Unification with Sequence Variables and Flexible Arity Symbols and its Extension with Pattern-Terms[⋆]

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University Linz
A-4040, Linz, Austria
kutsia@risc.uni-linz.ac.at

Software Competence Center Hagenberg
Hauptstrasse, 99
A-4232, Hagenberg, Autsria
teimuraz.kutsia@scch.at

Abstract. A minimal and complete unification procedure for a theory with individual and sequence variables, free constants and free fixed and flexible arity function symbols is described and a brief overview of an extension with pattern-terms is given.

1 Introduction

We design a unification procedure for a theory with individual and sequence variables, free constants and free fixed and flexible arity function symbols. The subject of this research was proposed by B. Buchberger in [4] and in a couple of personal discussions [5]. The research described in this paper is a part of the author's PhD thesis.

We refer to unification in a theory with individual and sequence variables, free constants and free fixed and flexible arity function symbols shortly as unification with sequence variables and flexible arity symbols, underlining the importance of these two constructs. Sequence variables are variables which can be instantiated by an arbitrary finite (possibly empty) sequence of terms. Flexible arity function symbols can take arbitrary finite (possibly empty) number of arguments. In the literature the symbols with similar property are also referred to as "variable arity", "variadic" or "multiple arity" symbols. Languages with sequence variables and variable arity symbols have been used in various areas. Here we enumerate some of them:

– Knowledge management - Knowledge Interchange Format KIF ([10]) and its version SKIF ([23]) are extensions of first order language with (among other constructs) individual and sequence variables and variable arity function symbols. KIF is used to interchange knowledge among disparate computer systems. Another example of using sequence variables and variable arity symbols in knowledge systems is Ontolingua ([8]) - a tool which provides a distributed collaborative environment to browse, create, edit, modify, and use ontologies.

⋆ Supported by the Austrian Science Foundation (FWF) under Project SFB F1302 and by Software Competence Center Hagenberg (Austria) under MathSoft project.

– Databases - sequences and sequence variables provide flexibility in data representation and manipulation for genome or text databases, where much of the data has an inherently sequential structure. Numerous formalisms involving sequences and sequence variables, like Sequence Logic ([11]), Alignment Logic ([12]), Sequence Datalog ([20]), String Calculus ([13],[3]), have been developed for this field.
– Rewriting - variable arity symbols used in rewriting usually come from flattening terms with associative top function symbol. Sequences and sequence variables (sometimes called also patterns), which are used together with variable arity symbols, make the syntax more flexible and expressive, and increase the performance of a rewriting system (see [29], [14]).
– Programming languages - variable arity symbols are supported by many of them. The programming language of Mathematica ([30]) is one of such examples, which uses the full expressive power of sequence variables as well. A relation of Mathematica programming language and rewrite rule languages, and the role of sequence variables in this relation is discussed in [4].
– Theorem proving - the Epilog package ([9]) can be used in programs that manipulate information encoded in Standard Information Format (SIF) - a subset of KIF ([10]) language, containing sequence variables and variable arity symbols. Among the other routines, Epilog includes pattern matchers of various sorts, and an inference procedure based on model elimination.

These applications involve (and in some cases, essentially depend on) solving equations with sequence variables and variable arity symbols. The most used solving technique is matching. However, for some applications, like theorem proving or completion, more powerful solving techniques (unification, for instance) are needed.

The problem whether Knuth-Bendix completion procedure ([16]) can be extended to handle term rewriting systems with function symbols of variable arity, sequences and sequence variables (patterns) is stated as an open problem in [29]. The primary reason why it is an open problem is the absence of appropriate unification algorithm.

In this paper, we make the first step towards solving this problem, providing a unification procedure with individual and sequence variables, fixed and flexible arity function symbols and its extension with pattern-terms. Sequence variables and pattern-terms can be seen as particular examples of the pattern construct of [29]. The term "flexible arity" was suggested by Buchberger ([5]) instead of "variable arity", mainly because of the following reason: variable arity symbols, as they are understood in theorem proving or rewriting, are flattened associative symbols, i.e. flat symbols which take at least two arguments, while flexible arity symbols can have zero or one argument as well and are not necessarily flat. Nonflatness is one of main differences between unification with sequence variables and flexible arity symbols and associative unification: the unification problem $f(x, f(y, z)) \overset{?}{=} f(f(a, b), c)$, with the variables x, y, z and constants a, b, c, has no unifier, if f has a flexible arity, but admits a unifier $\{x \leftarrow a, y \leftarrow b, z \leftarrow c\}$ for associative f. Even when f is a flat flexible arity symbol the problem would not be equivalent to A-unification: the substitution $\{x \leftarrow f(a), y \leftarrow f(b,c), z \leftarrow f()\}$ is a unifier for a flat f, but not for an associative f.

The type of unification with sequence variables and flexible arity symbols in the Siekmann unification hierarchy ([28]) is infinitary: for any unification problem there exists the minimal complete set of unifiers which is infinite for some problems.

It should be mentioned that in the theorem proving context quantification over sequence variables naturally introduces flexible arity symbols and constructs that we call patterns. For instance, Skolemizing the expression $\forall \overline{x} \exists y \Phi[\overline{x}, y]$, where $\overline{x}$ is a sequence variable, y is an individual variable and $\Phi[\overline{x}, y]$ is a formula which depends on $\overline{x}$ and y, introduces a flexible arity Skolem function f: $\forall \overline{x} \Phi[\overline{x}, f(\overline{x})]$. On the other hand, Skolemizing the expression $\forall x \exists \overline{y} \Phi[x, \overline{y}]$ introduces a pattern $h_{1,n(x)}(x)$, which can be seen as an abbreviation of a sequence of terms $h_1(x), \ldots, h_{n(x)}(x)$ of unknown length, where $h_1, \ldots, h_{n(x)}$ are Skolem functions.

The procedure we describe can be used in the theorem proving context in the way similar to [24]: building in equational theories. Although unification with sequence variables and flexible arity symbols is infinitary, special cases can be identified when the procedure terminates.

It is shown in [17] that unification with sequence variables and flexible arity symbols is decidable. Based on the decision procedure, a constraint-based approach to theorem proving with sequence variables and flexible arity symbols can be developed (compare [22], [25]).

Particular instances of unification with sequence variables and flexible arity symbols are word equations ([1],[15], [26]), equations over free semigroups ([19]), equations over lists of atoms with concatenation ([7]), pattern matching.

