A Constraint Solver for Equations over Sequences and Contexts

Mariam Beriashvili and Besik Dundua

Abstract In this paper we propose a solving algorithm for equational constraints over unranked terms, contexts, and sequences. Unranked terms are constructed over function symbols which do not have fixed arity. For some function symbols, the order of the arguments matters (ordered symbols). For some others, this order is irrelevant (unordered symbols). Contexts are unranked terms with a single occurrence of hole. Sequences consist of unranked terms and contexts. Term variables stand for single unranked terms, sequence variables for sequences, context variables for contexts, and function variables for function symbols. We design an terminated and incomplete constraint solving algorithm, and indicate a fragment for which the algorithm is complete.

1 Introduction

Unranked terms are built over function symbols which do not have a fixed arity (unranked symbols). They are nearly ubiquitous in XML-related applications [18]. They model variadic procedures used in programming languages [2, 22, 23]. Moreover, they appear in rewriting [10], knowledge representation [9, 20], theorem proving [12, 14], program synthesis [3], just to name a few.

When working with unranked terms, it is a pragmatic necessity to consider variables which can be instantiated by a finite sequences of terms (called sequences). Such variables are referred to as sequence variables. An example of an unranked term is $f(\overline{x}, f, x, \overline{y})$, where f is an unranked function symbol, $\overline{x}$ and $\overline{y}$ are sequence variables, and x is a usual term variable which can be instantiated by a single term. We can match this term, e.g., to the term $f(f, a, f, b)$ in two different ways, with the substitutions $\{\overline{x} \mapsto (\,), x \mapsto a, \overline{y} \mapsto (f, b)\}$ and $\{\overline{x} \mapsto (f, a), x \mapsto f, \overline{y} \mapsto b\}$, where $(\,)$ is the empty sequence and (f, a) is a sequence consisting of two terms f and a. Terms are singleton sequences.

M. Beriashvili · B. Dundua (✉)
Vekua Institute of Applied Mathematics, Tbilisi State University, 0183 Tbilisi, Georgia
e-mail: bdundua@gmail.com

T.B. Nguyen et al. (eds.), *Advanced Computational Methods for Knowledge Engineering*, Advances in Intelligent Systems and Computing 453, DOI 10.1007/978-3-319-38884-7_9

Sequences can be concatenated to each other. In this way, sequences can "grow horizontally" and sequence variables help explore it by filling gaps between siblings. However, such a concatenation has limited power, since it does not affect the depth of sequences, i.e., it does not permit sequences "to grow vertically". To address this problem, Bojańczyk and Walukiewicz [1] introduced forest algebras, where alongside sequences (thereby called forests), context also appears. Contexts are sequences with a single occurrence of the hole symbol placed in some leaf. Contexts can be composed by putting one of them in the hole of the other. Moreover, context can apply to a sequence by putting it into the hole, resulting in a sequence. One can introduce context variables to stand for such contexts, and function variables to stand for function symbols.

Reasoning about sequences gives rise to constraints which should be solved. This turned out to be quite a difficult task. Even if we consider unification problems, in the presence of sequence variables or context variables alone they are infinitary [13, 17]. They both generalize word unification [19]. Several finitary fragments and variants of context and sequence unification problems have been identified. Solving in a theory which combines both context and sequence variables is relatively less studied, with the exception of context sequence matching [15] and its application in rule-based programming [8].

We may have function symbols whose argument order does not matter (unordered symbols): A kind of generalization of the commutativity property to unranked terms. The programming language of Mathematica [23] is an example of successful application in programming of both syntactic and equational unranked pattern matching (including unordered matching) algorithms with sequence variables.

Various forms of constraint solving are in the center of declarative programming paradigms. Unification is the main computational mechanism for logic programming. Matching plays the same role in rule-based and functional programming. Constraints over special domains are in the heart of constraint logic programming languages.

In [7] we have studied a constraint solver for unranked sequences built over ordered and unordered function symbols. In this paper, we generalize this approach by combining contexts and unranked sequences in a single framework. Such a language is rich, possesses powerful means to traverse trees both horizontally and vertically in a single or multiple steps, and allows the user to naturally express data structures (e.g., trees, sequences, multisets) and to write code concisely. We propose a solving algorithm for constraints over terms, contexts, and sequences. The algorithm works on the input in disjunctive normal form and transforms it to the partially solved form. It is sound and terminating. The latter property naturally implies that the solver is incomplete for arbitrary constraints, because the problem it solves is infinitary: There might be infinitely many incomparable solutions to constraints that involve sequence and context variables, see, e.g., [11, 17]. However, there are fragments of constraints for which the solver is complete, i.e., it computes all the solutions. One of such fragments is the so called the well-moded fragment [7], where

variables on one side of equations (or in the left hand side of the membership atom) are guaranteed to be instantiated with ground expressions at some point. This effectively reduces constraint solving to sequence matching and context matching (which are known to be NP-complete [16, 21]), plus some early failure detection rules.

