Context unification with one context variable

Journal of Symbolic Computation 45 (2010) 173–193

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Context unification with one context variable

Adrià Gascón [a], Guillem Godoy [a,1], Manfred Schmidt-Schauß [b], Ashish Tiwari [c]

[a] LSI Department, Technical University of Catalonia, Jordi Girona, 1-3 08034 Barcelona, Spain
[b] Institut für Informatik, Johann Wolfgang Goethe-Universität, Postfach 11 19 32, D-60054 Frankfurt, Germany
[c] SRI International, 333 Ravenswood Ave, Menlo Park, CA, USA

ARTICLE INFO

Article history:
Received 15 May 2008
Accepted 13 October 2008
Available online 21 June 2009

Keywords:
Unification
Interprocedural program analysis
Context unification
Redundancy

ABSTRACT

The context unification problem is a generalization of standard term unification. It consists of finding a unifier for a set of term equations containing first-order variables and context variables. In this paper we analyze the special case of context unification where the use of at most one context variable is allowed and show that it is in NP. The motivation for investigating this subcase of context unification is interprocedural program analysis for programs described using arbitrary terms, generalizing the case where terms were restricted to using unary function symbols. Our results imply that the redundancy problem is in coNP, and that the finite redundancy property holds in this case. We also exhibit particular cases where one context unification is polynomial.

1. Introduction

There is an interesting link between unification and interprocedural program analysis. The goal of interprocedural program analysis is to compute all simple invariants, typically of a specific form, of imperative procedural programs. The process of computing a class of such invariants requires solving a special case of context unification: one context unification. However, the application requires more than just solving unification problems. It requires determining whether an equation is redundant in

E-mail addresses: adriagascon@gmail.com (A. Gascón), ggodoy@lsi.upc.es (G. Godoy), schauss@cs.uni-frankfurt.de (M. Schmidt-Schauß), tiwari@csl.sri.com (A. Tiwari).
[1] Tel.: +34 93 413 7815; fax: +34 93 413 7833.

0747-7171/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2008.10.005

a set of equations w.r.t. unification and whether nonredundant sequences of equations are finite. We answer all these questions in this paper.

1.1. Unification and context unification

In mathematical logic, in particular as applied to computer science, a unifier of two terms s and t is a substitution σ for variables occurring in s and t such that $\sigma(s) = \sigma(t)$. The classical first-order term unification problem seeks to find solutions for term equations built over uninterpreted function symbols and *first-order* variables. The input is a set of equations, $S := \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$, over terms s_i, t_i containing some *first-order* variables. The term unification problem asks for the existence of a substitution σ that maps the variables to *first-order* terms such that for all i, $\sigma(s_i)$ and $\sigma(t_i)$ are syntactically equal. For example, the set of equations $\{f(f(x_2, x_2), f(x_3, x_3)) \doteq f(x_1, x_2)\}$ has a solution (called unifier) $\langle x_1 \mapsto f(f(x_3, x_3), f(x_3, x_3)), x_2 \mapsto f(x_3, x_3)\rangle$. First-order unification arises in many areas, such as, automated deduction, type checking, deductive databases, logic programming, artificial intelligence, information retrieval, and compiler design.

The term unification problem is known to be solvable in linear time (Paterson and Wegman, 1978; Martelli and Montanari, 1982; Baader and Snyder, 2001). However, its expressiveness is often insufficient and hence, several variants and extensions have been considered in the literature. For example, in unification modulo theories (Baader and Snyder, 2001) some function symbols are interpreted and $=$ is interpreted as *semantic* equality.

A generalization of the first-order unification problem is the context unification problem, where apart from first-order variables, there are also *context variables* in the terms. Context variables can be substituted by contexts, which are terms with a single hole. For example, consider the term equation $F(f(x, b)) \doteq f(a, F(y))$, where x, y are first-order variables and F is a context variable that can be substituted by any context. This concrete equation has several solutions (i.e. unifiers), such as, $\{F \mapsto f(a, \bullet), x \mapsto a, y \mapsto b\}$ and $\{F \mapsto f(a, f(a, \bullet)), x \mapsto a, y \mapsto b\}$, where $\bullet$ denotes the hole of the context.

Context unification is a special case of second-order unification. Second-order unification is undecidable, while the decidability of context unification is not known, although several decidable subclasses have been identified (Levy, 1996; Schmidt-Schauß, 2002; Schmidt-Schauß and Schulz, 2002; Levy et al., 2005). Context unification has application in linguistics (Niehren et al., 1997a,b).

Schmidt-Schauß and Schulz (2002) showed decidability of the case where at most two distinct context variables appear in the equations, but their algorithm is rather involved and they showed no complexity bounds. Levy et al. (2006) recently showed that stratified context unification is NP-complete. The main technical approach used to show membership of stratified context unification in NP is based on the use of singleton tree grammars to succinctly represent large terms (Levy et al., 2006), see also Busatto et al. (2008).

In contrast, in this paper we study the case where only *one context variable* appears in the input set, and obtain an NP algorithm for this *one context unification* problem, which can be implemented with moderate effort. The one context unification problem is not contained in the class of stratified context unification problems.

Our algorithm for one context unification performs unification on special forms of terms with iterations using integer exponents. More generally, Comon (1995) proved decidability of unification of terms with iterations using integer exponents, but did not obtain complexity bounds. Schmidt-Schauß and Schulz (1998) showed that a size-minimal unifier of a context unification problem implies some bounds on repeated subparts: specifically, if there is an iterated occurrence of $\underbrace{C \ldots C}_{n}[\bullet]$ of a context C in the unifier, then the number n has at most a linear number of digits (Schmidt-Schauß and Schulz, 1998). In this paper, we achieve concise representations by using a simple exponent-based notation for terms. We also show that the maximally required exponent is linear in the size of the problem, which is far less than the bound for exponents of minimal solutions, which is exponential (Schmidt-Schauß and Schulz, 1998). We also remark that unifiability of word equations with one variable is a special case of context unification with one context variable. The decidability problem for word

```
   P() {
1    x:=f(gx, z); y:=gf(y, z);
2    while(*) {
3      x:=f(x, z); y:=f(y, z);
4    }
5  }
       Summary Computation for P
5,4:   x = F[y]
3-:    f(x, z) = F[f(y, z)]
3-:    f(x, z) = F[f(y, z)], f(f(x, z), z) = F[f(f(y, z), z)], ...
2-:    x = F[y], f(x, z) = F[f(y, z)], f(f(x, z), z) = F[f(f(y, z), z)], ...
1-:    f(gx, z) = F[gf(y, z)], f(f(gx, z), z) = F[f(gf(y, z), z)], ...
```

Fig. 1. Computing summaries of programs using context unification. The procedure computes the set of facts that should hold at each line for $x = F[y]$, where $F[\bullet]$ is unknown, to be an invariant at Line 5. The while loop gives rise to an unbounded number of facts at Line 3. The negative sign − indicates that the facts hold *before* the program in that line is executed. Only the "nonredundant" equations are shown; the . . . symbol represents the "redundant" equations that can be eliminated.

equations is decidable (Makanin, 1977), and the special case with one variable was shown to be solvable in $O(n\log(n))$ time (Dabrowski and Plandowski, 2002).

1.2. Motivation: Interprocedural program analysis

The one context unification problem is motivated by our attempt to solve the (open) problem of interprocedural assertion checking (global value numbering). The assertion checking problem asks if two given expressions evaluate to the same value on all paths of a given program. The assertion checking problem is parameterized by the *class of programs* and the *expression language*. It is decidable when we consider some *simple* classes of programs – such as programs that only contain assignments, nondeterministic conditionals, and loops – and some *simple* expression languages — such as uninterpreted symbols (Gulwani and Necula, 2004; Gulwani and Tiwari, 2007a) or linear arithmetic (Karr, 1976; Gulwani and Tiwari, 2007a). Allowing conditionals in the programs makes the problem undecidable (Müller-Olm and Seidl, 2004; Müller-Olm et al., 2005).

The assertion checking problem becomes significantly more difficult if the simple programming model is extended with procedure calls. If we restrict our attention to programs with assignments, nondeterministic conditionals, loops, and procedure calls, and the expression language of uninterpreted symbols, then the general problem of deciding if terms s and t evaluate to the same value on all paths of a program is open. This problem is related to the interprocedural global value numbering problem, see Gulwani and Tiwari (2007b) for more discussion.

It was recently pointed out that in the absence of procedure calls, the assertion checking problem is closely related to classical term unification and in the presence of procedure calls, it is related to the one context unification problem (Gulwani and Tiwari, 2007b). This connection has led to many new complexity results for assertion checking in the absence of procedure calls (Gulwani and Tiwari, 2007a). The study of one context unification will thus lead to new results on the decidability and complexity of assertion checking in programs with procedure calls.

In the special case when the signature contains only unary function symbols, Gulwani and Tiwari (2007b) showed that the class of one context unification problems — that arises in interprocedural analysis of a class of programs where e.g. procedures are restricted to have at most one argument – is decidable in polynomial time. This case exactly corresponds to word unification with only one variable, which is decidable in time $O(n\log(n))$ time (Dabrowski and Plandowski, 2002). Using procedure calls with more than one argument, variables correspond to contexts rather than words. This leads to the generalization of the one context unification problem considered in this paper.

We only provide an example here to illustrate the connection of assertion checking with one context unification. Consider the procedure in Fig. 1. Since the procedure $P()$ could be called at multiple locations in the program under different "contexts" (that is, different valuations for program variables and procedure call stack), we compute a *summary* for $P()$ by analyzing it for a *generic*

context (Gulwani and Tiwari, 2007b). The exact form of the invariant guaranteed by $P()$ (at Line 5) depends on the form of the invariant that is given to it (at Line 1). Since we do not know the latter, we assume that a *generic* invariant, $x = F[y]$, where F is an unknown context, holds at Line 5, and we try to compute the *condition* on F and x, y at Line 1 that will guarantee that $x = F[y]$ is indeed an invariant at Line 5. This condition is computed by propagating $x = F[y]$ backwards. Across loops, backward propagation is performed until a fixpoint is reached. The important point to note here is that programs that contain only nondeterministic conditionals and assignments (such as the procedure $P()$ in Fig. 1) cannot distinguish between equation sets that have the *same set of unifiers*, such as $\{f(x,z) = f(y,z)\}$ and $\{x = y\}$. (In other words, for such programs, $f(x,z) = f(y,z)$ will be an invariant at a program point iff $x = y$ is an invariant at that program point.) Thus, we can detect a fixpoint by checking if a newly added equation is redundant — that is, it does not change the set of unifiers. In the summary computation shown in Fig. 1, at Line 3, we can determine that adding the equation $f(f(f(x,z),z),z) = F[f(f(f(y,z),z),z)]$ to the set $\{f(x,z) = F[f(y,z)], f(f(x,z),z) = F[f(f(y,z),z)]\}$ does not change the set of unifiers, which is $\{F \mapsto f(\bullet,z)^n, x \mapsto F[y] \mid n = 0, 1, \ldots\}$. Hence $f(f(f(x,z),z),z) = F[f(f(f(y,z),z),z)]$ is redundant. Finally, at Line 1, we can again see that only the shown equations are non-redundant: the unifiers of $f(gx,z) = F[gf(y,z)]$ are $\{F \mapsto f(\bullet,z), x \mapsto f(y,z)\}$ and $\{F \mapsto f(gF',z), x \mapsto F'[f(y,z)]\}$; and using also the second equation, we obtain only the unifier $\{F \mapsto f(\bullet,z), x \mapsto f(y,z)\}$. This unifier also solves all the other equations and is the required precondition that guarantees the postcondition $x = F[y]$ at Line 5.

1.3. Overview of obtained results

The *one context unification* problem takes as input a set Δ of term equations containing at most one context variable F, and asks for the existence of a substitution σ (for the first-order variables and the context variable F) that is a unifier of all equations in Δ.

