Unification modulo α-equivalence in a mathematical assistant system

Temur Kutsia

RISC, Johannes Kepler University, Linz, Austria

Abstract

We study unification modulo α-equivalence in a language that combines permissive nominal terms and sequence unknowns. Such unification problems originate from reasoning tasks in the mathematical assistant system Theorema. We propose an algorithm that combines a version of permissive nominal unification with length-bounded sequence unification. It is terminating, sound, minimal, and satisfies a restricted version of completeness. We also consider two special cases when the boundedness restriction can be lifted: (1) matching fragment and (2) the fragment where sequence unknowns appear in the last argument positions in subterms. They permit minimal and complete algorithms. All three algorithms are implemented and included in the unification package of the Theorema system.

Keywords and phrases: Permissive nominal unification, α-equivalence, mathematical assistant systems, Theorema, sequence variables.

AMS subject classification: 03B70, 68Q42, 68W30, 68T15.

1 Introduction

Unification is a procedure for symbolic equation solving, used as the main computational mechanism in many automated deduction methods. Given two logical expressions, unification algorithms try to find instantiations of variables to make the expressions identical (syntactic unification) or equal modulo an equational theory (equational unification). Unification is a key ingredient in theorem provers, proof assistants, and declarative programming systems.

In this paper we consider a particular variant: unification modulo α-equivalence. Two logical expressions are α-equivalent, if they are the same modulo renaming of bound variables. The algorithm described here corresponds to what is implemented in the mathematical assistant system Theorema [10].

The specific features of Theorema influenced the design of the algorithm. Theorema provides a pretty liberal higher-order syntax. Its expression may be a constant, a variable, an application of an expression to a sequence of expressions, or a quantified expression. Variables are of two kinds: for individual expressions (individual variables, here called term variables) and for sequences of expressions (sequence variables). Arities of function constants, in general, are not fixed.

As an example of α-unification, consider an equation between two statements about sets from [10]: $X + 1 \in \{x \mid_x x > X\} \approx^?_\alpha Y \in \{y \mid_y y > a\}$, where X and Y are individual variables to be instantiated, and x and y are variables bound by the set quantifier. The algorithm computes the unifier $\sigma = \{Y \mapsto a + 1, X \mapsto a\}$, which maps Y to $a + 1$ and X to a. Applying σ to the given problem, we get $a + 1 \in \{x \mid_x x > a\}$ in the left and $a + 1 \in \{y \mid_y y > a\}$ in the right, which are not identical, but equal modulo renaming the bound variable y into x.

Sequence variables are very handy in knowledge representation and rule-based programming. They play an important role both in Theorema and in the programming language this system is implemented in: the Wolfram language of the symbolic computation system Mathematica [67]. However, the expressive power of sequence variables makes unification with them pretty hard. There are problems which may have infinitely many independent unifiers even for the syntactic case. For instance, the equation $f(\overline{x}, a) =^? f(a, \overline{x})$ has infinitely many solutions, mapping $\overline{x}$ to finite (including empty) sequences of a's: $\{\overline{x} \mapsto ()\}, \{\overline{x} \mapsto (a)\}, \{\overline{x} \mapsto (a, a)\}, \ldots$.

As it was pointed out in [10], despite the fact that Theorema provides higher-order syntax, there is no hidden default higher-order logic behind it. In the process of developing an α-unification algorithm for this language, we chose a pragmatic, minimalistic approach, since α-equivalence is the fundamental property of languages with binders. The idea was to provide the basic algorithm that deals with language constructs such as quantifiers/binders, applicative expressions, and sequence variables. In specific reasoners, the algorithm either can be used as provided, or it may be extended/modified to meet the needs of that particular reasoner. For instance, for a special prover for higher-order logic, one may wish to extend the algorithm to deal with equalities modulo β and η rules, while, e.g., for first-order reasoning, the provided algorithm would suffice.

To illustrate the mentioned features of this approach, we recall examples from [10]: For instance, in our language, the equation $X(a) \approx^?_\alpha f(a, a)$ does not have a solution, because unification is not done modulo β (in contrast to four unifiers when the β-rule is permitted). Note also that $f(a)(a)$ and $f(a, a)$ are not seen as equal. The problem $X(a) \approx^?_\alpha f(a)(a)$ can be solved by $\{X \mapsto f(a)\}$.

To distinguish between the variables that are bound in (the context of) an expression, and the variables that are free and can be instantiated by unification, we call the former *atoms* (as in nominal unification [63]) and keep the word 'variable' only for the latter.

An important feature in the algorithm described in this paper is the use of so called permission sets, like in permissive nominal unification [17]. The permission set of a variable explicitly indicates which atoms may appear in the instantiation of that variable during unification. For instance, $x^{\{a,b\}}$ means that in the instantiations of x, only a and b are permitted from the atoms: $\{a, b\}$ is the permission set for x. Hence, $x^{\{a,b\}}$ may be unified, e.g., with the terms $f(a, g(a))$ or $f(y^{\{b\}}, a)$, but not with $f(c)$, where c is an atom, because c does not belong to the permission set.

We call pairs consisting of a variable and a permission set *unknowns*. In Theorema, they arise in the context of proving. For instance, an attempt to prove $\forall x.\exists y.\ f(x) = y$ gives a unification problem $f(a) =^? y^{\{a\}}$, where a is an atom (an arbitrary but fixed constant obtained after removing the universal quantifier) and $y^{\{a\}}$ is the unknown, whose instantiation is to be computed. The atom a is permitted in the instantiation. The unification problem can be solved by the substitution $\{y^{\{a\}} \mapsto f(a)\}$, which leads to the proof of the statement. On the other hand, proving $\exists y.\forall x.\ f(x) = y$ fails, because it gives the unification problem $f(a) =^? y^{\emptyset}$, which does not have a solution: $f(a)$ is not permitted in $y^{\emptyset}$, since the empty permission set forbids the atom a to appear in the instantiation.

In this work, we describe an algorithm Unif-Alg for solving such unification problems. They may contain unknowns for terms and sequences, atoms, variadic function symbols, applications, and binders. To guarantee the termination of the algorithm, the length of instantiations of sequence unknowns is limited. Termination, soundness, and restricted completeness of the algorithm are shown. The set of unifiers it computes is minimal. We also identify two fragments, for which termination and completeness can be obtained without limiting the length of sequence unknown instantiations: matching fragment (equations where one side is unknown-free) and the fragment, where sequence unknowns occupy the last argument positions in subterms they occur.

The plan of the paper is following: after a brief overview of related work, we introduce the language, define terms, substitutions, unification problems and related notions in Section 2. In Section 3, we describe the unification algorithm Unif-Alg and two other algorithms for special fragments: Match-Alg and Unif-Alg-Last. Properties of these three algorithms are investigated in Section 4, where theorems about termination, soundness, completeness and restricted completeness are proved.

The algorithms are implemented in Mathematica and are a part of the Theorema system.

Related work

α-Unification

Unification modulo α-equivalence has been studied in [63] in the context of nominal terms. Nominal techniques, introduced in [25, 26], extend first-order syntax by names and bindings, where binders quantify names in their arguments. The syntax still remains first-order. Functional abstraction λ, logic quantifiers $\forall, \exists$, integral $\int$ are some well-known examples of binders.[1] The motivation for introducing nominal techniques was to formally represent and study systems with binding. These techniques syntactically distinguish between atoms (object level variables), which can be bound, and unknowns (meta-variables), which can be substituted. Substitutions may cause atom capture by binders. Renaming of atoms is made explicit by their name swapping (which avoids capture). Informal 'fresh variable conditions' is made a part of the language under freshness constraints.

Nominal unification has good algorithmic properties: it is decidable, unitary, and can be solved in polynomial time. Unification, matching, and related problems in nominal setting are quite actively investigated nowadays. Various kinds of equation solving methods between nominal terms, and their relations to similar problems have been studied by several authors, see, e.g., [2, 3, 6, 11–13, 23, 24, 47, 48, 60]. Permissive nominal unification, introduced in [17], differs from nominal unification in that it changes the idea of 'specifying which atoms are forbidden in instantiations' into 'specifying which atoms are permitted in instantiations'. It has several advantages, outlined in [17], including the possibility to always choose a fresh atom and the substitution-only based notion of unifier. A nice survey on permissive-nominal logic can be found in [27].

Permission sets, in general, may be infinite, but in the context of their application in proof-search, finite ones suffice [28]. This applies to our case as well, because our unification problems originate from tasks in Theorema reasoners. Note that our unification problems avoid binding atoms from permissive sets. Hence, substitutions do not cause atom capture.

[1]See [57] for rules about introducing new binders in the language.

Also, two distinct unknowns do not share the same variable. Another difference from [17] is the way how atoms are renamed. In nominal techniques, this is done by permutations, which has many advantages [28]. However, we stick to a more familiar way of atom replacements such as $[a := b]$ and rely on the capabilities of the meta-language to generate fresh names.

Unification with sequence variables

Sequence variables come hand-in-hand with variadic symbols (i.e., those without the fixed arity). Such symbols are pretty common. They can be, e.g., names in Common Logic [34] and KIF [29], XML tags, symbols originated from different knowledge bases after their integration, functions and constructors implemented in symbolic computation systems (e.g., Mathematica), arithmetic operations written in variadic form, flexary symbols in OpenMath [33], etc. Unification with sequence variables is infinitary (the minimal complete set of unifiers for some problems can be infinite).

