaming, so we assume arbitrary substitutions.

Definition 4.1 (relaxed core)**.** If $Core(\sigma) \subseteq \{x_1, \ldots, x_n\}$, where variables $x_1, \ldots, x_n$ are pairwise distinct, then $\{x_1, \ldots, x_n\}$ shall be called a *relaxed core* and $\begin{pmatrix} x_1 & \ldots & x_n \\ \sigma(x_1) & \ldots & \sigma(x_n) \end{pmatrix}$ shall be called a *relaxed core representation* for σ. If we fix a relaxed core for σ, it shall be denoted $C(\sigma) := \{x_1, \ldots, x_n\}$. The associated range $\sigma(C(\sigma))$ we denote as $R(\sigma)$. The set of variables of σ is as expected, $V(\sigma) := C(\sigma) \cup Vars(R(\sigma))$. To get back to the traditional representation, we denote by $[\sigma]$ the (non-relaxed) core representation of σ.

For extending, substitutions are treated like sets of active bindings, so (disjoint) union may be used:

Definition 4.2 (sum of substitutions)**.** If $\sigma = \begin{pmatrix} x_1 & \ldots & x_n \\ s_1 & \ldots & s_n \end{pmatrix}$ and $\theta = \begin{pmatrix} y_1 & \ldots & y_m \\ t_1 & \ldots & t_m \end{pmatrix}$ are substitutions in relaxed representation such that $\{y_1, \ldots, y_m\} \not\bowtie \{x_1, \ldots, x_n\}$, then $\sigma \uplus \theta := \begin{pmatrix} x_1 & \ldots & x_n & y_1 & \ldots & y_m \\ s_1 & \ldots & s_n & t_1 & \ldots & t_m \end{pmatrix}$ is the *sum* of σ and θ.

For Subsection 6.2, backward compatibility of an extension shall be needed.

Lemma 4.3 (backward compatibility)**.** *Let σ, θ be substitutions and x be a variable. Then $(\sigma \uplus \theta)(x) = \sigma(x)$ iff $\theta(x) = x$.*

Proof. If $x \notin C(\theta)$, then $\theta(x) = x$, and $(\sigma \uplus \theta)(x) = \sigma(x)$. If $x \in C(\theta)$, then $(\sigma \uplus \theta)(x) = \theta(x)$ and also $x \notin C(\sigma)$, hence $\sigma(x) = x$. The condition $(\sigma \uplus \theta)(x) = \sigma(x)$ collapses to $\theta(x) = x$. $\diamondsuit$

Passivity of θ on a term t is guaranteed if σ is ”complete” for t, i.e. lays claim to all its variables:

Definition 4.4 (complete for term)**.** Let σ be given in relaxed core representation. We say that σ is *complete* for t if $Vars(t) \subseteq C(\sigma)$.

In such a case there is no danger that an extension of σ might map t differently from σ:

Corollary 4.5 (backward compatibility)**.** *If σ is complete for t, then for any θ holds: $\sigma \uplus \theta$ is complete for t and $(\sigma \uplus \theta)(t) = \sigma(t)$.*

5 Prenaming

In practice, one would like to change the variables in a term without bothering to check whether this change is a permutation of variables, i.e. a renaming in the sense of Definition 3.1. For example, the term $p(z, u, x)$ can be changed to $p(y, z, x)$ using mapping $z \mapsto y$, $u \mapsto z$, $x \mapsto x$. Let us call such a mapping *prenaming*[5].

[5]Finding an appropriate name can be a struggle. Shortlisted were *pre-renaming* and *proto-renaming*.

Like any substitution, a prenaming α shall also be represented finitely, but in relaxed core representation, in order to capture possible $x \mapsto x$ pairings. The set $C(\alpha)$ is fixed by the terms to map. Obviously, injectivity is important for such a mapping, since $p(z,u,x)$ cannot be mapped on $p(y,y,x)$ without losing a variable. Hence,

Definition 5.1 (prenaming)**.** A *prenaming* α is a variable-pure substitution injective on a finite set of variables $C(\alpha) \supseteq Core(\alpha)$.

Clearly, any renaming is a prenaming. For Theorem 6.5, we need to extend a given prenaming.

Lemma 5.2 (extension of prenaming)**.** *Let* $\alpha = \binom{x_1 \,\dots\, x_n}{y_1 \,\dots\, y_n}$ *and* $\beta = \binom{u_1 \,\dots\, u_m}{v_1 \,\dots\, v_m}$ *be prenamings such that* $\{u_1,\dots,u_m\} \not\bowtie \{x_1,\dots,x_n\}$ *and* $\{v_1,\dots,v_m\} \not\bowtie \{y_1,\dots,y_n\}$*. Then* $\alpha \uplus \beta = \binom{x_1 \,\dots\, x_n\ u_1 \,\dots\, u_m}{y_1 \,\dots\, y_n\ v_1 \,\dots\, v_m}$ *is also a prenaming, with* $C(\alpha \uplus \beta) = C(\alpha) \uplus C(\beta)$ *and* $R(\alpha \uplus \beta) = R(\alpha) \uplus R(\beta)$*.*

Plotkin's concept of *translation* [14, p. 46] corresponds to prenaming without passive bindings. There, the *inverse translation* for $\tau := \binom{z\ u}{y\ z}$ would be $\tau_{inv} := \binom{y\ z}{z\ u}$. Clearly, $\tau_{inv} \cdot \tau = \binom{y\ z}{z\ u} \cdot \binom{z\ u}{y\ z} = \binom{y}{z}$, which is not identity substitution. Although $(\tau_{inv} \cdot \tau)\restriction_{\{z,u\}} = \varepsilon$, so τ is reversible and thus ”safe” to use on $\{z,u\}$, one might instinctively be wary of the possibility that handling several translations in the same computation could somehow produce ”unsafety”. Presumably for that reason, the concept of translation did not catch on, and it is meanwhile customary to define alphabetic variance using renaming rather than translation ([2]). We revisit Plotkin's concept, enriched with passive bindings and deemed fit for a new name, *prenaming*, and show that its safe application on a term and safe (even backward-compatible) extension are easily achievable, thus justifying the intuitive practice.

5.1 The question of inverse

So a prenaming is more natural in practice, but a ”full” renaming is better mathematically tractable, due to its being invertible on $\boldsymbol{V}$. The next property shows how to extend a prenaming α to obtain a renaming, and a *relevant* one at that, i.e. active only on the variables from $V(\alpha)$. The claim is essentially given in [12], [2] and [1] with emphasis on the core[6] of such an extension. Originally the claim appears in [6], with emphasis on the extent of coincidence[7], which is our concern as well. We rephrase the claim around the notion of prenaming, and provide a constructive proof based on Lemma 3.5.

