AAECC
DOI 10.1007/s00200-009-0106-4

Simplifying the signature in second-order unification

Jordi Levy · Mateu Villaret

Received: 14 May 2008 / Revised: 13 March 2009

Abstract Second-Order Unification is undecidable even for very specialized fragments. The signature plays a crucial role in these fragments. If all symbols are monadic, then the problem is NP-complete, whereas it is enough to have just one binary constant to lose decidability. In this work we reduce Second-Order Unification to Second-Order Unification with a signature that contains just one binary function symbol and constants. The reduction is based on partially currying the equations by using the binary function symbol for explicit application @. Our work simplifies the study of Second-Order Unification and some of its variants, like Context Unification and Bounded Second-Order Unification.

Keywords Second-Order Unification · Context Unification · Lambda calculus

Mathematics Subject Classification (2000) 03B40 · 03B15 · 68N18 · 03F03

This research has been partially supported by the research projects Mulog-2 (TIN2007-68005-C04-01) and SuRoS (TIN2008-04547) funded by the CICyT.

J. Levy (✉)
IIIA, CSIC, Campus de la UAB, Barcelona, Spain
e-mail: levy@iiia.csic.es
URL: http://www.iiia.csic.es/∼levy

M. Villaret
IMA, UdG, Campus de Montilivi, Girona, Spain
e-mail: villaret@ima.udg.edu
URL: http://ima.udg.edu/∼villaret

Published online: 04 September 2009

1 Introduction

Second-Order Unification is undecidable even for very specialized fragments [11, 17,22]. The signature plays a crucial rule in this problem: Goldfarb's undecidability proof for Second-Order Unification [9,11] requires the use of a binary function symbol, whereas the first known decidable fragment of Second-Order Unification was Monadic Second-Order Unification, where function symbols can be at most unary [8,13,19,32]. In this work we reduce Second-Order Unification to Second-Order Unification with a signature that contains constants (0-ary function symbols) and just one binary function symbol. The reduction is based on partially currying terms.

Currying is usually defined as the encoding of n-ary application into unary application. For instance, the curry form of a term like $f(a, b)$ is $(f(a))(b)$, that in λ-calculus is usually written as $(f\,a)\,b$. We go a step further by making application explicit. Thus, the curry form of the term $f(a, b)$ is $@(@(f, a), b)$, where $@$ denotes the explicit application, hence a binary function symbol. When dealing with First-Order Unification, this transformation reduces a unification equation to another one containing only one binary symbol. The sizes of the new equation, and of the unifier, are linear in the sizes of the original equation and original unifier. Therefore, from the point of view of complexity there is not a significant difference, but in practical implementations this implies that terms can be represented as binary trees, and contexts as subterms. This transformation has been shown useful in term indexing data structures [10].

When dealing with second-order terms, where variables can have arguments, the transformation is not so obvious. We can curryfy constant symbol applications and second-order variable applications, obtaining a first-order term. For instance, for $f(F(a), Y) \stackrel{?}{=} f(a, b)$, where F is a second-order variable and Y a first-order one, we obtain $@(@(f', @(F', a)), Y) \stackrel{?}{=} @(@(f', a), b)$, where now F' and Y are both first-order typed. However, solvability of unification equations is not preserved by this transformation. In our example, the original equation has a solution $\sigma = [F \mapsto \lambda x.x, Y \mapsto b]$, whereas its Curry form is unsolvable.

Another alternative is reducing Second-Order Unification to First-Order Unification modulo β-equivalence. In our example, the application of σ to the Curry form of the equation results in $@(@(f', @(\lambda x.x, a)), b)$ and $@(@(f', a), b)$, which are equivalent modulo the β-equivalence $@(\lambda x.t_1, t_2) = t_1[x \mapsto t_2]$. However, the right-hand side of β-equivalence is a meta-term. This means that this is not an instance of E-Unification, where E is an algebraic theory.

A third alternative is adding all equations of explicit substitution [1], obtaining an algebraic theory, and doing First-Order Unification modulo this theory. Roughly speaking this is what is done in the so called Explicit Unification [3,7].

Here, we propose to curryfy function symbol applications, but not variable applications. Therefore, the new equation we get is also second-order typed. For instance, for $F(G(a), b) \stackrel{?}{=} g(a)$, we get $F(G(a), b) \stackrel{?}{=} @(g, a)$, that is also solvable. With this *partial curryfication* we do not reduce the *order* of the unification equation (we reduce Second-Order Unification to a fragment of Second-Order Unification), but we reduce the number of function symbols to just one: the

application symbol @. This simplification in the signature is very interesting when studying the un/decidability of Second-Order Unification and related problems.

The partial curryfication is a linear transformation. We prove that, when variables *do not touch*, i.e. when terms do not have any variable just above another variable, then the transformation preserves solvability of second-order equations. Hence, we can P-reduce Second-Order Unification where equations satisfy this *"no-touch"-condition* to Second-Order Unification with just one binary function symbol. In a second step, we also prove that general Second-Order Unification can be NP-reduced to Second-Order Unification where all equations are non-touching. Basically, the idea of this second reduction is to guess a function symbol to *separate* touching variables. Therefore, the composition of both reductions results into an NP-reduction that allows us to simplify signatures, preserving decidability. With respect to complexity, since the resulting reduction is NP, it can serve to investigate the complexity of the problems using a simplified signature, whenever they are NP-hard.

Although Second-Order Unification was already known to be undecidable for just one binary function symbol [9,11], applying the reduction described in this paper, to the results of [22], we prove that Second-Order Unification is undecidable for equations with just one binary function symbol and one second-order variable occurring four times.

The reduction also works for Context Unification [5,24,31] and for Bounded Second-Order Unification [21,29]. Context Unification is a variant of Second-Order Unification. Its decidability is still unknown. Roughly speaking the variant consists of constraining unifiers to instantiate second-order variables by linear terms. The problem has also several decidable fragments [16,20,28,30]. One of them is Word Unification [12,25,27] where function symbols are at most unary.

Bounded Second-Order Unification [21,29] is a decidable variant of Second-Order Unification. It is like Second-Order Unification where possible instantiations of second-order variables are constrained by limiting the number of occurrences of bound variables. In [29] it is shown that Bounded Second-Order Unification can be NP-reduced to the particular case where the limiting number is one and the arity of second-order variables is also one. Again we show that only one binary function symbol is required to study Context Unification and Bounded Second-Order Unification. This allows researchers to concentrate on a simplified version of these problems, and possibly of other further variants.

Curryfication has already been useful to prove decidability of Sequence Unification [14,15], an extension of First-Order Unification for unranked terms, where sequence variables can be instantiated by sequences of terms.

Finally, the study of currying in Second-Order Unification problems could be also of interest when considering terms with unranked signatures for information extraction of XML documents. In fact, curryfication has been shown useful in querying unranked trees and XML documents using tree automata [4,26].

This paper proceeds as follows. In Sect. 2 we introduce some assumptions and considerations of the work. Most of our results hold for Second-Order and for Context Unification, and sometimes we do not make the distinction explicit. In Sect. 3 we define the curry forms where only function symbol applications are made explicit and show the limitations of the technique. In Sect. 4 we define a labeling on curry

forms that is used to characterize "well-curried" terms, i.e. terms that are the curry form of some well-built term. In Sect. 5 we present a property of some equations that ensure that solvability is preserved by curryfication: when variables do not touch. In Sect. 6 we prove our main result: Second-Order, Context and Bounded Second-Order Unification can be reduced to a simplified form where only a single binary function symbol and unary constants are used. We conclude in Sect. 7 showing the difficulties of extending these results to higher-order problems. A preliminar version of this article was presented at the *13th International Conference on Rewriting Techniques and Applications, (RTA'02)* with the title "Currying Second-Order Unification Problems".

