Variadic Equational Matching

Besik Dundua[1,2], Temur Kutsia[3], and Mircea Marin[4]

[1] CTE, International Black Sea University, Tbilisi, Georgia
[2] VIAM, Ivane Javakhishvili Tbilisi State University, Georgia
[3] RISC, Johannes Kepler University Linz, Austria
[4] West University of Timişoara, Romania

Abstract. In this paper we study matching in equational theories that specify counterparts of associativity and commutativity for variadic function symbols. We design a procedure to solve a system of matching equations and prove its soundness and completeness. The complete set of incomparable matchers for such a system can be infinite. From the practical side, we identify two finitary cases and impose restrictions on the procedure to get an incomplete terminating algorithm, which, in our opinion, describes the semantics for associative and commutative matching implemented in the symbolic computation system Mathematica.

1 Introduction

In variadic languages, function symbols do not have a fixed arity. They can take arbitrary number of arguments. In the literature, such symbols are known by different names: flexary, of flexible arity, polyadic, multi-ary, unranked. They are a convenient and useful tool for formalizing mathematical texts, representing symbolic computation data structures, modeling XML documents, expressing patterns in declarative programming, etc. Usually, variadic languages contain variables not only for individual terms, but also for finite sequences of terms, which help to take the full advantage of flexibility of such languages.

On the other hand, the increased expressiveness of variadic languages has its price, from the computational perspective. Solving equations involving sequence variables is a nontrivial task [15,16] and pattern matching, a very common operation in the above-mentioned applications, becomes pretty involved.

In this paper we address the problem of pattern matching in variadic languages, where some function symbols satisfy (the variadic counterparts of) the commutativity (C) and associativity (A) properties. Equational matching in these theories has been intensively studied in languages with ranked alphabets (see, e.g., [2,7,8,11]). Variadic equational matching so far attracted less attention.

We try to address this shortcoming, approaching the problem both from the theoretical and application points of view. From the theoretical side, we propose a modular rule-based system for solving matching equations with A, C, and AC function symbols. Our focus was not on coming up with an optimized, efficient procedure, especially since A-matching problems might even have infinitely many incomparable solutions. Rather, we chose a declarative, modular approach, which

makes proving properties easier. From the application perspective, we show how some intuitive modifications of the rules can lead to terminating cases.

The final part of the paper is devoted to the analysis of the behavior of equational variadic matching algorithm implemented in the symbolic computation system Mathematica [22]. Its programming language, called Wolfram, has a powerful matching engine, using sequence variables and working modulo A and C theories, called there flat and orderless theories, respectively. The matching mechanism is explained in tutorials and help files, but to the best of our knowledge, its formal description has never been published. We try to fill this gap, proposing rules which, in our opinion, describe the input-output behavior and properties of Mathematica's flat and orderless pattern matching.

Related work. In [17], a procedure for flat (A) matching was described and its relation to the correspondent algorithm in Mathematica was discussed. The current work builds on it and extends the results from that paper. Recently, a library to extend Python with variadic matching and sequence variables has been developed [13]. Pattern matching compiler Tom supports associative matching [4]. Usefulness of variadic operators and sequence variables in logical frameworks has been discussed in [10,9]. Variadic matching with sequence variables has been used in the mathematical assistant system Theorema [3], rule-based systems PρLog [6] and its predecessor FunLog [20], programming package Sequentica [21], XML processing language CLP(Flex) [5]. Variadic matching with regular expression types was studied in [19]. Common Logic (CL) [12] is a framework for a family of logic-based languages, designed for knowledge exchange in a heterogeneous network. It comes with variadic symbols and sequence markers (a counterpart of our sequence variables). Syntactic matching for CL was studied in [18].

2 Preliminaries

We assume some familiarity with the standard notions of unification theory [1]. We consider four pairwise disjoint sets: function symbols $\mathcal{F}$, individual variables $\mathcal{V}_{\mathrm{Ind}}$, sequence variables $\mathcal{V}_{\mathrm{Seq}}$, and function variables $\mathcal{V}_{\mathrm{Fun}}$. All the symbols in $\mathcal{F}$ are *variadic*, i.e., their arity is not fixed. We will use x, y, z for individual variables, $\overline{x}, \overline{y}, \overline{z}$ for sequence variables, X, Y, Z for function variables, and a, b, c, f, g, h for function symbols. The set of variables $\mathcal{V}_{\mathrm{Ind}} \cup \mathcal{V}_{\mathrm{Seq}} \cup \mathcal{V}_{\mathrm{Fun}}$ is denoted by $\mathcal{V}$. *Terms* t and *sequence elements* s are defined by the grammar:

$$t ::= x \mid f(s_1, \ldots, s_n) \mid X(s_1, \ldots, s_n), \quad n \geq 0, \qquad\qquad s ::= t \mid \overline{x}.$$

When it is not ambiguous, we write f for the term $f()$ where $f \in \mathcal{F}$. In particular, we will always write a, b, c for $a(), b(), c()$. Terms are denoted by t, r and sequence elements by s, q. Finite, possibly empty sequences of terms are denoted by $\tilde{t}$, $\tilde{r}$, while $\tilde{s}$, $\tilde{q}$ are used to denote sequences of sequence elements.

The *set of variables* of a term t is denoted by $\mathcal{V}(t)$. We can use the subscripts Ind, Seq, and Fun to indicate the sets of individual, sequence, and function variables of a term, respectively. A *ground* term is a term without variables.

The size of a term t, denoted $size(t)$, is the number of symbols in it. These definitions are generalized for any syntactic object throughout the paper. The *head* of a term is its root symbol. The head of a variable is the variable itself.

A *substitution* is a mapping from individual variables to terms, from sequence variables to finite sequences of sequence elements, and from function variables to function symbols or function variables, such that all but finitely many variables are mapped to themselves. (We do not distinguish between a singleton term sequence and its sole element.) We will use lower case Greek letters for substitutions, with ε reserved for the identity substitution.

For a substitution σ, the domain is the set of variables $dom(\sigma) = \{v \in \mathcal{V} \mid \sigma(v) \neq v\}$. A substitution can be represented explicitly as a function by a finite set of bindings of variables in its domain: $\{v \mapsto \sigma(v) \mid v \in dom(\sigma)\}$. For readability, we put term sequences in parentheses. For instance, the set $\{x \mapsto f(a, \overline{y}), \overline{x} \mapsto (), \overline{y} \mapsto (a, X(f(b)), x), X \mapsto g\}$ is such a representation of the substitution, which maps x to the term $f(a, \overline{y})$, $\overline{x}$ to the empty sequence, $\overline{y}$ to the sequence of three elements $(a, X(f(b)), x)$, and X to g.

