Proof search when equality is a logical connective: Extended Abstract

Alexandre Viel and Dale Miller

INRIA-Saclay & LIX/Ecole Polytechnique
Palaiseau, France

Abstract. We explore how one might do proof search in a simple, first-order logic where equality is treated as a logical connective. That is, equality over first-order terms has a left and right introduction rule and the formulation of the left-introduction rule involves unification of eigenvariables. The usual strategy for implementing proof search also involves unification, but this time for "existential variables" (not eigenvariables). In this paper, we show such unification over these two species of variables can at times be solved by a reduction to a particular subset of second-order unification problems that we call the *argument-constrained* subset. We then show that solving such constrained unification is undecidable. Finally, we show that solvability of a given second-order unification problem can be related to solvability of an argument-constrained version of that unification problem.

1 Introduction

We consider a first-order logic in which equality on first-order terms is interpreted as the equality on trees. Sequents are written as the triple $\Sigma; \Gamma \longrightarrow \Delta$ where Σ is a set of eigenvariables and Γ and Δ are multisets of formulas all of which can contain free variable only if they appear in Σ.

The rules for first-order quantification are

$$\frac{x, \Sigma; \Gamma \longrightarrow B(x), \Delta}{\Sigma; \Gamma \longrightarrow \forall x.B(x), \Delta}\ \forall_R \qquad \frac{\Sigma; \Gamma \longrightarrow B(t), \Delta}{\Sigma; \Gamma \longrightarrow \exists x.B(x), \Delta}\ \exists_R$$

where t is a term built on the signature Σ. In the $\forall_R$ rule, the eigenvariable x is assumed to not be in Σ and, as a consequence, it is not free in the concluding sequent. The rules for first-order equality are

$$\frac{}{\Sigma; \Gamma \longrightarrow t = t, \Delta}\ eq_R \qquad \frac{}{\Sigma; \Gamma, t = t' \longrightarrow \Delta}\ eq_L\dagger \qquad \frac{\theta\Sigma; \theta\Gamma \longrightarrow \theta\Delta}{\Sigma; \Gamma, t = t' \longrightarrow \Delta}\ eq_L\ddagger$$

The eq_L rule is rewritten as two rules: the proviso † requires that t and t' are not unifiable and the proviso ‡ requires that t and t' are unifiable with most general unifier θ. The signature denoted by $\theta\Sigma$ is the one that results from removing

from Σ all those variables that are in the domain of θ and add all those variables that are free in some term in the range of θ.

This treatment of equality was proposed by Schroeder-Heister [9] and by Girard [3]. In this treatment, it is a theorem that equality is, for example, an equivalence relation. This approach stands in contrast to other approaches where equality is a non-logical predicate that is axiomatized (see, for example, Gallier's textbook [2]). In such a setting, equivalence is proposed as an axiom: additionally, equality could be extended to allow for much richer identification of terms, such as that which can be found in, say, reasoning about strings or within various algebras.

2 Representing unification as a formula

We want to study the first-order mechanics independently from the rest of the logic: for example, the logic might be linear or classical or intuitionistic. We shall assume that propositional logical connective (*e.g.*, conjunction, implication, disjunction) do not interact with the first-order structures. As a consequence, we shall concentrate our efforts on those connectives that deal directly with the first-order term structure: in particular, the quantifiers and equality. For any given proof tree, we can extract a sub-tree (a skeleton) containing all the information relevant to the first-order logic. Then we can translate that sub-tree into a formula containing exactly the first-order information that the proof contains.

The subtree is obtained by erasing all the rules but the ones above, dealing with the signature and equality of terms. Then, the leaves of the subtree are the eq_R rules, and the void $eq_L\dagger$ rules. The $eq_L\ddagger$ rules are unary nodes in the tree. The $\forall$ and $\exists$ rules are kept as unary nodes that may add to the signature. The other branching will be translated with a conjunction $\wedge$. If a proof subtree has no use of any first-order rule, (and in particular does not have any $eq_L\dagger$ or eq_R leaf),then it uses no first-order information and can be completely omitted in the extraction.

