On the Limits of Second-Order Unification

Jordi Levy

Institut d'Investigació en Intel·ligència Artificia (IIIA-CSIC)
Barcelona, Spain
levy@iiia.csic.es

Abstract

Second-Order Unification is a problem that naturally arises when applying automated deduction techniques with variables denoting predicates. The problem is undecidable, but a considerable effort has been made in order to find decidable fragments, and understand the deep reasons of its complexity. Two variants of the problem, Bounded Second-Order Unification and Linear Second-Order Unification –where the use of bound variables in the instantiations is restricted–, have been extensively studied in the last two decades. In this paper we summarize some decidability/undecidability/complexity results, trying to focus on those that could be more interesting for a wider audience, and involving less technical details.

1 Introduction

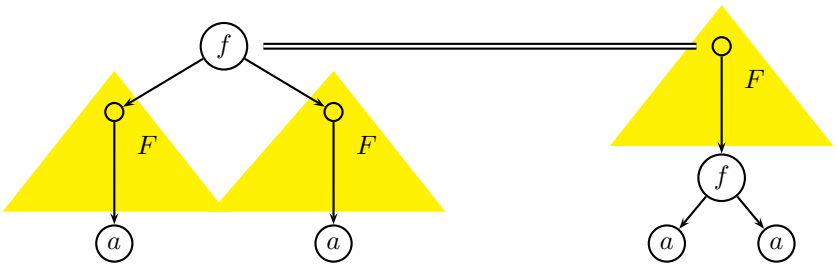
Unification consists on solving equations over expressions or terms. In the basic form of first-order unification, terms may contain (first-order) variables. When unifying, these variables may be replaced by terms, in order to make both sides of an equation $t \stackrel{?}{=} u$ syntactically equal. If we are not concerned with efficiency, we can decide first-order unifiability, and eventually compute the substitution or unifier, applying the following two transformations till we get an empty set of equations:

Simplification: $\{f(t_1, \ldots, t_n) \stackrel{?}{=} f(u_1, \ldots, u_n)\} \cup E \Rightarrow \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\} \cup E$

Instantiation: $\{X \stackrel{?}{=} t\} \cup E \Rightarrow E\rho$
when X does not occur in t,
and the unifier is extended with the instantiation $\rho = [X \mapsto t]$

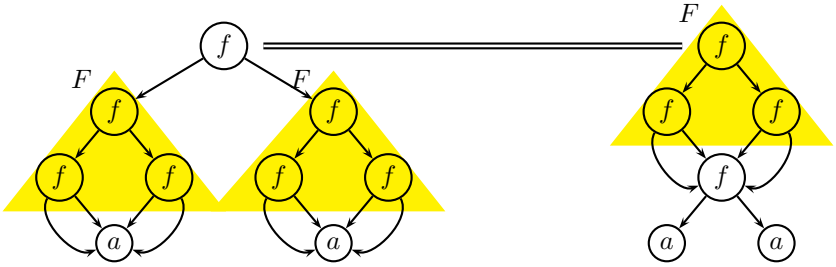
In second-order unification, variables may be applied to terms, and instances of variables may use these arguments, once or several times. Formally, variables are replaced by (second-order typed) lambda-terms, and then β-reduced. In general, Second-Order Unification allows the use of bound variables and binders also in equations. However, we know from experience that they do not make any remarkable difference with respect to the decidability or complexity of the problem. Therefore, for simplicity, we will avoid them in equations along this paper.

Consider the problem $f(F(a), F(a)) \stackrel{?}{=} F(f(a, a))$, where capital letters denote variables. One solution is $[F \mapsto \lambda x \,.\, x]$, that instantiates the equation as $f(a, a) = f(a, a)$.



But, there are other solutions like $[F \mapsto \lambda x\,.\,f(f(x,x),f(x,x))]$, that using the argument 4 times, instantiates the equation as:

$$f(f(f(a,a),f(a,a)),f(f(a,a),f(a,a))) = f(f(f(a,a),f(a,a)),f(f(a,a),f(a,a)))$$



In this example we can observe some of the properties that distinguishes second-order unification from first-order unification. First, the number of most general unifiers may be infinite. Second, equations of the form $F(t) \stackrel{?}{=} u$, where F occurs in u, that are unsolvable in first-order (occur-check), may be solvable in second-order (depending on the arguments). Notice also that, these type of equations, even when F does not occur in u, are not trivially solvable, because instances of F may use one or several times the argument t, and we must identify occurrences of t in u to find the value of F.