2 The Language

The alphabet $\mathcal{A}$ contains the sets of *term variables* $\mathcal{V}_\mathsf{T}$, *sequence variables* $\mathcal{V}_\mathsf{S}$, *function variables* $\mathcal{V}_\mathsf{F}$, *context variables* $\mathcal{V}_\mathsf{C}$, *unranked unordered function symbols* $\mathcal{F}_\mathsf{u}$ and *ordered function symbols* $\mathcal{F}_\mathsf{o}$. All these sets are assumed to be mutually disjoint. Henceforth, we shall assume that the symbols: x, y and z range over $\mathcal{V}_\mathsf{T}$; $\overline{x}, \overline{y}, \overline{z}$ over $\mathcal{V}_\mathsf{S}$; X, Y, Z over $\mathcal{V}_\mathsf{F}$; $X_\bullet, Y_\bullet, Z_\bullet$ over $\mathcal{V}_\mathsf{C}$; $f_\mathsf{u}, g_\mathsf{u}, h_\mathsf{u}$ over $\mathcal{F}_\mathsf{u}$ and $f_\mathsf{o}, g_\mathsf{o}, h_\mathsf{o}$ over $\mathcal{F}_\mathsf{o}$. Moreover, *function symbols* denoted by f, g, h are elements of the set $\mathcal{F} = \mathcal{F}_\mathsf{u} \cup \mathcal{F}_\mathsf{o}$, a *variable* is an element of the set $\mathcal{V} = \mathcal{V}_\mathsf{T} \cup \mathcal{V}_\mathsf{S} \cup \mathcal{V}_\mathsf{F} \cup \mathcal{V}_\mathsf{C}$ and a *functor* F is a common name for a function symbol or a function variable. The alphabet also contains the *special constant* $\bullet$, the *propositional constants* $\mathsf{true}, \mathsf{false}$, the *logical connectives* $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, the *quantifiers* $\exists, \forall$ and the *binary equality predicate* $\doteq$.

Definition 1 We define inductively *terms, sequences, contexts* and other syntactic categories over $\mathcal{A}$ as follows:

t	$::= x \mid F(S) \mid X_\bullet(t)$	Term
T	$::= t_1, \ldots, t_n \quad (n \geq 0)$	Term sequence
$\tilde{s}$	$::= t \mid \overline{x}$	Sequence element
S	$::= \tilde{s}_1, \ldots, \tilde{s}_n \quad (n \geq 0)$	Sequence
C	$::= \bullet \mid F(S, C, S) \mid X_\bullet(C)$	contexts

For readability, we put parentheses around sequences, writing, e.g., $(f(a), \overline{x}, b)$ instead of $f(a), \overline{x}, b$. The empty sequence is written as $()$. Besides the letter t, we use also r and s to denote terms. Two sequences are *disjoint* if they do not share a common element. For instance, $(f(a), x, b)$ and $(f(x), f(b, f(a)))$ are disjoint, whereas $(f(a), x, b)$ and $(f(b), f(a))$ are not.

A context C may be applied to a term t (resp. context C'), written $C[t]$ (resp. a context $C[C']$), and the result is the term (resp. context) obtained from C by replacing the hole $\bullet$ with t (resp. with C'). Besides the letter C, we use also D to denote contexts.

The set of terms is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and the set of contexts is denoted by $\mathcal{C}(\mathcal{F}, \mathcal{V})$.

Definition 2 A *formula* over the alphabet $\mathcal{A}$ is defined inductively as follows:

1. true and false are formulas.
2. If t and r are terms, then $t \doteq r$ is a formula.
3. If C and D are contexts, then $C \doteq D$ is a formula.

4. If $\mathbf{F}_1$ and $\mathbf{F}_2$ are formulas, then so are $(\neg\mathbf{F}_1)$, $(\mathbf{F}_1 \vee \mathbf{F}_2)$, $(\mathbf{F}_1 \wedge \mathbf{F}_2)$, $(\mathbf{F}_1 \Rightarrow \mathbf{F}_2)$, and $(\mathbf{F}_1 \Leftrightarrow \mathbf{F}_2)$.
5. If $\mathbf{F}$ is a formula and $v \in \mathcal{V}_{\mathsf{S}}$, then $\exists v.\mathbf{F}$ and $\forall v.\mathbf{F}$ are formulas.

The formulas defined by the items (2) and (3) are called *primitive constraints.* A *constraint* $\mathcal{C}$ is an arbitrary formula built over true, false and primitive constraints.

A *substitution* is a mapping from term variables to terms, from sequence variables to sequences, from function variables to functors, and from context variables to contexts, such that all but finitely many term, sequence, and function variables are mapped to themselves, and all but finitely many context variables are mapped to themselves applied to the hole. Substitutions extend to terms, sequences, contexts, formulas. The *sets of free and bound variables* of a formula $\mathbf{F}$, denoted $\mathtt{fvar}(\mathbf{F})$ and $\mathtt{bvar}(\mathbf{F})$ respectively, are defined in the standard way as can be seen in [6].

3 Semantics

For a given set S, we denote by S^* the set of finite, possibly empty, sequences of elements of S, and by S^n the set of sequences of length n of elements of S. Given a sequence $\bar{\mathsf{s}} = (\mathsf{s}_1, \mathsf{s}_2, \dots, \mathsf{s}_n) \in \mathsf{S}^n$, we denote by $perm(\mathsf{s})$ the set of sequences $\{(\mathsf{s}_{\pi(1)}, \mathsf{s}_{\pi(2)}, \dots, \mathsf{s}_{\pi(n)}) \mid \pi \text{ is a permutation of } \{1, 2, \dots, n\}\}$. The set of functions from a set S_1 to a set S_2 is denoted by $\mathsf{S}_1 \longrightarrow \mathsf{S}_2$. The notion $f : \mathsf{S}_1 \longrightarrow \mathsf{S}_2$ means that f belongs to $\mathsf{S}_1 \longrightarrow \mathsf{S}_2$.

