One Context Unification Problems Solvable in Polynomial Time

Adrià Gascón and Ashish Tiwari
SRI International, Menlo Park, USA
adriagascon@gmail.com, tiwari@csl.sri.com

Manfred Schmidt-Schauss
Goethe-Universität, Frankfurt, Germany
schauss@cs.uni-frankfurt.de

***Abstract*—One context unification extends first-order unification by introducing a single context variable, possibly with multiple occurrences. One context unification is known to be in NP, but it is not known to be solvable in polynomial time. In this paper, we present a polynomial time algorithm for certain interesting classes of the one context unification problem. Our algorithm is presented as an inference system that non-trivially extends the usual inference rules for first-order unification. The algorithm is of independent value as it can be used, with slight modifications, to solve other problems, such as the first-order unification problem that tolerates one clash.**

I. Introduction

The problem of checking satisfiability of a set of equations plays a central role in any mathematical science. From the perspective of computer science, a lot of effort is devoted to finding efficient decision procedures for different families of equations. The problem of *satisfiability of word equations*, also known as *word unification*, figures prominently as one of the most intriguing problems of that form. The first algorithm for that problem was given by Makanin [15], and the best known upper bound (PSPACE) is due to Plandowski [19]. On the other hand, the best known lower bound is NP, which is widely believed to be the true complexity of the problem. Several particular cases of that problem, such as the ones that result from fixing the number of variables in the equations to a constant, have also been studied. For instance, efficient algorithms for satisfiability of word equations with one [4], [17], [11] and two [3] variables have been discovered.

Another fundamental operation in symbolic computation systems is the well-known *first-order unification problem*. This problem consists of solving equations of the form $s \doteq t$, where s and t are terms with first-order variables. The goal is to find a mapping from variables to (first-order) terms that would make the terms s and t syntactically equal. This problem was first introduced as such in the work by J.A. Robinson, which established the foundations of automated theorem proving and logic programming. More concretely, Robinson presented in [20] a procedure to determine the validity of a first-order sentence that has term unification as its main ingredient. Later, term unification was also used by Knuth and Bendix as a key component of their critical pairs method to determine local confluence of term rewrite systems (see [1] for a general survey on unification theory). The syntactic unification and matching problems were deeply investigated in the last century. Among other results, linear time algorithms were discovered [16], [18]. Moreover, more expressive variants of term unification such as *unification modulo theories* have also drawn a lot of attention. In this notion of term unification, equality between terms is interpreted under equational theories such as associativity, commutativity, and distributivity, among others [1].

An interesting connection between word and term unification is the *context unification problem*. In context unification, the terms s, t in the equation $s \doteq t$ may contain context variables. For example, consider the equation $F(f(x, b)) \doteq f(a, F(y))$. where x, y are first-order variables ranging over terms and F is a context variable that can be replaced by any context. One of the possible solutions of this instance is the substitution $\{F \mapsto f(a, \bullet), x \mapsto a, y \mapsto b\}$. Note that when we instantiate F by $f(a, \bullet)$ in the equation, replacing the occurrence of $\bullet$ by the argument of F in each of its occurrences, we get $f(a, f(x, b)) \doteq f(a, f(a, y))$, and thus both sides of the equations become equal after applying $\{x \mapsto a, y \mapsto b\}$. Note that, simply using a unary signature, word unification reduces to context unification. On the other hand, context unification is a restricted form of second-order unification, which is undecidable [9]. The decidability of context unification remained open for a long time, until recently a PSPACE algorithm was presented by Jeż [12].

Several variants of context unification with decision procedures in NP, such as stratified context unification and well-nested context unification, have been considered. Such variants have applications in computational linguistics and unification up to several positions [13], [5], [14]. Furthermore the variant of context unification where one of the sides of the equation is ground, called *context matching*, has also been investigated [21], [6] and, although the problem is in general NP-hard, polynomial time algorithms are known for some subclasses.

In this paper we revisit the particular case of context unification where only one context variable, possibly with many occurrences, occurs in the input equations. This problem is known as *one context unification* (1-CU). In [7], a non-deterministic polynomial time algorithm for 1-CU was presented. That result has later been extended [2] also to the

This work was sponsored, in part, by National Science Foundation under grant CCF-1423296 and ONR under subaward 60106452-107484-C under prime grant N00014-12-1-0914. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the funding agencies.

case where the input terms are represented with Singleton Tree Grammars, a grammar-based compression mechanism more general than Directed Acyclic Graphs. On the other hand, 1-CU is not known to be NP-hard nor solvable in polynomial time. This gap motivates the work described in this paper. We also remark here that initial interest in 1-CU came from interprocedural program analysis [10], where context variables are used to represent (the yet unknown) summaries of procedures. In particular, 1-CU problems (over uninterpreted terms) arise when analyzing programs using an abstract domain consisting of (uninterpreted) terms.

A. Related Work

A non-deterministic polynomial time procedure for 1-CU was presented in [7]. There are instances where that algorithm provably takes exponential worst-case (deterministic) running time to decide unifiability. For example, let s be $f(x_0, x_0)$ and let t^n be the term recursively defined as $f(f(x_n, x_n), t^{n-1})$ for $n > 0$, and $t^0 = f(a, b)$. Consider the 1-CU instance $\{F(a) \doteq s, F(b) \doteq t^n\}$, where $x_1, \ldots, x_n$ are pairwise different first-order variables and F is a context variable. Although this instance has an exponential number of solutions, the algorithm from [7] requires (deterministic) exponential time to conclude unifiability, i.e. solve the decision version of the problem. However, our algorithm solves this class of problems in polynomial time.

This paper extends the results of [7] by providing polynomial time solutions to several interesting classes of the one context unification problem. Our approach consists on reducing a general 1-CU instance to solving polynomially many instances of a restricted form that we call *reduced instances*. This approach immediately yields a polynomial time procedure for classes of 1-CU whose reduced instances can be efficiently solved. We present some such classes in this paper, which include classes containing left- and right-ground instances, a class containing instances with disjoint first-order variables on the two sides of equations, and a certain class of instances with a constant number of equations. In fact, we do not have a class of examples for which an algorithm consisting on recursive applications of our procedure takes exponential time.

Our algorithm can actually be seen as following a "divide and conquer" paradigm and it relies on polynomial time algorithms for some base cases. One of the most important base cases are instances of the form $\{F(r_1) \doteq s, F(r_2) \doteq t\}$, where the context variable F does not occur in r_1 and r_2. We call this the 2-restricted 1-CU problem. The algorithm for this case is presented in a companion paper [8].

II. Preliminaries

We assume a fixed ranked alphabet $\mathcal{F}$ and a set of variables $\mathcal{X}$ containing first-order variables and exactly one context variable. By $\texttt{maxarity}$ we denote the maximum arity of the symbols in $\mathcal{F}$. We denote the context variable by F and first-order variables by x with possible subindices. Our algorithm introduces fresh first-order variables from a set $\mathcal{Y}$, which we denote by y with possible subindices. We denote $\mathcal{X} \cup \mathcal{Y}$ as $\mathcal{V}$. We will argue about terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \mathcal{Y}$. A term containing exactly one occurrence of a special symbol, $\bullet$, is called a context. The unique position of the hole $\bullet$ in a context $C[\bullet]$ is called the *hole position* of C and denoted by $\texttt{hp}(C)$. The set of contexts constructed using symbols in $\mathcal{F}$ and variables in $\mathcal{X}$ is denoted by $\mathcal{C}(\mathcal{F}, \mathcal{X})$.

With $<$ we denote the prefix relation among positions and with $\prec$ the subterm relation among terms. We also define, for a term $t = \alpha(t_1, \ldots, t_n)$, $\texttt{topsymbol}(t) = \alpha$. The *exponentiation* of a position p to a natural number n, denoted p^n, is the position recursively defined as $p^n = p.p^{n-|p|}$ if $n > |p| > 0$, as $p^n = p_1$ if $n \leq |p|$, $p = p_1.p_2$, and $|p_1| = n$, and $p^n = \lambda$, otherwise.

In this work, we deal with *multiequations on terms*, denoted by m with possible subindices. Given a multiequation $m = (t_1 \doteq \ldots \doteq t_n)$, we call the set $\bigcup_i^n \{t_i\}$ the *top terms of* m, denoted $\texttt{topterms}(m)$. Similarly, for a multiset Δ of multiequations, $\texttt{topterms}(\Delta)$ denotes $\bigcup_{m \in \Delta} \texttt{topterms}(m)$. Similarly, $\texttt{topvars}(m) = \texttt{topterms}(m) \cap \mathcal{V}$. We also extend $\texttt{topsymbols}$ from terms to multiequations as $\texttt{topsymbols}(m) = \bigcup_{t \in \texttt{topterms}(m)} \{\texttt{topsymbol}(t)\}$. By $|\Delta|$ we denote the number of multiequations in Δ. Moreover, our multiequations are ordered and we use $m[i]$ to refer to t_i, for every $i \in \{1, \ldots, n\}$.

A *substitution*, denoted by σ, θ, η, is a total function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{C}(\mathcal{F}, \mathcal{V})$ such that $\alpha\sigma \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ if α is a first-order variable and $\alpha\sigma \in \mathcal{C}(\mathcal{F}, \mathcal{V})$ if α is a context variable.