Thus the first-order information of the proof can be described by a formula Φ of the following grammar.

$$\Phi ::= \quad t = t' \mid (t = t') \supset \Phi \mid \Phi \wedge \Phi \mid \exists x.\Phi \mid \forall x.\Phi$$

We use the operator $\wedge$ to represent the branching of the proof tree, it is not related to any connector of the original logic.

Another way to think of this extraction process is to consider the idealized interpreter for L_λ given in [6]. That interpreter's state was encoded using *state formulas* that are composed of the logical connectives $\forall$, $\exists$, and $\wedge$ and equality judgments and sequent judgment. The order in which one schedules solving equality and sequent judgments is flexible, one could, in fact, attempt to solve all sequent judgments first. If they are all removed, what is left is a formula involving $\forall$, $\exists$, $\wedge$, and equality judgments. The main difference between [6] and

this paper is that inequalities and hypothetical equalities appear (equality was not treated as a logical connective in [6]).

Example. Consider the formula

$$\forall a \forall b \exists X.\ ((a = b \supset X = b) \wedge X = a).$$

A naive approach to solving this problem could select the first conjunction $(a = b \supset X = b)$ and then substitute b for a, leaving us to prove the (unchanged) formula $X = b$. Thus, we could, in essence, be trying to prove $X = b \wedge X = a$, which fails. This approach failed because in $(a = b \supset X = b)$ there are two solutions for X: either $X = a$ or $X = b$, and we were unlucky in our choice of the substitution to apply. Instead, in order to simplify $(a = b \supset X = b)$ while keeping the two solutions, we have to make explicit that a and b can occur in X. If we replace X with $(X'\ a\ b)$ (where X' is a second-order variable) then it is correct that $(a = b \supset (X'\ a\ b) = b)$ is equivalent to $X\ b\ b = b$. Thus, we are trying to prove $X\ b\ b = b \wedge X\ a\ b = a$, and we find the solution $X = \lambda a \lambda b.\ a$. This example illustrates the usefulness of using second-order variables to study this class of formulas. **End of example.**

3 Translation into second-order unification

The proof of such an extracted formula Φ contains all the data needed to solve the unification problems implied by an attempt to build a proof. To make the unification problem embedded in such an extracted formula more apparent, we can transform it directly into a single, second-order unification problem.

First, we transform the formula into a normal form of the shape

$$\exists Y_1 \ldots \exists Y_m.[\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n], \qquad (n, m \geq 0)$$

where the formulas $\mathcal{C}_i$ are *conditional equations* (also called *clauses*): that is, they are formulas of the form

$$\forall x_1 \cdots \forall x_p[s_1 = t_1 \supset \ldots \supset s_q = t_q \supset s_0 = t_0]. \qquad (p, q \geq 0)$$

The conjunction $s_1 = t_1 \wedge \ldots \wedge s_q = t_q$ will be called the *condition* and $s_0 = t_0$ the *target* of this clause. In order to move existential quantifiers to the outermost location, the type of the existential variables are raised [7]: in particular, in the original formula, the existentials are quantified over the type of terms ι. In the resulting formula, their raised type will be $\iota^n \to \iota$ where n is the number of universal quantifiers preceding their introduction.

Within Φ formulas, we note the following restriction on variable occurrences: if an existential variable, say, $Y_i : \iota^q \to \iota$ (for some $q \geq 0$) appears in the scope of the quantifiers $\forall x_1 \cdots \forall x_p$ then Y_i is applied to *exactly* the first q variables in the that list: that is, Y_i occurs as $(Y_i\ x_1\ \ldots\ x_q)$.