2 Preliminary definitions

We will use the standard notation and definitions of the *simply typed lambda calculus* [2] and *Higher-Order Unification* [6].

Consider a set of *types* built over a finite set of *base types* b, with the grammar $\tau ::= b | \tau \to \tau$. The *order* of a type is given by $\text{order}(b) = 1$ and $\text{order}(\tau_1 \to \tau_2) = \max\{\text{order}(\tau_1) + 1, \text{order}(\tau_2)\}$, with the usual convention that $\to$ is associative to the right. As in [11], we only use one base type o, and second-order types. Hence, every type is either o (of order one), or of the form $o \to o \to \cdots \to o$ (of order two), which also means that we do not allow symbols or expressions of third or a higher-order type. The order of a symbol is the order of its type. Any symbol of type $\underbrace{o \to \cdots \to o}_{n+1}$ is said to have *arity* n. Hence, the arity of a symbol determines its type and order, and we will usually specify the arity instead of the type of the symbols.

A *second-order signature* Σ is a finite disjoint union of finite sets of symbols $\Sigma = \bigcup_{n \geq 0} \Sigma_n$, where symbols $f \in \Sigma_n$ are said to be n-ary, denoted by $\text{arity}(f) = n$. We distinguish between *constants*, when their arity is 0, hence first-order typed, and *function symbols*, when their arity is greater than 0, hence second-order typed. Similarly, we define the set of variables $\mathcal{X} = \bigcup_{n \geq 0} \mathcal{X}_n$, and distinguish between first-order variables (the set $\mathcal{X}_0$) and second-order variables (the rest of variables $\bigcup_{n \geq 1} \mathcal{X}_n$). We use lambda bindings and the usual notion of bound and free variables. For simplicity, we assume that bound and free variables have distinct names, and use lower case letters like $x, y, z \ldots$ for the bound variables (they are always first-order typed) and upper case letters for free variables or unknowns. We typically use $X, Y, \ldots$ for free first-order variables, $F, G, \ldots$ for second-order variables, and Z for indistinctively first or second-order variables.

The set of well-typed *terms* $\mathcal{T}(\Sigma, \mathcal{X})$ is defined as usual in the simply typed lambda calculus. The order of a term is the order of its type. We use the compact notation $s(t_1, \ldots, t_n)$ to denote $(\ldots (s t_1) \ldots t_n)$ for a term s of arity n, and $\lambda x_1, \ldots, x_n \cdot t$ to denote $\lambda x_1, \ldots, \lambda x_n \cdot t$. When we say *normal form* we mean η-long β-reduced normal form, defined as usual. Since we do not consider third or higher-order constants, first-order typed terms in normal form do not contain λ-abstractions, and second-order typed terms only contain λ-abstractions in outermost positions. Notice that any term in normal form can be written, with the compact notation, using the syntax $t ::= a | X | f(t_1, \ldots, t_m) | F(t_1, \ldots, t_m)$, for first-order typed terms, and $s ::= \lambda x_1, \ldots, x_n \cdot t$

for second-order typed terms, where a, X, and the x_i's have arity zero and f and F have arity m. Written in normal form, the term $\lambda x_1, \ldots, x_n \cdot t$ has arity n, when t has not a lambda as outermost symbol. A term is first-order typed when it has arity zero, and second-order typed when its arity is greater than zero. We typically use $s, t, u, \ldots$ to denote indistinctively first or second-order typed terms, specifying the arity or order when it is relevant. Thus, from the first-order perspective, we extend terms by allowing second-order variables to be applied to arguments, and lambda abstractions in outermost positions.

A *second-order substitution* σ is a set of (variable,term) pairs, like $[Z_1 \mapsto t_1, \ldots, Z_n \mapsto t_n]$, where Z_i and t_i have the same arity, and t_i is said to be the *instance* of Z_i. Therefore, instances of second-order variables contain λ-abstractions. The set $\{Z_1, \ldots, Z_n\}$ is called the *domain* of the substitution, and is denoted by $Dom(\sigma)$. The application of a substitution σ to a term t in normal form is defined recursively as follows

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$$

$$\sigma(X) = \begin{cases} t & \text{if the pair } X \mapsto t \text{ is in } \sigma \\ X & \text{otherwise} \end{cases}$$

$$\sigma(F(t_1, \ldots, t_n)) = \begin{cases} \rho(u) & \text{if the pair } F \mapsto \lambda x_1 \cdots \lambda x_n \cdot u \text{ is in } \sigma \\ & \text{where } \rho = [x_1 \mapsto \sigma(t_1), \ldots, x_n \mapsto \sigma(t_n)] \\ F(\sigma(t_1), \ldots, \sigma(t_n)) & \text{otherwise} \end{cases}$$

$$\sigma(\lambda x_1, \ldots, x_n \cdot t) = \lambda x_1, \ldots, x_n \cdot \sigma(t) \qquad \text{assuming that } x_i \notin Dom(\sigma)$$

Given two substitutions σ and ρ, their composition is defined by $(\sigma \circ \rho)(t) = \sigma(\rho(t))$, for any term t, and is also a substitution, hence $Dom(\sigma \circ \rho)$ is finite. Given a set of variables V and a substitution σ, the restriction of σ to the domain V is denoted by $\sigma|_V$. Given a set of variables V, we say that a substitution σ is *more general* w.r.t. V than another substitution ρ, denoted $\sigma \preceq_V \rho$, if there exists a substitution τ such that $\rho(Z) = \tau(\sigma(Z))$, for any variable $Z \in V$, i.e. $\rho|_V = (\tau \circ \sigma)|_V$. This defines a preorder relation on substitutions.

A *Second-Order Unification problem* E is a pair of first-order terms (or *equation*), denoted by $t \stackrel{?}{=} u$. Notice that all variable occurrences of $t \stackrel{?}{=} u$ are free. A substitution σ is said to be a *unifier* (or solution) of a Second-Order Unification equation $t \stackrel{?}{=} u$ if $\sigma(t) = \sigma(u)$, i.e. if $\sigma(t)$ and $\sigma(u)$ are identical. *Most general unifiers* of E are unifiers that are minimal w.r.t. $\preceq_{FV(E)}$.

A *Context Unification problem* is also a pair of first-order terms over a second-order signature. In the area of Context Unification, second-order variables are called *context variables*. A *context substitution* is a second-order substitution $[Z_1 \mapsto t_1, \ldots, Z_n \mapsto t_n]$ where, for all second-order term $t_i = \lambda x_1 \cdots \lambda x_n \cdot u$, every x_j occurs exactly once in u, i.e. where instances of context variables must be linear. Then, a *context unifier* of a Context Unification equation $t \stackrel{?}{=} u$ is a context substitution σ satisfying $\sigma(t) = \sigma(u)$. Sometimes, Context Unification is defined restricting context variables to be unary. Here we consider n-ary variables, and use bound variables to denote the "holes" of the context. For more detailed comments and comparison between these approaches to Context Unification see [31].