Theorem 5.3 (embedding)**.** *If* α *is a prenaming, there is a renaming* $\overline{\alpha}$ *which coincides with* α *on* $\boldsymbol{V} \setminus (R(\alpha) \setminus C(\alpha))$ *such that* $Vars(\overline{\alpha}) \subseteq V(\alpha)$*. Additionally, if* $\alpha(x) \neq x$ *on* $C(\alpha)$*, then* $\overline{\alpha}(x) \neq x$ *on* $V(\alpha)$*.*

$$\overline{\alpha}(x) := \begin{cases} \alpha(x), & \text{if } x \in C(\alpha) \\ z, & \text{if } x \in R(\alpha) \setminus C(\alpha) \text{ and } \alpha^m(z) = x \text{ for maximal } m \leq n \\ x, & \text{outside of } C(\alpha) \cup R(\alpha) \end{cases}$$

Figure 1: Closure, the natural relevant embedding

[6] [2, p. 23]: ”Every finite 1-1 mapping f from A onto B can be extended to a permutation g of $A \cup B$. Moreover, if f has no fixpoints, then it can be extended to a g with no fixpoints.”

[7] [6, p. 35]: ”Let W be a co-finite set of variables (...) and let σ be a W-renaming. Then there is a permutation π which coincides with σ on the set W.”

Proof. If α is a prenaming, then $C(\alpha)$ and $R(\alpha)$ are sets of n distinct variables each. The wanted renaming is constructed in Figure 1, with the intention to close the possibly open chain $x, \alpha(x), \alpha^2(x), \ldots$ So let us see whether for every x there is a j such that $\overline{\alpha}^j(x) = x$. If $x \in C(\alpha)$, we start as in the proof of Lemma 3.5, and consider the sequence $x, \alpha(x), \alpha^2(x), \ldots$ Since $C(\alpha)$ is finite, either we get two equals (and proceed as there), or we get $\alpha^k(x) \notin C(\alpha)$ and are stuck. For $y := \alpha^k(x)$ we know $\overline{\alpha}(y) = z$ such that $\alpha^m(z) = y$ with maximal m, so $m \geq k$. Therefore, $\alpha^m(\overline{\alpha}(y)) = y = \alpha^k(x)$. Due to injectivity of α on $C(\alpha)$, we get $\alpha^{m-k}(\overline{\alpha}(\alpha^k(x))) = x$, and hence $\overline{\alpha}^{m+1}(x) = x$.

The cases $x \in R(\alpha) \setminus C(\alpha)$ or $x \notin C(\alpha) \cup R(\alpha)$ are easy. By Lemma 3.5, $\overline{\alpha}$ is injective. By Legacy 3.4, $\overline{\alpha}$ is a renaming. The discussion of the case $\alpha(x) \neq x$ on $C(\alpha)$ is straightforward. ♢

Definition 5.4 (closure of a prenaming)**.** The renaming $\overline{\alpha}$ constructed in Figure 1 shall be called the *closure* of α.

Remark 5.5 (relevant embedding is not unique). Let $\alpha = \binom{z\ u\ y\ w_1}{y\ z\ x\ w_2}$, and let us embed it in a relevant renaming. The Figure 1 gives $\overline{\alpha} = \binom{z\ u\ y\ w_1\ x\ w_2}{y\ z\ x\ w_2\ u\ w_1}$. But $\rho = \binom{z\ u\ y\ w_1\ \ x\ \ w_2}{y\ z\ x\ w_2\ w_1\ \ u}$ is also a relevant renaming which is embedding α. In the usual notation for cycle decomposition, $\rho = \{(x, w_1, w_2, u, z, y)\}$ and $\overline{\alpha} = \{(x, u, z, y), (w_1, w_2)\}$.

If we reverse the prenaming, the closure algorithm shall be closing the same open chains but in the opposite direction, hence

Lemma 5.6 (reverse prenaming)**.** *Let* $\alpha := \binom{x_1\ \ldots\ x_n}{y_1\ \ldots\ y_n}$ *and* $\alpha_{inv} := \binom{y_1\ \ldots\ y_n}{x_1\ \ldots\ x_n}$. *Then* $\overline{\alpha_{inv}} = \overline{\alpha}^{-1}$.

Remark 5.7 (closure is not compositional). Take $\alpha := \binom{z\ u\ y}{y\ z\ x}$ and $\rho := \binom{x\ y}{y\ x}$. Then $\overline{\alpha} = \binom{z\ u\ y\ x}{y\ z\ x\ u}$, $\rho \cdot \overline{\alpha} = \binom{z\ u\ x}{x\ z\ u}$, $\rho \cdot \alpha = \binom{z\ u\ x}{x\ z\ y}$ and $\overline{\rho \cdot \alpha} = \binom{z\ u\ x\ y}{x\ z\ y\ u}$.

Remark 5.8 (closure is not monotone). If $\alpha \supseteq \alpha'$, then not always $\overline{\alpha} \supseteq \overline{\alpha'}$. To see this, let $\alpha = \binom{z\ u\ y}{y\ z\ x}$ and $\alpha' = \binom{z\ u}{y\ z}$. Then $\overline{\alpha'} = \binom{z\ u\ y}{y\ z\ u}$ and $\overline{\alpha} = \binom{z\ u\ y\ x}{y\ z\ x\ u}$.

5.2 Staying safe

Let us look more closely into Remark 5.8: $\alpha(y) = x$ and $\alpha(x) = x$, so y and x may not simultaneously occur in the candidate term. Otherwise, a variable shall be lost, which we call ”aliasing”, like in $\binom{y}{x}(p(x, f(y))) = p(x, f(x))$.

Definition 5.9 (aliasing)**.** Let α be a prenaming. If $x \neq y$ but $\alpha(x) = \alpha(y)$, then α is *aliasing* x and y.

So what Remark 5.8 means is: if we want to use α on a larger set than $C(\alpha)$, then the set $Pit(\alpha) := R(\alpha) \setminus C(\alpha)$ should be avoided, because aliasing may happen. But, luckily, its complement is safe:

Lemma 5.10 (larger set)**.** *A prenaming α is injective on the co-finite set $\mathbf{V} \setminus Pit(\alpha)$. The set is maximal containing $C(\alpha)$.*

Proof. Let $x, y \in \mathbf{V} \setminus Pit(\alpha)$. Is it possible that $\alpha(x) = \alpha(y)$? Possible cases: If $x, y \in C(\alpha)$, then by definition of prenaming $\alpha(x) \neq \alpha(y)$. If $x, y \notin C(\alpha)$, then $\alpha(x) = x \neq y = \alpha(y)$. It remains to consider the mixed case $x \in C(\alpha), y \notin C(\alpha)$. We have $\alpha(x) \in R(\alpha)$ and $\alpha(y) = y$. So is $\alpha(x) = y$ possible? If yes, then $y \in R(\alpha)$, but since $y \notin C(\alpha)$, that would mean $y \in Pit(\alpha)$. Contradiction.

