Solving Context Related Unification Problems Using Regular Languages

Tomer Líbal

Institute of Computer Languages (E185)
Vienna University of Technology
Favoritenstraße 9, 1040 Vienna, Austria
shaolin@logic.at

Abstract

G. P. Huet ([7]) gave a pre-unification algorithm, which does not terminate, for unrestricted higher-order unification problems. In this paper we adapt Huet's pre-unification algorithm to context and bounded second-order unification problems. By showing termination of the algorithm, we give a compact and elegant proof for the decidability of the bounded second-order unification problem ([16]). This can be extended to a proof of the decidability of the stratified context unification problem ([15]). In light of our previous results, we were able to provide a description of the infinitely many pre-unifiers of context related unification problems by the use of regular languages. Thus, we postulate that the integration of a subset of algorithms, which decide infinitary unification problems, can be fully automated in deduction systems, such as resolution ([14]).

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Decision algorithms, Second-order unification, Resolution, Pre-unification, Regular languages

1 Introduction

Syntactic unification is the process of finding a substitution which makes two terms, both containing variables, syntactically equal. J. A. Robinson ([14]) gave an algorithm, which decides the first order unification problem, as part of his resolution method. The problem was shown to be undecidable even for unary second-order unification problems ([4]). Despite that, G. S. Makanin ([10]) showed that the string unification problem, a restricted second-order unification problem, is decidable.

The context unification problem ([2]) is a generalization of string unification, that poses restrictions on both the second-order problems and on the searched for substitutions. The first restriction is, that in addition to first order variables, only unary second-order variables, called context variables, are allowed. The second restriction is that context variables are mapped to contexts, which have exactly one occurrence of the bound variable.

Context unification has applications in several fields in computer science, such as computational linguistics ([11],[12]) and rewrite systems ([13]). The decidability question of context unification is still an open problem. Despite that, there are several sub-classes of the problem, which were shown to be decidable ([15],[2],[8],[18]).

A related unification problem is bounded second-order unification. The range of unifiers for these problems contains terms with a maximum number of bound variables. In [16] it was shown, that these problems can be reduced to $Z-$context unification, which allows context variables to be mapped to normal terms as well.

A general unification algorithm for higher-order logic, which does not terminate, was given by G. P. Huet ([7]). Huet also introduced the notion of pre-unifiers, which is based


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on the fact that equations of two variables, called flat equations, are trivially unifiable. Pre-unifiers are unifiers up to the equality of two higher-order variables.

Second-order unification problems in general, and the problems discussed in this paper in particular, are infinitary unification problems, which means that such problems possess infinitely-many most general unifiers. This is the reason why decision procedures for these problems look for subsets of all most general unifiers ([10],[15],[16],[19]), which are the sets of all minimal unifiers ([17]). The infinitary nature of these problems is due to unifiable cycles.

In this paper we adapt the pre-unification algorithm to context unification problems and take a different approach for dealing with cycles. This approach restricts the depth of terms the algorithm can generate and in addition, adds rules that eliminate cycles by using regular languages of contexts.

The first result in this paper is the enumeration of all pre-unifiers of context and $Z-$context unification problems in a parallel way to Huet's algorithm. An enumeration of all most general unifiers of context unification problems is given in [8] while in [16] an enumeration of all pre-unifiers of $Z-$context unification problems is given using a more technically involved algorithm.

As a second result we introduce new proofs for the decidability of both the stratified context unification problem ([15]) and the bounded second-order unification problem ([16]). The current proofs for these problems involve a very specialized algorithms consisting of many derivation rules. The result is that the proofs of soundness and completeness are given from scratch and are pretty involved. As the algorithm we utilize in our proofs is based on Huet's, we obtain both a simpler algorithm and we reuse existing soundness and completeness proofs.

The main contribution of this paper is the enumeration of all pre-unifiers of these problems by using regular languages of contexts. A finite representation of all unifiers for subclasses of the word problem by using graphs and regular expressions is given in [1]. Since this task is impossible for the general word problem ([5]) and certainly also for context unification, our aim is to describe infinite inference sequences by using regular expressions. Using this method, we can incrementaly obtain finite representations of all pre-unifiers by having smaller unification problems but over regular contexts. We mention two possible applications of such an enumeration, a practical one and a theoretical one. First, the algorithm can be used in fully-automated unification based procedures in a practical way as unlike algorithms which compute minimal unifiers only, it computes all pre-unifiers and unlike non-terminating algorithms ([7]), it can be used eagerly ([6]). One way of doing it is to eagerly compute the regular contexts and give a dynamic bound on folding them using the current search space. Second, since regular languages have many closure properties and membership is decidable, it might be possible to shed more light on the decidability of the context unification problem.

Another advantage of the algorithm, which is relatively compact and with clearly defined bounds, is that it can be easily translated into a computer program. Such an implementation is currently under development in the GAPT[1] framework.

The paper is organized as follows: In the first section we give definitions and some results with regard to context unification and pre-unification, including an adaptation of the pre-unification algorithm to contexts and $Z-$contexts. The second section is dedicated to the presentation of a modified algorithm, which restricts the term and context depths

[1] http://code.google.com/p/gapt/

and eliminate cycles by the introduction of regular languages of contexts. In the third section we prove that the adapted Huet's algorithm and our algorithm compute the same pre-unifiers of the discussed problems. The last section is mainly based on results from papers discussing minimal unifiers. In it we show that if the only concern is the decidability of the unification problems, we can restrict the algorithm from the second section such that it terminates. Two results of this section are the compact proofs of the decidability of the stratified context unification problem and the bounded second-order unification problem.

2 Preliminaries

The algorithm that decides the bounded second-order unification problem in [16] is indirect in the sense that it requires the problem to be transformed first into a Z-context problem. A direct treatment for this problem (and for the more general bounded higher-order unification problem) appears in [19]. Because we deal with second-order problems and due to the limitation of space, we will follow the indirect approach and assume that a bounded second-order unification problem is first transformed into a Z-context unification problem using the method described in [16].

2.1 Contexts and Z-Contexts

The following notations will follow those in [15] and [16].

