INFORMATIK
BERICHTE

376 – 11/2017

A term matching algorithm and substitution generality

Marija Kulaš

FernUniversität in Hagen

Fakultät für Mathematik und Informatik
D-58084 Hagen

A term matching algorithm and substitution generality

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

***Abstract*—We revisit a simple non-deterministic algorithm for *term matching* given in (Klop and de Vrijer et al. 2003) and employ it for deciding *substitution generality* (and thus equivalence), using a *witness term* technique. The technique alleviates the need for ad-hoc proofs involving generality of substitutions.**

***Index Terms*—substitution, term matching, generality**

I. Introduction

Computer scientists are often wary of using variable substitutions as mathematical functions, due to some counter-intuitive pitfalls ((Palamidessi 1990), (Shepherdson 1994)). One of those pitfalls is *substitution generality*. With common understanding of generality, it is not obvious that substituting y for x, written as $\binom{x}{y}$, is not more general than $\binom{x}{a}$, where a is a constant, and neither that $\binom{x}{f(y,z)}$ is not more general than $\binom{x}{f(a,a)}$. Resorting to the definition does not remove all doubt: we learn that a substitution σ is more general than θ if there exists δ satisfying $\theta = \delta \cdot \sigma$ (Definition IV.1), but how to find δ? This paper proposes a remedy, at least for the latter shortcoming, by observing that substitution generality can be decided with a *term matching* algorithm.

The algorithm in its non-deterministic form originates from (Klop and de Vrijer et al. 2003). We present a deterministic version, using *relaxed core* representation of substitutions, defined in Section II. Term matching and the algorithm are discussed in Section III, and then used, together with *witness term*, to decide substitution generality in Section IV. This is further used for checking substitution equivalence in Section V and most general unifiers in Section VI. Apart from examples with concrete substitutions, we apply the technique of witness terms to prove two classical claims (Legacy V.3 and Legacy VI.3) in a more direct way.

II. Preliminaries

First we need a bit of notation, and let us begin with the concept of *term*[1]. Assume two disjoint sets: a countably infinite set $\mathbf{V}$ of *variables*, in this text x, y, z, u, possibly with indices, and a set $\boldsymbol{Fun}$ of *functors*, in this text f, g, a, b and $\circ, nil, |, []$. Associated with every functor f[2] shall be one or more natural numbers n denoting its number of arguments, *arity*. For disambiguation, the notation f/n will be used. Functors of arity 0 are called *constants*, in this text $a, b, nil, []$.

[1]Here, as in the programming language Prolog, *term* shall be the topmost syntactic concept: everything is a term.

Starting from $\mathbf{V}$ and $\boldsymbol{Fun}$ we build data objects, *terms*. Any variable x[2] $\in \mathbf{V}$ is a term. If $t_1, ..., t_n$ are terms and $f/n \in \boldsymbol{Fun}$, then $f(t_1, ..., t_n)$ is a term with *constructor* f and *outline* f/n. In case of $f/0$, the term shall be written without parentheses. If a term s occurs within a term t, we say s is a *subterm* of t and write $s \in t$.

The *ordered pair* of terms h and t is written in McCarthy's dot-notation as $(h \circ t)$, where h is called the *head* and t the *tail* of the pair (McCarthy 1960). A special ordered pair is a *non-empty list*, distinguished by its tail being a special term *nil* called the *empty list*, or a non-empty list itself. In Edinburgh Prolog notation, the above ordered pair would be written with brackets instead of parentheses and "|" instead of "∘", resulting in $[h|t]$, and empty list as $[]$ (Clocksin and Mellish 2003). A *list of n elements* is the term $[t_1|[t_2|[...[t_n|[]]]]]$, conveniently written as $[t_1, ..., t_n]$.

Let $\mathit{Vars}(t)$ 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 $s \not\bowtie t$. The list of all variables of t, in order of appearance, shall be denoted as $\mathit{VarList}(t)$.

To present the matching algorithm, we further need notions of *substitution*, its *relaxed cores*, and *subterm position*.

A. Substitution

Definition II.1 (substitution)**.** A *substitution* σ is a function mapping variables to terms, which is identity almost everywhere. In other words, a function σ with domain $\mathit{Dom}(\sigma) = \mathbf{V}$ such that $\mathit{Core}(\sigma) := \{x \in \mathbf{V} \mid \sigma(x) \neq x\}$ is finite. The set $\mathit{Core}(\sigma)$ shall be called the *active domain*[3] or *core* of σ, and its elements *active variables*[4] of σ. The *active range* of σ is $\mathit{Ran}(\sigma) := \sigma(\mathit{Core}(\sigma))$. A variable x such that $\sigma(x) = x$ shall be called a *passive variable* for σ. Also, we say that σ is

[2]Strictly speaking, there should be two sets of symbols: one for *object language* (terms themselves), and another for *meta-language* (meta-variables for denoting terms, e. g. capitalized as in Prolog, as well as meta-functors for handling terms, like $\mathbf{V}$ or Vars). However, due to our having lots of meta-variables, such strictness would result in lots of capital letters. So we decided to adopt the usual sloppy approach, with mostly lower-case meta-variables, meaning that f can be both an object-functor and a meta-variable denoting *any* object-functor, as befits the context; similarly for x etc. Meta-functors shall always be capitalized.

[3]Traditionally called just *domain*. This is confusing, since in the usual mathematical sense it is always the whole $\mathbf{V}$ (moreover, the set of all terms, p. 2) that is the domain of any substitution.

[4]The name *active variable* appears in (Jacobs and Langen 1992).

active on the variables from $\mathit{Core}(\sigma)$, and *passive* on all the other variables.

If $\mathit{Core}(\sigma) = \{x_1, ..., x_k\}$, with $x_1, ..., x_k$ pairwise distinct, and σ maps each x_i to t_i, then the *core representation* of σ is $\{x_1/t_1, ..., x_k/t_k\}$, often depicted as $\left(\begin{smallmatrix} x_1 & \dots & x_k \\ t_1 & \dots & t_k \end{smallmatrix}\right)$. Each x_i/t_i is called the *binding* for x_i in σ.