A *structure* $\mathfrak{S}$ for a language $\mathcal{L}(\mathcal{A})$ is a tuple $\langle \mathcal{D}, \mathcal{I} \rangle$ made of a non-empty *carrier set* of individuals $\mathcal{D}$ and an interpretation function $\mathcal{I}$ that maps each function symbol $f \in \mathcal{F}$ to a function $\mathcal{I}(f) : \mathcal{D}^* \longrightarrow \mathcal{D}$. Moreover, if $f \in \mathcal{F}_{\mathsf{u}}$ then $\mathcal{I}(f)(\mathsf{s}) = \mathcal{I}(f)(\mathsf{s}')$ for all $\mathsf{s} \in D^*$ and $\mathsf{s}' \in perm(\mathsf{s})$. Given such a structure, we also define the operation $\mathcal{I}_{\mathsf{C}} : (\mathcal{D}^* \longrightarrow \mathcal{D}) \longrightarrow \mathcal{D}^* \longrightarrow \mathcal{D}^* \longrightarrow (\mathcal{D} \longrightarrow \mathcal{D}) \longrightarrow (\mathcal{D} \longrightarrow \mathcal{D})$ by $\mathcal{I}_{\mathsf{C}}(\psi)(\bar{\mathsf{s}}_1)(\bar{\mathsf{s}}_2)(\phi)(\mathsf{d}) := \psi(\bar{\mathsf{s}}_1, \phi(\mathsf{d}), \bar{\mathsf{s}}_2)$ for all $\psi : \mathcal{D}^* \longrightarrow \mathcal{D}$, $\bar{\mathsf{s}}_1, \bar{\mathsf{s}}_2 \in \mathcal{D}^*$, $\mathsf{d} \in \mathcal{D}$, and $\phi : \mathcal{D} \longrightarrow \mathcal{D}$.

A *variable assignment* for such a structure is a function with the domain $\mathcal{V}$ that maps term variables to elements of $\mathcal{D}$; sequence variable to elements of $\mathcal{D}^*$; function variables to functions in $\mathcal{D}^* \longrightarrow \mathcal{D}$; and context variables to functions in $\mathcal{D} \longrightarrow \mathcal{D}$.

The interpretations of our syntactic categories with respect to a structure $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ and variable assignment ρ is shown below. The *interpretation* of simple sequences $[\![S]\!]_{\mathfrak{S},\rho}$ and of contexts $[\![C]\!]_{\mathfrak{S},\rho}$ are defined as follows:

$$
\begin{aligned}
[\![x]\!]_{\mathfrak{S},\rho} &:= \rho(x).\\
[\![f(S)]\!]_{\mathfrak{S},\rho} &:= \mathcal{I}(f)([\![S]\!]_{\mathfrak{S},\rho}).\\
[\![X(S)]\!]_{\mathfrak{S},\rho} &:= \rho(X)([\![S]\!]_{\mathfrak{S},\rho}).\\
[\![X_\bullet(t)]\!]_{\mathfrak{S},\rho} &:= \rho(X_\bullet)([\![t]\!]_{\mathfrak{S},\rho}).\\
[\![\bar{x}]\!]_{\mathfrak{S},\rho} &:= \rho(\bar{x}).
\end{aligned}
$$

$$
\begin{aligned}
[\![(\tilde{s}_1, \ldots, \tilde{s}_n)]\!]_{\mathfrak{S},\rho} &:= ([\![\tilde{s}_1]\!]_{\mathfrak{S},\rho}, \ldots, [\![\tilde{s}_n]\!]_{\mathfrak{S},\rho}).\\
[\![\bullet]\!]_{\mathfrak{S},\rho} &:= Id_D.\\
[\![f(S_1, C, S_2)]\!]_{\mathfrak{S},\rho} &:= \mathcal{I}_C(\mathcal{I}(f))([\![S_1]\!]_{\mathfrak{S},\rho})([\![S_2]\!]_{\mathfrak{S},\rho})([\![C]\!]_{\mathfrak{S},\rho}).\\
[\![X(S_1, C, S_2)]\!]_{\mathfrak{S},\rho} &:= \mathcal{I}_C(\rho(X))([\![S_1]\!]_{\mathfrak{S},\rho})([\![S_2]\!]_{\mathfrak{S},\rho})([\![C]\!]_{\mathfrak{S},\rho}).\\
[\![X_\bullet(C)]\!]_{\mathfrak{S},\rho} &:= \rho(X_\bullet) \circledcirc [\![C]\!]_{\mathfrak{S},\rho}, \text{ where } \circledcirc \text{ stands for composition.}
\end{aligned}
$$

Note that terms are interpreted as elements of $\mathcal{D}$, sequences as elements of $\mathcal{D}^*$, and contexts as elements of $\mathcal{D} \longrightarrow \mathcal{D}$. We may omit ρ and write simply $[\![E]\!]_{\mathfrak{S}}$ for the interpretation of a variable-free (i.e., *ground*) expression E.