The *redundancy problem for one context unification* takes as input a set Δ of term equations and an equation $s \doteq t \in \Delta$, where Δ contains at most one context variable F, and asks whether for every unifier σ of $\Delta \setminus \{s \doteq t\}$, the equation $\sigma(s) = \sigma(t)$ holds. If it does, then we say that $s \doteq t$ is *redundant* in Δ.

The *finite redundancy property for one context unification* is the question whether any non-redundant sequence $e_1, e_2, e_3, \ldots$ of equations is finite, under the condition that terms are built using a fixed finite set of first-order variables and a single context variable F. Here we say that a sequence $e_1, e_2, \ldots$ is *non-redundant* if for every i, the equation e_i is not redundant in $\{e_1, \ldots, e_i\}$.

Note that redundancy notions for first-order unification were also discussed in Lassez et al. (1988). There the relationship between disunification, redundancy and entailment of constraints are investigated.

We prove the following main results in this paper.

Theorem 1.1. *One context unification is in NP. Moreover, a complete set of unifiers can be computed that is at most exponentially large, where every unifier can be represented in polynomial space, and this set can be computed in at most exponential time.*

Note that one context unification is finitary if our representation using exponent expressions is used, but infinitary in the usual sense: the equation $F[f(a)] \doteq f(F[a])$, which mimics the word equation $xa \doteq ax$, has no finite complete set of unifiers.

We leave open the question about NP-hardness of one context unification. This appears to be far from straightforward, since to our knowledge, there is also no NP-hardness proof for context unification with a fixed number of context variables. In particular there is no known nontrivial lower complexity bound for two context unification (Schmidt-Schauß and Schulz, 2002).

Theorem 1.2. *The redundancy problem for one context unification is in coNP.*

Theorem 1.3. *The finite redundancy property for one context unification holds.*

Our algorithm to solve the one context unification problem runs in two phases. In the first phase, we nondeterministically guess the value of the context variable F to be either a fixed context $C[\bullet]$

or a context $C^{|p|N}[\bullet]$ with unknown integer exponent N (where p is the position of the hole in C and $C^{|p|N}$ denotes the context $C[C[\ldots[\bullet]\ldots]]$ obtained by iterating C until the hole is at depth $|p|N$). The number $|p|$ is polynomial in the input size of the equations. We replace F by $C[\bullet]$ or $C^{|p|N}[\bullet]$ and thus get a unification problem containing first-order variables and possibly the unknown N occurring in *exponent expressions* $C^{|p|N}[x]$. The second phase solves the unification problem over such exponent expressions in polynomial time. We show that we only need to search for solutions that map N to *small values*, linearly dependent on the size. Once N is fixed to a concrete integer value, the exponent equation unification problem reduces to the classical first-order term unification problem.

The first phase may require search and is responsible for "in NP", whereas the second phase can be performed in polynomial time. In special situations, the first phase can be done in polynomial time too, and hence one context unification is polynomial in these cases.

Theorem 1.4. *One context unification is solvable in polynomial time if the input contains an equation of the form* $F(s) \doteq C[F(t)]$.

Theorem 1.5. *If the number of first-order variables is fixed (say k), then one context unification is solvable in polynomial time.*

The proofs we give for these results, except for the last one (Theorem 1.5), are valid even if we assume that the input is represented by a directed acyclic graph, i.e. terms of the input are already represented by some compression. But in our proof of Theorem 1.5 we need to assume that the input terms are given as trees without compression.

2. Preliminaries: Term equations and solutions

A *signature* $\Sigma = \bigcup_i \Sigma_i$ is a finite set of function symbols and constants indexed by their *arity i*. A first-order *term t* is constructed over the signature Σ using function symbols, constants and variables in the usual way. We use the standard notation $f(t_1, \ldots, t_m)$ for a term. The set of variables that occurs in a term t is denoted as $V(t)$. A *position* is a string of non-negative integers. We use . as the string concatenation operator and $|p|$ to denote the length of the string p. The *position* p in a term, *subterm* $t|_p$ of term t at position p, and the term $t[s]_p$ obtained by replacing $t|_p$ by s are defined in the standard way. For example, if t is $f(a, g(b, h(c)), d)$, then $t|_{2.2.1} = c$, and $t[d]_{2.2} = f(a, g(b, d), d)$. The empty sequence corresponds to the root position, where we sometimes write $\mathtt{root}(s)$ to denote the function symbol at the root. The *depth* of a term t corresponds to the maximal length of a position in t.

A (first-order) *context* C, sometimes written as $C[\bullet]$, is a first-order term with a single hole, where, syntactically, the hole is treated like a distinguished constant from the signature. Sometimes we make the position p of the hole explicit by writing $C[\bullet]_p$. We use $\mathtt{hp}(C)$ to denote the position of the hole (*h*ole *p*osition) of C. When the hole is replaced by a given term t we represent it as $C[t]$ (or $C[t]_p$ when the position is to be made explicit).

We will also use *second-order terms*, generalizing first-order terms, where unary context variables, written $F, F', \ldots$ are permitted in function positions. In the solution process, we will also use *extended second-order terms*, which allow *exponent expressions* of the form $C^{aN+b}[\bullet]$ at function positions, where C is a (first-order) context, a, b are integers and N is an integer variable. The notation $C^m[\bullet]$ is formally defined in Definition 3.1. A *substitution* σ may replace first-order variables by terms, context variables by contexts and integer variables by non-negative integers, with the implicit condition that exponents must be non-negative. For example, $\{F \mapsto f(\bullet, g(x))\}$ is a substitution. In the case of ground terms (i.e. without occurrences of variables), we assume that the terms, in particular the exponent expressions, are simplified to first-order ground terms.

An *equation* is a pair of terms, written as $t_1 \doteq t_2$. The terms can contain variables. If terms are restricted to be first order, second order or otherwise, this will be made explicit later. A *solution* for $t_1 \doteq t_2$ is a substitution σ for the variables satisfying $\sigma(t_1) = \sigma(t_2)$ and such that $\sigma(t_1), \sigma(t_2)$ are ground terms.

Summarizing the notation, we use the symbols $x, y, z, \ldots$ for denoting first-order variables; $F, F', \ldots$ for context variables; $N, N', \ldots$ for integer variables, $f, g, h, \ldots$ for function symbols in Σ; $C[\bullet], D[\bullet], \ldots$ for contexts; $s, t, t', u, \ldots$ for terms; $\sigma, \theta, \ldots$ for substitutions; $k, n, \ldots$ for non-negative integers; p, q for positions in terms or of the hole in a context; and Δ for representing a set

of equations. The application of a substitution σ to a term s is denoted by $\sigma(s)$ and $\sigma \circ \theta$ denotes the composition of the substitutions σ, θ, where $(\sigma \circ \theta)(s)$ means $\sigma(\theta(s))$.

3. The general scheme

Given a set Δ of equations over terms containing some first-order variables and at most one context variable F, we present an algorithm that determines if the set Δ has a solution. We will argue that our procedure is in NP.

The algorithm is described as an inference system that manipulates the set Δ. The inference system runs in two phases. In a first phase, the inference rules deal with the context variable F. The first phase ends when F is eliminated. At the end of the first phase, we obtain either a first-order term unification problem, or a term unification problem containing subexpressions of the form $C^{|p|N}$, where C is a context, p is the position of the hole in C, and N is a variable ranging over non-negative integers. The inference rules in the second phase deal with this new kind of expressions and eliminate them.

We use $\mathtt{hp}(C)$ to denote the position of the hole (hole position) of C and $\mathtt{sp}(C)$ to denote the list of function symbols occurring on the path from the root to $\mathtt{hp}(C)$ (spine) in C. More formally, if $\mathtt{hp}(C)$ is of the form $i_1.i_2.\cdots.i_{|\mathtt{hp}(C)|}$, then $\mathtt{sp}(C)$ is $\mathtt{root}(C), \mathtt{root}(C|_{i_1}), \mathtt{root}(C|_{i_1.i_2}), \ldots, \mathtt{root}(C|_{\mathtt{hp}(C)-1})$.

The following definition clarifies the semantics of C^n.

Definition 3.1 (*Rotation and Context with Exponent*). Let $C[\bullet] := f(u_1, \ldots, u_{i-1}, D[\bullet], u_{i+1}, \ldots, u_m)$ be a non-empty context. For any non-negative integer $n \geq 0$, we define $\mathtt{rot}(C, n)$ and $C^n[\bullet]$ recursively as follows:

$$
\begin{aligned}
\mathtt{rot}(C, 0) &:= C \\
\mathtt{rot}(C, n) &:= \mathtt{rot}(D[f(u_1, \ldots, u_{i-1}, \bullet, u_{i+1}, \ldots, u_m)], n-1) \\
C^0[\bullet] &:= \bullet \\
C^n[\bullet] &:= f(u_1, \ldots, \mathtt{rot}(C, 1)^{n-1}[\bullet], \ldots, u_m)
\end{aligned}
$$

For any non-empty context C and any $n \geq 0$, note that $\mathtt{rot}(C[\bullet]_p, n) = \mathtt{rot}(C[\bullet]_p, n \bmod |p|)$ and $|\mathtt{hp}(C)| = |\mathtt{hp}(\mathtt{rot}(C, n))|$. Also note that C^1 need not be equal to C, but $C^{|\mathtt{hp}(C)|} = C$. Similarly, while $C^m[C^n[\bullet]]$ need not be equal to $C^{m+n}[\bullet]$, it is the case that $C^m[\mathtt{rot}(C, m)^n[\bullet]] = C^{m+n}[\bullet]$.

Example 3.2. If f and g are unary symbols and $C[\bullet] = f(g(\bullet))$, then $\mathtt{rot}(C, 0) = \mathtt{rot}(C, 2) = \mathtt{rot}(C, 4) = C$ and $\mathtt{rot}(C, 1) = \mathtt{rot}(C, 3) = g(f(\bullet))$. While $C^1(a)$ represents $f(a)$, the notation $C^5(a)$ succinctly represents the term $f(g(f(g(f(a)))))$.

We next define *unifiers* and allow for exponent expressions to appear in unifiers.

Definition 3.3. Given a set Δ of equations of second-order terms (i.e. without exponent expressions), a *unifier* σ is a substitution, possibly containing expressions of the form $C_p^{|p|N}$ for some fixed single integer variable N, such that for every substitution δ that instantiates N by a non-negative number and for every equation $s \doteq t \in \Delta$, we have $\delta(\sigma(s)) = \delta(\sigma(t))$. We speak of a *ground unifier* or a *solution*, if the unifier maps all variables to ground terms, in which case no exponent expressions are required.

A set S of unifiers, where we assume that only a single integer variable N is used, is a *complete set of unifiers* of Δ, iff for every ground unifier γ of Δ (i.e. for every solution), there is a $\sigma \in S$, and a ground substitution ρ such that for all variables x occurring in Δ, we have $\gamma(x) = \rho(\sigma(x))$. By abuse of notations we will speak of *most general unifiers* for the elements of S.

4. PHASE1 inference system: Eliminating the context variable

The PHASE1 inference rules are given in Fig. 2. These inference rules are applied non-deterministically to transform the current set Δ into one of the finitely many possible sets $\Delta_1, \ldots, \Delta_k$. The first phase ends when either a contradiction ($\perp$) is reached, or the context variable disappears.

Decompose:
$$\frac{\Delta \cup \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}}{\Delta \cup \{t_1 \doteq u_1,\ldots,t_n \doteq u_n\}}$$

where α is either a function symbol ($n = \mathtt{arity}(\alpha)$) or a first-order variable ($n = 0$) or a context variable ($n = 1$). One of $\alpha(t_1,\ldots,t_n)$ or $\alpha(u_1,\ldots,u_n)$ must have maximal depth among all terms in $\Delta \cup \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}$

Var-Elim:
$$\frac{\Delta \cup \{x \doteq t\}}{\langle x \mapsto t\rangle(\Delta)}$$

where x is a first order variable that does not occur in t.