Depending on the number of times that the instance of a second-order variable may use their arguments, we distinguish three variants of second-order unification:

1. If there is no restriction on the number of times that a variable uses its arguments, we have *Second-Order Unification (SOU)*.
2. If the arguments are used exactly once, we have *Linear Second-Order Unification (LSOU)*. When, additionally, second-order variables are unary, we have *Context Unification (CU)*.
3. If the arguments may be used once or none, we have *Bounded Second-Order Unification (BSOU)*.

In the previous example $f(F(a),F(a)) \stackrel{?}{=} F(f(a,a))$, the substitution $[F \mapsto \lambda x\,.\,x]$ is a context unifier, and also a bounded second-order unifier, whereas $[F \mapsto \lambda x\,.\,f(f(x,x),f(x,x))]$ is not linear nor bounded. Despite this restriction, CU and BSOU are also infinitary, as the equation $F(f(a)) \stackrel{?}{=} f(F(a))$ and the infinite set of unifiers $\{[F \mapsto \lambda x\,.\,f(\stackrel{n}{\ldots}\,f(x)\ldots)]\}_{n\geq 0}$ show.

2 Some General Decidability and Undecidability Results

Second-Order Unification (SOU) was proved undecidable by Goldfarb (1981) by reducing the Hilbert's Tenth Problem. The more general case of Third-Order Unification was already proved undecidable, independently, by Lucchesi (1972) and Huet (1973). Pietrzykowski (1973) described the first complete second-order unification procedure, that was later extended to the higher-order case by Jensen and Pietrzykowski (1976). Gould (1966) was the first who found a complete second-order matching algorithm.

Context Unification (CU) was introduced independently by Comon (1993) and Schmidt-Schauß (1995). Comon (1993) studied the problem to solve membership constraints. He proved that context unification is decidable when any occurrence of the same context variable is always applied to the same term. Schmidt-Schauß (1995) was interested in reducing the

problem of unification modulo distributivity to a subset of such context unification problems. He proved that context unification is decidable when terms are *stratified*. By stratified we mean that the string of second-order variables we find going from the root of a term to any occurrence of the same variable, is always the same. Very recently, Jez (2014) has proved that CU is in PSPACE, answering a question that has remained open for 21 years.

Linear Second-Order Unification (LSOU) was introduced by Levy (1996). The generalization w.r.t. CU comes from two facts 1) we consider second-order terms, thus expressions may contain λ-bindings, and 2) second-order variables are not restricted to be unary like context variables. This generalization is motivated by the following example. The unification problem $F(a) \stackrel{?}{=} G(f(a))$ has two minimum linear second-order unifiers:

$$\sigma_1 = [F \mapsto \lambda x \,.\, G(f(x))]$$
$$\sigma_2 = [F \mapsto \lambda x \,.\, H(x, f(a))][G \mapsto \lambda x \,.\, H(a, x)]$$

However, if we restrict ourselves to unary second-order variables (context variables), we can not represent the second minimum unifier. Levy (1996) described a complete procedure for this problem and proved that the problem is decidable in the same cases studied in (Comon, 1993) and (Schmidt-Schauß, 1995), and also when no variable occurs more that twice. We will come back to this case later. Levy and Villaret (2000) proved that linear second-order unification can be reduced to context unification with tree-regular constraints, and commented on the possibility that linear second-order unification is decidable, if context unification is decidable (something that was unknown by that time, and should be revisited now). de Groote (2000) proved that Linear Higher-Order Matching is NP-complete.

Finally, Bounded Second-Order Unification was introduced and proved decidable by Schmidt-Schauß (2004). Later, Levy et al. (2006a) proved that BSOU is in fact NP-complete, using a similar technique as to prove that Monadic SOU is NP-complete (Levy et al., 2004).