Often we identify a substitution with its core representation, and thus regard it as a syntactical object, a term representing a finite set. So we write $x_i/t_i \in \sigma$. For the same reason, the set of variables of a substitution is defined as $\mathit{Vars}(\sigma) := \mathit{Core}(\sigma) \cup \mathit{Vars}(\mathit{Ran}(\sigma))$.

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 $\sigma\restriction_W$ of a substitution σ to[5] a set of variables $W \subseteq \boldsymbol{V}$ is defined as follows: if $x \in W$ then $\sigma\restriction_W(x) := \sigma(x)$, otherwise $\sigma\restriction_W(x) := x$. The restriction of σ to the variables of t is abbreviated as $\sigma\restriction_t := \sigma\restriction_{\mathit{Vars}(t)}$. To denote the restriction of σ to variables outside of t, we use $\sigma\restriction_{-t} := \sigma\restriction_{\mathit{Core}(\sigma)\setminus \mathit{Vars}(t)}$.

The identity function on $\boldsymbol{V}$ is $\varepsilon := ()$. The *composition* $\theta \cdot \sigma$ of substitutions θ and σ is defined by $(\theta \cdot \sigma)(x) := \theta(\sigma(x))$. A substitution σ satisfying the equality $\sigma \cdot \sigma = \sigma$ is called *idempotent*. A *renaming* of variables is a finite permutation of variables, i.e. a substitution ρ with $\mathit{Core}(\rho) = \rho(\mathit{Core}(\rho))$. It has inverse ρ^{-1}, satisfying $\rho^{-1} \cdot \rho = \varepsilon$.

Substitution domain is enhanced from variables to arbitrary terms in a structure-preserving way by $\sigma(f(t_1, ..., t_n)) := f(\sigma(t_1), ..., \sigma(t_n))$. We say that $\sigma(t)$ is an *instance* of t via σ.

We shall refer to some well-known claims about substitutions as *legacy* claims. The first one is from (Eder 1985).

Legacy II.2 (idempotence)**.** *A substitution σ is idempotent, iff $\mathit{Core}(\sigma) \not\bowtie \mathit{Ran}(\sigma)$.*

B. Relaxed core representation

As seen above, only active pairs $x/\sigma(x)$, i.e. those with $x \neq \sigma(x)$, go into the core representation of σ. But the passive pairs can be interesting as well, as *placeholders*.

For example, assume there is a substitution σ mapping s on t; it is mapping each variable in s on a subterm of t, so it is possible that a variable stays the same. If we want our mapping to account for *all* variables in s, necessarily x/x would have to be tolerated as a "binding", a *passive binding*. In other words, the core of the substitution σ would have to be *relaxed* to allow some passive variables, raising those above the rest, as it were (Kulaš 2017).

Definition II.3 (relaxed core)**.** If $\mathit{Core}(\sigma) \subseteq \{x_1, ..., x_n\}$ and variables $x_1, ..., x_n$ are pairwise distinct, then $\{x_1, ..., x_n\}$ is a *relaxed core* for σ, and $\{x_1/\sigma(x_1), ..., x_n/\sigma(x_n)\}$ its *relaxed core representation*. If we fix a relaxed core for σ, it shall be denoted $C(\sigma) := \{x_1, ..., x_n\}$. The associated range $\sigma(C(\sigma))$ we denote as $R(\sigma)$.

[5]As remarked in (Klop and de Vrijer et al. 2003), the traditional wording "restriction to a set" is somewhat unfortunate, since the definition only gives $\mathit{Core}(\sigma) \subseteq W$ but not necessarily $\mathit{Core}(\sigma) = W$, as would be expected.

Actually, the placeholding capability of passive bindings is implicitly used each time when applying the well-known scheme for composing substitutions and a variable gets deactivated: $\left(\begin{smallmatrix} x & y \\ y & x \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) = \left(\begin{smallmatrix} \cancel{x} & \cancel{x} & y \\ \cancel{x} & \cancel{y} & x \end{smallmatrix}\right) = \left(\begin{smallmatrix} y \\ x \end{smallmatrix}\right)$. Here x gets deactivated, but it is not free, hence x/y has to be discarded. This is depicted by striking out the columns for x/x and x/y.

Inspired by this, we may strike out any passive pairs when depicting a relaxed core representation, like $\left(\begin{smallmatrix} \dots & \cancel{x} & \dots \\ \dots & \cancel{x} & \dots \end{smallmatrix}\right)$, to visually reconcile it with the traditional representation.

For extending, substitutions are treated like sets of active bindings, so disjoint union $\uplus$ may be used:

Definition II.4 (sum of substitutions)**.** Let $\sigma = \left(\begin{smallmatrix} x_1 & \dots & x_n \\ s_1 & \dots & s_n \end{smallmatrix}\right)$ and $\theta = \left(\begin{smallmatrix} y_1 & \dots & y_m \\ t_1 & \dots & t_m \end{smallmatrix}\right)$ be substitutions in relaxed representation, such that $\{y_1, ..., y_m\} \not\bowtie \{x_1, ..., x_n\}$. Then $\sigma \uplus \theta := \left(\begin{smallmatrix} x_1 & \dots & x_n & y_1 & \dots & y_m \\ s_1 & \dots & s_n & t_1 & \dots & t_m \end{smallmatrix}\right)$ is the *sum* of σ and θ.

Given a term t, does an extension of σ still map t in the same way?

Lemma II.5 (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 II.6 (complete for term)**.** Substitution σ in relaxed core representation is *complete for* t if $\mathit{Vars}(t) \subseteq C(\sigma)$.

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

Corollary II.7 (backward compatibility)**.** *If σ is complete for t, then for any θ holds: $\sigma \uplus \theta$ is complete for t and $(\sigma \uplus \theta)(t) = \sigma(t)$.*

Relaxed core representation shall be needed for the third argument and for the result of the matching algorithm *Match* (Section III), otherwise we assume substitutions in traditional, non-relaxed representation. As a visual reminder, a relaxed core of σ is denoted $C(\sigma)$, and the traditional core $\mathit{Core}(\sigma)$. The passive pairs are shown striked-out (Figure 4, Figure 6).