A *Bounded Second-Order Unification problem* is also a pair of first-order terms over a second-order signature, and a number m. A *bounded second-order substitution* is a second-order substitution $[Z_1 \mapsto t_1, \ldots, Z_n \mapsto t_n]$ where, for all second-order term $t_i = \lambda x_1, \ldots, x_n \cdot u$, every x_j occurs at most m times in u. If the value of m is given in unary encoding, then Bounded Second-Order Unification can be NP-reduced to the particular case where $m = 1$ [29]. Therefore, we will consider only this special case.

Notice that second-order, context and bounded second-order equations have the same presentation. Moreover, any solvable context equation is a solvable bounded second-order equation, and any solvable bounded second-order equation is a solvable second-order equation.

If nothing is said, the signature of an equation is given by the set of constants that it contains and a denumerable infinite set of variables, for every arity. For technical reasons we also assume that the signature contains, at least, a binary function symbol and a constant (that can be added if the equation does not contain any).

The following is a basic property of most general second-order, context and bounded unifiers that will be required in some proofs.

Lemma 1 *Let $t \stackrel{?}{=} u$ be a second-order equation, and σ be a most general second-order [context or bounded second-order] unifier. Then, for any variable Z, $\sigma(Z)$ does not contain constants not occurring in the equation $t \stackrel{?}{=} u$.*

Proof Suppose that a most general unifier σ introduces a constant f not occurring in the equation. Then, we can replace every occurrence of this constant by a fresh variable F of the same arity. This would result into another unifier σ' that is strictly more general than σ: we have $\sigma = [F \mapsto \lambda x_1 \cdots \lambda x_n \cdot f(x_1, \ldots, x_n)] \circ \sigma'|_{FV(t \stackrel{?}{=} u)}$, but $\sigma \not\preceq_{FV(t \stackrel{?}{=} u)} \sigma'$. This would contradict the fact that σ is most general. Moreover, σ' is a context [bounded second-order] unifier, if σ is a context [bounded second-order] unifier. □

Notice that this property is true for Context Unification thanks to the fact that we allow n-ary context variables (see [31]), otherwise it would not be true.

3 Currying terms

In this section we formally define our *curryfication* transformation and characterize the terms in curry-form that are well-curried, i.e. the ones that are curried forms of some term.

Definition 1 Given a second-order signature $\Sigma = \bigcup_{n \geq 0} \Sigma_n$, the *curried signature* $\Sigma^c = \bigcup_{n \geq 0} \Sigma^c_n$ is defined by

$$\begin{aligned} \Sigma^c_0 &= \bigcup_{n \geq 0} \Sigma_n \\ \Sigma^c_2 &= \{@\} \\ \Sigma^c_n &= \emptyset \quad \text{for } n \neq 0, 2 \end{aligned}$$

The currying function $\mathcal{C} : \mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\Sigma^c, \mathcal{X})$ is defined recursively as follows:

$$\begin{gathered}\mathcal{C}(a) = a \\ \mathcal{C}(x) = x \\ \mathcal{C}(X) = X \\ \mathcal{C}(f(t_1, \ldots, t_n)) = @(\overset{n}{\cdots} @(f, \mathcal{C}(t_1)) \overset{n}{\cdots}, \mathcal{C}(t_n)) \\ \mathcal{C}(F(t_1, \ldots, t_n)) = F(\mathcal{C}(t_1), \ldots, \mathcal{C}(t_n)) \\ \mathcal{C}(\lambda x_1, \ldots, x_n \cdot t) = \lambda x_1, \ldots, x_n \cdot \mathcal{C}(t)\end{gathered}$$

for any constant $a \in \Sigma_0$, bound variable x, free first-order variable X, function symbol $f \in \Sigma_n$, and second-order variable $F \in \mathcal{X}_n$.

Definition 2 Given a signature Σ, and a term $t \in \mathcal{T}(\Sigma^c, \mathcal{X})$, we say that t is *well-curried* w.r.t. Σ, if $\mathcal{C}^{-1}(t)$ is defined, i.e. if there exists a term $u \in \mathcal{T}(\Sigma, \mathcal{X})$ such that $\mathcal{C}(u) = t$.

Lemma 2 *Given a signature Σ, and a substitution σ, the substitution $\sigma_{\mathcal{C}}$ defined by $\sigma_{\mathcal{C}}(F) = \mathcal{C}(\sigma(F))$, for all $F \in Dom(\sigma)$, satisfies $\mathcal{C}(\sigma(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(t))$.*

Proof By structural induction on t. □

This substitution lemma allows us to conclude the following result.

Lemma 3 (Soundness) *If the second-order [context, bounded] equation $t \stackrel{?}{=} u$ over Σ is solvable, then the second-order [context, bounded] equation $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ over Σ^c is also solvable.*

Proof Let σ be a unifier of $t \stackrel{?}{=} u$. We have $\sigma_{\mathcal{C}}(t) = \mathcal{C}(\sigma(t)) = \mathcal{C}(\sigma(u)) = \sigma_{\mathcal{C}}(u)$. Therefore, the substitution $\sigma_{\mathcal{C}}$ is a unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$. The proof can be represented graphically by means of the commutativity of the following category diagram. □

$$\begin{array}{ccc} t \stackrel{?}{=} u & \xrightarrow{\mathcal{C}} & \mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u) \\ \Big\downarrow \sigma & \Longrightarrow^{\mathcal{C}} & \Big\downarrow \sigma_{\mathcal{C}} \\ \sigma(t) & \xrightarrow{\mathcal{C}} & \sigma_{\mathcal{C}}(\mathcal{C}(t)) \end{array}$$

The inverse implication in the previous lemma is not true, as shown in Example 1. The reason is that the currying function is not onto.

Example 1 The following second-order equation

$$g(F(G(a)), F(a), G(a)) \stackrel{?}{=} g(f(a, b), H(a, b), H(X, a))$$

does not have context unifiers. However, its Curry form

$$@(@(@(g, F(G(a))), F(a)), G(a)) \stackrel{?}{=} @(@(@(g, @(@(f, a), b)), H(a, b)), H(X, a))$$

has the following context unifier

$$\sigma = [F \mapsto \lambda x.@(x, b), G \mapsto \lambda x.@(f, x), H \mapsto \lambda xy.@(x, y), X \mapsto f]$$

Similarly, the following second-order equation

$$g(F(G(a)), F(G(a')), F(a), F(a'), G(a), G(a')) \stackrel{?}{=} g(f(a, b), f(a', b), H(a, b), H(a', b), H(X, a), H(X, a'))$$

does not have second-order unifiers, whereas its curry form does. Moreover, the equation does not have any bounded second-order unifier, but its curry form does.

In the previous example, $\sigma(F), \sigma(G), \sigma(H)$ and $\sigma(X)$ are not "well-curried", i.e. they are not the curry form of any well-typed term. For instance, $\sigma(F) = \lambda x.@(x, b)$ is the curry form of $\lambda x.\, x(b)$, but this term is third-order typed (and F is a second-order typed variable), and $\sigma(G) = \lambda x.\ @(f, x)$ is the curry form of $\lambda x.f(x)$, but f has two arguments. This disallows us to reconstruct a unifier for the original equation from the unifier we get for its curry form.

We can also see that the original unification equations contain variables that "touch". For instance, F touches G in $F(G(a))$, and H touches X in $H(X, a)$. In the next sections we will prove, for Second-Order, for Context, and for Bounded Second-Order Unification, that, if no variable touches any other variable, then solvability of the equations is preserved in both directions by our currying transformation. We will also show how to reduce Second-Order, Context and Bounded Second-Order Unification to equations accomplishing such property.