Incomplete unification algorithms, motivated by applications, have been proposed in [30, 58]. A complete procedure was introduced in [40, 42]. Decidability was proved in [40, 44] and various terminating fragments have been studied in [44,46]. Matching with sequence variables modulo equational theories and its relation with the built-in pattern matching mechanism of Mathematica was investigated in [21]. Among various applications of equation solving with sequence variables one can mention knowledge representation [52], rule-based and constraint (logic) programming [19, 22, 50, 61], rewriting [20], theorem proving [41], XML processing [14,15,45], etc. The main variant we consider in this paper corresponds to bounded-length sequence unification. The idea of imposing such a length bound has been used earlier for dealing with sequence equations and constraints in constraint programming solver [61], program synthesis [58], ontology reasoning [56], etc. To the best of our knowledge, sequence unification and permissive nominal unification have not been combined before.

Theorema

The Theorema project has been initiated by Bruno Buchberger in the mid-1990s [8]. The goal was to develop a software system that aids all the phases of mathematical theory exploration. It includes invention of mathematical concepts; formulation and proof of propositions; formulation of problems; formulation, verification, and execution of algorithms for solving problems; maintenance of knowledge bases developed and verified in this process, and retrieval of mathematical knowledge. Many proof assistant systems and dedicated tools support various aspects of theory exploration, see, e.g., [1, 5, 7, 16, 31, 32, 35–37, 51, 53–55, 62]. Theorema has been used, for instance, for the development and implementation of a new method for solving linear boundary value problems [59], for the automated synthesis of Buchberger's algorithm for computing Gröbner bases [9], for formalizing pillage games in theoretical economics [38], for theory exploration in reduction rings [49], for synthesis of sorting algorithms for binary trees [18], just to name a few.

The object language of Theorema is a version of a higher-order language with sequence variables. Its meta-language for implementing reasoners (provers, solvers, simplifiers) is Mathematica. Theorema provides a modern GUI [66] and an infrastructure for developing special reasoners and for combining them into more general tools. Examples of special reasoners implemented in Theorema are provers for first-order predicate logic [39], set theory [65], equational logic [43], elementary analysis [64], a package for Green's algebra [59], etc. They all rely in a way or another on the unification package of Theorema. This package has been modified and improved several times since its initial implementation in the first version of the system at the end of 1990s. The algorithms we describe in this paper are a part of the new unification package in Theorema 2.0 [10].

2 Preliminaries

We consider an alphabet A consisting of the following pairwise disjoint countable sets of symbols:

- $\mathcal{V}_\mathrm{T}$: the set of term variables,
- $\mathcal{V}_\mathrm{S}$: the set of sequence variables,
- $\mathcal{A}$: the set of atoms,
- $\mathcal{F}$: the set of variadic function symbols,
- $\mathcal{Q}$: the set of quantifiers.

Definition 1 (Terms, s-terms)**.** A *term* t and an *s-term* s over the alphabet A are defined by the grammar:

$$t ::= x^P \mid a \mid f \mid t(s_1, \dots, s_n) \mid Qa.t$$
$$s ::= t \mid \overline{x}^P$$

where $x \in \mathcal{V}_\mathrm{T}$, $P \subset \mathcal{A}$, $a \in \mathcal{A}$, $f \in \mathcal{F}$, $Q \in \mathcal{Q}$, $\overline{x} \in \mathcal{V}_\mathrm{S}$, and $n \geq 0$. The set P is assumed to be finite. It is called a *permission set.*

The expressions x^P and $\overline{x}^P$ are called term and sequence unknown, respectively. Note that variadic function symbols may apply to arbitrary number of arguments, the terms f and $f()$ are not assumed to be the same, and quantifiers operate on atoms. Note also a restricted use of sequence unknowns. Namely, a sequence unknown can be neither the head of a term nor the body of a quantifier, i.e., expressions such as $\overline{x}^P(a, b)$ and $Qa.\overline{x}^P$ are not terms.

We write sequences of s-terms in parentheses for readability. Below the following meta-variables are used:

- for term variables: x, y, z,
- for sequence variables $\overline{x}, \overline{y}, \overline{z}$,
- for term or sequence variables: v, w,
- for atoms: a, b, c, d,
- for function symbols: f, g, h,
- for quantifiers: Q,
- for terms: t, u,
- for s-terms: s, r,
- for finite sequences of terms: $\tilde{t}, \tilde{u}$,
- for finite sequences of s-terms: $\tilde{s}, \tilde{r}$.

Example 1. Let $a, b, c \in \mathcal{A}$, $f, g, h \in \mathcal{F}$, $\lambda \in \mathcal{Q}$. Then the following expressions are terms: f, $f()$, $f(a, f, x^{\emptyset})$, $f(a,b)(c, \overline{x}^{\{a,c\}})$, $\lambda a.f(a, x^{\{a\}})$, $(\lambda a.f(a))(g)$, $(\lambda a.f(a)(b))(g(a, \overline{x}^{(\{a,b\})})$.

The *head* of a term t, denoted by $\mathsf{head}(t)$, is defined as $\mathsf{head}(x^P) = x^P$, $\mathsf{head}(a) = a$, $\mathsf{head}(f) = f$, $\mathsf{head}(t(\tilde{s})) = t$, and $\mathsf{head}(Qa.t) = Q$.

Definition 2 (Free, bound atoms)**.** The sets of *free and bound atoms* of an s-term s, denoted respectively by $\mathsf{fa}(s)$ and $\mathsf{ba}(s)$, are defined as follows:

$$\mathsf{fa}(x^P) = P, \quad \mathsf{fa}(\overline{x}^P) = P, \quad \mathsf{fa}(f) = \emptyset, \quad \mathsf{fa}(a) = \{a\},$$
$$\mathsf{fa}(t(s_1, \dots, s_n)) = \mathsf{fa}(t) \cup \cup_{i=1}^{n} \mathsf{fa}(s_i),$$
$$\mathsf{fa}(Qa.t) = \mathsf{fa}(t) \setminus \{a\}.$$

$$\mathsf{ba}(x^P) = \mathsf{ba}(\overline{x}^P) = \mathsf{ba}(f) = \mathsf{ba}(a) = \emptyset,$$
$$\mathsf{ba}(t(s_1, \dots, s_n)) = \mathsf{ba}(t) \cup \cup_{i=1}^{n} \mathsf{ba}(s_i),$$
$$\mathsf{ba}(Qa.t) = \mathsf{ba}(t) \cup \{a\}.$$

Further:

$$\mathsf{fa}((s_1, \dots, s_n)) = \mathsf{fa}(s_1) \cup \dots \cup \mathsf{fa}(s_n).$$
$$\mathsf{ba}((s_1, \dots, s_n)) = \mathsf{ba}(s_1) \cup \dots \cup \mathsf{ba}(s_n).$$

$$\mathsf{atoms}(s) = \mathsf{fa}(s) \cup \mathsf{ba}(s). \qquad \mathsf{atoms}(\tilde{s}) = \mathsf{fa}(\tilde{s}) \cup \mathsf{ba}(\tilde{s}).$$

The set of all unknowns of s (resp. of $\tilde{s}$) is denoted by $\mathsf{unkn}(s)$ (resp. $\mathsf{unkn}(\tilde{s})$).

Definition 3 (Substitution)**.** A *substitution* σ is a mapping, which maps unknowns to terms and to sequences of s-terms and is defined as follows:

- for each x^P, $\sigma(x^P)$ is a term,
- for each $\overline{x}^P$, $\sigma(\overline{x}^P)$ is a finite sequence of s-terms

such that

- $\mathsf{fa}(\sigma(v^P)) \subseteq P$ for all $v \in \mathcal{V}_{\mathrm{T}} \cup \mathcal{V}_{\mathrm{S}}$,
- $\sigma(x^P) = x^P$ for all but finitely many term unknowns,
- $\sigma(\overline{x}^P) = (\overline{x}^P)$ for all but finitely many sequence unknowns,
- if $\sigma(v^P) \neq v^P$ for some $v \in \mathcal{V}_{\mathrm{T}} \cup \mathcal{V}_{\mathrm{S}}$ and P, then $\sigma(v^R) = v^R$ for the same v and all $B \neq R$.

Usually, substitutions are written as finite sets of mapping pairs. For instance, $\{x^{\{a,b\}} \mapsto Qa.f(a)(\overline{y}^{\{b\}}),\ y^{\{a\}} \mapsto g(a, f),\ \overline{x}^{\{a,b\}} \mapsto (f(a), Qb.b, b), \overline{y}^{\{b\}} \mapsto ()\}$ is a substitution, which maps $x^{\{a,b\}}$ to $Qa.f(a)(\overline{y}^{\{b\}})$, $y^{\{a\}}$ to $g(a, f)$, $\overline{x}^{\{a,b\}}$ to the sequence $(f(a), Qb.b, b)$, and $\overline{y}^{\{b\}}$ to the empty sequence $()$. The other term unknowns are mapped to themselves, and the other sequence unknowns are mapped to themselves as singleton sequences.

We use lower case Greek letters for substitutions. The only exception is the identity substitution, denoted by Id. The *domain* and *range* of a substitution σ are defined as

$$dom(\sigma) := \{x^P \mid \sigma(x^P) \neq x^P\} \cup \{\overline{x}^P \mid \sigma(\overline{x}^P) \neq (\overline{x}^P)\}$$

$$ran(\sigma) := \{\sigma(v^P) \mid v^P \in dom(\sigma)\}.$$

Given a set of unknowns S, the *restriction* of a substitution σ on S, denoted by $\sigma|_S$, is a substitution for which $\sigma|_S(v^P) = \sigma(v^P)$ if $v^P \in S$, and $\sigma|_S(v^P) = v^P$ otherwise.

Substitutions can be composed in the usual way, see, e.g., [4]. We write $\sigma\vartheta$ for the composition of substitutions σ and ϑ (the order matters).

A substitution σ is *idempotent*, if $\sigma\sigma = \sigma$. The defining property of idempotent substitutions is $dom(\sigma) \cap \mathsf{unkn}(ran(\sigma)) = \emptyset$.

Definition 4 (Replacement)**.** An *atom replacement* or, shortly, *replacement* is a mapping from an atom to an atom, written as $[a := b]$.