Instances of a sequence element s and a sequence $\tilde{s}$ under a substitution σ, denoted, respectively, by $s\sigma$ and $\tilde{s}\sigma$, are defined as follows:

$$x\sigma = \sigma(x), \qquad \overline{x}\sigma = \sigma(\overline{x}), \qquad (f(s_1, \ldots, s_n))\sigma = f(s_1\sigma, \ldots, s_n\sigma),$$
$$(X(s_1, \ldots, s_n))\sigma = \sigma(X)(s_1\sigma, \ldots, s_n\sigma), \quad (s_1, \ldots, s_n)\sigma = (s_1\sigma, \ldots, s_n\sigma).$$

Example 1. Let $\sigma = \{x \mapsto f(a), \overline{x} \mapsto (b, c), \overline{y} \mapsto (), X \mapsto g\}$. Then $(X(x, \overline{x}, f(\overline{y})), \overline{x}, \overline{y}, x)\sigma = (g(f(a), b, c, f), b, c, f(a))$. This example also shows that nested sequences are not allowed: they are immediately flattened.

Composition of two substitutions σ and ϑ, written $\sigma\vartheta$, is a substitution defined by $(\sigma\vartheta)(v) = \vartheta(\sigma(v))$ for all $v \in \mathcal{V}$.

An *equation* is a pair of terms. Given a set E of equations over $\mathcal{F}$ and $\mathcal{V}$, we denote by $\doteq_E$ the least congruence relation on the set of finite sequences of sequence elements (over $\mathcal{F}$ and $\mathcal{V}$) that is closed under substitution application and contains E. The set $\doteq_E$ is called an *equational theory* defined by E. Slightly abusing the terminology, we will also call the set E an equational theory or an E-theory. The *signature* of E, denoted $sig(E)$, is the set of all function symbols occurring in E. A function symbol is called *free* with respect to E if it does not occur in $sig(E)$.

A substitution σ is *more general* than a substitution ϑ on a set of variables V modulo an equational theory E, denoted $\sigma \preceq_E^V \vartheta$, if there exists a substitution φ such that $\chi\sigma\varphi \doteq_E \chi\vartheta$ for all individual and sequence variables $\chi \in V$, and $X()\sigma\varphi \doteq_E X()\vartheta$ for all function variables $X \in V$.

Solving equations in an equational theory E is called *E-unification*. If one of the sides of an equation is ground, then it is called a *matching equation*, and solving such equations in a theory E is called *E-matching*. We write E-matching equations as $s \ll_E t$, where t is ground. An *E-matching problem* over $\mathcal{F}$ is a finite set of E-matching equations over $\mathcal{F}$ and $\mathcal{V}$, which we usually denote by Γ: $\Gamma = \{s_1 \ll_E t_1, \ldots, s_n \ll_E t_n\}$.

An *E-matcher* of Γ is a substitution σ such that $s_i\sigma \doteq_E t_i$ for all $1 \le i \le n$. The set of all E-matchers of Γ is denoted by $match_E(\Gamma)$. Γ is *E-matchable*, or *E-solvable*, if $match_E(\Gamma) \neq \emptyset$.

A *complete set of E-matchers* of Γ is a set S of substitutions with the following properties:

1. (Correctness) $S \subseteq match_E(\Gamma)$, i.e., each element of S is an E-matcher of Γ;
2. (Completeness) For each $\vartheta \in match_E(\Gamma)$ there exists $\sigma \in S$ with $\sigma \leq_E^{\mathcal{V}(\Gamma)} \vartheta$.

The set S is a *minimal complete set of matchers* of Γ with respect to $\mathcal{V}(\Gamma)$ if it is a complete set of matchers satisfying the minimality property:

3. (minimality) If there exist $\sigma, \vartheta \in S$ such that $\sigma \leq_E^{\mathcal{V}(\Gamma)} \vartheta$, then $\sigma = \vartheta$.

In this paper we consider equational theories that specify pretty common properties of variadic function symbols: counterparts of *associativity* and *commutativity*. They are defined by the axioms below (for a function symbol f):

$f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z})$	variadic associativity for f
$f(\overline{x}, x, \overline{y}, y, \overline{z}) \doteq f(\overline{x}, y, \overline{y}, x, \overline{z})$	variadic commutativity for f

For an f, we denote these axioms respectively by $\mathsf{A}(f)$ and $\mathsf{C}(f)$. The $\mathsf{A}(f)$ axiom asserts that the nested occurrences of f can be flattened out. The $\mathsf{C}(f)$ says that the order of arguments of f does not matter. Below we often omit the word "variadic" and write associativity and commutativity instead of variadic associativity and variadic commutativity. We also say f is A, C, or AC if, respectively, only $\mathsf{A}(f)$, only $\mathsf{C}(f)$, or both $\mathsf{A}(f)$ and $\mathsf{C}(f)$ hold for f.

An associative normal form (A-normal form) of a term is obtained by applying to it the associativity axiom from left to right as a rewrite rule as long as possible. The notion of normal form extends to substitutions straightforwardly.

In A- and AC-theories there exist matching problems that have infinitely many solutions. This is related to the flatness property of variadic associative symbols, and originates from flat matching [14,17]. The simplest such problem is $f(\overline{x}) \ll_E f()$, where $\mathsf{A}(f) \in E$. Its complete solution set is $\{\{\overline{x} \mapsto ()\}, \{\overline{x} \mapsto f()\}, \{\overline{x} \mapsto (f(), f())\}, \ldots\}$, which is based on the fact that $f(f(), \ldots, f()) \doteq_E f()$ when $\mathsf{A}(f) \in E$. It, naturally, implies that any matching procedure that directly enumerates a complete set of A- or AC-matchers is non-terminating.

In general, our matching problems are formulated in a theory that may contain several A, C, or AC-symbols.

3 Matching Procedure

We formulate our matching procedure in a rule-based manner. The rules operate on a matching equation and return a set of matching equations. They also produce a substitution, which is denoted by ϑ in the rules and is called the local substitution. The matching procedure defined below will be based on a certain

strategy for rule application. It is important to note that terms in the rules are kept in normal forms with respect to associativity. The transformation rules are divided into three groups: the common rules, rules for associative symbols, and the permutation rule that deals with commutativity.

Common Rules. In the common rules there are no restriction on the involved function symbols. They can be free, associative, commutative, or AC.

T: **Trivial**
$s \ll_E s \rightsquigarrow_\varepsilon \emptyset$.

S: **Solve**
$x \ll_E t \rightsquigarrow_\vartheta \emptyset, \qquad$ where $\vartheta = \{x \mapsto t\}$.

FVE: **Function Variable Elimination**
$X(\tilde{s}) \ll_E f(\tilde{t}) \rightsquigarrow_\vartheta \{f(\tilde{s}) \ll_E f(\tilde{t})\}, \qquad$ where $\vartheta = \{X \mapsto f\}$.