The *domain of a substitution* σ, denoted $\texttt{dom}(\sigma)$, is defined as usual, i.e. $\texttt{dom}(\sigma) = \{z \in \mathcal{V} \mid z\sigma \neq z\}$. The *composition* of σ and θ, denoted $\theta \circ \sigma$, is defined as $\{\alpha \mapsto (\alpha\sigma)\theta \mid \alpha \in \texttt{dom}(\sigma) \cup \texttt{dom}(\theta)\}$.

For substitutions σ, θ, $\sigma = \theta$ holds if $\forall z \in \mathcal{V} : z\sigma = z\theta$. Moreover, σ is *more general* than θ, denoted $\sigma \leq \theta$, if there exists η such that $\theta = \sigma \circ \eta$.

Substitutions are extended to be mappings from terms to terms, i.e. $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, as $g(t_1, \ldots, t_n)\sigma = g(t_1\sigma, \ldots, t_n\sigma)$ and $F(t)\sigma = F\sigma(t\sigma)$. In addition, substitutions are also extended, in a similar way, to be mappings from contexts to contexts, i.e. $\sigma : \mathcal{C}(\mathcal{F}, \mathcal{V}) \to \mathcal{C}(\mathcal{F}, \mathcal{V})$. We also define $m\sigma = (t_1\sigma \doteq \ldots \doteq t_n\sigma)$, for a multiequation $m = (t_1 \doteq \ldots \doteq t_n)$, $\Delta\sigma = \biguplus_{m \in \Delta}(m\sigma)$, for a multiset of multiequations Δ, and $L\sigma = \langle t_1\sigma, \ldots, t_n\sigma \rangle$, for a list of terms $L = \langle t_1, \ldots, t_n \rangle$.

A *unifier* of two terms s, t is a substitution σ such that $s\sigma = t\sigma$. A unifier does not always exist. We capture that situation by simply saying that the unifier of s and t is $\bot$. We define the *most general unifier* of terms s and t, denoted $\texttt{mgu}(s = t)$, as *any* substitution σ such that, for every unifier θ of s and t, $\sigma \leq \theta$ holds. If such substitution does not exist we say that $\texttt{mgu}(s = t)$ is *not defined*, denoted $\texttt{mgu}(s = t) = \bot$. In an abuse of notation, we assume that $\alpha\sigma = \bot$ if $\sigma = \bot$, for every term or multiequation or multiset of multiequations or list of terms α. Moreover, $\texttt{mgu}$ is extended to multiequations in the natural way.

Although 1-CU is defined as a set of equations over terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we can always transform an 1-CU instance (by

several applications of the usual first-order unification rules) to the following more restricted definition without loss of generality.

Definition II.1 (1-CU)**.** *An instance $\mathcal{I}$ of the 1-CU problem is a set of equations $\{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\}$, where $\forall i \in \{1, \ldots, n\}$: $\texttt{topsymbol}(t_i) \neq F$. A solution of $\mathcal{I}$ is a substitution σ such that $F(s_i)\sigma = t_i\sigma$, for every $i \in \{1, \ldots, n\}$.*

Definition II.2 (size)**.** *Let $\mathcal{I}$ be a 1-CU instance. The size of $\mathcal{I}$, denoted $||\mathcal{I}||$, is defined as the sum of the sizes of the terms in the equations of $\mathcal{I}$.*

We assume the DAG representation for terms and hence the size $||\mathcal{I}||$ is just the number of nodes in the DAG representing all terms in $\mathcal{I}$. In the sequel, we assume this measure for 1-CU instances and hence we do not state it explicitly every time we refer to a *polynomial time* algorithm. Moreover, when we state that a unification problem *can be solved* in polynomial time, we refer to both its decisional version (deciding unifiability) and functional version (finding a unifier).

III. Overview of the Procedure

In this section, we present a very high-level overview of our procedure for solving the one context unification problem.

Our procedure is based on the observation that a solution σ of a 1-CU instance $\mathcal{I} = \{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\}$ is completely characterized by the position $p = \texttt{hp}(F\sigma)$. Hence, our procedure for 1-CU proceeds by searching for p. However, as illustrated by the example given in the introduction, a naive brute force enumeration of all possibilities for p may take exponential time. Instead, our algorithm carefully prunes the search space for p while preserving unifiability.

Our procedure is closely related to the standard syntactic (first-order) unification procedure. A syntactic unification procedure on $s \doteq t$ can be interpreted as searching for a "conflict position" p in s and t; that is, a position p such that if we unify $s|_q \doteq t|_q$ for all positions q parallel to p, say with a unifier σ, then $s\sigma|_p$ and $t\sigma|_p$ are definitely not unifiable – due to an obvious "clash" or "occur check violation". The search for such a p proceeds by starting with $p = \lambda$ and then considering larger positions p. When we try to unify $\mathcal{I} = \{F(u) \doteq s, F(v) \doteq t\}$, our procedure also tries to search for "conflict" positions p in s, t because these positions are likely candidates for $\texttt{hp}(F\sigma)$, where σ is a unifier for $\mathcal{I}$.

IV. Special Cases

Our procedure for 1-CU follows a "divide and conquer" paradigm, much like standard term unification procedures, and it relies on dedicated 1-CU procedures for certain base cases called elementary 1-CU instances. In this section we present these elementary 1-CU instances, and show that they can be solved in polynomial time.

The first elementary instance is the analog of the clash rule for first-order unification.

Claim IV.1 (Clash)**.** *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq f(s_1, \ldots, s_m), F(u_2) \doteq g(t_1, \ldots, t_{m'})\}$, with $f \neq g$. Then, $\mathcal{I}$ can be solved in polynomial time and every solution σ satisfies $F\sigma = [\bullet]$.*

Next, consider another special case where we have one equation that contains F on both sides. This special case was solved in a previous paper [7]. In the case of the equation $F(s) \doteq C[F(t)]$, one of the key observations is that the hole position of every context that is a solution for F has to be a prefix or exponentiation of $\texttt{hp}(C)$. In the problem considered in [7], the terms at the input were given explicitly. Since we assume the DAG representation for terms, the result from [7] needs to be extended for our setting. However, the proof ideas are the same.

Theorem IV.1 ([7])**.** *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(s) \doteq C[F(t)]\}$, where C is a non-empty context. Then, $\mathcal{I}$ can be solved in polynomial time.*

The second elementary instance is the analog of the "occur check" rule of term unification and is characterized by equations $F(u_1) \doteq s$ and $F(u_2) \doteq C[s]$, where C is nonempty.

Claim IV.2 (Cyclic)**.** *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq s, F(u_2) \doteq C[s]\}$, where C is a non-empty context. Then, $\mathcal{I}$ can be solved in polynomial time.*

Proof. Consider the instance $\mathcal{I}'' = \mathcal{I} \cup \{F(u_2) \doteq C[F(u_1)]\}$. The lemma follows from the fact that $\mathcal{I}$ and $\mathcal{I}''$ have the same set of solutions, $||\mathcal{I}''||$ is polynomial w.r.t. $||\mathcal{I}||$, and $\mathcal{I}''$ can be solved in polynomial time by Theorem IV.1. □

Our third elementary instance is called a reduced 1-CU instance.

Definition IV.2. *An 1-CU instance $\mathcal{I}$ is called* reduced *if it is of the form*

$$\{F(u_i) \doteq x_i \mid i = 1, 2, \ldots\} \cup \{F(v_j) \doteq s \mid j = 1, 2, \ldots\} \cup \{F(w_k) \doteq t \mid k = 1, 2, \ldots\} \quad (1)$$

where s, t do not contain F; that is, the right hand-side of the equations have at most two non-variable terms.

For now, we will assume that we can solve reduced 1-CU instances in polynomial time. In Section VI, we will return to reduced 1-CU instances. We will now present our 1-CU procedure, assuming we have access to a procedure `solve` that can solve any elementary 1-CU instance (in polynomial time).

V. Inference Rules for One Context Unification

We present inference rules for solving the one context unification problem in this section.

A. Defining the State

Our inference rules operate on states (configurations), which are pairs of the form

$$\langle \Delta, L \rangle$$

where Δ is a multiset of multiequations and L is a list of terms. Given a 1-CU instance $\{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\}$, the initial state of our algorithm is

$$\langle \{\!\{t_1 \doteq \ldots \doteq t_n\}\!\}, \langle s_1, \ldots, s_n \rangle \rangle$$

Intuitively, our algorithm tries to first-order unify $t_1, \ldots, t_n$ and find a "conflict position", which will be the hole position of F in some solution σ, i.e. $\mathtt{hp}(F\sigma)$.

Our states satisfy the invariant that each multiequation in Δ has $|L|$ top terms and the value $|L|$ remains unchanged. Hence, if we start with the above initial state, then for every state $\langle \Delta, L \rangle$ generated by our inference rules, we will have that

(a) $|L| = n$ and
(b) every multiequation in Δ will have n top terms.

In the sequel, especially in the section proving correctness of our procedure, whenever we say *state*, we implicitly assume that it satisfies (a) and (b).

Since Δ is a multiset, we fix the following convention: whenever we refer to a multiequation $m \in \Delta$, we will mean *one specific* element of Δ and not all the multiple occurrences of the element m in Δ. We will later see that our inference rules will guarantee that Δ always turns into a set, and it is a multiset only in "transient" states.

B. Mapping State to 1-CU Instances

We will next formally define a mapping from states to 1-CU instances. This mapping will help in stating the soundness and completeness of the inference rules.