C. Subterm

Definition II.8 (subterm occurrence)**.** A character subsequence of the term t which is itself a term, s, shall be called an *occurrence* of the subterm s of t, denoted nondeterministically by $s \in t$. This may also be pictured as $t = \boxed{s}$.

There may be several occurrences of the same subterm in a term. Each occurrence is uniquely determined by its position. To identify positions in a term, we have to represent it as a tree. A variable x is represented by the tree consisting of the root labeled x and nothing else. A term $f(t_1, ..., t_n)$ is represented by the tree consisting of the root labeled f and of the trees for $t_1, ..., t_n$ as subtrees, ordered from left to right. Thus, the *root label* for a term t is t itself, if t is a variable, otherwise the constructor of t. Position of a subterm is defined via *access path*, which shall be a variation of the notion in (Apt 1997, p. 27), and used to define *pendants*, which include *disagreement pairs* from (Robinson 1965).

Definition II.9 (access path and pendants)**.** Let t be a term and consider an occurrence of its subterm s, denoted as $s \in t$. The *access path* of $s \in t$ is defined as follows. If $s = t$, then $AP(s \in t)$ is $r/0$, where r is the root label for t. If $t = f(t_1, ..., t_n)$ and $s \in t_k$ (where the same occurrence of s in t is meant), then $AP(s \in t) := f/k \circ AP(s \in t_k)$.

By extracting the integers from the access path of $s \in t$, we obtain the *position* of $s \in t$. For disambiguation, we may write $(s \in t)_p$ if p is the position of the chosen occurrence $s \in t$. By extracting the root labels, save for the last one, we obtain the *ancestry* of $s \in t$.

If $s \in t$ has the same position and ancestry as $s' \in t'$, then we say $s \in t$ and $s' \in t'$ are *pendants* in t and t'. A *disagreement pair* between t and t' is a pair of pendants therein differing in their last root label.

For example, let $t := [f(y), z]$ and $s := z$. There is only one ocurrence $s \in t$. According to preliminaries, $[f(y), z]$ is an abbreviation of $[f(y)|[z|[]]]$, so $AP(s \in t) = |/2 \circ |/1 \circ z/0$, The position of $s \in t$ is $2 \circ 1 \circ 0$ and its ancestry is $| \circ |$. An example of pendants: $f(y) \in [f(y), z]$ and $g(a, b) \in [g(a, b), h(x)]$. This is also a disagreement pair.

III. Term Matching

"Matching" or "pattern matching" is a central notion in text processing (e. g. *regular expression* matching) and artificial intelligence (for responding to the environment). It is a flexible way of function definition in programming languages (Hudak et al. 2000), and of rule application in term rewriting (Klop and de Vrijer et al. 2003).

In this paper a simple kind of syntactic comparison called *term matching* shall be handled. Consider terms $f(x, y)$ and $f(z, x)$. Intuitively, they "match" each other, while $f(x)$ and $g(x)$ do not. If asked about $f(x, x)$ and $f(x, y)$, we might have a harder time of it, but probably would consent that they match only in one direction. Namely, $f(x, y)$ can be "made to look like" $f(x, x)$ by substituting x for y, but not the other way around, meaning that $f(x, x)$ is "more concrete", "more specific" than $f(x, y)$.

So shall we now say that $f(x, y)$ matches $f(x, x)$, or vice versa? Let us agree upon the usage "general matches specific". Hence, in the above example $f(x, y)$ matches $f(x, x)$.

Such a notion appears under the name *subsumption* in (Robinson 1965), as well as *generalization* in (Plotkin 1971) and (Reynolds 1970). Other sources (Huet 1976, p. 44) and (Dwork et al. 1984) speak also of *(term) matching*.

Definition III.1 (term matching)**.** Let g, s be terms. If there is a substitution σ such that $\sigma(g) = s$, then we say that g *matches*[6] s, written as $g \leq s$,[7] and also that s is an *instance* of g. The substitution σ is then a *matcher* of g on s.

A matcher is *relevant*, if it has no extraneous variables, i. e. if $Vars(\sigma) \subseteq Vars([g, s])$.

The notion of *subsumption* has been used synonymously with term matching (Huet 1976). In Prolog programming, however, there has been a need for a stricter notion of subsumption, codified by the ISO standardization committee in (ISO 2012, Section 8.2.4) along the following lines.

Desideratum III.2 (strict subsumption)**.** Prolog built-in predicate $subsumes\_term/2$ must satisfy: $subsumes\_term(G, S)$ is true iff there is a substitution σ such that $\sigma(G) = \sigma(S) = S$.

Thus, $f(x)$ matches $f(g(x))$, but in the strict sense does not subsume it, while $f(x, y)$ subsumes $f(x, x)$.

How to verify whether a given term matches another one? For the introductory example, it is easy to find a matcher $\sigma = \binom{y}{x}$ and thus verify that $f(x, y)$ indeed matches $f(x, x)$ according to the definition. For arbitrary g, s it was observed in (Robinson 1965) that matching is actually one-sided *unification* (Section VI): g matches s iff g and $Freeze(s)$ unify, where $Freeze(s)$ is obtained from s by replacing each variable therein with a new constant. Hence, matching can be checked with a unification algorithm; but also, there are algorithms specifically made for matching. An efficient parallel algorithm is given in (Dwork et al. 1984). A simple non-deterministic algorithm is presented in (Klop and de Vrijer et al. 2003). We give a deterministic, one-pass version of it (Algorithm III.1).

---

variable Assume L is a variable. If $L/S \in \delta$ and $S \neq R$, then stop with $Failure("divergence")$. Otherwise, $Match(L, R, \delta) := \delta \cup \binom{L}{R}$.

failure: shrinkage If L is a non-variable, but R is a variable, stop with $Failure("shrinkage")$.

failure: clash If L and R are non-variables with different outlines, stop with $Failure("clash")$.

decomposition Let $L = f(s_1, ..., s_n)$ and $R = f(t_1, ..., t_n)$. If there exist substitutions $\delta_1 := Match(s_1, t_1, \delta)$, $\delta_2 := Match(s_2, t_2, \delta_1)$, ... up to $\delta_n := Match(s_n, t_n, \delta_{n-1})$, then $Match(L, R, \delta) := \delta_n$.