Previous results analyze the general versions of these three variants of second-order unification. However, there are other papers where some restrictions on the classes of problems are studied. These results are also important because they shed light on the sources of complexity of these problems, and the possibility of finding (efficient) algorithms for some subclasses of problems. In the rest of this paper we will comment on some of these restrictions, focusing specially on those that could be interesting for a wide audience, and could help to understand the nature of this class of problems.

3 Pre-Unification

Huet (1975) introduced the notion *pre-unification*, a form of "lazy" unification, and found a non-redundant procedure for it. The idea is to forget about equations of the form $F(\dots) \stackrel{?}{=} G(\dots)$, called flex-flex, since SOU problems only containing such kind of equations are trivially solvable. SO pre-unification reminds first-order unification, but instead of instantiation rule, we have two new rules, imitation and projection.

Simplification: $\{f(t_1,\ldots,t_n) \stackrel{?}{=} f(u_1,\ldots,u_n)\} \cup E \Rightarrow \{t_1 \stackrel{?}{=} u_1,\ldots,t_n \stackrel{?}{=} u_n\} \cup E$

Imitation: $\{X(t_1,\ldots,t_n) \stackrel{?}{=} f(u_1,\ldots,u_m)\} \cup E \Rightarrow$
$\Big(\{X'(t_1,\ldots,t_n) \stackrel{?}{=} u_1,\ldots,X'(t_1,\ldots,t_n) \stackrel{?}{=} u_m\} \cup E\Big)\,\rho$
where $X'$are fresh variables, and the unifier is extended with the instantiation
$\rho = [X \mapsto \lambda y_1,\ldots,y_n\,.\,f(X'_1(y_1,\ldots,y_n),\ldots,X'_m(y_1,\ldots,y_n))]$

Projection: $\{X(t_1,\ldots,t_n) \stackrel{?}{=} f(u_1,\ldots,u_m)\} \cup E \Rightarrow$
$\Big(\{t_i \stackrel{?}{=} f(u_1,\ldots,u_m)\} \cup E\Big)\,\rho$
where $X'$are fresh variables, and the unifier is extended with the instantiation
$\rho = [X \mapsto \lambda y_1,\ldots,y_n\,.\,y_i]$

Pre-unification is complete for SOU, in the sense that a problem is solvable if, and only if, pre-unification leads to a set of flex-flex equations. However, some sequences of pre-unification transformations do not terminate. Pre-unification is also complete for BSOU, but we must re-adapt imitation rule in order to avoid repetition of bound variables. So, we instantiate $[X \mapsto \lambda y_1,\ldots,y_n\,.\,f(X'_1(y_{i_1},\ldots,y_{i_r}),\ldots,X'_m(y_{i_s},\ldots,y_{i_n}))]$, where $i_1,\ldots,i_n$ is a permutation of $1,\ldots,n$. Pre-unification is not complete for CU because some sets of flex-flex equations, like $\{X(a) \stackrel{?}{=} X(b)\}$, are not solvable (notice that instances of variables *must* use their arguments). Therefore, to get a complete procedure we must deal with flex-flex pairs, like is done in (Levy, 1996).

Word Unification is the problem of solving equations between words containing variables denoting words. The problem is decidable (Makanin, 1977), and in many aspects is quite similar to SOU. For instance, instances of variables may *overlap* with other occurrences of the same variable, and rise to an infinite number of most general solutions. For instance, the word unification equation $X \cdot a \stackrel{?}{=} a \cdot X$, has infinitely many solutions of the form $[X \mapsto a \cdot a \ldots a]$.

Non-termination of pre-unification is shown by the problem $X(f(a)) \stackrel{?}{=} f(X(a))$, that requires n imitation steps to generate the unifier $[X \mapsto \lambda y\,.\,f^n(y)]$. Notice that this problem is quite similar to the previous word unification problem $X \cdot a \stackrel{?}{=} a \cdot X$. To prove decidability of BSOU Schmidt-Schauß (2004) used the same technique as in word unification (see Makanin, 1977). He proved a *exponent of periodicity lemma* (see Schmidt-Schauß, 2004, Lemma 4.1), that states that we can (exponentially) bound the value of exponents in solutions without compromising solvability.