Definition V.1. *Let $S = \langle \Delta, \langle u_1, \ldots, u_n \rangle \rangle$ be a state, let $m \in \Delta$ be a multiequation, and let $\theta = \mathtt{mgu}(\Delta \setminus \{\!\{m\}\!\})$. We define the 1-CU instance spanned by m in S, denoted $P(m, S)$ (or simply $P(m)$ if S is clear from the context), as*

$$P(m) = \begin{cases} \bigcup_{i \in \{1,\ldots,n\}} \{F(u_i\theta) \doteq (m[i])\theta\} & \text{if } \theta \neq \bot \\ \bot & \text{otherwise.} \end{cases}$$

where $m[i]$ denotes the i-th top term in the multiequation m.

Note that several copies of the same multiequation in Δ span the same problem.

We now extend our definition of a solution of an 1-CU instance to define solution of a state.

Definition V.2. *Let $\langle \Delta, L \rangle$ be a state of our procedure. A solution for $\langle \Delta, L \rangle$ is a pair $\langle m, \theta \rangle$, where $m \in \Delta$ is a multiequation such that $P(m) \neq \bot$, and θ is a solution for $P(m)$.*

The inference rules for 1-CU are presented in Figure 1 and Figure 2. We now explain these rules.

C. The `ForcedDecompose` Rule

Recall the decomposition rule for first-order unification which replaces the equation $f(s,t) \doteq f(u,v)$ by two equations $s \doteq u$ and $t \doteq v$. Our algorithm uses a variant of this common decomposition rule. Our rule is applied on the multiequations in Δ and produces a multiset of multiequations. For example, consider the following 1-CU instance:

$$\{F(s_0) \doteq f(x,x),\ F(s_1) \doteq f(u_1,v_1),\ \ldots,\ F(s_n) \doteq f(u_n,v_n)\}$$

As mentioned above, this 1-CU instance corresponds to the following initial state of our algorithm:

$$S_0 = \langle \{\!\{f(x,x) \doteq f(u_1,v_1) \doteq \ldots \doteq f(u_n,v_n)\}\!\}, \langle s_0, \ldots, s_n \rangle \rangle$$

Checking whether $\mathtt{hp}(F\sigma) = \lambda$ (or, equivalently $F\sigma = [\bullet]$) is a solution reduces to first-order unification. If we can find a solution σ where $F\sigma = [\bullet]$, we are done. If not, then we should consider the cases where $|\mathtt{hp}(F\sigma)| > 0$. After decomposition, we obtain the following state:

$$S_1 = \langle \{\!\{x \doteq u_1 \doteq \ldots \doteq u_n,\ x \doteq v_1 \doteq \ldots \doteq v_n, \}\!\}, \langle s_0, \ldots, s_n \rangle \rangle$$

This state, according to Definition V.1, spans the following two 1-CU instances:

1) $\{F(s_0\sigma_1) \doteq x\sigma_1, F(s_1\sigma_1) \doteq u_1\sigma_1, \ldots, F(s_n\sigma_1) \doteq u_n\sigma_1, \}$, where $\sigma_1 = \mathtt{mgu}(x \doteq v_1 \doteq \ldots \doteq v_n)$, and
2) $\{F(s_0\sigma_2) \doteq x\sigma_2, F(s_1\sigma_2) \doteq v_1\sigma_2, \ldots, F(s_n\sigma_2) \doteq v_n\sigma_2, \}$, where $\sigma_2 = \mathtt{mgu}(x \doteq u_1 \doteq \ldots \doteq u_n)$.

If σ is a solution for the original problem and $\mathtt{hp}(F\sigma) = i.p$, where $i \in \{1,2\}$, then the instance (1) has a solution σ' s.t. $\mathtt{hp}(F\sigma') = p$ if $i = 1$, and the instance (2) has a solution σ' s.t. $\mathtt{hp}(F\sigma') = p$ if $i = 2$. Hence our new state S_1 does not miss any solution of the original problem where $|\mathtt{hp}(F\sigma)| > 0$. However, considering both cases above and solving the corresponding subproblem *separately* is not a good idea, since the algorithm may end up exploring too many equivalent possibilities. In fact, this is one reason for the exponential running time of the algorithm presented in [7], on the class of instances presented in the introduction.

A possible alternative is to "delay" the computation and application of the substitutions σ_1 and σ_2 as much as possible. So, we keep state S_1 as such and try to proceed. But, we can not apply the usual *decomposition* rule on state S_1 since both multiequations in it have a variable x. So, how do we make progress? Here we use our *first key idea*: we extend the decomposition rule to allow minimal instantiation of x that will allow us to progress (with decomposition steps).

For simplicity, let us assume that the u_i's and v_i's of our example are of the forms $u_i = f(u_i^1, u_i^2)$ and $v_i = f(v_i^1, v_i^2)$. In this case, note that σ_1 would have instantiated x by a term of the form $f(\_,\_)$ and σ_2 would also have instantiated x by a (possibly different) term of the *same form*. Hence, we can keep both options open for x in the next state by instantiating x by $f(y_1, y_2)$ where y_1, y_2 are *fresh* variables. This allows us to proceed with decompose and avoiding committing to any one branch. We call this inference rule the `ForcedDecompose` rule, and it is shown in Figure 2. Unfortunately, the `ForcedDecompose` rule adds new variables to our problem.

After applying the `ForcedDecompose` rule, we get the following state in our example:

$$\begin{aligned} S_2 = \langle \{\!\{ & y_1 \doteq u_1^1 \doteq \ldots \doteq u_n^1, \\ & y_2 \doteq u_1^2 \doteq \ldots \doteq u_n^2, \\ & y_1 \doteq v_1^1 \doteq \ldots \doteq v_n^1, \end{aligned}$$

$$y_2 \doteq v_1^2 \doteq \ldots \doteq v_n^2\}\!\},$$
$$\langle s_0\{x \to f(y_1, y_2)\}, \ldots, s_n\{x \to f(y_1, y_2)\}\rangle\rangle$$

Our approach consists of applying this lazy instantiation followed by term decomposition (the `ForcedDecompose` rule), but ensuring that, every time we apply this rule, we decrease the measure $|\texttt{subterms}(\texttt{topterms}(\Delta)) \setminus \mathcal{Y}|$, i.e. the total number of terms occurring in the multiequations of Δ that are not fresh variables. If we apply the `ForcedDecompose` rule arbitrarily, then the above measure may not decrease. Even in regular first-order unification, a decomposition step is not guaranteed to remove some subterm from Δ.

Here we use our *second key idea*: instead of decomposing arbitrarily any/all multiequations in Δ, we ensure that the above measure decreases by decomposing only a submultiset $\Gamma \subseteq \Delta$ of multiequations at a time. The selected submultiset Γ satisfies that every term in $\texttt{topterms}(\Gamma)$ is maximal with respect to the subterm relation in Δ. More concretely, a subset $\Gamma \subseteq \Delta$ is called a *root class* if for every top term t in Γ, (a) t is not a proper subterm of any term in Δ, and (b) t is not a top term in $\Delta \setminus \Gamma$. A consequence of this side condition in our `ForcedDecompose` rule is that the variables in $\mathcal{Y}$ will never occur in the multiequations in Δ as subterms of other terms, i.e., we will maintain the invariant that $\texttt{topterms}(\Delta) \subset (\mathcal{T}(\Sigma, \mathcal{X}) \cup \mathcal{Y})$. The idea of decomposing on maximal equations w.r.t. to the subterm relation is already used in the Paterson-Wegman linear unification algorithm [18], where equations in a *root class* are decomposed first.

We formally define the `ForcedDecompose` rule next. First, let us recall a variant of the traditional term decomposition rule for multiequations.

Definition V.3. *Let Δ be a set of multiequations and let $m = f(s_1^1, \ldots, s_l^1) \doteq \ldots \doteq f(s_1^k, \ldots, s_l^k)$ be a multiequation in Δ such that $\texttt{topvars}(m) = \emptyset$ and $\texttt{topsymbols}(m) = \{f\}$, with $l = \texttt{ar}(f)$.*

- *By $\texttt{Decompose}(m)$ we denote the multiset of multiequations $\bigcup_{i=1}^{l}(\{s_i^1 \doteq \ldots \doteq s_i^k\})$,*
- *By $\texttt{Decompose}(\Delta, m)$ we denote $\Delta \setminus \{m\} \cup \texttt{Decompose}(m)$, and*
- *`Decompose` is also extended to a submultiset of multiequations $\Gamma \subseteq \Delta$ as $\texttt{Decompose}(\Delta, \Gamma) = (\Delta \setminus \Gamma) \cup \bigcup_{m \in \Gamma} \texttt{Decompose}(m)$.*

Definition V.4. *Let $S = \langle \Delta, L\rangle$ be a state of the algorithm, let $\Gamma \subseteq \Delta$ be a subset of multiequations such that $\texttt{topsymbols}(\Gamma) = \{f\}$, and let $\mathcal{Y}$ be a set of first-order variables disjoint with $\texttt{vars}(S)$. Let $\{x_1, \ldots, x_k\}$ be $\texttt{topvars}(\Gamma)$ and let σ be $\bigcup_{i=1}^{k}(\{x_i \to f(y_1^i, \ldots, y_{\texttt{ar}(f)}^i)\})$, where $y_j^i \in \mathcal{Y}$, for $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, \texttt{ar}(f)\}$.*

Then, $\texttt{ForcedDecompose}(S, \Gamma, \mathcal{Y})$ is defined as the state $\langle(\Delta \setminus \Gamma) \cup \texttt{Decompose}(\Delta\sigma, \Gamma\sigma), L\sigma\rangle$.