Replacements and substitutions can be applied to terms according to the following definitions:

Definition 5 (Replacement application)**.** Application of a replacement $[a := b]$ to an s-term s, denoted $s[a := b]$, is defined as follows:

$$v^P[a := b] = v^{P[a:=b]}, \quad \text{where}$$
$$\text{if } a \in P, \text{ then } P[a := b] = (P \setminus \{a\}) \cup \{b\}, \text{ else } P[a := b] = P.$$
$$f[a := b] = f, \quad a[a := b] = b, \quad c[a := b] = c \text{ if } c \neq a,$$
$$t(s_1, \dots, s_n)[a := b] = t[a := b](s_1[a := b], \dots, s_n[a := b]),$$
$$(Qc.t)[a := b] = Qc[a := b].t[a := b].$$

Note that replacement application allows atom capture: $Qa.f(a, b)[b := a] = Qa.f(a, a)$.

Definition 6 (Substitution application)**.** Application of a substitution σ to an s-term s, denoted $s\sigma$, is defined as follows:

$$v^P\sigma = \sigma(v^P), \quad a\sigma = a, \quad f\sigma = f, \quad t(s_1, \dots, s_n)\sigma = t\sigma(s_1\sigma, \dots, s_n\sigma),$$
$$(Qa.t)\sigma = Qb.t[a := b]\sigma, \text{ where } b \notin \mathsf{atoms}(t).$$

Substitution application avoids atom capture, unlike replacements. For instance, we have $Qa.f(a, x^{\{a\}})\{x^{\{a\}} \mapsto a\} = Qb.f(b, a)$.

Definition 7 (The relation $\approx_\alpha$)**.** The relation $\approx_\alpha$ on s-terms is the smallest relation that satisfies the following:

$$v^P \approx_\alpha v^P, \quad a \approx_\alpha a, \quad f \approx_\alpha f,$$
$$t^1(s^1_1, \dots s^1_n) \approx_\alpha t^2(s^2_1, \dots s^2_n) \text{ if } t^1 \approx_\alpha t^2 \text{ and } s^1_i \approx_\alpha s^2_i \text{ for } i = \overline{1, n},$$
$$Qa.t \approx_\alpha Qb.u, \text{ if } t[a := c] \approx_\alpha u[b := c] \text{ where } c \notin \mathsf{atoms}(t) \cup \mathsf{atoms}(u).$$

It can be proved that $\approx_\alpha$ is a congruence relation. It is called the α-equivalence. Essentially, two s-terms are α-equivalent if they are equal modulo bound atom renaming.

Definition 8 (Instantiation quasi-ordering)**.** An s-term s is *more general* than r (r is an *instance* of s), written $s \precsim r$, if there exists a substitution σ such that $s\sigma \approx_\alpha r$.

A substitution σ is more general than ϑ, written $\sigma \precsim \vartheta$, if there exists a substitution φ such that $x^P\sigma\varphi \approx_\alpha x^P\vartheta$ for any x^P.

The relation $\precsim$ is quasi-ordering (a reflexive and transitive binary relation). It is called *instantiation quasi-ordering*. It induces an equivalence relation (both on terms and on substitutions), denoted by $\sim$.

Definition 9 (Unification problem, unifier)**.** A *unification problem* Γ is a finite set of unification equations (term pairs):

$$\Gamma = \{t_1 \approx^?_\alpha u_1, \dots, t_n \approx^?_\alpha u_n\}.$$

Γ does not contain two different unknowns with the same variable: If v^{P_1} and v^{P_2} occur in Γ, then $P_1 = P_2$. For each v^P occurring in Γ, the atoms in P are free in the equation where v^P occurs.

A substitution σ is a *unifier* of Γ if $t_i\sigma \approx_\alpha u_i\sigma$ for all $1 \le i \le n$. It is called a most general unifier, if $\sigma \precsim \vartheta$ for any unifier ϑ of Γ.

It is known [42, 44] that when unification problems contains sequence variables, there might be infinitely many unifiers, which are not comparable with each other by $\precsim$. (In other words, the problem is infinitary.) In such cases, one talks about minimal complete sets of unifiers:

Definition 10 (Minimal complete set of unifiers)**.** Let Γ be a unification problem and S be a set of substitutions. Then S is called a *complete set of unifiers* of Γ, if the following two properties are satisfied:

Soundness: Every $\sigma \in S$ is a unifier of Γ.

Completeness: For each unifier ϑ of Γ, there exists $\sigma \in \Gamma$ such that $\sigma \precsim \vartheta$.

S is a *minimal complete set of unifiers* of Γ, if, in addition, the minimality property holds:

Minimality: If there exist $\sigma_1, \sigma_2 \in S$ such that $\sigma_1 \precsim \sigma_2$, then $\sigma_1 = \sigma_2$.

We denote S in this case by $\mathsf{mcsu}(\Gamma)$.

A simple example of a unification problem with infinite minimal complete set of unifiers is $\Gamma = \{f(\overline{x}^{\{a\}}, a) \approx_\alpha f(a, \overline{x}^{\{a\}})\}$. We have $\mathsf{mcsu}(\Gamma) = \{\{\overline{x}^{\{a\}} \mapsto ()\}, \{\overline{x}^{\{a\}} \mapsto (a)\}, \{\overline{x}^{\{a\}} \mapsto (a, a)\}, \{\overline{x} \mapsto (a, a, a)\}, \dots\}$.

Example 2. Here we show some unification problems Γ and their minimal complete sets of unifiers. ($\forall, \lambda \in \mathcal{Q}$, $p, >, \langle\cdot\rangle \in \mathcal{F}$):

$$\Gamma = \{\forall a.\forall b.\langle a > b, p(a, b)\rangle \approx^?_\alpha \forall b.\forall a.\langle b > a, p(b, a)\rangle\},$$
$$\mathsf{mcsu}(\Gamma) = \{Id\}.$$

$$\Gamma = \{\forall a.p(a, x^{\{b\}}) \approx^?_\alpha \forall b.p(b, b)\},$$
$$\mathsf{mcsu}(\Gamma) = \emptyset : \text{not unifiable.}$$

$$\Gamma = \{\forall a.p(a, x^{\{b\}}) \approx^?_\alpha \forall c.p(c, b)\},$$
$$\mathsf{mcsu}(\Gamma) = \{\{x^{\{b\}} \mapsto b\}\}.$$

$$\Gamma = \{\forall a.p(a, x^{\{b\}}) \approx^?_\alpha \forall b.p(b, c)\},$$
$$\mathsf{mcsu}(\Gamma) = \emptyset : \text{not unifiable.}$$

$$\Gamma = \{\forall a.p(a, x^{\emptyset}) \approx^?_\alpha \forall a.p(a, \lambda b.b)\},$$

$\mathsf{mcsu}(\Gamma) = \{\{x^{\emptyset} \mapsto \lambda b.b\}\}$.

$\Gamma = \{\forall a.p(a, x^{\emptyset}) \approx^?_\alpha \forall a.p(a, b)\}$,
$\mathsf{mcsu}(\Gamma) = \emptyset$: not unifiable.

$\Gamma = \{\forall a.p(a, x^{\{b\}}) \approx^?_\alpha \forall c.p(c, f(b, g))\}$,
$\mathsf{mcsu}(\Gamma) = \{\{x^{\{b\}} \mapsto f(b, g)\}\}$.

$$\begin{aligned}
\Gamma &= \{\forall a.p(a, \overline{x}^{\{b\}}, \overline{y}^{\{b,c,d\}}) \approx^?_\alpha \forall a.p(a, b, f(b), c)\},\\
\mathsf{mcsu}(\Gamma) &= \{\{\overline{x}^{\{b\}} \mapsto (),\ \overline{y}^{\{b,c,d\}} \mapsto (b, f(b), c)\},\\
&\qquad \{\overline{x}^{\{b\}} \mapsto (b),\ \overline{y}^{\{b,c,d\}} \mapsto (f(b), c)\},\\
&\qquad \{\overline{x}^{\{b\}} \mapsto (b, f(b)),\ \overline{y}^{\{b,c,d\}} \mapsto (c)\}\}.
\end{aligned}$$

$\Gamma = \{p(\overline{x}^{\{a,b\}}) \approx^?_\alpha p(\overline{y}^{\{a\}}, \overline{z}^{\{a,b,c\}})\}$,
$\mathsf{mcsu}(\Gamma) = \{\{\overline{x}^{\{a,b\}} \mapsto (\overline{y}^{\{a\}}, \overline{z}'^{\{a,b\}}), \overline{z}^{\{a,b,c\}} \mapsto \overline{z}'^{\{a,b\}}\}\}$.

$\Gamma = \{\forall a.p(a, x^{\{c,c'\}}) \approx^?_\alpha \forall b.p(b, f(y^{\{c,c''\}}))\}$,
$\mathsf{mcsu}(\Gamma) = \{\{x^{\{c,c'\}} \mapsto f(y'^{\{c\}}), y^{\{c,c''\}} \mapsto y'^{\{c\}}\}\}$.