The set cannot be made larger: if $y \in Pit(\alpha)$, then there is $x \in C(\alpha)$ with $x \neq y$ and $\alpha(x) = y = \alpha(y)$. ♢

Definition 5.11 (injectivity domain)**.** Since $\mathit{InDom}(\alpha) := \boldsymbol{V} \setminus \mathit{Pit}(\alpha)$ is the largest co-finite set containing $C(\alpha)$ on which α is injective, it shall be called the *injectivity domain* of α.

The injectivity domain of a prenaming is clearly the only safe place for it to be mapping terms from.

Definition 5.12 (safety of prenaming)**.** A prenaming α is *safe*[8] for a term t if $\mathit{Vars}(t) \subseteq \mathit{InDom}(\alpha)$.

Clearly, $\mathit{InDom}(\alpha) = C(\alpha) \cup (\boldsymbol{V} \setminus R(\alpha))$, so α is safe for its relaxed core. Hence,

Corollary 5.13 (complete and safe)**.** *If a prenaming is complete for a term, it is safe for that term.*

For a prenaming α with the quality $R(\alpha) = C(\alpha)$, i.e. a renaming, it is no surprise that $\mathit{InDom}(\alpha) = \boldsymbol{V}$ and hence safety is guaranteed for any term.

A prenaming behaves like a renaming on its injectivity domain, since it coincides with its closure there. This follows immediately from Theorem 5.3:

Lemma 5.14 (injectivity domain)**.** *Let $x \in \mathit{InDom}(\alpha)$. Then $\alpha(x) = \overline{\alpha}(x)$.*

Corollary 5.15 (prenaming stability)**.** *A generalization of Lemma 3.2 holds: Let s,t be terms and α be a prenaming safe for s,t. Then $s = t$ iff $\alpha(s) = \alpha(t)$ and also $s \in t$ iff $\alpha(s) \in \alpha(t)$. As a consequence, $s \not\bowtie t$ iff $\alpha(s) \not\bowtie \alpha(t)$.*

Our definition of prenaming was inspired by the following more general notion from [6].

Definition 5.16 (W-renaming, [6])**.** Let $W \subseteq V$. A substitution σ is a *W-renaming* if σ is variable-pure on W, and σ is injective on W.

With this notion, Lemma 5.10 can be summarized as: $\mathit{InDom}(\alpha)$ is a co-finite set of variables, and the largest set $W \supseteq C(\alpha)$ such that α is a W-renaming.

What about safety of extension? If α is safe for t, $\alpha \uplus \beta$ does not have to be, even if $\beta(t) = t$, as the following example shows: $\alpha := \binom{v}{w}$, $\beta := \begin{pmatrix} z\,u\,y \\ y\,z\,x \end{pmatrix}$, $t := p(x)$ (here no aliasing happened, though). The next two claims address safety of extension.

Lemma 5.17 (monotonicity)**.** *Assume $\alpha \uplus \beta$ is defined. Then*

1. *$\mathit{InDom}(\alpha) \cup \mathit{InDom}(\beta) = \boldsymbol{V}$*
2. *$\mathit{InDom}(\alpha) \cap \mathit{InDom}(\beta) \subseteq \mathit{InDom}(\alpha \uplus \beta)$*

Proof. Since $(\boldsymbol{V} \setminus A) \cup (\boldsymbol{V} \setminus B) = \boldsymbol{V} \setminus (A \cap B))$, and $\mathit{Pit}(\alpha) \not\bowtie \mathit{Pit}(\beta)$, we get $\mathit{InDom}(\alpha) \cup \mathit{InDom}(\beta) = \boldsymbol{V}$. Further, $(\boldsymbol{V} \setminus A) \cap (\boldsymbol{V} \setminus B) = \boldsymbol{V} \setminus (A \cup B)$ and so $\mathit{Pit}(\alpha \uplus \beta) = (R(\alpha) \uplus R(\beta)) \setminus (C(\alpha) \uplus C(\beta)) \subseteq (R(\alpha) \setminus C(\alpha)) \cup (R(\beta) \setminus C(\beta)) = \mathit{Pit}(\alpha) \cup \mathit{Pit}(\beta)$. $\diamondsuit$

In Remark 5.8, $\mathit{Pit}(\alpha') = \{y\}$, $\mathit{Pit}(\binom{y}{x}) = \{x\}$, and $\mathit{Pit}(\alpha) = \{x\}$, hence $\mathit{InDom}(\alpha') = \boldsymbol{V} \setminus \{y\}$, $\mathit{InDom}(\binom{y}{x}) = \boldsymbol{V} \setminus \{x\}$ and $\mathit{InDom}(\alpha) = \boldsymbol{V} \setminus \{x\}$.

By the last claim, staying within $\mathit{InDom}(\alpha)$ and $\mathit{InDom}(\beta)$ ensures staying within $\mathit{InDom}(\alpha \uplus \beta)$. By assuming a bit more about α than just safety, we may ignore the nature of extension β, and still ensure safety and even backward compatibility of $\alpha \uplus \beta$. This shall be used in Section 6.

Theorem 5.18 (safety of extension)**.** *Assume $\alpha \uplus \beta$ is defined.*

1. *If α is safe for t and β is safe for t, then $\alpha \uplus \beta$ is safe for t.*
2. *If α is complete for t, then $\alpha \uplus \beta$ is complete (hence safe) for t, and $(\alpha \uplus \beta)(t) = \alpha(t)$.*

The first part follows from Lemma 5.17 and the second from Corollary 4.5 and Corollary 5.13.

[8] Safe prenaming is more general than *renaming for a term* in [11, p. 22], since we do not require $\mathit{Core}(\alpha) \subseteq \mathit{Vars}(t)$.

5.3 Variant of term and substitution

The traditional notion of term variance, which is term renaming, shall be generalized to prenaming. As a special case, substitution variance is defined, inspired by substitution renaming from [1]. For this, substitution shall once again be regarded as a special case of term. The term is of course the relaxed core representation. This concept shall come in handy for proving properties of renamed derivations (Subsection 6.2).

5.3.1 Term variant

Definition 5.19 (term variant)**.** If α is a prenaming safe for t, then $\alpha(t)$ is a *variant* of t, written $\alpha(t) \cong t$. The particular variance and the direction of its application may be explicated by $s =_\alpha t$ iff $s = \alpha(t)$.

If $s \cong t$, then there is a unique α mapping s to t in a complete and relevant[9] manner, i.e. mapping each variable pair and nothing else, as computed by Figure 2. The algorithm makes do with only one set for equations and bindings, thanks to different types. Termination can be seen from the tuple $(lfun_=(E), card_=(E))$ decreasing in lexicographic order with each rule application, where $lfun_=(E)$ is the number of function symbols in equations in E, and $card_=(E)$ is the number of equations in E.