The possibility to choose between imitation and projection is also the responsible of NP-hardness of all variants of second-order unification. Typically (see Schmidt-Schauß, 2004), NP-hardness is proved by encoding 1-in-3SAT. We can interpret $[X \mapsto \lambda x\,.\,x]$ as false and $[X \mapsto \lambda x\,.\,f(x)]$ as true, and then encode clauses $X \vee Y \vee Z$ as $X(Y(Z(a))) \stackrel{?}{=} f(a)$, with the additional equation $X(f(a)) \stackrel{?}{=} f(X(a))$, in the case of BSOU and SOU, to ensure that all variables use their arguments.

4 Monadic Second-Order Unification

Farmer (1991) extended Golfarb's proof to prove that SOU is undecidable even if we restrict all second-order variables to be unary. Golfarb's proof requires the use of at least a binary function symbol, and this use is a crucial fact: Farmer (1988) also proved that SOU is decidable if all function symbols are at most unary (i.e. all constants are 0-ary or 1-ary), even if we do not restrict the arity of second-order variables. This fragment of SOU is call *Monadic Second-Order Unification (MSOU).*

Completeness of SOU (and BSOU and LSOU) decision procedures ensure that all constants occurring in a most general unifier, already occur in the original problem.[1] On the contrary, even if all original variables are unary, like in $X(a) \stackrel{?}{=} X(b)$, to represent the most general unifier $[X \mapsto \lambda x \,.\, Z(x,b), Y \mapsto \lambda y \,.\, Z(a,y)]$ we can require non-unary variables.

Levy and Villaret (2002, 2009) proved that the general forms of SOU, BSOU and LSOU can be reduced to a restricted form with just one binary function symbol. This is done by some sort of curryfication where terms like $f(g(a,b),c)$ are translated into $@(@(f,@(@(g,a),b)),c)$.

Levy et al. (2004, 2008) characterized the complexity of MSOU as NP-complete. To prove that MSOU is in NP, they showed how, for any solvable set of equations, we can represent at least one of the unifiers (in fact all size-minimal unifiers) in polynomial space. Then, they proved that we can check if a substitution (written in such representation) is a solution in polynomial time. There are two key in this proof: One is the result on the exponential upper bound on the exponent of periodicity of size-minimal unifiers. This upper bound allows us to represent exponents in linear space. The other key is a result of Plandowski (1994) where he proves that, given two context-free grammars with just one rule for every non-terminal symbol, we can check if they define the same language in polynomial time on the size of the grammars. This comprehension techniques, using context-free grammars generating singleton languages, and later using so called *tree context grammars* (Levy et al., 2006a), have been used to generalize matching and unification on compressed terms (see Gascón et al., 2008, 2009, for instance).

5 Two Occurrences per Variable

Word unification is trivially solvable when variables do not occur more than twice. In this case, when we apply a sort of imitation rule:

$$\textsf{Imitation:} \quad \{X \cdot w_1 \stackrel{?}{=} a \cdot w_2\} \cup E\} \Rightarrow \Big(\{X' \cdot w_1 \stackrel{?}{=} w_2\} \cup E\}\Big)\,\rho \quad \text{where } \rho = [X \mapsto a \cdot X']$$

the size of the problem does not increase. Notice that we remove an occurrence of a, and ρ introduces another occurrence of a, when instantiates the other occurrence of X. Similarly, Levy (1996) proves that the size of the problem does not increase during the execution of the LSOU procedure, when variables do not occur more than twice (see Levy, 1996, Theorem 3). Therefore, LSOU and CU are (trivially) decidable when no variable occurs more than twice. However, this is not the case for SOU.

Levy (1998) proved that SOU unification is undecidable even when no second-order variable occurs more than twice. The key in this proof is the observation that reachability can be encoded as SOU:

Given a ground term rewriting system $\{t_i \to u_i\}$, we can rewrite v into w, noted

$$t_1 \to u_1, \ldots, t_m \to u_m \vdash v \to w$$

if, and only if, the following SOU equation

$$X(f(a,v), u_1, \ldots, u_m) \stackrel{?}{=} f(X(a, t_1, \ldots, t_m), w)$$

is solvable, where f and a are symbols not used in the signature of the rewriting system.