Dec: **Decomposition**
$\{f(s, \tilde{s}) \ll_E f(t, \tilde{t})\} \rightsquigarrow_\varepsilon \{s \ll_E t,\ f(\tilde{s}) \ll_E f(\tilde{t})\}, \qquad$ if $s \notin \mathcal{V}$.

IVE: **Individual Variable Elimination**
$f(x, \tilde{s}) \ll_E f(t, \tilde{t}) \rightsquigarrow_\vartheta \{f(\tilde{s}\vartheta) \ll_E f(\tilde{t})\}, \qquad$ where $\vartheta = \{x \mapsto t\}$.

SVP: **Sequence Variable Projection**
$f(\overline{x}, \tilde{s}) \ll_E f(\tilde{t}) \rightsquigarrow_\vartheta \{f(\tilde{s}\vartheta) \ll_E f(\tilde{t})\}, \qquad$ where $\vartheta = \{\overline{x} \mapsto ()\}$.

SVW: **Sequence Variable Widening**
$f(\overline{x}, \tilde{s}) \ll_E f(t, \tilde{t}) \rightsquigarrow_\vartheta \{f(\overline{x}, \tilde{s}\vartheta) \ll_E f(\tilde{t})\}, \qquad$ where $\vartheta = \{\overline{x} \mapsto (t, \overline{x})\}$.

Rules for associative symbols. These rules apply when the involved function symbol satisfies the associativity axiom, i.e., if it is A or AC.

IVE-AH: **Individual Variable Elimination under Associative Head**
$f(x, \tilde{s}) \ll_E f(\tilde{t}_1, \tilde{t}_2) \rightsquigarrow_\vartheta \{f(\tilde{s}\vartheta) \ll_E f(\tilde{t}_2)\}$,
where $\mathsf{A}(f) \in E$ and $\vartheta = \{x \mapsto f(\tilde{t}_1)\}$.

FVE-AH: **Function Variable Elimination under Associative Head**
$f(X(\tilde{s}_1), \tilde{s}_2) \ll_E f(\tilde{t}) \rightsquigarrow_\vartheta \{f(\tilde{s}_1, \tilde{s}_2)\vartheta \ll_E f(\tilde{t})\}$,
where $\mathsf{A}(f) \in E$ and $\vartheta = \{X \mapsto f\}$.

SVW-AH: **Sequence Variable Widening under Associative Head**
$f(\overline{x}, \tilde{s}) \ll_E f(\tilde{t}_1, \tilde{t}_2) \rightsquigarrow_\vartheta \{f(\overline{x}, \tilde{s}\vartheta) \ll_E f(\tilde{t}_2)\}$,
where $\mathsf{A}(f) \in E$ and $\vartheta = \{\overline{x} \mapsto (f(\tilde{t}_1), \overline{x})\}$.

Permutation rule. This is a straightforward rule which permutes arguments of a C or AC function symbol:

Per: **Permutation**

$f(\tilde{s}) \ll_E f(t_1, \ldots, t_n) \rightsquigarrow_\varepsilon \{f(\tilde{s}) \ll_E f(t_{\pi(1)}, \ldots, t_{\pi(n)})\}$,

where $\mathsf{C}(f) \in E$ and π is a permutation of $(1, \ldots, n)$.

The matching procedure. The matching procedure $\mathfrak{M}$ works on triples $\Gamma_1; \Gamma_2; \sigma$. It selects an equation from Γ_1 or from Γ_2 and applies one of the rules above to it. If several rules are applicable, one is chosen nondeterministically (unless the control defined below forbids it). Such a nondeterminism introduces branching in the derivation tree. Assume the rule transforms $s \ll_E t \rightsquigarrow_\vartheta \Gamma$. One step of $\mathfrak{M}$ is performed as follows:

- If the selected equation is from Γ_1 and the applied rule is the Per rule, Γ contains a single equation, which is moved to Γ_2:

  $$\{s \ll_E t\} \uplus \Gamma_1'; \Gamma_2; \sigma \rightsquigarrow \Gamma_1'; \Gamma_2 \cup \Gamma; \sigma.$$

  It is forbidden to apply any other rule to $s \ll_E t$ in Γ_1 if Per is applicable to it. However, Per rule itself can be applied nondeterministically with different permutations, i.e., only Per can cause branching in this case.
- If the selected equation is from Γ_1, Per is not applicable, and a common or associative rule applies to it, then the new equations remain in Γ_1:

  $$\{s \ll_E t\} \uplus \Gamma_1'; \Gamma_2; \sigma \rightsquigarrow (\Gamma_1' \cup \Gamma)\vartheta; \Gamma_2\vartheta; \sigma\vartheta.$$

- If the selected equation is from Γ_2, then the Per rule does not apply. The equation should be transformed by common or associative rules. Any rule except Dec leaves the new equations in Γ_2:

  $$\Gamma_1; \{s \ll_E t\} \uplus \Gamma_2'; \sigma \rightsquigarrow \Gamma_1\vartheta; (\Gamma_2' \cup \Gamma)\vartheta; \sigma\vartheta.$$

  If the applied rule is Dec, then the first new equation moves to Γ_1, and the second one (whose head does not change) remains in Γ_2:

  $$\Gamma_1; \{f(s, \tilde{s}) \ll_E f(t, \tilde{t})\} \uplus \Gamma_2'; \sigma \rightsquigarrow \Gamma_1 \cup \{s \ll_E t\}; \Gamma_2' \cup \{f(\tilde{s}) \ll_E f(\tilde{t})\}; \sigma.$$

Hence, to summarize, $\mathfrak{M}$ applies common and associative rules to Γ_1 if the head of the involved equation is not commutative or associative-commutative. Otherwise, the equation is transformed by the permutation rule, moved to Γ_2 and is processed there by common or associative rules. Only the equations generated by decomposition can go back to Γ_1. This process roughly can be described as "permute the arguments of C- and AC-symbols and apply syntactic and A-matching rules." It is a sound and complete approach (as we will show below), but not necessarily the most efficient one.

Given a matching problem Γ, we create the initial system $\Gamma; \emptyset; \varepsilon$ and start applying the described procedure. The process stops either at $\emptyset; \emptyset; \sigma$ (success), or at $\Gamma_1; \Gamma_2; \sigma$ such that $\mathfrak{M}$ can not make a step (failure). The procedure might also run forever. The substitution σ at the success leaf $\emptyset; \emptyset; \sigma$ is called an *answer of Γ computed by $\mathfrak{M}$*, or just a *computed answer* of Γ. We denote by $comp(\Gamma)$ the set of answers of Γ computed by the matching procedure $\mathfrak{M}$.

Recall that our matching problems are formulated in a theory that may contain one or more A, C, or AC-symbols. In the soundness and completeness theorems below, the equational theory E in $match_E(\Gamma)$ refers to such a theory.