$\Gamma = \{x^{\{a,b\}}(y^{\{a,c\}}) \approx^?_\alpha f(y^{\{a,c\}})(g(z^{\{c\}}, a))\}$,
$\mathsf{mcsu}(\Gamma) = \{\{x^{\{a,b\}} \mapsto f(g(z'^{\emptyset}, a)), y^{\{a,c\}} \mapsto g(z'^{\emptyset}, a), z^{\{c\}} \mapsto z^{\emptyset}\}\}$.

$$\begin{aligned}
\Gamma &= \{p(\overline{x}^{\{a,b\}}, \overline{y}^{\{a,c\}}) \approx^?_\alpha p(f(\overline{y}^{\{a,c\}}), g(z^{\{b\}}, a))\},\\
\mathsf{mcsu}(\Gamma) &= \{\{\overline{x}^{\{a,b\}} \mapsto (f(g(z'^{\emptyset}, a))), \overline{y}^{\{a,c\}} \mapsto (g(z'^{\emptyset}, a)), z^{\{b\}} \mapsto z^{\emptyset}\},\\
&\qquad \{\overline{x}^{\{a,b\}} \mapsto (f(), g(z^{\{b\}}, a)),\ \overline{y}^{\{a,c\}} \mapsto ()\}\}.
\end{aligned}$$

3 The algorithm

In this section we formulate our unification algorithm in a rule-based way. Rules operate on states, which is a pair $\Gamma; \sigma$, where Γ is a unification problem and σ is a substitution. Intuitively, a state shows the problem "still to be solved" and the unifier "computed so far".

In the rules we use renaming substitutions, defined as follows: A substitution σ is called a *renaming substitution* if it injectively maps term unknowns to term unknowns and sequence unknowns to sequence unknowns.

The rules are the following (the symbol symb is use as a metavariable for a function symbol, atom, or a term unknown):

T: **Trivial**

$\{t \approx^?_\alpha t\} \uplus \Gamma;\ \sigma \rightsquigarrow \Gamma;\ \sigma.$

HD: **Head Decomposition**

$\{t(\tilde{s}) \approx^?_\alpha u(\tilde{r})\} \uplus \Gamma;\ \sigma \rightsquigarrow \{t \approx^?_\alpha u\} \cup \Gamma' \cup \Gamma;\ \sigma,$

if $t \notin \mathcal{F} \cup \mathcal{A}$ or $u \notin \mathcal{F} \cup \mathcal{A}$. If $\tilde{s} = \tilde{r} = ()$, then $\Gamma' = \emptyset$, otherwise $\Gamma' = \{f(\tilde{s}) \approx^?_\alpha f(\tilde{r})\}$, where f is an arbitrary function symbol.

TD: **Total Decomposition**

$$\{\mathsf{symb}(t_1, \ldots, t_n) \approx^?_\alpha \mathsf{symb}(u_1, \ldots, u_n)\} \uplus \Gamma;\ \sigma \rightsquigarrow \{t_1 \approx^?_\alpha u_1, \ldots, t_n \approx^?_\alpha u_n\} \cup \Gamma;\ \sigma,$$

where $n > 0$.

PD-L: **Partial Decomposition Left**

$$\{\mathsf{symb}(t_1, \ldots, t_n, \overline{x}^P, \tilde{s}) \approx^?_\alpha \mathsf{symb}(u_1, \ldots, u_n, \tilde{r})\} \uplus \Gamma;\ \sigma \rightsquigarrow \\ \{t_1 \approx^?_\alpha u_1, \ldots, t_n \approx^?_\alpha u_n, \mathsf{symb}(\overline{x}^P, \tilde{s}) \approx^?_\alpha \mathsf{symb}(\tilde{r})\} \cup \Gamma;\ \sigma.$$

where $n > 0$.

Q: **Quantifiers**

$$\{Qa.t \approx^?_\alpha Qb.u\} \uplus \Gamma;\ \sigma \rightsquigarrow \{t[a := c] \approx^?_\alpha u[b := c]\} \cup \Gamma;\ \sigma.$$

where $c \notin \mathsf{atoms}(t) \cup \mathsf{atoms}(u)$.

TUE-L: **Term Unknown Elimination Left**

$$\{x^P \approx^?_\alpha u\} \uplus \Gamma;\ \sigma \rightsquigarrow \Gamma\vartheta\rho;\ \sigma\vartheta\rho,$$

where

- $x^P \notin \mathsf{unkn}(u) = \{v_1^{P_1}, \ldots, v_n^{P_n}\}$,
- $\mathsf{fa}(u) \setminus (P_1 \cup \cdots \cup P_n) \subseteq P$,
- $\rho = \{v_i^{P_i} \mapsto w_i^{P_i \cap P} \mid i \in \{1, \ldots, n\}, P_i \cap P \neq P_i\}$ is a renaming substitution with fresh variables w_i, and
- $\vartheta = \{x^P \mapsto u\rho\}$.

The rule below depends on the global parameter ℓ which specifies the maximum length of instantiations of sequence unknowns.[2] It affects completeness, but is necessary for termination.

FIXED-SUE-L: **Fixed-Size Sequence Unknown Elimination Left**

$$\{\mathsf{symb}(\overline{x}^P, \tilde{s}) \approx^?_\alpha \mathsf{symb}(\tilde{r})\} \uplus \Gamma;\ \sigma \rightsquigarrow (\{\mathsf{symb}(\overline{x}^P, \tilde{s}) \approx^?_\alpha \mathsf{symb}(\tilde{r})\} \cup \Gamma)\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x}^P \mapsto (x_1^P, \ldots, x_k^P)\}$, where the x's are fresh variables and $k \leq \ell$.

We also have the **Right** counterparts of the **left** rules. We do not explicitly write them here to save space. They are just dual to the corresponding **Left** rules: If a **Left** rule operates on $t \approx^?_\alpha u$, the right rule would apply to an equation of the form $u \approx^?_\alpha t$. The names of **Right** rules have the suffix -R is place of -L.

To unify two terms t and u, we create the initial state $\{t \approx^?_\alpha u\}; Id$ and apply the above-mentioned rules exhaustively, generating derivations. When the Trivial rule T applies to the

[2]Instead of the global parameter ℓ, we could impose individual length-bounds for each sequence unknown occurring in the given unification problem. It would not change the algorithm and its properties.

selected equation, the other rules are not used. If an elimination rule and its right counterpart (i.e., TUE-L and TUE-R, FIXED-SUE-L and FIXED-SUE-R) are applicable to the same equation at the same time, we use only one of them (usually the left one).

FIXED-SUE-L (and FIXED-SUE-R) can transform the same equation in finitely many ways, depending on the choice of k. It can cause branching in the derivation tree, leading to computing multiple answers.

The derivations stop in two cases. Either a state of the form $\emptyset; \sigma$ is generated, or no rule can be applied to the last state $\Gamma; \vartheta$ where $\Gamma \neq \emptyset$. In the first case, the derivation is called successful and $\sigma|_{\mathsf{unkn}(\Gamma)}$ is called the *computed answer*. In the second case, the derivation is called failed.

The described algorithm is denoted by Unif-Alg. The set of answers computed by Unif-Alg for a unification problem Γ with a given ℓ is denoted by $\mathsf{Unif\text{-}Alg}(\Gamma, \ell)$.

There are special fragments of terminating sequence unification (see, e.g., [46]). We can accommodate them in our framework as well, replacing FIXED-SUE-L by rules suitable to the particular fragment. Here we consider two such special cases: (1) when no unknown occurs in the right hand side of an unification problem (sequence matching fragment, SEQ-MATCH) and (2) when all sequence unknowns occur in the last argument positions (sequence last fragment, SEQ-LAST).

For SEQ-MATCH, the rule that replaces FIXED-SUE-L is MATCH-SUE.

MATCH-SUE: **Sequence Unknown Elimination,** SEQ-MATCH

$$\{\mathsf{symb}(\overline{x}^P, \tilde{s}) \approx^?_\alpha \mathsf{symb}(\tilde{r}_1, \tilde{r}_2)\} \uplus \Gamma;\ \sigma \rightsquigarrow$$
$$(\{\mathsf{symb}(\tilde{s}) \approx^?_\alpha \mathsf{symb}(\tilde{r}_2)\} \cup \Gamma)\vartheta;\ \sigma\vartheta,$$

where $\mathsf{fa}(\tilde{r}_1) \subseteq P$ and $\vartheta = \{\overline{x}^P \mapsto \tilde{r}_1\}$.

We do not need the **Right** counterpart of this rule and also for the other elimination rules, since in the matching fragment no unknown occurs in the right hand side.

For SEQ-LAST, FIXED-SUE-L is replaced by LAST-SUE-L.

LAST-SUE-L: **Sequence Unknown Elimination Left,** SEQ-LAST

$$\{\mathsf{symb}(\overline{x}^P) \approx^?_\alpha \mathsf{symb}(\tilde{r})\} \uplus \Gamma;\ \sigma \rightsquigarrow \Gamma\vartheta\rho;\ \sigma\vartheta\rho,$$

where

- $\overline{x}^P \notin \mathsf{unkn}(\tilde{r}) = \{v_1^{P_1}, \dots, v_n^{P_n}\}$,
- $\mathsf{fa}(\tilde{r}) \setminus (P_1 \cup \dots \cup P_n) \subseteq P$,
- $\rho = \{v_i^{P_i} \mapsto w_i^{P_i \cap P} \mid i \in \{1, \dots, n\}, P_i \cap P \neq P_i\}$ is a renaming substitution with fresh variables w_i, and
- $\vartheta = \{\overline{x} \mapsto \tilde{r}\rho\}$.

We have also the **Right** counterpart of this rule, called LAST-SUE-R. If both LAST-SUE-L and LAST-SUE-R are applicable to the same equation, we apply only LAST-SUE-L.

For SEQ-MATCH and SEQ-LAST fragments, there is no global parameter ℓ anymore. Derivations are performed as defined above. The MATCH-SUE rule causes branching, depending on the choice of $\tilde{r}_1$, and leads to finitely many answers. LAST-SUE-L and LAST-SUE-R do not introduce branching. We denote by Match-Alg and Unif-Alg-Last the corresponding algorithms, and by $\mathsf{Match\text{-}Alg}(\Gamma)$ and $\mathsf{Unif\text{-}Alg\text{-}Last}(\Gamma)$ the sets of answers computed by them for Γ.