Note that the substitution σ is not applied to $\Delta \setminus \Gamma$ in the definition of $\texttt{ForcedDecompose}(S, \Gamma, \mathcal{Y})$: since Γ is a root class, variables in $\texttt{dom}(\sigma)$ do not occur in $\Delta \setminus \Gamma$.

D. The Shrinking Rules

If we only rely on decomposition, we will end up with exponentially many multiequations in Δ. To avoid this explosion, we exhaustively apply a sequence of *shrinking operations* to Δ before applying every decomposition step. Such shrinking rules are shown in Figure 1. The shrinking rules simplify the current state $\langle \Delta, L\rangle$ of the algorithm by either completely solving one of the problems spanned by $\langle \Delta, L\rangle$ in polynomial time (rules `CycleOrClash` and `TwoNonVar`), or applying substitutions that preserve all solutions (rules `InvEq`, `NoSol`).

A crucial property of our algorithm (captured in Lemma V.17) is that, if none of the shrinking rules can be applied, then $|\Delta|$ is (bounded by) a polynomial function of $|\texttt{subterms}(\texttt{topterms}(\Delta)) \setminus \mathcal{V}|$. This fact, together with the fact that every application of `ForcedDecompose` reduces $|\texttt{subterms}(\texttt{topterms}(\Delta)) \setminus \mathcal{V}|$ and the application of the other rules does not increase that value, forms the core of our termination argument.

We will now describe the shrinking inference rules. The first three rules each remove a multiequation m from the multiset Δ for which the instance $P(m)$ is an elementary instance. The last rule will simplify the problem by applying a substitution.

a) The `NoSol` inference rule: If $\Delta \setminus \{\!\{m\}\!\}$ is *not* unifiable, then it means that hole position of F can not lie at (or below) the position corresponding to m, and in fact note that $P(m) = \bot$ in this case. Hence, the multiequation m can be discarded. To discard m, we have to unify m, apply its unifier to all other multiequations in Δ and continue. This is captured in the `NoSol` rule in Figure 1, which states that if Δ is of the form $\Delta' \uplus \{\!\{m\}\!\}$ (where $\uplus$ denotes disjoint multiset union) and $\texttt{mgu}(\Delta') = \bot$, then we can discard m by updating Δ to be $\Delta'\texttt{mgu}(m)$.

b) The `CycleOrClash` inference rule: If the 1-CU instance $P(m)$ has two equations of the form $F(u_1) = f(\ldots)$ and $F(u_2) = g(\ldots)$ for some $f \neq g$, then we can easily determine if $P(m)$ has a solution using Claim IV.1. Similarly, if the 1-CU instance $P(m)$ has two equations of the form $F(u_1) = s$ and $F(u_2) = C[s]$ for some nonempty context C, then we can easily determine if $P(m)$ has a solution using Claim IV.2. In both these cases, if $P(m)$ has a solution, we can terminate the search and report success. If $P(m)$ has no solution, then we can discard m as in the `NoSol` rule. This process is formalized in the `CycleOrClash` inference rule.

c) The `TwoNonVar` inference rule: Recall that we assumed access to a procedure `solve` that works on reduced 1-CU instances. The `TwoNonVar` rule works in the same way as the rule `CycleOrClash`, except that it is applied when $P(m)$ is a reduced 1-CU instance.

d) The `InvEq` inference rule: Rather than discarding multiequations m from Δ, our last shrinking rule `InvEq` removes variables from the state by applying substitutions that can be deduced to hold in every branch of the search tree. How to find such "globally" valid substitutions? We need a few definitions for this purpose.

Definition V.5. *Let Δ be a multiset of term multiequations. Let Δ^l be the set obtained by marking each occurrence of a multiequation m in Δ with a different mark*[1]*. The graph $G(\Delta)$ is defined as the undirected graph that has* $\texttt{topterms}(\Delta) \cup \Delta^l$ *as the nodes and the relation* $\{(s, m^l) \mid m^l \in \Delta^l, s \in \texttt{topterms}(m)\}$ *as the edges.*

Cycles in the graph $G(\Delta)$ are special: if terms s, t lie on a cycle, then every solution σ of every problem $P(m)$ should unify s and t.

Definition V.6. *Let Δ be a set of multiequations and let $s, t \in$* topterms *be distinct terms. We say that Δ* induces the equality *$s = t$, denoted $\Delta \models (s = t)$, if s and t lie on some cycle in $G(\Delta)$.*

It follows directly from the definition above that equations induced by Δ can be computed efficiently.

Lemma V.7. *Given a multiset Δ of multiequations, terms $s, t \in$* topterms *such that $\Delta \models (s = t)$, if they exist, can be found in polynomial time with respect to $|\Delta|$.*

If $s = t$ is an induced equality, then every solution σ of the 1-CU problem should make $s\sigma = t\sigma$, and hence, we can unify s and t and apply the unifier to our state without losing any solutions. This action is performed by the InvEq rule.

Note that all the shrinking rules rely on the application of a most general unifier. Such a unifier θ does not necessarily always exist and hence, by our convention mentioned in Section II, in that case $\theta = \bot$ and $\langle \Delta\theta, L\theta \rangle = \langle \bot, \bot \rangle$. This allows us to simplify the presentation by having a single failing rule Fail (Figure 2) while still making the failing cases explicit.

Remark V.8. *If Δ has two copies of the same multiequation m, say $s_1 \doteq \cdots \doteq s_n$, then we have $\Delta \models (s_1 = \cdots = s_n)$. The* InvEq *can be used to unify all the s_i's in the state, and as a result the problem $P(m)$ has all right-hand side terms equal. The inference rule* TwoNonVar *can remove such m. Hence, if the multiset Δ has multiple occurrences of m, it gets converted to a set.*

E. The algorithm

All our inference rules are presented in Figures 1 and 2. To obtain a polynomial time procedure, we will apply our inference rules according to a particular strategy. Specifically, our strategy gives priority to the shrinking rules over the ForcedDecompose rule and thus can be described as sequences of rule application of the form: $(\texttt{Shrink}^! \cdot \texttt{ForcedDecompose})^!$ where Shrink refers to (CycleOrClash | TwoNonVar | InvEq | NoSol). Here the notation R! refers to applying rule R repeatedly until it is not applicable any more.

Assuming that the procedure solve runs in polynomial time, to prove polynomial running time of our procedure, we have to argue that

(a) all intermediate subproblems have polynomial size,
(b) every application of ForcedDecompose yields polynomially many subproblems, and
(c) the derivations in our algorithm have polynomial length.

Note that, roughly speaking, to prove (a) and (b) it suffices to argue that every state $\langle \Delta, L \rangle$ considered by our algorithm can be represented in polynomial space. Regarding correctness, we must argue that our rules neither miss solutions (completeness) nor introduce new solutions (soundness).

F. Correctness

The following lemma formalizes the correctness of the process of eliminating a multiequation m from Δ. Its proof easily follows from Definitions V.1, and V.2 by induction on $|\Delta|$.

Lemma V.9. *Let $S = \langle \{m^\ell\} \uplus \Delta, L \rangle$ be a state of our algorithm. S has a solution if and only if either $P(m)$ has a solution or $\langle \Delta\sigma, L\sigma \rangle$ has a solution, where $\sigma = mgu(m)$.*

We denote an application of an inference rule using infix $\rightarrow$ notation, possibly labeled with the name of the inference rule. Correctness of rules CycleOrClash, TwoNonVar, and NoSol stated in the next two lemmas, is a direct consequence of the previous lemma.

Lemma V.10. *Let $\langle \Delta, L \rangle \rightarrow_r (\texttt{solve}(P(m)) \mid \langle \Delta', L' \rangle)$ be an inference step with $r \in \{\texttt{CycleOrClash}, \texttt{TwoNonVar}\}$. Then, $\langle \Delta, L \rangle$ has a solution if and only if either $P(m, \Delta)$ has a solution or $\langle \Delta', L' \rangle$ has a solution.*

Lemma V.11. *Let $\langle \Delta, L \rangle \rightarrow_{\texttt{NoSol}} \langle \Delta', L' \rangle$ be an inference step. Then, $\langle \Delta, L \rangle$ has a solution if and only if $\langle \Delta', L' \rangle$ has a solution.*

Proof. Note that $P(m) = \bot$, where $m \in \Delta$ is the multiequation that satisfies the condition for the application of NoSol. □

Lemma V.12. *Let $\langle \Delta, L \rangle \rightarrow_{\texttt{ForcedDecompose}} (\texttt{solve}(P(m_1)\sigma) \mid \ldots \mid \texttt{solve}(P(m_{|\Delta|})\sigma) \mid \langle \Delta', L' \rangle)$ be an inference step. Then, $\langle \Delta, L \rangle$ has a solution $\langle m, \theta \rangle$ if and only if either $F\theta = [\bullet]$ and the first-order unification problem $P(m)\sigma$ has a solution or $\langle \Delta', L' \rangle$ has a solution.*

Proof. Note that, if $\langle \Delta, L \rangle$ has a solution $\langle m, \theta \rangle$ then either θ maps F to $[\bullet]$ or it does not. The former case is checked by solving the first-order unification problem $P(m)\sigma = P(m)\{F \mapsto [\bullet]\}$, whereas the latter case implies that $\langle \Delta', L' \rangle$ has a solution, since the ForcedDecompose operation simply decomposes some multiequations without making any assumptions. For the other implication, note that the ForcedDecompose operation is only defined if the top symbol in all the non-variable terms in the decomposed multiequations are the same, and hence this rule application does not introduce new solutions. □