Var-Elim2:
$$\frac{\Delta \cup \{F(t) \doteq C[x]\}}{(\langle x \mapsto F'(\rho(t))\rangle \circ \rho)(\Delta)}$$

where $C \neq \bullet$ is guessed (note that C does not contain F) and $\rho = \langle F \mapsto C[F'(\bullet)]\rangle$, and where x is not contained in t, and if F occurs in t, then x must not occur in C.

CVar-Elim:
$$\frac{\Delta \cup \{F(t) \doteq C[u]\}}{\langle F \mapsto C[\bullet]\rangle(\Delta \cup \{t \doteq u\})}$$

where C is guessed (note that C does not contain F).

CVar-Elim2:
$$\frac{\Delta \cup \{F(t) \doteq C[F(u)]_p\}}{\langle F \mapsto C^{|p|N+k}[\bullet]\rangle(\Delta \cup \{t \doteq \mathtt{rot}(C,k)[u]\})}$$

where k is guessed such that $0 \leq k < |p|$ and $p = \mathtt{hp}(C[\bullet])$ (note that C does not contain F).

Occurs-Check:
$$\frac{\Delta \cup \{x \doteq t\}}{\bot}$$

where x occurs in t and $t \neq F(\ldots F(x)\ldots)$, i.e. t is not a term consisting only of F's and x.

Fail:
$$\frac{\Delta \cup \{f(t_1,\ldots,t_n) \doteq g(u_1,\ldots,u_m)\}}{\bot}$$

where $f \neq g$.

Fig. 2. PHASE1 inference system for eliminating a context variable. The inference rules are applicable only when the side-condition holds.

For efficiency reasons we will use a DAG-representation of terms and contexts, but use a notation as for terms.

The first two rules – `Decompose` and `Var-Elim` – are the standard rules to simplify a unification problem and eliminate a first-order variable (Baader and Snyder, 2001). The rule `Var-Elim2` partially guesses the context variable F in terms of a new context variable F' and eliminates a first-order variable x. The rule `CVar-Elim` eliminates F by (non-deterministically) guessing $C[\bullet]$ as a value for F. Note that $C[\bullet]$ here means a (first-order) context, thus it cannot contain a context variable. The rule `CVar-Elim2` eliminates F by again (non-deterministically) guessing a position of length $0 \leq k < |p|$ that determines the value of F using the exponent notation, cf. Definition 3.1. Note that it introduces a new variable N ranging over the non-negative integers. The instantiation $F \mapsto C^{|p|N+k}$ introduces exponent expressions. The term $C^{|p|N+k}$ is actually represented as $C^{|p|N}[C^k]$, that is, *only*

exponent expressions of the form $C^{|p|N}$ are introduced, and the term C^k is expanded (into a regular term) according to Definition 3.1.

Example 4.1. Let $\Delta := \{f(F[f(y,z)],z) \doteq F[f(f(y,z),z)]\}$. Rule `CVar-Elim2` is applicable to Δ. Here $C = f(\bullet,z)$ and hence $|p| = 1$. Therefore, k can only be 0 and we get the substitution $\langle F \mapsto f(\bullet,z)^N\rangle$. Applying `CVar-Elim2` using this substitution, we get the new set $\{f(f(y,z),z) \doteq \mathtt{rot}(f(\bullet,z),0)[f(y,z)]\}$, which is simply $\{f(f(y,z),z) \doteq f(f(y,z),z)\}$. Fig. 5 contains a bigger example.

The following correctness statements and their proof sketches provide further intuition for the inference rules. Note that in each inference step, the new set Δ' is obtained from the old set Δ by removing some equations, adding some new equations, and applying a substitution to all the terms.

Lemma 4.2 (*Soundness*). *Let $\Delta_1 \vdash \Delta_2$ be a* PHASE1 *inference step and let σ_1 be the substitution used in this inference step. If σ is a solution for Δ_2, then $\sigma \circ \sigma_1$ is a solution for Δ_1.*

Proof. For each PHASE1 inference rule, we can write Δ_2 as $\sigma_1(\Delta_1 - \Delta_d \cup \Delta_a)$, where Δ_d are the deleted equations and Δ_a are the added equations. If $\Delta_a \neq \emptyset$, it is easily verified that if σ is a solution for $\sigma_1(\Delta_a)$, then $\sigma \circ \sigma_1$ is a solution of Δ_d. If $\Delta_a = \emptyset$, as in the rules `Var-Elim` and `Var-Elim2`, it is easily verified that σ_1 is a solution of Δ_d, hence also $\sigma \circ \sigma_1$ is a solution of Δ_d.

Consider, for example, the rule `CVar-Elim2`. Let $\sigma_1 := \langle F \mapsto C^{|p|N+k}[\bullet]\rangle$ and suppose σ is a solution of $\sigma_1(\Delta \cup \{t \doteq \mathtt{rot}(C,k)[u]\})$. Obviously, $\sigma \circ \sigma_1$ solves Δ. We show that $\sigma \circ \sigma_1$ also solves $F(t) \doteq C[F(u)]_p$ as follows:

$$\begin{aligned}
\sigma \circ \sigma_1(F(t)) &= \sigma(C^{|p|N+k})[\sigma \circ \sigma_1(t)] && = \sigma(C^{|p|N+k})[\sigma \circ \sigma_1(\mathtt{rot}(C,k)[u])]\\
&= \sigma(C^{|p|N+k})[\sigma(\mathtt{rot}(C,k)[\sigma_1(u)])] && = \sigma(C^{|p|N+k}[\mathtt{rot}(C,k)[\sigma_1(u)]])\\
&= \sigma(C[C^{|p|N+k}[\sigma_1(u)]]) && = \sigma \circ \sigma_1(C[F(u)])
\end{aligned}$$

Apart from the definitions of σ and σ_1, we also use the fact that $C^{|p|N+k}[\mathtt{rot}(C,k)[\bullet]] = C[C^{|p|N+k}[\bullet]]$ above. Soundness of all other rules can be argued similarly. □

Lemma 4.3 (*Completeness*). *Suppose that Δ can be transformed to one of $\Delta_1, \ldots, \Delta_m$ by the* PHASE1 *inference system using substitutions $\sigma_1, \ldots, \sigma_m$ respectively. If σ is a solution of Δ, then there exists a solution θ of some Δ_i such that $\sigma(x) = \theta \circ \sigma_i(x)$, $\sigma(F) = \theta \circ \sigma_i(F)$ and $\sigma(N) = \theta \circ \sigma_i(N)$, for all variables x, F, N occurring in Δ.*

Proof. Suppose that the solution σ for Δ instantiates F by a context $D[\bullet]_p$. We consider different cases based on the form of the equations in Δ. The interesting case is when Δ contains an equation $F(t) \doteq s$. The choice of which inference rule to apply to Δ to identify the required Δ_i can be guided by $D[\bullet]_p$ and the form of s.

Case 1: p is a position of s. Then p is a prefix of all positions labeled with F in s, because, if not, then the context D will properly contain D, which is not possible. Thus, s can be written as $D'[u]_p$ for a context D' not containing F, and such that $\sigma(D') = D$. In this case, the rule `CVar-Elim` can be used to get the desired Δ_i from Δ.

Case 2: p is not a position of s. Then s contains a unique maximal position q that is a proper prefix of p, such that the roots of $s|_{q'}$ and $D|_{q'}$ are identical signature symbols for all proper prefixes q' of q. There are two cases: $s|_q = x$ for some first-order variable x and $s = C[x]_q$, or $s|_q = F(s')$ and $s = C[F(s')]$, and F does not occur in C: Otherwise, the context D would properly contain D, which is not possible. If $s|_q = x$ is a first-order variable, then s is of the form $C[x]_q$, and D is of the form $\sigma(C)[D'] = \sigma(F)$. A further equality is $D[\sigma(t)] = \sigma(C)\sigma(x)]$. In this case, the rule `Var-Elim2` can be used: the substitution $\rho = \langle F \mapsto C[F'(\bullet)]\rangle$ replaces the equation $F(t) \doteq C[x]$ by the equation $C[F'(\rho(t))] \doteq C[x]$, which is equivalent to the $F'(\rho(t)) \doteq x$. Since the system is unifiable, x is not contained in $F'(\rho(t))$, hence the second part of the conditions of `Var-Elim2` is satisfied. The second substitution part is then $\langle x \mapsto F'(\rho(t))\rangle$, where $x \notin FV(F'(\rho(t)))$. We obtain the required Δ_i where the required θ is the same as σ except that $\theta(F') = D'$.

Now, suppose $s|_q$ is $F(s')$. We know that F does not occur in C. Unifiability of the equations and the fact that D cannot be a proper subcontext of itself enforce that p has to be of the form $q^n q'$ for a prefix

q' of q, and if s is written as $C[F(s')]_q$, then D has to be of the form $\sigma(C)^{|q|n+|q'|}$. This means that the rule CVar-Elim2 can be used to get the required Δ_i.
For other choices of the solution σ and the set Δ, we can argue similarly to complete the proof. □

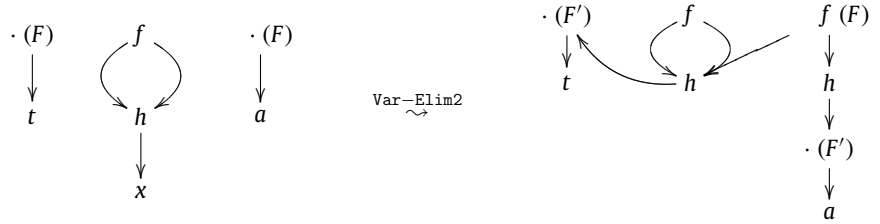
Note that if Δ contains a context variable, then at least one of the PHASE1 rules can be used. Hence, when no more rules can be applied, we are guaranteed to have eliminated all context variables. We now argue that the use of directed acyclic graphs (DAGs) helps to keep efficiency; in particular that (a) the size of the DAGs generated in the first phase is polynomially bounded, and (b) the first phase terminates in a polynomial number of (non-deterministic) steps.

The representation of terms as DAGs is standard. Contexts are represented like terms (in the DAG) with the hole $\bullet$ as a constant. A slight exception is $C^{|p|N}$ which is represented using a single node labeled with the triple $(C, |p|, N)$. Recall that $C^{|p|N+k}$ is always represented as $C^{|p|N}[C^k]$, where C^k is represented as a regular term.

The only rules that may increase the number of nodes in the DAG are rules that instantiate the context variable F, namely Var-Elim2, CVar-Elim, and CVar-Elim2. The rules CVar-Elim and CVar-Elim2 can be applied at most once, since these rules eliminate all occurrences of context variables. The number of additional nodes to create a context C is at most $\mathtt{hp}(C)$, which is bounded by the current number of nodes in the DAGs. This implies that the application of CVar-Elim and CVar-Elim2 may cause at most a quadratic blowup. Now consider the application of the substitution $\langle F \mapsto C[F'(\bullet)]_p\rangle$ in Var-Elim2. This causes an addition of $l * |p|$ new nodes in the DAG $-$ l copies of each node in $\mathtt{sp}(C)$ $-$ where l is the number of occurrences of F in Δ.

Note that an application of the rule Var-Elim2 implies (by the definition of contexts) that there is no occurrence of F in $C[x]_p$. Hence, an application of Var-Elim2 makes a copy of a node only when it has no context variable F below it. However, each newly added node will necessarily have a context variable (F') below itself. The property that a node has a context variable below it is preserved by the rules Decompose, Var-Elim, and Var-Elim2 for all nodes. Therefore, a newly created node will never be copied again. Thus, the number of new nodes that can be added by Var-Elim2 is bounded by $l * n$, where n is the node count of the original DAG. It is easy to see that the current number of nodes labeled with the context variable is also bounded above by n, since the sum of the number of first-order variable nodes and of the nodes labeled with the context variable is not increased by Var-Elim2. This proves that the size of the DAG created in the first phase is polynomially bounded.