Algorithm III.1: One-pass term matching by $Match(L, R, \delta)$

The algorithm from (Klop and de Vrijer et al. 2003) was made deterministic using the placeholding facility of relaxed core representation. Actually, *without* the placeholding facility it would be difficult to capture the failure in matching $f(x, x)$ on $f(x, y)$ in just one pass along the terms, and without auxiliary registers.

[6]It has also been said that g *schematises* s (Huet 1976).

[7]Some authors like (Reynolds 1970), (Klop and de Vrijer et al. 2003) turn the symbol $\leq$ around. The bias towards $\leq$ stems from the original use of subsumption in automated theorem proving (Robinson 1965), where a disjunction of literals ("clause") is represented as set of literals, so a clause C *subsumes* a clause D if there is a substitution σ with $\sigma(C) \subseteq D$.

Theorem III.3 (term matching)**.** *If $Match(L, R, \varepsilon)$ stops with failure, then L does not match R. Otherwise, it stops with a substitution δ that is a relevant matcher of L on R. This follows from:*

1) *If $Match(L, R, \delta)$ stops with failure, there is no μ with $\mu(L) = R$ and $\mu \supseteq \delta$.*
2) *If $Match(L, R, \delta) = \delta'$, then $\delta'(L) = R$ and $\delta' \supseteq \delta$. In other words, $Match(L, R, \delta)$ is a matcher of L on R containing δ. Additionally, δ' is complete for L and $Vars(\delta') \subseteq Vars([L, R, \delta])$.*

Proof. The algorithm clearly always terminates. For 1), we need two observations, readily verified by structural induction:

- If μ maps L on R, then it maps any $s \in L$ on its pendant $t \in R$.
- Each time *Match* is (re-) called, its arguments L and R denote either the original terms, or some pendants therein.

Thus, in the non-variable failure cases of Algorithm III.1 there can be no matcher for the original terms, notwithstanding δ.

In the variable failure case, the purported matcher μ would have to map one variable on two different terms (Figure 1).

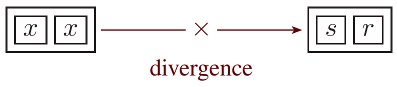

Fig. 1. One variable, two terms

To prove 2), we again use structural induction. In case of variable, the claim holds. Assume we have a case of decomposition and the claim holds for the argument terms, i. e. $\delta_1(s_1) = t_1$, $\delta_1 \supseteq \delta$, $\delta_2(s_2) = t_2$, $\delta_2 \supseteq \delta_1$, ..., $\delta_n(s_n) = t_n$, $\delta_n \supseteq \delta_{n-1}$, and each δ_i is complete for s_i as well as relevant for s_i, t_i. Due to completeness and Corollary II.7, from $\delta_n \supseteq ... \supseteq \delta_2 \supseteq \delta_1$ follows $\delta_n(s_1) = ... = \delta_2(s_1) = t_1$ and so forth. Hence, $\delta_n(L) = R$. Clearly, $\delta_n \supseteq \delta$ and δ_n is complete for L and relevant.

Finally, recall that the representation of δ_n may contain passive pairs, which can be discarded. ◇

IV. Substitution generality

As an application of the matching algorithm Algorithm III.1, we can solve the problem of generality and equivalence between two substitutions.

Definition IV.1 (more general)**.** A substitution σ is *more general*[8] than a substitution θ, written[9] as $\sigma \leq^\bullet \theta$, if σ is a right-divisor of θ, i. e. if there exists a substitution δ with the property $\theta = \delta \cdot \sigma$.

[8]Rather "no less general" or "at least as general", because the definition allows equal generality (Section V). However, such a formulation is somewhat cumbersome and thus rarely seen in literature.

[9]Some authors like (Jacobs and Langen 1992) and (Amato and Scozzari 2009) turn the symbol $\leq^\bullet$ around. Indeed the choice may appear to be abitrary. But we shall stick to the notion that a more general object is "smaller", because it correlates with the "smallness" of the substitution stack.

Clearly, $\varepsilon \leq^\bullet \theta$ for any θ. The claim $\binom{x}{y} \not\leq^\bullet \binom{x\ y}{y\ x}$ may not be so obvious, but shall soon be easy to prove.

Remark IV.2 ("$\leq$" versus "$\leq^\bullet$"). Term generality (as given in Definition III.1) and substitution generality have in common that an object is deemed "less concrete" than another one. Also, when generality goes both ways, the two objects are connected by a renaming (in case of substitutions, Legacy V.3).

Yet there is a difference. If we regard substitution as a special case of term (which is often the case, e. g. for extension), then an analogue for $s = \delta(t)$ would be $\sigma = \delta(\theta)$. But $\delta(\theta)$ is only meaningful for variable-pure δ injective on θ, and even then $\delta(\theta) \neq \delta \cdot \theta$ (Kulaš 2017, Sec. 5.3.2).

Hence, the analogy does not go the whole way, and perhaps for that reason it is not usual to speak of "substitution instance" or "substitution matching". As a token, we have not used the same symbol for both kinds of generality, despite tradition.

How to check whether $\sigma \leq^\bullet \theta$? One possibility would be to look for a counter-example, i. e. try to find a term w such that for no renaming δ holds $\delta(\sigma(w)) = \theta(w)$. Let us call such a term a *witness term* for σ, θ. How to obtain a witness term? Intuitively, we may take w to be the list of all variables of σ, θ, denoted $w := VarList([\sigma, \theta])$, and see if we can find an impasse, i. e. some parts of $\sigma(w)$ that cannot possibly simultaneously be mapped on the respective parts of $\theta(w)$. It turns out this is sufficient.

Theorem IV.3 (witness)**.** *Let σ, θ be substitutions and $w := VarList([\sigma, \theta])$. Then holds: $\sigma \leq^\bullet \theta$ iff $\sigma(w) \leq \theta(w)$.*

Additionally, if $Match(\sigma(w), \theta(w), \varepsilon) = \delta$, then $\delta \cdot \sigma = \theta$.