The following equivalences are sufficient to rewrite Φ formulas into those involving only clauses.

$$\begin{aligned}
\psi \supset (\phi_1 \wedge \phi_2) &\equiv (\psi \supset \phi_1) \wedge (\psi \supset \phi_2) \\
\psi \supset (\forall x\ \phi) &\equiv \forall x\ (\psi \supset \phi) \\
\psi \supset (\exists_\tau x\ \phi) &\equiv \exists_\tau x\ (\psi \supset \phi) \\
(\exists_\tau x\ \phi_1) \wedge \phi_2 &\equiv \exists_\tau x\ (\phi_1 x \wedge \phi_2) \\
\forall x\ (\phi_1 \wedge \phi_2) &\equiv (\forall x\ \phi_1) \wedge (\forall x\ \phi_2) \\
\forall x\ \exists_\tau y\ \phi &\equiv \exists_{\iota \to \tau} h\ \forall x\ \phi[(hx)/y]
\end{aligned}$$

Before we proceed, we shall assume that the underlying signature of constants contains at least two different constructors. Notice that this inserts into our logic a false formula: that is, if c (of arity, say, 1) and d (of arity, say, 2) are two different constructors then false is provably equivalent to the formula $\exists x \exists y.(c\ x) = (d\ x\ y)$.

The problem of solving (proving) some of these clauses can be reduced to equivalent second-order unification problems. We shall focus on such a subset in the rest of this paper.

Many conditions can be reduced using provable equivalences. In particular, the equivalences

$$c(s_1, \ldots, s_p) = c(t_1, \ldots, t_p) \equiv [s_1 = t_1 \wedge \ldots \wedge s_p = t_p] \quad \text{and} \quad c(\bar{s}) = d(\bar{t}) \equiv \bot$$

(where c and d are different constructors) can be used to remove all conditional equalities in a clause involving two terms with constructors at their heads. Additionally, let x be either an existentially or universally quantified variable and let t be a term containing x strictly: that is, there is an occurrence of x in t such that the path from the root of t to that occurrence of x contains one or more constructors and does not contain any existentially (raised) variable. Then the two equivalences $x = x \equiv \top$ and $x = t \equiv \bot$ can also be used to simplify clauses. Repeatedly using the above equivalences essentially attempts to perform the computation of first-order most general unifiers. Finally, one can use the (provable) equivalence $(\forall x.x = t \supset B(x) \equiv B(t)$, where x is not free in t, to replace some conditions within a clause with substitution instances. This additional rewriting phase is then used to apply the computed most general unifier.

Clauses that remain after this phase of simplification may still have non-trivial conditions. In particular, these conditions may be of the following form.

- $x = t$ where every occurrence of (the universally quantified variable) x in t is in the scope of an existential (raised) variable. In this case, it may or may not be the case that x occurs free in substitution instances of t.
- $(X\ \bar{s}) = (Y\ \bar{t})$ or $(X\ \bar{s}) = (c\ \bar{t})$ where X and Y are raised existential variables, c is a constructor, and $\bar{s}$ and $\bar{t}$ are lists of terms. There may or may not be instantiations of X and Y that make these equations true.

We shall treat any clause that remains with such conditions as *suspended* constrains on the search for solutions to Φ-formulas. That is, we shall not explicitly

attempt to find unifiers for them. Such suspended clauses may change status when substitutions are made for existential variables: thus, we can then chose to reconsider them in an attempt to further simplify them. Here, we only propose to search for unifiers for clauses without conditions.

Example. Consider the formula

$$\forall a\ \exists X\ \forall b\ \exists Y\ [(a = b \supset X = Y) \wedge (a = s(b) \supset X = s(Y))].$$

After transforming the formula, we get

$$\exists X : \iota \to \iota\ \exists Y : \iota \to \iota \to \iota \left[\begin{array}{l}(\forall a\ \forall b.\ a = b \quad \supset X\ a = Y\ a\ b \quad) \wedge \\ (\forall a\ \forall b.\ a = s(b) \supset X\ a = s(Y\ a\ b)\ )\end{array}\right]$$

(Notice that the type of X and Y have changed between these two formulas.) These conditional equations can be simplified further but using various equivalences of the form $(\forall x.x = t \supset B\ x) \equiv (B\ t)$ to get

$$\exists X : \iota \to \iota\ \exists Y : \iota \to \iota \to \iota[(\forall a.\ X\ a = Y\ a\ a) \wedge (\forall a.\ X\ (s\ a) = s(Y\ (s\ a)\ a))]$$

which is another way to write the the second-order unification problem (for free variables $X : \iota \to \iota$ and $Y : \iota \to \iota \to \iota$)

$$\begin{array}{ll}\lambda a.\ X\ a & = \lambda a.\ Y\ a\ a \\ \lambda a.\ X\ (s\ a) & = \lambda a.\ s\ (Y\ (s\ a)\ a)\end{array}$$