Let Σ be a signature of function symbols, denoted f, g, h and let ar be a function mapping Σ to natural numbers, which denotes its arity. Symbols with arity 0 are called constant symbols. Let V_1 be the set of first order variables, denoted x, y, z and V_2 be the set of context variables, denoted X, Y, Z such that both are disjoint from each other and from the signature. If we want to refer to a variable in $V_1 \cup V_2$, we will denote it by w. We will denote a sequence of n terms (or variables, positions, etc.) using the notation $\overline{t_n}$. All denotations may contain superscripts and subscripts. *Terms* are formed by the grammar:

$$t ::= x | f(t, .., t) | X(t) \tag{1}$$

Terms are denoted s, t, v, u. If a function symbol or a context variable has only one argument, we may drop the parentheses. A term of the form $X(t)$ or x is called a *flex term* while $f(\overline{t_n})$ is called *rigid*. A variable not occurring as an argument to another variable in a term is called a *rigid variable occurrence*. Positions (denoted p) of sub-terms are defined as usual as sequences of natural numbers such that $t|_\epsilon = t$ and $t|_{p.i} = s_i$ if $t|_p = f(\overline{s_n})$ and $0 < i \leq n$. Note that f must be a function symbol and cannot be a variable. The set of all positions of t such that no sub-term in prefix position is a variable is denoted $Pos(t)$. The size $|p|$ of a position p is the number of natural numbers in it. The depth of a term t is the maximal size of a position in it and is denoted by $d(t)$. If there is a position p_3 such that $p_1 = p_2.p_3$ and $|p_3| = n$, then we notate $p_2 \gg_n p_1$. $Var(s)$ is the set of all variables in s. $Var_1(s) = Var(s) \cap V_1$. $Var_2(s) = Var(s) \cap V_2$. The term s is first order if $Var_2(s) = \emptyset$ and it is ground if $Var(s) = \emptyset$. *Contexts* extend the term grammar by:

$$C[.] ::= [.] | f(t, .., C[.], .., t) | X(C[.]) \tag{2}$$

and denotes terms containing exactly one occurrence of [.]. *Z-contexts* extend the term grammar by:

$$C[.] ::= [.] | f(t, .., C[.], .., t) | X(C[.]) | t \tag{3}$$

and represents terms where [.] occurs at most once. The notation $C(t)$ means replacing [.] in $C[.]$ with t. A *ground (Z-)context* is a (Z-)context with no variables. A *(ground) (Z-)context substitution*, denoted σ, maps first order variables to (ground) terms and context variables to (ground) (Z-)contexts. Variables not mapped to terms or (Z-)contexts are mapped to themselves. Id denotes the empty substitution. $t\sigma$ is defined as the simultaneous replacement of all first order variables $x \in Var(t)$ with $\sigma(x)$ and all sub-terms $X(s)$ of t with $\sigma(X)\sigma(s)$. Composition of substitutions is the usual function composition and is denoted by $\circ$. In the rest of the paper a (ground) substitution will refer to a (ground) Z-contextsubstitution. A *(Z-)context unification problem (`(Z)CUP`)* is a set of equations Γ, denoted as $\{s_1 \doteq t_1, .., s_n \doteq t_n\}$, with $\doteq$ a symmetric relation. Applying a substitution to a `(Z)CUP` means applying the substitution to the two terms of each equation in the `(Z)CUP`. An equation with two flex terms is called a *flex-flex equation*. A flex-flex equation whose two flex terms have context variables as heads is called a *flat equation*.

2.2 Pre-unification

In this section we will adapt the pre-unification algorithm ([7]) presented in [20] to (Z-)context unification problems. The definitions are taken from [20]. A *unifier* of Γ is a substitution σ such that for all $s \doteq t \in \Gamma$, $s\sigma = t\sigma$. Having a congruence relation $=_{con}$ on the terms as $\{(u, v)|$ both are flexible terms$\}$, a *pre-unifier* is obtained similarly but with $s\sigma =_{con} t\sigma$. For a `(Z)CUP` P, a unifier σ is called *more general than* a unifier θ if there is a substitution τ such that $(w)\sigma \circ \tau = (w)\theta$ for all $w \in Var(P)$. A pair $w \doteq t$ is called *solved* in a `(Z)CUP` P if t is a rigid term and w does not occurs elsewhere in t or P. The pair is in *pre-solved form* if it is either solved or flex-flex. A *partial binding* ($\mathtt{PB}(t)$) for $t = f(\overline{s_n})$ is the set of contexts $\{f(x_1, .., H_k[.], .., x_n)|0 < k \leq n\}$. The $x_i \in V_1$ and $H_k \in V_2$ are new variables. If we allow Z-contexts, then we also add the term $f(x_1, .., x_k, .., x_n)$ to the set of partial bindings. Let S be a `(Z)CUP` and σ a substitution, the pre-unification algorithm `PUA` consists of the set of transformations given in Fig. 1.

$$\frac{\Gamma \cup \{A \doteq A\}}{\Gamma}\ \texttt{(Delete)} \qquad \frac{\Gamma \cup \{f(\overline{s_n}) \doteq f(\overline{t_n})\}}{\Gamma \cup \{s_1 \doteq t_1, .., s_n \doteq t_n\}}\ \texttt{(Decomp)}$$

$$\frac{\Gamma \cup \{x \doteq t\} \qquad x \notin Var(t)}{\Gamma\{x \mapsto t\}}\ \texttt{(Bind}_x\texttt{)} \qquad \frac{\Gamma \cup \{X([.]) \doteq t\} \qquad X \notin Var(t)}{\Gamma\{X \mapsto t\}}\ \texttt{(Bind}_X\texttt{)}$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(P), u \in \mathtt{PB}(t)}{\Gamma \cup \{X([.]) \doteq u, X(s) \doteq t\}}\ \texttt{(Imitate)}$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(P)}{\Gamma \cup \{X([.]) \doteq [.], X(s) \doteq t\}}\ \texttt{(Project)}$$

Figure 1 `PUA` - Huet's Pre-Unification Rules

We sometimes refer to the rules `(Bind`$_x$`)` and `(Bind`$_X$`)` as the `(Bind)` rule.

Given a derivation φ of `PUA` on a `(Z)CUP` P, the substitution *obtained* by φ is the composition of all substitutions generated by the `(Bind)` rules and restricted to the variables in P. A `(Z)CUP` is *solved* if all its equations are solved in it and it is *pre-solved* if all equations are in pre-solved form. A unifier is obtained from a solved form by mapping the variable of each equation to the rigid term in the equation and restricting the domain to the original variables. A pre-unifier is obtained from a pre-solved form by using the solved equations

only. A *complete set of pre-unifiers* for a (Z)CUP P is a set of substitutions, such that for every unifier σ of P, there is a substitution $\theta \in \mathtt{CSU}$ and a substitution δ such that $\sigma = \theta \circ \delta$. A *complete and sound set of pre-unifiers (CSU)* for a (Z)CUP P is a complete set of pre-unifiers S of P, such that for every pre-unifier $\theta \in S$, there is a unifier σ of P and a substitution δ such that $\sigma = \theta \circ \delta$.

