International Conference on Rewriting Techniques and Applications 2010 (Edinburgh), pp. 103-118
http://rewriting.loria.fr/rta/

THE UNDECIDABILITY OF TYPE RELATED PROBLEMS IN TYPE-FREE STYLE SYSTEM F

KEN-ETSU FUJITA [1] AND ALEKSY SCHUBERT [2]

[1] Gunma University, Tenjin-cho 1-5-1, Kiryu 376-8515, Japan
E-mail address: fujita@cs.gunma-u.ac.jp

[2] The University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
E-mail address: alx@mimuw.edu.pl

Abstract. We consider here a number of variations on the System F, that are predicative second-order systems whose terms are intermediate between the Curry style and Church style. The terms here contain the information on where the universal quantifier elimination and introduction in the type inference process must take place, which is similar to Church forms. However, they omit the information on which types are involved in the rules, which is similar to Curry forms. In this paper we prove the undecidability of the type-checking, type inference and typability problems for the system. Moreover, the proof works for the predicative version of the system with finitely stratified polymorphic types. The result includes the bounds on the Leivant's level numbers for types used in the instances leading to the undecidability.

1. Introduction

The type systems can be viewed as formalisms in which reasonable properties of computational entities can be precisely formulated. The traditional approach distinguishes two kinds of computational entities considered in connection with type systems, i.e., Church-style λ-terms and Curry-style ones. The System F ($\lambda 2$) of Girard-Reynolds, however, allows of introducing intermediate families of terms between Church-style and Curry-style.

From the functional programming perspective, the Curry-style terms are interesting since they can serve as a useful notation to define programs with little notational overhead. However, more typing information can make the process of the program understanding easier. In order to broaden the scope of different compromise choices between these extremes, it is useful to study intermediate systems with different amounts of notational burden.

One family of such terms is the family of domain-free terms. It arose in [HS99] as a good target language for CPS transformations. This style corresponds to removing from the

1998 ACM Subject Classification: F.4.1.

Key words and phrases: Lambda calculus and related systems, type checking, typability, partial type inference, 2nd order unification, undecidability, Curry style type system, Church style type system, finitely stratified polymorphic types.

[2] This work was partly supported by the Polish government grant no N N206 355836.

Proceedings of the 21st International Conference on Rewriting Techniques and Applications, Edinburgh, July, 2010
Editors: Christopher Lynch
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.RTA.2010.103

terms the information on which types were involved in the application of the λ abstraction rule. Type systems with quantification permit also removing the information on which types are involved in places where the quantification rules are used, but to leave the trace of the quantification rule itself. This gives rise to *type-free systems* which we consider here.

The type-checking, type inference and typability problems were thoroughly studied for the polymorphic lambda calculus and its various fragments and variations, e.g. [Boe85, Pfe93, Wel99, Sch98, FS00, KTU90b, Mai90]. These and other studies show that even a little bit more polymorphism than in ML gives rise to an undecidable system. It is, however, still unknown if the polymorphic lambda calculus in the predicative formulation (i.e. a system where types are divided into levels and a variable of a particular level can be substituted for by types of a lower level), the so called finitely stratified polymorphic lambda calculus [Lei91], is decidable with this regard.[1] The current paper gives a bound on the undecidable cases of the predicative system. The proof of the undecidability for the type-free $\lambda 2$, we present here, works in the system with limited level number (level 1 in case of type-checking and type inference, and level 2 in case of typability). This proof, however, cannot be adapted easily to the Curry-style $\lambda 2$ as it is based on a reduction from second-order unification, while the Curry-style requires a technique which is based on semiunification [Wel99, KTU90a].

Although this papers gives negative results, it provides an interesting perspective to the investigation of type systems as it presents a system with light-weight annotation burden, but which uses second-order unification as the basis for type reconstruction instead of the semiunification. This has the advantage that the research on higher-order unification is much more active and gives more opportunities for type systems with decidable type reconstruction procedures.

This paper is structured as follows. We introduce the type systems to deal with in Section 2. Section 3 is devoted to the undecidability of the restricted 2nd order unification problem. The undecidability proof for the type-free $\lambda 2$ systems is presented in Section 4.

2. Second-order Type Systems

We consider second-order polymorphic systems $\lambda 2$ of Girard-Reynolds in the type-free style.

2.1. Polymorphic System $\lambda 2$ (System F)

The polymorphic system $\lambda 2$ (System F) employs the connective $\to$ and second-order universal quantification $\forall$ to form expressions called $\lambda 2$-types:

$$A ::= X \mid (A \to A) \mid \forall X.A$$

The contexts of the system (written as $\Gamma, \Gamma', \Gamma_1$ etc.) are as usual finite mappings from term variables to types. The domain $\mathrm{Dom}(\Gamma)$ of a context Γ is the set of the term variables the context assigns types to. The terms of $\lambda 2$ in the *type-free style* are defined as follows. The

[1] To be precise, the construction of Wells [Wel99] gives rise to undecidable type-checking in the Leivant's level 2, the remaining cases i.e. type inference and typability, in general, and type-checking in level 1 are to our knowledge open.

system is an intermediate system between Church and Curry styles, where the symbols [] and Λ mark applications of the $\forall$ rules without the type information in terms.

$$M ::= x \mid (\lambda x.M) \mid (MM) \mid (\Lambda.M) \mid (M[])$$

The inference rules are as follows:

$$\frac{}{\Gamma, x{:}A \vdash_{\lambda 2\mathrm{tf}} x : A}\ (\mathrm{var})$$

$$\frac{\Gamma, x{:}A_1 \vdash_{\lambda 2\mathrm{tf}} M : A_2}{\Gamma \vdash_{\lambda 2\mathrm{tf}} \lambda x.M : A_1 \to A_2}(\to I) \quad \frac{\Gamma \vdash_{\lambda 2\mathrm{tf}} M_1 : A_1 \to A_2 \quad \Gamma \vdash_{\lambda 2\mathrm{tf}} M_2 : A_1}{\Gamma \vdash_{\lambda 2\mathrm{tf}} M_1 M_2 : A_2}(\to E)$$

$$\frac{\Gamma \vdash_{\lambda 2\mathrm{tf}} M : A}{\Gamma \vdash_{\lambda 2\mathrm{tf}} \Lambda.M : \forall X.A}\ (\forall I)^\star \qquad \frac{\Gamma \vdash_{\lambda 2\mathrm{tf}} M : \forall X.A}{\Gamma \vdash_{\lambda 2\mathrm{tf}} M[] : A[X := A_1]}\ (\forall E)$$

where $(\forall I)^\star$ denotes the eigenvariable condition $X \notin \mathrm{FV}(\Gamma)$.

The predicative version of the system divides the type variables into levels [Lei91].[2] We write $X^{(k)}$ to mark that X is in the level k. Then, the types in the level 0 and k for $k > 0$ are defined as follows:

$$\begin{aligned} A^{(0)} &::= X^{(0)} \mid (A^{(0)} \to A^{(0)}) \\ A^{(k)} &::= X^{(k)} \mid A^{(k-1)} \mid (A^{(k)} \to A^{(k)}) \mid \forall X^{(k-1)}.A^{(k)} \end{aligned}$$

The minimum level of a type A is denoted as $\mathsf{lvl}(A)$. Now, the rules of the predicative type-free $\lambda 2$ are as usual except the rules for the quantification:

$$\frac{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{tf}} M : A}{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{tf}} \Lambda.M : \forall X^{(k)}.A}\ (\forall I)^\star \quad \frac{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{tf}} M : \forall X^{(k)}.A \quad \mathsf{lvl}(A_1) \le k}{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{tf}} M[] : A[X^{(k)} := A_1]}\ (\forall E)$$

where $(\forall I)^\star$ denotes the eigenvariable condition $X \notin \mathrm{FV}(\Gamma)$.

We say that a derivation of a judgement is done within level k, if for every judgement $\Gamma \vdash M : A$ in the derivation, we have $\mathsf{lvl}(A) \le k$ and $\mathsf{lvl}(B) \le k$ for all types B occurring in Γ.