We have implemented the unification procedure (without decision algorithm) as a Mathematica package and incorporated it into the Theorema system [6], which aims at extending computer algebra systems by facilities for supporting mathematical proving. Currently the package is used in the Theorema Equational Prover. It makes Theorema probably the only system being able to handle equations which involve sequence variables and flexible arity symbols. The package also enhances Mathematica solving capabilities, considering unification as a solving method. We used the package, for instance, to find matches for S-polynomials in non-commutative Gröbner Bases algorithm [21].

The results in this paper are given without proofs. They can be found in [17].

2 Preliminaries

We consider an alphabet consisting of the following pairwise disjoint sets of symbols: the set of individual variables $\mathcal{IV}$, the set of sequence variables $\mathcal{SV}$, the set of object constants $\mathcal{CONST}$, the set of fixed arity function symbols $FFIX$, the set of flexible arity function symbols $\mathcal{FFLEX}$ and a singleton consisting of a binary predicate symbol $\doteq$ (equality).

Let now $\mathcal{V}$ stand for $(\mathcal{IV}, \mathcal{SV})$ (variables), $\mathcal{C}$ - for $(\mathcal{CONST}, \mathcal{FFIX}, \mathcal{FFLEX}, \doteq)$ (a domain of constants) and $\mathcal{P}$ - for $\{(,), ,\}$ ("parentheses and comma"). We define terms and equations over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$.

Definition 1 (Term). *The set of terms (over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$) is the smallest set of strings over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$ that satisfies the following conditions:*

- *If $t \in \mathcal{IV} \cup \mathcal{SV} \cup \mathcal{CONST}$ then t is a term.*
- *If $f \in \mathcal{FFIX}$, f is n-ary, $n \geq 0$ and $t_1, \ldots, t_n$ are terms such that for all $1 \leq i \leq n$, $t_i \notin \mathcal{SV}$, then $f(t_1, \ldots, t_n)$ is a term.*
- *If $f \in \mathcal{FFLEX}$ and $t_1, \ldots, t_n$ $(n \geq 0)$ are terms, then so is $f(t_1, \ldots, t_n)$.*

f is called the head of $f(t_1, \ldots, t_n)$.

Definition 2 (Equation). *The set of equations (over the alphabet $(\mathcal{V}, \mathcal{C}, \mathcal{P})$) is the smallest set of strings over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$ that satisfies the following condition:*

- *If t_1 and t_2 are terms over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$ such that $t_1 \notin \mathcal{SV}$ and $t_2 \notin \mathcal{SV}$, then $\doteq(t_1, t_2)$ is an equation over $(\mathcal{V}, \mathcal{C}, \mathcal{P})$. $\doteq$ is called the head of $\doteq(t_1, t_2)$.*

If not otherwise stated, the following symbols, with or without indices, are used as metavariables: x, y and z - over individual variables, $\overline{x}$, $\overline{y}$ and $\overline{z}$ - over sequence variables, v and u - over (individual or sequence) variables, c - over object constants, f, g and h - over (fixed or flexible arity) function symbols, s and t - over terms. We generalize standard notions of unification theory ([2]) for a theory with sequence variables and flexible arity symbols.

Definition 3 (Substitution). *A substitution is a finite set $\{x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n, \overline{x}_1 \leftarrow t_1^1, \ldots, t_{k_1}^1, \ldots, \overline{x}_m \leftarrow t_1^m, \ldots, t_{k_m}^m\}$ where*

- *$n \geq 0$, $m \geq 0$ and for all $1 \leq i \leq m$, $k_i \geq 0$,*
- *$x_1, \ldots, x_n$ are distinct individual variables,*
- *$\overline{x}_1, \ldots, \overline{x}_m$ are distinct sequence variables,*
- *for all $1 \leq i \leq n$, s_i is a term, $s_i \notin \mathcal{SV}$ and $s_i \neq x_i$,*
- *for all $1 \leq i \leq m$, $t_1^i, \ldots, t_{k_i}^i$ is a sequence of terms and if k_i=1 then $t_{k_i}^i \neq \overline{x}_i$.*

Greek letters are used to denote substitutions. The empty substitution is denoted by ε.

Definition 4 (Instance). *Given a substitution θ, we define an instance of a term or equation with respect to θ recursively as follows:*

- $x\theta = \begin{cases} s \text{ if } x \leftarrow s \in \theta, \\ x \text{ otherwise} \end{cases}$
- $\overline{x}\theta = \begin{cases} s_1, \ldots, s_m \text{ if } \overline{x} \leftarrow s_1, \ldots, s_m \in \theta,\ m \geq 0, \\ \overline{x} \qquad\qquad \text{otherwise} \end{cases}$
- $f(s_1, \ldots, s_n)\theta = f(s_1\theta, \ldots, s_n\theta)$
- $(s_1 \doteq s_2)\theta = s_1\theta \doteq s_2\theta.$

We extend the notion of instance to sequences in a straightforward way - instance of a sequence is a sequence of instances.

Definition 5 (Composition of Substitutions). *Let $\theta = \{x_1 \leftarrow s_1, \dots, x_n \leftarrow s_n, \overline{x_1} \leftarrow t^1_1, \dots, t^1_{k_1}, \dots, \overline{x_m} \leftarrow t^m_1, \dots, t^m_{k_m}\}$ and $\lambda = \{y_1 \leftarrow d_1, \dots, y_n \leftarrow d_l, \overline{y_1} \leftarrow e^1_1, \dots, e^1_{q_1}, \dots, \overline{y_r} \leftarrow e^r_1, \dots, e^r_{q_r}\}$ be two substitutions. Then the composition of θ and λ is the substitution, denoted by $\theta \circ \lambda$, obtained from the set*

$$\begin{aligned}\{\, & x_1 \leftarrow s_1\lambda, \dots, x_n \leftarrow s_n\lambda, \overline{x_1} \leftarrow t^1_1\lambda, \dots, t^1_{k_1}\lambda, \dots, \overline{x_m} \leftarrow t^m_1\lambda, \dots, t^m_{k_m}\lambda, \\ & y_1 \leftarrow d_1, \dots, y_l \leftarrow d_l, \overline{y_1} \leftarrow e^1_1, \dots, e^1_{q_1}, \dots, \overline{y_r} \leftarrow e^r_1, \dots, e^r_{q_r}\}\end{aligned}$$

by deleting

- *all the elements $x_i \leftarrow s_i\lambda$ ($1 \le i \le n$) for which $x_i = s_i\lambda$,*
- *all the elements $\overline{x_i} \leftarrow t^i_1\lambda, \dots, t^i_{k_i}\lambda$ ($1 \le i \le m$) for which $k_i = 1$ and $\overline{x}_i = t^i_1\lambda$,*
- *all the elements $y_i \leftarrow d_i$ ($1 \le i \le l$) such that $y_i \in \{x_1, \dots, x_n\}$,*
- *all the elements $\overline{y_i} \leftarrow e^i_1, \dots, e^i_{q_i}$ ($1 \le i \le r$) such that $\overline{y_i} \in \{\overline{x_1}, \dots, \overline{x_m}\}$.*