▶ **Theorem 1.** *Given a (Z)CUP P, PUA computes a complete and sound set of pre-unifiers for P.*

Proof. Based on [7]. ◀

The name pre-unification algorithm is used because pre-solved problems are trivially unifiable in the unrestricted higher-order case. Since flat equations in context unification problems are not always unifiable (see Example 2), we get a weaker result than in [7]. Therefore, the terminating algorithm given later in this paper does not decide the context unification problem. Instead of referring to context unification problems as pre-solved, when they contain only flat or solved equations, we will call them *flat problems*. On the other hand, Z-context unification problems can be decided by the algorithm in this paper, which implies the decidability of the bounded second-order unification problem ([16]).

▶ **Example 2.** The following flat context unification problem is not unifiable: $\{Ya \doteq Yb\}$.

It is well known that there are two types of non-determinism in PUA. Choosing the next equation to apply a rule on is a "dont-care" non-determinism, which means that the completeness of the procedure is not affected by the choice of an equation. We will use this fact in the rest of the paper. Please refer to [20] (especially Lemma 4.20) for more information.

3 The Unification Algorithm

The algorithm PUA given in Sect. 2.2 does not terminate on (Z)CUPs. In this section we describe a sound and complete pre-unification algorithm for (Z)CUPs, which is based on PUA and in addition to it, has both bounds on the size of terms and rules for cycle elimination.

3.1 Contexts Pre-Unification Bounds

In order to show the termination and completion of the algorithm, we will place bounds on the depth of (Z-)contexts, which are mapped to context variables. These bounds will be dynamic in the sense that they are re-computed during the execution of the algorithm. The trigger for re-computation is the manipulation of context variables in the (Project), ($\mathtt{Rec}_0$) and ($\mathtt{Rec}^*$) rules.

▶ **Definition 3** (Standard cycles ([15])). A *cycle* c in a (Z)CUP P, is a sequence $\overline{t_n \doteq s_n}$ of equations of P such that for all $1 \leq i \leq n$, the head symbol of s_i is a context variable that occurs rigidly in $t_{i-1 \bmod n}$ and that there is $1 \leq j \leq n$, such that t_j is not a variable. A cycle is called a *standard cycle* if there is exactly one such j. The variables $\overline{X_n}$ are called *cyclic variables* of c and are also denoted by $X_i \in c$. The number $n = m - 1$, where m is the number of equations in the cycle, is called the *size of the cycle*. Another equivalent definition of the standard cycle size is the number of flat equations in it. Let $t_j = C[X_{j+1 \bmod n}]$ in a standard cycle s, then C is called the *cycle context* and is denoted by $\mathtt{cc}(c)$. The set of all standard cycles in a unification problem P is denoted by $\mathtt{scy}(P)$.

▶ **Example 4.** In $\{Xfa \doteq fYa, Yfa \doteq fZa, Zfa \doteq fXa\}$ the equations form a cycle. In the following we have the standard cycle of size 2 $\{Xfa \doteq Ya, Yfa \doteq Za, Zfa \doteq fffXfa\}$ with $fff[.]$ as cycle context.

▶ **Definition 5** (First-order bound)**.** Given a `(Z)CUP` P, its first-order bound is $k * l$, where k is the number of variables and l is the maximum rigid depth of a term in P. It is denoted by $b^1(P)$.

3.2 Regular Languages of Contexts and the Unification Algorithm

The algorithm `PUA` enumerates infinitely-many pre-unifiers, as can be seen in Example 6. The reason for that was shown to be the existence of cycles in the problems ([10],[15],[16]). The solution taken in the above papers is to search for specific (minimal) unifiers. This method, although sufficient for proving the decidability of the problems discussed in these papers, produces only a subset of the pre-unifiers of a problem and therefore cannot be used (eagerly) in unification-based automated deduction methods, like resolution. Our solution to the problem is to describe all possible pre-unifiers by using regular languages of contexts.

▶ **Example 6.** The set of all pre-unifiers of the `ZCUP` $\{Xfa \doteq fXa\}$ is $\{X \mapsto f^n[.] | n \geq 0\}$.

▶ **Definition 7** (Regular contexts)**.** Regular contexts extend the context grammar from Sec. 2.1 by using the kleene star:

$$C_r[.] ::= C[.] | C(C_r[.]) | C^*(C_r[.]) \tag{4}$$

As can be seen, for a given signature, the set C_S of all regular contexts over the signature contains also all finite contexts. We therefore define the set C_C of all contexts defined in Sec. 2.1 over the signature and the set C_R of strictly regular contexts is defined as $C_S \setminus C_C$.

▶ **Example 8.** The regular context $f(f([.])) \in C_C$ while the regular context $(f[.])^*([.]) \in C_R$.

▶ **Definition 9** (Regular context variables)**.** Regular context variables are denoted by brackets subscripted with a context variable as in $[]_X$. The set V_r of regular context variables is disjoint from V_1 and V_2. For a given `(Z)CUP` we hold also a function $\eta : V_r \to C_S$ from regular context variables to regular contexts. In order to simplify the presentation of the algorithm we will use a non-standard notation and describe this function within the regular context variables. I.e. $[D]_X$ will describe the regular context variable $[]_X$ such that $\eta([]_X) = D$. The notation $\{[D_1]_X \mapsto [D_2]_X\}$ means $\eta([]_X)$ is equal to D_2 (instead of to D_1), while $\{[D_1]_X \mapsto C\}$ means a normal substitution of the variable $[]_X$ with context C. We will also say that the regular context variable $[]_X$ is labeled by D if we have $[D]_X$.

▶ **Example 10.** The two regular context variables $[f^*(g^*([.]))]_X$ and $[g^*([.])]_X$ denote the same regular context variable $[]_X$. In the first case $\eta([]_X) = f^*(g^*([.]))$ and in the second $\eta([]_X) = g^*([.])$.

The intuition behind the regular context variables is that these variables, together with η, define a regular language of contexts. The rules of the algorithm in Fig. 2 will allow such variables to be mapped to contexts from these languages only.

The following function generates such regular languages of contexts given a standard cycle.

▶ **Definition 11** (The function `rec`)**.** Let c be a standard cycle with cycle context C and size m and let $X_i \in c$. Then, $\texttt{rec}(c,X_i) = [\texttt{rec}_0(m,C)]_{X_i}$ and

- if $m = 0$ then $\mathtt{rec}_0(m, C) = C^*C'$ where there is C'' such that $C'C'' = C$.
- if $m > 0$ then compute two contexts C' and C'' such that C'' starts with a non-unary function symbol f and $C'C'' = C$. Assume $C'' = f(t_1, ..D_k, .., t_n)$ and let I be a subset of $\{i | 0 < i \leq ar(f), i \neq k\}$. $\mathtt{rec}_0(m, C) = (C'C'')^*(C'f(v_1, ..\mathtt{rec}_0(m - 1, D_k(C'f(v_1, .., [.]_k, .., v_n)))_k, .., v_n))]$ where $v_i = x_i$ if $i \in I$ and $v_i = t_i$ otherwise. The x_i are new first order variables.