Formulas with respect to a structure $\mathfrak{S}$ and a variable assignment ρ are *interpreted* as follows:

$$
\begin{aligned}
&\mathfrak{S} \models_\rho \mathsf{true}.\\
&\text{Not } \mathfrak{S} \models_\rho \mathsf{false}.\\
&\mathfrak{S} \models_\rho t_1 \doteq t_2 \text{ iff } [\![t_1]\!]_{\mathfrak{S},\rho} = [\![t_2]\!]_{\mathfrak{S},\rho}.\\
&\mathfrak{S} \models_\rho C_1 \doteq C_2 \text{ iff } [\![C_1]\!]_{\mathfrak{S},\rho} = [\![C_2]\!]_{\mathfrak{S},\rho}.
\end{aligned}
$$

Interpretation of an arbitrary formula with respect to a structure and a variable assignment is defined in the standard way. Also, the notions $\mathfrak{S} \models \mathbf{F}$ for validity of an arbitrary formula $\mathbf{F}$ in $\mathfrak{S}$, and $\models \mathbf{F}$ for validity of $\mathbf{F}$ in any structure are defined as usual.

An *intended structure* is a structure $\mathfrak{I}$ with a carrier set $\mathcal{T}(\mathcal{F})$ (the set of ground simple terms) and interpretation $\mathcal{I}$ defined for every $f \in \mathcal{F}$ by $\mathcal{I}(f)(S) := f(S)$. It follows that $\mathcal{I}_C(\mathcal{I}(f))(S_1)(S_2)(C) := f(S_1, C, S_2)$. Thus, intended structures identify terms, sequences and contexts with themselves. Also, $[\![\mathbb{R}]\!]_{\mathfrak{I}}$ is the same in all intended structures, and will be denoted by $[\![\mathbb{R}]\!]$. Other remarkable properties of intended structures $\mathfrak{I}$ are: $\mathfrak{I} \models_\rho t_1 \doteq t_2$ iff $t_1\rho = t_2\rho$ and $\mathfrak{I} \models_\rho C_1 \doteq C_2$ iff $C_1\rho = C_2\rho$.

A ground substitution ρ is a *solution* of a constraint $\mathcal{C}$ if $\mathfrak{I} \models \mathcal{C}\rho$ for all intended structures $\mathfrak{I}$.

4 Solved and Partially Solved Constraints

We say a variable is *solved* in a conjunction of primitive constraints $\mathcal{K} = \mathbf{c}_1 \wedge \cdots \wedge \mathbf{c}_n$, if there is a $\mathbf{c}_i$, $1 \le i \le n$, such that

- the variable is x, $\mathbf{c}_i$ is $x \doteq t$, and x occurs neither in t nor elsewhere in $\mathcal{K}$, or
- the variable is $\overline{x}$, $\mathbf{c}_i$ is $\overline{x} \doteq S$, and $\overline{x}$ occurs neither in $\tilde{s}$ nor elsewhere in $\mathcal{K}$, or
- the variable is X, $\mathbf{c}_i$ is $X \doteq F$ and X occurs neither in F nor elsewhere in $\mathcal{K}$, or
- the variable is $X_\bullet$, $\mathbf{c}_i$ is $X_\bullet \doteq C$, and $X_\bullet$ occurs neither in C nor elsewhere in $\mathcal{K}$, or

In this case we also say that $\mathbf{c}_i$ is *solved in* $\mathcal{K}$. Moreover, $\mathcal{K}$ is called *solved* if for any $1 \leq i \leq n$, $\mathbf{c}_i$ is solved in it. $\mathcal{K}$ is *partially solved*, if for any $1 \leq i \leq n$, $\mathbf{c}_i$ is solved in $\mathcal{K}$, or has one of the following forms:

- $(\overline{x}, S_1) \doteq (\overline{y}, S_2)$ where $\overline{x} \neq \overline{y}$, $S_1 \neq ()$ and $S_2 \neq ()$.
- $(\overline{x}, S_1) \doteq (S, \overline{y}, S_2)$, where S is a sequence of terms, $\overline{x} \notin var(S)$, $S_1 \neq ()$, and $S \neq ()$. The variables $\overline{x}$ and $\overline{y}$ are not necessarily distinct.
- $f_u(S_1, \overline{x}, S_2) \doteq f_u(S_3, \overline{y}, S_4)$ where $(S_1, \overline{x}, S_2)$ and $(S_3, \overline{y}, S_4)$ are disjoint.
- $X_\bullet(t) \doteq r$ where $r \neq X_\bullet(t')$ contains term, context or sequence variables,
- $X_\bullet(C_1) \doteq C_2$ where $C_2 \neq X_\bullet(C_3)$ and C_2 is not strict.

A constraint is *solved*, if it is either true or a non-empty quantifier-free disjunction of solved conjunctions. A constraint is *partially solved*, if it is either true or a non-empty quantifier-free disjunction of partially solved conjunctions.

5 Solver

In this section we present a constraint solver. It is based on rules, transforming a constraint in *disjunctive normal form* (DNF) into a constraint in DNF. We say a constraint is in DNF, if it has a form $\mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$, where $\mathcal{K}$'s are conjunctions of true, false, and primitive constraints. The number of solver rules is not small (as it is usual for such kind of solvers, cf., e.g., [4, 5]). To make their comprehension easier, we group them so that similar ones are collected together in subsections. Within each subsection, for better readability, the rule groups are put between horizontal lines.