Example 1. Let $\theta = \{x \leftarrow f(y),\ \overline{x} \leftarrow \overline{y}, \overline{x},\ \overline{y} \leftarrow \overline{y}, \overline{z}\}$ and $\lambda = \{y \leftarrow g(c, c),\ \overline{x} \leftarrow c,\ \overline{z} \leftarrow\}$. Then $\theta \circ \lambda = \{x \leftarrow f(g(c, c)),\ y \leftarrow g(c, c),\ \overline{x} \leftarrow \overline{y}, c,\ \overline{z} \leftarrow\}$.

These versions of the notions of substitution, composition, and instance have the same important properties as the standard versions of the same notions:

Theorem 1. *For a term t and substitutions θ and λ $t\theta \circ \lambda = t\theta\lambda$.*

Theorem 2. *For any substitutions θ, λ and σ, $(\theta \circ \lambda) \circ \sigma = \theta \circ (\lambda \circ \sigma)$.*

3 Equational Theory with Sequence Variables and Flexible Arity Symbols

A set of equations E (called representation) defines an equational theory, i.e. the equality of terms induced by E. We use the term E-theory for the equational theory defined by E. We will write $s \doteq_E t$ for $s \doteq t$ modulo E. Solving equations in an E-theory is called E-unification. The fact that the equation $s \doteq_E t$ has to be solved is written as $s \overset{?}{\doteq}_E t$. A finite system of equations $\langle s_1 \overset{?}{\doteq}_E t_1, \dots, s_n \overset{?}{\doteq}_E t_n\rangle$ is called an E-unification problem. Some examples of E-theories are:

1. Free theory ($\emptyset$): $E = \emptyset$;
2. Flat theory (F): $E = \{f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z})\}$.
3. Restricted flat theory (RF): $E = \{f(\overline{x}, f(\overline{y_1}, x, \overline{y_2}), \overline{z}) \doteq f(\overline{x}, \overline{y_1}, x, \overline{y_2}, \overline{z})\}$.
4. Orderless theory (O): $E = \{f(\overline{x}, x, \overline{y}, y, \overline{z}) \doteq f(\overline{x}, y, \overline{y}, x, \overline{z})\}$.
5. FO: $E = \{f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z}), f(\overline{x}, x, \overline{y}, y, \overline{z}) \doteq f(\overline{x}, y, \overline{y}, x, \overline{z})\}$.
6. RFO:

$$E = \{f(\overline{x}, f(\overline{y_1}, x, \overline{y_2}), \overline{z}) \doteq f(\overline{x}, \overline{y_1}, x, \overline{y_2}, \overline{z}), f(\overline{x}, x, \overline{y}, y, \overline{z}) \doteq f(\overline{x}, y, \overline{y}, x, \overline{z})\}.$$

Definition 6 (Unifier). *A substitution θ is called an E-unifier of an E-unification problem $\langle s_1 \overset{?}{\doteq}_E t_1, \dots, s_n \overset{?}{\doteq}_E t_n\rangle$ iff $s_i\theta \doteq_E t_i\theta$ for all $1 \le i \le n$.*

Definition 7 (More General Substitution). *A substitution θ is more general than a substitution σ on a finite set of variables Var modulo a theory E (denoted $\theta \ll_E^{Var} \sigma$) iff there exists a substitution λ such that*

- *for all $\overline{x} \in Var$,*
  - $\overline{x} \leftarrow \ \notin \lambda$;
  - *there exist terms $t_1, \ldots, t_n, s_1, \ldots, s_n$, $n \geq 0$ such that $\overline{x}\sigma = t_1, \ldots, t_n$, $\overline{x}\theta \circ \lambda = s_1, \ldots, s_n$ and for each $1 \leq i \leq n$, either t_i and s_i are the same sequence variables or $t_i \doteq_E s_i$;*
- *for all $x \in Var$, $x\sigma \doteq_E x\theta \circ \lambda$.*

Example 2. $\{\overline{x} \leftarrow \overline{y}\} \ll_\emptyset^{\{\overline{x},\overline{y}\}} \{\overline{x} \leftarrow a, \overline{z},\ \overline{y} \leftarrow a, \overline{z}\}$, but not $\{\overline{x} \leftarrow \overline{y}\} \ll_\emptyset^{\{\overline{x},\overline{y}\}} \{\overline{x} \leftarrow, \overline{y} \leftarrow\}$.

Definition 8 (The Minimal Complete Set of Unifiers). *The minimal complete set of E-unifiers of Γ, denoted $MCU_E(\Gamma)$, is an E-minimal set of substitutions with respect to the set of variables Var of Γ, satisfying the following conditions:*

E-Correctness - for all $\theta \in MCU_E(\Gamma)$, θ is an E-unifier of Γ.
E-Completeness - for any E-unifier σ of Γ there exists $\theta \in MCU_E(\Gamma)$ such that $\theta \ll_E^{Var} \sigma$.
E-minimality - for all θ, $\sigma \in MCU_E(\Gamma)$, $\theta \ll_E^{Var} \sigma$ implies $\theta = \sigma$.