Proof. If $\delta \cdot \sigma = \theta$, then surely $\delta(\sigma(w)) = \theta(w)$ for any w.

For the other direction, assume there is a matcher μ of $\sigma(w)$ on $\theta(w)$, so $\mu(\sigma(w)) = \theta(w)$. Due to Theorem III.3, we can choose a relevant matcher by setting $\mu := Match(\sigma(w), \theta(w), \varepsilon)$, so $Vars(\mu) \subseteq Vars([\sigma, \theta])$. If for some $x \in \mathbf{V}$ holds $\mu(\sigma(x)) \neq \theta(x)$, then clearly $x \notin Vars([\sigma, \theta])$, hence the inequality becomes $\mu(x) \neq x$, meaning $x \in Core(\mu)$, which contradicts relevance of μ. ◇

The claim shows that $w := VarList([\sigma, \theta])$ is the only potential witness term ever needed: if there is an impasse, $Match(\sigma(w), \theta(w), \varepsilon)$ will find it, and if there is no impasse, it will find a matcher. Therefore, w shall be called the *complete candidate* for witness term for σ, θ.

As a consequence of the claim, we obtain a simple visual criterion.

Corollary IV.4 (witness)**.** *The relation $\sigma \leq^\bullet \theta$ does not hold, iff for some w with $Vars(w) \subseteq Vars([\sigma, \theta])$ any of the following holds:*

1) *At some corresponding positions, $\sigma(w)$ exhibits a non-variable, and $\theta(w)$ exhibits a variable ("shrinkage"), or a non-variable with a different outline ("clash").*
2) *$\sigma(w)$ exibits two occurrences of variable x, but at the corresponding positions in $\theta(w)$ there are two mutually distinct terms ("divergence of x").*

Some test runs are shown in the following pictures and in the two remaining legacy claims. Mostly the complete candidate is used, with the exception of Legacy VI.3.

Example IV.5 (subtlety of "more general")**.** As noted in (Apt 1997), $\sigma := \binom{x}{y}$ is more general than $\begin{pmatrix} x & y \\ a & a \end{pmatrix}$, but not more general than $\theta := \binom{x}{a}$. The former claim is justified by $\begin{pmatrix} x & y \\ a & a \end{pmatrix} = \binom{y}{a} \cdot \binom{x}{y}$. The matcher was here not difficult to guess, but in any case it can be found by Algorithm III.1 (Figure 2).

The latter claim is a simplified form of an example by Hai-Ping Ko, reported in (Shepherdson 1994, p. 148), which was pivotal in showing that the strong completeness theorem for SLD-derivation in (Lloyd 1987) does not always hold. The Ko example purports that $\sigma := \binom{x}{f(y,z)}$ is not more general than $\theta := \binom{x}{f(a,a)}$. For proof, it was observed: If $\delta \cdot \binom{x}{f(y,z)} = \binom{x}{f(a,a)}$, then $y/a, z/a \in \delta$, therefore even if one of y, z (but not both) were equal to x, at least one of the bindings $y/a, z/a$ has to be in $\binom{x}{f(a,a)}$, which does not hold.

Instead of such custom-made proofs, Algorithm III.1 could be used, giving divergence (Figure 3).

$$\begin{tikzcd}
 & {[x,y]} \arrow[dl, "\sigma = \begin{pmatrix} x & y \\ y & x \end{pmatrix}"'] \arrow[dr, "\theta = ()"] & \\
{[y,x]} \arrow[rr, "\delta = \begin{pmatrix} y & x \\ x & y \end{pmatrix}"'] & & {[x,y]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & {[x,y]} \arrow[dl, "\sigma = \binom{x}{y}"'] \arrow[dr, "\theta = \begin{pmatrix} x & y \\ a & a \end{pmatrix}"] & \\
{[y,y]} \arrow[rr, "\delta = \binom{y}{a}"'] & & {[a,a]}
\end{tikzcd}$$

Fig. 2. Successfull check on $\leq^\bullet$

V. Substitution Equivalence

The set of substitutions is not partially ordered by $\leq^\bullet$, namely it is possible that $\sigma \leq^\bullet \theta$ and $\theta \leq^\bullet \sigma$ for $\sigma \neq \theta$. Such cases form an equivalence relation, called simply *equivalence* and denoted by $\sigma \sim \theta$.

Example V.1 (subtlety of "equivalent")**.** Any two renamings ρ, δ are equivalent, as shown by $(\delta \cdot \rho^{-1}) \cdot \rho = \delta$ and vice versa. Hence also $\rho \sim \varepsilon$, so permuting any number of variables amounts to doing nothing. In particular, $\begin{pmatrix} x & y \\ y & x \end{pmatrix} \sim \varepsilon$, which is another often-cited example of counter-intuitive behaviour of substitutions (Palamidessi 1990).[10]

Clearly, if an algorithm decides substitution generality, then equivalence as well (Figure 4).

[10]Perhaps a new name like *equigeneral* instead of simply *equivalent* would be less confusing?

$$\begin{tikzcd}
 & {[x,y]} \arrow[dl, "\sigma = \binom{x}{a}"'] \arrow[dr, "\theta = \binom{x}{y}"] & \\
{[\underline{a},y]} \arrow[rr, "\times", "\text{shrinkage}"'] & & {[\underline{y},y]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & {[x,y]} \arrow[dl, "\sigma = \binom{x}{a}"'] \arrow[dr, "\theta = \binom{x}{b}"] & \\
{[\underline{a},y]} \arrow[rr, "\times", "\text{clash}"'] & & {[\underline{b},y]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & {[x,y,z]} \arrow[dl, "\sigma = \binom{x}{f(y,z)}"'] \arrow[dr, "\theta = \binom{x}{f(a,a)}"] & \\
{[f(\underline{y},z),\underline{y},z]} \arrow[rr, "\times", "\text{divergence}"'] & & {[f(\underline{a},a),\underline{y},z]}
\end{tikzcd}$$