Lemma V.13. *Let $\langle \Delta, L \rangle \rightarrow_{\texttt{InvEq}} \langle \Delta', L' \rangle$ be a derivation in our algorithm. Then, $\langle \Delta, L \rangle$ has a solution if and only if $\langle \Delta', L' \rangle$ has a solution.*

[1] If Δ has m twice in it, then the set Δ^l has two elements, say m^{l_1}, m^{l_2}.

$$\texttt{TwoNonVar:}\quad \frac{\langle \Delta = \Delta' \cup \{\!\{m\}\!\}, L\rangle}{\texttt{solve}(P(m)) \mid \langle \Delta'\texttt{mgu}(m), L\texttt{mgu}(m)\rangle}$$
if $P(m)$ is a reduced 1-CU instance

$$\texttt{CycleOrClash:}\quad \frac{\langle \Delta = \Delta' \cup \{\!\{m\}\!\}, L\rangle}{\texttt{solve}(P(m)) \mid \langle \Delta'\texttt{mgu}(m), L\texttt{mgu}(m)\rangle}$$
if $P(m)$ satisfies the conditions of Claim IV.2 or Claim IV.1.

$$\texttt{InvEq:}\quad \frac{\langle \Delta, L\rangle}{\langle \Delta\sigma, L\sigma\rangle} \quad \text{if } \Delta \models (s = t) \text{ and } \sigma = \texttt{mgu}(s \doteq t)$$

$$\texttt{NoSol:}\quad \frac{\langle \Delta = \Delta' \cup \{\!\{m\}\!\}, L\rangle}{\langle \Delta'\texttt{mgu}(m), L\texttt{mgu}(m)\rangle} \quad \text{if } \texttt{mgu}(\Delta') = \bot$$

Fig. 1. Shrinking rules of the 1-CU algorithm

$$\texttt{ForcedDecompose:}\quad \frac{\langle \Delta = \Gamma \cup \Delta', L\rangle}{\texttt{solve}(P(m_1)\sigma) \mid \ldots \mid \texttt{solve}(P(m_{|\Delta|})\sigma) \mid \texttt{ForcedDecompose}(\langle \Delta, L\rangle, \Gamma, \mathcal{Y})}$$
if $|\texttt{topsymbols}(\Gamma)| = 1$, $\sigma = \{F \to [\bullet]\}$, and Γ is a root class.

$$\texttt{Fail:}\quad \frac{\langle \bot, \bot\rangle}{fail}$$

Fig. 2. Decomposition and failing rules of the 1-CU algorithm

Proof. The correctness of rule `InvEq` follows from Definition V.6 and the fact that $\Delta \models (s = t)$ holds for some state $S = \langle \Delta, L\rangle$ if and only if $\texttt{mgu}(s, t) \leq \theta$, for any solution $\langle m, \theta\rangle$ of S. □

We conclude that each inference rule preserves unifiability. The following Lemma states that our inference system makes *progress*; that is, any derivation starting from an initial state either terminates early (with success) or it reaches a "terminal" state, if we apply the rules exhaustively. The proof is omitted due to space limitations.

Lemma V.14. *Let $S = \langle \Delta, L\rangle$ be a state of our procedure such that no rule can be applied to S. Then, $\Delta = \emptyset$.*

The correctness of the algorithm follows from the previous lemmas.

Theorem V.15. *The algorithm is correct regardless of the rule application strategy.*

G. Runtime analysis

We now establish the time complexity of our procedure. Henceforth we assume that our algorithm applies the rules of Figures 1 and 2 according to the strategy $(\texttt{Shrink}^! \cdot \texttt{ForcedDecompose})^!$, where `Shrink` is the collection of all four shrinking rules. Let us remark that we assume that terms are represented using a Directed Acyclic Graph (DAG). The size of a DAG is defined as its number of nodes. We assume that all the terms (including subterms) involved in our problem are represented as nodes in a single DAG D. Without loss of generality, we assume that D is minimal in size, and then the correspondence between terms and nodes is bijective. Hence, we can then refer to nodes of D and subterms of the problem as if they were the same thing. A crucial observation is that the *number* of terms represented in DAG is preserved by the application of substitutions resulting from the unification of terms of the DAG. This is because application of such substitutions can be achieved by manipulating only the *edges* of the DAG, leaving its nodes untouched.

The only source of difficulty is that our procedure introduces fresh variables from a set $\mathcal{Y}$, and hence the DAG D will grow. However, in this section we prove a bound on the size of D that does not depend on the number of introduced variables. In the rest of the paper we will not refer to D. We instead prove that our algorithm is polynomial w.r.t. the total number of different subterms occurring in the input $\mathcal{I}$, which is the same as the size of the DAG representing $\mathcal{I}$.

The following property, mentioned in the previous sections, is related to the previous observation about the DAG representation for terms: The algorithm does not increase the total number of subterms in the multiequations of Δ that are not freshly introduced variables from $\mathcal{Y}$. Moreover, the `ForcedDecompose` rule strictly reduces this measure. The requirement that Γ be a root class (in the conditions for `ForcedDecompose`) is crucial here.

Lemma V.16. *Let $\langle \Delta_0, L_0\rangle \to^* \langle \Delta_k, L_k\rangle$ be a derivation starting from a valid initial state. Then, $|\texttt{subterms}(\texttt{topterms}(\Delta_k) \setminus \mathcal{Y})| \leq |\texttt{subterms}(\texttt{topterms}(\Delta_0) \setminus \mathcal{Y})|$. Moreover, if $\langle \Delta_0, L_0\rangle \to^* \langle \Delta_k, L_k\rangle \to_{\texttt{ForcedDecompose}} \langle \Delta_{k+1}, L_{k+1}\rangle$ is a derivation from a valid initial state, then, $|\texttt{subterms}(\texttt{topterms}(\Delta_{k+1}))\setminus\mathcal{V}| < |\texttt{subterms}(\texttt{topterms}(\Delta_k)) \setminus \mathcal{V}|$.*

Fig. 3. An example acyclic graph with an unbounded number of m-nodes (red nodes), where each red node has two neighbours that can both reach some f-node (blue node) (using light blue v-nodes).

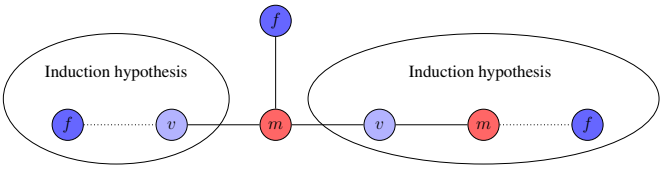

Fig. 4. Proof of induction step: the node m in the middle is removed from the graph to obtain at least two disjoint graphs shown in the left and right.

One of the key facts about our inference system is that the cardinality of Δ in any state generated in a derivation is polynomially bounded. The main part of the proof can be explained as a puzzle: given n blue nodes, assume we have to construct a bipartite graph by adding any number of red nodes and any number of light blue nodes with the following constraints: (a) the graph is acyclic, (b) all red nodes are in one partition and the blue and light blue nodes are in the other partition, and (c) each red node has 3 neighbours, and there is a path from each neighbour to a blue node. The problem is to find a bound on the maximum possible number of red nodes that one can add. As part of the proof of the following lemma, we prove a quadratic bound for the above puzzle. Note that if each red node is required to have only 2 neighbours (with the same property), then the number of red nodes can be unbounded, as demonstrated in the graph in Figure 3.

Lemma V.17. *Let $\langle \Delta_0, L_0 \rangle \to^* \langle \Delta_k, L_k \rangle$ be a derivation starting from a valid initial state. Then, $|\Delta_k| \leq |\texttt{topterms}(\Delta_k) \setminus \mathcal{V}|^2\texttt{maxarity}$.*

Proof. We prove the lemma by induction on the length of the derivation, taking into account the strategy $(\texttt{Shrink}^! \cdot \texttt{ForcedDecompose})^!$. The lemma trivially holds for Δ_0, since $|\Delta_0| = 1$. Note that the property of the lemma is preserved by the application of the four shrinking rules, namely `CycleOrClash`, `TwoNonVar`, `InvEq` and `NoSol`, since these rules do not increase the size of Δ. Hence, it suffices to show that, if (i) a state $S = \langle \Delta_{k-1}, L_{k-1} \rangle$ satisfies the condition of the lemma, (ii) the rules `CycleOrClash`, `TwoNonVar`, `InvEq` and `NoSol` cannot be applied to S, and (iii) $S \to_{\texttt{ForcedDecompose}} \langle \Delta_k, L_k \rangle$, then $|\Delta_k| \leq |\texttt{subterms}(\texttt{topterms}(\Delta_k)) \setminus \mathcal{V}|^2\texttt{maxarity}$.

Consider the graph $G(\Delta_{k-1})$ (Definition V.5). Recall that the set of nodes V of $G(\Delta_{k-1})$ is $\Delta^l_{k-1} \cup \texttt{topterms}(\Delta_{k-1})$. First note that G must be acyclic due to the non-applicability of `InvEq` and hence it is a forest. Therefore, Δ_{k-1} is in fact a set; that is, Δ^l_{k-1} in Definition V.5 is the same as Δ_{k-1}. We refer to the nodes in $V \cap \Delta_{k-1}$, i.e. the nodes of G that are multiequations, as m-nodes, to the nodes of G that are non-variable terms as f-nodes, and to the nodes of G that are variables and v-nodes.