Theorem 1 (Soundness). *$comp(\Gamma) \subseteq match_E(\Gamma)$ for any E-matching problem Γ.*

Proof (sketch). The theorem follows from the fact that each transformation rule is sound: If $\Gamma_1; \Gamma_2; \sigma \Longrightarrow_{\mathsf{R}} \Gamma_1'; \Gamma_2'; \sigma\vartheta$ by a transformation rule R, and $\varphi \in match_E(\Gamma_1' \cup \Gamma_2')$, then $\vartheta\varphi \in match_E(\Gamma_1 \cup \Gamma_2)$. This in itself is not difficult to check: a straightforward analysis of rules suffices. □

Theorem 2 (Completeness). *Let Γ be an E-matching problem. Assume $\sigma \in match_E(\Gamma)$ and it is in the A-normal form. Then $\sigma \in comp(\Gamma)$.*

Proof. We prove the theorem by constructing the derivation that starts from $\Gamma; \emptyset; \varepsilon$ and ends with $\emptyset; \emptyset; \sigma$. We denote such a derivation by $\mathcal{D}(\Gamma; \emptyset \rightsquigarrow \sigma)$.

The proof idea is as follows: We associate a complexity measure $M(\Gamma_1; \Gamma_2; \gamma)$ to each system $(\Gamma_1; \Gamma_2; \gamma)$ where $\gamma \in match_E(\Gamma_1 \cup \Gamma_2)$. Measures are ordered by a well-founded ordering. Then we use well-founded induction to prove that $\mathcal{D}(\Gamma; \emptyset \rightsquigarrow \sigma)$ exists: First, assume that for all $\Gamma_1; \Gamma_2; \gamma$ with $\gamma \in match_E(\Gamma_1 \cup \Gamma_2)$ and $M(\Gamma; \emptyset; \sigma) > M(\Gamma_1; \Gamma_2; \gamma)$ there exists a derivation $\mathcal{D}(\Gamma_1; \Gamma_2 \rightsquigarrow \gamma)$. Then, analyzing the selected equation in Γ, we prove that there is a matching rule R getting us (maybe after an application of Per rule) to one of such $\Gamma_1; \Gamma_2$'s, say, to $\Gamma_1'; \Gamma_2'$, from which there is a derivation $\mathcal{D}(\Gamma_1'; \Gamma_2' \rightsquigarrow \sigma')$, which is a part of σ. Combining the step R with $\mathcal{D}(\Gamma_1'; \Gamma_2' \rightsquigarrow \sigma')$ (and, maybe, with Per), we construct the desired derivation $\mathcal{D}(\Gamma; \emptyset \rightsquigarrow \sigma)$.

The above mentioned complexity measure is defined as a pair of multisets $M(\Gamma_1; \Gamma_2; \gamma) = (M_1, M_2)$, where $M_1 = \{size(v\gamma) \mid v \in dom(\gamma)\}$ and $M_2 = \{size(r) \mid l \ll_E r \in \Gamma_1 \cup \Gamma_2\}$. Measures are ordered by the lexicographic combination of two multiset extensions of the standard natural number ordering.

Let Γ have the form $\{s \ll_E t\} \uplus \Gamma'$. We proceed by induction as described above. It has to be shown that there is at least one matching rule that transforms the selected equation $s \ll_E t$. We distinguish between two alternatives: either s and t have the same heads, or they do not.

Case 1: $head(s) = head(t)$. The case when $s = t$ is trivial. Assume $s \neq t$ and their common head satisfies the associativity and commutativity axioms. (For free or commutative symbols, the reasoning is similar and we do not spell out its details.) Let $s = f(s_1, \tilde{s})$ and $t = f(t_1, \ldots, t_n)$. Then $s\sigma$ tells us which permutation of t we should take. Performing Per with that permutation π, we get $\Gamma'; \{s \ll_E$

$f(t_{\pi(1)}, \ldots, t_{\pi(n)})\}; \varepsilon$. To construct the desired derivation we should look at s_1. If it is neither a variable nor a term whose head is a function variable, we apply the Dec rule, which gives $\Gamma_1' \cup \{s_1 \ll_E t_{\pi(1)}\}; \{f(\tilde{s}) \ll_E f(t_{\pi(2)}, \ldots, t_{\pi(n)})\}; \varepsilon$. The complexity measure of the system is strictly smaller than of the previous one, therefore, by the IH we can construct the desired derivation. If s_1 is a sequence variable, then $s_1\sigma$ tells us whether we should use SVP, SVW, SVW-AH rule. For instance, if $s_1\sigma = (f(t_{\pi(1)}, \ldots, t_{\pi(k)}), t_{\pi(k+1)}, \ldots, t_{\pi(m)})$, $m \leq n$, we use the rule SVW-AH with $\vartheta = \{s_1 \mapsto (f(t_{\pi(1)}, \ldots, t_{\pi(k)}), s_1)\}$ and get $\Gamma_1'\vartheta; \{f(s_1, \tilde{s}) \ll_E f(t_{\pi(k+1)}, \ldots, t_{\pi(n)})\}$. It has a matcher $\gamma = (\sigma \setminus \{s_1 \mapsto ((f(t_{\pi(1)}, \ldots, t_{\pi(k)}), s_1))\}) \cup \{s_1 \mapsto (t_{\pi(k+1)}, \ldots, t_{\pi(m)})\}$. Then the complexity measure of the obtained system is strictly smaller that the current one and we can use the IH again. If s_1 is an individual variable, we proceed similarly, using either IVE or IVE-AH. If the head of s_1 is a function variable X, we use FVE-AH or Dec, depending on $X\sigma$. In all cases, we can construct the desired derivation by using the IH.

Case 2: $head(s) \neq head(t)$. Then we use IVE or FVE. They decrease the complexity measure of the involved systems and we can again use the IH. □

4 Finitary Fragment and Variant

A *fragment* of equational matching is obtained by restricting the form of the input, while *variants* require computing solutions of some special form without restricting the input. It is obvious that the procedure $\mathfrak{M}$ stops for any matching problem that does not contain sequence variables, but there are some other interesting terminating fragments and variants as well.[5]

4.1 Bounded Fragment

We start with a fragment that restricts occurrences of sequence variables.

Definition 1. *Let Γ be a matching problem. A sequence variable $\overline{x}$ is called* bounded *in Γ if it occurs under at least two different function symbols or only under free symbol in a subterm of Γ. The problem Γ is called* bounded *if all sequence variables occurring in Γ are bounded in it.*

Example 2. Let $E = \{\mathsf{A}(f), \mathsf{C}(f), \mathsf{A}(g), \mathsf{C}(g)\}$. The following matching problems are bounded:

- $\{f(\overline{x}) \ll_E f(a, g(b)),\ g(\overline{x}) \ll_E g(f(a), b)\}$, which has two solutions $\{\overline{x} \mapsto (f(a), g(b))\}$ and $\{\overline{x} \mapsto (g(b), f(a))\}$.