Fig. 3. Failed check on $\leq^\bullet$

Example V.2 ("$\sim$" is not compositional)**.** (Eder 1985) shows that equivalence is not compatible with composition, as follows. Let $\sigma := \binom{y}{x}$, $\sigma' := \binom{x}{y}$ and $\theta := \binom{x}{z}$. Then $\sigma \sim \sigma'$, but $\theta \cdot \sigma = \begin{pmatrix} y & x \\ z & z \end{pmatrix} \not\sim \theta \cdot \sigma' = \binom{x}{y}$. The non-equivalence is verified by Algorithm III.1 in Figure 4.

$$\begin{tikzcd}
 & {[x,y,z]} \arrow[dl, "\begin{pmatrix} x & y \\ z & z \end{pmatrix}"'] \arrow[dr, "\binom{x}{y}"] & \\
{[z,\underline{z},\underline{z}]} \arrow[rr, "\times", "\text{divergence}"'] & & {[y,\underline{y},\underline{z}]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & {[x,y,z]} \arrow[dl, "\binom{x}{y}"'] \arrow[dr, "\begin{pmatrix} x & y \\ z & z \end{pmatrix}"] & \\
{[y,y,z]} \arrow[rr, "\delta = \begin{pmatrix} y & \not z \\ z & \not z \end{pmatrix}"'] & & {[z,z,z]}
\end{tikzcd}$$

Fig. 4. Checking equivalence

The following property has been proved in (Eder 1985), in similar form; the present formulation is from (Apt 1997). It can also be proved using witness terms.

Legacy V.3 (equivalence)**.** *θ is more general than θ' and θ' is more general than θ iff for some renaming ρ such that $Vars(\rho) \subseteq Vars(\theta) \cup Vars(\theta')$ holds $\rho \cdot \theta = \theta'$.*

Proof. If $\rho \cdot \theta = \theta'$ for an invertible substitution ρ, then clearly $\theta \leq^\bullet \theta'$ and $\theta \leq^\bullet \theta'$.

For the other direction, assume $\theta \leq^\bullet \theta'$ and $\theta' \leq^\bullet \theta$, and let $w := VarList([\theta, \theta'])$. By double application of Theorem IV.3, $Match(\theta(w), \theta'(w), \varepsilon)$ succeeds and $Match(\theta'(w), \theta(w), \varepsilon)$ succeeds. Observe that we have the *same* two terms in both of the *Match* calls, so the case where one of a pendant pair is a variable and the other a non-variable is clearly not possible (shrinkage failure). Hence, any bindings obtained are necessarily variable-pure, i. e. in both cases we have a variable-pure substitution with mutually distinct variables in range.[11] So let $\delta := \mathit{Match}(\theta(w), \theta'(w), \varepsilon)$. By construction, δ is relevant for θ, θ', and satisfies the generality equation $\delta{\cdot}\theta = \theta'$. Yet δ does not have to be a renaming, and we want one.

So assume δ is not a renaming, i. e. not a permutation of variables, and let us embed it in a relevant renaming. There is $y \in R(\delta)$ with $y \notin C(\delta)$, as in Figure 5. By construction, $y \in w$, so its whereabouts may be

1) $y \in \mathit{Core}(\theta)$
2) $y \in \mathit{Core}(\theta') \setminus \mathit{Core}(\theta)$
3) $y \in (\mathit{Ran}(\theta) \cup \mathit{Ran}(\theta')) \setminus (\mathit{Core}(\theta) \cup \mathit{Core}(\theta'))$

Actually, only the first case is possible: In the last case, $\theta(y) = \theta'(y) = y$, so $y/y \in \delta$, which contradicts the assumption $y \notin C(\delta)$ and is thus impossible. In the middle case $\theta(y) = y$ and $\theta(y) \neq y$, so $y/\theta'(y) \in \delta$, also impossible.

Therefore, y is bound by θ, hence $(\delta \uplus \binom{y}{\_}) \cdot \theta = \delta \cdot \theta$ for any binding $y/\_$. Thus, a binding for y can be added without disturbing the generality equation. The empty places _ in the (finitely many) added bindings can be freely[12] populated so as to obtain a finite permutation of variables.

For the example in Figure 5, there is only one choice for embedding: $\left(\begin{smallmatrix} x & y \\ y & x \end{smallmatrix}\right)$. ◇

$$\begin{tikzcd}
 & {[x, y]} \arrow[dl, "\theta = \begin{pmatrix} x & y \\ f(x) & x \end{pmatrix}"'] \arrow[dr, "\theta' = \begin{pmatrix} x \\ f(y) \end{pmatrix}"] & \\
{[f(x), x]} \arrow[rr, "\delta = \begin{pmatrix} x \\ y \end{pmatrix}"'] & & {[f(y), y]}
\end{tikzcd}$$

Fig. 5. Variable-pure matcher which is not a permutation

VI. Unification

Substitution generality is also involved in the concept of a *most general unifier* of two terms.

Definition VI.1 (unification)**.** Let s and t be terms. If there is a substitution θ such that $\theta(s) = \theta(t)$, then s and t are said to be *unifiable*, and θ is their *unifier*, the set of all such being $\mathit{Unifs}(s,t)$. We say θ is a *relevant* unifier, if it has no extraneous variables, i. e. if $\mathit{Vars}(\theta) \subseteq \mathit{Vars}([s,t])$.

A unifier θ of s and t is their *most general unifier* (*mgu*), if it is more general than any other; the set of all such shall be $\mathit{Mgus}(s,t) := \{\theta \in \mathit{Unifs}(s,t) \mid \theta \leq^{\bullet} \alpha \text{ for every } \alpha \in \mathit{Unifs}(s,t)\}$.

Any two unifiable terms have an idempotent (and relevant) most general unifier, as provided by Robinson's unification algorithm (Robinson 1965) or Martelli-Montanari's unification scheme (Martelli and Montanari 1982) recalled here briefly (Algorithm VI.1). It acts on finite equation sets, with the rationale that unifiability of an equation $f(s_1, \ldots, s_n){=}f(t_1, \ldots, t_n)$ entails unifiability of a set of equations $\{s_1{=}t_1, \ldots, s_n{=}t_n\}$, and vice versa.