The solutions to this unification problem are the pairs $(X = \lambda a.s^n(a); Y = \lambda ab.s^n(b))$. We get a bijection between the solutions of this problem and natural numbers. **End of example.**

The class of second-order unification problems that arise by normalizing and simplifying our original Φ-formulas are of a special form. Notice that all arguments to (raised) existential variables in an equality are prefixes of a common list of arguments. This specific structure arises from the use of raising to flatten the first-order quantificational structure. Such regularity suggests a new notation: instead of writing raised variables applied to arguments, such as $(X\ t_1 \ldots t_n)$, we shall instead write simply $\widehat{X}$ along with an explicit context $[t_1, \ldots, t_n, \ldots]$. The type of X is needed to determine which prefix of this context is needed as its argument list. For example, the unification problem above will be written more simply as

$$\begin{array}{lll}\widehat{X} = & \widehat{Y} & [a \quad a] \\ \widehat{X} = & (s\ \widehat{Y}) & [(s\ a)\ a]\end{array}$$

Reconstructing the second-order unification problem from this notion is straightforward: simply replace the hatted variable, say $\widehat{X}$, with X applied to the prefix of the bracketed context following the equation of length determined from X's type. Thus, in the first equation, $\widehat{X}$ is replaced by $(X\ a)$ but in the second equation it is replaced by $(X\ (s\ a))$. Also, even though the names of the free variables in the context could be renamed to anything, we choose to keep them close to the name of the first-order variables they are replacing.

An *argument-constrained* (second-order) unification problem is a list of *constrained* equalities of the form $s = t\ [r_1 \dots r_n]$ such that s, t and $r_1, \dots, r_n$ are *first-order* terms with free hatted variables.

While trying to solve a unification problem, if we have not specified a variable name for the arguments of existential variables, we allow the use of "projections" to refer to them in the body of the existential variables: in particular, π_i can be used to denote the i^{th} argument.

4 Cyclic dependencies

Given a set of equations of the form above, we can graph the direct dependencies between existential variables. If we have an equation $X = t$ (or $t = X$) where Y occurs in t, we have an arrow from X to Y in the direct dependencies graph of the problem. The occasional existential variables appearing in contexts of equations, and so in the arguments of X form indirect dependencies, and will be ignored in this section.

Having cycles of dependencies can lead to complex behavior: in the previous example, we found the simplest cyclic dependencies between X and Y, which forced the solutions to the problem to have the structure of the natural numbers.

Here we study in more depth what can be done via those cycles. Let p be a generic binary constructor, and consider the following argument-constrained unification problem:

$$V_0, W : \iota^n \longrightarrow \iota \qquad Y, Z : \iota^{n+2} \longrightarrow \iota$$

$$\begin{array}{ll} \widehat{Y} = \widehat{Z} & [t_1 \dots t_n \star \quad \star] \\ \widehat{Y} = p(\widehat{V_0}, \widehat{Z}) & [t'_1 \dots t'_n \star\, p(\widehat{W}, \star)] \end{array}$$

Here, $\star$ is a variable name that does not appear in the terms t_i, t'_i. The equality of second-order terms after application of a context is still an equivalence relation on second-order terms. Let us denote the equality after applying the first context with $=_1$ and equality after applying the second context with $=_2$. In this problem, we have a cycle of dependencies

$$Z =_1 Y =_2 p(V, Z)$$

We can solve this starting in two ways.

– Y can start with the constructor p, and then so does Z: There are second-order variables W_1, V_1, Y', Z' of the same type as Y and Z such that

$$Y = p(W_1, Y') \qquad Z = p(V_1, Z')$$

The equations now become:

$$V_0 =_2 W_1 =_1 V_1 \quad \text{and} \quad Z' =_1 Y' =_2 p(V_1, Z)$$

We have again to solve the same problem, except V_0 has changed into a V_1.