Example 3. Compute the minimal complete set of unifiers in the $\emptyset$, F and RF theories (f and g are free flexible arity symbols, h is flat, rh - restricted flat):

1. $MCU_\emptyset(\langle f(\overline{x}, a) \overset{?}{\doteq}_\emptyset f(a, \overline{x})\rangle) = \{\{\overline{x} \leftarrow\},\ \{\overline{x} \leftarrow a\},\ \{\overline{x} \leftarrow a, a\}, \ldots\}$.
2. $MCU_\emptyset(\langle f(g(a, g(\overline{y}, c)), \overline{x}) \overset{?}{\doteq}_\emptyset f(\overline{u},\ g(b, \overline{v}))\rangle) = \{\{\overline{u} \leftarrow g(a, \overline{x}),\ \overline{y} \leftarrow b,\ \overline{v} \leftarrow c\},\ \{\overline{x} \leftarrow, \overline{u} \leftarrow g(a), \overline{y} \leftarrow b, \overline{v} \leftarrow c\},\ \{\overline{u} \leftarrow g(a, \overline{x}), \overline{y} \leftarrow b, \overline{y},\ \overline{v} \leftarrow \overline{y}, c\},\ \{\overline{x} \leftarrow,\ \overline{u} \leftarrow g(a),\ \overline{y} \leftarrow b, \overline{y},\ \overline{v} \leftarrow \overline{y}, c\}\}$.
3. $MCU_F(\langle x \overset{?}{\doteq}_F h(x)\rangle) = \{\{x \leftarrow h(x)\}\}$.
4. $MCU_F(\langle h(\overline{x}) \overset{?}{\doteq}_F h(a)\rangle) = \{\{\overline{x} \leftarrow a\}, \{\overline{x} \leftarrow h(a)\}, \{\overline{x} \leftarrow a, h()\},\ \{\overline{x} \leftarrow h(a), h()\},\ \{\overline{x} \leftarrow h(), a\},\ \{\overline{x} \leftarrow h(), h(a)\},\ \{\overline{x} \leftarrow h(), a, h()\},\ \ldots\}$.
5. $MCU_{RF}(\langle rh(\overline{x}) \overset{?}{\doteq}_{RF} rh(a)\rangle) = \{\{\overline{x} \leftarrow a\}, \overline{x} \leftarrow rh(a)\}$.

Below in this paper we consider only the $\emptyset$-theory, although the results that are valid for arbitrary E-theories are formulated in a general setting.

4 General Unification Procedure in the Free Theory with Sequence Variables and Flexible Arity Symbols

In this section we design a unification procedure to solve general unification problem of the form $t_1 \overset{?}{\doteq}_\emptyset t_2$[1], built over the alphabet which consists of sequence

[1] In the case of $\emptyset$-theory it is enough to consider single equations instead of systems of equations in unification problems, because $\langle s_1 \overset{?}{\doteq}_\emptyset t_1, \ldots, s_n \overset{?}{\doteq}_\emptyset t_n\rangle$ has the same set of unifiers as $f(s_1, \ldots, s_n) \overset{?}{\doteq}_\emptyset f(t_1, \ldots, t_n)$, where f is a free flexible arity symbol.

and individual variables, free flexible arity function symbols, free constants and free fixed arity function symbols. We denote it as $GUP_\emptyset$. The unification procedure is a tree generation process based on two basic steps: projection and transformation.

4.1 Projection

The idea of projection ([1]) is to eliminate some sequence variables from the given unification problem UP. Let $\Pi(UP)$ be the following set of substitutions: $\{\{\overline{x} \leftarrow \ \mid \overline{x} \in S\} \mid S \subseteq vars(UP) \cap \mathcal{SV}\}$, where $vars(UP)$ is a set of variables of UP. $\Pi(UP)$ is called the set of projecting substitutions for UP. Each $\pi \in \Pi$ replaces some sequence variables from UP with the empty sequence. The projection rule is shown in Figure 1.

Projection: $s \overset{?}{=}_\emptyset t \rightsquigarrow \langle\langle s\pi_1 \overset{?}{=}_\emptyset t\pi_1,\ \pi_1\rangle, \ldots,$ $\langle s\pi_k \overset{?}{=}_\emptyset t\pi_k,\ \pi_k\rangle\rangle$ where $\{\pi_1, \ldots, \pi_k\} = \Pi(s \overset{?}{=}_\emptyset t)$.

Fig. 1. Projection rule.

4.2 Transformation

Each of the transformation rules for unification have one of the following forms: $UP \rightsquigarrow \bot$ or $UP \rightsquigarrow \langle\langle SUC_1, \sigma_1\rangle, \ldots, \langle SUC_n, \sigma_n\rangle\rangle$ where each of the successors SUC_i is either $\top$ or a new unification problem.

The full set of transformation rules are given on Figure 2. It consists of four family of rules: Success, Failure, Elimination and Splitting. Note the usage of widening techniques (similar to [18],[24],[27],[28]) in the elimination rules for sequence variables.

4.3 Unification Procedure - Tree Generation

In [15] and [26] tree generation construction is used for solving word equations. We use the similar idea for unification procedure with sequence variables and flexible arity symbols. Projection and transformation can be seen as single steps in a tree generation process. Each node of the tree is labeled either with a unification problem, $\top$ or $\bot$. The edges of the tree are labeled by substitutions. The nodes labeled with $\top$ or $\bot$ are terminal nodes. The nodes labeled with unification problems are non-terminal nodes. The children of a non-terminal node are constructed in the following way:

Given a nonterminal node, let UP be a unification problem attached to it. First, we decide whether UP is unifiable. If the answer is negative, we replace UP