On the other hand, $\lambda 2$ in the Curry style has well-known rules as follows:

$$\frac{\Gamma \vdash_{\lambda 2\mathrm{C}} M : A}{\Gamma \vdash_{\lambda 2\mathrm{C}} M : \forall X.A}\ (\forall I)^\star \qquad \frac{\Gamma \vdash_{\lambda 2\mathrm{C}} M : \forall X.A}{\Gamma \vdash_{\lambda 2\mathrm{C}} M : A[X := A_1]}\ (\forall E)$$

A predicative version of $\lambda 2$ in the Curry style is also defined for stratified types:

$$\frac{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{C}} M : A}{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{C}} M : \forall X^{(k)}.A}\ (\forall I)^\star \quad \frac{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{C}} M : \forall X^{(k)}.A \quad \mathsf{lvl}(A_1) \le k}{\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{C}} M : A[X^{(k)} := A_1]}\ (\forall E)$$

Then a type erasing map from terms in type-free style to those in Curry style is naturally defined so that $|\Lambda.M| = |M|$ and $|M[]| = |M|$. A type flatting map from stratified types to non-stratified ones is also defined so that $|X^{(k)}| = X$ and $|\forall X^{(k)}.A| = \forall |X^{(k)}|.|A|$. We have the following basic properties between Curry and type-free styles; and stratified and flatted types.

Proposition 2.1 (Erasing, lifting, flatting)**.** *Let a pair of systems* $(X, Y) = (\lambda 2\mathrm{tf}, \lambda 2\mathrm{C})$ *or* $(\mathrm{p}\lambda 2\mathrm{tf}, \mathrm{p}\lambda 2\mathrm{C})$*. Let* $(Z, W) = (\mathrm{p}\lambda 2\mathrm{tf}, \lambda 2\mathrm{tf})$ *or* $(\mathrm{p}\lambda 2\mathrm{C}, \lambda 2\mathrm{C})$*.*

[2]Levels should not be confused with type ranks [Lei83, KW94] defined as: $\mathrm{rank}(A) = 0$ when A contains no quantifier, $\mathrm{rank}(A_1 \to A_2) = \max(\mathrm{rank}(A_1) + 1, \mathrm{rank}(A_2))$, and $\mathrm{rank}(\forall X.A) = \mathrm{rank}(A)$. For instance, $(\forall X.(X \to X)) \to \forall Y.Y$ has rank 2, but has level 1 if $\mathsf{lvl}(X) = \mathsf{lvl}(Y) = 0$. Note also that the level here is different from the stratified level in [GR94].

Term\System	pλ2tf	λ2tf	pλ2C	λ2C
$(\lambda x.xx)(\lambda x.xx)$	No	No	No	No
$(\lambda x.xx)(\lambda x.xyx)$	No	No	No	Yes
$\lambda x.xx$	No	No	Yes	Yes
$\lambda x.x[]x$	No	Yes	-	-
$\lambda x.x[](x[])$	Yes	Yes	-	-

Figure 1: Typability for example terms in type-free and Curry style disciplines

(1) *If* $\Gamma \vdash_X M : A$ *then* $\Gamma \vdash_Y |M| : A$.
(2) *If* $\Gamma \vdash_Y M : A$ *then there exists an* X*-term* N *such that* $|N| \equiv M$ *and* $\Gamma \vdash_X N : A$.
(3) *If* $\Gamma \vdash_Z M : A$ *then* $\Gamma \vdash_W M : |A|$.

Notice that the systems λ2tf and pλ2tf, unlike λ2C and pλ2C, are syntax directed in the sense that the form of a term determines exactly which rule should be the last one in a derivation of its type. Still, in case of the $(\forall E)$, we do not know which types are used in the particular instance of the rule.

We show simple examples of terms in the systems on Figure 1, where "Yes" means typable but "No" untypable in the corresponding system (the second term is due to Urzyczyn [Lei91]).

The *type checking problem* (TCP) is the problem: given a term M, a type A, and a context Γ, is $\Gamma \vdash M : A$ derivable? The *type inference problem* (TIP) is the problem: given a term M and a context Γ, is there a type A such that $\Gamma \vdash M : A$ is derivable? Finally, the *typability problem* (TP) is the problem: given a term M, are there a context Γ and a type A such that $\Gamma \vdash M : A$ is derivable?

Note that in case of the predicative system we may consider each of the problems above in two versions: non-bounded ones (TCP, TIP, TP) in which we ask about the derivability in the predicative system in general and k-bounded ones (TCP^k, TIP^k, TP^k) in which we are given data that falls within the level k and want an answer if a derivation can be done within the level k exclusively.

In general, there is no direct relation between the undecidability of TCP (or TIP, or TP) and the undecidability of TCP^k. In particular the instances of TCP have no fixed level so the identical translation does not give the undecidability. The reduction in the other direction is not straightforward either as the systems do not enjoy the conservativity property.

Proposition 2.2. *For each* k *there are context* Γ*, term* M*, and a type* A *such that* Γ *and* A *are in level* k*,* $\Gamma \vdash_{\mathrm{p}\lambda\mathrm{2tf}} M : A$ *is derivable, but there is no derivation for* $\Gamma \vdash_{\mathrm{p}\lambda\mathrm{2tf}} M : A$ *in level* k.

Proof. We adapt the terms proposed by Urzyczyn (see [Lei91]). Let the term $M_1 \equiv \lambda x.x[](x[])$ and $M_{k+1} \equiv \lambda y.y[]M_k(y[])$. The term M_{k+1} is typable in level $k+1$ but is not typable in level k. Consider now the judgement $\vdash (\lambda x.\lambda y.y)M_{k+1} : X^{(k)} \to X^{(k)}$. This term is typable in level $k+1$ only, but is not typable in level k while the context and the type $X^{(k)}$ are in level k. ∎

However, a slightly weaker version of conservativity indeed holds for the systems. We say that a type-free $\lambda 2$ term M is in normal form if it has one of the following shapes:[3]

(1) $xM_1 \dots M_n$ $(n \geq 0)$ where M_i is either a term in normal form or [], or
(2) $\lambda x.M$ where M is in normal form, or
(3) $\Lambda.M$ where M is in normal form.

Proposition 2.3. *Let M be a term in normal form and Γ, A be in level k. If $\Gamma \vdash_{\mathrm{p}\lambda 2\mathrm{tf}} M : A$ is derivable then it is derivable in level k.*

Proof. Induction on the term M in normal form. ∎

2.2. Connections with Partial Type Reconstruction

The problem of partial type reconstruction is a crucial problem for functional programming languages. Consider the following situation. We would like to write a generic function which transforms an existing polymorphic function so that the resulting function prints out some textual information. Currently, a definition of such a function could look like as follows (in the syntax of OCaml)

```
# let addInfo = fun f x -> print "we call polymorphic"; f x;;
```

However, this is not possible in functional languages such as OCaml as the functional arguments cannot be polymorphic. Still, extension of the language to make such definitions possible is apparently useful. One of the drawbacks of the notation above is that it breaks the current convention that an application of a function is always monomorphic. Therefore, it is useful to warn a programmer that a particular function is polymorphic, for instance in the following fashion:

```
# let addInfo = fun f x -> print "we call polymorphic"; (f[]) x ;;
```

Since the functional programming languages give the programmer a freedom of giving the typing information wherever it is suitable for readability or comprehensiveness. Therefore, the functional programming languages require not only the procedures to decide the type inference, but stronger procedures that tackle the partial type reconstruction problem where some of the typing information is provided and some is omitted.