– Y and Z can be one of their arguments. There is only one way to do it:

$$Y = \pi_{n+2} \qquad Z = \pi_{n+1}$$

The equations now simply become $V_0 =_2 W$

Thus the solutions are all of the following form:

$$Y = p(W_1, p(W_2, \dots p(W_m, \pi_{n+2}) \dots))$$

$$Z = p(V_1, p(V_2, \dots p(V_m, \pi_{n+1}) \dots))$$

$$\text{with} \qquad V_0 =_2 W_1 =_1 V_1 =_2 W_2 =_1 V_2 \dots W_m =_1 V_m =_2 W$$

In particular, the problem is solvable if and only if $V_0(=_2=_1)^* W$.

Note that even though $=_1$ and $=_2$ are equivalence relations by themselves, they may be different, so their composition may not be one, which is why $(=_2=_1)^*$ can become an interesting relation that can not be described by a single simple context.

From the fact that $=_1$ and $=_2$ are symmetric and reflexive relations, we can show that $(=_1=_2)^*$ is also symmetric reflexive, so is equivalent to $(=_2=_1)^*$. If we switch the t_i with the t'_i, the solvability of the problem does not change, only the solutions for Y and Z will be slightly different.

If we recall how the contexts are related to some normalized conditions γ_1 and γ_2, V_0 is related to W if and only if we can rewrite V_0 into W assuming those conditions, i.e. we have a rewriting proof of $(\gamma_1 \wedge \gamma_2) \longrightarrow V_0 = W$, where the rewrite rules are the equalities in γ_1 and γ_2.

Using a longer cycle, we can mix three or more different contexts or conditions in the same way.

4.1 Algebraic datatypes

Using this simple scheme, we can force two arbitrary existential variables to be related in a very precise way. We can for example use it to force a variable to be of a simple algebraic type.

Suppose you want to force a variable $X : \iota^n \longrightarrow \iota$, to be built on a selection $\{C_i\}$ of m constructors, and to use a strict subset S of its n arguments. Let us denote by the variables (x_j) the p arguments of X that are in S, (y_k) the q arguments of X that are not in S.

We pick an additional variable $\star$. We want to find two contexts, inducing two equalities $=_1$ and $=_2$ such that their composition allows these rewriting steps:

$$\begin{array}{ll} \forall i \in \{1 \dots m\} & \star =_2=_1 C_i(\bar{\star}) \\ \forall j \in S, & \star =_2=_1 x_j \end{array}$$

Then, X is of the desired datatype if and only if $\star(=_2=_1)^* X$.

To do that, we add another m variables (z_i). We pick $\gamma_1 = \forall i, z_i = C_i(\bar{\star})$ and $\gamma_2 = \forall i, z_i = \star \wedge \forall j, x_j = \star$. Using this, we have $\star =_2 z_i =_1 C_i(\bar{\star})$, and $\star =_2$

$x_j =_1 x_j$. Putting everything together, we use a cycle between two existential variables $Y, Z : \iota^{p+q+m+3} \longrightarrow \iota$. Assuming that S represents the first p of the arguments of X, X has the desired structure if and only if there exists solutions to the following equations:

$$\begin{array}{ll} \widehat{Y} = \widehat{Z} & [x_1 \ldots x_p \; y_1 \ldots y_q \; C_1(\bar{\star}) \ldots C_m(\bar{\star}) \star \diamond \quad \diamond] \\ \widehat{Y} = p(\widehat{X}, \widehat{Z}) & [\star \ldots \star \quad y_1 \ldots y_q \quad \star \ldots \star \quad \star \diamond p(\star, \diamond)] \end{array}$$

5 Undecidability of first-order formulas

We show in this section how to use the cyclic dependencies to embed simple arithmetic operations, as well as translate any second-order unification problem into an argument-constrained second-order unification problem

5.1 Embedding of Diophantine equations

Goldfarb [4] encoded natural numbers, addition, and multiplication in second-order unification problems in order to show its undecidability. The encoding is as follows:

Suppose we have a unary constructor s and a ternary constructor t.