Without loss of generality we assume that G is a tree. Let us give some intuition on how the properties of Δ_{k-1} translate to G. Consider a multiequation $m \in \Delta_{k-1}$ and let $G_1, \ldots, G_s$ be all subtrees adjacent to m. By Definition V.1, computing $P(m)$ involves unifying f-nodes and v-nodes in the same G_i. Note that, since `TwoNonVar` is not applicable to Δ_{k-1}, there must be at least three distinct trees $\{G_i, G_j, G_k\} \subseteq \{G_1, \ldots, G_s\}$ containing f-nodes. Hence, there are *disjoint* paths in G from every m-node to at least 3 f-nodes and, in particular, the degree of m is greater than 3.

We prove that the number M of m-nodes is bounded by T^2, where T is the number of f-nodes. (The following argument is the solution to the above puzzle.) Note that $T = |\texttt{topterms}(\Delta_{k-1}) \setminus \mathcal{V}|$. We use induction on M. For base case, if $M = 1$, then $T \geq 3$ due to the non-applicability of `TwoNonVar`, and hence $M \leq T^2$ in this case.

For the inductive step, note that, again due to the non-applicability of `TwoNonVar`, G must contain an f-node. Among all the f-nodes, pick one, say v_f, that has degree one: such an f-node must exist, since if every f-node has degree at least 2, then the graph will have a cycle (recall that every m-node is connected to at least 3 f-nodes as observed above).

The f-node v_f should have an edge to some m-node, say v_m. If we remove all outgoing edges from v_m, we should get at least 3 disjoint trees (see illustration in Figure 4) that contain f-nodes (due to the non-applicability of `TwoNonVar` as observed above). One of those trees contains just one node – the f-node v_f. Assume that we get $l + l'$ other subtrees $G_1, \ldots, G_l, G_{l+1}, \ldots, G_{l+l'}$, and only the first l contain f-nodes. (In Figure 4, the number l is 2 and l' is 0.)

Let M_i be the number of m-nodes in G_i, and let T_i be the number of f-nodes in G_i, for all i. Hence, $T_{l+1} = \cdots = T_{l+l'} = 0$ holds, and it implies that $M_{l+1} = \cdots = M_{l+l'} = 0$ by, again, non-applicability of `TwoNonVar` (every tree G_i that contains an m node must contain also at least two f-nodes). Thus, the trees G_{l+i}'s consist of just one v-node each. Therefore, we have that $M = M_1 + \ldots + M_l + 1$ and that $T = T_1 + \ldots + T_l + 1$, where the last one is for the node v_f.

By assumption, every G_i contains an f-node, i.e. $T_i > 0$, and fewer m-nodes than m. We want to use the induction hypothesis, but before we can do that we need to make sure each G_i satisfies our original constraint that every m-node has at least 3 neighbours such that there is a path from each of those neighbours to a different f-node. Removing node v_m may cause violation of this property. This happens if v_m has an edge to a v-node in G_i. This is the case in the illustration in Figure 4 for the subgraphs that are encircled. We treat that v-node as an f-node and then apply induction hypothesis to

get $M_i \leq (T_i + 1)^2$ for all i. Thus, we now have

$$\begin{aligned} M &= (\sum_{i=1}^{l} M_i) + 1 \\ &\leq \sum_{i=1}^{l} (T_i + 1)^2 + 1 \\ &= ((\sum_{i=1}^{l} T_i) + 1)^2 - \sum_{j \neq k} 2T_j T_k + l \\ &= T^2 - \sum_{j \neq k} 2T_j T_k + l \\ &\leq T^2 - (\sum_{j \neq k} 2) + l \\ &= T^2 - l(l-1) + l \quad \leq \quad T^2 \end{aligned}$$

For the last step in the above derivation, note that $l \geq 2$ and hence, $l \leq l(l-1)$ is always true.

Altogether implies that $|\Delta_{k-1}|$ is bounded by $|\texttt{topterms}(\Delta_{k-1}) \setminus \mathcal{V}|^2$ and, since the application of `ForcedDecompose` increases the number of multiequations by at most a factor of `maxarity`, we have that $|\Delta_k| \leq |\texttt{topterms}(\Delta_k) \setminus \mathcal{V}|^2 \texttt{maxarity}$. □

Every subproblem generated during a derivation will have polynomial size.

Lemma V.18. *Let $\langle \Delta_0, L_0 \rangle \rightarrow^* \langle \Delta_k, L_k \rangle$ be its corresponding derivation in our algorithm. Then, for every multiequation $m \in |\Delta_k|$, $P(m)$ has polynomial size with respect to $||\mathcal{I}|$ and k.*

Proof. The Lemma follows from the fact that, thanks to the DAG representation, the terms in L_k and $m(\texttt{mgu}(\Delta_k \setminus \{\!\{m\}\!\}))$ have polynomial size in k and $||\mathcal{I}||$. □

Finally, our main result is the following. Its proof follows from Lemmas V.16 and V.17.

Theorem V.19. *The 1-CU problem is solvable in polynomial time assuming existence of a polynomial time procedure* `solve` *for 1-CU instances with at most two non-variable terms in the right hand-side of equations.*

VI. One Context Unification Problems Solvable in Polynomial Time

The results in the previous section (Theorem V.19 and Lemmas IV.2 and IV.1) give us a reduction from the general 1-CU problem to a reduced 1-cu problem.

So far, we relied on an oracle to solve reduced instances. We will present special classes of 1-CU problems whose reduced instances can be solved in polynomial time. Certain reduced instances could have only one or two "non-trivial" equations. So, we first present results on solving 1-CU instances that have exactly one or two equations. These will help in solving more general reduced instances later.

A. One Equation 1-CU Problem

We prove that a single equation 1-CU problem can be efficiently solved. If the equation is of the form $F(s) = C[F(t)]$ and C is nonempty, then we can use Theorem IV.1 to solve it. Next, consider an equation of the form $F(C[F(s)]) = t$. It has the nice property that the hole position of any context that is a solution for F can not be an extension of a *nonlinear* positions in t. A position p is *nonlinear* in t if there exists another position $q \neq p$ such that $t|_p = t|_q$. We also call $t|_p$ a non-linear subterm of t. Note that, for every term t and non-linear position p of t, every position $q \in \texttt{pos}(t)$ that is an extension of p is also non-linear.

Lemma VI.1. *Let $\mathcal{I}$ be a 1-CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t (but F can occur in C). Let $P = \{p \in \texttt{pos}(t) \mid t|_p = v$ and v is a non-linear subterm of $t\}$. Then, $\forall p \in P : \texttt{hp}(F\sigma) \neq p$, for every solution σ of $F(C[F(s)]) \doteq t$.*

Proof. Let σ be a solution contradicting the conditions of the lemma, i.e. there is a term v occurring at two distinct positions p and q of t such that $\texttt{hp}(F\sigma) = p$. It follows that $F\sigma = t\sigma[\bullet]_p$, $t\sigma[\bullet]_p|_q = v\sigma$ and hence we have $v\sigma = C[F(s)]\sigma = C\sigma[F\sigma(s\sigma)] = C\sigma[t\sigma[\bullet]_p[s\sigma]] = C\sigma[t\sigma[\bullet]_p[s\sigma]]|_{\texttt{hp}(C).q} = v\sigma|_{\texttt{hp}(C).q} = v\sigma$, a contradiction. □

There are only a (linear number of) linear positions in a term. (In contrast, there can be exponentially many non-linear positions in a term). It follows from Lemma VI.1 that for the equation $F(C[F(s)]) \doteq t$, we only need to test the (small number of) linear positions as possible hole positions. In fact, we can enumerate a *complete* set of unifiers: a set of unifiers is *complete* if any other unifier (for the problem) is an instance of some unifier in the set. Here, a unifier is allowed to instantiate a context variable F in terms of a new context variable F'. The following lemma follows from Lemma VI.1 above.

Lemma VI.2. *Let $\mathcal{I}$ be a 1-CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t. Then, a complete set of unifiers U of $\mathcal{I}$ of polynomial size can be computed in polynomial time. Any substitution σ in U satisfies one of the two conditions below:*

1) *Either $F\sigma = t[\bullet]_p$, with $p \in \texttt{pos}(t)$,*
2) *Or $\sigma = \{F \mapsto t[F'(\bullet)]_q,\ x \mapsto F'(C[t[F'(s)]_q])\}$, where x does not occur in $F(C[F(s)])$, $t|_q = x$, and F' is a new context variable different from F.*

Using the special cases in Lemma VI.2 and Theorem IV.1, we can now prove that the special case when we have only one equation can be solved.