Along the lines of Boehm [Boe85], Pfenning [Pfe93] has proved that the partial type reconstruction problem for the pure polymorphic $\lambda 2$ is equivalent to the second-order unification problem as far as decidability is concerned. This means the problem is undecidable. He also proved the undecidability result for the predicative system. Our problem TIP for type-free $\lambda 2$ can be regarded as a restricted instance of the partial type reconstruction problem in which the instances cannot contain types in terms, but may contain placeholders where types should be instantiated in type derivations. The question of undecidability for terms in this form has been mentioned in the paper [Pfe93] and we confirm here that the problem is undecidable. Note that this strategy is very practical as the additional typing information in the terms is provided very rarely in real functional programs.

In [FS00], we demonstrated that TIP is undecidable for the predicative domain-free $\lambda 2$ by a reduction from the second-order unification problem. The problem TIP here for type-free $\lambda 2$ is a very restricted form of both problems. In general the proof methods applied in [Pfe93, FS00] do not work for the problem for the type-free style, since the previous

[3]The study of appropriate notion of reduction goes beyond the main topic of the paper.

methods essentially refer to type information which is to be erased in the type-free case. Even the direct application of the unification for flat forms to the construction of Pfenning does not bring the main result of the current paper.

It is worth pointing out that the results of [Pfe93, Boe85] indicate that it is impossible to devise an algorithm which allows the authors of functional programs to freely insert and omit the type annotations in the full polymorphic type discipline. We strengthen the result in such a way that already the strategy of showing where the second-order polymorphism is used, but omitting the information on how it is used gives rise to undecidability. Still, we hope that the existing decidable cases of the second-order unification can bring systems with decidable type related problems and the resulting systems will have the pragmatic advantage that the uses of polymorphisms are clearly marked in the source code of programs.

3. Undecidability of Restricted Unification

The proofs of undecidability for type-free system are done as a reduction of a strongly restricted version of the 2nd order unification problem to the problems. The version of the unification has been introduced in [FS09, FS]. The proof of the main theorem of the section is in [FS09].

We define here expressions for unification problems. We assume that a countably infinite set of type variables with level k is divided into three subsets: The first one contains first-order variables denoted by $X, Y, \ldots$, the second one constants denoted by $C, D, \ldots$, and the third one second-order variables denoted by $\mathsf{F}^n, \mathsf{G}^n, \ldots$ where the superscript n indicates their arity. The expressions we deal with in unification equations have the following form:

$$A, B \quad ::= \quad X \mid C \mid (A \to B) \mid \mathsf{F}^n A_1 \cdots A_n$$

Whenever it does not lead to confusion, we drop the superscript with arity. The expressions which do not contain second-order variables are called *first-order expressions*.

An instance of the unification problem consists of a set of equations

$$E = \{A_1 \doteq B_1, \ldots, A_n \doteq B_n\}.$$

We say that the instance E is solvable if there exists a substitution S such that $S(A_1) = S(B_1), \ldots, S(A_n) = S(B_n)$. We write $\mathrm{Dom}(S)$ for the domain of S. A set of free variables in expressions of unification problems is defined as follows:

$$\mathrm{FV}_u(C) = \emptyset; \mathrm{FV}_u(X) = \{X\}; \mathrm{FV}_u(\mathsf{F}) = \{\mathsf{F}\}; \mathrm{FV}_u(A \to B) = \mathrm{FV}_u(A) \cup \mathrm{FV}_u(B).$$

We have to define a restricted second-order unification problem which can fit into the form of type constraints that arise in the type-free type system and is still undecidable. The basic idea here is to exploit the observation that the rules $(\forall E)$ work in a way similar to the application of a second-order expression to an argument. As the type-free systems omit from terms the information on an expression to which a second-order term is applied, we have to restrict the arguments of second-order variables so that the arguments can take up any form (monadic restriction below).

Definition 3.1 (Flat form)**.** An instance E of the unification problem is in *flat form* when it complies with all three restrictions below:

(1) *Root restriction*: Second-order variables occur only at root positions.

(2) *Monadic restriction*: If second-order variable occurs as $\mathsf{F}A_1 \cdots A_n$ then either all A_i are constants or all A_i are first-order variables. Moreover, the symbols A_i occur only in the equation where $\mathsf{F}A_1 \cdots A_n$ occurs.
(3) *Constant restriction*: Each time a second-order variable F is applied to a vector $X_1, \ldots, X_n$ of first-order variables as $\mathsf{F}X_1 \cdots X_n \doteq A$, there is a set of pairwise distinct constants $C_1, \ldots, C_n$ such that there is exactly one equation $\mathsf{F}C_1 \cdots C_n \doteq B \in E$, where $C_1, \ldots, C_n$ occur, all $C_1, \ldots, C_n$ occur in B, and the positions where the constants occur in B exist in A.

The idea similar to the one behind root restriction and monadic restriction can be found in [Ami90]. The constant restriction is necessary as it provides a clear indication on which positions correspond to the arguments of second-order variables. The presence of the constants can be simulated in the process of type inference by means of the $(\forall I)$ rules.

Theorem 3.2 (Undecidabiliy of unification with flat form)**.** *The problem of deciding if a given set of equations E in the flat form can be solved is undecidable.*

Proof. From a reduction of the unification of simple instances [Sch98, Sch01, LV00] to the unification of equations in the flat form. See [FS09, FS] for the details of the proof. ∎

The translation in the theorem above applied to the particular simple instances that give rise to the undecidability results in instances which contain 10 equations with second-order variables. Six of them involve one second-order variable F, while the other four another one G. To facilitate the understanding of the proofs below we present here the equations with second-order variables (omitting the other ones) that arise in the proof of Theorem 3.2:

$$\begin{array}{rcl}
\mathsf{F}X^1_1 \ldots X^1_n & \doteq & X_{\mathsf{F}A_1 \ldots A_n} \to A_1 \to \cdots \to A_n \to o \\
\mathsf{F}C^1_1 \ldots C^1_n & \doteq & X'_{\mathsf{F}A_1 \ldots A_n} \to C^1_1 \to \cdots \to C^1_n \to o \\
\mathsf{F}X^2_1 \ldots X^2_n & \doteq & X_{\mathsf{F}A'_1 \ldots A'_n} \to A'_1 \to \cdots \to A'_n \to o \\
\mathsf{F}C^2_1 \ldots C^2_n & \doteq & X'_{\mathsf{F}A'_1 \ldots A'_n} \to C^2_1 \to \cdots \to C^2_n \to o \\
\mathsf{F}X^3_1 \ldots X^3_n & \doteq & X_{\mathsf{F}A''_1 \ldots A''_n} \to A''_1 \to \cdots \to A''_n \to o \\
\mathsf{F}C^3_1 \ldots C^3_n & \doteq & X'_{\mathsf{F}A''_1 \ldots A''_n} \to C^3_1 \to \cdots \to C^3_n \to o \\
\mathsf{G}X^4_1 \ldots X^4_m & \doteq & Y_{\mathsf{G}B_1 \ldots B_m} \to B_1 \to \cdots \to B_m \to o \\
\mathsf{G}C^4_1 \ldots C^4_m & \doteq & Y'_{\mathsf{G}B_1 \ldots B_m} \to C^4_1 \to \cdots \to C^4_m \to o \\
\mathsf{G}X^5_1 \ldots X^5_m & \doteq & Y_{\mathsf{G}B'_1 \ldots B'_m} \to B'_1 \to \cdots \to B'_m \to o \\
\mathsf{G}C^5_1 \ldots C^5_m & \doteq & Y'_{\mathsf{G}B'_1 \ldots B'_m} \to C^5_1 \to \cdots \to C^5_m \to o
\end{array} \tag{3.1}$$

where F, G are fresh second-order variables of arity n, m, resp.; X with annotations and Y with annotations are fresh first-order variables; C^i_j with appropriate indices are subject to the constant restriction; and o is a distinguished type constant. We show here the equations in pairs (see case (3) in Definition 3.1).

4. Undecidability of Type Related Problems for Type-free System

Now, we embark on the reduction of the unification for the instances in the flat form to the type related problems. For a set of equations E we provide a λ-term M such that if a type derivation for M exists then a unifier S for E can be extracted from it. The main idea of the construction comes from [FS09, FS], and can be traced back to [Pfe93]. However, we

do not follow the latter construction in detail as we want to obtain a tight bound on type levels and avoid the occurrence of type variables in terms. The general idea of the approach here is that the shape of a type derived for a variable x_A occurring in M, related by the translation to a subexpression A in the set of equations E, strictly corresponds to the result of the substitution $S(A)$.