$X : \iota \longrightarrow \iota$ is the representation of a natural if and only if $X\ (s(a)) = s(X\ a)$

$X_1 + X_2 = X_3$ if and only if $X_2\ (X_1\ a) = X_3\ a$

$X_1 X_2 = X_3$ if and only if there exists $Y : \iota^3 \longrightarrow \iota$, such that

$$Y\ a\ b\ t(X3\ a, X2\ b, \star) = t(a, b, Y\ (X1\ a)\ s(b)\ \star)$$

In a way, this equation says that one can go from 0 to X_3 by adding X_1 in exactly the same number of steps as one goes from 0 to X_2 by adding 1.

The translation into argument-constrained second-order unification is as follows : A natural is a solution $(X : \iota \longrightarrow \iota, Y : \iota^2 \longrightarrow \iota)$ to the equations

$$\begin{array}{ll} \widehat{X} = \widehat{Y} & [a \quad a] \\ \widehat{X} = s(\widehat{Y}) & [s(a)\ a] \end{array}$$

A solution $(X = s^n(\pi_1), Y = s^n(\pi_2)) = \bar{n}$ to this system corresponds to the natural n.

If $\bar{n_i} = (X_i, Y_i)$, then $n_1 + n_2 = n_3$ if and only if

$$\widehat{X_3} = \widehat{Y_2}\ [a\ \widehat{X_1}]$$

Indeed, after applying the context, $\widehat{X_3} = X_3\ a = s^{n_3}(a)$ and $\widehat{Y_2} = Y_2\ a\ (X_1\ a) = Y_2\ a\ (s^{n_1}(a)) = s^{n_2}(s^{n_1}(a))$

And finally, $n_1 n_2 = n_3$ if and only if there is a solution $(Z, Z' : \iota^6 \longrightarrow \iota)$ to

$$\begin{array}{ll} \widehat{Z} = \widehat{Z'} & [a\ b\ \ a \quad b\ \ \star \quad \star] \\ \widehat{Z} = t(\widehat{X_3}, \widehat{Y_2}, \widehat{Z'}) & [a\ b\ \widehat{X_1}\ s(b)\ \star\ t(a, b, \star)] \end{array}$$

with the additional constraint that Z is built with the constructors $t()$, $s()$, but must not use arguments 1 and 2, while Z' is built with the same constructors, and must not use arguments 3 and 4. So there must be solutions $(V, V', W, W' : \iota^{11} \longrightarrow \iota)$ to

$$\begin{array}{ll}
\widehat{V} = \widehat{V'} & [a\ b\ c\ d\ e\ f\ s(\star)\ t(\star,\star,\star)\ \star\ \diamond\ \ \ \diamond] \\
\widehat{V} = p(\widehat{Z}, \widehat{V'}) & [a\ b\ \star\ \star\ \star\ \star\ \ \star\ \ \ \ \star\ \ \ \star\ \diamond\ p(\star,\diamond)] \\
\widehat{W} = \widehat{W'} & [a\ b\ c\ d\ e\ f\ s(\star)\ t(\star,\star,\star)\ \star\ \diamond\ \ \ \diamond] \\
\widehat{W} = p(\widehat{Z'}, \widehat{W'}) & [\star\ \star\ c\ d\ \star\ \star\ \ \star\ \ \ \ \star\ \ \ \star\ \diamond\ p(\star,\diamond)]
\end{array}$$

These added constraints force the rewriting steps $a \longrightarrow X_1 a$ and $b \longrightarrow s(b)$ to always happen in the computation seen in the variables Z and Z', so they are always done synchronously.