[1] Alternatively, if a unifier contains constants not occurring in the original problem, we can replace them by fresh second-order variables with the same arity, obtaining a more general unifier.

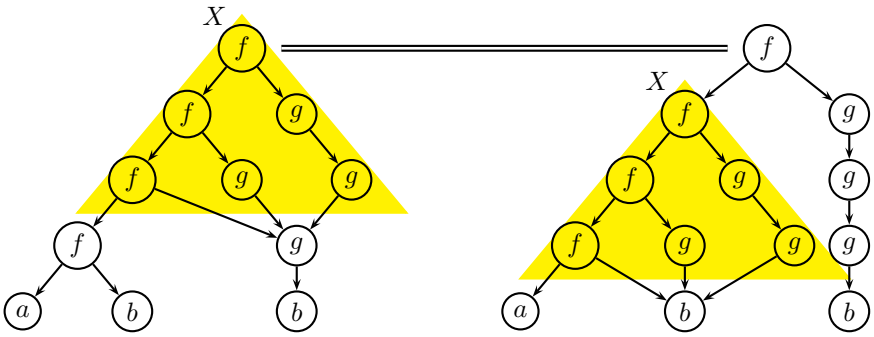
For instance, for the derivation:

$$b \to g(b) \vdash b \to g(b) \to g(g(b)) \to g(g(g(b)))$$

the equation would be:

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

and the instantiation of the equation as follow:



In this encoding, the number of times that a rewriting rule is used in the derivation corresponds to the number of times that the instance of X uses the corresponding argument. In fact, the unifier, if exists, encodes the sequence of rewriting steps. Ground reachability is a decidable problem, so this reduction does not proves undecidability of SOU.

If we allow the use of (first-order) variables in rewriting equations, but we restrict them to be instantiated always the same way, then we get a more general problem called *Rigid Reachability*. Notice that the rewriting rule $X \to a$ allows us to rewrite $f(b,c)$ into $f(a,a)$. However, $X \to a \vdash f(b,c) \to f(a,a)$ is an unsolvable rigid reachability problem (we cannot simultaneously instantiate X by b, and by c). If we allow the use of several rigid reachability equations we have *Simultaneous Rigid Reachability*. The undecidable of this problem was proved by Degtyarev and Voronkov (1996) by reduction of SOU. They consider a special kind of SOU equations, called *interpolation equations*, of the form $F(t_1, \ldots, t_n) \stackrel{?}{=} u$, where neither t_i nor u contain occurrences of second-order variables. By simplifying equations, and replacing subterms of the form $F(t_1, \ldots, t_n)$ by fresh first-order variables X and the equation $F(t_1, \ldots, t_n) \stackrel{?}{=} X$, we can express any SOU problem as a set of interpolation equations, without increasing the number of second-order variables, or its number of occurrences. Then, in Degtyarev and Voronkov's reduction, every interpolation equation $F(t_1, ..., t_n) \stackrel{?}{=} u$ is translated as $a_1 \to t_1, \ldots, a_n \to t_n \vdash X \to u$, where X is fresh.

We can reduce rigid reachability to SOU, like its is done for reachability. However, in this case we need some additional equations. For instance, the following rigid equation

$$c \to X \vdash g(c,c) \to g(d,e)$$

is unsolvable, whereas the corresponding SOU problem

$$F(f(a, g(c,c)), X) \stackrel{?}{=} f(F(a,c), g(d,e))$$

has a solution $[F \mapsto \lambda x, y \,.\, y][X \mapsto f(c, g(d,e))]$.