Success:	$t \overset{?}{=}_\emptyset t \rightsquigarrow \langle\langle \top,\ \varepsilon \rangle\rangle.$	
	$x \overset{?}{=}_\emptyset t \rightsquigarrow \langle\langle \top,\ \{x \leftarrow t\} \rangle\rangle,$	if $x \notin vars(t)$.
	$t \overset{?}{=}_\emptyset x \rightsquigarrow \langle\langle \top,\ \{x \leftarrow t\} \rangle\rangle,$	if $x \notin vars(t)$.
Failure:	$c_1 \overset{?}{=}_\emptyset c_2 \rightsquigarrow \bot,$	if $c_1 \neq c_2$.
	$x \overset{?}{=}_\emptyset t \rightsquigarrow \bot,$	if $t \neq x$ and $x \in vars(t)$.
	$t \overset{?}{=}_\emptyset x \rightsquigarrow \bot,$	if $t \neq x$ and $x \in vars(t)$.
	$f_1(\tilde{t}) \overset{?}{=}_\emptyset f_2(\tilde{s}) \rightsquigarrow \bot,$	if $f_1 \neq f_2$.
	$f() \overset{?}{=}_\emptyset f(t_1, \tilde{t}) \rightsquigarrow \bot.$	
	$f(t_1, \tilde{t}) \overset{?}{=}_\emptyset f() \rightsquigarrow \bot.$	
	$f(\overline{x}, \tilde{t}) \overset{?}{=}_\emptyset f(s_1, \tilde{s}) \rightsquigarrow \bot,$	if $s_1 \neq \overline{x}$ and $\overline{x} \in vars(s_1)$.
	$f(s_1, \tilde{s}) \overset{?}{=}_\emptyset f(\overline{x}, \tilde{t}) \rightsquigarrow \bot,$	if $s_1 \neq \overline{x}$ and $\overline{x} \in vars(s_1)$.
	$f(t_1, \tilde{t}) \overset{?}{=}_\emptyset f(s_1, \tilde{s}) \rightsquigarrow \bot,$	if $t_1 \overset{?}{=}_\emptyset s_1 \rightsquigarrow \bot$.
Eliminate:	$f(t_1, \tilde{t}) \overset{?}{=}_\emptyset f(s_1, \tilde{s}) \rightsquigarrow \langle\langle g(\tilde{t}\sigma) \overset{?}{=}_\emptyset g(\tilde{s}\sigma),\ \sigma \rangle\rangle,$	if $t_1 \overset{?}{=}_\emptyset s_1 \rightsquigarrow \langle\langle \top,\ \sigma \rangle\rangle$.
	$f(\overline{x}, \tilde{t}) \overset{?}{=}_\emptyset f(\overline{x}, \tilde{s}) \rightsquigarrow \langle\langle g(\tilde{t}) \overset{?}{=}_\emptyset g(\tilde{s}),\ \varepsilon \rangle\rangle.$	
	$f(\overline{x}, \tilde{t}) \overset{?}{=}_\emptyset f(s_1, \tilde{s}) \rightsquigarrow$	if $s_1 \notin \mathcal{SV}$ and $\overline{x} \notin vars(s_1)$,
	$\langle\langle g(\tilde{t}\sigma_1) \overset{?}{=}_\emptyset g(\tilde{s}\sigma_1),\ \sigma_1 \rangle,$	where $\sigma_1 = \{\overline{x} \leftarrow s_1\}$,
	$\langle g(\overline{x}, \tilde{t}\sigma_2) \overset{?}{=}_\emptyset g(\tilde{s}\sigma_2),\ \sigma_2 \rangle\rangle,$	$\sigma_2 = \{\overline{x} \leftarrow s_1, \overline{x}\}$.
	$f(s_1, \tilde{s}) \overset{?}{=}_\emptyset f(\overline{x}, \tilde{t}) \rightsquigarrow$	if $s_1 \notin \mathcal{SV}$ and $\overline{x} \notin vars(s_1)$,
	$\langle\langle g(\tilde{s}\sigma_1) \overset{?}{=}_\emptyset g(\tilde{t}\sigma_1),\ \sigma_1 \rangle,$	where $\sigma_1 = \{\overline{x} \leftarrow s_1\}$,
	$\langle g(\tilde{s}\sigma_2) \overset{?}{=}_\emptyset g(\overline{x}, \tilde{t}\sigma_2),\ \sigma_2 \rangle\rangle,$	$\sigma_2 = \{\overline{x} \leftarrow s_1, \overline{x}\}$.
	$f(\overline{x}, \tilde{t}) \overset{?}{=}_\emptyset f(\overline{y}, \tilde{s}) \rightsquigarrow$	where
	$\langle\langle g(\tilde{t}\sigma_1) \overset{?}{=}_\emptyset g(\tilde{s}\sigma_1),\ \sigma_1 \rangle,$	$\sigma_1 = \{\overline{x} \leftarrow \overline{y}\}$,
	$\langle g(\overline{x}, \tilde{t}\sigma_2) \overset{?}{=}_\emptyset g(\tilde{s}\sigma_2),\ \sigma_2 \rangle,$	$\sigma_2 = \{\overline{x} \leftarrow \overline{y}, \overline{x}\}$,
	$\langle g(\tilde{t}\sigma_3) \overset{?}{=}_\emptyset g(\overline{y}, \tilde{s}\sigma_3),\ \sigma_3 \rangle\ \rangle,$	$\sigma_3 = \{\overline{y} \leftarrow \overline{x}, \overline{y}\}$.
Split:	$f(t_1, \tilde{t}) \overset{?}{=}_\emptyset f(s_1, \tilde{s}) \rightsquigarrow$	if $t_1, s_1 \notin \mathcal{IV} \cup \mathcal{SV}$ and
	$\langle\langle f(r_1, \tilde{t}\sigma_1) \overset{?}{=}_\emptyset f(q_1, \tilde{s}\sigma_1),\ \sigma_1 \rangle, \ldots,$	$t_1 \overset{?}{=}_\emptyset s_1 \rightsquigarrow \langle\langle r_1 \overset{?}{=}_\emptyset q_1,\ \sigma_1 \rangle, \ldots,$
	$\langle f(r_k, \tilde{t}\sigma_k) \overset{?}{=}_\emptyset f(q_k, \tilde{s}\sigma_k),\ \sigma_k \rangle\rangle$	$\langle r_k \overset{?}{=}_\emptyset q_k,\ \sigma_k \rangle\rangle$.

Fig. 2. Transformation rules. $\tilde{t}$ and $\tilde{s}$ are possibly empty sequences of terms. $f, f_1, f_2 \in \mathcal{FFIX} \cup \mathcal{FFLEX}$. $g \in \mathcal{FFLEX}$ is a new symbol, if in the same rule $f \in \mathcal{FFIX}$. Otherwise $g = f$.

with the new label $\bot$. If UP is unifiable, we apply projection or transformation on UP and get $\langle\langle SUC_1, \sigma_1\rangle, \dots, \langle SUC_n, \sigma_n\rangle\rangle$. Then the node UP has n children, labeled respectively with $SUC_1, \dots, SUC_n$ and the edge to the SUC_i node is labeled with σ_i $(1 \le i \le n)$. The set $\{\sigma_1, \dots, \sigma_n\}$ is denoted by $SUB(UP)$.

We design the general unification procedure as a breadth first (level by level) tree generation process. Let $GUP_\emptyset$ be a unification problem. We label the root of the tree with $GUP_\emptyset$ (zero level). First level nodes (the children of the root) of the tree are obtained from the original problem by projection. Starting from the second level, we apply only a transformation step to a unification problem of each node, thus getting new successor nodes. The branch which ends with a node labeled by $\top$ is called a successful branch. The branch which ends with a node labeled by $\bot$ is a failed branch. For each node in the tree, we compose substitutions (top-down) displayed on the edges of the branch which leads to this node and attach the obtained substitution to the node together with the unification problem the node was labeled with. The empty substitution is attached to the root. For a node N, the substitution attached to N in such a way is called the associated substitution of N. Let $\Sigma(GUP_\emptyset)$ be the set of all substitutions associated with the $\top$ nodes. We call the tree a unification tree for $GUP_\emptyset$ and denote it $UT(GUP_\emptyset)$.