Before going into the details, we introduce a more conventional way of writing expressions, some kind of syntactic sugar, that should make reading easier. We write $F_1 \doteq F_2$ instead of $F_1() \doteq F_2()$, $X_\bullet \doteq C$ instead of $X_\bullet(\bullet) \doteq C$. The symmetric closure of $\doteq$ is denoted by $\simeq$.

5.1 Logical Rules

The logical rules perform logical transformations of the constraints and have to be applied in constraints, at any depth modulo associativity and commutativity of disjunction and conjunction. $\mathbf{F}$ stands for any formula. We denote the whole set of rules by Log.

$\mathbf{F} \wedge \mathbf{F} \rightsquigarrow \mathbf{F}$	$\mathsf{true} \wedge \mathbf{F} \rightsquigarrow \mathbf{F}$	$\mathsf{false} \wedge \mathbf{F} \rightsquigarrow \mathsf{false}$	$S \simeq S \rightsquigarrow \mathsf{true}$
$\mathbf{F} \vee \mathbf{F} \rightsquigarrow \mathbf{F}$	$\mathsf{true} \vee \mathbf{F} \rightsquigarrow \mathsf{true}$	$\mathsf{false} \vee \mathbf{F} \rightsquigarrow \mathbf{F}$	$C \simeq C \rightsquigarrow \mathsf{true}$

5.2 Failure Rules

In the second group there are rules for failure detection. The first two rules detect function symbol clash:

(F1) $f_1(S_1) \simeq f_2(S_2) \rightsquigarrow \mathsf{false}, \quad \text{if } f_1 \neq f_2.$

(F2) $f_1(S_1, C_1, S_2) \simeq f_2(S_3, C_2, S_4) \rightsquigarrow \mathsf{false}, \quad \text{if } f_1 \neq f_2.$

The next three rules perform occurrence check. Peculiarity of this operation for our language is that the variable occurrence into a term/context does not always leads to failure. For instance, the equation $x \doteq X_\bullet(x)$, where the variable x occurs in $X_\bullet(x)$, still has a solution $\{X_\bullet \mapsto \bullet\}$. Therefore, the occurrence check should fail on equations of the form *var* $\doteq$ *nonvar* only if no instance of the non-variable expression *nonvar* can become the variable *var*. To achieve this, the rules below require the non-variable terms to contain F (the first two rules) and t (the third rule), which can not be erased by a substitution application:

(F3) $x \simeq C[F(S)] \rightsquigarrow \mathsf{false}, \quad \text{if } x \in var(C[F(S)]).$

(F4) $X_\bullet \simeq C_1[F(C_2)] \rightsquigarrow \mathsf{false}, \quad \text{if } X_\bullet \in var(C_1[F(C_2)]).$

(F5) $\overline{x} \simeq (S_1, t, S_2) \rightsquigarrow \mathsf{false}, \quad \text{if } \overline{x} \in var((S_1, t, S_2)).$

Further, we have two more rules which lead to failure in the case when the hole is unified with a context whose all possible instances are nontrivial contexts (guaranteed by the presence of F), and when the empty sequence is attempted to match to an inherently nonempty sequence (guaranteed by t):

(F6) $\bullet \simeq C_1[F(C_2)] \rightsquigarrow \mathsf{false}.$ (F7) $(\,) \simeq (S_1, t, S_2) \rightsquigarrow \mathsf{false}.$

We denote this set of rules (F1)–(F7) by Fail.

5.3 Deletion Rules

There are five rules which delete identical terms, sequence variables or context variables from both sides of an equation. They are more or less self-explanatory. Just note that under unordered head, we delete an arbitrary occurrence of a term (that is not a sequence variable).

(Del1) $X_\bullet(t_1) \simeq X_\bullet(t_2) \wedge \rightsquigarrow t_1 \doteq t_2$.

(Del2) $X_\bullet(C_1) \simeq X_\bullet(C_2) \rightsquigarrow C_1 \doteq C_2$.

(Del3) $f_{\mathsf{u}}(S_1, S, S_2) \simeq f_{\mathsf{u}}(S_3, S, S_4) \rightsquigarrow f_{\mathsf{u}}(S_1, S_2) \doteq f_{\mathsf{u}}(S_3, S_4)$.

(Del4) $(\overline{x}, S_1) \simeq (\overline{x}, S_2) \rightsquigarrow S_1 \doteq S_2$.

(Del5) $\overline{x} \simeq (S_1, \overline{x}, S_2) \rightsquigarrow S_1 \doteq (\,) \wedge S_2 \doteq (\,)$,

In the last rule $\tilde{s}_1$ is not the empty sequence.

We denote the set of rules (Del1)–(Del5) by Del.

5.4 Decomposition Rules

Like the membership rules, each of the decomposition rules operates on a conjunction of constraint literals and gives back either a conjunction again, or a disjunction of conjunctions. These rules should be applied to disjuncts of constraints in DNF, to preserve the DNF structure.

(D1) $f_{\mathsf{o}}(S_1) \simeq f_{\mathsf{o}}(S_2) \rightsquigarrow S_1 \doteq S_2$.

(D2) $f_{\mathsf{u}}(S_1) \simeq f_{\mathsf{u}}(S_2) \wedge \mathcal{K} \rightsquigarrow \bigvee_{S' \in perm(S_2)} \big(S_1 \doteq S' \wedge \mathcal{K}\big)$,

where S_2 is a sequence of terms, S_1 and S_2 are disjoint.