This problem can be easily fixed by adding a pair of equations $X \stackrel{?}{=} G_x(f_1(\overrightarrow{Y_x}), \ldots, f_N(\overrightarrow{Y_x})), b \stackrel{?}{=} G_x(b, \ldots, b)$, for all u_i that are variables X, where

$f_1, \ldots, f_N$ are all the function symbols of the rigid reachability signature. Notice that these additional equations disable X from being instantiated by a term with f in the head. Alternatively (see Levy and Veanes, 2000), we can add the reachability equation $f_1(a, \ldots, a) \to a, \ldots, f_N(a, \ldots, a) \vdash X \to a$, for any variable X, to ensure that instances of X do not contain f. This extension of Simultaneous Rigid Reachability is call *Guarded Simultaneous Rigid Reachability*. Guarded Simultaneous Rigid Reachability can be polynomially reduced to SOU where every second-order variable occurs twice. Levy and Veanes (1998) proved that Guarded Simultaneous Rigid Reachability is undecidable, even when restricted to systems of at most *two* reachability constraints. Therefore, SOU is undecidable under the following restrictions: there are at most two distinct second-order variables and two equations, every second-order variable occurs at most twice and in only one of the equations.

6 Just One Second-Order Variable

The number of different second-order variables in SOU plays a minor role compared to the total number of occurrences of second-order variables. Levy and Veanes (2000) present a straightforward reduction of arbitrary systems of second-order equations to equations using just one second-order variable and additional first-order variables.

Assume we have a system of SOU interpolation equations

$$\bigcup_{1 \le i \le m} \bigcup_{1 \le j \le k_i} F_i(t^1_{ij}, \ldots, t^n_{ij}) \stackrel{?}{=} u_{ij},$$

where $\{F_1, \ldots, F_m\}$ is the set of pair-wise distinct second-order variables, and every variable F_i has k_i occurrences. Assume without loss of generality that the arities of the F_i's are equal to n. If some F_i has smaller arity then increase the arity of F_i to the maximal arity replacing each $F_i(t^1_{ij}, \ldots, t^n_{ij})$ by $F_i(t^1_{ij}, \ldots, t^m_{ij}, \ldots, t^m_{ij})$, i.e. repeating the last argument as necessary. Then, we can reduce these equations to

$$\bigcup_{1 \le i \le m} \bigcup_{1 \le j \le k_i} G(t^1_{ij}, \ldots, t^n_{ij}) \stackrel{?}{=} g(\underbrace{\_, \ldots, \_}_{i-1}, u_{ij}, \underbrace{\_, \ldots, \_}_{m-i}),$$

where G is a fresh variable, g an appropriate constant, and "$\_$" denotes fresh and distinct first-order variables. If some of the t^k_{ij} is a variable, then we add also $G(c, \ldots, c) \stackrel{?}{=} g(\_, \ldots, \_)$. This reduction increases the number of occurrences of second-order variables in at most one. Notice also that the maximal arity of second-order variables and their set of arguments are preserved.

Combining this reduction with the results in previous section, we obtain that SOU is undecidable even for problems containing only one second-order variable occurring 4 times.

7 Restricting the Form of Arguments

One way to get a partial decidability result for SOU is to restrict the form of arguments of second-order variables. Levy and Veanes (2000, Corollary 15) proved that, even if we restrict arguments to be ground terms, SOU is undecidable. We have already seen that instances of second-order variables may "encode" sequences of transformations obtained from a rewriting system. The idea is to encode the execution steps of a universal Turing Machine, and reduce this way the Halting Problem. We represent the execution as a sequence of pairs of states

$$((v_1, v_1^+), (v_2, v_2^+), \ldots, (v_k, v_k^+)$$

where v^+ represents a successor state of v in the Turing Machine. Then, we use two equations, the first one has the form $F(\bar{t}, f(b,a)) \stackrel{?}{=} f(X, F(\overline{u}, a))$. It ensures that any solution satisfies $v_{i+i} = v_i^+$, because the instance of $F(\bar{t}, f(t', a))$ encodes $(v_1, \ldots, v_k, b)$, the instance of $f(X, F(\overline{u}, a))$ encodes $(X, v_1^+, \ldots, v_k^+)$, and the instance of X encodes the initial state. The second equation has the form $G(\bar{l}, f'(a, a')) \stackrel{?}{=} f'(F(\overline{v}, a), G(\overline{r}, a'))$. It ensures that v_i^+ is a valid successor of v_i in the TM. The sequence $\bar{l}$ and $\overline{r}$ encode the transitions of the Turing Machine, whereas $\bar{t}$, $\overline{u}$ and $\overline{v}$ only depends on the alphabet of the Turing Machine.