Example 4. Figure 3 shows development of successful branches of the unification tree for $GUP_\emptyset = f(x, b, \overline{y}, f(\overline{x})) \overset{?}{=}_\emptyset f(a, \overline{x}, f(b, \overline{y}))$. $\Sigma(GUP_\emptyset) = \{\{x \leftarrow a, \overline{x} \leftarrow b, \overline{x}, \overline{y} \leftarrow \overline{x}\}, \{x \leftarrow a, \overline{x} \leftarrow b, \overline{y} \leftarrow\}\}$.

$$\begin{tikzcd}
 & f(x,b,\overline{y},f(\overline{x})) \overset{?}{=} f(a,\overline{x},f(b,\overline{y})) \arrow[dl, "\varepsilon"] \arrow[dr, "\{\overline{y} \leftarrow\}"] & \\
f(x,b,\overline{y},f(\overline{x})) \overset{?}{=} f(a,\overline{x},f(b,\overline{y})) \arrow[d, "\{x \leftarrow a\}"] & & f(x,b,f(\overline{x})) \overset{?}{=} f(a,\overline{x},f(b)) \arrow[d, "\{x \leftarrow a\}"] \\
f(b,\overline{y},f(\overline{x})) \overset{?}{=} f(\overline{x},f(b,\overline{y})) \arrow[d, "\{\overline{x} \leftarrow b, \overline{x}\}"] & & f(b,f(\overline{x})) \overset{?}{=} f(\overline{x},f(b)) \arrow[d, "\{\overline{x} \leftarrow b\}"] \\
f(\overline{y},f(b,\overline{x})) \overset{?}{=} f(\overline{x},f(b,\overline{y})) \arrow[d, "\{\overline{y} \leftarrow \overline{x}\}"] & & f(f(b)) \overset{?}{=} f(f(b)) \arrow[d, "\varepsilon"] \\
f(f(b,\overline{x})) \overset{?}{=} f(f(\overline{b,\overline{x}})) \arrow[d, "\varepsilon"] & & SUCCESS \\
SUCCESS & &
\end{tikzcd}$$

Fig. 3. *Successful branches of* $UT(f(x, b, \overline{y}, f(\overline{x})) \overset{?}{=}_\emptyset f(a, \overline{x}, f(b, \overline{y})))$.

A stronger notion than minimality - disjointness - is used to prove Theorem 3 below. Formally, disjointness is defined as follows:

Definition 9. *A set of substitutions Σ is called disjoint modulo E with respect to a set of variables Var iff for all θ, $\sigma \in \Sigma$, if there exist substitutions λ_1, λ_2 such that*

- *for all sequence variables $\overline{x} \in Var$,*
  - $\overline{x} \leftarrow \notin \lambda_1$,
  - $\overline{x} \leftarrow \notin \lambda_2$,
  - *there exist terms $t_1, \ldots, t_n, s_1, \ldots, s_n$, $n \geq 0$ such that $\overline{x}\theta \circ \lambda_1 = t_1, \ldots, t_n$, $\overline{x}\sigma \circ \lambda_2 = s_1, \ldots, s_n$ and for all $1 \leq i \leq n$, either t_i and s_i are the same sequence variables or $t_i \doteq_E s_i$ and*
- *for all individual variables $x \in Var$,*
  - $x\theta \circ \lambda_1 \doteq_E x\sigma \circ \lambda_2$,

then $\theta = \sigma$.

The main result of this paper says that $\Sigma(GUP_\emptyset)$ is a minimal complete set of free unifiers of $GUP_\emptyset$:

Theorem 3. $\Sigma(GUP_\emptyset) = MCU_\emptyset(GUP_\emptyset)$.

Proof. Here we briefly sketch the idea. Details can be found in [17].

Completeness follows from the fact that for every unifier ϕ of $GUP_\emptyset$ there exists a branch β in $UT(GUP_\emptyset)$ such that for every substitution θ associated with a unification problem in β we have $\theta \ll_{\emptyset}^{vars(GUP_\emptyset)} \phi$.

Minimality is implied by disjointness of $\Sigma(GUP_\emptyset)$, which itself follows from the facts that for each non-terminal node UP in $UT(GUP_\emptyset)$ the set $SUB(UP)$ is a disjoint set of substitutions and every projecting or transforming substitution preserves disjointness. □

It is clear that the unification procedure terminates if $GUP_\emptyset$ contains no sequence variables (in this case the problem can be considered as a Robinson unification). Another terminating case is when one of the terms to be unified is ground. It yields to the following result:

Theorem 4. *Matching in a theory with individual and sequence variables, free constants, free fixed and flexible arity function symbols is finitary.*

We can add a cycle-checking method to the procedure: stop with failure if a unification problem attached to a node of unification tree coincides with a unification problem in the same branch of the tree. Then the following theorem holds:

Theorem 5. *The unification procedure with cycle-checking for $GUP_\emptyset$ terminates if no individual and sequence variables occur more than twice in $GUP_\emptyset$.*

If $GUP_\emptyset = t_1 \overset{?}{=} t_2$ has the property that sequence variables occur only as arguments of t_1 or t_2, then we can weaken the condition of the previous theorem:

Theorem 6. *The unification procedure with cycle-checking for $GUP_\emptyset = t_1 \overset{?}{=} t_2$, where sequence variables occur only as arguments of t_1 or t_2, terminates if no sequence variable occurs more than twice in $GUP_\emptyset$.*

The following termination condition does not require cycle-checking and does not depend on the number of occurrences of sequence variables. Instead, it requires for a unification problem of the form $f(\overline{x}) \overset{?}{=}_\emptyset f(t_1, \ldots, t_n)$, $n > 1$, to check whether $\overline{x}$ occurs in $f(t_1, \ldots, t_n)$. We call it *the sequence variable occurrence checking.* We can tailor this checking into the unification tree generation process as follows: if in the tree a successor of the unification problem of the form $f(\overline{x}) \overset{?}{=}_\emptyset f(t_1, \ldots, t_n)$, $n > 1$, has to be generated, perform the sequence variable occurrence checking. If $\overline{x}$ occurs in $f(t_1, \ldots, t_n)$, label the node with $\perp$, otherwise proceed in the usual way (projection or transformation).

Theorem 7. *If $GUP_\emptyset$ is a unification problem such that all sequence variables occurring in $GUP_\emptyset$ are only the last arguments of the term they occur, then the unification procedure with the sequence variable occurrence checking terminates.*

The fact that in most of the applications sequence variables occur precisely only at the last position in terms, underlines the importance of Theorem 7. The theorem provides an efficient method to terminate unification procedure in many practical applications.