For example, the Z and Z' encoding the statement $3 * 2 = 6$ are:

$$\begin{array}{c}
Z = t(s^3(\pi_3), s(\pi_4), t(\pi_3, \pi_4, \pi_6)) \\
Z' = t(s^3(\pi_1), s(\pi_2), t(\pi_1, \pi_2, \pi_5))
\end{array}$$

Those two terms are solutions of the system:

Applying the context $[a\ b\ a\ b\ \star\ \star]$ gives

$$\widehat{Z} = \widehat{Z'} = t(s^3(a), s(b), t(a, b, \star))$$

Applying the context $[a\ b\ s^3(a)\ s(b)\ \star\ t(a, b, \star)]$ gives

$$\widehat{Z} = t(s^6(a), s^2(b), \widehat{Z'}) = t(s^6(a), s^2(b), t(s^3(a), s(b), t(a, b, \star)))$$

If we do not restrain the arguments of Z and Z', we get all kinds of wrong computations, since the rewriting is no longer synchronous, such as $3 * 0 = 6$:

$$\begin{array}{c}
Z = t(s^3(\pi_3), \pi_2, t(\pi_3, \pi_2, \pi_6)) \\
Z' = t(s^3(\pi_1), \pi_2, t(\pi_1, \pi_2, \pi_5))
\end{array}$$

Applying the context $[a\ b\ s^3(a)\ s(b)\ \star\ t(a, b, \star)]$ gives

$$\widehat{Z} = t(s^6(a), b, \widehat{Z'}) = t(s^6(a), b, t(s^3(a), s(b), t(a, b, \star)))$$

which would be valid if not for the type constraints on Z and Z'.

Using these embeddings, one can embed any Diophantine problem into an argument-constrained second-order unification problem. In particular, this is enough to show using the undecidability of Hilbert's tenth Problem, that this class of problems is undecidable.

In our construction, we had problematic cycles put on top of each other. There are cycles between the variables V, V', W, W', which constrains the variables Z, Z'. In turn, there is a cycle between Z, Z' that constrains the variables X_i, Y_i. And lastly, there are cycles between X_i, Y_i that forces them to be natural numbers.

So one could try to define a notion of cycle height describing the complexity of the problem. A cycle height of three is enough to embed undecidable problems. But it is not known whether less complex problems with cycle height one or two are decidable.

5.2 Embedding second-order unification problems

Using these structure-enforcing cycles, one can also translate a standard second-order unification problem into an argument-constrained second-order unification problem, and if one wishes, into the satisfiability of a Φ-formula.

We want to translate a single equation $t = t'$, solving for existential variables $Y_i : \iota^{n_i} \longrightarrow \iota$. We will need to declare the set S of constructors C_i appearing in the equation. If there is a solution to the original problem using constructors that do not explicitly appear in the problem, then those constructors can be uniformly changed into anything else, so we do not lose completeness if we are only searching for solutions using only this selection S of constructors.

If necessary, we eta-expand the terms t and t' in the equation so that every occurrence of a Y_i does come with all its n_i arguments. We will need to make distinct variables for each occurrence of Y_i in the problem. If the j^{th} occurrence of Y_i is $Y_i \quad t_1 \dots t_{n_i}$, we create one existential variable Y_i^j and n_i variables $x^j_{(i,1)} \dots x^j_{(i,n_i)}$, and we name $t^j_{(i,k)} = t_k$.

We have to order all the $x^j_{(i,k)}$ and the Y_i^j so that the dependency order is compatible with the sub-term ordering: If there is an occurrence $Y_{i'}^{j'}$ in $t^j_{(i,k)}$, then we want to have $Y_{i'}^{j'} < x^j_{(i,k)}$. We also need to have $x^j_{(i,k)} < Y_i^j$ in order for Y_i^j to be able to depend on its intended arguments.

We pose the constraint that Y_i^j are terms depending on the $x^j_{(i,k)}$ only, and using the finite set of constructors S. In order to do this, we need two extra existential variables and a cycle between two contexts that describe all the type of all the Y_i^j.