Start from the set $E := \{s = t\}$ and transform according to the following rules. The transformation is bound to stop. If the stop was not due to failure, then the final set E is the prenaming of s to t, $Pren(s,t)$.

elimination $E \uplus \{x = y\} \rightsquigarrow E$, if $x/y \in E$

failure: alias $E \uplus \{x = y\} \rightsquigarrow$ failure, if $(x/z \in E, z \neq y)$ or $(z/y \in E, z \neq x)$

binding $E \uplus \{x = y\} \rightsquigarrow E \cup \{x/y\}$, if $(x/\_ \notin E)$ and $(\_/y \notin E)$

failure: instance $E \uplus \{x = t\} \rightsquigarrow$ failure, if $t \notin \mathbf{V}$; $E \uplus \{t = x\} \rightsquigarrow$ failure, if $t \notin \mathbf{V}$

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

failure: clash $E \uplus \{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)\} \rightsquigarrow$ failure, if $f \neq g$ or $m \neq n$

Figure 2: Computing the prenaming of s to t

Notation 5.20 (epsoid). The prenaming constructed in Figure 2 shall be simply called *the prenaming* of s to t, and denoted $Pren(s,t)$. It is complete for s and relevant for s to t.

In case $s = t$, we obtain for $Pren(s,t)$ essentially the identity substitution. However, regarded as prenamings, $Pren(t,t)$ and ε are not the same. A prenaming α with relaxed core W mapping each variable on itself (in other words, $C(\alpha) = W$ and $[\alpha] = \varepsilon$) shall be called the *W-epsoid* and denoted ε_W. For a term t, we abbreviate $\varepsilon_t := \varepsilon_{Vars(t)}$.

Regarding composition, an epsoid behaves just like ε. Its use is for providing completeness, and hence extensibility, by means of placeholder bindings x/x.

5.3.2 Special case: substitution variant

Even substitutions themselves can be renamed. To rename a substitution, one regards it as a syntactical object, a set of bindings, and renames those bindings. If ρ is a renaming and σ is a substitution, [1]

[9]"Relevant" in the binary sense (page 28). In case of prenaming, we naturally use C as *Dom* and R as *Range*.

defines substitution renaming by $\rho(\sigma) := \{\rho(x)/\rho(\sigma(x)) \mid x \in \mathit{Core}(\sigma)\}$. It is easy to see that $\rho(\sigma)$ is a substitution in core representation. For this only two properties of ρ were needed: variable-purity on $\mathit{Vars}(\sigma)$ and injectivity on $\mathit{Vars}(\sigma)$. These requirements are clearly fulfilled by prenamings safe on σ as well. Hence,

Definition 5.21 (substitution variant)**.** Let σ be a substitution and let α be a prenaming safe for σ, i.e. $\mathit{Vars}(\sigma) \subseteq \mathit{InDom}(\alpha)$. Then a *variant of* σ *by* α is

$$\alpha(\sigma) := \{\alpha(x)/\alpha(\sigma(x)) \mid x \in \mathit{Core}(\sigma)\} \tag{1}$$

We may write $\theta =_\alpha \sigma$ if $\theta = \alpha(\sigma)$, as with any other terms. As can be expected, the concept of variance by prenaming is well-defined, owing to safety. Otherwise, the result of prenaming would not even have to be a substitution again, as in the case of $\alpha = \binom{y}{x}$ and $\sigma = \binom{x\ y}{a\ b}$.

Lemma 5.22 (well-defined)**.** *Substitution variant is well-defined, i.e.* (1) *is a core representation of a substitution, and* α *does not introduce aliasing.*

Proof. Let $\mathit{Core}(\sigma) = \{x_1,\ldots,x_n\}$. Due to injectivity of α on $\mathit{Vars}(\sigma)$, if $\alpha(x_i) = \alpha(x_j)$, then $x_i = x_j$, so $i = j$. To finish the proof that (1) a core representation, observe $x \in \mathit{Core}(\sigma)$ iff $x \neq \sigma(x)$ iff $\alpha(x) \neq \alpha(\sigma(x))$, due to injectivity again. Re aliasing, by Corollary 5.15, if $\alpha(\sigma(x_i)) \bowtie \alpha(\sigma(x_j))$, then $\sigma(x_i) \bowtie \sigma(x_j)$, meaning that α does not introduce aliasing. $\diamondsuit$

From Definition 5.21 and Lemma 5.14 follows

Lemma 5.23. *Let* σ *be a substitution,* α, β *be prenamings and* $\alpha(\sigma)$ *and* $(\alpha \cdot \beta)(\sigma)$ *be defined. Then*

1. $(\alpha \cdot \beta)(\sigma) = \alpha(\beta(\sigma))$
2. $\alpha(\sigma) = \overline{\alpha}(\sigma)$

For the case of ”full” renaming, there is a way to dissolve the new expression:[10]

Legacy 5.24 ([1])**.** *For any renaming* ρ *and substitution* σ

$$\rho(\sigma) = \rho \cdot \sigma \cdot \rho^{-1}$$

Would such a claim hold for the weakened case, prenamings?

Theorem 5.25 (substitution variant)**.** *Let* σ *be a substitution and* α *be a prenaming safe for* σ*. Then*

1. $\alpha(\sigma) \cdot \alpha = \alpha \cdot \sigma$
2. $\alpha(\sigma) = \overline{\alpha} \cdot \sigma \cdot \overline{\alpha}^{-1}$

Proof. First part: According to Definition 5.21, for every $x \in \boldsymbol{V}$ holds $(\alpha(\sigma) \cdot \alpha)(x) = \alpha(\sigma(x))$. Since any substitution is structure-preserving, the claim holds for any term t as well. Second part: From the first part we know $\overline{\alpha}(\sigma) \cdot \overline{\alpha} = \overline{\alpha} \cdot \sigma$, hence $\overline{\alpha}(\sigma) = \overline{\alpha} \cdot \sigma \cdot \overline{\alpha}^{-1}$. By Lemma 5.14, $\alpha(\sigma) = \overline{\alpha}(\sigma)$. $\diamondsuit$

It is known that idempotence and equivalence of substitutions are not compatible with composition [6]. Luckily, the concept of variance, with constant prenaming, does not share this handicap:

Theorem 5.26 (compositionality)**.** *Let* σ, θ *be substitutions and* α *be their safe prenaming. Then*

$$\alpha(\sigma \cdot \theta) = \alpha(\sigma) \cdot \alpha(\theta)$$

Proof. Since $\mathit{Vars}(\sigma \cdot \theta) \subseteq \mathit{Vars}(\sigma) \cup \mathit{Vars}(\theta)$, clearly $\mathit{Vars}(\sigma \cdot \theta) \subseteq \mathit{InDom}(\alpha)$. By Theorem 5.25, $\alpha(\sigma) \cdot \alpha(\theta) = \overline{\alpha} \cdot \sigma \cdot \overline{\alpha}^{-1} \cdot \overline{\alpha} \cdot \theta \cdot \overline{\alpha}^{-1} = \overline{\alpha} \cdot \sigma \cdot \theta \cdot \overline{\alpha}^{-1} = \alpha(\sigma \cdot \theta)$. $\diamondsuit$

[10] an immediate consequence being $\rho(\sigma) \neq \rho \cdot \sigma$

6 Application

Implementing logic programming means that the freedom of Horn clause logic (HCL) must be restrained:

- most general unifier (*mgu*) is provided by a fixed algorithm,
- standardization-apart is provided by a fixed algorithm.