We simply write $A^n \to B$ for $A \to \cdots \to A \to B$ with n occurrences of A, and MN^n for $MN \cdots N$ with n applications to N.

Since we have a countably infinite set of term variables of λ-calculus, we can assume a one-to-one mapping between expressions of unification problems and term variables of λ-terms. Based on this, we write term variables x_A and y_A corresponding to an expression A of a unification problem. For instance, we have term variables x_X and y_X from a first-order variable X, and similarly term variables x_C and y_C from a constant C. In particular, the distinguished constant o gives rise to the term variable x_o.

4.1. Enforcing the Shape of First-order Terms

To achieve the goal sketched above, we have to provide a translation from instances of the second-order unification to terms which enforce particular forms of solutions. We start with enforcing of the shape of terms that do not involve second-order variables.

Terms which involve the variables associated with subexpressions of a given unification instance need to be put together in a single λ-term. To this end we use special variables that allow to glue terms. They are contained in a context

$$\Gamma_o = \{x_o : o, y_{o_1} : o \to o, \ldots, y_{o_n} : o^n \to o\}$$

for a fixed $n = 21$, where y_{o_i} are fresh term variables and o is the distinguished constant. We often shorten y_{o_i} to y_o when this does not lead to confusion. The following proposition holds.

Proposition 4.1. *There is a term M_o such that $\mathrm{dom}(\Gamma_o) \subseteq \mathrm{FV}(M_o)$ and for each Γ if $\Gamma \vdash M_o : A$ is derivable and Γ contains $x_o : o$ then $\Gamma_o \subseteq \Gamma$ and $A = o$.*

Moreover, if the level of o is $k \geq 0$ then the derivation can be done in level k.

Proof. Observe that a term $y_o x_o$ forces the argument of y_o to be of type o then $y_o(y_o x_o)$ forces also the result of y_o to be of type o so $y_o : o \to o$. The details of the construction and proof are left to the reader. ∎

The idea of enforcing employed in the sketch of the proof above can also be used below.

Definition 4.2 (Encoding of first-order expressions)**.** For a first-order expression A, we define a λ-term M_A, as follows:

(1) case $A \equiv X$ (first-order variable): the term $M_X \equiv y_o(y_X x_X)$,
(2) case $A \equiv C$ (constant): the term $M_C \equiv y_o(y_C x_C)$,
(3) case $A \equiv (A_1 \to A_2)$: the term
$$M_{A_1 \to A_2} \equiv y_o \left(y_{A_2}(x_{A_1 \to A_2} x_{A_1})\right) (y_{A_2} x_{A_2})\, M_{A_1}\, M_{A_2}.$$

Note that $\mathrm{FV}(M_A) \subseteq \mathrm{Dom}(\Gamma_o) \cup \{x_B, y_B \mid B \text{ is subexpression of } A\}$.

The encoding in Definition 4.2 is constructed in such a way that the types of variables x_A follow the structure of the corresponding expression A. In addition, the encoding gives enough freedom to enable the operation of first-order substitutions. This is precisely expressed by the following lemma.

Lemma 4.3. *For any substitution S, with $\Gamma \supseteq \Gamma_o$ defined by*

- $\Gamma(x_B) = S(B)$,
- $\Gamma(y_B) = S(B) \to o$

for all subexpressions B of A, we have that $\Gamma \vdash M_A : o$ is derivable.

Moreover, when variables in image of S are assigned level k then the derivation can be done in level k.

Proof. The proof is immediate by Definition 4.2 since $S(A_1 \to A_2) = S(A_1) \to S(A_2)$. ∎

We can now present the way how the first-order unification is encoded in λ-terms.

Definition 4.4 (Encoding of first-order unification)**.** Let E be a finite set of equations of first-order unification. In case $E = \emptyset$, we define $M_E \equiv x_o$. In case $E = \{A_1 \doteq B_1\} \cup E_0$, we define M_E to be:

$$M_E \equiv y_{o_5}\ (y_{A_1} x_{A_1})\ (y_{A_1} x_{B_1})\ M_{A_1}\ M_{B_1}\ M_{E_0}.$$

In the definition above, we use M_{A_1}, M_{B_1} to encode the shape of the expressions A_1, B_1 respectively. This is done in such a way that x_{A_1}, x_{B_1} have the types $S(A_1), S(B_1)$ for some substitution S. We can now force them to be equal by placing x_{A_1}, x_{B_1} as arguments to the same variable y_{A_1}. The rest of the set of equations can be taken into account in the same fashion in the subterm M_{E_0}. Note that

$$\mathrm{FV}(M_E) \subseteq \quad \{x_B, y_B \mid B \text{ is subexpression of some expression in } E\} \cup \mathrm{Dom}(\Gamma_o).$$

The lemma below relates the solutions of first-order unification with derivations in the type-free System F.

Lemma 4.5. *Let E be a finite set of equations of first-order unification and S a substitution. The substitution S solves E if and only if there is a context $\Gamma \supseteq \Gamma_o$ such that $\Gamma \vdash M_E : o$ is derivable in level k and $\Gamma(x_B) = S(B)$, $\Gamma(y_B) = S(B) \to o$ for each subexpression B of expressions in E.*

Proof. The proof is by induction on the size of E. The case $E = \emptyset$ holds trivially. Consider now the case $E = \{A_1 \doteq B_1\} \cup E_0$.

($\Rightarrow$) Suppose that E is solvable under S, i.e., $S(A_1) = S(B_1)$ and S solves E_0. From Lemma 4.3, we have that $\Gamma_S \vdash M_{A_1} : o$ in level k and $\Gamma'_S \vdash M_{B_1} : o$ are derivable for some $\Gamma_S, \Gamma'_S \supseteq \Gamma_o$. By induction hypothesis we have in addition that $\Gamma''_S \vdash M_{E_0} : o$ is derivable. We may assume that domains of $\Gamma_S, \Gamma'_S, \Gamma''_S$ are minimal, i.e., they contain the domain of Γ_o and the free variables in M_{A_1}, M_{B_1}, M_{E_0} respectively. In this case, the intersection of domains of Γ_S, Γ'_S, Γ''_S contains only the domain of Γ_o and the variables of the form x_B, y_B. Therefore, each of the contexts assigns the same type to the variables. In this light we can let $\Gamma = \Gamma_S \cup \Gamma'_S \cup \Gamma''_S$ and preserve the derivability of $\Gamma \vdash M_{A_1} : o$, $\Gamma \vdash M_{B_1} : o$, and $\Gamma \vdash M_{E_0} : o$. Since S solves E we have $\Gamma(x_{A_1}) = S(A_1) = S(B_1) = \Gamma(x_{B_1})$, and therefore $\Gamma \vdash y_{A_1} x_{A_1} : o$ and $\Gamma \vdash y_{A_1} x_{B_1} : o$ are derivable. Now, we can easily assemble the derivation for $\Gamma \vdash M_E : o$.