We denote by $C \in [D]_X$ the fact that a context C is a member of the regular language defined by D.

▶ **Example 12.** Given a signature consisting of constants a and b, unary function symbols e and h and binary functions symbols f and g:

- $\mathtt{rec}_0(2, ef(ha, hg([.], b)), X) = [(ef(ha, hg([.], b)))^*(ef(x_1, \mathtt{rec}_0(1, hg(ef(x_1, [.]), b))))]_X$ for $C' = e[.]$, $C'' = f(ha, hg([.], b))$ and $I = \{1\}$.
- $\mathtt{rec}_0(1, hg(ef(x_1, [.]), b)) = (hg(ef(x_1, [.]), b))^* hg(\mathtt{rec}_0(0, ef(x_1, hg([.], x_2))), x_2)$ for $C' = h[.]$, $C'' = g(ef(x_1, [.]), b)$ and $I = \{2\}$.
- $\mathtt{rec}_0(0, ef(x_1, hg([.], x_2))) = ef(x_1, hg([.], x_2))^* ef(x_1, h[.])$ for $C' = ef(x_1, h[.])$.

An application of a substitution to a regular context is performed in the same way as on normal contexts. The rules $\mathtt{(Bind}_X\mathtt{)}$, $\mathtt{(Imitate)}$, $\mathtt{(Project)}$ and $\mathtt{(Rec)}$ are only applicable to context variables while the rules $\mathtt{(Rec}_0\mathtt{)}$ and $\mathtt{(Rec}^*\mathtt{)}$ are only applicable to regular contexts.

In the algorithm given in Fig. 2, we hold for each context variable X an extra value $b(X)$, denoting a bound on the depth of contexts that can be mapped to it using the $\mathtt{(Imitate)}$ rules. The two differences between $\mathtt{PUA}$ and $\mathtt{CUA}$ are that $\mathtt{CUA}$ has a bound on the number of $\mathtt{(Imitate)}$ rule applications and that standard cycles in the problem are being eliminated by the use of the $\mathtt{(Rec)}$ rule.

The rules $\mathtt{(Rec}_0^1\mathtt{)}$ and $\mathtt{(Rec}_0^2\mathtt{)}$ will be also referred to as the $\mathtt{(Rec}_0\mathtt{)}$ rule.

4 Soundness and Completeness

In this section we will prove the soundness and completeness of $\mathtt{CUA}$ with regard to $\mathtt{PUA}$. Because $\mathtt{PUA}$ is sound and complete with regard to all possible pre-unifiers, this will show that the algorithm indeed enumerates all pre-unifiers.

We will prove that a sequences of $\mathtt{(Imitate)}$ rule applications without an application of the $\mathtt{(Project)}$ or $\mathtt{(Rec)}$ rules on a $\mathtt{(Z)CUP}$ P can be restricted to $b^1(P)$ steps because otherwise, there must be a standard cycle. In that case, we prove that the cycle can be eliminated by an application of the rule $\mathtt{(Rec)}$ while preserving completeness.

▶ **Lemma 13** (Soundness). *Every substitution obtained from a pre-solved form of a $\mathtt{CUA}$ run on a $\mathtt{(Z)CUP}$ P is a pre-unifier of P.*

Proof. By induction on the length of a derivation in $\mathtt{CUA}$. The only case not proved in Theorem 1 is $\mathtt{(Rec)}$ followed by n $\mathtt{(Rec}^*\mathtt{)}$s and a $\mathtt{(Rec}_0\mathtt{)}$. Since only the $\mathtt{(Rec}^*\mathtt{)}$ and $\mathtt{(Rec}_0\mathtt{)}$ rules can be applied to a regular context variable and since we can choose the equations to use in an arbitrary way, we can treat them together. Such a sequence describes an applied substitution δ, replacing a variable with a context. Assume by induction hypothesis that the obtained system is unifiable by σ, then the original system is unifiable by $\delta \circ \sigma$. ◀

The following lemmas state that if a context variable is mapped, in some pre-unifier, to a large context, then this variable, or a smaller one, is related to a standard cycle.

$$\frac{\Gamma \cup \{A \doteq A\}}{\Gamma}\ \texttt{(Delete)} \qquad \frac{\Gamma \cup \{f(\overline{s_n}) \doteq f(\overline{t_n})\}}{\Gamma \cup \{s_1 \doteq t_1, .., s_n \doteq t_n\}}\ \texttt{(Decomp)}$$

$$\frac{\Gamma \cup \{x \doteq t\} \qquad x \notin Var(t)}{\Gamma\{x \mapsto t\}}\ \texttt{(Bind}_x\texttt{)} \qquad \frac{\Gamma \cup \{X([.]) \doteq t\} \qquad X \notin Var(t)}{\Gamma\{X \mapsto t\}}\ \texttt{(Bind}_X\texttt{)}$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(P), u \in \texttt{PB}(t), b(X) > 0}{\Gamma \cup \{X([.]) \doteq u\}}\ \texttt{(Imitate)}^3$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(P)}{\Gamma \cup \{X([.]) \doteq [.]\}}\ \texttt{(Project)}^4 \qquad \frac{\Gamma \qquad c \in \texttt{scy}(\Gamma), X \in c}{\Gamma \cup \{X[.] \doteq \texttt{rec}(c, X)\}}\ \texttt{(Rec)}$$

$$\frac{\Gamma \qquad [C^*(C_2[.])]_X \doteq t \in \Gamma, C_2 \in C_C}{\Gamma\{[C^*(C_2[.])]_X \mapsto C_2[.]\}}\ \texttt{(Rec}_0^1\texttt{)}^4 \qquad \frac{\Gamma \qquad [C^*(C_2[.])]_X \doteq t \in \Gamma, C_2 \in C_R}{\Gamma\{[C^*(C_2[.])]_X \mapsto [C_2[.]]_X\}}\ \texttt{(Rec}_0^2\texttt{)}^4$$

$$\frac{\Gamma \qquad [C^*(C_2[.])]_X \doteq t \in \Gamma}{\Gamma\{[C^*(C_2[.])]_X \mapsto C([C^*(C_2[.])]_X\}}\ \texttt{(Rec}^*\texttt{)}^4$$

1. for all variables X, we initialize $b(X) = b^1(P)$.
2. each call of `(Imitate)`, `(Project)` and `(Rec)` is followed by $\texttt{(Bind}_X\texttt{)}$.
3. if a fresh variable X' is introduced in u, then $b(X') = b(X) - 1$.
4. for all variables $X \in Var_2(P)$, $b(X) = b^1(P)$.