Example 4.4. Consider $\Delta := \{F(t) \doteq f(h(x), h(x)), F(a) \doteq x\}$. Applying Var-Elim2 with $\langle x \mapsto F'(t)\rangle(\langle F \mapsto f(h(x), h(F'))\rangle)$ gives $\Delta' := \{f(h(F'(t)), h(F'(a))) \doteq F'(t)\}$. The DAG representations of (all terms in) Δ and Δ' are shown below. Note that the node representing $F(a)$ in the original DAG causes *copying* and the creation of two new nodes, marked with f and h. However, these new nodes will never get copied in the future as they will continue to have a context variable F' below them.



Lemma 4.5 (*Termination*). *If directed acyclic graphs (DAGs) are used as a data-structure to represent terms, any* PHASE1 *derivation terminates in a polynomial number of steps.*

Proof. Any PHASE1 derivation is immediately terminated if we apply CVar-Elim, CVar-Elim2, Var-Elim2 where the second part is Occurs-Check, Occurs-Check, or Fail. Single rule applications together with their applicability checks can be done in polynomial time on DAGs using standard techniques. The rules Var-Elim and Var-Elim2 eliminate a first-order variable, and hence they can be applied at most a linear number of times. The rule Decompose preserves the number of variables. Note that the maximal number of possible equations is quadratic in the number of nodes

in the DAG. Sequences of applications that consist only of Decompose have an at most polynomial length, since in every step at least one equation is processed that cannot occur again in this sequence. Since the number of nodes in the DAG is polynomially bounded, termination is guaranteed in (non-deterministic) polynomial time. □

Lemma 4.5 and the polynomial bound on the node count of the DAG show that the first phase runs in non-deterministic polynomial time. We remark here that the rules Decompose and Var-Elim can be applied eagerly (since they correspond to "don't care" non-determinism). The rules Var-Elim2, CVar-Elim, and CVar-Elim2 involve "don't know" non-deterministic guesses.

Lemma 4.6 (*Result of Phase 1*). *The first phase terminates either with Fail (i.e. $\bot$), or with success and the output is an empty set of equations or a set of equations including exponent expressions. If the output is an empty set, then the unifier σ is the composition of the substitutions of the rule applications. The unifier can be represented in polynomial space if the substitution is performed using DAGs. If the output is a set of equations including exponent expressions, then the combined partial solution can be represented in polynomial space, as well as the set of equations.*

Proof. This follows from termination in Lemma 4.5 and the arguments on the polynomial number of nodes in the DAG. □

5. PHASE2 inference system: Solving exponent equations

In this section we solve the unification problem for sets of equations containing terms constructed over a signature Σ of uninterpreted function symbols, a set of first-order variables, and a special expression $C^{|p|N}$, where C is a context, $p = \mathtt{hp}(C)$ and N is a variable taking values in the set of non-negative integers. Note that we may assume that terms are represented as DAGs (see below). For the purposes of this section, we assume that C is fixed. Thus p is also fixed. The expression $C^{|p|N}$ may occur several times, but it is unique. Equations over such terms are called *initial exponent equations*. This form of equation corresponds to the output of the first phase, but the output of the first phase has to be preprocessed to ensure efficiency, as explained below.

We prove that the unification problem for initial exponent equations is solvable in polynomial time, and moreover, that an explicit description of a complete set of unifiers can be computed also in polynomial time. This is done by splitting the solutions into small ones and big ones, depending on the instantiation for N. We prove the following interesting property: A set of initial exponent equations Δ has a solution if and only if it has a solution where N is replaced by a non-negative integer bounded by $\mathtt{nf}(\Delta) + 2$, where $\mathtt{nf}(\Delta)$ is just the number of occurrences of function symbols in Δ, without counting the ones in expressions $C^{|p|N}$. Thus, an efficient decision algorithm is directly obtained by considering all these possible replacements and solving each of them with a fast algorithm for first-order unification.

In order to prove this bound for N, we solve initial exponent equations Δ using an inference system. While the initial set Δ only contains exponent terms of the form $C^{|p|N}$, the sets derived using the inference rules can, in general, contain special expressions (of arity 1) of the form $D^{|p|N-k}$, where D is a rotation of the original context C (after flattening, see the preprocessing step below), and k is a non-negative integer. The interpretation of these expressions under substitution is analogous to before, but now, $|p|N - k \geq 0$ is an implicit condition for N. Hence, when N is replaced by a non-negative integer n, it should be the case that $|p|n - k \geq 0$ and $D^{|p|n-k}$ is then the corresponding context according to Definition 3.1. A set of equations containing these kinds of expressions is called *a set of exponent equations*.

Our inference system uses multi-equations instead of just equations. A *multi-equation* M is a set of terms denoted as $s_1 \doteq s_2 \doteq s_3 \doteq \ldots s_{n-1} \doteq s_n$. It has the same meaning as the set of equations $s_1 \doteq s_2, s_2 \doteq s_3, \ldots, s_{n-1} \doteq s_n$. But having multi-equations has some advantages from a computational point of view. They avoid duplication of terms, in particular they avoid the substitution of non-variable terms t for x, when an equation $x \doteq t$ occurs.

Our inference system deals just with *flattened terms*, that is, terms with depth at most one, and such that all the expressions of the form $D^{|p|N-k}$ satisfy that D is a *flattened context*, i.e. a context whose

subterms without the hole have depth 0. Since our original set of equations is not necessarily flattened, we need to transform it into a flattened one, while preserving its set of solutions.

Definition 5.1 (*Flattening*). Consider Δ to be a set of equations with first-order terms including also expressions of the form $D^{|p|N-k}$. We define $\texttt{flatten}(\Delta)$ to be the set of equations resulting from applying the following transformation process to Δ as many times as possible.

- *(Flattening step)* Let t be either a proper (non-variable) subterm of a term in Δ or a proper (non-variable) subterm of a context D. In the latter case we assume that t does not contain the hole. Then, we create a new variable z, replace t by z everywhere in Δ, including the occurrences of t in any expression of the form $D^{|p|N-k}$, and add the equation $z \doteq t$ to Δ.

We say that a set of equations is *flattened* when the flattening step can not be applied to it.

The output of the first phase is a set of equations, where the terms are represented by DAGs. Applying the flattening to DAGs is exactly the same as for terms, and the results are terms, where every term-node in the DAG is represented by a first-order variable. Flattening of contexts produces contexts where every subterm not containing the hole is a first-order variable. Flattening causes an at most linear space increase.

In order to describe the inference rules, we need the following concepts of compatible contexts and of expansion of two compatible contexts.

Definition 5.2 (*Compatible Contexts and Expansion*). Two contexts C_1 and C_2 are *compatible* if $\texttt{hp}(C_1) = \texttt{hp}(C_2)$ and $\texttt{sp}(C_1) = \texttt{sp}(C_2)$.

For compatible contexts C_1 and C_2, we recursively define the set $\texttt{expand}(C_1, C_2)$ as follows: if $C_1 := \bullet$ and $C_2 := \bullet$, then $\texttt{expand}(C_1, C_2) := \emptyset$; and if $C_1 := f(u_1, \ldots, u_{i-1}, D_1[\bullet], u_{i+1}, \ldots, u_n)$ and $C_2 := f(v_1, \ldots, v_{i-1}, D_2[\bullet], v_{i+1}, \ldots, v_n)$, then $\texttt{expand}(C_1, C_2) := \{u_1 \doteq v_1, \ldots, u_{i-1} \doteq v_{i-1}, u_{i+1} \doteq v_{i+1}, \ldots, u_n \doteq v_n\} \cup \texttt{expand}(D_1, D_2)$.

5.1. *The* PHASE2 *inference system*

The idea behind the PHASE2 inference system (shown in Fig. 3) is to simulate the usual decomposition rules for term unification, but applied to terms that may also contain expressions $D^{|p|N-k}$ of arity 1. For convenience, we define the inference system on $\Delta_s; \Delta_u$, where the solved part Δ_s is a set of equations, and the unsolved part Δ_u is a set of multi-equations. Initially, the solved part Δ_s is empty. The inference system operates essentially only on the unsolved part, except that certain rules may move equations from the unsolved part into the solved part (Fig. 4).

Example 5.3. Consider $\Delta := \{f(\bullet, y)^{N-2}[z_2] \doteq f(\bullet, z)^N[x_4]\}$. The inference rule NN is applicable with $C_1 = f(\bullet, y)$, $C_2 = f(\bullet, z)$, $|p| = 1$, $k_1 = 2$ and $k_2 = 0$. The contexts C_1 and C_2 are compatible, and $\texttt{expand}(C_1, C_2) = \{y \doteq z\}$. Thus, applying NN to Δ gives $\{y \doteq z,\ \texttt{flatten}(z_2 \doteq f(\bullet, y)^2[x_4])\}$. Fig. 5 contains a bigger example.

The PHASE2 inference system is intended to compute only "big" unifiers, i.e. unifiers for which the replacement for N is "big". The solutions with a small instantiation for N are computed by scanning over all small instantiations of N and then using first-order unification for each. For a set of multi-equations Δ we define some measures on the unsolved part that will help in formally defining "big":

- $\texttt{nf}(\Delta)$ denotes the number of occurrences of function symbols in Δ_u without counting the ones in expressions $D^{|p|N-k}$,
- $\texttt{nx}(\Delta)$ denotes the number of occurrences of first-order variables in Δ_u, including the ones in the contexts.
- $\texttt{nx0}(\Delta)$ denotes the number of occurrences of first-order variables at depth 0 in Δ_u.
- $\texttt{nx1}(\Delta)$ denotes the number of occurrences of first-order variables at depth 1 or more in Δ_u, including the ones in the contexts.
- $\texttt{nk}(\Delta)$ denotes the sum of all the k's for all the occurrences of expressions $D^{p|N|-k}$ in Δ_u, and
- $\texttt{nc}(\Delta)$ denotes the number of occurrences of expressions $D^{|p|N-k}$ in Δ_u.

$$\text{(xx)}\quad \frac{\Delta_s;\Delta\cup\{x\doteq M_1, x\doteq M_2\}}{\Delta_s;\Delta\cup\{x\doteq M_1\doteq M_2\}}$$

$$\text{(xy)}\quad \frac{\Delta_s;\Delta\cup\{x\doteq y\doteq M\}}{\Delta_s\cup\{x\doteq y\};\langle x\mapsto y\rangle(\Delta\cup\{y\doteq M\})}\qquad \text{if } x\neq y$$

$$\text{(xM)}\quad \frac{\Delta_s;\Delta\cup\{x\doteq t\doteq M\}}{\Delta_s\cup\{x\doteq t\};\Delta\cup\{t\doteq M\}}\qquad \text{if } x \text{ does not occur in } \Delta, M, t.$$

$$\text{(1-alone)}\quad \frac{\Delta_s;\Delta\cup\{t\}}{\Delta_s;\Delta}$$

$$\text{(ff)}\quad \frac{\Delta_s;\Delta\cup\{f(x_1,\ldots,x_m)\doteq f(y_1,\ldots,y_m)\doteq M\}}{\Delta_s;\Delta\cup\{f(y_1,\ldots,y_m)\doteq M, x_1\doteq y_1,\ldots,x_m\doteq y_m\}}$$

$$\text{(NN)}\quad \frac{\Delta_s;\Delta\cup\{C_1^{|p|N-k_1}(x_1)\doteq C_2^{|p|N-k_2}(x_2)\doteq M\}}{\Delta_s;\Delta\cup\{C_2^{|p|N-k_2}(x_2)\doteq M\}\cup \mathtt{expand}(C_1,C_2)\cup \mathtt{flatten}(\{x_1\doteq(\mathtt{rot}(C_2,k_1'))^{k_1-k_2}(x_2)\})}$$
if C_1, C_2 are compatible where $k_1\geq k_2$ and $k_1' = |p| - k_1 \bmod |p|$

$$\text{(Nf)}\quad \frac{\Delta_s;\Delta\cup\{f(x_1,\ldots,x_m)\doteq(f(y_1,\ldots,y_{i-1},C_1[\bullet],y_{i+1},\ldots,y_m))^{|p|N-k}(x)\}}{\Delta_s;\Delta\cup\{x_1\doteq y_1,\ldots,x_{i-1}\doteq y_{i-1},x_{i+1}\doteq y_{i+1},\ldots,x_m\doteq y_m\}\cup\{x_i=(C_1[f(y_1,\ldots,\bullet,\ldots,y_m)])^{|p|N-k-1}(x)\}}$$