($\Leftarrow$) Suppose that $\Gamma \vdash M_E : o$ is derivable (in level k) for some context $\Gamma \supseteq \Gamma_o$ with $\Gamma(x_B) = S(B)$, $\Gamma(y_B) = S(B) \to o$. This means that $\Gamma \vdash M_{E_0} : o$ is derivable. The induction hypothesis gives immediately that S solves E_0. We have also that $\Gamma \vdash M_{A_1} : o$ and $\Gamma \vdash M_{B_1} : o$ are derivable. As y_{A_1} is applied to both x_{A_1} and x_{B_1}, we have $\Gamma(x_{A_1}) = \Gamma(x_{B_1})$. Then we can infer that $S(A_1) = \Gamma(x_{A_1}) = \Gamma(x_{B_1}) = S(B_1)$. Hence all $A \doteq B \in E$ are solvable under S. ∎

4.2. Enforcing the Shape of Terms with Second-order Variables

Now that we know how to translate the equations with no second-order variables, we have to provide a translation for the equations that contain the variables. In order to make the presentation more concrete, we provide here a translation for the particular equations that result from the translation of the simple-instances to the equations in the flat form. Recall that the translation gives two pairs of equations for each second-order variable and the undecidable instances of the second-order unification of simple instances contain two variables F and G. We provide the following encoding for equations with G only, displayed in (3.1) on page 107. The case with F follows exactly the same pattern. See also Definition 3.1 and Theorem 3.2. We refrain from presentation of a general definition for equations with second-order variables as such a presentation is involved and obscures the main idea of the encoding. We believe that it is easier for a reader to understand the idea of the proof when it is presented in this more concrete fashion.

As a useful shorthand, we define a λ-term $M[]^{n+1} \equiv (M[])[]^n$ and $M[]^0 \equiv M$, which means successive application of $(\forall E)$. We also define a λ-term $\Lambda^{n+1}.M \equiv \Lambda^n.(\Lambda.M)$ and $\Lambda^0.M \equiv M$, which means successive application of $(\forall I)$.

Definition 4.6 (Encoding of second-order unification)**.** For a set of equations E such that the set $E\backslash E'$ consists of the equations in the flat form containing the second-order variable G of arity n:

$$\mathsf{G}X_1 \cdots X_n \doteq B_1, \qquad \mathsf{G}C_1 \cdots C_n \doteq B_2, \qquad \mathsf{G}Y_1 \cdots Y_n \doteq B_3, \qquad \mathsf{G}C'_1 \cdots C'_n \doteq B_4,$$

where $B_1 \equiv (X \to A_1 \to \cdots \to A_n \to o)$, $B_3 \equiv (Y \to A'_1 \to \cdots \to A'_n \to o)$,
$B_2 \equiv (X' \to C_1 \to \cdots \to C_n \to o)$, and $B_4 \equiv (Y' \to C'_1 \to \cdots \to C'_n \to o)$;
we define a λ-term M_E as

$$\begin{array}{ll} y_o & (y_{B_1} x_{B_1})\ (y_{B_1}(x_{\mathsf{G}}[]^n))\ (y_{B_2} x_{B_2})\ (y_{B_2}(x_{\mathsf{G}}[]^n)) \\ & (y_{B_3} x_{B_3})\ (y_{B_3}(x_{\mathsf{G}}[]^n))\ (y_{B_4} x_{B_4})\ (y_{B_4}(x_{\mathsf{G}}[]^n)) \\ & (y_{\mathsf{G}} x_{\mathsf{G}})\ (y_{\mathsf{G}}(\Lambda^n.\lambda z_1 \ldots \lambda z_{n+1}.x_o))\ M_{B_1}\ M_{B_2}\ M_{B_3}\ M_{B_4}\ M_{E'} \end{array} \tag{4.1}$$

where x_o is as in Γ_o; $M_{B_1}, \ldots, M_{B_4}$ are encodings of the expressions $B_1, \ldots, B_4$; and $M_{E'}$ is an encoding of the set E' of equations. In case $E = \emptyset$, $M_E = x_o$.

Note that

$$\begin{array}{ll} \mathrm{FV}(M_E) \subseteq & \{x_B, y_B \mid B \text{ is subexpression of some first-order expression in } E\} \cup \\ & \{x_{\mathsf{H}}, y_{\mathsf{H}} \mid \mathsf{H} \text{ is a second-order variable in } E\} \cup \mathrm{Dom}(\Gamma_o). \end{array}$$

Let us discuss a few issues concerning the construction of the term M_E. First note that $\Gamma_o(y_o) = o^{15} \to o$ and that all the 15 arguments from $(y_{B_1} x_{B_1})$ through $M_{E'}$ must have the same type o. Now, the variables y_D for $D \in \{B_1, B_2, B_3, B_4, \mathsf{G}\}$ are used to enforce that the left hand sides of the equations are equal to the right hand ones. The type of the variable x_{G} is supposed to be the result of the solution on the variable G. The monadic restriction allows us to simulate the work of variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ as well as the constants $C_1, \ldots, C_n$ and $C'_1, \ldots, C'_n$ by means of the type hole application $x_{\mathsf{G}}[]^n$. We exploit the knowledge of the shape of B_2, B_4 that we have from (3.1) on page 107 to enforce that the constants $C_1, \ldots, C_n$ and $C'_1, \ldots, C'_n$ are used in the quantifier eliminations when y_D is applied to $x_{\mathsf{G}}[]^n$ (where $D = B_1, B_2, B_3$ or B_4). The term to which y_{G} is applied enforces that the type of x_{G} has exactly n universal quantifiers in its head and $(n+1)$

arrows inside. Note also that the term M'_E includes a similar encoding for the variable F. That encoding requires the use of $o^{21} \to o$ as we have 6 equations with F^4.

For a second-order substitution S we define $\Gamma_{o,S}$ as the smallest with regard to $\subseteq$ context such that $\Gamma_o \subseteq \Gamma_{o,S}$, $\Gamma_{o,S}(x_B) = S(B)$, and $\Gamma_{o,S}(y_B) = S(B) \to o$ for each subexpression B of a first-order expression in E, and $\Gamma_{o,S}(x_{\mathsf{G}}) = \forall X_1, \ldots, X_n.S(\mathsf{G})X_1 \cdots X_n$ and $\Gamma_{o,S}(y_{\mathsf{G}}) = (\forall X_1, \ldots, X_n.S(\mathsf{G})X_1 \cdots X_n) \to o$ for each second-order variable G occurring in E. Observe that it is possible to assign levels to type variables in $\Gamma_{o,S}$ so that the context as well as o are in level $k+1$ for each $k \geq 0$. We further work with such a version of the context.

Proposition 4.7. *Let E be the equations in the flat form above and S a substitution such that each type variable has level k. (1) If the substitution S solves E then there is a context $\Gamma \supseteq \Gamma_{o,S}$ such that $\Gamma \vdash M_E : o$ is derivable in level $k+1$. (2) If there is a context $\Gamma \supseteq \Gamma_{o,S}$ such that $\Gamma \vdash M_E : o$ is derivable in level $k+1$, then there is a substitution $S^\sharp$ such that $S^\sharp$ solves E and that $S^\sharp$ differs from S only on variables to which the second-order variables are applied.*

Proof. We assume here the notation as in Definition 4.6. The proof is by induction on the size of E. In case $E = \emptyset$ the claim holds trivially.

(1) Suppose that E is solvable under S, where each expression is constructed from type variables in level k. We show now that $\Gamma_{o,S} \vdash M_E : o$ is derivable in level $k+1$. First, observe that $\Gamma_{o,S}$ and o are in level $k+1$. Lemma 4.3 gives that M_{B_1}, M_{B_2}, M_{B_3}, and M_{B_4} are typable to o under appropriate contexts and as the contexts coincide with $\Gamma_{o,S}$ on the variables which occur free in the terms we may assume that the terms are typable to o under $\Gamma_{o,S}$. Similar reasoning starting with the use of Lemma 4.5 (in case E' is first-order) or the induction hypothesis (in case E' is second-order) gives that $\Gamma_{o,S} \vdash M_{E'} : o$ is derivable. Now, the derivability of $\Gamma_{o,S} \vdash y_D x_D : o$ for $D \in \{B_1, B_2, B_3, B_4, \mathsf{G}\}$ follows immediately from the definition of $\Gamma_{o,S}$. As the lemmas state, the derivations can be done in level $k+1$.