Every implementation of HCL is parametrized by the two algorithms. Here we shall consider only the unification algorithm, so by HCL($\boldsymbol{U}$) an implementation of HCL using unification algorithm $\boldsymbol{U}$ is denoted. From the literature (*variant lemma*) we know that such a restriction is not compromising soundness and completeness of SLD-resolution. Yet, there may be lots of ”lowlier” claims which more or less implicitly rely on freedom of mgus and standardization-apart. For example, with both choices fixed we may not any more just rename an SLD-derivation wholesale (the resolvents, the mgus, the input clauses), as was possible in Horn clause logic, based on Lemma 3.2. This is because the two algorithms do not have to be renaming-compatible. In fact, the second one cannot be, which makes claims like Lemma 6.3 necessary.

Let us cast a look at the first restriction. For any two unifiable terms s,t holds that the set of their mgus, written as $Mgus(s,t)$, is infinite. On the other hand, in practice any unification algorithm $\boldsymbol{U}$ produces, for the given two unifiable terms, just one deterministic value as their mgu. We shall denote this particular mgu of s and t as $\boldsymbol{U}(s,t)$, the *algorithmic mgu* of s and t obtained by $\boldsymbol{U}$.

The abundancy of mgus is not only good, it also stands in the way of proofs. The simplest unification problem $p(x) = p(y)$ has among others two equally attractive candidate mgus, $\binom{x}{y}$ and $\binom{y}{x}$. Assume our unification algorithm decided upon $\binom{x}{y}$. Assume further that we rename the protagonists and obtain $p(x) = p(z)$. What mgu shall be chosen this time? To ensure some dependability in this issue, we shall require of any unification algorithm the following simple requirement, postulated as an axiom:

Axiom 6.1 (renaming compatibility)**.** Let $\boldsymbol{U}$ be a unification algorithm. For any renaming ρ and any equation E, it has to hold $\boldsymbol{U}(\rho(E)) = \rho(\boldsymbol{U}(E))$.

Since classical unification algorithms like Robinson’s and Martelli-Montanari’s do not depend upon the actual names of variables (as observed in [1]), this requirement is in practice always satisfied.

Remark 6.2 (renaming compatibility of *Mgus*)*.* For every ρ and E holds $Mgus(\rho(E)) = \rho(Mgus(E))$. This is due to Theorem 5.25 and Lemma 3.2. Assume $\sigma \in Mgus(s,t)$, then $\rho(\sigma)(\rho(s)) = \rho(\sigma(s)) = \rho(\sigma(t)) = \rho(\sigma)(\rho(t))$. Further, if θ is a unifier of $\rho(s), \rho(t)$, then $\theta \cdot \rho$ is a unifier of s,t, hence there is a renaming δ with $\theta \cdot \rho = \delta \cdot \sigma$, giving $\theta = \delta \cdot \sigma \cdot \rho^{-1} = \delta \cdot \rho^{-1} \cdot \rho \cdot \sigma \cdot \rho^{-1} = (\delta \cdot \rho^{-1}) \cdot \rho(\sigma)$, meaning $\rho(\sigma) \in Mgus(\rho(E))$. For the other direction, observe $\theta = \rho \cdot \rho^{-1} \cdot \delta \cdot \sigma \cdot \rho^{-1} = \rho(\rho^{-1} \cdot \delta \cdot \sigma)$.

6.1 Handling local variables in HCL($\boldsymbol{U}$)

With $\boldsymbol{U}$ complying to Axiom 6.1 and yielding relevant mgus, that is to say with practically any $\boldsymbol{U}$,[11] a propagation result for SLD-derivations can be proved, which leads to a constructive and incremental version of the variant lemma.

Regarding SLD-derivations, for the most part we shall assume traditional concepts as given in [11] and [2], but with some changes and additions listed below. An input clause $\mathscr{K}_i$ obtained from a program clause $\mathscr{K}$ by replacing the variables in order of appearance with $t_1,\ldots,t_n$ may be denoted as $\mathscr{K}_i = \mathscr{K}[t_1,\ldots,t_n]$. Assume now an SLD-derivation $\boldsymbol{D}$ for G of the form $G \hookrightarrow_{\mathscr{K}_1:\sigma_1} G_1 \hookrightarrow_{\mathscr{K}_2:\sigma_2} \cdots \hookrightarrow_{\mathscr{K}_n:\sigma_n} G_n$.

[11]Classical unification algorithms not only satisfy Axiom 6.1 but also yield idempotent mgus. Idempotent mgus are always relevant ([2]).

- $\mathscr{K}_i$ is here the *actually used* variant of a program clause (i.e., the current *input clause*) and not the program clause itself.
- The substitution $\sigma_n \cdot \ldots \cdot \sigma_1$ shall be called the *partial answer* for G at step n of the derivation. A final partial answer, whenever $G_n = \Box$, shall be called a *complete answer* for G.

Relation to earlier concepts of answer: In a derivation, the *resultant* of level n is defined in [12] as $\sigma_n \cdot \ldots \cdot \sigma_1(G) \leftarrow G_n$. So partial answer is at heart of resultant. An *answer substitution* for G is defined in [3] exactly as complete answer; later ([11], [2]) it is made relevant by restricting to variables of G, hence $(\sigma_n \cdot \ldots \cdot \sigma_1)\lceil_G$ whenever $G_n = \Box$, and called *computed answer substitution (c.a.s.)*.

Showing the actually used variants of program clauses (instead of program clauses themselves) enables a simple definition of derivation variables: the annotations $\mathscr{K}_i{:}\sigma_i$ are regarded as part of the derivation, so $Vars(\boldsymbol{D}) := (Vars(G) \cup \ldots \cup Vars(G_n)) \cup (Vars(\sigma_1) \cup \ldots \cup Vars(\sigma_n)) \cup (Vars(\mathscr{K}_1) \cup \ldots \cup Vars(\mathscr{K}_n))$.

Now to the propagation result. Assume the program ”$son(S) \leftarrow male(S), child(S,P).$”, and let us enquire about *son* in two derivations. If we know that one query, say $son(X)$, is a variant of the other, $son(A)$, does the same connection hold between the resolvents as well?