Fig. 3. The PHASE2 unification rules for multi-equations.

$$\text{(cycle)}\quad \frac{\Delta_s;\Delta\cup\{x_1\doteq t_1\doteq M_1\}\cup\ldots\cup\{x_n\doteq t_n\doteq M_n\}}{\bot}\qquad \begin{array}{l}\text{if } t_n\neq x_1\in \mathtt{V}(t_n),\\ t_1\neq x_2\in\mathtt{V}(t_1),\ldots\\ t_{n-1}\neq x_n\in\mathtt{V}(t_{n-1})\end{array}$$

$$\text{(clash)}\quad \frac{\Delta_s;\Delta\cup\{s\doteq t\doteq M\}}{\bot}\quad \text{if } \mathtt{root}(s),\mathtt{root}(t)\in\Sigma \text{ and } \mathtt{root}(s)\neq\mathtt{root}(t)$$

$$\text{(clashfC)}\quad \frac{\Delta_s;\Delta\cup\{f(\ldots)\doteq(g(\ldots,C_1[\bullet],\ldots))^{|p|N-k}(x)\doteq M\}}{\bot}\quad \text{if } f\neq g$$

$$\text{(clashCC)}\quad \frac{\Delta_s;\Delta\cup\{C_1^{|p|N-k_1}(x_1)\doteq C_2^{|p|N-k_2}(x_2)\doteq M\}}{\bot}\quad \text{if } C_1, C_2 \text{ are not compatible}$$

Fig. 4. The PHASE2 unification failure rules for multi-equations.

Let Δ_0 be some fixed set of initial exponent multi-equations. Recall that C and $p = \mathtt{hp}(C)$ are fixed. Define $B(\Delta_0) := 2 + \frac{\mathtt{nf}(\Delta_0)}{|p|}$.

Definition 5.4 (*Big Unifiers*). Let Δ be a set of exponent equations derived from the (implicitly assumed) set Δ_0. We say that σ is a *big unifier* of Δ if σ is a solution of Δ and $\sigma(N)\geq B(\Delta_0)$.

Example 5.5. Let Δ be $\{f(x)\doteq y, (g(f(\bullet)))^{2N-2}(y)\doteq g(z)\doteq(g(f(\bullet)))^{2N-5}(w)\}$. Then, $\mathtt{nf}(\Delta)=2$, $\mathtt{nc}(\Delta)=2$, $\mathtt{nk}(\Delta)=7$ and $\mathtt{nx}(\Delta)=5$. Here, $C=g(f(\bullet))$ and $|p|=2$.

We shall sometimes use the measures nf, nx, nk and nc without explicitly showing their argument (Δ), which either means the measures are being used as functions, or the argument is unambiguous and determined by the context.

Termination

We first show that any derivation using the above inference rules terminates in a polynomial number of steps. The following lemma follows by inspecting the inference rules.

Lemma 5.6. *Let $\Delta_1, \Delta_2, \ldots, \Delta_n$ be a* PHASE2 *derivation. Then,* $\mathtt{nf}(\Delta_n)+\mathtt{nk}(\Delta_n)\leq\mathtt{nf}(\Delta_1)+\mathtt{nk}(\Delta_1)$.

$$\{f(x) \doteq F[f(y)], f(f(x)) \doteq F[f(f(y))], f(f(f(x))) \doteq F[f(f(f(y)))]\}$$

$$\left\langle \begin{array}{l} x \mapsto F'[f(y)], \\ F \mapsto f(F'[\bullet]) \end{array} \right\rangle \quad \left\{ \begin{array}{c} f(f(F'[f(y)])) \doteq f(F'[f(f(y))]), \\ f(f(f(F'[f(y)]))) \doteq f(F'[f(f(f(y)))]) \end{array} \right\} \qquad \texttt{Var-Elim2}$$

$$\{f(F'[f(y)]) \doteq F'[f(f(y))], f(f(f(F'[f(y)]))) \doteq f(F'[f(f(f(y)))])\} \qquad \texttt{Decompose}$$

$$\langle F' \mapsto f^N[\bullet] \rangle \quad \{f(f(f(f^N[f(y)]))) \doteq f(f^N[f(f(f(y)))])\} \qquad \texttt{CVar-Elim2}$$

$$\left\{ \begin{array}{c} f(x_1) \doteq f(z_1), x_1 \doteq f(x_2), x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], x_4 \doteq f(y), \\ z_1 \doteq f^N[z_2], z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{Flatten}$$

$$\left\{ \begin{array}{c} x_1 \doteq z_1 \doteq f(x_2) \doteq f^N[z_2], x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{ff}$$

$$\left\langle \begin{array}{c} x_1 \mapsto z_1, \\ z_1 \mapsto f(x_2) \end{array} \right\rangle \quad \left\{ \begin{array}{c} f(x_2) \doteq f^N[z_2], x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{xy,xM}$$

$$\left\{ \begin{array}{c} x_2 \doteq f^{N-1}[z_2] \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{Nf}$$

$$\langle x_2 \mapsto f(x_3) \rangle \quad \left\{ \begin{array}{c} x_3 \doteq f^{N-2}[z_2] \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{Nf}$$

$$\langle x_3 \mapsto f^{N-2}[z_2] \rangle \quad \left\{ \begin{array}{c} z_2 \doteq f^2[x_4] \doteq f(z_3), \\ x_4 \doteq f(y), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\} \qquad \texttt{NN}$$

$$\langle z_2 \mapsto f^2[x_4] \rangle \quad \{z_3 \doteq f(x_4) \doteq f(z_4), x_4 \doteq f(y), z_4 \doteq f(y)\} \qquad \texttt{xM}$$

$$\langle z_3 \mapsto f(x_4) \rangle \quad \{x_4 \doteq z_4 \doteq f(y) \doteq f(y)\} \qquad \texttt{xM}$$

$$\langle x_4 \mapsto z_4, z_4 \mapsto f(y) \rangle \qquad \texttt{xM}$$

Fig. 5. Example illustrating the inference rules of the two phases. Some trivial applications are not shown, and we only show the insertions into the solved part.

Proof. It suffices to see that $\mathtt{nf} + \mathtt{nk}$ is either preserved or decreased after every rule application. This is easy by inspecting the inference rules. Rules (xx), (xy) and (xM) preserve $\mathtt{nf}$ and $\mathtt{nk}$. Rule (1-alone) either preserves or reduces $\mathtt{nf}$ and $\mathtt{nk}$. Rule (ff) reduces $\mathtt{nf}$ and preserves $\mathtt{nk}$. Rule (Nf) reduces $\mathtt{nf}$ by 1 and increases $\mathtt{nk}$ by 1. For the case of rule (NN), note that $\mathtt{expand}(C_1, C_2)$ does not add any function symbol. Note also that the addition of the term $(\mathtt{rot}(C_2, k_1'))^{k_1 - k_2}(t_2)$ increases $\mathtt{nf}$ by $k_1 - k_2$, while $\mathtt{nk}$ is reduced by k_1 since the term $C_1^{|p|N-k_1}(x_1)$ is removed. □

Corollary 5.7. *Let $\Delta_0, \Delta_1, \ldots, \Delta_n$ be a* PHASE2 *derivation starting from the initial set Δ_0 (whose $\mathtt{nk}(\Delta_0)$ is 0). Then, $\mathtt{nk}(\Delta_n) + \mathtt{nf}(\Delta_n) \leq \mathtt{nf}(\Delta_0)$, and $2 + \frac{(\mathtt{nk}(\Delta_n) + \mathtt{nf}(\Delta_n))}{|p|} \leq B(\Delta_0)$.*

Lemma 5.8. *The* PHASE2 *inference system terminates. The number of inference steps of a derivation starting from a given starting set of flattened exponent equations Δ is bounded by $(3 * \mathtt{MaxArity} * \mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + \mathtt{nx0} + 2 * \mathtt{nx1})(\Delta)$, where $\mathtt{MaxArity}$ is the maximum arity of the function symbols occurring in Δ.*

Proof. For termination, it suffices to see that the tuple $(\mathtt{nc}, \mathtt{nf}, \mathtt{nx})$ decreases after every rule application, where tuples are compared using the lexicographic extension of the usual ordering on integers. This is trivial by inspecting the inference rules. Rules (xx), (xy) and (xM) preserve nc and nf, but they reduce nx. Rule (1-alone) reduces some of nc, nf, nx, and either reduces or preserves the rest. Rules (ff) and (Nf) reduce nf and preserve nc and nx. For the case of rule (NN), we recall that $(\mathtt{rot}(C_2, k_1'))^{k_1-k_2}$ is a first-order context and not an expression of the form $D^{|p|N-k}$. Hence, this rule always reduces nc.

Now, for the termination measure, it is easy to see that $(3 * \mathtt{MaxArity} * \mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + \mathtt{nx0} + 2 * \mathtt{nx1})(\Delta)$ decreases after applying any of the inference rules. The most complicated case is rule (NN), due to the application of the flattening process. In this process, the number of newly added occurrences of variables at depth 1 is bounded by $\mathtt{MaxArity} * \mathtt{nk}$, and this is bounded by $\mathtt{MaxArity} * (\mathtt{nf} + \mathtt{nk})$. Similarly, the maximum number of newly added occurrences of variables at depth 0 is bounded by nk, and this is bounded by $\mathtt{nf} + \mathtt{nk}$. Thus, since nc is reduced by 1 when applying (NN), the expression $(3 * \mathtt{MaxArity} * \mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + \mathtt{nx0} + 2 * \mathtt{nx1})(\Delta)$ decreases if we only take into account the flattening process. But we must also take care of the expand process. Note that the occurrences of variables at depth 1 or more in C_1 disappear, but a double number of occurrences of variables at depth 0 is added due to the expand process. Summarizing, we conclude that $(3 * \mathtt{MaxArity} * \mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + \mathtt{nx0} + 2 * \mathtt{nx1})(\Delta)$ decreases after applying (NN). □

Correctness

The following property of big unifiers will allow us to justify compatibility of contexts of certain equations below.

Corollary 5.9. *In any* PHASE2 *derivation* $\Delta_0, \Delta_1, \ldots,$ *if* $D^{|p|N-k}$ *occurs in any* Δ_i *and* σ *is a big unifier of* Δ_i*, then* $\sigma(|p|N-k) \geq 2|p|$.

Proof. If $D^{|p|N-k}$ occurs in Δ_i, then note that $\sigma(|p|N-k) = |p|\sigma(N) - k \geq |p|(2+\mathtt{nf}(\Delta_0)/|p|) - k = 2|p| + \mathtt{nf}(\Delta_0) - k \geq 2|p| + \mathtt{nf}(\Delta_i) + \mathtt{nk}(\Delta_i) - k \geq 2|p|$, using Lemma 5.6 and Definition 5.4. □

The soundness of the PHASE2 inference system follows directly from inspecting the rules and applying Corollary 5.9.

Lemma 5.10 (*Soundness*). *Let* $\Delta_1 \vdash \Delta_2$ *be an inference step, and let* σ *be a big unifier of* Δ_2*. Then,* σ *is also a big unifier of* Δ_1.