4 Labeling terms

The first step to find a sufficient condition ensuring that the currying function preserves satisfiability is to characterize well-curried terms. This is done by labeling application symbols @ with the *arity* of their left argument, and using a *hat* to mark the roots of right arguments. If left arguments have positive arity, and right arguments and the root have arity zero, then the term is well-curried (see Definition 2).

Definition 3 Given a signature $\Sigma = \bigcup_{n\geq 0} \Sigma_n$, the *curried and labeled signature* $\Sigma^{cl} = \bigcup_{n\geq 0} \Sigma_n^{cl}$ is defined by:

$$\begin{aligned}\Sigma_0^{cl} &= \bigcup_{n\geq 0} \Sigma_n \\ \Sigma_2^{cl} &= \{@^l, \widehat{@^l} | l \in \mathbb{Z}\} \\ \Sigma_n^{cl} &= \emptyset \quad \text{for } n \neq 0, 2\end{aligned}$$

The labeling function $\mathcal{L} : \mathcal{T}(\Sigma^c, \mathcal{X}) \to \mathcal{T}(\Sigma^{cl}, \mathcal{X})$ adds labels to the @'s. It is defined by the following rules:

1. If the left child of an @ is the constant f corresponding to an n-ary symbol $f \in \Sigma_n$, then it has label $l = \text{arity}(f) - 1 = n - 1$.
2. If the head of the left child of an @ is a variable $Z \in \mathcal{X}$, or a bound variable, then it has label -1, regardless what the arity of the variable is.
3. If the head of the left child of an @ is another @ with label n, then it has label $n - 1$.

The labeling function $\widehat{\mathcal{L}} : \mathcal{T}(\Sigma^c, \mathcal{X}) \to \mathcal{T}(\Sigma^{cl}, \mathcal{X})$ is defined using the three previous rules plus the following one:

4. If an @ is the head of the right child of another @, or it is the head of the child of a variable, or it is the root of the term, then, apart from the label, it also has a hat.

Notice that the terms of $\mathcal{T}(\Sigma^{cl}, \mathcal{X})$ are second-order typed terms.

Example 2 The $\widehat{\mathcal{L}}$-labeling of the term

$$@(@(@(g, @(@(f, a), b)), @(a, b)), @(f, a)),$$

used in Example 1 (see Fig. 1) as follows (see Fig. 2).

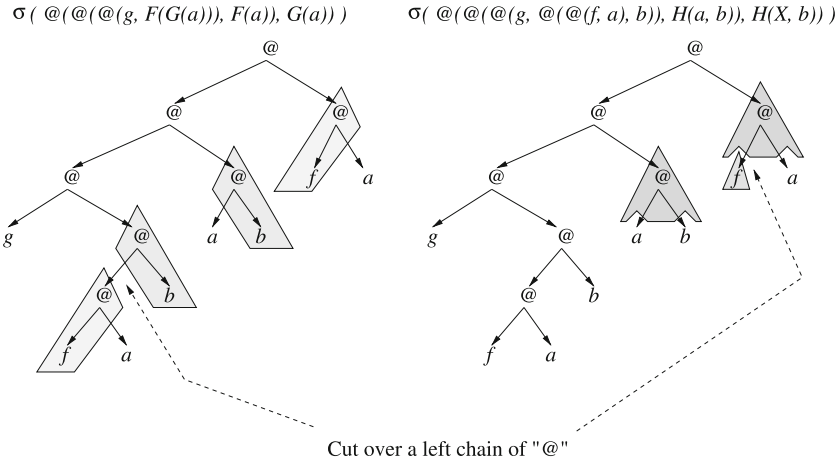


Fig. 1 Common instance of the curried second-order equation of Example 1

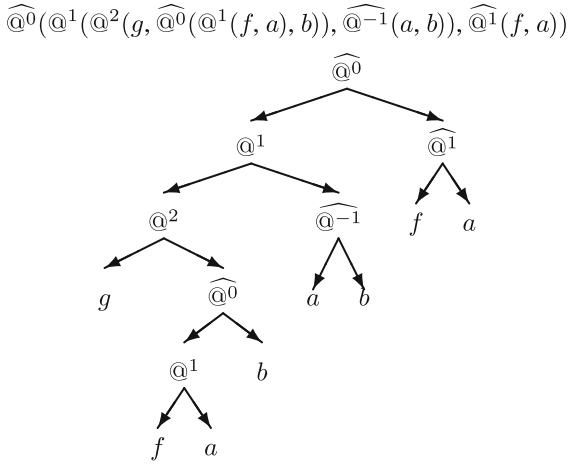


Fig. 2 Representation of the bad-curried labelled term $\widehat{@^0}(@^1(@^2(g, \widehat{@^0}(@^1(f, a), b)), \widehat{@^{-1}}(a, b)), \widehat{@^1}(f, a))$

Notice that labels can be negative numbers. These negative labels can not appear in labeling of well-curried terms.

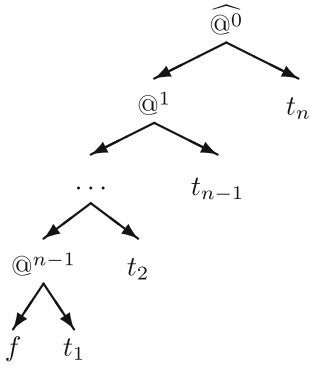
Based on these labels, it is easy to characterize well-curried terms.

Lemma 4 *Given a signature Σ, a term $t \in \mathcal{T}(\Sigma^c, \mathcal{X})$ is well-curried if, and only if, the following two conditions are satisfied*

1. *$\widehat{\mathcal{L}}(t)$ does not contain application symbols with negative labels, i.e. $@^{-n}$, with $n > 0$) and*
2. *$\widehat{\mathcal{L}}(t)$ does not contain application symbols with hat and non-zero labels, i.e. $\widehat{@^n}$ with $n \neq 0$.*

Proof For the left-to-right implication, if t is well-curried, then there exists a term u such that $\mathcal{C}(u) = t$. By induction on the structure of u, we can prove easily that t satisfies the two conditions.

To prove the right-to-left implication, assume that the labeling $\widehat{\mathcal{L}}(t)$ does not contain $@^{-n}$, with $n > 0$, or $\widehat{@^n}$, with $n \neq 0$. Then, any @ symbol is in a sequence of the form:



where $\widehat{@^0}$ is the head of the right child of another @, or of the child of a variable F, or it is the root of the term. We can prove, by the definition of currying, that this is the currying of $f(\mathcal{C}^{-1}(t_1), \ldots, \mathcal{C}^{-1}(t_n)) \in \mathcal{T}(\Sigma, \mathcal{X})$, because f has n arguments and arity n. □

5 When variables do not touch

In this section, we introduce a "no-touch"-condition ensuring that, when we have a unifier for $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, we can find a unifier for $t \stackrel{?}{=} u$. The strategy to prove this result is summarized in the following diagram:

$$\begin{array}{ccccc}
t \stackrel{?}{=} u & \xrightarrow{\mathcal{C}} & \mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u) & \xrightarrow{\widehat{\mathcal{L}}} & \widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u)) \\
\downarrow \sigma & \xleftarrow{\mathcal{C}^{-1}} & \downarrow \sigma_{\mathcal{C}} & \xrightarrow{\widehat{\mathcal{L}}} & \downarrow \sigma_{\widehat{\mathcal{L}}} \\
\sigma(t) & \xleftarrow{\mathcal{C}^{-1}} & \sigma_{\mathcal{C}}(\mathcal{C}(t)) & \xrightarrow{\widehat{\mathcal{L}}} & \sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t)))
\end{array}$$