As can be seen from Table 1 and Figure 3, in a resolution some new variables may crop up, originating from standardization-apart. Let us call them *local variable*s, as opposed to *query variable*s. Likely causes are a pattern in a clause head ($nat/1$ in Table 1) or surplus variables in a clause body ($son/1$ in Figure 3). But even without those, local variables can appear ($p/1$ in Table 1), except with a restriction to *normal SLD-derivation* ([4]), preventing ”needless renaming of variables”. Were it not for local variables, the resolvents in both derivations would clearly be variants, with the same prenaming as for the original queries. Yet, even though the variables new in one derivation do not have to be new in the other, the prenaming can (under reasonable conditions) be extended to accomodate them. The claim is proved in a constructive manner.

query	input clause	resolvent
$nat(X)$	$nat(s(A)) \leftarrow nat(A).$	$nat(A)$
$nat(A)$	$nat(s(B)) \leftarrow nat(B).$	$nat(B)$
$p(X)$	$p(A) \leftarrow q(A).$	$q(X)$ or $q(A)$
$p(A)$	$p(B) \leftarrow q(B).$	$q(A)$ or $q(B)$

Table 1: Resolution may produce local variables

Lemma 6.3 (propagation of variance)**.** *Assume a unification algorithm* $\boldsymbol{U}$ *satisfying Axiom 6.1. Assume an SLD-derivation* $\boldsymbol{D}$ *ending with* G *and an SLD-derivation* $\boldsymbol{D}'$ *ending with* G' *such that* $\alpha(G) = G'$ *for some prenaming* α *which is complete for* G *and relevant for* $\boldsymbol{D}$ *to* $\boldsymbol{D}'$.

Further assume that $G \hookrightarrow_{\mathscr{K}:\sigma} H$ *and* $G' \hookrightarrow_{\mathscr{K}':\sigma'} H'$ *such that in* G *and* G' *atoms in the same positions were selected and* $\mathscr{K}, \mathscr{K}'$ *are variants. Lastly assume that* σ *is a relevant mgu. Then for* $\lambda := Pren(\mathscr{K}, \mathscr{K}')$ *holds*

1. $\alpha \uplus \lambda$ *is complete for* H *and* σ
2. $\alpha \uplus \lambda$ *is relevant for* $\boldsymbol{D} \hookrightarrow_{\mathscr{K}:\sigma} H$ *to* $\boldsymbol{D}' \hookrightarrow_{\mathscr{K}':\sigma'} H'$
3. $H' = (\alpha \uplus \lambda)(H)$ *and* $\sigma' = (\alpha \uplus \lambda)(\sigma)$

The claim can be summarized in Figure 3, together with the rôle of relevance requirement. From Lemma 6.3 and Table 1 follows that resolution is not prenaming-stable (or even renaming-stable); extending the prenaming with local variables may be necessary.

$$\begin{tikzcd}
G \arrow[r, "\hookrightarrow\!\!\rhd"] \arrow[d, "\alpha"', "\alpha\uplus\lambda"] & H \arrow[d, "\alpha\uplus\lambda"] \\
G' \arrow[r, "\hookrightarrow\!\!\rhd"'] & H'
\end{tikzcd}
\qquad
\begin{tikzcd}
son(X) \arrow[r, "\hookrightarrow\!\!\rhd"] \arrow[d, "\alpha=\left(\begin{smallmatrix} X & B \\ A & X \end{smallmatrix}\right)"] & male(X), child(X,B) \arrow[d, "\alpha\uplus?"] \\
son(A) \arrow[r, "\hookrightarrow\!\!\rhd"'] & male(A), child(A,C)
\end{tikzcd}$$

Figure 3: Propagation of variance ...is not always possible

Proof. First let us establish that $\alpha \uplus \lambda$ is defined. Due to *relevance of* α *for* $\boldsymbol{D}, \boldsymbol{D}'$,

$$C(\alpha) \subseteq Vars(\boldsymbol{D}) \text{ and } R(\alpha) \subseteq Vars(\boldsymbol{D}') \tag{2}$$

Due to *standardization-apart*, $\mathscr{K} \not\bowtie \boldsymbol{D}$ and $\mathscr{K}' \not\bowtie \boldsymbol{D}'$, hence

$$C(\lambda) \not\bowtie \boldsymbol{D} \text{ and } R(\lambda) \not\bowtie \boldsymbol{D}' \tag{3}$$

Thus $C(\alpha) \not\bowtie C(\lambda)$ and $R(\alpha) \not\bowtie R(\lambda)$, so $\alpha \uplus \lambda$ is defined. Also, (3) proves that λ is passive on old variables, i.e. $\lambda(\boldsymbol{D}) = \boldsymbol{D}$. Let $G := (M,A,N)$ and $H := \sigma(M,B,N)$, where M,B,N may be conjunctions. Then $G' = (M',A',N') = (\alpha(M), \alpha(A), \alpha(N))$. Let $\mathscr{K} : A_1 \leftarrow B_1$ and $\mathscr{K}' : A_2 \leftarrow B_2$. Then $\sigma = \boldsymbol{U}(A, A_1)$, $B = \sigma(B_1)$ and $\sigma' = \boldsymbol{U}(A', A_2)$, $B' = \sigma'(B_2)$. Also,

$$\alpha \text{ is complete for } M,A,N\text{: } Vars((M,A,N)) \subseteq C(\alpha), \text{ by } \textit{completeness} \text{ of } \alpha \text{ for } G \tag{4}$$
$$\lambda \text{ is complete for } A_1,B_1\text{: } Vars(\mathscr{K}) = Vars((A_1,B_1)) = C(\lambda), \text{ by definition of } \lambda \tag{5}$$

Due to *relevance* of σ, we have $Vars(\sigma) \subseteq Vars(A) \cup Vars(A_1)$, which together with (4) and (5) gives

$$\alpha \uplus \lambda \text{ is complete for } \sigma\text{: } Vars(\sigma) \subseteq C(\alpha \uplus \lambda) \tag{6}$$

Having thus fielded all the assumptions, we obtain

$$\alpha \uplus \lambda \text{ is safe for } M,A,N,A_1,B_1,\sigma\text{: by (4), (5), (6) and Theorem 5.18(2)} \tag{7}$$

Proof of 1.: Completeness of $\alpha \uplus \lambda$ for σ is proved above. Completeness of $\alpha \uplus \lambda$ for H follows from $Vars(H) \subseteq Vars(G) \cup Vars(\mathscr{K}) \subseteq C(\alpha) \cup C(\lambda) = C(\alpha \uplus \lambda)$, by (4) and (5).

Proof of 2.: By definition, $C(\lambda) = Vars(\mathscr{K})$ and $R(\lambda) = Vars(\mathscr{K}')$. Hence, and due to relevance of α, $C(\alpha \uplus \lambda) = C(\alpha) \uplus C(\lambda) \subseteq Vars(\boldsymbol{D}) \cup Vars(\mathscr{K}) \subseteq Vars(\boldsymbol{D} \hookrightarrow\!\!\rhd_{\mathscr{K}:\sigma} H)$. Similarly, $R(\alpha \uplus \lambda) \subseteq Vars(\boldsymbol{D}' \hookrightarrow\!\!\rhd_{\mathscr{K}':\sigma'} H')$, therefore $\alpha \uplus \lambda$ is relevant for $\boldsymbol{D} \hookrightarrow\!\!\rhd_{\mathscr{K}:\sigma} H$ to $\boldsymbol{D}' \hookrightarrow\!\!\rhd_{\mathscr{K}':\sigma'} H'$.