(D3) $(t_1, S_1) \simeq (t_2, S_2) \rightsquigarrow t_1 \doteq t_2 \wedge S_1 \doteq S_2$,

where $S_1 \neq (\,)$ or $S_2 \neq (\,)$.

(D4) $f(S_1, C_1, S_2) \simeq f(S_3, C_2, S_4) \rightsquigarrow f(S_1, S_2) \doteq f(S_3, S_4) \wedge C_1 \doteq C_2$.

We denote the set of rules (D1)–(D4) by Dec.

5.5 Variable Elimination Rules

This set of rules eliminate variables from the given constraint, keeping only a single equation for them. The first four rules replace a variable with the corresponding expression, provided that the variable does not occur in the expression:

$$(\mathsf{E1}) \quad x \simeq t \wedge \mathcal{K} \rightsquigarrow x \doteq t \wedge \mathcal{K}\theta,$$

where $x \notin var(t)$, $x \in var(\mathcal{K})$ and $\theta = \{x \mapsto t\}$. If t is a variable then in addition it is required that $t \in var(\mathcal{K})$.

$$(\mathsf{E2}) \quad \overline{x} \simeq S \wedge \mathcal{K} \rightsquigarrow \overline{x} \doteq S \wedge \mathcal{K}\theta,$$

where $\overline{x} \notin var(S)$, $\overline{x} \in var(\mathcal{K})$, and $\theta = \{\overline{x} \mapsto S\}$. If $S = \overline{y}$ for some $\overline{y}$, then in addition it is required that $\overline{y} \in var(\mathcal{K})$.

$$(\mathsf{E3}) \quad X_\bullet \simeq C \wedge \mathcal{K} \rightsquigarrow X_\bullet \doteq C \wedge \mathcal{K}\theta,$$

where $X_\bullet \notin var(C)$, $X_\bullet \in var(\mathcal{K})$, and $\theta = \{X_\bullet \mapsto C\}$. If C has the form $Y_\bullet(\bullet)$, then in addition it is required that $Y_\bullet \in var(\mathcal{K})$.

$$(\mathsf{E4}) \quad X \simeq F \wedge \mathcal{K} \rightsquigarrow X \doteq F \wedge \mathcal{K}\theta,$$

where $X \neq F$, $X \in var(\mathcal{K})$, and $\theta = \{X \mapsto F\}$. If F is a function variable, then in addition it is required that $F \in var(\mathcal{K})$.

The rules (E5) and (E6) for sequence variable elimination assign to a variable an initial part of the sequence in the other side of the selected equation. The sequence has to be a sequence of terms in (E5). In (E6), only a split of the prefix of the sequence is relevant. The rest is blocked by the term t due to occurrence check: No instantiation of $\overline{x}$ can contain it.

$$(\mathsf{E5}) \quad (\overline{x}, S_1) \simeq S_2 \wedge \mathcal{K} \rightsquigarrow \bigvee_{S_2=(S',S'')} \Big(\overline{x} \doteq S' \wedge S_1\theta \doteq S'' \wedge \mathcal{K}\theta\Big)$$

where S_2 is a sequence of terms, $\overline{x} \notin var(S_2)$, $\theta = \{\overline{x} \mapsto S'\}$, and $S_1 \neq (\,)$.

$$(\mathsf{E6}) \quad (\overline{x}, S_1) \simeq (S, t, S_2) \wedge \mathcal{K} \rightsquigarrow \bigvee_{S=(S',S'')} \Big(\overline{x} \doteq S' \wedge S_1\theta \doteq (S'', t, S_2)\theta \wedge \mathcal{K}\theta\Big)$$

where S is a term sequence, $\overline{x} \notin var(S)$, $\overline{x} \in var(t)$, $\theta = \{\overline{x} \mapsto S'\}$, and $S_1 \neq (\,)$.

The rules (E7) and (E8) below can be seen as counterparts of (E5). In the rule (E8) we need conservative decomposition of contexts. Before giving those rules, we define the notion of conservativity.

We will speak about the *main path* of a context as the sequence of symbols (path) in its tree representation from the root to the hole. For instance, the main path in the context $f(X_{\bullet_1}(a), X(X_{\bullet_2}(b), g(\bullet)), \overline{x})$ is fXg, and in $f(X_{\bullet_1}(a), X(X_{\bullet_2}(b), X_{\bullet_3}(\bullet)), \overline{x})$ – $fXX_{\bullet_3}$. A context is called *strict* if its main path does not contain context variables. For instance, the context $f(X_{\bullet_1}(a), X(X_{\bullet_2}(b), g(\bullet)), \overline{x})$ is strict, while $f(X_{\bullet_1}(a), X(X_{\bullet_2}(b), X_{\bullet_3}(\bullet)), \overline{x})$ is not, because $X_{\bullet_3}$ is in its main path $fXX_{\bullet_3}$. We say that a context C is *decomposed* in two contexts C_1 and C_2 if $C = C_1[C_2]$.

We say that a context C is *conservative*, if for any instance $C\rho$ of C and for any decomposition $D_1[D_2]$ of $C\rho$ there exists a decomposition $C_1[C_2]$ of C such that $D_1 = C_1\rho$ and $D_2 = C_2\rho$. Strict contexts satisfy this property. Non-strict contexts violate it, as the following example shows: The context $C = X_\bullet(\bullet)$ has two decompositions into $C_1[C_2]$ with $C_1 = \bullet$, $C_2 = X_\bullet(\bullet)$ and $C_1 = X_\bullet(\bullet)$, $C_2 = \bullet$. Let $\rho = \{X_\bullet \mapsto f(g(\bullet))\}$. Then $C\rho = f(g(\bullet))$. One of its decomposition with $D_1 = f(\bullet)$, $D_2 = g(\bullet)$ is not an instance of any of the decompositions of C.