[5] It is also possible to modify $\mathfrak{M}$ so that it terminates and computes a finite representation of the infinite set of matchers with the help of regular expressions over substitution composition. For associative (flat) matching, an implementation of such a procedure can be found at `https://www3.risc.jku.at/people/tkutsia/software.html`.

– $\{f(\overline{x}) \ll_E f(g(), g()),\ g(\overline{x}) \ll_E g(f(), f(), f())\}$, which has 120 solutions: $\{\{\overline{x} \mapsto \tilde{t}\} \mid \tilde{t}$ is a permutation of $(g(), g(), f(), f(), f())\}$.

An important property of bounded matching problems is the existence of a bound on the size of their solutions. More precisely, the following lemma holds:

Lemma 1. *Let Γ be a bounded matching problem and σ be its solution. Assume that the terms in the range of σ are* A*-normalized. Then for every variable $v \in \mathcal{V}(\Gamma)$, we have $size(v\sigma) \leq \sum_{s \ll_E t \in \Gamma} size(t)$.*

Proof. Since Γ is bounded, there will be an occurrence of v in some $s \ll_E t \in \Gamma$ such that $v\sigma$ is not flattened in $s\sigma$. If the inequality does not hold, we will have $size(s\sigma) \geq size(v\sigma) > size(t)$, contradicting the assumption that σ solves Γ. □

To design a terminating procedure for bounded matching problems, we just add an extra failure check. Let Γ be the initial matching problem and $m = \sum_{s \ll_E t \in \Gamma} size(t)$. For any system $\Gamma_1; \Gamma_2; \sigma$ obtained during the run of the matching procedure, we check whether there exists $v \in \mathcal{V}(\Gamma)$ such that $size(v\sigma) > m$. The check is performed before any rule applies to the system. If it succeeds, we terminate the development of that derivation. Otherwise, we continue as usual. In this way, no solution will be missed, and the search tree will be finite. Let us call this procedure $\mathfrak{M}_{\mathrm{B}}$. We conclude that the following theorem holds:

Theorem 3. *The bounded fragment admits a terminating sound and complete matching procedure $\mathfrak{M}_{\mathrm{B}}$.*

4.2 Strict Variant

Infinitely many solutions to our matching problems are caused by sequences of $f()$'s in the matchers, where f is an A or AC function symbol. But one might be interested in solutions, which do not introduce such extra $f()$'s.

For a precise characterization, we modify the variadic associativity axiom into variadic strict associativity: $f(\overline{x}, f(\overline{y}_1, y, \overline{y}_2), \overline{z}) \doteq f(\overline{x}, \overline{y}_1, y, \overline{y}_2, \overline{z})$. For an f, this axiom is denoted by $\mathsf{A}_{\mathrm{s}}(f)$ and we use A_{s} for the corresponding equational theory. Obviously, any solution of a matching problem modulo A_{s} or $\mathsf{A}_{\mathrm{s}}\mathsf{C}$ is also a solution modulo A or AC. Hence, we can say that we are aiming at solving a variant of A or AC-matching. We call it the *strict* variant and adapt $\mathfrak{M}$ to compute matchers for it. The adaptation is done by small changes in the associative rules:

– We change IVE-AH and SVW-AH, adding the condition that $\tilde{t}_1$ is not the empty hedge. The modified rules are called IVE-AH-strict and SVW-AH-strict.
– In the rule FVE-AH, we add the condition that $\tilde{s}_1$ is not the empty hedge, obtaining the FVE-AH-strict rule.

Replacing the associative rules by the strict associative rules, we obtain a matching procedure denoted by $\mathfrak{M}_{\mathrm{S}}$.

Example 3. If $E = \{\mathsf{A}(f)\}$, the matching problem $f(\overline{x}) \ll_E f(a, a)$ has infinitely many solutions. If $E = \{\mathsf{A}_{\mathrm{s}}(f)\}$, there are five matchers: $\{\overline{x} \mapsto (a, a)\}$, $\{\overline{x} \mapsto f(a, a)\}$, $\{\overline{x} \mapsto (f(a), a)\}$, $\{\overline{x} \mapsto (a, f(a))$, $\{\overline{x} \mapsto (f(a), f(a))\}$.

Theorem 4. *The procedure $\mathfrak{M}_{\mathrm{S}}$ is sound and terminating.*

Proof. Soundness means that the procedure only computes strict solutions. That it computes solutions, follows from Theorem 1. Strictness of the computed solutions follows from the fact that no rule introduces terms like $f()$ in the matchers if these terms do not appear in the right hand sides of matching equations.

For termination, we associate to each matching problem Γ its complexity measure: a triple $\langle n, M_{ntr}, M_l \rangle$, where n is the number of distinct variables in Γ, M_{ntr} is the multiset $M_{ntr} = \{size(t) \mid s \ll_E t \in \Gamma \text{ for some } s, \text{ and } t \text{ does not have the form } f()\}$, and M_l is the multiset $M_l = \{size(s) \mid s \ll_E t \in \Gamma \text{ for some } t\}$.

The measures are compared lexicographically, yielding a well-founded ordering. The following table shows that each rule in $\mathfrak{M}_{\mathrm{S}}$ except Per, strictly decreases the measure. Permutation is applied only finitely many times, because it produces permutations for each subterm of the right hand sides of matching equations with commutative head. It implies that $\mathfrak{M}_{\mathrm{S}}$ is terminating:

Rule	n	M_{ntr}	M_l
T	$\geq$	$\geq$	$>$
Dec, SVW, SVW-AH-strict	$=$	$>$	
S, IVE, FVE, SVP, IVE-AH-strict, FVE-AH-strict	$>$		

□

The set of strict matchers computed by $\mathfrak{M}_{\mathrm{S}}$ is also minimal. Note that the size of matchers is bounded by the size of the right-hand side of matching equations both for bounded fragment and strict variant.

4.3 Complexity

Both bounded fragment and strict variant are NP-complete problems. Membership in NP is trivial, and hardness follows from the hardness of syntactic variadic matching problem. The latter can be shown by encoding systems of linear Diophantine equations over natural numbers: An equation $A_1x_1 + \cdots + A_nx_n = B$ is encoded as a syntactic matching equation $g(\overline{x}_1, \ldots, \overline{x}_1, \ldots, \overline{x}_n, \ldots, \overline{x}_n) \ll_E g(b, \ldots, b)$, where each $\overline{x}_i$ appears A_i times, $1 \leq i \leq n$, and b appears B times. To encode a system, we take matching equations $s_1 \ll_E t_1, \ldots, s_k \ll_E t_k$ and form a single matching equation as usual, using a free function symbol: $g(s_1, \ldots, s_k) \ll_E g(t_1, \ldots, t_k)$. We can obtain a solution of the original Diophantine system from a solution σ of the matching equation: For all $1 \leq i \leq n$, if $\overline{x}_i\sigma = \tilde{t}$, then $x_i = \mathit{length}(\tilde{t})$ gives a solution of the Diophantine system, where $\mathit{length}(\tilde{t})$ is the number of elements in the sequence $\tilde{t}$.