Proof of 3.:

$$\begin{aligned}
\sigma' &= \boldsymbol{U}(A', A_2) = \boldsymbol{U}(\alpha(A), \lambda(A_1)) = \boldsymbol{U}((\alpha \uplus \lambda)(A), (\alpha \uplus \lambda)(A_1)), \text{ by (4), (5) and Corollary 4.5} \\
&= \boldsymbol{U}((\overline{\alpha \uplus \lambda})(A), (\overline{\alpha \uplus \lambda})(A_1)) = (\overline{\alpha \uplus \lambda})(\sigma) = (\alpha \uplus \lambda)(\sigma), \text{ by (7), Lemma 5.14 and Axiom 6.1}
\end{aligned}$$

$$\begin{aligned}
B' &= \sigma'(B_2) = (\alpha \uplus \lambda)(\sigma)(\lambda(B_1)) = (\alpha \uplus \lambda)(\sigma)((\alpha \uplus \lambda)(B_1)), \text{ by (5) and Corollary 4.5} \\
&= (\alpha \uplus \lambda)(\sigma(B_1)) = (\alpha \uplus \lambda)(B), \text{ by Theorem 5.25}
\end{aligned}$$

$$\begin{aligned}
H' &= \sigma'(\alpha(M), B', \alpha(N)) = (\alpha \uplus \lambda)(\sigma)(\alpha(M), (\alpha \uplus \lambda)(B), \alpha(N)), \text{ by preceding line} \\
&= (\alpha \uplus \lambda)(\sigma)((\alpha \uplus \lambda)(M), (\alpha \uplus \lambda)(B), (\alpha \uplus \lambda)(N)), \text{ by (4) and Corollary 4.5} \\
&= (\alpha \uplus \lambda)(\sigma(M,B,N)) = (\alpha \uplus \lambda)(H), \text{ by Theorem 5.25}
\end{aligned}$$

$\diamondsuit$

6.2 Variant lemma for HCL($\boldsymbol{U}$)

Definition 6.4 (similarity)**.** SLD-derivations of the same length

$$\begin{array}{c} G \hookrightarrow_{\mathscr{K}_1:\sigma_1} G_1 \hookrightarrow_{\mathscr{K}_2:\sigma_2} \dots \hookrightarrow_{\mathscr{K}_n:\sigma_n} G_n \\ G' \hookrightarrow_{\mathscr{K}_1':\sigma_1'} G_1' \hookrightarrow_{\mathscr{K}_2':\sigma_2'} \dots \hookrightarrow_{\mathscr{K}_n':\sigma_n'} G_n' \end{array} \tag{8}$$

are *similar* if G and G' are variants and additionally at each step i holds: atoms in the same position are selected, and the input clauses $\mathscr{K}_i$ and $\mathscr{K}_i{}'$ are variants.

That the name "similarity" is justified, follows from the claim known as *variant lemma* ([11], [12], [2]), here in the formulation from [5]: *Finite derivations which are similar and start from variant queries have variant resultants*. For logic programming systems obeying Axiom 6.1 and relevance of mgu, a more precise claim can be proved. The added assumptions (the axiom and relevance) are practically void (see footnote on page 36), yet the added conclusion has substance: first, renaming a query now costs a degree of freedom – if we treat the two variants of the program clause at each step as independent, then the two mgus are not independent. Second, the precise variance is now known.

Theorem 6.5 (variant claim for HCL($\boldsymbol{U}$))**.** *Assume a unification algorithm* $\boldsymbol{U}$ *satisfying Axiom 6.1 and yielding relevant mgus. Then:*

- *finite SLD-derivations which are similar and start from variant queries have variant partial answers*
- *the variance depends only on the starting queries and input clauses.*

In particular, assume our similar derivations to be as in (8)*. Then for every* $i = 1,\dots,n$ *holds* $G_i' = \beta_i(G_i)$, $\sigma_i' = \beta_i(\sigma_i)$ *and* $\sigma_i' \cdot \ldots \cdot \sigma_1' = \beta_i(\sigma_i \cdot \ldots \cdot \sigma_1)$, *where* $\beta_i := \alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i$, $\alpha := Pren(G, G')$ *and* $\lambda_i := Pren(\mathscr{K}_i, \mathscr{K}_i{}')$.

Proof. By assumption, G and G' are variants, so

$$\alpha := Pren(G, G') \tag{9}$$

exists. Clearly, α is complete for G. By construction, α is also relevant for $\boldsymbol{D}_0 := G$ to $\boldsymbol{D}_0' := G'$. We may iterate Lemma 6.3, obtaining for every $i = 1,\dots,n$

$$\sigma_i' = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(\sigma_i) \tag{10}$$

$$G_i' = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(G_i) \tag{11}$$

where $\lambda_i := Pren(\mathscr{K}_i, \mathscr{K}_i{}')$. Due to completeness of α for G and Corollary 4.5,

$$\alpha(G) = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(G) \tag{12}$$

So $\sigma_i' \cdot \sigma_{i-1}' \cdot \ldots \cdot \sigma_1' = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(\sigma_i) \cdot (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_{i-1})(\sigma_{i-1}) \cdot \ldots \cdot (\alpha \uplus \lambda_1)(\sigma_1)$. We would like to extract $\sigma_i \cdot \sigma_{i-1} \cdot \ldots \cdot \sigma_1$ on the right, to have a connection between partial answers.

Assume $k < i$. Since $Vars(\sigma_k) \subseteq Vars(G) \cup Vars(\mathscr{K}_1) \cup \ldots \cup Vars(\mathscr{K}_k) \subseteq C(\alpha) \cup C(\lambda_1) \cup \ldots \cup C(\lambda_k) = C(\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_k)$, by Theorem 5.18(2) $(\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_k \uplus \ldots \uplus \lambda_i)(\sigma_k) = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_k)(\sigma_k)$. Hence,

$$\begin{array}{r} (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_{i-1})(\sigma_{i-1}) = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(\sigma_{i-1}) \\ \ldots \\ (\alpha \uplus \lambda_1)(\sigma_1) = (\alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i)(\sigma_1) \end{array} \tag{13}$$

Let us abbreviate $\beta_i := \alpha \uplus \lambda_1 \uplus \ldots \uplus \lambda_i$. Then from (10) and (13) by Theorem 5.26

$$\sigma'_i \cdot \sigma'_{i-1} \cdot \ldots \cdot \sigma'_1 = \beta_i(\sigma_i) \cdot \beta_i(\sigma_{i-1}) \cdot \ldots \cdot \beta_i(\sigma_1) = \beta_i(\sigma_i \cdot \sigma_{i-1} \cdot \ldots \cdot \sigma_1) \tag{14}$$

which is the promised connection between partial answers.