The proof has two parts. First, we find a sufficient condition that makes the right square commute (Lemma 7). The proof of this commutativity is similar to the proof of Lemma 2. We define a function on substitutions that maps $\sigma_{\mathcal{C}}$ into $\widehat{\mathcal{L}}(\sigma_{\mathcal{C}}) = \sigma_{\widehat{\mathcal{L}}}$, as follows: $\sigma_{\widehat{\mathcal{L}}}(F) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(F))$, for each variable $F \in Dom(\sigma_{\mathcal{C}})$. Then, when the "no-touch"-condition is satisfied, labelling and instantiation commute and we have

$$\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t))) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

This ensures that we map any unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ into a unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$. Second, we prove that, when the right square commutes, then the left one also commutes (Lemma 8). The commutativity of the left square ensures that the currying transformation preserves satisfiability, and allows us to translate any unifier of the curryfied equation $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ into a unifier of $t \stackrel{?}{=} u$.

The sufficient condition we find to ensure the commutativity of the right square is based on the following definition.

Definition 4 Given a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, we say that two variables $F, G \in \mathcal{X}$ *touch*, if t contains a subterm of the form $F(t_1, \ldots, G(u_1, \ldots, u_m), l \ldots, t_n)$. Similarly, $F, X \in \mathcal{X}$ touch if t contains a subterm of the form $F(t_1, \ldots, X, \ldots, t_n)$.

The non-touch condition prevents that non-well-curried terms recombine to well-curried terms after applying a substitution.

Example 3 In the second-order equation of Example 1, the variable F touches G, and the variable H touches X. Therefore, this example does not satisfy the "no-touch"-condition. The labelling of the substitution σ results on

$$\sigma_{\widehat{\mathcal{L}}} = \left[F \mapsto \lambda x.\widehat{@^{-1}}(x, b), \quad G \mapsto \lambda x.\widehat{@^1}(f, x), \quad H \mapsto \lambda xy.\widehat{@^{-1}}(x, y), \quad X \mapsto f \right]$$

that is not a unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$. In particular, we have three different terms:

$$\begin{aligned}
\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t))) &= \widehat{@^0}(@^1(@^2(g, \widehat{@^{-1}}(\widehat{@^1}(f,a),b)), \widehat{@^{-1}}(a,b)), \widehat{@^1}(f,a)) \\
\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(u))) &= \widehat{@^0}(@^1(@^2(g, \widehat{@^0}(@^1(f,a),b)), \widehat{@^{-1}}(a,b)), \widehat{@^{-1}}(f,a)) \\
\widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(t))) &= \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(u))) \\
&= \widehat{@^0}(@^1(@^2(g, \widehat{@^0}(@^1(f,a),b)), \widehat{@^{-1}}(a,b)), \widehat{@^1}(f,a))
\end{aligned}$$

Before proving the commutativity of the right diagram (Lemma 7), we prove a weaker version of this result using $\mathcal{L}$ instead of $\widehat{\mathcal{L}}$ as labelling function. First we prove that substitutions and labelling (without hats) commute (Lemma 5). Second, we prove that we can transform every most general unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ into a most general unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$ (Lemma 6).

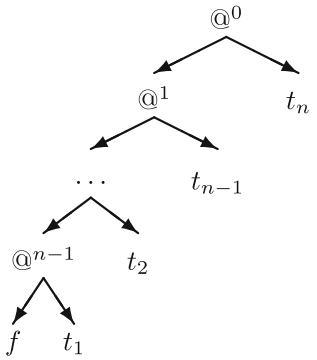
Lemma 5 *Let t be a term where variables do not touch, and $\sigma_{\mathcal{C}}$ a second-order [context, bounded] substitution on Σ^c. The second-order [context, bounded] substitution $\sigma_{\mathcal{L}}$ defined by $\sigma_{\mathcal{L}}(F) = \mathcal{L}(\sigma_{\mathcal{C}}(F))$, for each variable $F \in Dom(\sigma_{\mathcal{C}})$, satisfies*

$$\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

Proof Since $\sigma_{\mathcal{C}}$ and $\sigma_{\mathcal{L}}$ only differ in the introduction of labels, if $\sigma_{\mathcal{C}}$ is a context [bounded] substitution, then $\sigma_{\mathcal{L}}$ is also a context [bounded] substitution.

By the same reason, $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t)))$ and $\mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$ have the same form, except for the labels. Therefore, we only have to compare the labels of the corresponding @'s in both terms. Taking a particular @ occurrence in $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t)))$ and $\mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$, there are two cases:

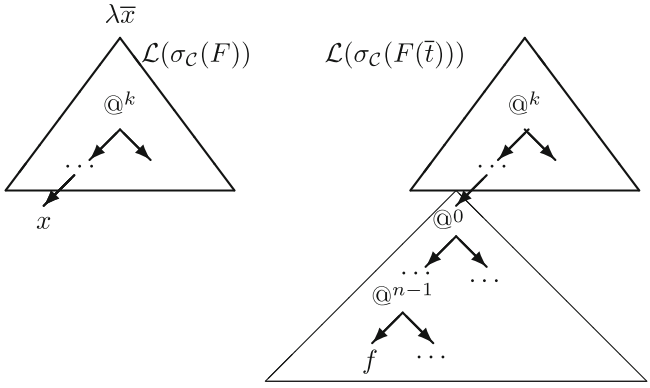
1. If the occurrence of the @ is outside the instance of any variable, then this @ already occurs in $\mathcal{C}(t)$, and it is in a sequence of the form (see proof of Lemma 4):



   where the occurrence of the f (corresponding to an n-ary function symbol), and all the @'s occurring between the particular @ and f, already occurred in $\mathcal{C}(t)$ (they have not been introduced by an instantiation either). Thus, the particular

@ gets the same label in $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t)))$ as in $\mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$, because this label only depends on the left descendants, and they have not been introduced by $\sigma_{\mathcal{C}}$ or $\sigma_{\mathcal{L}}$.

2. If the @ is inside the instance of a variable F, we have to prove that it gets the same label in $\sigma_{\mathcal{L}}(\mathcal{L}(F(\mathcal{C}(t_1),\ldots,\mathcal{C}(t_n))))$ as in $\mathcal{L}(\sigma_{\mathcal{C}}(F(\mathcal{C}(t_1),\ldots,\mathcal{C}(t_n))))$. For the first term we have $\sigma_{\mathcal{L}}(\mathcal{L}(F(\mathcal{C}(t_1),\ldots,\mathcal{C}(t_n)))) = \sigma_{\mathcal{L}}(F)(\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t_1))),\ldots,\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t_n)))) = \mathcal{L}(\sigma_{\mathcal{C}}(F))(\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t_1))),\ldots,\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t_n))))$. For the second term we have $\mathcal{L}(\sigma_{\mathcal{C}}(F(\mathcal{C}(t_1),\ldots,\mathcal{C}(t_n)))) = \mathcal{L}(\sigma_{\mathcal{C}}(F)(\sigma_{\mathcal{C}}(\mathcal{C}(t_1)),\ldots,\sigma_{\mathcal{C}}(\mathcal{C}(t_n))))$. Therefore, in the first case we label $\sigma_{\mathcal{C}}(F)$ before applying the substitution (so we have bound variables in the place of the arguments), whereas in the second case we label $\sigma_{\mathcal{C}}(F)$ after applying the substitution (so we already have the arguments $\mathcal{C}(t_i)$). As we will see, in both cases the labels we get are the same. The root of one of the arguments $\mathcal{C}(t_i)$ can be a left descendant of the @, and its label will depend on such argument. However, if variables do not touch, the head of any argument t_i of F is a constant or a function symbol. Therefore, the head of $\mathcal{C}(t_i)$ is either a 0-ary constant a or an @ with label 0. Hence, the labels of the ancestors of the argument inside $\sigma_{\mathcal{C}}(F)$ will be the same if we replace the argument by a bound-variable, and the label of the corresponding @ inside $\sigma_{\mathcal{L}}(F)$ will be the same.