5 Experimenting with the Mathematica Variant

The programming language of Mathematica, called Wolfram, supports equational variadic matching in A, C, AC theories with individual and sequence vari-

ables. The terminology is a bit different, though. Variadic associative symbols there are called *flat* and commutative ones *orderless*. Individual variables correspond to patterns like `x_`, and sequence variables to patterns like `y___`.

The matching variants used in Mathematica are efficiently implemented, but the algorithm is not public. In this section we first show Mathematica's behavior on some selected characteristic examples and then will try to imitate it by variants of our rules. In the experiments we used the Mathematica built-in function `ReplaceList[expr,rules]`, which attempts to transform the entire expression `expr` by applying a rule or list of rules in all possible ways, and returns a list of the results obtained. In transformation, Mathematica tries to match `rules` to `expr`, exhibiting the behavior of the built-in matching mechanism. The equational theories can be specified by setting attributes (flat, orderless) to symbols.

The examples below are used to illustrate the behavior of Mathematica, but we prefer to write those examples in the notation of this paper. We compare it to our strict variant, because it also does not compute extra $f()$'s in the answer. However, they are not the same, as the examples below show. We report only sets of matchers, ignoring their order and how many times the same (syntactically or modulo an equational theory) matcher was computed.

Problem:	$f(\overline{x}) \ll_E f(a),\ f$ is A or AC.	(1)
Strict matchers:	$\{\overline{x} \mapsto a\}, \{\overline{x} \mapsto f(a)\}$.	
Mathematica:	$\{\overline{x} \mapsto a\}$.	
Problem:	$f(f()) \ll_E f(),\ f$ is A or AC.	(2)
Strict matchers:	No solutions.	
Mathematica:	ε.	
Problem:	$f(x, y) \ll_E f(a, b),\ f$ is AC.	(3)
Strict matchers:	$\{x \mapsto t_1, y \mapsto t_2\},$ with $t_1 \in \{a, f(a)\}, t_2 \in \{b, f(b)\}$,	
	$\{x \mapsto t_1, y \mapsto t_2\},$ with $t_1 \in \{b, f(b)\}, t_2 \in \{a, f(a)\}$.	
Mathematica:	$\{x \mapsto f(a), y \mapsto f(b)\}, \{x \mapsto a, y \mapsto b\},$	
	$\{x \mapsto f(b), y \mapsto f(a)\}, \{x \mapsto b, y \mapsto a\}$	
Problem:	$f(\overline{x}, \overline{y}) \ll_E f(a, b),\ f$ is AC.	(4)
Strict matchers:	$\{\overline{x} \mapsto (), \overline{y} \mapsto (t_1, t_2)\},$ with $t_1 \in \{a, f(a)\}, t_2 \in \{b, f(b)\}$,	
	$\{\overline{x} \mapsto (), \overline{y} \mapsto (t_1, t_2)\},$ with $t_1 \in \{b, f(b)\}, t_2 \in \{a, f(a)\}$,	
	$\{\overline{x} \mapsto (), \overline{y} \mapsto t\},$ with $t \in \{f(a, b), f(b, a)\}$,	
	$\{\overline{x} \mapsto (t_1, t_2), \overline{y} \mapsto ()\},$ with $t_1 \in \{a, f(a)\}, t_2 \in \{b, f(b)\}$,	
	$\{\overline{x} \mapsto (t_1, t_2), \overline{y} \mapsto ()\},$ with $t_1 \in \{b, f(b)\}, t_2 \in \{a, f(a)\}$,	
	$\{\overline{x} \mapsto t, \overline{y} \mapsto ()\},$ with $t \in \{f(a, b), f(b, a)\}$,	
	$\{\overline{x} \mapsto t_1, \overline{y} \mapsto t_2\},$ with $t_1 \in \{a, f(a)\}, t_2 \in \{b, f(b)\}$,	
	$\{\overline{x} \mapsto t_1, \overline{y} \mapsto t_2\},$ with $t_1 \in \{b, f(b)\}, t_2 \in \{a, f(a)\}$,	
Mathematica:	$\{\overline{x} \mapsto a, \overline{y} \mapsto b\}, \{\overline{x} \mapsto b, \overline{y} \mapsto a\},$	

	$\{\overline{x} \mapsto (a,b), \overline{y} \mapsto ()\}, \{\overline{x} \mapsto (), \overline{y} \mapsto (a,b)\},$	
Problem:	$f(x,\overline{y}) \ll_E f(a,b,c),\ f$ is A.	(5)
Strict matchers:	$\{x \mapsto a, \overline{y} \mapsto f(b,c)\}, \{x \mapsto f(a), \overline{y} \mapsto f(b,c)\},$	
	$\{x \mapsto f(a,b), \overline{y} \mapsto c\}, \{x \mapsto f(a,b), \overline{y} \mapsto f(c)\}.$	
	$\{x \mapsto t_1, \overline{y} \mapsto (t_2,t_3)\},$	
	with $t_1 \in \{a, f(a)\}, t_2 \in \{b, f(b)\}, t_3 \in \{c, f(c)\},$	
	$\{x \mapsto f(a,b,c), \overline{y} \mapsto ()\}.$	
Mathematica:	$\{x \mapsto a, \overline{y} \mapsto (b,c)\}, \{x \mapsto f(a), \overline{y} \mapsto (b,c)\},$	
	$\{x \mapsto f(a,b), \overline{y} \mapsto c\}, \{x \mapsto f(a,b,c), \overline{y} \mapsto ()\}.$	
Problem:	$g(f(\overline{x}), \overline{x}) \ll_E g(f(a), f(a)),\ f$ is A or AC, g is free.	(6)
Strict matchers:	$\{\overline{x} \mapsto f(a)\}.$	
Mathematica:	No solutions.	
Problem:	$g(f(\overline{x}), g(\overline{x})) \ll_E g(f(b,a), g(b,a)),\ f$ is C, g is free.	(7)
Strict matchers:	$\{\overline{x} \mapsto (b,a)\}.$	
Mathematica:	No solutions.	

In (2), strictness does not allow to flatten the left hand side, but Mathematica does not have this restriction and transforms the term into $f()$.

Interestingly, the behavior of Mathematica's matching changed from the version 6.0 to the version 11.2. As it was reported in [17], for the problem (5), Mathematica 6.0 would return three out of four substitutions reported above. It would not compute $\{x \mapsto a, \overline{y} \mapsto (b,c)\}$.