Proof. For all rules, a big unifier σ of Δ_2 is also a big unifier of Δ_1, by inspecting the rules. It is crucial to take into account the solved part of Δ_2 for the rules (xy) and (xM). The detailed proof is straightforward, but tedious. We illustrate it for the rule (NN): Let σ be a big unifier of $\Delta \cup \{C_2^{|p|N-k_2}(x_2) \doteq M\} \cup \mathtt{expand}(C_1, C_2) \cup \mathtt{flatten}(\{x_1 \doteq (\mathtt{rot}(C_2, k_1'))^{k_1-k_2}(x_2)\})$ where C_1,C_2 are compatible, $k_1 \geq k_2$ and $k_1' = |p| - k_1 \bmod |p|$. We only need to prove that σ is a unifier of $C_1^{|p|N-k_1}(x_1) \doteq C_2^{|p|N-k_2}(x_2)$. Since σ is big, it follows from Corollary 5.9 that $\sigma(|p|N-k_1) > 0$ and $\sigma(|p|N-k_2) > 0$. Since σ unifies $\mathtt{expand}(C_1, C_2)$ and C_1 and C_2 are compatible, we can conclude that $\sigma(C_1) = \sigma(C_2)$. Since σ is a unifier of $x_1 \doteq (\mathtt{rot}(C_2, k_1'))^{k_1-k_2}(x_2)$, we see that σ is also a unifier of $C_1^{|p|N-k_1}(x_1)$ and $C_1^{|p|N-k_1}(\mathtt{rot}(C_2, k_1'))^{k_1-k_2}(x_2)$. The term $\sigma(C_1^{|p|N-k_1}(\mathtt{rot}(C_2, k_1'))^{k_1-k_2}(x_2))$ is equal to $\sigma(C_2^{|p|N-k_2}(x_2))$ using properties of rot and exponents, and the fact that $\sigma(C_1) = \sigma(C_2)$. □

Completeness of the inference system of Phase 2 depends on the following lemma that shows compatibility of two contexts C_1, C_2 that occur in exponent equations that are (roughly) of the form $C_1[C_1[\ldots]] \doteq C_2[C_2[\ldots]]$.

Lemma 5.11. *Let* C_1, C_2 *be two flattened contexts such that each one is a rotation of the other. Let* t_1, t_2 *be terms. Let* k_1, k_2 *be non-negative integers greater than or equal to* $2|p|$*. If* $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$ *has a solution, then* C_1 *and* C_2 *are compatible.*

Proof. We prove this by contradiction, i.e. under the assumption of incompatibility of C_1 and C_2 we prove that $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$ has no solution.

Let q be the maximum position that is a prefix of $\mathtt{hp}(C_1)$ and $\mathtt{hp}(C_2)$. If $|q| = |\mathtt{hp}(C_1)| = |\mathtt{hp}(C_2)|$, then $\mathtt{hp}(C_1) = \mathtt{hp}(C_2)$. In this case, the incompatibility of C_1 and C_2 implies $\mathtt{sp}(C_1) \neq \mathtt{sp}(C_2)$, from which we conclude that $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$ has no solution. Hence, assume that $q < \mathtt{hp}(C_1)$ and $q < \mathtt{hp}(C_2)$. If some $q' \leq q$ satisfies $\mathtt{root}(C_1|_{q'}) \neq \mathtt{root}(C_2|_{q'})$, then $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$ has no solution again. Therefore, assume $\mathtt{root}(C_1|_{q'}) = \mathtt{root}(C_2|_{q'})$ for all $q' \leq q$. The equation $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$ has solvability of $\mathtt{rot}(C_1, |q|)^{k_1-|q|}(t_1) \doteq \mathtt{rot}(C_2, |q|)^{k_2-|q|}(t_2)$ as a necessary condition. We write $\mathtt{rot}(C_1, |q|)$ and $\mathtt{rot}(C_2, |q|)$ more explicitly of the form $f(x_1, \ldots, x_{i-1}, D_1[\bullet], x_{i+1}, \ldots, x_m)$ and $f(y_1, \ldots, y_{j-1}, D_2[\bullet], y_{j+1}, \ldots, y_m)$, respectively. Note that, by the maximality of q, the indexes i and j are different. Any solution σ of $C_1^{k_1}(t_1) \doteq C_2^{k_2}(t_2)$, and hence of $\mathtt{rot}(C_1, |q|)^{k_1-|q|}(t_1) \doteq \mathtt{rot}(C_2, |q|)^{k_2-q}(t_2)$, makes $\sigma(y_i)$ equal to $\sigma(D_1[t_1])$. But since k_1 is greater than or equal to $2|p|$ and C_1 is a rotation of C_2, D_1 properly contains the variable y_i as well, and hence $\sigma(y_i)$ is a proper subterm of itself, a contradiction. □

Lemma 5.12. *Let Δ be a set of exponent multi-equations with $\Delta_u \neq \emptyset$, such that (cycle), (clash), (clashfC) or (clashCC) can be applied to it. Then Δ does not have any big unifier.*

Proof. Assume that Δ has a big unifier. We show then that (cycle) or (clash) cannot be applicable. From Corollary 5.9, we know that the existence of a big unifier implies that for every expression $D^{|p|N-k}(s)$, the exponent is at least $2|p|$, and hence the context is at least $D[D[\ldots[\bullet]\ldots]]$.

For all multi-equations of the form $s \doteq t \doteq M$ where s, t are rooted by function symbols, we must have $\mathtt{root}(s) = \mathtt{root}(t)$. Hence (clash) is not applicable. However, (cycle) is not applicable, either. The reason is that the existence of equations $x_1 \doteq t_1 \doteq M_1, x_2 \doteq t_2 \doteq M_2, \ldots, x_{n-1} \doteq t_{n-1} \doteq M_{n-1}, x_n \doteq t_n \doteq M_n$ in Δ such that x_2 occurs in $t_1, \ldots, x_n$ occurs in t_{n-1}, and x_1 occurs in t_n together with the existence of a big unifier would imply that $\sigma(x_1)$ is a proper subterm of $\sigma(x_1)$, which is impossible. If there is a multi-equation of the form $C_1^{|p|N-k_1}(x_1) \doteq C_2^{|p|N-k_2}(x_2) \doteq M$, then by Lemma 5.11 C_1, C_2 are compatible, hence (clashCC) is also not applicable. If there is a multi-equation of the form $f(\ldots) \doteq (g(\ldots, D[\bullet], \ldots))^{|p|N-k}(x) \doteq M$, then the root of $\sigma(g(\ldots, D[\bullet], \ldots))^{|p|N-k}(x)$ is $g = f$ for every big unifier σ, hence (clashfC) is not applicable. From this contradiction, we conclude that Δ cannot have a big unifier. □

Lemma 5.13 (*Completeness for Big Unifiers*). *Let $\Delta_1 \vdash \Delta_2$ be an inference step, and let σ be a big unifier of Δ_1. Then there is an extension σ_1 of σ that is also a big unifier of Δ_2.*

Proof. For the case of rules (xx), (xy), (xM), (1-alone), (ff), and (Nf) the same σ serves as a big unifier.

For the case of rule (NN), the same σ serves for $\Delta \cup \{C_2^{|p|N-k_2}(x_2) \doteq M\}$. It also serves for $\mathtt{expand}(C_1, C_2)$, because, by Corollary 5.9, the replacement of N by $\sigma(N)$ makes $|p|N - k_1$ greater than or equal to $2p$, and hence, solvability of $\mathtt{expand}(C_1, C_2)$ is a necessary condition for solvability of Δ_1 by Lemma 5.11. But σ is not enough for $\mathtt{flatten}(\{x_1 \doteq (\mathtt{rot}(C_2, k_1'))^{k_1-k_2}(x_2)\})$, since it contains new variables. We just need to extend σ in the following way. Whenever the occurrences of a term t are replaced by a new variable z along the flattening process, we extend σ by $\{z \mapsto \sigma(t)\}$. The final extension of σ after the complete flattening process is a big unifier of the resulting set of multi-equations. Finally, note that by Lemma 5.12, the remaining rules, (cycle), (clash), (clashfC) and (clashCC), are not applicable, and this completes the proof. □

Combining the soundness and completeness results for big unifiers, we conclude that the inference rules can be applied in a "don't care" manner.

Corollary 5.14. *If $\Delta_1 \vdash \Delta_2$ then every big unifier σ of Δ_1 can be extended to a big unifier of Δ_2, and every big unifier of Δ_2 is a big unifier of Δ_1.*

We finally show that the inference system is progressive, that is, if there is a solution and $\Delta_u \neq \emptyset$, then we can apply some rule.

Proposition 5.15. *Let Δ_0 be a set of initial exponent multi-equations. Then the following are equivalent:*
(A) *there is a* PHASE2 *derivation $\Delta_0 \vdash \Delta_1 \vdash \cdots \vdash \Delta_n$ with $\Delta_{n,u} = \emptyset$,*
(B) *Δ_0 has a big unifier, and*
(C) *for all $n \geq B(\Delta_0)$, Δ_0 has a solution σ with $\sigma(N) = n$.*

Proof. (A)⇒(C): If $\Delta_0 \vdash \Delta_1 \vdash \cdots \vdash \Delta_n$ is a maximal derivation with $\Delta_{n,u} = \emptyset$, then the final Δ_n has a solution: $\Delta_{n,s}$ can be arranged in the form $x_1 \doteq t_1, x_2 \doteq t_2, \ldots, x_m \doteq t_m$, such that all x_i are different and such that x_i is not contained in any t_j with $j \geq i$. Hence a solution can be computed by iterated instantiation. Since none of our inference rules instantiate N, we can set $\sigma(N) := n'$ for any $n' \geq B(\Delta_0)$. By Lemma 5.10, each of these solutions for Δ_n are also solutions for Δ_0.
(C)⇒(B): This is obvious.
(B)⇒(A): If Δ_0 has a big unifier, then let $\Delta_0 \vdash \Delta_1 \vdash \cdots \vdash \Delta_n$ be a maximal derivation. By Lemma 5.13, Δ_n has a big unifier. The assumption that (clash), (clashfC) and (clashCC) are not applicable implies that for any $s \doteq t \in \Delta_{n,u}$ such that s, t are not variables, one of the rules (ff), (NN), (Nf) could be applied. Since these rules are not applicable, this implies that multi-equations in $\Delta_{n,u}$ must be of the form $x \doteq t$, and the variable x occurs in t or somewhere else in $\Delta_{n,u}$. Since (cycle) is not applicable, this is in conflict with the non-applicability of the rule (xM). Hence the final set $\Delta_{n,u}$ must be empty. □

Complexity

An important consequence of Proposition 5.15 is the following corollary.

Corollary 5.16. *A flattened set of initial exponent multi-equations Δ_0 has a big unifier if and only if the first-order problem $\{N \mapsto \lceil B(\Delta_0)\rceil\}(\Delta_0)$ is unifiable.*

Proof. This follows from Proposition 5.15. □

Therefore, for deciding the existence of big unifiers of a given Δ_0, it suffices to ask for the solution of the first-order unification problem $\{N \mapsto \lceil(\mathtt{nf})(\Delta_0)/p + 2\rceil\}\Delta_0$, which can be solved efficiently. Hence, for deciding the existence of any (big or small) solution we just need to consider several first-order unification problems obtained by substituting N by $0, 1, 2, \ldots, \lceil(\mathtt{nf})(\Delta_0)/p + 2\rceil$.

Theorem 5.17. *The unification problem for flattened exponent multi-equations is solvable in $\mathtt{O}(n^3\mathtt{log}(n))$ time.*

Proof. By iteratively instantiating N by $0, 1, \ldots, \lceil B(\Delta_0)\rceil$, and expanding the exponents in each case, we get $O(n)$ first-order unification problems of size $O(n^2)$ each. Since first-order unifiability of multi-equations of size n is decidable in $O(n * \mathtt{log}(n))$ time (Martelli and Montanari, 1982; Paterson and Wegman, 1978), we get the desired result. □

Combining Theorem 5.17 and the NP-process of Phase 1 gives the following.