Figure 2 `CUA` - Unification Rules[1,2]

▶ **Definition 14** (Ordering of variables)**.** Let P be a `(Z)CUP`. The relation $<_c^0$ is an ordering on the variables in P if for all equations $t \doteq s \in P$ such that $t|_{p_1} = w_1$ and $s|_{p_2} = w_2$ and $p_1 \gg_n p_2$ with $n > 0$, then $w_2 <_c^0 w_1$. $<_c$ is any arbitrary extension of $<_c^0$ into a total order, such that if $<_c^0$ is acyclic, so is $<_c$.

▶ **Example 15.** Given the `CUP` $\{f(Xa, g(x, y)) \doteq f(h(z), Xb)\}$, $<_c^0 = \{(z, X), (x, X), (y, X)\}$.

▶ **Definition 16** (Repeated variables)**.** Let P be a `(Z)CUP` and P' the system obtained from P using `PUA` via any sequence φ of transformations not including `(Project)`. let σ be the substitution obtained from P'. A variable X in P is called a repeated variable in φ if $d(\sigma(X)) > b^1(P)$.

▶ **Lemma 17.** *Let X be a context variable in a `(Z)CUP` P such that all variables in P are acyclic according to $<_c$. Then, for any substitution σ obtainable from P by using `PUA` without the application of a `(Project)`, we have $d(\sigma(X)) \leq b^1(P)$.*

Proof. By induction on the number m of variables, ordered by $<_c$ and let $md(P)$ be the maximum (rigid) depth of a term in P. If $m = 1$, we can apply `(Decomp)` exhaustively until w occurs rigidly only in head positions and clearly, for any substitution σ, $d(\sigma(w)) \leq md(P) = b^1(P)$. Otherwise, let w be a maximal variable according to $\leq_c$. Then, there is no other variable w' in P, such that there is an equation $t \doteq s \in P$, $t|_{p_1} = w'$, $s|_{p_2} = w$ with $p_1 \gg_n p_2$ and $n > 0$. Let P_0 be the problem after the removal of all equations containing w in rigid positions and let σ_0 be a substitution obtained from P_0 using `PUA`. By induction hypothesis, for all variables w' in P other than w, $d(\sigma_0(w')) \leq b^1(P_0)$. As $md(P) \geq md(P_0)$, we get that for all variables w' other than w, $d(\sigma_0(w')) \leq b^1(P)$. As we can choose the order of equations in `PUA` freely, then also $d(\sigma(w')) \leq b^1(P)$. Now, let $t \doteq s \in P$ be an equation containing w

and $w \doteq s'$ the equation after applying several `(Decomp)`s. Let V_0 be the set of all variables in s', then, as s' is not a variable, $d(\sigma(w)) \leq d(s') + max_{w' \in V_0}(d(\sigma(w')))$. As $d(s') \leq md(P)$ and $d(\sigma(w')) \leq md(P)*(m-1)$, we get that $d(\sigma(w)) \leq md(P)+md(P)*(m-1) = b^1(P)$. ◀

▶ **Definition 18** (Sub-equations)**.** Given a rigid-rigid equation $e = t \doteq s$, the set $sub(e)$ of sub-equations of e is $\{t' \doteq s' \mid t|_p = t', s|_p = s', p \in Pos(t) \cap Pos(s)\}$.

▶ **Definition 19** (Problem Restriction)**.** Let P be a `(Z)CUP`, a restriction of P with regard to a variable set $V^0 \subseteq Var(P)$ is the set of all sub-equations of equations in P such that a variable from V^0 is one of the terms in the sub-equation.

▶ **Example 20.** $sub(e)$ for $e = f(XgXa, gb) \doteq f(ga, gz)$ is $\{f(XgXa, gb) \doteq f(ga, gz), XgXa \doteq ga, gb \doteq gz, b \doteq z\}$. When restricted to $V^0 = \{X\}$, we get the problem restriction $\{XgXa \doteq ga\}$.

▶ **Lemma 21.** *If σ unifies a `(Z)CUP` P, then it unifies a problem restriction P' of P.*

Proof. P' is just a subset of the equations generated from P after the application of `(Decomp)` transformations. ◀

▶ **Lemma 22.** *If a variable X in a `(Z)CUP` P is repeated, then it is either cyclic or there is a smaller variable in P, according to $<_c$, which is cyclic.*

Proof. Assume none of them is cyclic, then the order $<_c$ is well-founded over the set V^0 of these variables and we consider the problem restriction P' of P with regard to V^0. We know that there is a derivation φ in `PUA` such that we obtain σ with $d(\sigma(X)) > b^1(P) \geq b^1(P')$. We would like to get a contradiction to the existence of such an unifier which will imply that no such (extension of a) unifier also exists for P using Lemma 21. We choose φ such that it does not contain a `(Project)` call. We can do so as `(Project)` reset repeatability (according to Def. 16) so we can choose φ starting after the last `(Project)` before obtaining σ. Now, as all the variables in P' are acyclic, we can use Lemma 17 to get a contradiction. ◀

▶ **Lemma 23.** *Given a cycle $\overline{e_m = X_m \doteq t_m}$ in a `(Z)CUP` P, we can derive a standard cycle using the `CUA` rules `(Delete)`, `(Decomp)`, `(Bind)` and `(Imitate)`.*

Proof. We prove by induction on $n = \Sigma_{i=1}^{m-1} i * m_i$ where m_i is the size of the position of X_{i+1} in t_i in each equation. If $n = 0$, then we have a standard cycle as e_m cannot be a flat equation (see next). If $n > 0$, then we apply `(Imitate)` on an equation e_j with $0 < j < m$. The result, after applying `(Decomp)`, is again a cycle with a new variable X'_j instead of X_j. n is decreased in the new cycle as either $j > 1$ and then we get m_j is decreased by j and m_{j-1} is increased by $j-1$, or $j = 1$ and then m_1 is decreased by 1. In the second case, m_m is increased but we don't count it. Since we must get at some point $j = 1$, we also get that at the end of the derivation e_m is not flat. ◀

In the following lemma we show that in any pre-unifier of a problem containing a standard cycle, one of the variables of the cycle must be mapped to a context described by the regular context variable generated by the rule `(Rec)` for the variables of this cycle. The proof is using an essential step from the proof of Lemma 5.8 in [15]. More information about this essential step can be found in [15], [16] and [9]. Despite the similarities of the proofs, our proof differs in several important points. First, the proof is about the completeness of using regular languages and not about searching for minimal unifiers. Second, our algorithm is using a minimal set of rules, in contrast to the specialized many rules in the other papers. A consequence of this is that we indirectly treat cases, which are treated directly in the other papers, such as the treatment of ambiguous cycles.