In problems like (3) Mathematica does not compute a matcher in which one individual variable is mapped to a subterm from the right hand side, and the other one is instantiated by f applied to a *single* subterm from the right hand side. (The same is true when f is A.) Examining more examples, e.g. $f(x,y,z) \ll_E f(a,b,c,d)$, for f being AC, confirms this observation. One can see there matchers like $\{x \mapsto a, y \mapsto b, z \mapsto f(c,d)\}$ and $\{x \mapsto f(a), y \mapsto f(b), z \mapsto f(c,d)\}$ (and many more, the solution set consists of 72 matchers), but not $\{x \mapsto a, y \mapsto f(b), z \mapsto f(c,d)\}$ or similar.

It is interesting to see how sequence variables behave in such a situation. Example (4) shows that in Mathematica matchers, f is not applied to terms from the right hand side. Besides, when a sequence variable is instantiated by a sequence, the order of elements in that sequence coincide with their relative order in the *canonical form* of the right hand side of the equation. For instance, in (4) Mathematica does not return $\{\overline{x} \mapsto (b,a), \overline{y} \mapsto ()\}$. Note that if f were only C in (4), Mathematica would still compute exactly the same set of matchers.

The canonical form of the right hand side is important. There is a so called canonical order imposed on all Mathematica expressions,[6] and whenever there is a commutative symbol, the system rearranges its arguments according to this order. This is why Mathematica returns $\{\overline{x} \mapsto (a, b)\}$ to the matching problem $f(\overline{x}) \ll_E f(b, a)$, when $\mathsf{C}(f) \in E$. The arguments of $f(b, a)$ are first rearranged by the canonical order into $f(a, b)$, and matching is performed afterwards. These issues affect solvability of problems, as one can see in (7). It also indicates that imitating Mathematica's nonlinear matching (i.e. when the same variable occurs more than once in matching equations) is not trivial.

The final set of examples concerns function variables. One can see from the examples that even if a function variable gets instantiated by an equational symbol, Mathematica treats that instance as a free symbol (in (9) and (10), $f(b, a)$ is first normalized to $f(a, b)$):

Problem:	$X(x) \ll_E f(a),\ f$ is A or AC.	(8)
Strict matchers:	$\{X \mapsto f, x \mapsto a\}, \{X \mapsto f, x \mapsto f(a)\}$.	
Mathematica:	$\{X \mapsto f, x \mapsto a\}$.	
Problem:	$X(x) \ll_E f(a, b),\ f$ is AC.	(9)
Strict matchers:	$\{X \mapsto f,\ x \mapsto f(a, b)\}$.	
Mathematica:	No solutions.	
Problem:	$X(a, b) \ll_E f(b, a),\ f$ is C.	(10)
Strict matchers:	$\{X \mapsto f\}$.	
Mathematica:	$\{X \mapsto f\}$.	
Problem:	$X(b, a) \ll_E f(b, a),\ f$ is C.	(11)
Strict matchers:	$\{X \mapsto f\}$.	
Mathematica:	No solutions.	
Problem:	$f(x, X(b, c)) \ll_E f(a, b, c), f$ is A or AC.	(12)
Strict matchers:	$\{X \mapsto f\}$.	
Mathematica:	No solutions.	

These observations suggest that an algorithm that tries to imitate Mathematica's equational matching behavior should first linearize the matching problem by giving unique names to all variable occurrences, try to solve the obtained linear problem, and at the end check whether the obtained solutions are consistent with the original variable names, i.e, if v_1 and v_2 were the unique copies of the same variable, then the computed matcher should map v_1 and v_2 to the same expression. Linearization makes substitution application to the remaining matching problems unnecessary.

[6] Roughly, the canonical order orders symbols alphabetically and extends to trees with respect to left-to-right pre-order.

To model the behavior of Mathematica's equational matching, we need to look into the behavior of each kind of variable.

Imitating the behavior of individual variables under A *and* AC *symbols.* This concerns equations of the form $f(\tilde{s}) \ll_E f(\tilde{t})$, where $\mathsf{A}(f) \in E$. As we observed, if individual variables x and y occur as arguments of $f(\tilde{s})$, and t_1 and t_2 are two terms among $\tilde{t}$, then for the same matcher σ it can not happen that $x\sigma = t_1$ and $y\sigma = f(t_2)$: Either $x\sigma$ should also have the head f, or $y\sigma$ should have more arguments. This is what Mathematica does.

To imitate this behavior, we introduce markings for equations of the form $f(\tilde{s}) \ll_E f(\tilde{t})$, where $\mathsf{A}(f) \in E$. Initially, they are not marked. If IVE transforms such an unmarked equation, the obtained equation is marked by 0. IVE-AH-strict introduces marking 1, if $\tilde{t}_1$ is a single term in this rule. Otherwise, it does not mark the obtained equation. Further, if an equation is marked by 1, then IVE can not apply to it, while IVE-AH-strict can. If an equation is marked by 0, IVE-AH-strict may not use a singleton sequence $\tilde{t}_1$: more terms should be put in it. The equation can be also transformed by IVE.

Imitating the behavior of sequence variables under A *and* AC *symbols.* For equations $f(\tilde{s}) \ll_E f(\tilde{t})$, where $\mathsf{A}(f) \in E$ and a sequence variable $\overline{x}$ appears in its solution σ, no element of the sequence $\overline{x}\sigma$ should have f as its head. For this, we simply drop SVW-AH-strict.

Imitating the behavior of function variables. Equations of the form $X(\tilde{s}) \ll_E f(\tilde{t})$ are transformed by a new rule, which we denote by FVE-M: $X(\tilde{s}) \ll_E f(\tilde{t}) \rightsquigarrow_\vartheta \{g(\tilde{s}) \ll_E g(\tilde{t})\}$, where $\vartheta = \{X \mapsto f\}$ and g is free. At the same time, if $\mathsf{C}(f) \in E$, then $g(\tilde{t})$ is brought to the canonical form. FVE-M replaces FVE. Besides, FVE-AH-strict is dropped.

We denote the obtained algorithm by $\mathfrak{M}_{\mathrm{Mma}}$ to indicate that it (tries to) imitate the Mathematica variant of variadic equational matching.

Example 4. We apply $\mathfrak{M}_{\mathrm{Mma}}$ to the problem (3): $f(x, y) \ll_E f(a, b)$ with $E = \{\mathsf{A}(f), \mathsf{C}(f)\}$. Consider only the branch generated by applying Per with the identity permutation. IVE marks the obtained equation $f(y) \ll_E f(b)$ by 0 and records substitution $\{x \mapsto a\}$. To $f(y) \ll_E f(b)$, we can not apply IVE-AH-strict, because the marker is 0 and $f(b)$ does not have enough arguments this rule could assign to y. Therefore, IVE applies again, returning the matcher $\{x \mapsto a, y \mapsto b\}$.

On the alternative branch, IVE-AH-strict gives the substitution $\{x \mapsto f(a)\}$ and marks $f(y) \ll_E f(b)$ by 1. In the next step only the same rule applies and we get $\{x \mapsto f(a), y \mapsto f(b)\}$.