□

Lemma 6 *If the variables of the second-order equation $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general second-order [context, bounded] unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the substitution $\sigma_{\mathcal{L}}$ defined by $\sigma_{\mathcal{L}}(F) = \mathcal{L}(\sigma_{\mathcal{C}}(F))$, for each variable $F \in Dom(\sigma_{\mathcal{C}})$, is a most general second-order [context, bounded] unifier of the equation*

$$\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$$

Proof By Lemma 5, we have $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$ and $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(u))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(u)))$. As $\sigma_{\mathcal{C}}(\mathcal{C}(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(u))$, we can conclude that $\sigma_{\mathcal{L}}$ is a unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$.

Given a unifier $\sigma_{\mathcal{L}}$ of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$, we can find a unifier $\sigma_{\mathcal{C}}$ of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ by removing labels. Moreover, if $\sigma'_{\mathcal{L}}$ is more general than $\sigma_{\mathcal{L}}$, then by removing the labels from $\sigma'_{\mathcal{L}}$ we get a unifier more general than the one resulting from removing the labels of $\sigma_{\mathcal{L}}$. Therefore, if $\sigma_{\mathcal{C}}$ is most general for $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then $\sigma_{\mathcal{L}}$ is also most

general for $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$, otherwise, there would be a unifier more general than $\sigma_{\mathcal{L}}$, and removing labels we could obtain a unifier more general than $\sigma_{\mathcal{C}}$. □

Lemma 7 *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general second-order [context, bounded] unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the substitution $\sigma_{\widehat{\mathcal{L}}}$ defined by $\sigma_{\widehat{\mathcal{L}}}(F) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(F))$, for each variable $F \in Dom(\sigma_{\mathcal{C}})$, satisfies*

$$\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t))) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

and is a most general second-order [context, bounded] unifier of the equation

$$\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$$

Proof First, we prove that $\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t))) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$.

We already know that both terms have the same form and the same labels, thus we only have to prove that they have the same hats. Again, there are two cases:

- If the occurrence of the @ is outside the instance of any variable, then the only situation we have to consider is the following. If the @ has as father a variable F in $\mathcal{C}(t)$, and after instantiation, it becomes a left child of an @ inside $\sigma_{\widehat{\mathcal{L}}}(F)$, then it will loose the hat. However, if variables do not touch in the equation, this situation is not possible: As $\mathcal{C}(t)$ and $\mathcal{C}(u)$ are trivially well-curried, by Lemma 4, $\mathcal{L}(\mathcal{C}(t))$ and $\mathcal{L}(\mathcal{C}(u))$ will not contain @'s with negative labels. Let $\sigma_{\mathcal{L}}$ be the most general unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$ given by Lemma 6. Now, by Lemma 1, as $\sigma_{\mathcal{L}}$ is a most general unifier, for any variable F, $\sigma_{\mathcal{L}}(F)$ will not contain @'s with negative labels, either. We can conclude then that the head of any argument t_i of F cannot be a left child of an @, because, as the heads of $\sigma_{\mathcal{L}}(t_i)$ have zero label or are 0-ary constants, this situation would introduce a negative label in some @ inside $\sigma_{\mathcal{L}}(F)$.
- If the occurrence of the @ is inside the instance of a variable F, then we have to prove that the fact that @ has a hat or not, does not depend on the arguments of F. This is obvious because this fact does not depend on the descendants of the @.

Finally, using the same argument as in Lemma 6, we conclude that $\sigma_{\widehat{\mathcal{L}}}$ is a most general unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$. □

Once we have proved the commutativity of the right diagram, we can prove the commutativity of the left diagram, i.e. the completeness of the currying transformation: if variables do not touch, then the transformation preserves solvability of equations.

Lemma 8 (Completeness) *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general second-order [context, bounded] unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the substitution defined by $\sigma(F) = \mathcal{C}^{-1}(\sigma_{\mathcal{C}}(F))$ for each variable $F \in Dom(\sigma_{\mathcal{C}})$, satisfies*

$$\mathcal{C}(\sigma(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(t))$$

and is a most general second-order [context, bounded] unifier of $t \stackrel{?}{=} u$.

Proof Let $\sigma_{\widehat{\mathcal{L}}}$ be the most general unifier of the equation $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$ given by Lemma 7. As $\mathcal{C}(t)$ and $\mathcal{C}(u)$ are well-curried, by Lemma 4, they do not contain negative labels nor hats over non-zero labeled @'s. Then, by Lemma 1, $\sigma_{\widehat{\mathcal{L}}}$ does not introduce such kind of labels or hats. Therefore, as $\sigma_{\widehat{\mathcal{L}}}(F)$ is defined as the labeling of $\sigma_{\mathcal{C}}(F)$, using again Lemma 4, $\sigma_{\mathcal{C}}(F)$ will be well-curried, and we can define $\sigma(F) = \mathcal{C}^{-1}(\sigma_{\mathcal{C}}(F))$ for each variable $F \in Dom(\sigma_{\mathcal{C}})$.

By the properties of the currying, this substitution satisfies $\mathcal{C}(\sigma(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(t))$. This allows us to conclude that it is a unifier of $t \stackrel{?}{=} u$.

The proof for its most-generality follows the same argument as in Lemma 6. □

6 Main results

In this section we prove that Second-Order, Context and Bounded Second-Order Unification can be reduced to their corresponding version where variables do not touch. This concludes that the three problems can be reduced to the fragment with just one binary symbol and constants.

Theorem 1 *Decidability of Second-Order Unification can be NP-reduced to decidability of Second-Order Unification with just one binary function symbol, and constants.*

Proof By Lemmas 3 and 8, we know that, when variables do not touch, satisfiability of second-order problems is preserved by currying. Now, we will prove that we can NP-reduce solvability of Second-Order Unification to solvability of the corresponding problems without touching variables.

The reduction algorithm guesses a substitution ρ as follows. For every n-ary variable F of the original unification problem (notice that n can be 0), we guess one of the following possibilities:

- *Projection*: $F \mapsto \lambda x_1 \cdots x_n.x_i$, for some $i \in \{1, \ldots, n\}$.
- *Instantiation*: $F \mapsto \lambda x_1 \cdots x_n.f(F_1(x_1, \ldots, x_n), \ldots, F_m(x_1, \ldots, x_n))$, for some function symbol $f \in \Sigma_m$ occurring in the original unification problem, and m fresh free variables $F_1, \ldots, F_m$ of arity n.
  Notice that first-order variables are instantiated using this rule for $n = 0$ by $f(X_1, \ldots, X_m)$ for some function symbol $f \in \Sigma_m$ and fresh first-order variables $X_1, \ldots, X_m$.