Theorem 5.18. *One context unification is in NP.*

From the first phase, we know that any set of equations of one context unification containing an equation of the form $F(s) = C[F(t)]$ can be transformed into a unification problem of exponent flattened multi-equations by guessing just *once* over a *linear* number of possibilities. Hence, we obtain the following.

Theorem 5.19. *The unification problem for one context term equations containing an equation of the form $F(s) = C[F(t)]$ is solvable in polynomial time.*

We can also construct a complete set of unifiers for Δ_0 following a similar approach. In the second phase, for unifiers σ where $\sigma(N) < B(\Delta_0)$, we simply solve the term unification problems obtained by setting $\langle N \mapsto n\rangle$ for every $n < B(\Delta_0)$; and for unifiers σ where $\sigma(N) \geq B(\Delta_0)$, we use the PHASE2 inference system to compute Δ_n. By Lemma 5.8, this runs in polynomial time. If $\Delta_{n,u} = \emptyset$, then $\Delta_{n,s}$ represents exactly *all* big unifiers of Δ_0 since, by Corollary 5.14, we know that solutions are not lost. We get the following analogues of Theorems 5.17 and 5.18.

Theorem 5.20. *Let Δ be a set of initial exponent equations. Then, a complete set of unifiers for Δ with polynomially many unifiers (representable in polynomial space) can be generated in polynomial time.*

Theorem 5.21. *Given a one context set of equations Δ, a complete set of most general unifiers for Δ can be generated in exponential time. Each unifier is represented in polynomial space using mappings from variables to exponent terms.*

We will give a polynomial complexity bound, if the number of first-order variables is fixed. Although all the claims and proofs we have given above are valid by assuming that the input is already represented by a DAG structure, the following result requires the input to be given without any compression, i.e. the terms are represented with a size proportional to the size of such terms (as trees).

Theorem 5.22. *If the number of first-order variables is fixed (say k), then one context unification is solvable in polynomial time.*

Proof. Assume for the purposes of this proof that we implement phase 1 without DAGS, i.e. no reuse of equal subterms occurs. Then, an assignment for a variable produces as many copies of the assigned term as the number of occurrences of this variable in the equations. If the number k of first-order variables is fixed, then the size increase of the equations and terms by the algorithm in phase 1 without DAGs remains polynomial: Instantiation by rules `Var-Elim` and `Var-Elim2` does at most square the size, which happens at most $k+1$ times. That is, if n is the size of the input, then the size (as terms without using DAGs) of the equations and substitutions during the whole phase 1 is at most $n^{2^{k+1}}$. Now it is sufficient to argue that there is a polynomial upper bound for the maximal total number of necessary guessing possibilities in phase 1, since we have already established polynomiality of phase 2, and since phase 1 computation is in NP. The rules `Decompose` and `Var-Elim` are don't care non-deterministic and thus it is not necessary to explore alternatives. The possibilities of the rules `Var-Elim2`, `CVar-Elim`, and `CVar-Elim2` have to be counted. They are applied at most $k+1$ times. Every rule has at most $\left(n^{2^{k+1}}\right)^2$ possibilities, due to the selection of the context C. Thus we have shown that there is a polynomial number of possibilities, and every possibility can be completely checked including phase 2 in polynomial time. In summary, we have shown the claim of the theorem. □

6. Redundancy testing

Recall that the procedure for doing interprocedural program analysis, outlined in Section 1.2, is guaranteed to terminate and compute finite summaries if every sequence of non-redundant equations is finite. This motivates the need to solve the following two problems related to redundancy testing:
Redundancy of an equation: Given a set Δ of equations and an equation $s \doteq t \in \Delta$, check whether every unifier σ of $\Delta \setminus \{s \doteq t\}$ is also a unifier of the equation $s \doteq t$.
Finite redundancy property: Is there an infinite sequence $s_i \doteq t_i, i = 1, 2, \ldots$ of equations such that only a fixed finite set of variables and one context variable F appear in the terms s_i, t_i, and such that for every $n \in \mathbb{N}$, the equation $s_n \doteq t_n$ is not redundant in the set $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$? If there is no such infinite sequence, then we say that the finite redundancy property holds.

Theorem 6.1. *For the one context unification problem, checking redundancy of equations is in coNP.*

Proof. An equation $s \doteq t \in \Delta$ is non-redundant for Δ, if there is a unifier σ of $\Delta \setminus \{s \doteq t\}$, such that $\sigma(s) \neq \sigma(t)$. This is equivalent to the condition that there is a most general unifier σ' of $\Delta \setminus \{s \doteq t\}$, such that $\sigma'(s) \neq \sigma'(t)$. Since we can compute all most general unifiers using an NP-algorithm, we can guess the most general unifier σ' in polynomial time. We need to argue that checking $\sigma'(s) \neq \sigma'(t)$ can be done in polynomial time. We only have to be careful in treating the instantiation into the integer variable N. In the case that $N \mapsto n$ where n is a small positive integer ($n \leq B$), we can expand the expression, which causes only a polynomial size increase. If the unifier does not instantiate N, then Corollary 5.16 and Proposition 5.15 show that it is sufficient to check the instantiation $N \mapsto \lceil B(\Delta) + 1 \rceil$, which leads only to a polynomial blow-up. Since we can check the equality of expressions by checking their syntactic equality, which again can be done in polynomial time using standard methods, all operations can be done in polynomial time. Hence the non-redundancy is in NP, and redundancy is in coNP. □

Now we show that the finite redundancy property holds. Intuitively, finite redundancy will hold if, whenever we add a non-redundant equation $s_n \doteq t_n$ to $\Delta_{n-1} := \{s_1 \doteq t_1, \ldots, s_{n-1} \doteq t_{n-1}\}$ and compute a most general unifier σ_n of $\sigma_{n-1}(\{s_n \doteq t_n\} \cup \Delta_{n-1})$, then we are always forced to instantiate either a first-order variable, or N. The tricky case occurs when this does not happen, that is, when

$$\text{NC1}\quad \frac{f(s_1,\dots,C_1^{a|p|N-k}(r),\dots,s_n)}{C_2^{a|p|N-(k-1)}(r)} \qquad \text{if } C_2 = \mathtt{rot}(C_1,|p|-1) = f(s_1,\dots,C',\dots,s_n) \text{ for some context } C'$$

$$\text{NC2}\quad \frac{C_1^{a|p|N-k_1}(C_2^{b|p|N-k_2}(s))}{C_1^{(a+b)|p|N-(k_1+k_2)}(s)} \qquad \text{if } C_2 = \mathtt{rot}(C_1,k_1') \text{ where } k_1' = |p| - k_1 \bmod |p|$$

$$\text{NC3}\quad \frac{C_1^{a|p|N-k}(f(s_1,\dots,s_n))}{C_1^{a|p|N-(k-1)}(s_i)} \qquad \text{if } f(s_1,\dots,s_{i-1},C',s_{i+1},\dots,s_n) = \mathtt{rot}(C_1,k') \text{ for some context } C' \text{ where } k' = |p| - k \bmod |p|$$

Fig. 6. Normalization rules for EEE-terms.

$\sigma_{n-1}(\{s_n \doteq t_n\} \cup \Delta_{n-1})$ is unifiable with a big unifier, but the big unifier does not instantiate any first-order variable.

We first show, in Proposition 6.3, that exponent equations that are unifiable with a big unifier, but without requiring instantiation of any first-order variable, are already solved. Using this result, we then prove the finite redundancy property in Theorem 6.4.

Consider the exponent equation $(fgf)^{3N-2}(ga) \doteq f(gff)^{3N-2}(a)$, where f and g are unary function symbols. Every substitution $N \mapsto n$, where $n \geq 1$, is a unifier of this equation. For example, when $N \mapsto 1$, then $(fgf)^{3N-2}(ga) = (fgf)^1(ga) = fga$ and $f(gff)^{3N-2}(a) = f(gff)^1(a) = fga$. We wish to argue that the equation $(fgf)^{3N-2}(ga) \doteq f(gff)^{3N-2}(a)$ is already solved (i.e., redundant) – just as $fg(a) \doteq fg(a)$ is redundant. Unfortunately, the exponent terms $(fgf)^{3N-2}(ga)$ and $f(gff)^{3N-2}(a)$ are not *syntactically* equal. However, these two terms can be made syntactically equal if they are both normalized by the rules in Fig. 6. Using rule (NC3) the term $(fgf)^{3N-2}(ga)$ can be rewritten as $(fgf)^{3N-1}(a)$ and using rule (NC1) the term $f(gff)^{3N-2}(a)$ can also be rewritten into the same term $(fgf)^{3N-1}(a)$.

An EEE-term (extended exponent expression term) is defined as a term that contains no first-order variable, no context variable, but it can contain unary exponent expressions of the form $D^{a|p|N-k}$ where a is a positive integer, $|p| = |\mathtt{hp}(D)|$, N is the unique integer variable, and k is an integer. Given terms s and t, if every substitution $\beta = \{N \mapsto b\}$, where $b \geq k/(a|p|)$ for all exponents $a|p|N-k$ in s or t, is a solution for $s \doteq t$, then we denote this as $s \approx t$. We let $NFC(s)$ denote the normal form of s obtained using the rules NC1, NC2 and NC3 from Fig. 6 exhaustively, where application to subexpressions is also permitted.

Lemma 6.2. *If s rewrites to s' by normalization rules from Fig. 6, then $s \approx s'$. Hence, we also have $s \approx NFC(s)$.*

Proposition 6.3 (*0-1-Property of Exponent Equations*). *Let $\Delta := \{s \doteq t\}$, where s, t are EEE-terms that are normalized using the rules* NC1, NC2, *and* NC3. *If the equation $s \doteq t$ has a big unifier $\gamma := \{N \mapsto n_0\}$ with $n_0 \geq \mathtt{nf}(\Delta) + \mathtt{nk}(\Delta) + 2$, then s and t are syntactically equal.*

Proof. We show by induction that $s = t$. For equation $s \doteq t$, the induction measure is a lexicographic combination of (i) the sum of all a that occur in exponents $a|p|N-k$ in s and t, (ii) the size of the terms s and t, not counting the exponents, and (iii) the sum of $|k|$ for all k occurring in exponents $a|p|N-k$ in s and t. We consider the following cases based on the form of $s \doteq t$:

$f(s_1,\dots,s_n) \doteq f(t_1,\dots,t_n)$: In this case, for each i, the terms s_i and t_i are normalized and the measure of equation $s_i \doteq t_i$ is smaller than the measure of $s \doteq t$. Hence, by induction hypothesis, we infer that for every i, it holds that $s_i = t_i$. It follows that $f(s_1,\dots,s_n) = f(t_1,\dots,t_n)$.

$f(s_1,\dots,s_n) = C_1^{a|p|N-k}(t_0)$: Wlog let $C_1 = f(t_1,\dots,t_{i-1},C_1',t_{i+1},\dots,t_n)$. For $j \neq i$, we can apply induction hypothesis to $s_j \doteq t_j$ and conclude that $s_j = t_j$. Next consider $s_i \doteq \mathtt{rot}(C_1,1)^{a|p|N-k-1}(t_0)$. Induction hypothesis can again be applied, since the number of occurrences of function symbols is smaller here than in $s \doteq t$ and the existence of a big unifier of $s \doteq t$ implies that there is also a big unifier of $s_i \doteq \mathtt{rot}(C,1)^{a|p|N-k-1}$, using Corollary 5.16. Using induction hypothesis, we get $s_i = \mathtt{rot}(C_1,1)^{a|p|N-k-1}(t_0)$. But if this is

the case, then the rule NC1 can be applied to $f(s_1, \ldots, s_n)$, contradicting the assumption that normalization rules are not applicable on s.