The notion of unifier is extended to a set of equations by $\mathit{Unifs}(E) := \{\theta \mid \text{for every } s{=}t \in E \text{ holds } \theta(s) = \theta(t)\}$. Similarly for $\mathit{Mgus}(E)$.

To find an mgu of a finite set of equations E_0, take $E := E_0$ and transform E according to the (mutually disjoint) rules below. The transformation is bound to stop. If the stop is not due to failure, then the final set E determines an idempotent mgu of E_0 as $\{x/t \mid x{=}t \in E\}$.

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

elimination $E \uplus \{x{=}x\} \rightsquigarrow E$

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

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

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

Algorithm VI.1: Martelli-Montanari's non-deterministic unification scheme

Example VI.2. Consider unifying terms $s := f(x,y,u)$ and $t := f(z,z,u)$. Martelli-Montanari scheme applied on $s = t$ produces one mgu, $\mu := \left(\begin{smallmatrix} x & y \\ z & z \end{smallmatrix}\right)$. For the same but inverted task $t = s$ the scheme gives another two mgus, $\left(\begin{smallmatrix} x & z \\ y & y \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} y & z \\ x & x \end{smallmatrix}\right)$.

Apart from mgus obtained by the scheme, there are other relevant unifiers of s,t like $\sigma := \left(\begin{smallmatrix} x & y & z \\ u & u & u \end{smallmatrix}\right)$ and $\theta := \left(\begin{smallmatrix} x & y & z & u \\ u & u & u & z \end{smallmatrix}\right)$. Are they mgus as well? By the witness term technique we easily obtain $\sigma \not\leq^{\bullet} \theta$, i. e. σ is not more general than θ (and hence not an mgu), despite $\sigma \subseteq \theta$ (Figure 6). Also, $\theta \leq^{\bullet} \mu$, hence θ is a further mgu.

$$\begin{tikzcd}
 & {[x, y, z, u]} \arrow[dl, "\sigma = \begin{pmatrix} x & y & z \\ u & u & u \end{pmatrix}"'] \arrow[dr, "\theta = \begin{pmatrix} x & y & z & u \\ u & u & u & z \end{pmatrix}"] & \\
{[u, u, \underline{u}, \underline{u}]} \arrow[rr, "\times" description, "\text{divergence}"'] & & {[u, u, \underline{u}, \underline{z}]}
\end{tikzcd}$$

$$\begin{tikzcd}
 & {[x, y, z, u]} \arrow[dl, "\theta = \begin{pmatrix} x & y & z & u \\ u & u & u & z \end{pmatrix}"'] \arrow[dr, "\sigma = \begin{pmatrix} x & y & z \\ u & u & u \end{pmatrix}"] & \\
{[u, u, u, z]} \arrow[rr, "\delta = \begin{pmatrix} \cancel{u} & z \\ \cancel{u} & u \end{pmatrix}"'] & & {[u, u, u, u]}
\end{tikzcd}$$

Fig. 6. Restriction is not always more general

How many mgus can a pair of terms have? If $\sigma \in \mathit{Mgus}(E)$,

[11]In (Kulaš 2017), such substitutions are called *prenaming*s.
[12]For a "natural" choice, see (Kulaš 2017, Theorem 5.3).

then for any renaming ρ holds $\rho \cdot \sigma \in \mathit{Mgus}(E)$.[13] Hence, the set $\mathit{Mgus}(E)$ is either empty or infinite.

Finally, let us appply witness terms for an alternative proof of another well-known claim (Apt 1997):

Legacy VI.3 (relevance)**.** *Every idempotent mgu is relevant.*

Proof. Assume $\sigma \in \mathit{Mgus}(E)$ is idempotent, but not relevant, i. e. there is $z \in \mathit{Vars}(\sigma)$ with $z \notin \mathit{Vars}(E)$. Our plan is to refute the generality of σ, by finding a unifier θ of E such that $\neg(\sigma \leq^\bullet \theta)$. Technically, we construct θ and a witness term w satisfying Corollary IV.4.

a) Case $z \in \mathit{Core}(\sigma)$: Here we choose $\theta := \sigma\restriction_{-z}$. If σ is an idempotent unifier of E, then so is θ, by Legacy II.2.

Subcase 1: $\sigma(z)$ is ground. Take $w := z$. Then $\theta(w) = z$ is a variable, so we have shrinkage.

Subcase 2: $\sigma(z)$ contains a variable, say x, pictured as $\sigma(z) = \boxed{x}$. By Legacy II.2, $x \notin \mathit{Core}(\sigma)$, so $x \neq z$. Let $w := [x, z]$. Then $\sigma([x, z]) = [x, \boxed{x}]$, whereas $\theta([x, z]) = [x, z]$. If $\boxed{x}$ is not a variable, we have shrinkage, otherwise divergence.

b) Case $z \in \mathit{Ran}(\sigma) = \mathit{Ran}(\sigma) \setminus \mathit{Core}(\sigma)$: There is $x \in \mathit{Core}(\sigma)$ (and therefore $x \neq z$) with $\sigma(x) = \boxed{z}$. Here we take $w := [x, z]$ and θ to be a relevant mgu of E (e. g. an outcome of the Martelli-Montanari scheme). Then $\sigma([x, z]) = [\boxed{z}, z]$ and $\theta([x, z]) = [\theta(x), z]$, with $z \notin \theta(x)$ due to relevance. Even if $\boxed{z} \leq \theta(x)$, we obtain a failure (divergence of z). $\Diamond$

As a bonus, we also give an indirect proof, via Legacy V.3. It is similar in spirit to the one from (Apt 1997), insofar as they both rely on "groping around" for a contradiction, which is then put together as a logical, yet somewhat artificial chain of reasoning. Arguably, the witness-based proof above is more focused.