Now, for the proof of derivability of the terms in the form of $y_D(x_{\mathsf{G}}[]^n)$ for $D \in \{B_1, B_2, B_3, B_4\}$, we use in the type instantiations in places marked by [] that are indicated by appropriate equations, i.e. $S(X_1), \ldots, S(X_n)$; $C_1, \ldots, C_n$; $S(Y_1), \ldots, S(Y_n)$; $C'_1, \ldots, C'_n$, respectively. Then the derivability in level $k+1$ follows from the definition of $\Gamma_{o,S}$ and the fact that S unifies the equations displayed in Definition 4.6.

For the proof of derivability of $\Gamma_{o,S} \vdash y_{\mathsf{G}}(\Lambda^n.\lambda z_1 \ldots \lambda z_{n+1}.x_o) : o$, observe that $S(\mathsf{G}) = \lambda x_1 \ldots \lambda x_n.U_1 \to \cdots \to U_{n+1} \to B$ for some $U_1, \ldots, U_{n+1}$ and B. Otherwise it would be impossible to unify both the equation $\mathsf{G}X_1 \cdots X_n \doteq B_1$ and $\mathsf{G}C_1 \cdots C_n \doteq B_2$ by the constant restriction. Moreover, $B = o$ as the number of arguments is n and both B_1 and B_2 end with o. Therefore, we can assign the types $U_1, \ldots, U_{n+1}$ to $z_1, \ldots, z_{n+1}$ in $\lambda z_1 \ldots \lambda z_{n+1}.x_o$. Now, the derivability in level $k+1$ follows from the definition of $\Gamma_{o,S}$ and from the fact that all the steps above are done with help of types from $\Gamma_{o,S}$, the derivation is in level $k+1$.

Since all arguments of y_o are typable to o in $\Gamma_{o,S}$ and $y_o : o^{15} \to o$ is in Γ_o. We obtain a derivation for $\Gamma_{o,S} \vdash M_E : o$ in level $k+1$.

(2) Suppose, now, that $\Gamma \vdash M_E : o$ is derivable (in level $k+1$) for some $\Gamma \supseteq \Gamma_{o,S}$. As $\mathrm{FV}(M_E) = \mathrm{Dom}(\Gamma_{o,S})$, we can assume that $\Gamma_{o,S} \vdash M_E : o$. Now, we can prove that S is indeed a solution of E. Note first that either by induction hypothesis or by Lemma 4.5, S solves E'. As y_{G} is applied to both x_{G} and $\Lambda^n.\lambda z_1 \ldots \lambda z_{n+1}.x_o$, they have the same types. The type of $\Lambda^n.\lambda z_1 \ldots \lambda z_{n+1}.x_o$ (and at the same time the type of x_{G}) must be of

[4] $21 = 12 + 2 + 6 + 1$

the form $\forall X_1 \ldots \forall X_n.(U_1 \to \cdots \to U_{n+1} \to o)$ (*) (as the type system is syntax directed). From the derivability we obtain that $S(B_1) = S(\mathsf{G})D_1 \cdots D_n$ and $S(B_2) = S(\mathsf{G})D'_1 \cdots D'_n$ for some $D_1, \ldots, D_n, D'_1, \ldots, D'_n$, because of the way how y_{B_1}, y_{B_2} are used. Note that B_2 contains n positions (the positions of $C_1, \ldots, C_n$) at which terms differ from the terms in the corresponding positions in B_1. Moreover, all these positions occur in the type marked as (*). Therefore, the only possible way to ensure that the equalities above hold is when variables occupy the positions in the type of x_{G}. Then it follows that the sequence $D'_1, \ldots, D'_n$ is a permutation of $C_1, \ldots, C_n$. We can now permute the application of the type instantiations so that the constants are applied in the order $C_1, \ldots, C_n$. Then, the arguments $D_1, \ldots, D_n$ must be permuted in the same way. Then, however, A_i must occur in $S(B_1) = S(\mathsf{G})D_{i_1} \cdots D_{i_n}$ where C_i is used in $S(B_2) = S(\mathsf{G})C_1 \cdots C_n$ for each fixed $i = 1, \ldots, n$. This is because A_i occurs in B_1 in position where C_i occurs in B_2. In the end, we obtain that $S(B_1) = S(\mathsf{G})S(A_1) \cdots S(A_n)$ and $S(B_2) = S(\mathsf{G})C_1 \cdots C_n$. As variables $X_1, \ldots, X_n$ occur only in $\mathsf{G}X_1 \cdots X_n$, we may change $S(X_l)$ to D_{i_l} and in this way obtain $S^{\sharp,1}$. A similar reasoning may be used to infer the equalities $S(B_3) = S(\mathsf{G})Y_1 \cdots Y_n$ and $S(B_4) = S(\mathsf{G})C'_1 \cdots C'_n$. In this way, we obtain the required solution $S^{\sharp}$ to the whole E. ∎

This ends the technical lemmas which are enough to prove undecidability of TCP and TIP.

4.3. Enforcing the Content of Contexts for TP

When TCP and TIP are considered, a context is a part of the initial data so we can provide one which is useful for the purpose of undecidability proof. This is no longer the case when the goal is the undecidability of TP. In particular, we have to make sure that the types of different constants used in unification expressions are indeed different. Moreover, the expressions from the substitution must not occur neither in the context nor in the resulting type. This can be obtained with the help of the following construction.

The first step is to turn the type variables which correspond to constants in the unification expressions into quantified variables. Let $\vec{x_C} = \{x_{C_1}, \ldots, x_{C_m}\}$, where all the constants in E subject to the constant restriction are collected. Let $\vec{y} = (\mathrm{FV}(M_E) \cup \mathrm{FV}(M_o)) \backslash (\{x_o\} \cup \vec{x_C})$ and $\hat{M}_E$ be the following term:

$$\hat{M}_E \equiv f(\Lambda^{m+1}.\lambda x_{C_1}.\cdots \lambda x_{C_m}.\lambda x_o.\hat{K}(\lambda \vec{y}.y_o M_E M_o))$$

where M_o is as in Proposition 4.1 and $\hat{K} = \lambda x.g$. Note that $\mathrm{FV}(\hat{M}_E) = \{f, g\}$. Let

$$\Gamma_E = \{f : (\forall C_1^{(k)}.\cdots \forall C_m^{(k)}.\forall C_{m+1}^{(k)}.(C_{i_1} \to \cdots \to C_{i_{m+1}} \to W_1)) \to W_2,\ g : W_1\}$$

where W_1 and W_2 are any types in level $k+1$ that do not use $C_1, \ldots, C_{m+1}$, $\Gamma_E(f)$ is in level $k+1$ and $i_1, \ldots, i_{m+1}$ is a permutation of $1, \ldots, m+1$.

With help of the term $\hat{M}_E$ we reduce the problem of making $C_1, \ldots, C_m$ different to the problem to enforce the particular shape of a type for f. This is guaranteed by the following lemma.

Proposition 4.8. *Let E be the equations in the flat form above and $k \geq 0$. The problem E is solvable if and only if $\Gamma_E \vdash \hat{M}_E : W_2$ is derivable.*

Moreover, if E is solvable the derivation of $\Gamma_E \vdash \hat{M}_E : W_2$ can be done in level $k+1$.

Proof. ($\Rightarrow$) If E is solvable by S then we can use the substitution as in Proposition 4.7 to derive $\Gamma_{o,S} \vdash M_E : o$ in level $k+1$. As $\Gamma_o \subseteq \Gamma_{o,S}$, we can also derive $\Gamma_{o,S} \vdash M_o : o$ by Proposition 4.1. Since M_E, M_o are typable to o and $\Gamma_{o,S}$ contains types of level $k+1$ we can derive a type B for $M_* \equiv \lambda\vec{y}.y_o M_E M_o$ which is in level $k+1$. Let us define $\Gamma_C = \{x_C : C \mid x_C \in \vec{x_C}\} \cup \{x_o : o\}$. We can directly check that M_* is typable in $\Gamma_E \cup \Gamma_C$ and all of the derivation of $\Gamma_E \cup \Gamma_C \vdash M_* : B$ can be done in level $k+1$. Now, we can do the type inference for $\hat{K} = \lambda x.g : B \to W_1$ which allows us to derive $\Gamma_E \cup \Gamma_C \vdash \hat{K} M_* : W_1$. Then we can abstract all the variables from Γ_C and then do the type abstraction for the type variables that serve as constants i.e. $C_1, \ldots, C_m, o$. Observe, that the way we use Proposition 4.7 means that $C_1, \ldots, C_m, o$ are in level k. With this in mind we can see that it is possible to derive in level $k+1$ the type W_2 for $f(\Lambda^{m+1}.\lambda x_{C_1}.\cdots\lambda x_{C_m}.\lambda x_o.\hat{K}M_*)$, which is the required result.