Then the solvability of a problem $t \stackrel{?}{=} u$ is reduced to solvability of $\rho(t) \stackrel{?}{=} \rho(u)$. Obviously, this reduction can be performed in polynomial non-deterministic time, and the resulting problem satisfies that variables do not touch.

Since the new problem is an instance of the original one, if the new problem is solvable, then the original one is also solvable.

If the original problem $t \stackrel{?}{=} u$ is solvable, then there is an appropriate guessing of ρ that results in another solvable problem $\rho(t) \stackrel{?}{=} \rho(u)$. Let σ be a most general unifier of $t \stackrel{?}{=} u$. The substitution ρ can be constructed as follows. For every variable F of the original problem, let $\sigma(F) = \lambda x_1 \ldots x_n.t$ be written in normal form. Taking t as a tree, descend from the root to the left-most leaf, discarding free variables, until you

get a bound variable x_i, a 0-ary variable or a function symbol f (this must be a symbol occurring in the problem, by Lemma 1). Then the instantiation $F \mapsto \lambda x_1 \dots x_n.x_i$, if we find a bound variable x_i, $F \mapsto \lambda x_1 \dots x_n.a$ for some fixed constant a, if we find a first-order variable, or $F \mapsto \lambda x_1 \dots x_n.f(F_1(x_1, \dots, x_n), \dots, F_m(x_1, \dots, x_n))$, if we find a subterm like $f(t_1, \dots, t_m)$, results in a solvable problem that can be constructed using *projection* and *instantiation*. □

Corollary 9 of [22] (see also [23]) states that Second-Order Unification is undecidable even for problems with just one second-order variable and 4 occurrences of this variable. Theorem 1 and this corollary provides us the following result.

Corollary 1 *Second-Order Unification is undecidable for one binary function symbol and one second-order variable occurring four times.*

Theorem 2 *Decidability of Context Unification can be NP-reduced to decidability of Context Unification with just one binary function symbol, and constants.*

Proof Again, by Lemmas 3 and 8, we know that, when variables do not touch, satisfiability of Context Unification problems is preserved by currying. Now, we will prove that we can NP-reduce solvability of Context Unification to solvability of the corresponding problems without touching variables.

The reduction works as follows. We construct a substitution ρ, guessing, for every n-ary variable F of the original problem (notice that n can be zero), one of the following possibilities:

- *Projection*: $F \mapsto \lambda x.x$, if it is unary.
- *Instantiation*:

$$F \mapsto \lambda x_1 \dots x_n.f(F_1(x_{\pi(1)}, \dots, x_{\pi(r_1)}), \dots, F_m(x_{\pi(r_{m-1}+1)}, \dots, x_{\pi(n)}))$$

  for some function symbol $f \in \Sigma_m$ occurring in the original unification problem, some permutation π over n, and some values $1 \leq r_1 \leq \dots \leq r_{m-1} \leq n$, and being $m = \text{arity}(f)$, and $F_1, \dots, F_m$ fresh free variables of appropriate arity.

Notice that these substitutions preserve linearity of context variables instantiations. Again, this reduction can be performed in polynomial non-deterministic time, and the resulting problem satisfies that variables do not touch.

Since the new problem is an instance of the original one, if the new problem is solvable, then the original one is also solvable.

Now, like in the second-order case we prove the other direction. However, in this case assuming that the signature Σ contains, at least, a binary function symbol (say h) and a 0-ary constant (say a) is crucial because our proof will consider ground unifiers (that map variables to terms without free occurrences of variables).

Let the original problem $t \stackrel{?}{=} u$ be solvable, and σ be a most general unifier. Let $\sigma' = \tau \circ \sigma$ be a ground unifier, where τ maps free n-ary ($n \geq 1$) variables occurring in $\sigma(t)$ to $\lambda x_1 \dots x_n.h(x_1, h(x_2, h(\dots, h(x_n, a)\dots)))$ and maps the 0-ary variables occurring in $\sigma(t)$ to a, for some binary function symbol h and constant a from Σ.

Now, guided by substitution σ', by applying the *projection* and *instantiation* rules to the original problem, we can build a solvable Context Unification problem where variables do not touch: for every free variable F in the original problem

- if $\sigma'(F) = \lambda x_1 \ldots x_n.f(t_1, \ldots, t_m)$, then to obtain the new problem we instantiate F by $\lambda x_1 \ldots x_n.f(F_1(x_{\pi(1)}, \ldots, x_{\pi(r_1)}), \ldots, F_m(x_{\pi(r_{m-1}+1)}, \ldots, x_{\pi(r_m)}))$, where $\{\pi(r_{i-1} + 1), \ldots, \pi(r_i)\}$ is the set of indices of bound variables from $x_1, \ldots, x_n$ occurring in t_i. Notice that $n = 0$ is a particular case that deals with first-order variables.
- Otherwise, if $\sigma'(F) = \lambda x.x$, then instantiate F by $\lambda x.x$.

Then, the substitution that maps the free variables F_i, occurring in the new problem where variables do not touch, to the terms $\lambda x_{\pi(r_{i-1}+1)} \ldots x_{\pi(r_i)}.\ t_i$, is obviously a solution of the new problem. □

Example 4 We present an example of how the reduction from Context Unification to non-touching Context Unification works.

The equation $F(X, Y) \stackrel{?}{=} G(Z)$ has touching variables. Consider the following most general unifier $\sigma = [G \mapsto \lambda x.F(F'(x), Y)\ ,\ X \mapsto F'(Z)]$. As described in the proof of Theorem 2, assuming that we have a binary function symbol h and a constant a in the signature, we can construct the substitution $\tau = [F \mapsto \lambda x, y.h(x, h(y, a))\ ,\ F' \mapsto \lambda x.h(x, a)\ ,\ Z \mapsto a\ ,\ Y \mapsto a]$, such that $\sigma' = \tau \circ \sigma$ is a (non-most-general) ground unifier. Guided by σ', we can construct the substitution ρ that would have to be guessed by the reduction algorithm:

$$\rho = [\ F \mapsto \lambda x, y.h(F_1(x), F_2(y)),\quad G \mapsto \lambda x.h(G_1(x), G_2),\quad X \mapsto h(X_1, X_2),\quad Y \mapsto a,\quad Z \mapsto a]$$

Therefore, solvability of $F(X, Y) \stackrel{?}{=} G(Z)$ is reduced to solvability of $\rho(F(X, Y) \stackrel{?}{=} G(Z))$, i.e. $h(F_1(h(X_1, X_2)), F_2(a)) \stackrel{?}{=} h(G_1(a), G_2)$. Since ρ is more general that σ' w.r.t. the variables of the original equation, and σ' solves the original equation, we can find a substitution that solves the new equation.

Theorem 3 *Decidability of Bounded Second-Order Unification can be NP-reduced to decidability of Bounded Second-Order Unification with just one binary function symbol, and constants.*

Proof Again, by Lemmas 3 and 8, we know that, when variables do not touch, satisfiability of Bounded Second-Order Unification problems is preserved by currying. The proof of the NP-reducibility of solvability of Bounded Second-Order Unification to solvability of the corresponding problems without touching variables is like the proof of Theorem 2. We only have to change the *instantiate* rule:

For every n-ary variable F of the original problem, we guess one of the following possibilities (notice that n can be zero):

- *Project*: $F \mapsto \lambda x.x$, if it is unary.
- *Instantiate*:

  $$F \mapsto \lambda x_1 \ldots x_n.f(F_1(x_{\pi(1)}, \ldots, x_{\pi(r_1)}), \ldots, F_m(x_{\pi(r_{m-1}+1)}, \ldots, x_{\pi(r_m)}))$$

  for some m-ary function symbol $f \in \Sigma_m$ occurring in the original unification problem, some permutation π over $r_m \leq n$, some values $1 \leq r_1 \leq \cdots \leq r_m \leq n$, and being $F_1, \ldots, F_m$ fresh free variables of appropriate arity.