Example 5. Let the equation be $g(X(x, y), X(y, x)) \ll_E g(f(a, b), f(b, a))$, where $E = \{\mathsf{C}(f)\}$. Linearization and decomposition give two matching equations $X_1(x_1, y_1) \ll_E f(a, b)$ and $X_2(y_2, x_2) \ll_E f(b, a)$. The first one gives the solution $\sigma_1 = \{X_1 \mapsto f, x_1 \mapsto a, y_1 \mapsto b\}$. In the second one, normalization changes the right hand side into $f(a, b)$ and the solution is $\sigma_2 = \{X_2 \mapsto f, x_2 \mapsto b, y_2 \mapsto a\}$. Since $x_1\sigma_1 \neq x_2\sigma_2$, the solutions are inconsistent and the problem is not solvable.

6 Discussion and Conclusion

We studied matching in variadic equational theories for associativity, commutativity, and their combination. A-matching is infinitary, which leads to a nonterminating procedure. It is still possible to have a terminating algorithm, which produces a finite representation of the infinite minimal complete set of matchers, but we did not consider this option in this paper. Instead, we formulated a modular procedure, which combines common and associative matching rules and deals with commutativity by permutation. The procedure can be easily changed to obtain special terminating cases. We illustrated two such cases: bounded fragment and strict variant. Further modifying the latter, we tried to imitate the behavior of the powerful equational matching algorithm of the Mathematica system.

Acknowledgments. This research has been partially supported by the Austrian Science Fund (FWF) under the project 28789-N32 and the Shota Rustaveli National Science Foundation of Georgia (SRNSFG) under the grant YS-18-1480.

References

1. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 445–532. Elsevier and MIT Press (2001)
2. Benanav, D., Kapur, D., Narendran, P.: Complexity of matching problems. J. Symb. Comput. **3**(1/2), 203–216 (1987). https://doi.org/10.1016/S0747-7171(87)80027-5
3. Buchberger, B., Craciun, A., Jebelean, T., Kovács, L., Kutsia, T., Nakagawa, K., Piroi, F., Popov, N., Robu, J., Rosenkranz, M., Windsteiger, W.: Theorema: Towards computer-aided mathematical theory exploration. J. Appl. Logic **4**(4), 470–504 (2006). https://doi.org/10.1016/j.jal.2005.10.006
4. Cirstea, H., Kirchner, C., Kopetz, R., Moreau, P.E.: Anti-patterns for rule-based languages. J. Symb. Comput. **45**(5), 523–550 (2010). https://doi.org/10.1016/j.jsc.2010.01.007
5. Coelho, J., Florido, M.: CLP(Flex): constraint logic programming applied to XML processing. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, Agia Napa, Cyprus, October 25-29, 2004, Proceedings, Part II. Lecture Notes in Computer Science, vol. 3291, pp. 1098–1112. Springer (2004). https://doi.org/10.1007/978-3-540-30469-2_17
6. Dundua, B., Kutsia, T., Reisenberger-Hagmayer, K.: An overview of PρLog. In: Lierler, Y., Taha, W. (eds.) Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10137, pp. 34–49. Springer (2017). https://doi.org/10.1007/978-3-319-51676-9_3
7. Eker, S.: Single elementary associative-commutative matching. J. Autom. Reasoning **28**(1), 35–51 (2002). https://doi.org/10.1023/A:1020122610698
8. Eker, S.: Associative-commutative rewriting on large terms. In: Nieuwenhuis, R. (ed.) Rewriting Techniques and Applications, 14th International Conference, RTA

2003, Valencia, Spain, June 9-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2706, pp. 14–29. Springer (2003). https://doi.org/10.1007/3-540-44881-0_3
9. Horozal, F.: A Framework for Defining Declarative Languages. Ph.D. thesis, Jacobs University Bremen (2014)
10. Horozal, F., Rabe, F., Kohlhase, M.: Flexary operators for formalized mathematics. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8543, pp. 312–327. Springer (2014). https://doi.org/10.1007/978-3-319-08434-3_23
11. Hullot, J.: Associative commutative pattern matching. In: Buchanan, B.G. (ed.) Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI 79, Tokyo, Japan, August 20-23, 1979, 2 Volumes. pp. 406–412. William Kaufmann (1979)
12. International Organization for Standardization: Information technology — Common Logic (CL) — a framework for a family of logic-based languages. International Standard ISO/IEC 24707:2018(E) (2018), available online at https://www.iso.org/standard/66249.html
13. Krebber, M., Barthels, H., Bientinesi, P.: Efficient pattern matching in python. In: Schreiber, A., Scullin, W., Spotz, B., Thomas, R. (eds.) Proc. 7th Workshop on Python for High-Performance and Scientific Computing. ACM (2017)
14. Kutsia, T.: Solving and proving in equational theories with sequence variables and flexible arity symbols. RISC Report Series 02-09, RISC, Johannes Kepler University Linz (2002)
15. Kutsia, T.: Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In: Calmet, J., Benhamou, B., Caprotti, O., Henocque, L., Sorge, V. (eds.) Artificial Intelligence, Automated Reasoning, and Symbolic Computation, Joint International Conferences, AISC 2002 and Calculemus 2002, Marseille, France, July 1-5, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2385, pp. 290–304. Springer (2002). https://doi.org/10.1007/3-540-45470-5_26
16. Kutsia, T.: Solving equations with sequence variables and sequence functions. Journal of Symbolic Computation **42**(3), 352–388 (2007). https://doi.org/10.1016/j.jsc.2006.12.002
17. Kutsia, T.: Flat matching. Journal of Symbolic Computation **43**(12), 858–873 (2008). https://doi.org/10.1016/j.jsc.2008.05.001
18. Kutsia, T., Marin, M.: Solving, reasoning, and programming in Common Logic. In: Voronkov, A., Negru, V., Ida, T., Jebelean, T., Petcu, D., Watt, S.M., Zaharie, D. (eds.) 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012. pp. 119–126. IEEE Computer Society (2012). https://doi.org/10.1109/SYNASC.2012.27
19. Kutsia, T., Marin, M.: Regular expression order-sorted unification and matching. J. Symb. Comput. **67**, 42–67 (2015). https://doi.org/10.1016/j.jsc.2014.08.002
20. Marin, M., Kutsia, T.: On the implementation of a rule-based programming system and some of its applications. In: Konev, B., Schmidt, R. (eds.) Proc. 4th International Workshop on the Implementation of Logics WIL'03. pp. 55–68 (2003)
21. Marin, M., Tepeneu, D.: Programming with sequence variables: The Sequentica package. In: Challenging The Boundaries Of Symbolic Computation. Proc. 5th International Mathematica Symposium, pp. 17–24. World Scientific (2003)
22. Wolfram, S.: The Mathematica Book. Wolfram Media, 5th edn. (2003)