We pose the constraint that every Y_i^j represents the same variable $Y_i : \iota^{n_i} \longrightarrow \iota$:

$$\left(\bigwedge x^j_{(i,k)} = x^{j'}_{(i,k)}\right) \longrightarrow \bigwedge Y_i^j = Y_i^{j'}$$

Then we properly translate the equation $t = t'$:

$$\left(\bigwedge x^j_{(i,k)} = \widetilde{t^j_{(i,k)}}\right) \longrightarrow t = t'$$

where $\widetilde{t^j_{(i,k)}}$ is the term $t^j_{(i,k)}$ with each occurrence of an old existential variable Y_i replaced with its corresponding new existential variable Y_i^j.

If we ordered the variables and existential variables in an order compatible with the subterm ordering, it ensures that the conditions of these two equations admit a context (they have a most general unifier whatever Y_i^j may be instantiated into).

Then this new unification problem can be translated as a first-order formula, and has a solution if and only if the original second-order unification problem has one.

Thus we only need four distinct contexts to have a simulation of arbitrary second-order unification problem. It is not known if a problem with three or less distinct contexts is decidable.

Example. Consider the second-order unification problem $Y\ a = Y\ b$ where $Y : \iota \to \iota$ and a and b are two constant constructors. The solutions to this problem are those substitutions for Y that involve a top-level vacuous binder. We have two occurrences of Y, so we will need two existential variables Y_1, Y_2 and two variables as their arguments, x_1, x_2. When ordering the variables, the only constraints are $x_i < Y_i$, so we can choose the following ordering in the prefix:

$$\forall x_1 \exists Y_1 \forall x_2 \exists Y_2$$

meaning we have the types $Y_1 : \iota \to \iota, \quad Y_2 : \iota^2 \to \iota$.

The type constraints are simple. Y_1 can only depend on x_1 so we do not need to add any constraint on Y_1. On the other hand, we have to make sure Y_2 does not use x_1. In fact, Y_2 can only be x_2, a, or b. We do not need to use a cycle to force this, instead we say that there exists $Z_2 : \iota^4 \to \iota$ such that

$$\begin{aligned} \widehat{Y_2} &= \widehat{Z_2}\ [x_1\ x_2\ \ a\quad b] \\ \widehat{Z_2} &= x_2\ [x_1\ x_2\ x_2\ x_2] \end{aligned}$$

The rest of the translation is

$$\begin{aligned} \widehat{Y_1} &= \widehat{Y_2}\ [x\ x] \\ \widehat{Y_1} &= \widehat{Y_2}\ [a\ \ b] \end{aligned}$$

The solutions to the translated problems are the solutions of the original problems that only use the constructors a and b, so there are two of them. For example, the whole solution corresponding to $Y = a$ is:

$$\begin{aligned} Y_1 &= \lambda x_1. && a \\ Y_2 &= \lambda x_1 x_2. && a \\ Z_2 &= \lambda x_1 x_2 x_3 x_4. && x_3 \end{aligned}$$

End of example.

6 Related work and conclusions

There has, of course, been a lot of work already done with solving inequalities within, say, a logic programming setting, under the topics "disunification" and "term-complementation": see, for example, the early papers by Barbuti *et. al.* [1] and Maher [5]. This problem has also been studied in the setting of higher-order logic by Momigliano and Pfenning [8]. Most of these papers consider reasoning about exact nature of the signature of constants over which unification and disunification is carried out. Here, this is less important since we are simply considering the reduction of one kind of unification problem to another and not with the listing of their complete set of solutions.

Acknowledgments. We thank Gopalan Nadathur for his comments on the first part of an earlier draft of this abstract.

References

1. R. Barbuti, P. Mancarella, D. Pedreschi, and F. Turini. Intensional negation of logic programs: Examples and implementation techniques. In *Proc. of the TAPSOFT '87*, number 250 in LNCS, pages 96–110. Springer, 1987.
2. J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
3. J.-Y. Girard. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu, Feb. 1992.
4. W. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
5. M. J. Maher. Complete axiomatizations of the algebras of finite rational and infinite trees. In *Proceedings of LICS*, pages 348–357, 1988.
6. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.
7. D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.
8. A. Momigliano and F. Pfenning. Higher-order pattern complement and strict λ-calculus. *ACM Trans. on Computational Logic*, 4(4):493–529, Oct. 2003.
9. P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.