Example 3. Let $\Gamma = \{f(x^{\{a,b\}}, \lambda b.y^{\{a,c\}}(b)) \approx^?_\alpha f(f(y^{\{a,c\}}), \lambda d.g(z^{\{c\}}, a)(d))\}$. Then $\mathsf{Unif\text{-}Alg}$ generates the following derivation:

$$\{f(x^{\{a,b\}}, \lambda b.y^{\{a,c\}}(b)) \approx^?_\alpha f(f(y^{\{a,c\}}), \lambda d.g(z^{\{c\}}, a)(d)); \mathit{Id} \rightsquigarrow_{\mathsf{TD}}$$
$$\{x^{\{a,b\}} \approx^?_\alpha f(y^{\{a,c\}}),\ \lambda b.y^{\{a,c\}}(b) \approx^?_\alpha \lambda d.g(z^{\{c\}}, a)(d)\}; \mathit{Id} \rightsquigarrow_{\mathsf{TUE\text{-}L}}$$
$$\{\lambda b.y_1^{\{a\}}(b) \approx^?_\alpha \lambda d.g(z^{\{c\}}, a)(d)\}; \{x^{\{a,b\}} \mapsto f(y_1^{\{a\}}), y^{\{a,c\}} \mapsto y_1^{\{a\}}\} \rightsquigarrow_{\mathsf{Q}}$$
$$\{y_1^{\{a\}}(d') \approx^?_\alpha g(z^{\{c\}}, a)(d')\}; \{x^{\{a,b\}} \mapsto f(y_1^{\{a\}}), y^{\{a,c\}} \mapsto y_1^{\{a\}}\} \rightsquigarrow_{\mathsf{HD}}$$
$$\{y_1^{\{a\}} \approx^?_\alpha g(z^{\{c\}}, a),\ d' \approx^?_\alpha d'\}; \{x^{\{a,b\}} \mapsto f(y_1^{\{a\}}), y^{\{a,c\}} \mapsto y_1^{\{a\}}\} \rightsquigarrow_{\mathsf{T}}$$
$$\{y_1^{\{a\}} \approx^?_\alpha g(z^{\{c\}}, a)\}; \{x^{\{a,b\}} \mapsto f(y_1^{\{a\}}), y^{\{a,c\}} \mapsto y_1^{\{a\}}\} \rightsquigarrow_{\mathsf{TUE\text{-}L}}$$
$$\emptyset;\ \{x^{\{a,b\}} \mapsto f(g(z_1^{\emptyset}, a)),\ y^{\{a,c\}} \mapsto g(z_1^{\emptyset}, a),\ y_1^{\{a\}} \mapsto g(z_1^{\emptyset}, a),\ z^{\{c\}} \mapsto z_1^{\emptyset}\}.$$

The computed answer is $\{x^{\{a,b\}} \mapsto f(g(z_1^{\emptyset}, a)),\ y^{\{a,c\}} \mapsto g(z_1^{\emptyset}, a),\ z^{\{c\}} \mapsto z_1^{\emptyset}\}$.

Example 4. Let $\Gamma = \{f(\overline{x}^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, \overline{x}^{\{a,b\}})\}$. Let $\ell = 2$. Then $\mathsf{Unif\text{-}Alg}$ generates the following derivations:

1. $\{f(\overline{x}^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, \overline{x}^{\{a,b\}})\};\ \mathit{Id} \rightsquigarrow_{\mathsf{FIXED\text{-}SUE\text{-}L},\,\mathsf{k=0}}$
   $\{f(a, b) \approx^?_\alpha f(a, b)\};\ \{\overline{x}^{\{a,b\}} \mapsto ()\} \rightsquigarrow_{\mathsf{T}}$
   $\emptyset;\ \{\overline{x}^{\{a,b\}} \mapsto ()\}$

2. $\{f(\overline{x}^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, \overline{x}^{\{a,b\}})\};\ \mathit{Id} \rightsquigarrow_{\mathsf{FIXED\text{-}SUE\text{-}L},\,\mathsf{k=1}}$
   $\{f(x_1^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, x_1^{\{a,b\}})\};\ \{\overline{x}^{\{a,b\}} \mapsto (x_1^{\{a,b\}})\} \rightsquigarrow_{\mathsf{TD}}$
   $\{x_1^{\{a,b\}} \approx^?_\alpha a,\ a \approx^?_\alpha b,\ b \approx^?_\alpha x_1^{\{a,b\}}\};\ \{\overline{x}^{\{a,b\}} \mapsto (a)\} \rightsquigarrow_{\mathsf{TUE\text{-}L}}$
   $\{a \approx^?_\alpha b,\ b \approx^?_\alpha a\};\ \{\overline{x}^{\{a,b\}} \mapsto (a),\ x_1^{\{a,b\}} \approx^?_\alpha a\}$
   FAIL

3. $\{f(\overline{x}^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, \overline{x}^{\{a,b\}})\};\ \mathit{Id} \rightsquigarrow_{\mathsf{FIXED\text{-}SUE\text{-}L},\,\mathsf{k=2}}$
   $\{f(x_1^{\{a,b\}}, x_2^{\{a,b\}}, a, b) \approx^?_\alpha f(a, b, x_1^{\{a,b\}}, x_2^{\{a,b\}})\};$
   $\quad\{\overline{x}^{\{a,b\}} \mapsto (x_1^{\{a,b\}}, x_2^{\{a,b\}})\} \rightsquigarrow_{\mathsf{TD}}$
   $\{x_1^{\{a,b\}} \approx^?_\alpha a,\ x_2^{\{a,b\}} \approx^?_\alpha b,\ a \approx^?_\alpha x_1^{\{a,b\}},\ b \approx^?_\alpha x_2^{\{a,b\}}\};$
   $\quad\{\overline{x}^{\{a,b\}} \mapsto (x_1^{\{a,b\}}, x_2^{\{a,b\}})\} \rightsquigarrow_{\mathsf{TUE\text{-}L}}$
   $\{x_2^{\{a,b\}} \approx^?_\alpha b,\ a \approx^?_\alpha a,\ b \approx^?_\alpha x_2^{\{a,b\}}\};$
   $\quad\{\overline{x}^{\{a,b\}} \mapsto (a, x_2^{\{a,b\}}),\ x_1^{\{a,b\}} \mapsto a\} \rightsquigarrow_{\mathsf{TUE\text{-}L}}$
   $\{a \approx^?_\alpha a,\ b \approx^?_\alpha b\};$
   $\quad\{\overline{x}^{\{a,b\}} \mapsto (a, b),\ x_1^{\{a,b\}} \mapsto a,\ x_2^{\{a,b\}} \mapsto b\} \rightsquigarrow^2_{\mathsf{T}}$
   $\emptyset;\ \{\overline{x}^{\{a,b\}} \mapsto (a, b),\ x_1^{\{a,b\}} \mapsto a,\ x_2^{\{a,b\}} \mapsto b\}.$

Hence, $\mathsf{Unif\text{-}Alg}(\Gamma, 2) = \{\{\overline{x}^{\{a,b\}} \mapsto ()\}, \{\overline{x}^{\{a,b\}} \mapsto (a, b)\}\}$.

Example 5. Let $\Gamma = \{f(\overline{x}^{\{a\}}, \overline{y}^{\{a,b,c\}}) \approx_\alpha^? f(a,b,c)\}$. It is a matching problem and we can apply $\mathsf{Match\text{-}Alg}$, which generates the following derivations:

1. $\{f(\overline{x}^{\{a\}}, \overline{y}^{\{a,b,c\}}) \approx_\alpha^? f(a,b,c)\};\ Id \rightsquigarrow_{\mathsf{MATCH\text{-}SUE}}$
   $\{f(\overline{y}^{\{a,b,c\}}) \approx_\alpha^? f(a,b,c)\};\ \{\overline{x}^{\{a\}} \mapsto ()\} \rightsquigarrow_{\mathsf{MATCH\text{-}SUE}}$
   $\{f() \approx_\alpha^? f()\};\ \{\overline{x}^{\{a\}} \mapsto (), \overline{y}^{\{a,b,c\}} \mapsto (a,b,c)\} \rightsquigarrow_{\mathsf{T}}$
   $\emptyset;\ \{\overline{x}^{\{a\}} \mapsto (), \overline{y}^{\{a,b,c\}} \mapsto (a,b,c)\}$.

2. $\{f(\overline{x}^{\{a\}}, \overline{y}^{\{a,b,c\}}) \approx_\alpha^? f(a,b,c)\};\ Id \rightsquigarrow_{\mathsf{MATCH\text{-}SUE}}$
   $\{f(\overline{y}^{\{a,b,c\}}) \approx_\alpha^? f(b,c)\};\ \{\overline{x}^{\{a\}} \mapsto (a)\} \rightsquigarrow_{\mathsf{MATCH\text{-}SUE}}$
   $\{f() \approx_\alpha^? f()\};\ \{\overline{x}^{\{a\}} \mapsto (a), \overline{y}^{\{a,b,c\}} \mapsto (b,c)\} \rightsquigarrow_{\mathsf{T}}$
   $\emptyset;\ \{\overline{x}^{\{a\}} \mapsto (a), \overline{y}^{\{a,b,c\}} \mapsto (b,c)\}$.

Hence, $\mathsf{Match\text{-}Alg}(\Gamma) = \{\{\overline{x}^{\{a\}} \mapsto (), \overline{y}^{\{a,b,c\}} \mapsto (a,b,c)\}, \{\overline{x}^{\{a\}} \mapsto (a), \overline{y}^{\{a,b,c\}} \mapsto (b,c)\}\}$.

If we apply $\mathsf{Unif\text{-}Alg}$ with $\ell = 1$, there will be no answer computed. All the derivation branches will fail. For any $\ell > 1$ we get the same answers as those computed by $\mathsf{Match\text{-}Alg}$.