Claim VI.1 (1-eqn)**.** *Let $\mathcal{I}$ be a 1-CU instance consisting of one single equation $F(s) \doteq t$. Then, $\mathcal{I}$ can be solved in polynomial time.*

Proof. Note that the case where F occurs in t holds by Theorem IV.1. Note that the case where F occurs in s holds by the previous lemma. So, we are left with the case where

F does not occur in s or t. In this case, it is easy to see that a unifier exists iff one exists where $\mathtt{hp}(F\sigma) \in \mathtt{pos}(t)$. To determine existence of solutions where $\mathtt{hp}(F\sigma) \in \mathtt{pos}(t)$, we just need to find a subterm of t that unifies with s. Whereas the total number of subterms of t might be exponential due to the DAG representation, the number of *distinct* subterms is polynomial. Hence, we can simply check, in polynomial time, all terms $v \in \mathtt{subterms}(t)$ that unify with s. □

B. Two Equation 1-CU Problem

We show that 1-CU problems consisting of exactly two equations can be solved efficiently, but under a technical condition. To motivate the technical condition, note that the instance $\mathcal{I} = \{F(r_1) = z, F(r_2) = z\}$ can encode an *arbitrary* 1-CU instance $\mathcal{I}' = \{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\}$, by having $r_1 = C[F(s_1), \ldots, F(s_n)]$ and $r_2 = C[t_1, \ldots, t_n]$. So, two equation case is as hard as an arbitrary number of equations. However, if we are interested in solutions σ so that $\mathtt{hp}(F\sigma)$ is not below a position p where both right-hand side terms have the same variable, then we can solve two equation problems.

First, consider two equations in which one has a nested F on one side. The proof of the following lemma follows from Lemma VI.2, Theorem IV.1, and Claim VI.1.

Lemma VI.3. *Let $\mathcal{I} = \{F(C[F(u)]) \doteq s, F(v) \doteq t\}$ be a 1-CU instance such that s, t are non-variable terms not containing F. If we are only interested in solutions σ such that*

$$\mathtt{not}(\exists p, x : (s|_p = t|_p = x \text{ and } \mathtt{hp}(F\sigma) > p)),$$

then such a solution σ of $\mathcal{I}$ can be found in polynomial time.

Note that the ignored solutions make s, t equal, but not all solutions that make s, t equal are ignored.

Using the previous result, we can now solve the two equation case, but also when it is extended with some "variable definitions", under an extension of the same technical condition.

Claim VI.2 (2-nonvar)**.** *Let $\mathcal{I}$ be a 1-CU instance of the form $\bigcup_{i=1}^{n}(\{F(u_i) \doteq x_i\}) \cup \{F(v_1) \doteq s, F(v_2) \doteq t\}$ such that the terms s,t, and $u_1, \ldots, u_n$ do not contain F, and $s, t, x_1, \ldots, x_n$ are pairwise different. If we are only interested in solutions σ such that*

$$\begin{aligned}\mathtt{not}(\ & s, t \text{ do not contain any } x_i, \text{ and} \\ & v_1 \text{ or } v_2 \text{ contains either } F \text{ or some } x_i, \text{ and} \\ & \exists p, x : (s|_p = t|_p = x \text{ and } \mathtt{hp}(F\sigma) > p)\ ),\end{aligned}$$

then such a solution σ for $\mathcal{I}$ can be found in polynomial time.

Proof. Note that, if some x_i occurs in either s or t, it has to occur properly, and the lemma follows from Lemma IV.2.

Hence assume that s, t do not contain any x_i. We define the instance $\mathcal{I}'$ as the result of exhaustively replacing x_i by $F(u_i)$ everywhere. If there is a cycle and this replacement can not be performed exhaustively, then $\mathcal{I}' = \bot$. Note that every solution of $\mathcal{I}'$ can be easily extended to be a solution of $\mathcal{I}$, that $\mathcal{I}'$ can be obtained from $\mathcal{I}$ in polynomial time, and that $||\mathcal{I}'||$ is polynomial w.r.t. $||\mathcal{I}||$. Moreover, $\mathcal{I}'$ is either of the form

1) $\bot$, or
2) $\{F(v_1) \doteq s, F(v_2) \doteq t\}$, where v_1, v_2, s, t do not contain F or
3) $\{F(v_1) \doteq s, F(v_2) \doteq t\}$, where s, t do not contain F and either v_1 or v_2 contains F.

In case 1 $\mathcal{I}'$ has no solutions. In case 2, $\mathcal{I}'$ is a 2-restricted 1-CU instance that we have shown can be efficiently solve in a companion paper [8]. Finally, case 3 follows from Lemma VI.3, while noting that the new technical condition maps to the condition in that lemma. □

C. Disjoint Variables and Constant Number of Equations

Now that we have results for one and two equations, we next present sufficient conditions for polynomial time solvability of 1-CU problem. In each case, we will show that we can solve the corresponding reduced problems in polynomial time. Let us fix the following notation for the rest of the section.

$$\begin{aligned}\mathcal{I} &= \{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\} \\ \mathcal{V}_1 &= \text{Set of all first-order variables in } s_1, \ldots, s_n \\ \mathcal{V}_2 &= \text{Set of all first-order variables in } t_1, \ldots, t_n \\ \mathcal{T}_1 &= \{u \mid u = F(u'),\ s_i|_p = u \text{ for some } i, p, u'\} \\ \mathcal{T}_2 &= \{u \mid u = F(u'),\ t_i|_p = u \text{ for some } i, p, u'\}\end{aligned}$$

Instance where $\mathcal{T}_2 \neq \emptyset$ can be solved in polynomial time using Theorem IV.1. Hence, the proofs below will implicitly assume $\mathcal{T}_2 = \emptyset$.

Theorem VI.4. *The class of 1-CU instances where $\mathcal{T}_1 = \emptyset$ and $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ is solvable in polynomial time.*

Proof. (Sketch) The reduced instances generated by our procedure from any instance from this class will also belong to this class. We solve reduced instances by unifying the left-hand sides corresponding to equal right-hand sides, and applying the unifier to the rest. Under the assumption that $\mathcal{T}_1 = \emptyset$, this unifier will be a first-order substitution. Since $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, the right-hand side terms do not change. The simplified reduced instance would be solved by either Theorem IV.1, or using Claim VI.1, or using Claim VI.2. □

Theorem VI.4 subsumes the $\mathcal{V}_1 = \emptyset$ case, for which the best known algorithm is the NPtime procedure from [7]. In fact, we showed a family of examples from such a class in the introduction that would cause the algorithm from [7] to run in worst-case exponential time, whereas Theorem VI.4 would solve it in polynomial time.

In general, we want to use Claim VI.2 to solve reduced 1-CU instances generated by our procedure. However, due to the technical condition in Claim VI.2, we could miss certain solutions that make two left-hand side terms (or equivalently, right-hand side terms) equal. But, if we have a *constant* number of equations, then we could systematically explore all possible combinations (of which left-hand sides are equal and which not) to get a complete procedure.

Theorem VI.5. *The class of 1-CU instances where $\mathcal{T}_1 = \emptyset$ and the cardinality $|\mathcal{I}|$ is assumed to be a constant k (independent of the input problem size) is solvable in polynomial time.*

Proof. For each possible equivalence relation $\sim$ on $\mathcal{I}$, we generate an instance $\mathcal{I}'$ and solve it. To generate $\mathcal{I}'$ from $\mathcal{I}$ and $\sim$, whenever we have $F(s) \doteq t \ \sim \ F(s') \doteq t'$, we (first-order) unify t, t' and s, s' and apply the most-general unifier to all the equations in $\mathcal{I}$. Let $\mathcal{I}'$ be the new instance. Note that the number of equations in $\mathcal{I}'$ is equal to the number of equivalence classes induced by $\sim$. We then apply our inference rules on $\mathcal{I}'$. We need to show how to solve the reduced instances generated from $\mathcal{I}'$. We use the following algorithm for this purpose: If the reduced instance has two equations with identical right-hand sides, then we return "no solution" (for that reduced instance). If not, the reduced instance should have the form mentioned in Claim VI.2 and we solve it using Claim VI.2.

We claim that $\mathcal{I}$ has a solution iff for some $\sim$, the procedure above finds a solution. If the procedure above finds a solution, then clearly $\mathcal{I}$ has a solution. If $\mathcal{I}$ has a solution σ, define $\sim$ as follows: $F(s) \doteq t \ \sim \ F(s') \doteq t'$ if $s\sigma = s'\sigma$. When we use our procedure to solve instance $\mathcal{I}'$ generated from $\mathcal{I}$ using $\sim$, we are guaranteed to find a solution. This is because, while Claim VI.2 can miss solutions, it never misses a solution that makes all left-hand side terms (in $\mathcal{I}'$) different. □

We can extend the previous result to allow $\mathcal{T}_1 \neq \emptyset$.

Corollary VI.6. *The class of 1-CU instances where the cardinality $|\mathcal{T}_1|+|\mathcal{I}|$ is assumed to be a constant k (independent of the input problem size) is solvable in polynomial time.*

Proof. If $\mathcal{T}_1 \neq \emptyset$, then we introduce new variables and get an instance where $\mathcal{T}_1 = \emptyset$. Specifically, if $u \in \mathcal{T}_1$, then we add the equation $u \doteq x$ to the instance (where x is a new variable), and replace u by x everywhere else. Applying this repeatedly, we get an instance for which $\mathcal{T}_1 = \emptyset$, and now we can use Theorem VI.5. □

D. Left- and Right-Ground 1-CU Problems

Right-ground 1-CU instances can be efficiently solved.

Theorem VI.7. *The class of 1-CU instances where $\mathcal{V}_2 = \emptyset$ is solvable in polynomial time.*

Proof. If the instance has 2 equations, we solve it using Claim VI.2. Note that we do not miss any solutions due to the technical condition. If the instance has 3 or more equations, we use our inference rules. The procedure will not generate any reduced instances to solve since all (≥ 3) right-hand side terms are always ground (in any generated subproblem). □

The result in Theorem VI.7 was already known [6], but now we can obtain a new proof using our procedure. We can generalize Theorem VI.7 to a class that does not require all right-hand sides terms to be ground, but just two of them.