($\Leftarrow$) Suppose that $\Gamma_E \vdash \hat{M}_E : W_2$ (in level $k+1$). From $\Gamma_E(f)$, we have

$$\Gamma_E \vdash \Lambda^{m+1}.\lambda x_{C_1}.\cdots\lambda x_{C_m}.\lambda x_o.\hat{K}(\lambda\vec{y}.y_o M_E M_o) \\ : \forall C_1^{(k)}.\cdots\forall C_m^{(k)}.\forall C_{m+1}^{(k)}.(C_{i_1} \to \cdots \to C_{i_m} \to C_{i_{m+1}} \to W_1).$$

Then we have $\Gamma_E \vdash \lambda x_{C_1}.\cdots\lambda x_{C_m}.\lambda x_o.\hat{K}(\lambda\vec{y}.y_o M_E M_o) : C_{i_1} \to \cdots \to C_{i_m} \to C_{i_{m+1}} \to W_1$, where $C_1, \ldots, C_{m+1}$ are fresh and pairwise distinct type variables from the variable condition of $(\forall I)$. Further, we obtain

$$\Gamma \equiv \Gamma_E \cup \{x_{C_1} : C_{i_1}, \ldots, x_{C_m} : C_{i_m}, x_o : o\} \vdash \hat{K}(\lambda\vec{y}.y_o M_E M_o) : W_1,$$

and then we have $\Gamma \vdash \lambda\vec{y}.y_o M_E M_o : B'$ for some type B'. Hence, we can derive $\Gamma \vdash M_o : o$, which implies that $\Gamma_o \subseteq \Gamma$ by Proposition 4.1. We can now define a substitution S as $S(X) = B$ for all $x_X : B \in \Gamma$ and $S(\mathsf{H}) = \lambda x_1 \ldots \lambda x_l.B[X_1 := x_1, \ldots, X_l := x_l]$ for each second-order variable H in E such that $x_{\mathsf{H}} : \forall X_1 \ldots \forall X_l.B \in \Gamma$. Observe now, that $\Gamma \supseteq \Gamma_{o,S}$ in the notation of Proposition 4.7 (up to inessential renaming of the constants). Then the case (2) of the proposition gives a slightly modified substitution $S^\sharp$ that solves E. ∎

Now, we must enforce a particular form of type for the variable f. We define, therefore, a term N_f, which forces the required type. For this, let $\mathrm{id} = \lambda x.x$ and $\mathrm{ID} = \Lambda.\lambda x.x$. We write $\lambda^{n+1}x.M$ for $\lambda x.(\lambda^n x.M)$ where $\lambda^0 x.M \equiv M$. Let us define:

$$N_f \equiv z\,(z_1(fa))\,(z_2(a[]^{m+1}\,\mathrm{id}^{m+1})) \\ (z_2(a[]^{m+1}\,\mathrm{ID}\,(\Lambda.\mathrm{ID})\ldots(\Lambda^m.\mathrm{ID})))\,(z_3 a)\,(z_3(\Lambda^{m+1}.\lambda^{m+1}v.z_4))$$

where $z, z_1, a, z_2, z_3, v, z_4$ are fresh term variables.

Lemma 4.9. (a) *If N_f is typable where the derivation contains types only in level $(k+2)$ with $k \geq 0$, then for all types $D_1, \ldots, D_6$ in level $(k+2)$ there is a context Γ_f such that*

$$\begin{array}{l} \Gamma_f(a) = \forall X_1^{(l_1)}.\cdots\forall X_{m+1}^{(l_{m+1})}.(X_1 \to \cdots \to X_m \to D_1) \text{ with} \\ \qquad\qquad X_1, \ldots, X_{m+1} \notin \mathrm{FV}(D_1), \quad l_1, \ldots, l_{m+1} \leq k+1, \\ \Gamma_f(f) = \Gamma_f(a) \to D_2, \qquad \Gamma_f(z_1) = D_2 \to D_3, \qquad \Gamma_f(z_2) = D_1 \to D_4, \\ \Gamma_f(z_3) = \Gamma_f(a) \to D_5, \Gamma_f(z_4) = D_1, \text{ and} \\ \Gamma_f(z) = D_3 \to D_4 \to D_4 \to D_5 \to D_5 \to D_6. \end{array} \tag{4.2}$$

(b) *Suppose that $D_1, \ldots, D_6$ are types in level $k+2$. A judgement $\Gamma_f \vdash N_f : D_6$ is derivable in level $k+2$, where Γ_f is defined as in (4.2).*

Proof. (a) If N_f is typable in level $(k+2)$ with $k \geq 0$, then the subterm $(z_1(fa))$ enforces an arrow type on type of f. Further the pair of subterms $(z_3 a)$ and $(z_3(\Lambda^{m+1}.\lambda^{m+1}v.z_4))$ enforce the shape $\forall X_1^{(l_1)}.\cdots\forall X_{m+1}^{(l_{m+1})}.(A_1 \to \cdots \to A_m \to D_1)$, with $l_1, \ldots, l_{m+1} \leq k+1$ and some types A_i, D_1, on the type of a (being the argument type of f). Moreover, the subterms $(a[]^{m+1}\,\mathrm{id}^{m+1})$ and $(a[]^{m+1}\,\mathrm{ID}\,(\Lambda.\mathrm{ID})\ldots(\Lambda^m.\mathrm{ID}))$ enforce the shape

$$\forall X_1^{(l_1)}.\cdots\forall X_{m+1}^{(l_{m+1})}.(X_{i_1} \to \cdots \to X_{i_{m+1}} \to D_1)$$

on type of a for some permutation $i_1, \ldots, i_{m+1}$ of $1, \ldots, m+1$. In addition, the typability of $(\Lambda^{m+1}.\lambda^{m+1}v.z_4)$ gives $X_1, \ldots, X_{m+1} \notin \mathrm{FV}(D_1)$ from the variable condition on $(\forall I)$, so that we have

$$\Gamma_f(f) = (\forall X_1^{(l_1)}.\cdots\forall X_{m+1}^{(l_{m+1})}.(X_{i_1} \to \cdots \to X_{i_{m+1}} \to D_1)) \to D_2.$$

For types of other free variables, the analysis is straightforward.

(b) A direct check gives us the case of the lemma. ∎

4.4. The Main Theorem

We can now prove the main result of the paper.

Theorem 4.10 (TP, TIP, TCP)**.** TP^2, TIP^1, *and* TCP^1 *together with* TP, TIP, *and* TCP *are undecidable for the type-free system of* $\lambda 2$.

Proof. Consider first TIP^1. We can take as an instance of the problem $M \equiv \lambda\vec{y}.y_o M_E M_o$ and $\Gamma \equiv \Gamma_o \cup \{x_C : C \mid C \text{ is a constant}\}$ where all the type variables in Γ are in level 1 and $\vec{y}$ are all free term variables in M_E except those in Γ. If $\Gamma \vdash M : A$ is derivable in level 1 then there is Γ' such that $\Gamma' \vdash M_E : A'$ is derivable in level 1. By Proposition 4.1 applied to M_o we obtain $A' = o$. Now, we can define a substitution S such that $S(X) = B$ for all $x_X : B \in \Gamma'$ and $S(\mathsf{H}) = \lambda x_1 \ldots x_l.B[X_1 := x_1, \ldots, X_l := x_l]$ for each second-order variable H in E such that $x_{\mathsf{H}} : \forall X_1 \ldots \forall X_l.B \in \Gamma'$. Immediate check verifies that, $\Gamma' \supseteq \Gamma_{o,S}$ (in notation of Proposition 4.7). Now, Proposition 4.7(2) gives that E is solvable.