Notice that these rules do not need to preserve linearity of bounded second-order variables instantiations, as far as we consider a permutation of $r_m \leq n$ many bound variables. The rest of the proof is similar as for the Context Unification case. □

7 Conclusions

Our technique serves for reducing decidability of Second-Order Unification, Context Unification and Bounded Second-Order Unification to their corresponding versions where just one binary function symbol is considered. This simplifies the study of these problems. The same technique could be applied to other variants of Second-Order Unification like Well-Nested Context Unification [18], Stratified Context Unification [28], and Sequence Unification [15].

The extension of our technique to third-order and higher orders sets out some difficulties. First, we must deal with instances of variables that are not connected. For instance, the following problem:

$$f(F(\lambda x.g(x), a), F(\lambda x.g'(x), a')) \stackrel{?}{=} f(f(g(h(a)), a), f(g'(h(a')), a'))$$

is solved by the substitution:

$$F \mapsto \lambda xy.f(x(h(y)), y)$$

where the instance of the third-order variable F is split into two pieces f and h. In such situations we have to guarantee that these pieces do not touch, to avoid that these splitting points could cut a left chain of @'s.

References

1. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.: Explicit substitutions. J. Funct. Prog. **1**(4), 375–416 (1998)
2. Barendregt, H.P.: The Lambda Calculus—it's Syntax and Semantics. North-Holland, Amsterdam (1984)
3. Bjorner, N., Muñoz, C.: Absoulte explicit unification. In: Proceedings of the 11th International Conference on Rewriting Techniques and Applications, RTA'00, LNCS, vol. 1833, pp. 31–46, Norwich (2000)
4. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: Proceedings of the 15th International Conference on Rewriting Techniques and Applications, RTA'04, LNCS, vol. 3091, pp. 105–118. Springer (2004)
5. Comon, H.: Completion of rewrite systems with membership constraints. J. Symb. Comput. **25**(4), 397–453 (1998)
6. Dowek, G.: Higher-order unification and matching. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning vol II, Chapter 16, pp. 1009–1062. Elsevier, Amsterdam (2001)
7. Dowek, G., Hardin, T., Kirchner, C.: Higher-order unification via explicit substitutions. Inf. Comput. **157**, 183–235 (2000)
8. Farmer, W.M.: A unification algorithm for second-order monadic terms. Ann. Pure Appl. Logic. **39**, 131–174 (1988)
9. Farmer, W.M.: Simple second-order languages for wich unification is undecidable. Theor. Comput. Sci. **87**, 173–214 (1991)

10. Ganzinger, H., Nieuwenhuis, R., Nivela, P.: Context trees. In: Proceedings of the 1st Internetional Conference on Automated Reasoning, LNCS, vol. 2083, pp. 242–256 (2001)
11. Goldfarb, W.D.: The undecidability of the second-order unification problem. Theor. Comput. Sci. **13**, 225–230 (1981)
12. Gutiérrez, C.: Satisfiability of equations in free groups is in PSPACE. In: ACM (ed.) Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, STOC'00, pp. 21–27. ACM Press (2000)
13. Huet, G.: A unification algorithm for typed λ-calculus. Theor. Comput. Sci. **1**, 27–57 (1975)
14. Kutsia, T.: Solving equations with sequence variables and sequence functions. J. Symb. Comput. **42**(3), 352–388 (2007)
15. Kutsia, T., Levy, J., Villaret, M.: Sequence unification through currying. In: Proceedings of the 18th International Conference on Rewriting Techniques and Applications, RTA'07, LNCS, vol. 4533, pp. 288–302. Springer (2007)
16. Levy, J.: Linear second-order unification. In: Proceedings of the 7th International Conference on Rewriting Techniques, LNCS, vol. 1103, pp. 332–346, New Brunswick (1996)
17. Levy, J.: Decidable and undecidable second-order unification problems. In: Proceedings of the 9th International Conference on Rewriting Techniques and Applications, RTA'98, LNCS, vol. 1379, pp. 47–60, Tsukuba (1998)
18. Levy, J., Niehren, J., Villaret, M.: Well-nested context unification. In: Proceedings of the 20th International Conference on Automated Deduction, CADD-20, LNAI, vol. 3632, pp. 149–163, Springer (2005)
19. Levy, J., Schmidt-Schauß, M., Villaret, M.: Monadic second-order unification is *np*-complete. In: Proceedings of the 15th Interantional Conference on Rewriting Techniques and Applications, RTA'04, LNCS, vol. 3091, pp. 55–69, Aachen (2004)
20. Levy, J., Schmidt-Schauß, M., Villaret, M.: Bounded second-order unification is *np*-complete. In: Proceedings of the 17th International Conference on Rewriting Techniques and Applications, RTA'06, LNCS, vol. 4098, pp. 400–414, Seattle (2006)
21. Levy, J., Schmidt-Schauß, M., Villaret, M.: Stratified context unification is *np*-complete. In: Proceedings of the 3rd International Conference on Automated Reasoning, IJCAR'06, LNCS, vol. 4130, pp. 82–96, Seattle (2006)
22. Levy, J., Veanes, M.: On unification problems in restricted second-order languages. In: Annual Conference of the European Association of Computer Science Logic, CSL'98, Brno, Czech Republic (1998)
23. Levy, J., Veanes, M.: On the undecidability of second-order unification. Inf. Comput. **159**(1–2), 125–150 (2000)
24. Levy, J., Villaret, M.: Context unification and traversal equations. In: Proceedings of the 12th International Conference on Rewriting Techniques and Applications, RTA'01, LNCS, vol. 2041, pp. 169–184, Utrecht (2001)
25. Makanin, G.S.: The problem of solvability of equations in a free semigroup. Math. USSR Sbornik **32**(2), 129–198 (1977)
26. Niehren, J., Planque, L., Talbot, J.-M., Tison, S.: N-ary queries by tree automata. In: Proceedings of the 10th International Symposium on Database Programming, DBPL'05, LNCS, vol. 3774, Springer (2005)
27. Plandowski, W.: Satisfiability of word equations with constants is in PSPACE. In: IEEE (ed.) Proceedings of the 40th Annual Symposium on Foundations of Computer Science, FOCS'99, pp. 495–500. IEEE Computer Society Press, New York City (1999)
28. Schmidt-Schauß, M.: A decision algorithm for stratified context unification. J. Logic Comput. **12**, 929–953 (2002)
29. Schmidt-Schauß, M.: Decidability of bounded second-order unification. Inf. Comput. **188**(2), 143–178 (2004)
30. Schmidt-Schauß, M., Schulz, K.U.: Solvability of context equations with two context variables is decidable. J. Symb. Comput. **33**(1), 77–122 (2002)
31. Villaret, M.: On some variants of second-order unification. PhD Thesis, Technical University of Catalonia (2004)
32. Zhezherun, A.P.: Decidability of the unification problem for second order languages with unary function symbols. Kibernetika (Kiev) **5**, 120–125 (1979). Translated as Cybernetics **15**(5), 735–741 (1980)