Theorem VI.8. *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I}' \cup \{F(s_1) \doteq s, F(s_2) \doteq t\}$, where s, t are distinct ground terms. Then $\mathcal{I}$ can be solved in polynomial time.*

Proof. Clearly, for every solution σ, $\mathtt{hp}(F\sigma) \in \mathtt{pos}(s) \cap \mathtt{pos}(t)$. Hence, $l = |\mathtt{hp}(F\sigma)|$ is polynomial even when s and t are DAGs. We now argue that, once l is fixed, there is only *one* choice for $\mathtt{hp}(F\sigma)$, and thus also $F\sigma$. If $l = 0$ the claim holds trivially. Hence assume $l > 0$ and, without loss of generality, let $s = f(s_1, s_2)$ and $t = f(t_1, t_2)$. Note that, since $s \neq t$, $\exists i \in \{1, 2\} : t_i \neq s_i$ holds. Also note that, if $\forall i \in \{1, 2\} : t_i \neq s_i$ holds then there is no solution of length l. In the remaining case either $s_1 = t_1$ or $s_2 = t_2$, say $s_2 = t_2$. Then $\mathtt{hp}(F\sigma) = 1.\mathtt{hp}(F')$, $F\sigma = t[F'(\bullet)]_1$, and, since $|\mathtt{hp}(F')| < l$ the claim holds by induction hypothesis.

Since there are only polynomially many choices for $\mathtt{hp}(F\sigma)$, we can enumerate them all and solve $\mathcal{I}$ using polynomially many calls to a first-order unification procedure. □

Left-ground 1-CU instances can also be efficiently solved.

Theorem VI.9. *The class of 1-CU instances where $\mathcal{V}_1 = \mathcal{T}_1 = \emptyset$ is solvable in polynomial time.*

Proof. We apply our inference rules and use the following algorithm to solve the generated reduced instances: If the reduced instance has two equations with identical right-hand sides, then we return "no solution" (for the reduced instance) if the left-hand sides are not identical, and we delete one equation from $\mathcal{I}$ if the left-hand sides are identical. Let $\mathcal{I}'$ be the new reduced instance. Let $F(u) \doteq x$ be an equation in $\mathcal{I}'$. If x occurs in any other right-hand side term in $\mathcal{I}'$, then we use Theorem IV.1. Since $\mathcal{V}_1 = \emptyset$, x can not occur in left-hand side terms. So, we can remove $F(u) \doteq x$ from $\mathcal{I}'$. Hence, finally $\mathcal{I}'$ will have at most two equations. If $\mathcal{I}'$ has zero or one equations, then we are done by Claim VI.1. So, assume $\mathcal{I}'$ has exactly two equations, say $F(v) \doteq s$ and $F(w) \doteq t$. If $v \neq w$, we can use Claim VI.2 (we will not miss any solutions since v and w are ground and therefore they will be different in all solutions.) If $v = w$, then we unify s and t and solve the resulting one equation. □

Finally, we generalize Theorem VI.9 to remove the requirement $\mathcal{T}_1 = \emptyset$, and consider instances where only $\mathcal{V}_1 = \emptyset$. Note that the context matching problem with a constant number of context variable was solved in polynomial time in [6], but the class $\mathcal{V}_1 = \emptyset$ falls out of the class solved in [6].

Theorem VI.10. *The class of 1-CU instances where $\mathcal{V}_1 = \emptyset$ is solvable in polynomial time.*

Proof. We follow along the lines of the proof of Theorem VI.9. We now use the following algorithm to solve any generated reduced instance, say of form given in Equation 1: If the reduced instance $\mathcal{I}'$ has two equations $F(u) \doteq r$ and $F(u') \doteq r$ with identical right-hand sides, then we use (first-order) decomposition rule on $u \doteq u'$ exhaustively to get a set of equations, where each equation is of the form $F(u_1) \doteq u_2$, where $F(u_1) \in \mathcal{T}_1$ and u_2 is a first-order ground term whose top symbol is not F, but u_2 possibly contains F. We add these equations to the reduced instance and remove one of the original equations, say $F(u) \doteq r$, from $\mathcal{I}'$. If u_2 contains F,

then we can solve $\mathcal{I}'$ using Theorem IV.1. Hence, assume the first-order ground term u_2 does not contain F.

Let $\mathcal{I}''$ denote the reduced instance we get after we have processed all repeated right-hand side equations as above. Let $F(u) \doteq x$ be an equation in $\mathcal{I}''$. If x occurs in any other right-hand side term in $\mathcal{I}''$, then we use Theorem IV.1. Since $\mathcal{V}_1 = \emptyset$, x can not occur in left-hand side terms. So, we can remove $F(u) \doteq x$ from $\mathcal{I}''$ without changing its solvability. Hence, finally $\mathcal{I}''$ will have the form:

$$F(v) \doteq s, F(w) \doteq t, F(u_1) \doteq u'_1, \ldots, F(u_k) \doteq u'_k$$

where s, t are the non-variable terms in the original reduced instance, and the last k equations are generated from unification of left-hand sides. The terms $u'_1, \ldots, u'_k$ are all ground. We can assume all u'_i are distinct, otherwise we could just repeat the above process.

If $k \geq 2$, we are done by Theorem VI.8.

If $k = 0$, then we solve using Claim VI.2, but to overcome the incompleteness there, we additionally solve the instance $\mathcal{I}'''$ obtained by unifying s and t, and then adding equations obtained by decomposing $v \doteq w$. The instance $\mathcal{I}'''$ has at most one non-ground right-hand side. If it has two or more distinct ground right-hand sides, then again we can use Theorem VI.8. If it has exactly one ground right-hand side, and one non-ground, then we Claim VI.2 again, and this time, the instance we need to solve to overcome the incompleteness there, will have only ground right-hand sides (for which we can use either Theorem VI.8 or Claim VI.1).

If $k = 1$, then we have 3 equations, where one has a ground right-hand side. We use our inference system on these three equations to obtain new reduced instances. Each new reduced instance has 3 equations: one has a variable on the right, one has a ground term, and the third can have an arbitrary first-order term. We can again remove the equation with a variable on the right and get instances like in case $k = 0$ above. □

E. The General 1-CU Problem

We can actually use our inference system presented in Section V to solve the general 1-CU problem in stages as follows: (a) first we reduce an instance $\mathcal{I}$ to polynomially many reduced instances $\mathcal{I}_1, \ldots, \mathcal{I}_k$, (b) for each reduced instance $\mathcal{I}_j$, we unify left-hand sides that have equal right-hand sides, apply the substitution and obtain new reduced instances $\mathcal{I}'_1, \ldots, \mathcal{I}'_k$, and (c) finally we apply our procedure recursively on each new reduced instance. We *conjecture* that this procedure yields a polynomial time algorithm for the general 1-CU problem.

VII. Conclusion

We presented an inference system for solving the one context unification problem. We proved that the inference system yields a polynomial time algorithm for several classes of one context unification problems. The inference system itself has many interesting features: the proof search continues along one main branch, while the side branches are immediately terminated using polynomial time procedures. The main branch itself can generate a large number of subproblems, but their number is bounded using a graph argument that might be of independent interest.

Acknowledgment

The authors thank Guillem Godoy, Carles Creus, and Lander Ramos for useful discussions, and the anonymous reviewers for their valuable comments.

References

[1] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[2] C. Creus, A. Gascón, and G. Godoy. One-context Unification with STG-Compressed Terms is in NP. In *RTA*, pages 149–164, 2012.

[3] R. Dabrowski and W. Plandowski. Solving Two-Variable Word Equations (extended abstract). In *ICALP*, pages 408–419, 2004.

[4] R. Dabrowski and W. Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011.

[5] K. Erk and J. Niehren. Dominance constraints in stratified context unification. *Inf. Process. Lett.*, 101(4):141–147, 2007.

[6] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context Matching for Compressed Terms. In *LICS*, pages 93–102, 2008.

[7] A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context Unification with One Context Variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

[8] Adrià Gascón, Manfred Schmidt-Schauss, and Ashish Tiwari. Two-Restricted One Context Unification in Polynomial Time. Available on: https://www.cs.upc.edu/~agascon/papers/1cu-2r.pdf, 2015.

[9] W. D. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

[10] S. Gulwani and A. Tiwari. Computing Procedure Summaries for Interprocedural Analysis. In *ESOP*, pages 253–267, 2007.

[11] A. Jeż. One-Variable Word Equations in Linear Time. In *ICALP*, pages 324–335, 2013.

[12] A. Jeż. Context unification is in PSPACE. In *ICALP*, volume 8573 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 2014.

[13] J. Levy, J. Niehren, and M. Villaret. Well-Nested Context Unification. In *CADE*, pages 149–163, 2005.

[14] J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of Bounded Second-Order Unification and Stratified Context Unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.

[15] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977.

[16] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[17] S. E. Obono, P. Goralcik, and M. N. Maksimenko. Efficient solving of the word equations in one variable. In *MFCS*, pages 336–341, 1994.

[18] M. Paterson and M. N. Wegman. Linear unification. *J. of Computer and System Sciences*, 16:158–167, 1978.

[19] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004.

[20] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[21] M. Schmidt-Schauß and J. Stuber. The complexity of linear and stratified context matching problems. *Theory Comput. Syst.*, 37(6):717–740, 2004.