Example 6. Let $\Gamma = \{f(\overline{x}^{\{a\}}) \approx_\alpha^? f(\overline{y}^{\{a,b,c\}})\}$. Application of $\mathsf{Unif\text{-}Alg}$ with $\ell = 2$ gives three computed answers:

$$\{\overline{x}^{\{a\}} \mapsto (),\ \overline{y}^{\{a,b,c\}} \mapsto ()\},\ \{\overline{x}^{\{a\}} \mapsto (z^{\{a\}}),\ \overline{y}^{\{a,b,c\}} \mapsto (z^{\{a\}})\},$$
$$\{\overline{x}^{\{a\}} \mapsto (z_1^{\{a\}}, z_2^{\{a\}}),\ \overline{y}^{\{a,b,c\}} \mapsto (z_1^{\{a\}}, z_2^{\{a\}})\}.$$

The problem also falls in the SEQ-LAST fragment. $\mathsf{Unif\text{-}Alg\text{-}Last}$ gives only one computed answer: $\{\overline{x}^{\{a\}} \mapsto (\overline{y}_1^{\{a\}}),\ \overline{y}^{\{a,b,c\}} \mapsto (\overline{y}_1^{\{a\}})\}$.

4 Properties of the algorithm

First, we define the sizes of an s-term, an equation, and a unification problem:

$$\mathsf{size}(f) = \mathsf{size}(a) = 1$$
$$\mathsf{size}(x^P) = \mathsf{size}(\overline{x}^P) = 2.$$
$$\mathsf{size}(t(s_1,\dots,s_n)) = \mathsf{size}(t) + \mathsf{size}((s_1,\dots,s_n)) + 1.$$
$$\mathsf{size}(Qa.t) = size(t) + 1.$$
$$\mathsf{size}(()) = 0.$$
$$\mathsf{size}((s_1,\dots,s_n)) = \mathsf{size}(s_1) + \cdots + \mathsf{size}(s_n).$$
$$\mathsf{size}(t \approx_\alpha^? u) = \mathsf{size}(t) + \mathsf{size}(u).$$
$$\mathsf{size}(\Gamma) = \{\!\{\mathsf{size}(t \approx_\alpha^? u) \mid t \approx_\alpha^? u \in \Gamma\}\!\},\ \text{where } \{\!\{\cdot\}\!\} \text{ stand for multiset.}$$

Theorem 1. *$\mathsf{Unif\text{-}Alg}$, $\mathsf{Match\text{-}Alg}$, and $\mathsf{Unif\text{-}Alg\text{-}Last}$ terminate.*

Proof. With each state $\Gamma; \sigma$, we associate its complexity measure, a triple $\langle n_1, n_2, M\rangle$, where n_1 and n_2 are respectively the numbers of distinct term and sequence unknowns occurring in

Γ, and $M = \mathsf{size}(\Gamma)$. The measures are compared lexicographically, where the first two components are compared by the standard ordering on natural numbers, and the third component is compared by the multiset extension of the standard natural number ordering. The obtained ordering on complexity measures is well-founded. The table below shows that each rule of our algorithms strictly reduces this measure (i.e., if a rule transforms $\Gamma_1; \sigma_1$ into $\Gamma_2; \sigma_2$, then the measure of Γ_2 is strictly smaller than the measure of Γ_1), which implies that Unif-Alg, Match-Alg and Unif-Alg-Last terminate.

Rules	n_1	n_2	M
FIXED-SUE-L, FIXED-SUE-R	$>$		
LAST-SUE-L, LAST-SUE-R	$>$		
MATCH-SUE	$>$		
TUE-L, TUE-R	$=$	$>$	
T	$\geq$	$\geq$	$>$
TD	$=$	$\geq$	$>$
HD, PD-L, PD-R, Q	$=$	$=$	$>$

□

For the other properties of our algorithms, we need the following lemma:

Lemma 1. *If $\Gamma_1; \sigma \rightsquigarrow \Gamma_2; \sigma\psi$ is a rule application, then $\Gamma_1\psi$ and Γ_2 have the same sets of unifiers.*

Proof. Assume the derivation step is made by the TUE-L rule. Then $\psi = \rho\vartheta$, $\Gamma_1 = \{x^P \approx^?_\alpha u\} \uplus \Gamma$, and $\Gamma_2 = \Gamma\rho\vartheta$, where ρ and ϑ are as defined by the rule. We have $x^P\rho\vartheta = u\rho$, $u\rho\vartheta = u\rho$ and, hence, $\Gamma_1\rho\vartheta = \{u\rho \approx^?_\alpha u\rho\} \cup \Gamma\rho\vartheta$. Obviously, $\Gamma_1\rho\vartheta$ and $\Gamma\rho\vartheta$ have the same set of unifiers i.e., $\Gamma_1\psi$ and Γ_2 have the same set of unifiers.

The proof is analogous for the other elimination rules. For trivial, decomposition, and quantifier rules the theorem follows directly from the definition of α-equivalence. □

Theorem 2 (Soundness of Unif-Alg)**.** *For a unification problem Γ and a length bound ℓ, every substitution $\sigma \in \mathsf{comp}(\mathsf{Unif\text{-}Alg}, \Gamma, \ell)$ is a unifier of Γ.*

Proof. Since $\sigma \in \mathsf{comp}(\mathsf{Unif\text{-}Alg}, \Gamma, \ell)$, there exists a derivation in Unif-Alg (with ℓ) of the form $\Gamma; \mathit{Id} \rightsquigarrow^+ \emptyset; \sigma$. Then the theorem can be proved by using the induction on the length of the derivation and Lemma 1. □

The Match-Alg and Unif-Alg-Last algorithms are sound as well. The corresponding theorems below can be proved similarly to Theorem 2.

Theorem 3 (Soundness of Match-Alg)**.** *For a matching problem Γ, every $\sigma \in \mathsf{Match\text{-}Alg}(\Gamma)$ is a matcher of Γ.*

Theorem 4 (Soundness of Unif-Alg-Last)**.** *If Γ is a unification problem where every sequence unknown appears in the last argument position, and $\mathsf{Unif\text{-}Alg\text{-}Last}(\Gamma) = \{\sigma\}$, then σ is a unifier of Γ.*

Unif-Alg is not complete, in general. It is obvious, since the length restriction on the instantiation of sequence unknowns, imposed by the parameter ℓ, prevents to compute unifiers in which the lengths of sequence unknown instances are larger than ℓ. For example, when $\ell =$

2, we can not compute the unifier $\{\overline{x}^{\{a\}} \mapsto (a,a,a)\}$ of the unification problem $f(\overline{x}^{\{a\}}, a) \approx_\alpha^? f(a, \overline{x}^{\{a\}})$.

Interestingly, there is another reason of incompleteness of $\mathsf{Unif\text{-}Alg}$, which is caused by the fact that a sequence unknown is always replaced by a sequence of term unknowns. Because of this, $\mathsf{Unif\text{-}Alg}$ can not compute a most general solution $\{\overline{x}^P \mapsto (\overline{y}^P)\}$ of $\Gamma = \{f(\overline{x}^P) \approx_\alpha^? f(\overline{y}^P)\}$. Instead, it returns ℓ solutions $\{\overline{x}^P \mapsto (), \overline{x}^P \mapsto ()\}$, $\{\overline{x}^P \mapsto (x^P), \overline{y}^P \mapsto (x^P)\}$, $\ldots$, $\{\overline{x}^P \mapsto (x_1^P, \ldots, x_\ell^P), \overline{y}^P \mapsto (x_1^P, \ldots, x_\ell^P)\}$.

However, the following restricted version of completeness holds:

Theorem 5 (Restricted completeness of $\mathsf{Unif\text{-}Alg}$)**.** *Let Γ be a unification problem and φ be its unifier such that $ran(\varphi)$ does not contain sequence unknowns. Then there exist ℓ and $\sigma \in \mathsf{comp}(\mathsf{Unif\text{-}Alg}, \ell)$ such that $\sigma|_{\mathsf{unkn}(\Gamma)} \precsim \varphi$.*

Proof. First, consider the case when Γ does not contain sequence unknowns. Assume without loss of generality that φ is idempotent and $dom(\varphi) \subseteq \mathsf{unkn}(\Gamma)$.

We will construct a derivation $\Gamma_1; \sigma_1 \rightsquigarrow^+ \Gamma_n; \sigma_n$, where $\Gamma_1 = \Gamma$, $\sigma_1 = Id$, $\Gamma_n = \emptyset$, and for each $1 \le i \le n$, there exists a substitution ψ_i such that

- $\varphi\psi_i$ is an idempotent unifier of Γ_i,
- $dom(\psi_i|_{\mathsf{unkn}(\Gamma)}) \subseteq dom(\varphi)$,
- $\sigma_i \precsim \varphi\psi_i$.

(For $i = 1$ such a ψ_i obviously exists: it is Id.)

If we build such a derivation, we get $\sigma_n \precsim \varphi\psi_n$, which implies $\sigma_n|_{\mathsf{unkn}(\Gamma)} \precsim (\varphi\psi_n)|_{\mathsf{unkn}(\Gamma)} = \varphi$ and we can take $\sigma = \sigma_n$.

Assume we have constructed $\Gamma_1; \sigma_1 \rightsquigarrow^* \Gamma_i; \sigma_i$ in this derivation and show how to make the step $\Gamma_i; \sigma_i \rightsquigarrow \Gamma_{i+1}; \sigma_{i+1}$.

We pick up an equation arbitrarily from Γ_i, represent the unification problem as $\Gamma_i = \{t \approx_\alpha^? u\} \uplus \Gamma_i'$, and proceed by case distinction on the form of $t \approx_\alpha^? u$.

If $t = u$, then the step is made by the T rule and $\Gamma_{i+1}; \sigma_{i+1}$ obviously satisfies all the desired properties. Assume $t \neq u$. We distinguish the following cases:

$\mathsf{head}(t) = \mathsf{head}(u)$. The applicable rules are TD or Q. In each case, it is easy to see that the obtained state is what we need.