$C_1^{a_1|p|N-k_1}(s_1) \doteq C_2^{a_2|p|N-k_2}(s_2)$: By using the induction hypothesis on all the subterms occurring at corresponding positions in the contexts C_1, C_2, as in the previous item and Lemma 5.11, we get that $C_1 = C_2$. If $1 < a_1$ and $1 < a_2$, then we can apply induction hypothesis on $C_1^{(a_1-1)|p|N-k_1}(s_1) \doteq C_2^{(a_2-1)|p|N-k_2}(s_2)$, and conclude that s and t are syntactically equal. If $1 = a_1 < a_2$, then we can apply induction hypothesis to $s_1 \doteq C_3^{(a_2-1)|p|N-k_2+k_1}(s_2)$, where $C_3 = \mathtt{rot}(C_2, k_1')$ and $k_1' = |p| - k_1 \bmod |p|$, and conclude that $s_1 = C_3^{(a_2-1)|p|N-k_2+k_1}(s_2)$. But then the rule NC2 would be applicable on s, which leads to a contradiction. The case $1 = a_2 < a_1$ is symmetric. If $1 = a_1 = a_2$ and $k_1 = k_2$, then we can apply induction hypothesis to $s_1 \doteq s_2$ and conclude that $s_1 = s_2$ and hence $s = t$. If $1 = a_1 = a_2$ and $k_1 \neq k_2$, then again induction hypothesis can be used on the equation obtained by stripping away one exponent expression and using a rotation on the other one. Since there are several very similar cases, we only consider the one where $k_1 > k_2 \geq 0$. We apply induction hypothesis to $s_1 \doteq C_3^{k_1-k_2}(s_2)$ where $C_3 = \mathtt{rot}(C_2, k_1')$ and $k_1' = |p| - k_1 \bmod |p|$ and conclude that $s_1 = C_3^{k_1-k_2}(s_2)$. This is not possible, since otherwise rule NC3 would apply on s, which is a contradiction.

In each case, we conclude that s and t have to be syntactically equal. □

Theorem 6.4. *The finite redundancy property holds for one context unification.*

Proof. Let V be a finite set of first-order variables and let $s_i \doteq t_i, i = 1, 2, \ldots$ be a sequence of equations that contain only variables from V and perhaps a context variable F. Furthermore, we assume that for all n: $s_n \doteq t_n$ is not redundant in $\Delta_n := \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$. We show that the sequence must be finite.

We build a sequence of multisets $M_i, i = 1, 2, \ldots$, of unifiers where the multiset M_i contains unifiers of Δ_i as follows:

$$M_0 = \{Id\}$$
$$M_{i+1} = \bigcup_{\substack{\sigma \in M_i \\ \sigma(s_{i+1}) \not\approx \sigma(t_{i+1})}} \mathit{Unifiers}(\{\sigma(s_{i+1} \doteq t_{i+1})\}) \circ \sigma \quad \cup \bigcup_{\substack{\sigma \in M_i \\ \sigma(s_{i+1}) \approx \sigma(t_{i+1})}} \{\sigma\}$$

where $\mathit{Unifiers}(\Delta)$ is the complete set of unifiers computed using the PHASE1 and PHASE2 inference rules (if Δ contains F) or only the PHASE2 rules (if Δ does not contain F). We will show that there is a well-founded ordering $\succ$ on these multisets such that $M_0 \succ M_1 \succ \cdots$, which proves the finite redundancy property.

Let $\mathtt{inst}(\sigma, S) := \{x \in S \mid \sigma(x) \neq x\}$ be the subset of variables from S that are instantiated by the substitution σ. We say $\sigma \succ \sigma'$ if either (i) $\mathtt{inst}(\sigma, V) \subsetneq \mathtt{inst}(\sigma', V)$, or (ii) $\mathtt{inst}(\sigma, V) = \mathtt{inst}(\sigma', V)$, and $\mathtt{inst}(\sigma, \{F\}) \subsetneq \mathtt{inst}(\sigma', \{F\})$, or (iii) $\mathtt{inst}(\sigma, V) = \mathtt{inst}(\sigma', V)$, $\mathtt{inst}(\sigma, \{F\}) = \mathtt{inst}(\sigma', \{F\})$, and $\mathtt{inst}(\sigma, \{N\}) \subsetneq \mathtt{inst}(\sigma', \{N\})$. Finally, for multisets M, M', we say $M \succ M'$ if M is greater than M' in the multiset extension of the ordering $\succ$ on substitutions (see Dershowitz and Manna (1979)). Note that if $M_i = \emptyset$, i.e. there is no unifier for $s_1 \doteq t_1, \ldots, s_i \doteq t_i$, then the sequence is terminated, since $s_{i+1} \doteq t_{i+1}$ will always be redundant. We now show that $M_i \succ M_{i+1}$. We first show that $M_i \succeq M_{i+1}$. Consider any $\sigma \in M_i$ and consider the set $S := \mathit{Unifiers}(\{\sigma(s_{i+1} \doteq t_{i+1})\}) \circ \sigma$ or $S := \{\sigma\}$ of substitutions in M_{i+1} generated from σ. We argue that $M_i \succeq M_{i+1}$ by showing that $\{\sigma\} \succeq S$. We have the following cases:

(1) $S = \{\sigma\}$: In this case, clearly $\{\sigma\} \succeq S$.
(2) $S = \emptyset$: In this case, clearly $\{\sigma\} \succ S$.
(3) $\{\sigma(s_{i+1} \doteq t_{i+1})\}$ contains F: Since the equation contains F, we use the PHASE1 system to compute S. Since we assume that cases (1) and (2) do not apply, the equation $\sigma(s_{i+1} \doteq t_{i+1})$ is unifiable and it is not trivial, and hence, at least one of the rules `Var-Elim` or `Cvar-Elim` from PHASE1 inference system must have been applied. Hence, each member of S either eliminates at least one first-order variable from V, or removes F and introduces N. This implies $\{\sigma\} \succ S$.

(4) $\{\sigma(s_{i+1} \doteq t_{i+1})\}$ does not contain F, but contains N: There are different cases for a unifier σ' in $\mathit{Unifiers}(\{\sigma(s_{i+1} \doteq t_{i+1})\})$. If N is guessed as a small number, then clearly $\{\sigma\} \succ \{\sigma'\}$. If σ' instantiates a first-order variable from V, then again $\{\sigma\} \succ \{\sigma'\}$, because the intermediate first-order variables introduced by flattening do not increase the measure — every new first-order variable x introduced by flattening comes with an equation $x = s$, where s is not a variable, and hence is instantiated. The remaining case is when σ' does not instantiate any first-order variable in $\sigma(s_{i+1} \doteq t_{i+1})$ and when it is a big unifier of $\sigma(s_{i+1} \doteq t_{i+1})$. Now we can treat all the first-order variables from V as constants (by simply instantiating them with fresh constants) and we can apply Proposition 6.3, which tells us that $\sigma(s_{i+1} \doteq t_{i+1})$ is solved by the identity unifier. But this has been considered in Case (1).

Since in every case, $\{\sigma\} \succeq S$ and since M_{i+1} is a union of all such S's, we have $M_i \succeq M_{i+1}$. Moreover, since we have assumed that the sequence of equations is such that $s_{i+1} \doteq t_{i+1}$ is non-redundant for Δ_i, there is at least one σ in M_i, such that $\sigma(s_{i+1}) \not\approx \sigma(t_{i+1})$. Hence, there is at least one σ in M_i that is replaced by a strictly smaller multiset of unifiers in M_{i+1}. This shows that $M_i \succ M_{i+1}$. □

7. Conclusion

This paper shows that one context unification, that is context unification over a single context variable, is in NP. The algorithm is presented using inference rules in the style of standard term unification algorithms. Contexts with exponents of the form $|p|N - k$ are used to efficiently represent large terms that may be generated in the process of solving a one context unification problem. This gives upper complexity bounds for the redundancy problem that is required for solving interprocedural program analysis problems.

We leave open the issue of a lower complexity bound of one context unification, and also complexity considerations for the finite redundancy property, e.g. giving upper bounds for the maximal length of a non-redundant sequence of terms, depending on the number of first-order variables and a size bound of the possible sequences.

Acknowledgements

We thank the anonymous referees of the ADDCT-JSC for their comments and hints, which greatly helped to improve the paper.

The work was supported by Spanish Min. of Educ. and Science by the FORMALISM project (TIN2007-66523) and by the LOGICTOOLS-2 project (TIN2007-68093-C02-01).

References

Baader, F., Snyder, W., 2001. Unification theory. In: Robinson, A., Voronkov, A. (Eds.), Handbook of Automated Reasoning, vol. I. Elsevier Science, pp. 445–532 (Chapter 8).

Busatto, G., Lohrey, M., Maneth, S., 2008. Efficient memory representation of XML document trees. Inf. Syst. 33 (4), 456–474.

Comon, H., 1995. On unification of terms with integer exponents. Math. Syst. Theory 28 (1), 67–88.

Dabrowski, R., Plandowski, W., 2002. On word equations in one variable. In: MFCS 2002. In: LNCS, vol. 1103. Springer-Verlag, pp. 212–220.

Dershowitz, N., Manna, Z., 1979. Proving termination with multiset orderings. Commun. ACM 22, 465–476.

Gulwani, S., Necula, G.C., 2004. A polynomial-time algorithm for global value numbering. In: Static Analysis Symposium. In: LNCS, vol. 3148. pp. 212–227.

Gulwani, S., Tiwari, A., 2007a. Assertion checking unified. In: Proc. Verification, Model Checking and Abstract Interpretation, VMCAI 2007. In: LNCS, vol. 4349. Springer, pp. 363–377.

Gulwani, S., Tiwari, A., 2007b. Computing procedure summaries for interprocedural analysis. In: Proc. European Symp. on Programming, ESOP 2007. In: LNCS, vol. 4421. Springer, pp. 253–267.

Karr, M., 1976. Affine relationships among variables of a program. Acta Inform. 6, 133–151.

Lassez, J.-L., Maher, M.J., Marriott, K., 1988. Unification revisited. In: Minker, J. (Ed.), Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, pp. 587–625.

Levy, J., 1996. Linear second-order unification. In: Proc. 7th Intl. Conf. on Rewriting Techniques and Applications, RTA'96. In: LNCS, vol. 1103. pp. 332–346.

Levy, J., Niehren, J., Villaret, M., 2005. Well-nested context unification. In: CADE 2005. In: LNCS, vol. 3632. pp. 149–163.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2006. Stratified context unification is NP-complete. In: Proc. Third Intl. Joint Conf. on Automated Reasoning, IJCAR 2006. In: LNCS, vol. 4130. Springer, pp. 82–96.

Makanin, G., 1977. The problem of solvability of equations in a free semigroup. Math. USSR Sbornik 32 (2), 129–198.
Martelli, A., Montanari, U., 1982. An efficient unification algorithm. ACM Trans. Program. Lang. Syst. 4 (2), 258–282.
Müller-Olm, M., Rüthing, O., Seidl, H., 2005. Checking Herbrand equalities and beyond. In: VMCAI. In: LNCS, vol. 3385. Springer, pp. 79–96.
Müller-Olm, M., Seidl, H., 2004. A note on Karr's algorithm. In: 31st International Colloquium on Automata, Languages and Programming, pp. 1016–1028.
Niehren, J., Pinkal, M., Ruhrberg, P., 1997a. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In: CADE-14. In: LNCS, vol. 1249. pp. 34–48.
Niehren, J., Pinkal, M., Ruhrberg, P., 1997b. A uniform approach to underspecification and parallelism. In: 35th ACL'97, Madrid, pp. 410–417.
Paterson, M.S., Wegman, M.N., 1978. Linear unification. J. Comput. System Sci. 16, 158–167.
Schmidt-Schauß, M., 2002. A decision algorithm for stratified context unification. J. Logic Comput. 12 (6), 929–953.
Schmidt-Schauß, M., Schulz, K.U., 1998. On the exponent of periodicity of minimal solutions of context equations. In: Proc. RTA. In: LNCS, vol. 1379. Springer-Verlag, pp. 61–75.
Schmidt-Schauß, M., Schulz, K.U., 2002. Solvability of context equations with two context variables is decidable. J. Symbolic Comput. 33 (1), 77–122.