5 Extension with Pattern-Terms

In this section we give a brief informal overview of an extension of the theory with patterns. Detailed exposition can be found in [17]. Pattern is an extended construct of the form $h_{m,k}(t_1, \ldots, t_n)$, where h is a fixed or flexible arity function symbol, m and k are linear polynomials with integer coefficients with the special types of variables - called index variables, which are disjoint from sequence and individual variables. Instances of patterns are: $h_{1,vn+3}(\overline{x}, y)$, $f_{vm,vk}(a)$, etc., where vn, vm and vk are index variables. The intuition behind patterns is that they abbreviate term sequences of unknown length: $h_{vm,vk}(t)$ abbreviates $h_{vm}(t), \ldots, h_{vk}(t)$. Patterns can occur as arguments in terms with flexible arity heads only. Such terms are called pattern-terms (P-terms). A minimal and complete unification procedure with patterns, individual and sequence variables, free constants, fixed and flexible arity function symbols is described in [17]. The procedure enumerates substitution/constraint pairs which constitute the minimal complete set of solutions of the problem.

Example 5. Let $\Gamma = f(\overline{x}, \overline{y}) \overset{?}{=}_\emptyset f(h_{vm,vk}(z))$. Then the unification procedure returns the set of substitution/constraint pairs

$$\begin{aligned}
\{\ &\{\{\overline{x} \leftarrow\ , \overline{y} \leftarrow h_{vm,vk}(z)\}, 1 \leq vm \wedge vm \leq vk\},\\
&\{\{\overline{x} \leftarrow h_{vm,vk}(z), \overline{y} \leftarrow\ \}, 1 \leq vm \wedge vm \leq vk\},\\
&\{\{\overline{x} \leftarrow h_{vm,vn}(z), \overline{y} \leftarrow h_{vn+1,vk}(z)\}, 1 \leq vm \wedge vm \leq vn \wedge vn+1 \leq k\}\ \},
\end{aligned}$$

with the property that each integer solution of a constraint, applied to the corresponding substitution, generates an element of the minimal complete set of solutions. For instance, the solution $vm = 1$, $vn = 3$, $vk = 4$ of the constraint $1 \leq vm \wedge vm \leq vn \wedge vn + 1 \leq vk$ applied on the substitution $\{\overline{x} \leftarrow h_{vm,vn}(z), \overline{y} \leftarrow h_{vn+1,vk}(z)\}$ gives a substitution $\{\overline{x} \leftarrow h_{1,3}(z), \overline{y} \leftarrow h_{4,4}(z), vm \leftarrow 1, vn \leftarrow 3, vk \leftarrow 4\}$ which belongs to the minimal complete set of solutions of Γ. In the expanded form the substitution looks like $\{\overline{x} \leftarrow h_1(z), h_2(z), h_3(z), \overline{y} \leftarrow h_4(z), vm \leftarrow 1, vn \leftarrow 3, vk \leftarrow 4\}$.

As we have already mentioned in the Introduction, patterns naturally appear in the proving context, when one wants to Skolemize, for instance, the expression $\forall x \exists \overline{y}\, (g(x) \doteq g(\overline{y}))$. Here $\overline{y}$ should be replaced with a sequence of terms $f_1(x), \ldots, f_{n(x)}(x)$, where $f_1, \ldots, f_{n(x)}$ are Skolem functions. The problem is that we can not know in advance the length of such a sequence. Note that in the unification we use an index variable vn instead of $n(x)$. This is because, given a unification problem UP in which $n(x)$ occurs, we can do a variable abstraction on $n(x)$ with a fresh index variable vn and instead of UP consider UP' together with the constraint $vn = n(x)$, where UP' is obtained from UP by replacing each occurrence of $n(x)$ with vn. One of the tasks for unification with patterns is to find a proper value for vn, if possible.

6 Applications

We have implemented the unification procedure (without the decision algorithm) as a Mathematica package and incorporated it into the Theorema system, where it is used by the equational prover. Besides using it in the proving context, the package can be used to enhance Mathematica solving capabilities, in particular, the Solve function of Mathematica. Solve has a rich arsenal of methods to solve polynomial and radical equations, equations involving trigonometric or hyperbolic functions, exponentials and logarithms. The following example shows, for instance, how a radical equation is solved:

In[1]:= $\mathbf{Solve}[\mathbf{x}^{1/3} + \sqrt{\mathbf{x}} == \mathbf{1}, \mathbf{x}]$

Out[1]= $\{\{\mathtt{x} \to -\frac{2}{3} - \frac{11}{3}\left(\frac{2}{101 + 15\sqrt{69}}\right)^{1/3} + \frac{1}{3}\left(\frac{1}{2}\left(101 + 15\sqrt{69}\right)\right)^{1/3}\}\}$

However, it is unable to solve a symbolic equation like $f(x, y) = f(a, b)$:

In[2]:= $\mathbf{Solve}[\mathbf{f}[\mathbf{x}, \mathbf{y}] == \mathbf{f}[\mathbf{a}, \mathbf{b}], \{\mathbf{x}, \mathbf{y}\}]$
Solve::"dinv": The expression `f[x, y]` involves unknowns in more than one argument, so inverse functions cannot be used
Out[2]= `Solve[f[x, y] == f[a, b], {x, y}]`

Also, Solve can not deal with equations involving sequence variables.

On the basis of the unification package, we implemented a function called SolveExtended, which has all the power of Solve and, in addition, deals with symbolic equations which can be solved using unification methods. An equation like $f(x, y) = f(a, b)$ becomes a trivial problem for SolveExtended:

```
In[3]:= SolveExtended[f[x, y] == f[a, b], {{x, y}, {}}]
       Answer 1
        {x → a, y → b}
        The procedure terminated
Out[3]= {{x → a, y → b}}
```

The function is able to solve (a system of) equations involving sequence variables as well. In the following example x is an individual variable and X and Y are sequence variables:

```
In[4]:= SolveExtended[f[x, b, Y, f[X]] == f[a, X, f[b, Y]], {{x}, {X, Y}}]
       Answer 1
        {x → a, X → b, Y → Sequence[]}
        Answer 2
        {x → a, X → Sequence[b, X], Y → X}
        The procedure terminated

Out[4]= {{x → a, X → b, Y → Sequence[]}, {x → a, X → Sequence[b, X], Y → X}}
```

All the problems Solve deals with can also be solved by SolveExtended, e.g.:

In[5]:= **SolveExtended**$\left[\mathbf{x^{1/3} + \sqrt{x} == 1, x}\right]$

Out[5]= $\left\{\left\{x \to -\frac{2}{3} - \frac{11}{3}\left(\frac{2}{101 + 15\sqrt{69}}\right)^{1/3} + \frac{1}{3}\left(\frac{1}{2}\left(101 + 15\sqrt{69}\right)\right)^{1/3}\right\}\right\}$

Options allow SolveExtended to switch on/off cycle-detecting and last sequence variable checking modes. Printing answers as they are generated is useful when the problem has infinitely many solutions and the procedure does not terminate, but this facility can also be switched off, if the user wishes so. It is possible to terminate execution after generating a certain number of solutions.