▶ **Lemma 24.** *Let P be a (Z)CUP with a standard cycle c (of length m and context C), then for any pre-unifier σ of P, there is a variable X in the cycle such that $\sigma(X) \in \mathtt{rec}(\sigma(c), X)$.*

Proof. Let $\overline{X_n}$ $(n = m+1)$ be the context variables in the cycle and define the context A to be the greatest common prefix of $\sigma(X_i)$ and $(\sigma(C))^*$ for $0 < i \leq n$ where $*$ is an arbitrary natural number. Then, $A = \sigma(C)^l(C')$ for some l where C' is a proper prefix of $\sigma(C)$ and let C'' be a context such that $C'(C'') = \sigma(C)$. The rules executed in the proof are from the complete PUA.

- for $n = 0$ we need to show that $\sigma(X_1) = A$. Assume that $\sigma(X) \neq A$, then $\sigma(X) = A(f(t_1, .., [.]_k, .., t_n))(D)$ for some context D, where $A(f(t_1, .., [.]_k, .., t_n))$ is not a prefix of $(\sigma(C))^*$. We get:
$$A(f(t_1, .., t'_k, .., t_n)) = \sigma(C)(A(f(t_1, .., t''_k, .., t_n))),$$
where t'_k, t''_k are terms. Applying decompositions we get:
$$f(t_1, .., t'_k, .., t_n) \doteq C''(C'(f(t_1, .., t''_k, .., t_n)))$$
Now we consider the first letter of the position of [.] in C''. If it is k, then we get a contradiction to the maximality of A as it should include f as well. Otherwise, it is $i \neq k$ and let $C''[.] = f(s_1, .., D'[.]_i, .., s_n)$, then we get from the equation, after one decomposition, that
$$t_i \doteq D'(C'(f(t_1, .., t_i, .., t_n)))$$
which is a contradiction to the unifiability of the pair as the positions of the holes in D' and C' are rigid (according to the definition of cycles).
- for $n > 0$, if there is $0 < i \leq n$ such that $\sigma(X_i) = A$ then we are done. Assume otherwise, that A is a proper prefix of $\sigma(X_i)$ for all $0 < i \leq n$. Let f be the first function symbol in C'' (of arity ar) and assume the first letter of the position of the hole in C'' is k. Then, f must be a non-unary function symbol as there must be at least one variable X_i $(1 \leq i \leq n)$, such that $\sigma(X_i) = Af(v^i_1, .., D^i_l[.]), .., v^i_{ar})$ where $k \neq l$. Otherwise, A is not a maximal common prefix. Now, let I contain all the indices of the variables X_i such that $\sigma(X_{I_j}) = Af(v^{I_j}_1, .., D^{I_j}_k[.]), .., v^{I_j}_{ar})$ (i.e. the character of the position of the hole in f is k). Then, after applying (Decomp)s, (Delete)s, (Imitate)s and (Bind)s, we get (among other equations) the standard cycle for the position k:
$$X'_{I_1}t'_1 \doteq X'_{I_2}s'_1, \; .. \; , X'_{I_{size(I)}}t'_{size(I)} \doteq D'C'(f(x'_1, ..X'_{I_1}s'_{size(I)}, ..x'_{ar}))$$
for $C''[.] = f(u_1, .., D'[.]_i, .., u_{ar})$ where $\overline{t'_{size(I)}}$ and $\overline{s'_{size(I)}}$ are permutations of subsets of $\overline{t_n}$ and $\overline{s_n}$. Because the cycle is of size $l = size(I) - 1 < m$, we can apply the induction hypothesis in order to obtain that there is j such that $\sigma(X'_{I_j}) \in [\mathtt{rec}_0(l, D'C'(f(\sigma(x'_1), ..[.]_k, ..\sigma(x'_{ar}))))]_{X'_{I_j}}$ and therefore,
$\sigma(X_{I_j}) = Af(u_1, .., \sigma(X'_{I_j}), .., u_{ar}) \in [\mathtt{rec}_0(l+1, \sigma(C))]_{X_{I_j}}$ and as $l+1 \leq m$, we get that $\sigma(X_{I_j}) \in [\mathtt{rec}(l+1, \sigma(C))] \subseteq [\mathtt{rec}(m, \sigma(C))]_{X_{I_j}}$ as well. Note, that I cannot be empty as we would get in this case (after applying the rules) the pair $x^i \doteq D'C'(f(v_1, ..x^i, ..v_{ar}))$ which cannot be unified as the positions of the holes in D' and C' are rigid (according to the definition of cycles).

◀

In the next example we demonstrate the transformations taking place in the proof:

▶ **Example 25.** The following example extends Ex. 12. Assume we have the following standard cycle containing the context variables X_1 ,X_2 and X_3 and arbitrary terms t_1,t_2,t_3,s_1,s_2 and s_3: $\{X_1t_1 \doteq X_2s_1, X_2t_2 \doteq X_3s_2, X_3t_3 \doteq ef(ha, hg(X_1s_3, b))\}$. Let A_1 be $(ef(ha, hg([.], b)))^{l_1}(e[.])$ for some l_1. f is the function symbol and $k = 2$. Assume $\sigma(X_1) =$

$A_1(f(\sigma(x_1), \sigma(X_1')[.]))$, $\sigma(X_2) = A_1(f(\sigma(X_2')[.], \sigma(x_2))$ and $\sigma(X_3) = A_1(f(\sigma(x_3), \sigma(X_3')[.]))$. In this case $I = \{1, 3\}$. Applying rules from PUA, we can get the standard cycle $\{X_1't_1 \doteq X_3's_2, X_3't_3 \doteq hg(ef(x_1, X_1's_3), b)\}$. Assume now that $A_2 = (hg(ef(x_1, [.]), b))^{l_2}(h[.])$ for some l_2. Now g is the function symbol and $k = 1$. Assume that $\sigma(X_1') = A_2g(\sigma(X_1'')[.], \sigma(x_1'))$ and $\sigma(X_3') = A_2g(\sigma(x_3'), \sigma(X_3'')[.])$. Therefore, $I = \{1\}$. Applying again rules from PUA, we can get the standard cycle $X_1''t_1 \doteq ef(x_1, hg(X_1''s_3, x_1'))$ and we can use the induction base to have $\sigma(X_1'') \in \texttt{rec}(0, ef(x_1, hg([.], x_1')), X_1'')$.
Because $\sigma(X_1') = (hg(ef(x_1, [.]), b))^{l_3}(hg(\sigma(X_1'')[.], \sigma(x_1')))$ for some l_3, we get that $\sigma(X_1') \in \texttt{rec}(1, hg(ef(x_1, [.]), b), X_1')$ and obtain that $\sigma(X_1) \in \texttt{rec}(2, ef(ha, hg([.], b)), X_1)$ as $\sigma(X_1) = (ef(ha, hg([.], b)))^{l_4}(ef(\sigma(x_1), \sigma(X_1')[.]))$ for some l_4.