$\mathsf{head}(t) \neq \mathsf{head}(u)$ and none of these terms is an unknown. Then the only possible case is $\mathsf{head}(t) \notin \mathcal{F} \cup \mathcal{A}$, or $\mathsf{head}(u) \notin \mathcal{F} \cup \mathcal{A}$. Otherwise Γ_i would not be unifiable. We apply the HD rule. Again, the obtained state satisfies the desired properties.

$\mathsf{head}(t) \neq \mathsf{head}(u)$ and at least one of them is an unknown x^P. Assume without loss of generality that it is t. hence, we have an equation $x^P \approx_\alpha^? u$. If $x^P \in \mathsf{unkn}(u)$, then for any substitution ϑ we will have $\mathsf{size}(x^P\vartheta) < \mathsf{size}(u\vartheta)$ and Γ_i would not be unifiable.

Assume $x^P \notin \mathsf{unkn}(u)$. Then we apply $\mathsf{TUE\text{-}L}$ rule and get $\Gamma_{i+1} = \Gamma'\vartheta_{i+1}\rho_{i+1}$, $\sigma_{i+1} = \sigma_i\vartheta_{i+1}\rho_{i+1}$, where

$$\rho_{i+1} = \{v^R \mapsto w^{P \cap R} \mid v^R \in \mathsf{unkn}(u), P \cap R \neq R, w \text{ is fresh}\},$$
$$\vartheta_{i+1} = \{x^P \mapsto u\rho_{i+1}\}.$$

We need to find ψ_{i+1} such that

- $\sigma_{i+1} = \sigma_i\vartheta_{i+1}\rho_{i+1} \precsim \varphi\psi_{i+1}$,

- $dom(\psi_{i+1}|_{\mathsf{unkn}(\Gamma)}) \subseteq dom(\varphi)$, and
- $\varphi\psi_{i+1}$ is an idempotent unifier of Γ_{i+1}.

Since $\sigma_i \precsim \varphi\psi_i$, there exists a substitution ν such that $v^R\sigma_i\nu \approx_\alpha v^R\varphi\psi_i$ for all v^R. On the other hand, $\varphi\psi_i$ is a unifier of $x^P \approx^?_\alpha u$. This gives $x^P\nu = x^P\sigma_i\nu \approx_\alpha u\sigma_i\nu = u\nu$. (The σ's are idempotent, therefore, x^P and unknowns from u are not in the domain of σ_i.)

Besides, we have

$$v^R\vartheta_{i+1}\rho_{i+1}\nu\mu \approx_\alpha v^R\nu\mu \text{ for any } v^R. \tag{1}$$

Define μ as

$$\mu = \{v^R\rho_{i+1} \mapsto v^R\nu \mid v^R \in \mathsf{unkn}(u)\}.$$

It is a well-defined substitution: $v^R \in \mathsf{unkn}(u)$ and we have $\mathsf{fa}(v^R\nu) \subseteq R$, since ν is a unifier of $x^P \approx^?_\alpha u$.

Let $\psi_{i+1} = \psi_i\mu$. Then we have $dom(\psi_{i+1}|_{\mathsf{unkn}(\Gamma)}) \subseteq dom(\varphi)$. Since $\varphi\psi_i$ is an idempotent unifier of Γ_i, we get that $\varphi\psi_{i+1}$ is an idempotent unifier of Γ_{i+1}.

Note that ρ_{i+1}, ϑ_{i+1}, μ and ψ_{i+1} are idempotent. Besides, $v^R\rho_{i+1}\mu = v^R\varphi\psi_i\mu$ for every $v^R \in dom(\rho_{i+1}) \cup ran(\rho_{i+1})$. Therefore, we get

- $v^R\rho_{i+1}\nu\mu = v^R\nu\nu\mu = v^R\nu\mu = v^R\varphi\psi_i\mu = v^R\varphi\psi_{i+1}$ for every $v^R \in dom(\rho_{i+1}) \cup ran(\rho_{i+1})$,
- $v^R\rho_{i+1}\nu\mu = v^R\nu\mu = v^R\varphi\psi_i\mu = v^R\varphi\psi_{i+1}$ for every other v^R.

In order to show $\sigma_{i+1} \precsim \varphi\psi_{i+1}$, we will prove $w^T\sigma_{i+1}\nu\mu \approx_\alpha w^T\varphi\psi_{i+1}$ for all w^T.

Let w^T be x^P. Since $\varphi\psi_i$ solves $x^P \approx^?_\alpha u$, we have

$$\begin{aligned} x^P\sigma_{i+1}\nu\mu &= x^P\sigma_i\vartheta_{i+1}\rho_{i+1}\nu\mu = x^P\vartheta_{i+1}\rho_{i+1}\nu\mu \\ &\approx_\alpha u\rho_{i+1}\rho_{i+1}\nu\mu = u\rho_{i+1}\nu\mu = u\nu\nu\mu = u\nu\mu \\ &\approx_\alpha x^P\nu\mu = x^P\varphi\psi_i\mu \\ &= x^P\varphi\psi_{i+1}. \end{aligned} \tag{2}$$

Now let $w^T \neq x^P$. For every unknown v^R from $w^T\sigma_i\vartheta_{i+1}$ we have $v^R\rho_{i+1}\varphi\psi_i\mu = v^R\varphi\psi_i\mu$. Therefore using (1), we get

$$w^T\sigma_{i+1}\nu\mu = w^T\sigma_i\vartheta_{i+1}\rho_{i+1}\nu\mu = w^T\sigma_i\nu\mu = w^T\varphi\psi_i\mu w^T\varphi\psi_{i+1}. \tag{3}$$

Hence, $\sigma_{i+1} \precsim \varphi\psi_{i+1}$. It finishes the proof that for any unifier φ of Γ, the algorithm Unif-Alg computes σ with the property $\sigma|_{\mathsf{unkn}(\Gamma)} \precsim \varphi$, when Γ does not contain sequence unknowns.

Now assume Γ contains sequence unknowns. Let $l := \max\{|\overline{x}^R\varphi| \mid \overline{x} \in dom(\varphi)\}$, i.e., l is the length of the longest sequence to which a sequence unknown from $dom(\varphi)$ is mapped. By fixing $\ell = l$, we can compute $\sigma \in \mathsf{Unif\text{-}Alg}(\Gamma, \ell)$ with the property $\sigma|_{\mathsf{unkn}(\Gamma)} \precsim \varphi$. □

The completeness theorems for Match-Alg and Unif-Alg-Last are easier to prove. We just state them here:

Theorem 6 (Completeness of Match-Alg)**.** *Let φ be a matcher of a matching problem Γ. Then* Match-Alg *computes a σ such that $\sigma = \varphi|_{\mathsf{unkn}(\Gamma)}$.*

Theorem 7 (Completeness of $\mathsf{Unif\text{-}Alg\text{-}Last}$)**.** *Let Γ be a unification problem where every sequence unknown appears in the last argument position, and φ be its unifier. Then there exists $\sigma \in \mathsf{Unif\text{-}Alg\text{-}Last}(\Gamma)$ such that $\sigma \precsim \varphi$.*

The sets $\mathsf{Unif\text{-}Alg}(\Gamma, \ell)$ and $\mathsf{Match\text{-}Alg}(\Gamma)$ are minimal. This follows from the fact that if there are two distinct σ_1 and σ_2 in such a set, then there exists $\overline{x}^P \in dom(\sigma_1) \cap dom(\sigma_2)$ such that the length of their instantiations are different: $|\overline{x}^P \sigma_1| \neq |\overline{x}^P \sigma_2|$. Such a difference can not be repaired by a substitution composition, because the ranges of σ's do not contain sequence markers by construction. Hence, we have neither $\sigma_1 \precsim \sigma_2$ nor $\sigma_2 \precsim \sigma_1$, which implies minimality.

The set $\mathsf{Unif\text{-}Alg\text{-}Last}(\Gamma)$ is singleton, since there is no branching in the derivation tree. The computed unifier is most general.

5 Conclusion

We described three algorithms for solving unification problems and their fragments for terms containing unknowns with permission sets, variadic function constants, atoms, applications, and binders that bind atoms. The design is guided by the syntax of Theorema system, where higher-order expressions are permitted. Unification and matching equations are solved modulo α-equivalence. Termination, soundness, and (restricted) completeness of algorithms are proved. They are implemented as a part of the Theorema system.

Acknowledgment

This work is partially supported by the Austrian Science Fund (FWF) under the project P 28789-N32.

References

[1] A. Asperti, W. Ricciotti, and C. Sacerdoti Coen. Matita tutorial. *J. Formalized Reasoning*, 7(2):91–199, 2014.

[2] M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, and D. Nantes-Sobrinho. Nominal C-unification. In F. Fioravanti and J. P. Gallagher, eds., *LOPSTR'17*, vol. 10855 of LNCS, 235–251. Springer, 2017.

[3] M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. Fixed-point constraints for nominal equational unification. In H. Kirchner, ed., *FSCD'18*, vol. 108 of *LIPIcs*, 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[4] F. Baader and W. Snyder. Unification theory. In J. A. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, 445–532. Elsevier and MIT Press, 2001.

[5] G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, K. Pak, and J. Urban. Mizar: State-of-the-art and beyond. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, eds., *CICM'15*, vol. 9150 of LNCS, 261–279. Springer, 2015.

[6] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Nominal anti-unification. In M. Fernández, ed., *RTA'15*, vol. 36 of *LIPIcs*, 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[7] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[8] B. Buchberger. Mathematica: Doing mathematics by computer? In A. Miola and M. Temperini, eds., *Advances in the Design of Symbolic Computation Systems*, 2–20. Springer Vienna, 1997.