If E is solved by S then we can impose that nullary symbols are in level 0. Then Proposition 4.7(1) gives derivation of $\Gamma_{o,S} \vdash M_E : o$ in level 1 and Proposition 4.1 gives $\Gamma_{o,S} \vdash M_o : o$. These two can be immediately combined into a derivation of $\Gamma \vdash M : A$ for some A. In this way we reduced the solvability of equations in flat form to TIP^1. The same proof works for the unbounded version of TIP and for the impredicative system.

The reasoning for TCP^1 follows the same lines, but we take as an instance the judgement $\Gamma \vdash (\lambda v.x_o)(\lambda\vec{y}.y_o M_E M_o) : o$, where v is fresh and Γ is the context given for the TIP^1 above.

As for the proof that $\mathrm{TP}^{(2)}$ is undecidable, we can assume w.l.o.g that $\mathrm{FV}(N_f) \cap \mathrm{FV}(\hat{M}_E) = \{f\}$. We show now that the typability of $zN_f\hat{M}_E$ in level 2 is equivalent to solvability of E. Indeed, suppose that the judgement $\Gamma \vdash zN_f\hat{M}_E : A$ is derivable in level 2 for some Γ, A. From Lemma 4.9(a),

$$\Gamma(f) = (\forall X_1^{(1)}.\cdots\forall X_{m+1}^{(1)}.(X_1 \to \cdots \to X_m \to D_1)) \to D_2$$

with $X_1, \ldots, X_{m+1} \notin \mathrm{FV}(D_1)$. Hence, the shape of $\hat{M}_E$ implies that $\Gamma \vdash \hat{M}_E : D_2$ is derivable. By Proposition 4.8, we obtain the solvability of E.

In case E is solvable, we combine Lemma 4.9(b) with Proposition 4.8 in a straightforward way. Notice that $\hat{M}_E$ is typed here in level 2.

We can now conclude that TP is undecidable in level 2. The same proof works for the unbounded version of TP and for the impredicative system. ∎

A careful reader might spot that the solution S obtained from a derivation can contain occurrences of $\forall$ which is not used in standard second-order unification expressions. This can, however, be mitigated by the following proposition.

Proposition 4.11. *If a set E of equations does not contain occurrence of the symbol $\forall$ then for each solution S of E there is a solution which does not use $\forall$.*

Proof. In order to prove the proposition, you should simply fix a constant (or a variable) C and each time $S(X)$ or $S(\mathsf{H})$ contains an occurrence of $\forall$ replace the subterm at that occurrence with C. In this way we obtain a substitution S'. For each equation $A \doteq B$ we obtain that $S'(A) = S'(B)$ as each time we have something different in $S'(A)$ than in $S(A)$ this must be C. This means that $S(B)$ at the same position has $\forall$. Since $\forall$ does not occur in B it must occur in a term which comes from S. Then this occurrence of $\forall$ is replaced with C in S'. ∎

5. Concluding Remarks

The current paper shows new paths of investigation on the type-free systems, interesting type systems for functional programming with moderate type annotation and the relation to the second-order unification. We have proved the undecidability of the type-checking, type inference and typability problems for the predicative version of the System F in the type-free style. The proof method even works for the impredicative version by flatting stratified types. Namely, TCP, TIP, and TP are all undecidable for the System F in the type-free style. Then, as in [NTKN08], the technique of CPS-translation can be applied to show that TCP, TIP, and TP are all undecidable for the existential system $\lambda^{\exists}$ [FS09] in the type-free style, consisting of $(\neg, \wedge, \exists)$. In [FS09], we proved that all of the type related problems are in general undecidable for the type-free system of $\lambda^{\exists}$ consisting of $(\rightarrow, \exists)$. We remark that a detailed analysis on the proof method in [FS09] reveals that TCP and TIP are still undecidable for the finitely stratified $\lambda^{\exists}$ of $(\rightarrow, \exists)$ in level 2 and that TP is undecidable for the system in level 3 as well. Moreover, the extended version [FS] leads to stricter borders, such that TCP and TIP are undecidable in level 1 and TP is undecidable in level 2 for the predicative system $\lambda^{\exists}$ of $(\rightarrow, \exists)$.

For the predicative version of the System F in the type-free style, our results provide a strict undecidability border for TCP and TIP problems as they are undecidable for level 1 types (level 0 types have no quantifiers so they are equivalent to the simply typed lambda calculus). We also prove undecidability for TP in level 2. We believe that the current construction can be adapted to prove undecidability of TP in level 1. In that case, however, the constants obtained in the proof of Proposition 4.8 must be simulated by more complicated (arrow) types which makes the whole construction much more involved.

References

[Ami90] Gilles Amiot. The undecidability of the second order predicate unification problem. *Archive for Mathematical Logic*, 30(3):193–199, May 1990.

[Boe85] H.-J. Boehm. Partial polymorphic type inference is undecidable. In *26th Annual Symposium on Foundations of Computer Science*, pages 339–345. IEEE, October 1985.

[FS] K. Fujita and A. Schubert. Existential type systems between Church and Curry style. Submitted. Available from `http://www.mimuw.edu.pl/~alx/papers/existential-chcu.pdf`.

[FS00] K. Fujita and A. Schubert. Partially typed terms between Church-style and Curry-style. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, number 1872 in LNCS, pages 505–520, 2000.

[FS09] K. Fujita and A. Schubert. Existential type systems with no types in terms. In P.-L. Curien, editor, *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009, Proceedings*, number 5608 in LNCS, pages 112–126, 2009.

[GR94] Paola Giannini and Simona Ronchi Della Rocca. A type inference algorithm for a stratified polymorphic type discipline. *Information Computation*, 109(1–2):115–173, 1994.

[HS99] John Hatcliff and Morten Heine B. Srensen. CPS translations and applications: The cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, September 1999.

[KTU90a] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 468–476, New York, NY, USA, 1990. ACM.

[KTU90b] Assaf J. Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. ML typability is DEXTIME-complete. In *CAAP '90: Proceedings of the 15th Colloquium on Trees in Algebra and Programming*, number 431 in LNCS, pages 206–220, London, UK, 1990. Springer-Verlag.

[KW94] Assaf J. Kfoury and Joeseph B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *LFP '94: Proceedings of the 1994 ACM conference on LISP and functional programming*, pages 196–207, New York, NY, USA, 1994. ACM.

[Lei83] Daniel Leivant. Polymorphic type inference. In *POPL '83: Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 88–98, New York, NY, USA, 1983. ACM.

[Lei91] D. Leivant. Finitely stratified polymorphism. *Information and Computation*, 93(1):93–113, 1991.

[LV00] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159(1–2):125–150, 2000.

[Mai90] Harry G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 382–401, New York, NY, USA, 1990. ACM.

[NTKN08] K. Nakazawa, M. Tatsuta, Y. Kameyama, and H. Nakano. Undecidability of type-checking in domain-free typed lambda-calculi with existence. In *CSL '08: Proceedings of the 22nd International Workshop on Computer Science Logic*, number 5213 in LNCS, pages 478–492, Berlin, Heidelberg, 2008. Springer-Verlag.

[Pfe93] F. Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundamenta Informaticae*, 19(1,2):185–199, September-October 1993.

[Sch98] Aleksy Schubert. Second-order unification and type inference for Church-style polymorphism. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 279–288, New York, NY, USA, 1998. ACM.

[Sch01] Aleksy Schubert. *Zastosowanie unifikacji do problemw wyprowadzania typw*. PhD thesis, The University of Warsaw, 2001. English title: Application of the unification to type inference problems, Polish text available from `http://www.mimuw.edu.pl/~alx/ftp-public/doktorat.pdf`.

[Wel99] J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1–3):111–156, 1999.