We obtain this way a system

$$\{F(\bar{t}, f(b,a)) \stackrel{?}{=} f(X, F(\overline{u}, a)), G(\bar{l}, f'(a, a')) \stackrel{?}{=} f'(F(\overline{v}, a), G(\overline{r}, a'))\}$$

that encodes the Halting Problem of a universal Turing Machine on input X. This proves undecidability of SOU for 5 occurrences of one second-order variable, even when the variable is only applied to ground terms.

A way to make SOU, or in general Higher-Order Unification, decidable is to restrict arguments of variables to be lists of pairwise distinct bound variables. This fragment is called *Higher-Order Patterns* and was proved decidable by Miller (1991). This restriction makes Higher-Order Pattern Unification equivalent to some sort of First-Order Unification with bindings and some kind of variable-capture restriction. In fact this extension of First-Order Unification has been studied extensively, and is called *Nominal Unification*. Urban et al. (2003) proved that Nominal Unification is decidable, and Levy and Villaret (2008) proved that it is in fact quadratic, by quadratic reduction to Higher-Order Patters Unification, that Qian (1996) proved decidable in linear time.

As we said, Jez (2014) decidability proof for CU closed a question open for more than 20 years. During this time, many other partial positive answers for CU have been found. For instance, the one-variable case (Gascón et al., 2010), the stratified case, used to prove decidability of unification modulo distributivity (Schmidt-Schauß, 2002; Levy et al., 2006b, 2011), the well-nested case, used in computational linguistics (Levy et al., 2005), the left-hole case, used to prove decidability of sequence unification (Kutsia et al., 2007, 2010), etc.

References

Comon, H., 1993. Completion of rewrite systems with membership constraints, part I: Deduction rules and part II: Constraint solving. Tech. rep., CNRS and LRI, Université de Paris Sud, (To appear in J. of Symbolic Computation).

de Groote, P., 2000. Linear higher-order matching is NP-complete. In: Proc. of the 11th Int. Conf. on Rewriting Techniques and Applications (RTA'00). Vol. 1833 of LNCS. pp. 127–140.

Degtyarev, A., Voronkov, A., 1996. The undecidability of simultaneous rigid E-unification. Theoretical Computer Science 166 (1-2), 291–300.

Farmer, W. M., 1988. A unification algorithm for second-order monadic terms. Annals of Pure and Applied Logic 39, 131–174.

Farmer, W. M., 1991. Simple second-order languages for wich unification is undecidable. Theoretical Computer Science 87, 173–214.

Gascón, A., Godoy, G., Schmidt-Schauß, M., 2008. Context matching for compressed terms. In: Proc. of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS'08). pp. 93–102.

Gascón, A., Godoy, G., Schmidt-Schauß, M., 2009. Unification with singleton tree grammars. In: Proc. of the 20th Int. Conf on Rewriting Techniques and Applications (RTA'09). Vol. 5595 of LNCS. pp. 365–379.

Gascón, A., Godoy, G., Schmidt-Schauß, M., Tiwari, A., 2010. Context unification with one context variable. J. Symb. Comput. 45 (2), 173–193.

Goldfarb, W. D., 1981. The undecidability of the second-order unification problem. Theoretical Computer Science 13, 225–230.

Gould, W. E., 1966. A matching procedure for ω-order logic. Ph.D. thesis, Princeton Univ.

Huet, G., 1973. The undecidability of unification in third-order logic. Information and Control 22 (3), 257–267.

Huet, G., 1975. A unification algorithm for typed λ-calculus. Theoretical Computer Science 1, 27–57.

Jensen, D. C., Pietrzykowski, T., 1976. Mechanizing ω-order type theory through unification. Theoretical Computer Science 3, 123–171.

Jez, A., 2014. Context unification is in PSPACE. In: Proc. of the 41st Int. Col. on Automata, Languages and Programming (ICAL'14).

Kutsia, T., Levy, J., Villaret, M., 2007. Sequence unification through currying. In: Proc. of the 18th Int. Conf. on Rewriting Techniques and Applications, RTA'07. Vol. 4533 of LNCS. Springer, pp. 288–302.