▶ **Lemma 26.** *The algorithm CUA is complete with respect to PUA.*

Proof. By induction on the number of context variables using the following induction hypothesis: for a given (Z)CUP P, if it is possible to obtain a pre-unifier of P using PUA, then it is possible to obtain the same pre-unifier using CUA. Induction base: CUA runs the same as the complete PUA on purely first order problems and in case of a flat problem we have nothing to do. Induction step: The only problematic step is an application of (Imitate) on a variable X in some system P'' with $b(X) = 0$ in PUA. Using Lemma 22 we know that there is some cyclic variable in a system P' preceding P''. Using Lemma 23 we can obtain a standard cycle and by using the (Rec) rule, we can replace a context variable in the standard cycle by a regular context variable. According to Lemma 24, for each pre-unifier of the problem there is a variable in the standard cycle that is mapped to a context described by the regular context labeling the variable. We can now fold the regular context variable by calling the ($\texttt{Rec}_0$) and the ($\texttt{Rec}^*$) rules and obtain the required context. The problem we obtain has one less context variable and we can apply the induction hypothesis. ◀

▶ **Theorem 27.** *Given a (Z)CUP P, CUA computes a complete and sound set of pre-unifiers for P.*

Proof. Following lemmas 13 and 26. ◀

5 Termination and Minimal Unifiers

In this section we will show that in practice, the number of recursive calls to ($\texttt{Rec}^*$) can be bounded due to the following result([17]):

▶ **Definition 28** (Minimal unifiers)**.** Given a (Z)CUP P, a unifier σ of P is called minimal if there is no other unifier σ' of P with $\Sigma_{X \in Var(P)} size(\sigma'(X)) < \Sigma_{X \in Var(P)} size(\sigma(X))$.

▶ **Definition 29** (Exponent of periodicity)**.** A ground unifier σ has an exponent of periodicity n iff n is the maximal number such that there is some context variable X and ground contexts A, B and C such that $\sigma(X) = AB^nC$.

▶ **Lemma 30** ([17],[16])**.** *There are constants c and d, such that for every unifiable (Z)CUP P and for every minimal unifier σ of P, its exponent of periodicity is less than $c * 2^{2.14 * d * size(P)}$.*

5.1 The Restricted CUA

The exponent computed in the above lemma allows us to bound the number of applications of the rule ($\texttt{Rec}^*$) on a specific regular variable.

▶ **Definition 31** (Restricted CUA (RCUA))**.** Let $e : V_r \times \mathbb{N} \to \mathbb{N}$ be a function from regular context variables and the identifiers of the kleene stars in them to natural numbers. Then, the restricted CUA is obtained by restricting the (Rec*) rule such that each application of the rule on a regular context variable D where the identifier of the first kleene star is i decreases $e(D, i)$ by one and the rule is applicable only if $e(D, i) > 0$. The values $e(D, i)$ for all regular context variables and the identifiers of the kleene stars in them are initialized to the exponent of periodicity of P.

▶ **Definition 32** (Minimal CSU)**.** the minimal CSU of a (Z)CUP P, is a CSU of P with regard to its minimal unifiers only.

▶ **Lemma 33.** *Given a (Z)CUP P, RCUA computes its minimal CSU.*

Proof. Based on Theorem 27 and Lemma 30. ◀

▶ **Theorem 34.** *Given a (Z)CUP P, RCUA terminates on P.*

Proof. The algorithm is finitely branching. We will show termination of a specific run by taking the lexicographic ordering of the following measure $\mu =< m_1, m_2, m_3, m_4, m_5 >$ where m_1 is the number of context variables, m_2 is the sum of all values in the range of e, m_3 is the sum of all the $b(X)$ values of the context variables, m_4 is the number of first order variables and m_5 is the number of symbols other than $\doteq$ in the problem. First, it is easy to see that:

- (Delete) and (Decomp) reduces m_5 without increasing $m_1 - m_4$.
- (Bind$_x$) decreases m_4 and does not increase $m_1 - m_3$.
- (Imitate) decreases m_3.
- (Rec$_0$) and (Rec*) decrease m_2.
- (Project), (Rec) and (Bind$_X$) decrease m_1.

Next, we claim that:

- if m_3 is increased, then either m_1 or m_2 are decreased. This is because m_3 can increase either after the application of a (Project), (Rec$_0$) or (Rec*) rule.
- if m_2 is increased, then m_1 must be decreased as we increase the sum only when (Rec) is called.
- m_1 can never be increased on an RCUA run.

Put together, the measure μ decreases after each application of RCUA. ◀

▶ **Corollary 35.** *The Z-context unification problem is decidable.*

Proof. Following Lemma 33 and Theorem 34. ◀

▶ **Definition 36** (Bounded second-order unification problems ([16]))**.** Let P be a second-order unification problem, V_2 the set of its second order variables and $b : V_2 \to \mathbb{N}$. The pair (P, b) is called a bounded second-order unification problem. A substitution σ is a unifier of (P, b) iff σ is a unifier of P and for every $X \in V_2$, the term $\sigma(X)$ contains at most $b(X)$ bound variables.

We now prove the following result from [16].

▶ **Corollary 37.** *The bounded second-order unification problem is decidable.*

Proof. Using the transformation from Sec. 3 in [16], we can transform any bounded second-order unification problem into a ZCUP, which are decidable following Cor. 35. ◀

5.2 Deciding the Stratified Context Unification Problem

The stratified context unification problem was shown to be decidable in [15]. This fragment also plays a role in computational linguistics [3]. In this section we give another algorithm to decide the problem, which is based on RCUA. All the definitions in this section and the treatment of flat equations are taken from [15]. The stratified context unification algorithms given in [15] is a very specialized algorithm, which contains many rules for the different cases possible and therefore has a relatively complex correctness proof. Our version, which follows the very general pre-unification algorithm [7], benefits from its correctness proof and this allows us to give, we hope, a simpler proof for the decidability of this problem.