[9] B. Buchberger and A. Craciun. Algorithm synthesis by Lazy Thinking: Examples and implementation in Theorema. *Electr. Notes Theor. Comput. Sci.*, 93:24–59, 2004.

[10] B. Buchberger, T. Jebelean, T. Kutsia, A. Maletzky, and W. Windsteiger. Theorema 2.0: Computer-assisted natural-style mathematics. *J. Formalized Reasoning*, 9(1):149–185, 2016.

[11] C. Calvès and M. Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

[12] C. Calvès and M. Fernández. Matching and alpha-equivalence check for nominal terms. *J. Comput. Syst. Sci.*, 76(5):283–301, 2010.

[13] J. Cheney. Equivariant unification. *J. Aut. Reas.*, 45(3):267–300, 2010.

[14] J. Coelho, B. Dundua, M. Florido, and T. Kutsia. A rule-based approach to XML processing and Web reasoning. In P. Hitzler and T. Lukasiewicz, eds., *RR'10*, vol. 6333 of LNCS, 164–172. Springer, 2010.

[15] J. Coelho and M. Florido. CLP(Flex): Constraint logic programming applied to XML processing. In R. Meersman and Z. Tari, eds., *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Part II*, vol. 3291 of LNCS, 1098–1112. Springer, 2004.

[16] S. Colton. *Automated Theory Formation in Pure Mathematics.* Distinguished dissertations. Springer, 2002.

[17] G. Dowek, M. J. Gabbay, and D. P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822, 2010.

[18] I. Dramnesc, T. Jebelean, and S. Stratulat. Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques. *J. Symb. Comput.*, 90:3–41, 2019.

[19] B. Dundua. *Programming with Sequence and Context Variables: Foundations and Applications.* PhD thesis, University of Porto, 2014.

[20] B. Dundua, T. Kutsia, and M. Marin. Strategies in Pρlog. In M. Fernández, ed., *WRS'09*, vol. 15 of EPTCS, 32–43, 2009.

[21] B. Dundua, T. Kutsia, and M. Marin. Variadic equational matching. In C. Kaliszyk, E. Brady, A. Kohlhase, and C. Sacerdoti Coen, eds., *CICM'19*, vol. 11617 of LNCS, 77–92. Springer, 2019.

[22] B. Dundua, T. Kutsia, K. Reisenberger-Hagmayr. An overview of ρLog. In Y. Lierler and W. Taha, eds., *PADL'17*, vol. 10137 of LNCS. 34–49, Springer, 2017.

[23] M. Fernández and M. Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

[24] M. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *LICS'99*, 214–224. IEEE Computer Society, 1999.

[25] M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[26] M. J. Gabbay. *A Theory of Inductive Definitions with alpha-Equivalence.* PhD thesis, University of Cambridge, 2001.

[27] M. J. Gabbay. Nominal terms and nominal logics: from foundations to meta-mathematics. In *Handbook of Philosophical Logic*, vol. 17, 79–178. Kluwer, 2013.

[28] M. J. Gabbay and C. Wirth. Quantifiers in logic and proof-search using permissive-nominal terms and sets. *J. Log. Comp.*, 25(2):473–523, 2015.

[29] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. TR Logic-92-1, Stanford University, 1992.

[30] M. L. Ginsberg. The MVL theorem proving system. *SIGART Bulletin*, 2(3):57–60, 1991.

[31] M. J. C. Gordon and T. F. Melham, eds.. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[32] J. Harrison. HOL light: An overview. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, eds., *TPHOLs'09*, vol. 5674 of LNCS, 60–66. Springer, 2009.

[33] F. Horozal, F. Rabe, and M. Kohlhase. Flexary operators for formalized mathematics. In S. M. Watt, J. H. Davenport, A. Sexton, P. Sojka, and J. Urban, eds., *CICM'14*, vol. 8543 of LNCS, 312–327. Springer, 2014.

[34] ISO/IEC. Information technology—Common Logic (CL): A framework for a family of logic-based languages. International Standard ISO/IEC 24707:2018, 2018. https://www.iso.org/standard/66249.html.

[35] M. Johansson. Automated theory exploration for interactive theorem proving - an introduction to the Hipster system. In M. Ayala-Rincón and C. A. Muñoz, eds., *ITP'17*, vol. 10499 of LNCS, 1–11. Springer, 2017.

[36] M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *J. Autom. Reasoning*, 47(3):251–289, 2011.

[37] M. Kaufmann, J. S. Moore, and P. Manolios. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.

[38] M. Kerber, C. Rowat, and W. Windsteiger. Using Theorema in the formalization of theoretical economics. In J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, eds., *CICM'11*, vol. 6824 of LNCS, 58–73. Springer, 2011.

[39] B. Konev and T. Jebelean. Combining level-saturation strategies and meta-variables for predicate logic proving in Theorema. RISC Report Series 00-40, RISC, Johannes Kepler University Linz, 2000.

[40] T. Kutsia. Solving and proving in equational theories with sequence variables and flexible arity symbols. TR 02-09, RISC, Johannes Kepler University Linz, 2002. PhD Thesis.

[41] T. Kutsia. Theorem proving with sequence variables and flexible arity symbols. In M. Baaz and A. Voronkov, eds., *LPAR'02*, vol. 2514 of LNCS, 278–291. Springer, 2002.

[42] T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, eds., *AISC'02&Calculemus'02*, vol. 2385 of LNCS, 290–304. Springer, 2002.

[43] T. Kutsia. Equational prover of Theorema. In R. Nieuwenhuis, ed., *RTA'03*, vol. 2706 of LNCS, 367–379. Springer, 2003.

[44] T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symb. Comput.*, 42(3):352–388, 2007.

[45] T. Kutsia and M. Marin. Can context sequence matching be used for querying XML? In L. Vigneron, ed., *UNIF'05*, 77–92, 2005.

[46] T. Kutsia and M. Marin. Solving, reasoning, and programming in common logic. In A. Voronkov, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, eds., *SYNASC'12*, 119–126. IEEE Computer Society, 2012.

[47] J. Levy and M. Villaret. An efficient nominal unification algorithm. In C. Lynch, ed., *RTA'10*, vol. 6 of *LIPIcs*, 209–226. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[48] J. Levy and M. Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31, 2012.

[49] A. Maletzky. Mathematical theory exploration in Theorema: Reduction rings. In M. Kohlhase, M. Johansson, B. R. Miller, L. de Moura, and F. W. Tompa, eds., *CICM'16*, vol. 9791 of LNCS, 3–17. Springer, 2016.

[50] M. Marin and T. Kutsia. On the implementation of a rule-based programming system and some of its applications. In B. Konev and R. Schmidt, eds., *Proceedings of the 4th International Workshop on the Implementation of Logics, WIL'03*, 55–68, 2003.

[51] R. L. McCasland, A. Bundy, and P. F. Smith. MATHsAiD: Automated mathematical theory exploration. *Appl. Intell.*, 47(3):585–606, 2017.

[52] C. Menzel. Knowledge representation, the World Wide Web, and the evolution of logic. *Synthese*, 182(2):269–295, 2011.

[53] O. Montano-Rivas, R. L. McCasland, L. Dixon, and A. Bundy. Scheme-based theorem discovery and concept invention. *Expert Syst. Appl.*, 39(2):1637–1646, 2012.

[54] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, ed., *CADE-11*, vol. 607 of LNCS, 748–752. Springer, 1992.

[55] L. C. Paulson, T. Nipkow, and M. Wenzel. From LCF to Isabelle/HOL. *Formal Asp. Comput.*, 31(6):675–698, 2019.

[56] A. Pease and G. Sutcliffe. First order reasoning on a large ontology. In G. Sutcliffe and S. Schulz, eds., *CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories*, number 257 in CEUR Workshop Proceedings, 59–69, 2007.

[57] S. Pkhakadze. *Some problems of the notation theory.* Tbilisi University Press, 1977. In Russian.

[58] J. Richardson and N. E. Fuchs. Development of correct transformation schemata for Prolog programs. In N. E. Fuchs, ed., *LOPSTR'97*, vol. 1463 of LNCS, 263–281. Springer, 1997.

[59] M. Rosenkranz. A new symbolic method for solving linear two-point boundary value problems on the level of operators. *J. Symb. Comput.*, 39(2):171–199, 2005.

[60] M. Schmidt-Schauß, T. Kutsia, J. Levy, and M. Villaret. Nominal unification of higher order expressions with recursive let. In M. V. Hermenegildo and P. López-García, eds., *LOPSTR'16*, vol. 10184 of LNCS, 328–344. Springer, 2016.

[61] J. D. Scott, P. Flener, J. Pearson, and C. Schulte. Design and implementation of bounded-length sequence variables. In D. Salvagnin and M. Lombardi, eds., *CPAIOR'17*, vol. 10335 of LNCS, 51–67. Springer, 2017.

[62] A. Steen and C. Benzmüller. The higher-order prover Leo-III (extended abstract). In C. Benzmüller and H. Stuckenschmidt, eds., *KI'19*, vol. 11793 of LNCS, 333–337. Springer, 2019.

[63] C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.

[64] D. Vasaru Dupre. Automated Theorem Proving by Integrating Proving, Solving and Computing. TR 00-19, RISC, Johannes Kepler University Linz, 2000.

[65] W. Windsteiger. An automated prover for Zermelo-Fraenkel set theory in Theorema. *J. Symb. Comput.*, 41(3-4):435–470, 2006.

[66] W. Windsteiger. Theorema 2.0: A graphical user interface for a mathematical assistant system. In C. Kaliszyk and C. Lüth, eds., *UITP'12*, vol. 118 of *EPTCS*, 72–82, 2012.

[67] S. Wolfram. *The Mathematica book, 5th Edition.* Wolfram-Media, 2003.