Clearly, variance of partial answers means variance of complete answers, c.a.s.es and resultants as well. For the cases when $G_n = \square$, we obtain, by (14), the expected relationship between the respective complete answers: $\sigma'_n \cdot \ldots \cdot \sigma'_1 = \beta_n(\sigma_n \cdot \ldots \cdot \sigma_1)$. C.a.s. differs from complete answer by the added restriction on the query variables. Due to renaming-compatibility of restriction, (14) and $\beta_n(G) = \alpha(G) = G'$, we obtain $\sigma'_n \cdot \ldots \cdot \sigma'_1 \restriction_{G'} = \beta_n(\sigma_n \cdot \ldots \cdot \sigma_1 \restriction_G)$, i.e. the same relationship. Finally, knowing that the resultant of step i is $R_i := \sigma_i \cdot \ldots \cdot \sigma_1(G) \leftarrow G_i$, we obtain

$$\begin{aligned} R'_i &= \sigma'_i \cdot \ldots \cdot \sigma'_1(G') \leftarrow G'_i = \beta_i(\sigma_i \cdot \ldots \cdot \sigma_1)(\alpha(G)) \leftarrow \beta_i(G_i), \text{ by (14), (9) and (11)} \\ &= \beta_i(\sigma_i \cdot \ldots \cdot \sigma_1)(\beta_i(G)) \leftarrow \beta_i(G_i) = \beta_i((\sigma_i \cdot \ldots \cdot \sigma_1)(G)) \leftarrow \beta_i(G_i), \text{ by (12) and Theorem 5.25} \\ &= \beta_i(R_i) \end{aligned}$$

So partial answers of step i (and in consequence c.a.s.es and resultants) are variant via β_i. ♢

Example 6.6 (similarity)**.** Let the logic program be

$$\begin{aligned} &son(X) \leftarrow male(X), child(X,A). && \% \overline{\mathscr{K}_1} \\ &male(c).\ male(d).\ child(d,a). && \% \overline{\mathscr{K}_2},\ \overline{\mathscr{K}_3},\ \overline{\mathscr{K}_4} \end{aligned}$$

An interpreter for LD-resolution may produce the following two derivations:

$$\begin{aligned} &son(A) \hookrightarrow_{\mathscr{K}_1:\sigma_1} male(A), child(A,C) \hookrightarrow_{\mathscr{K}_2:\sigma_2} child(d,C) \\ &son(B) \hookrightarrow_{\mathscr{K}'_1:\sigma'_1} male(B), child(B,D) \hookrightarrow_{\mathscr{K}'_2:\sigma'_2} child(d,D) \end{aligned}$$

They are obviously similar, with $\mathscr{K}_1 = \overline{\mathscr{K}_1}[\boldsymbol{X},C]$, $\mathscr{K}'_1 = \overline{\mathscr{K}_1}[\boldsymbol{Y},D]$, $\mathscr{K}_2 = \mathscr{K}'_2 = \overline{\mathscr{K}_3}$. The variables $\boldsymbol{X}, \boldsymbol{Y}$ stand for actually used variables, which are not discernible from the form of derivations. From the queries, input clauses and resolvents we can further deduce which relevant mgus were used: $\sigma_1 = \binom{\boldsymbol{X}}{A}$, $\sigma'_1 = \binom{\boldsymbol{Y}}{B}$, $\sigma_2 = \binom{A}{d}$ and $\sigma'_2 = \binom{B}{d}$. The mappings are $\alpha = \binom{A}{B}$, $\lambda_1 = \left(\begin{smallmatrix} \boldsymbol{X} & C \\ \boldsymbol{Y} & D \end{smallmatrix}\right)$ and $\lambda_2 = \varepsilon$. Clearly, they fulfill $(\alpha \uplus \lambda_1)(male(A), child(A,C)) = male(B), child(B,D)$ and $(\alpha \uplus \lambda_1)(\binom{B}{A}) = \binom{C}{B}$, as well as $(\alpha \uplus \lambda_1 \uplus \lambda_2)(child(d,C)) = child(d,D)$, and so forth.

7 Outlook

Concepts relating to variable renaming have been reviewed and built upon, with the aim of providing for practical needs of program analysis and formal semantics in logic programming. By *relaxing the core representation* and forgoing permutation requirement for renaming, the concept of *prenaming* is obtained. It is a mathematical underpinning of the intuitive practice of renaming terms by just considering the necessary bindings, where now x/x may be necessary. In other words, a prenaming is a variable-pure substitution with mutually distinct variables in range, possibly including some passive bindings.

Prenamings enable incremental claims like variance propagation in implemented Horn clause logic (Lemma 6.3, Theorem 6.5). There, prenamings made it possible to keep track of new (*local*) variables.

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] K. R. Apt & M. H. van Emden (1982): *Contributions to the theory of logic programming*. *J. of ACM* 29(3), pp. 841–862, doi:10.1145/322326.322339.

[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] K. Doets (1993): *Levationis laus*. *J. Logic and Computation* 3(5), pp. 487–516, doi:10.1093/logcom/3.5.487.

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

[7] J. Gallier (2015): *Logic for computer science: Foundations of automatic theorem proving*, 2. edition. Dover.

[8] D. Jacobs & A. Langen (1992): *Static analysis of logic programs for independent AND parallelism*. *J. of Logic Programming* 13(2-3), pp. 291 – 314, doi:10.1016/0743-1066(92)90034-Z.

[9] M. Kulaš (2005): *Toward the concept of backtracking computation*. In L. Aceto, W. J. Fokkink & I. Ulidowski, editors: *Proc. of the Workshop on Structural Operational Semantics (SOS'04), London*, *ENTCS* 128, Elsevier, pp. 39–59, doi:10.1016/j.entcs.2004.10.026.

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

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

[12] J. W. Lloyd & J. C. Shepherdson (1991): *Partial evaluation in logic programming*. *J. of Logic Programming* 11(3-4), pp. 217–242, doi:10.1016/0743-1066(91)90027-M.

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

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

[15] C. Pusch (1996): *Verification of compiler correctness for the WAM*. In: *Proc. TPHOLs*, *LNCS* 1125, Springer Berlin Heidelberg, pp. 347–361, doi:10.1007/BFb0105415.

[16] J. C. Shepherdson (1994): *The role of standardising apart in logic programming*. *Theor. Comp. Sci.* 129(1), pp. 143–166, doi:10.1016/0304-3975(94)90084-1.

[17] C. Urban, A. Pitts & M. Gabbay (2004): *Nominal unification*. *Theoretical Computer Science* 323(1-3), pp. 473–497, doi:10.1016/j.tcs.2004.06.016. Proof on http://www.inf.kcl.ac.uk/staff/urbanc/Unification/.