Proof (II). As before, assume $\sigma \in \mathit{Mgus}(E)$ is idempotent, but not relevant, so there is $z \in \mathit{Vars}(\sigma) \setminus \mathit{Vars}(E)$. Take θ to be an mgu for E obtained by Martelli-Montanari scheme, so θ is relevant and thus $z \notin \mathit{Vars}(\theta)$. Also, by Legacy V.3, there is a renaming ρ with

$$\sigma = \rho \cdot \theta \tag{1}$$

a) Case: $z \in \mathit{Core}(\sigma)$: We refute by showing $z = \sigma(z)$, which means $z \notin \mathit{Core}(\sigma)$:

$$y := \sigma(z) = \rho(\theta(z)) = \rho(z), \text{ by relevance of } \theta \text{ and (1)} \tag{2}$$
$$\sigma(y) = \sigma^2(z) = \sigma(z) = y, \text{ idempotency of } \sigma \text{ and (2)} \tag{3}$$
$$y = \sigma(y) = \rho(\theta(y)), \text{ by (3) and (1)} \tag{4}$$
$$\rho(z) = \rho(\theta(y)), \text{ by (2) and (4)} \tag{5}$$
$$z = \theta(y), \text{ by injectivity of } \rho \text{ and (5)} \tag{6}$$
$$z = y, \text{ by relevance of } \theta \text{ and } z \notin \mathit{Vars}(\theta) \text{ and (6)} \tag{7}$$

[13]In fact, any θ from $\mathit{Mgus}(E)$ can be thus obtained, due to Legacy V.3.

b) Case: $z \in \mathit{Ran}(\sigma)$: Then for some $x \in \mathit{Core}(\sigma)$ holds $z \in \sigma(x)$. By Legacy II.2, $z \neq x$. We refute by showing $z = x$.

$$\sigma(x) = \sigma(\sigma(x)) = \rho(\theta(\sigma(x))), \text{ idempotency and (1)} \tag{8}$$
$$\rho(\theta(x)) = \rho(\theta(\sigma(x))), \text{ by (1) and (8)} \tag{9}$$
$$\theta(x) = \theta(\sigma(x)), \text{ by injectivity of } \rho \tag{10}$$
$$z \in \theta(\sigma(x)), \text{ by } z \in \sigma(x) \text{ and } \theta(z) = z \tag{11}$$
$$z \in \theta(x), \text{ by (10) and (11)} \tag{12}$$
$$z = x, \text{ by relevance and (12)} \tag{13}$$

$\Diamond$

VII. Outlook

A simple one-pass *term matching* algorithm is proposed. It is a deterministic version of the algorithm given in (Klop and de Vrijer et al. 2003), enabled by *relaxed core* representation of substitutions, where some finite number of placeholding pairs x/x may appear.

The algorithm also decides *substitution generality*, and hence equivalence of two substitutions, when applied on their complete *witness term* candidate. The witness term method alleviates the need for ad-hoc proofs involving substitution generality.

Acknowledgement

Many thanks to M. D. Kulaš for spotting some ambiguities.

References

G. Amato and F. Scozzari, "Optimality in goal-dependent analysis of sharing," *Theory and Practice of Logic Programming*, vol. 9, no. 5, pp. 617–689, 2009.

K. R. Apt, *From logic programming to Prolog*. Prentice Hall, 1997.

W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, 5th ed. Springer-Verlag, 2003.

C. Dwork, P. Kanellakis, and J. C. Mitchell, "On the sequential nature of unification," *J. Logic Programming*, vol. 1, pp. 35–50, 1984.

E. Eder, "Properties of substitutions and unifications," *J. Symbolic Computation*, vol. 1, no. 1, pp. 31–46, 1985.

P. Hudak, J. Peterson, and J. Fasel, *A gentle introduction to Haskell*, 2000, version 0.98, http://www.haskell.org/tutorial.

G. Huet, "Résolution d'équations dans des langages d'ordre 1,2,...,ω," Ph.D. dissertation, U. Paris VII, 1976, available on http://cristal.inria.fr/~huet/bib.html.

ISO, *Information technology - Programming languages - Prolog - Part 1: General core. Technical Corrigendum 2*, ISO/IEC JTC 1/SC 22, 2012, iSO/IEC 13211-1:1995/Cor.2:2012(en). https://www.iso.org/obp/ui/#iso:std:58033:en.

D. Jacobs and A. Langen, "Static analysis of logic programs for independent AND parallelism," *J. of Logic Programming*, vol. 13, no. 2-3, pp. 291 – 314, 1992.

J. W. Klop and R. de Vrijer et al., Eds., *TeReSe: Term Rewriting Systems*. Cambridge University Press, 2003, ch. First-order term rewriting systems, excerpt on http://www.cs.vu.nl/~tcs/trs.

M. Kulaš, "A practical view on renaming," in *Proc. WLP'15/'16 and WFLP'16*, ser. EPTCS, S. Schwarz and J. Voigtländer, Eds., vol. 234, 2017, pp. 27–41, https://arxiv.org/abs/1701.00624.

J. W. Lloyd, *Foundations of logic programming*, 2nd ed. Springer-Verlag, 1987.

A. Martelli and U. Montanari, “An efficient unification algorithm,” *ACM Trans. on Prog. Lang. and Systems*, vol. 4, no. 2, pp. 258–282, 1982.

J. McCarthy, “Recursive functions of symbolic expressions and their computation by machine,” *Comm. of ACM*, vol. 3, no. 4, pp. 184–195, 1960.

C. Palamidessi, “Algebraic properties of idempotent substitutions,” in *Proc. 17th ICALP*, ser. LNCS, vol. 443. Springer-Verlag, 1990, pp. 386–399.

G. D. Plotkin, “Automatic methods of inductive inference,” Ph.D. dissertation, U. of Edinburgh, 1971, available on http://homepages.inf.ed.ac.uk/gdp.

J. C. Reynolds, “Transformational systems and the algebraic structure of atomic formulas,” in *Machine Intelligence 5*, B. Meltzer and D. Michie, Eds. Edinburgh University Press, 1970, pp. 135–151.

J. A. Robinson, “A machine-oriented logic based on the resolution principle,” *J. of ACM*, vol. 12, no. 1, pp. 23–41, 1965.

J. C. Shepherdson, “The role of standardising apart in logic programming,” *Th. Comp. Sci.*, vol. 129, no. 1, pp. 143–166, 1994.