The rules (E7) and (E8) are formulated now as follows:

$$(\mathsf{E7}) \quad X_\bullet(t_1) \simeq t_2 \wedge \mathcal{K} \rightsquigarrow \vee_{t_2 = C[t]} \Big(X_\bullet \doteq C \wedge t_1\theta \doteq t \wedge \mathcal{K}\theta \Big),$$

where t_2 does not contain term, sequence, and context variables, $t_1 \neq \bullet$, and $\theta = \{X_\bullet \mapsto C\}$.

$$(\mathsf{E8}) \quad X_\bullet(C_1) \simeq C_2 \wedge \mathcal{K} \rightsquigarrow \vee_{C_2 = C[C']} \Big(X_\bullet \doteq C \wedge C_1\theta \doteq C'\theta \wedge \mathcal{K}\theta \Big),$$

where C_2 is strict, $X_\bullet \notin var(C)$, $C_1 \neq \bullet$, and $\theta = \{X_\bullet \mapsto C\}$.

Finally, there are two rules for function variable elimination. Their behavior is standard:

$$(\mathsf{E9}) \quad X(S_1) \simeq F(S_2) \wedge \mathcal{K} \rightsquigarrow X \doteq F \wedge F(S_1)\theta \doteq F(S_2)\theta \wedge \mathcal{K}\theta.$$

where $X \neq F$, $\theta = \{X \mapsto F\}$, and $S_1 \neq (\,)$ or $S_2 \neq (\,)$.

$$(\mathsf{E10}) \quad X(S_1) \simeq X(S_2) \wedge \mathcal{K} \rightsquigarrow \bigvee_{f \in \mathcal{F}} \Big(X \doteq f \wedge f(S_1)\theta \doteq f(S_2)\theta \wedge \mathcal{K}\theta \Big),$$

where $\theta = \{X \mapsto f\}$, and $S_1 \neq S_2$.

We denote the set of rules (E1)–(E10) by Elim.

The constraint solver rewrites a constraint with respect to the rules specified in this section into a constraint in partially solved form. First, we define how rewriting is done in a single step:

$$\mathsf{step} := \mathsf{first}(\mathsf{Log}, \mathsf{Fail}, \mathsf{Del}, \mathsf{Dec}, \mathsf{Elim}).$$

When step is applied to a constraint, it will transforms the constraint by the *first* successful rule from the sets Log, Fail, Del, Dec, and Elim. If none of the rules apply, then the constraint is said to be in a *normal form* with respect to step.

The constraint solving method implements the strategy solve which is defined as a repeatedly application of the step:

$$\mathsf{solve} := \mathsf{NF}(\mathsf{step}).$$

That means, step is applied to a constraint repeatedly as long as possible.

It remains to show that this definition yields an algorithm, which amounts to proving that a normal form is reached by $\mathsf{NF}(\mathsf{step})$ for any constraint $\mathcal{C}$.

6 Properties of the Constraint Solver

In this section, we present theorems and lemmata which demonstrate that the constraint solver is terminated, sound and partially complete. The proofs are omitted and can be easily obtained from the proofs of the similar theorems and lemmata given in [6].

The solver halts for any input constraint and a normal form is reached.

Theorem 1 solve *terminates on any input constraint.*

Here we state that the solver reduces a constraint to its equivalent constraint.

Lemma 1 *If* $\mathsf{step}(C) = \mathcal{D}$, *then* $\mathfrak{F} \models \forall\Big(C \Leftrightarrow \overline{\exists}_{var(C)}\mathcal{D}\Big)$ *for all intended structures* $\mathfrak{F}$.

Theorem 2 *If* $\mathsf{solve}(C) = \mathcal{D}$, *then* $\mathfrak{F} \models \forall\Big(C \Leftrightarrow \overline{\exists}_{var(C)}\mathcal{D}\Big)$ *for all intended structures* $\mathfrak{F}$, *and* $\mathcal{D}$ *is either partially solved or the* false *constraint.*

Theorem 3 *If the constraint* $\mathcal{D}$ *is solved, then* $\mathfrak{F} \models \exists\mathcal{D}$ *for all intended structures* $\mathfrak{F}$.

7 Well-Moded Constraints

A sequence of primitive constraints $\mathbf{c}_1, \ldots, \mathbf{c}_n$ is *well-moded* if the following conditions are satisfied:

1. If for some $1 \leq i \leq n$, $\mathbf{c}_i$ is $t_1 \doteq t_2$, then $var(t_1) \subseteq \bigcup_{j=1}^{i-1} var(\mathbf{c}_j)$ or $var(t_2) \subseteq \bigcup_{j=1}^{i-1} var(\mathbf{c}_j)$.
2. If for some $1 \leq i \leq n$, $\mathbf{c}_i$ is $C_1 \doteq C_2$, then $var(C_1) \subseteq \bigcup_{j=1}^{i-1} var(\mathbf{c}_j)$ or $var(C_2) \subseteq \bigcup_{j=1}^{i-1} var(\mathbf{c}_j)$.