▶ **Definition 38** (Second-order prefixes (SO-prefixes)). SO-prefixes are words over $(V_2)^*$. Let P be a CUP, the SO-prefix of a term t is defined inductively:

- if t occurs in an equation $t \doteq s$, then t has an empty SO-prefix.
- if $t = f(\overline{s_n})$ has an SO-prefix w, then s_i, for $0 < i \leq n$, has SO-prefix w.
- if $t = X(s)$ has an SO-prefix w, then s has SO-prefix $w \cdot X$.

▶ **Definition 39** (Stratified CUPs (SCUPs)). A CUP P is called stratified if for every context and first-order variable in P, all its occurrences have the same SO-prefix. A flat SCUP is a SCUP in pre-solved form.

▶ **Example 40.** $\{Xa \doteq XYa, fXa \doteq Za\}$ is stratified while $\{Xa \doteq XYa, fXa \doteq Ya\}$ is not.

▶ **Definition 41** (Clusters). Let P be a flat SCUP, a cluster c of P is a minimum subset of equations of P such that if an equation $Xt \doteq Ys$ is in c and there is an equation $e = Xv \doteq u$ in P, then e is also in c.

▶ **Example 42.** the clusters of $\{X_1a \doteq X_2b, X_1c \doteq X_3d, Y_1e \doteq Y_2f\}$ are $\{X_1a \doteq X_2b, X_1c \doteq X_3d\}$ and $\{Y_1e \doteq Y_2f\}$.

▶ **Definition 43** (Clusters instantiations). Given a flat SCUP P and a cluster c in P with variables $\overline{X_n}$, inst(P, c) is the application of either:

- a (Project) on some equation in c or
- if $n > 1$ and there is a function symbol f in the signature with $m = ar(f) > 1$ then:
  - choose an index $0 < k_i \leq m$ for each variable X_i such that at least two indices are different.
  - replace each X_i with $f(x_{(i,1)}, .., X_{(i,k_i)}, .., x_{(i,m)})$ containing new variables.
  - apply (Decomp) in order to eliminate the f symbol.
  - apply (Bind$_x$)s in order to solve the new equations having first order variables as at least one head.

▶ **Lemma 44.** *Given a SCUP P with a minimal (with regard to the number of equations in it) cluster c, inst(P,c) results in a system with either less context variables or a smaller minimal cluster.*

Proof. If we apply a (Project) on a variable in the cycle, the number of context variables decreases. Therefore, we assume we choose the second option. After the application of the (Decomp) and (Bind$_x$) rules and as at least one variable was mapped to a term with a different index for the first character on the position to the hole, the resulted system will have a cluster smaller by at least one equation. Since we took a minimal cluster, the size of the minimal cluster in the resulted system will be smaller. In addition, since the head context variables in the clusters are all disjoint, the process does not affect other clusters. ◀

▶ **Example 45.** The cluster $\{Xt_1 \doteq Ys_1, Yt_2 \doteq Zs_2\}$ can be reduced using the substitution $X \mapsto f(X'[.], x)$, $Y \mapsto f(Y'[.], y)$ and $Z \mapsto f(z, Z'[.])$ and the applications of `(Decomp)` and `(Bind)` to the cluster $\{X't_1 \doteq Y's_1\}$.

▶ **Lemma 46.** *(Lemmas 6.6 and 6.7 in [15]) Given a flat SCUP P and a minimal cluster c in P, then applying inst(P, c) is sound and complete.*

▶ **Definition 47** (SCUA - An algorithm for stratified context unification)**.** Given a SCUP P, we apply one of the following transformations:

- if P is not flat, we apply `RCUA`.
- if P is flat, we apply `inst`(P, c) on a minimal cluster c in P.

▶ **Lemma 48.** *SCUA preserves the stratifiability of a problem.*

Proof. The transformations of `RCUA` clearly preserve stratifiability as all manipulations on SO-prefixes are done using substitutions, which apply to the whole problem. For `inst`, the proof is exactly as in Lemma 6.5 in [15]. ◀

▶ **Lemma 49.** *SCUA is sound and complete with regard to minimal unifiers.*

Proof. Using lemmas 33 and 46. ◀

▶ **Lemma 50.** *SCUA terminates on SCUPs.*

Proof. Using the lexicographic ordering on the measure $< m_1, m_2, m_3 >$ where m_1 is the number of context variables, m_2 is 1 if the problem is not pre-solved and 0 otherwise and m_3 is the size of a minimal cluster. First, it is clear that m_1 is never increased. If the problem is not flat, then we apply `RCUA` and obtain (following Theorem 34) either a pre-solved form or an error, resulting in a decrease in m_2. If the problem is flat, then we apply `inst` resulting, according to Lemma 44, either with less context variables and the reduction of m_1 or with a smaller minimal cluster and the reduction of m_3. It is easy to see that the application of `inst` results in a pre-solved form so m_2 does not increase. ◀

We can now prove the following result from [15].

▶ **Corollary 51.** *The stratified context unification problem is decidable.*

Proof. Using Lemmas 48, 49 and 50. ◀

References

1 Habib Abdulrab, Pavel Goralcik, and G. S. Makanin. Towards parametrizing word equations. *ITA*, 35(4):331–350, 2001.

2 Hubert Comon. Completion of rewrite systems with membership constraints. part i: Deduction rules. *J. Symb. Comput.*, 25(4):397–419, 1998.

3 Katrin Erk and Joachim Niehren. Dominance constraints in stratified context unification. *Inf. Process. Lett.*, 101(4):141–147, 2007.

4 William M. Farmer. Simple second-order languages for which unification is undecidable. *Theor. Comput. Sci.*, 87(1):25–41, 1991.

5 J.I. Hmelevskii. *Equations in Free Semigroups.* Proceedings of the Steklov Institute of Mathematics. American Mathematical Society, 1976.

6 Gérard P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972.

7 Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.

8 Jordi Levy. Linear second-order unification. In *RTA*, pages 332–346, 1996.

9 Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.

10 G. S. Makanin. On the decidability of the theory of free groups (in russian). In *FCT*, pages 279–284, 1985.

11 Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *CADE*, pages 34–48, 1997.

12 Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *ACL*, pages 410–417, 1997.

13 Joachim Niehren, Sophie Tison, and Ralf Treinen. On rewrite constraints and context unification. *Inf. Process. Lett.*, 74(1-2):35–40, 2000.

14 John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

15 Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Log. Comput.*, 12(6):929–953, 2002.

16 Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Inf. Comput.*, 188:143–178, January 2004.

17 Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, pages 61–75, 1998.

18 Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.

19 Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. *J. Symb. Comput.*, 40(2):905–954, August 2005.

20 Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symb. Comput.*, 8(1/2):101–140, 1989.