Kutsia, T., Levy, J., Villaret, M., 2010. On the relation between context and sequence unification. J. Symb. Comput. 45 (1), 74–95.

Levy, J., 1996. Linear second-order unification. In: Proc. of the 7th Int. Conf. on Rewriting Techniques and Applications (RTA'96). Vol. 1103 of LNCS. pp. 332–346.

Levy, J., 1998. Decidable and undecidable second-order unification problems. In: Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications (RTA'98). Vol. 1379 of LNCS. Tsukuba, Japan, pp. 47–60.

Levy, J., Niehren, J., Villaret, M., 2005. Well-nested context unification. In: Proc. of the 20th Int. Conf. on Automated Deduction, CADE-20. Vol. 3632 of LNCS. pp. 149–163.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2004. Monadic second-order unification is *np*-complete. In: Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04). Vol. 3091 of LNCS. Aachen, Germany, pp. 55–69.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2006a. Bounded second-order unification is NP-complete. In: Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA'06). Vol. 4098 of LNCS. pp. 400–414.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2006b. Stratified context unification is np-complete. In: Proceedings of the 3rd International Conference on Automated Reasoning (IJCAR 2006). Vol. 4130 of LNCS. Seattle, USA, pp. 82–96.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2008. The complexity of monadic second-order unification. SIAM Journal on Computing 38 (3), 1113–1140.

Levy, J., Schmidt-Schauß, M., Villaret, M., 2011. On the complexity of bounded second-order unification and stratified context unification. Logic Journal of the IGPL 19 (6), 763–789.

Levy, J., Veanes, M., 1998. On unification problems in restricted second-order languages. In: Annual Conference of the European Association for Computer Science Logic (CSL'98).

Levy, J., Veanes, M., 2000. On the undecidability of second-order unification. Information and Computation 159, 125–150.

Levy, J., Villaret, M., 2000. Linear second-order unification and context unification with tree-regular constraints. In: Proc. of the 11th Int. Conf. on Rewriting Techniques and Applications (RTA'00). Vol. 1833 of LNCS. pp. 156–171.

Levy, J., Villaret, M., 2002. Currying second-order unification problems. In: Proc of the 13th Int. Conf. on Rewriting Techniques and Applications (RTA'02). Vol. 2378 of LNCS. pp. 326–339.

Levy, J., Villaret, M., 2008. Nominal unification from a higher-order perspective. In: Proc. of the 19$^{\text{th}}$ Int. Conf on Rewriting Techniques and Applications, RTA'08. Vol. 5117 of LNCS. pp. 246–260.

Levy, J., Villaret, M., 2009. Simplifying the signature in second-order unification. Appl. Algebra Eng. Commun. Comput. 20 (5-6), 427–445.

Lucchesi, C. L., 1972. The undecidability of the unification problem for third-order languages. Tech. Rep. CSRR 2059, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo.

Makanin, G. S., 1977. The problem of solvability of equations in a free semigroup. Math. USSR Sbornik 32 (2), 129–198.

Miller, D., 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. J. of Logic and Computation 1 (4), 497–536.

Pietrzykowski, T., 1973. A complete mechanization of second-order logic. J. of the ACM 20 (2), 333–364.

Plandowski, W., 1994. Testing equivalence of morphisms in context-free languages. In: ESA'94. Vol. 855 of LNCS. pp. 460–470.

Qian, Z., 1996. Unification of higher-order patterns in linear time and space. J. of Logic and Computation 6 (3), 315–341.

Schmidt-Schauß, M., 1995. Unification of stratified second-order terms. Tech. Rep. 12/94, Johan Wolfgang-Goethe-Universität, Frankfurt, Germany.

Schmidt-Schauß, M., 2002. A decision algorithm for stratified context unification. Journal of Logic and Computation 12, 929–953.

Schmidt-Schauß, M., 2004. Decidability of bounded second order unification. Information and Computation 188 (2), 143–178.

Urban, C., Pitts, A. M., Gabbay, M. J., 2003. Nominal unification. In: Proc. of the 17th Int. Work. on Computer Science Logic, CSL'03. Vol. 2803 of LNCS. pp. 513–527.