We found another application of the unification package in computing minimal matches for non-commutative Gröbner basis procedure. We show it on the following example: let p and q be non-commutative polynomials $a*a*b*a+a*b$ and $a*b*a*a+b*a$. In order to find S-polynomials of p and q one needs to compute all minimal matches between the leading monomials $a*a*b*a$ and $a*b*a*a$, which, in fact, is a problem of solving equations (for instance, of the form $a*a*b*a*R1 = L2*a*b*a*a$ for the variables $R1$ and $L2$) in a free semigroup. Solving equations in a free semigroup is a particular case of unification with sequence variables and flexible arity symbols. Even more, the equations we need to solve to find the matches, belong to a case when the unification procedure terminates. The example below demonstrates how the function MinimalMatches, based on the unification package, computes matches for $a*a*b*a$ and $a*b*a*a$:

```
In[6]:= MinimalMatches[a * a * b * a, a * b * a * a]
Out[6]={{a * a * b * a * R1, L2 * a * b * a * a, {L2 → a, R1 → a}},
        {a * a * b * a * R1, L2 * a * b * a * a, {L2 → a * a * b, R1 → b * a * a}},
        {L1 * a * a * b * a, a * b * a * a * R2, {L1 → a * b, R2 → b * a}},
        {L1 * a * a * b * a, a * b * a * a * R2, {L1 → a * b * a, R2 → a * b * a}}}
```

7 Conclusion

We considered a unification problem for the equational theory with sequence and individual variables, free fixed and free flexible arity function symbols and gave a unification procedure which enumerates the minimal complete set of unifiers. Several sufficient termination conditions have been established. We gave a brief overview of a theory extended with constructs called patterns, which are used to abbreviate sequences of unknown lengths of terms matching a certain "pattern". Unification procedure for the extended theory enumerates substitution/constraint pairs which constitute the minimal complete set of solutions of the problem. Computer algebra related applications have been discussed.

8 Acknowledgments

I wish to thank Prof. Bruno Buchberger for supervising the work and for many helpful discussions.

References

1. H. Abdulrab and J.-P. Pécuchet. Solving word equations. *J. Symbolic Computation*, 8(5):499–522, 1990.
2. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 445–532. Elsevier Science, 2001.
3. M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. String operations in query languages. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001.
4. B. Buchberger. Mathematica as a rewrite language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming)*, pages 1–13, Shonan Village Center, Japan, 1–4 November 1996. World Scientific.
5. B. Buchberger. Personal communication, 2001.
6. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema project: A progress report. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000)*, pages 98–113, St.Andrews, 6–7 August 2000.
7. A. Colmerauer. An introduction to Prolog III. *CACM*, 33(7):69–91, 1990.
8. A. Farquhar, R. Fikes, and J. Rice. The Ontolingua Server: A tool for collaborative ontology construction. *Int. J. of Human-Computer Studies*, 46(6):707–727, 1997.

9. M. R. Genesereth. Epilog for Lisp 2.0 Manual. Epistemics Inc., Palo Alto, 1995.
10. M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, June 1992.
11. S. Ginsburg and X. S. Wang. Pattern matching by Rs-operations: Toward a unified approach to querying sequenced data. In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 293–300, San Diego, 2–4 June 1992.
12. G. Grahne, M. Nykänen, and E. Ukkonen. Reasoning about strings in databases. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 303–312, Minneapolis, 24–26 May 1994.
13. G. Grahne and E. Waller. How to make SQL stand for string query language. *Lecture Notes in Computer Science*, 1949:61–79, 2000.
14. M. Hamana. Term rewriting with sequences. In *Proceedings of the First International Theorema Workshop*, Hagenberg, Austria, 9–10 June 1997.
15. J. Jaffar. Minimal and complete word unification. *J. of ACM*, 37(1):47–85, 1990.
16. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–298, Oxford, 1967. Pergamon Press. Appeared 1970.
17. T. Kutsia. Unification in a free theory with sequence variables and flexible arity symbols and its extensions. Technical report, RISC-Linz, 2002. Available from ftp://ftp.risc.uni-linz.ac.at/pub/people/tkutsia/reports/unification.ps.gz.
18. A. Lentin. Equations in free monoids. In M. Nivat, editor, *Automata, Languages and Programming*, pages 67–85, Paris, 3–7 July 1972. North-Holland.
19. G. S. Makanin. The problem of solvability of equations on a free semigroup. *Math. USSR Sbornik*, 32(2), 1977.
20. G. Mecca and A. J. Bonner. Sequences, Datalog and transducers. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 23–35, San Jose, 22–25 May 1995.
21. F. Mora. Gröbner bases for non-commutative polynomial rings. In J. Calmet, editor, *Proceedings of the 3rd International Conference on Algebraic Algorithms and Error-Correcting Codes (AAECC-3)*, volume 229 of *LNCS*, pages 353–362, Grenoble, July 1985. Springer Verlag.
22. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19:321–351, 1995.
23. P.Hayes and C. Menzel. A semantics for the knowledge interchange format. Available from http://reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf, 2001.
24. G. Plotkin. Building in equational theories. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, 1972.
25. A. Rubio. Theorem proving modulo associativity. In *Proceedings of the Conference of European Association for Computer Science Logic*, LNCS, Paderborn, Germany, 1995. Springer Verlag.
26. K. U. Schulz. Word unification and transformation of generalized equations. *J. Automated Reasoning*, 11(2):149–184, 1993.
27. J. Siekmann. String unification. Research paper, Essex University, 1975.
28. J. Siekmann. Unification and matching problems. Memo, Essex University, 1978.
29. M. Widera and C. Beierle. A term rewriting scheme for function symbols with variable arity. TR 280, Prakt.Informatik VIII, FernUniversitaet Hagen, 2001.
30. S. Wolfram. *The Mathematica Book*. Cambridge University Press and Wolfram Research, Inc., fourth edition, 1999.