A conjunction of primitive constraints $\mathcal{K}$ is well-moded if there exists a sequence of primitive constraints $\mathbf{c}_1, \ldots, \mathbf{c}_n$ which is well-moded and $\mathcal{K} = \bigwedge_{i=1}^{n} \mathbf{c}_i$ modulo associativity and commutativity of $\wedge$. A constraint $\mathcal{C} = \mathcal{K}_1 \vee \cdots \vee \mathcal{K}_n$ is well-moded if each $\mathcal{K}_i$, $1 \leq i \leq n$, is well-moded.

The following Theorem states, that, the solver brings any well-moded constraints to a solved form or to false.

Lemma 2 *Let* C *be a well-moded constraint and* $\mathsf{step}(C) = C'$, *then* C' *is either well-moded,* true *or* false.

Theorem 4 *Let C be a well-moded constraint and* $\mathsf{solve}(C) = C'$*, where* $C' \neq \mathsf{false}$*. Then C' is solved.*

Acknowledgments Besik Dundua has been partially supported by the Shota Rustaveli National Science Foundation under the grants FR/325/4-120/14, YS/10/11-811/15 and YS15_2.1.2_70.

References

1. Bojanczyk, M., Walukiewicz, I.: Forest algebras. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata. Texts in Logic and Games, vol. 2, pp. 107–132. Amsterdam University Press (2008)
2. Boley, H.: A Tight, Practical Integration of Relations and Functions. Lecture Notes in Computer Science, vol. 1712. Springer, Berlin (1999)
3. Chasseur, E., Deville, Y.: Logic program schemas, constraints, and semi-unification. In: Fuchs, N.E. (ed.) LOPSTR. Lecture Notes in Computer Science, vol. 1463, pp. 69–89. Springer, Berlin (1997)
4. Comon, H.: Completion of rewrite systems with membership constraints. Part II: constraint solving. J. Symb. Comput. **25**(4), 421–453 (1998)
5. Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. **22**(5), 861–931 (2000)
6. Dundua, B.: Programming with Sequence and Context Variables:Foundations and Applications. Ph.D. thesis, Universidade do Porto (2014)
7. Dundua, B., Florido, M., Kutsia, T., Marin, M.: Constraint logic programming for hedges: A semantic reconstruction. In: Codish, M., Sumii, E. (eds.) Functional and Logic Programming—12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4—6, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8475, pp. 285–301. Springer, Switzerland (2014)
8. Dundua, B., Kutsia, T., Marin, M.: Strategies in Pρlog. In: Fernández, M. (ed.) WRS, EPTCS, vol. 15, pp. 32–43 (2009)
9. ISO/IEC. Information technology—Common Logic (CL): a framework for a family of logic-based languages. International Standard ISO/IEC 24707 (2007). http://standards.iso.org/ittf/PubliclyAvailableStandards/c039175_ISO_IEC_24707_2007(E).zip
10. Jacquemard, F., Rusinowitch, M.: Closure of hedge-automata languages by hedge rewriting. In: Voronkov, A. (ed.) RTA. LNCS, vol. 5117, pp. 157–171. Springer, Berlin (2008)
11. Kutsia, T.: Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols. RISC report Series 02–09, Research Institute for Symbolic Computation (RISC), University of Linz, Schloss Hagenberg, 4232 Hagenberg, Austria, May 2002. Ph.D. thesis
12. Kutsia, T.: Theorem proving with sequence variables and flexible arity symbols. In: Baaz, M., Voronkov, A. (eds.) LPAR. Lecture Notes in Computer Science, vol. 2514, pp. 278–291. Springer, Berlin (2002)
13. Kutsia, T.: Solving equations with sequence variables and sequence functions. J. Symb. Comput. **42**(3), 352–388 (2007)
14. Kutsia, T., Buchberger, B.: Predicate logic with sequence variables and sequence function symbols. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM. Lecture Notes in Computer Science, vol. 3119, pp. 205–219. Springer, Berlin (2004)
15. Kutsia, T., Marin, M.: Matching with regular constraints. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR. Lecture Notes in Computer Science, vol. 3835, pp. 215–229. Springer, Berlin (2005)
16. Kutsia, T., Marin, M.: Solving, reasoning, and programming in common logic. In: SYNASC, pp. 119–126. IEEE Computer Society (2012)
17. Levy, J.: Linear second-order unification. In: Ganzinger, H. (ed.) RTA. Lecture Notes in Computer Science, vol. 1103, pp. 332–346. Springer, Berlin (1996)

18. Libkin, L.: Logics for unranked trees: an overview. Logical Methods Comput. Sci. **2**(3) (2006)
19. Makanin, G.S.: The problem of solvability of equations in a free semigroup. Math. USSR-Sb. **32**(2), 129, 147–236 (1977)
20. Menzel, C.: Knowledge representation, the world wide web, and the evolution of logic. Synthese **182**(2), 269–295 (2011)
21. Schmidt-Schauß, M., Stuber, J.: The complexity of linear and stratified context matching problems. Theor. Comput. Syst. **37**(6), 717–740 (2004)
22. Wand, M.: Complete type inference for simple objects. In: LICS, pp. 37–44. IEEE Computer Society (1987)
23. Wolfram, S.: The Mathematica Book. Wolfram-Media, 5th edn. (2003)