

The undecidability of proof search when equality is a logical connective

Dale Miller, Alexandre Viel

▶ **To cite this version:**

Dale Miller, Alexandre Viel. The undecidability of proof search when equality is a logical connective. Annals of Mathematics and Artificial Intelligence, 2021, 10.1007/s10472-021-09764-0 . hal-03457312

HAL Id: hal-03457312
https://hal.science/hal-03457312

Submitted on 9 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

The undecidability of proof search when equality is a logical connective

Dale Miller and Alexandre Viel

the date of receipt and acceptance should be inserted later

Abstract One proof-theoretic approach to equality in quantificational logic treats equality as a logical connective: in particular, term equality can be given both left and right introduction rules in a sequent calculus proof system. We present a particular example of this approach to equality in a first-order logic setting in which there are no predicate symbols (apart from equality). After we illustrate some interesting applications of this logic, we show that provability in this logic is undecidable.

Keywords: equality, unification, undecidability, sequent calculus

Draft: June 18, 2021

1 Introduction

An elegant proof-theoretic treatment of equality for first-order terms was given independently by Girard [13] and Schroeder-Heister [29] in the early 1990s. Since then, their treatment has been extended to include the $\beta\eta$-equality of simply typed λ-terms [21], and that extension has been built into the Bedwyr model checker [3] and Abella theorem prover [2]. We shall show that adding this treatment of equality to a weak first-order logic makes provability undecidable. This fact is surprising since we only involve first-order terms and first-order quantification: in particular, the structural rule of contraction is not involved in the construction of proofs.

In order to motivate the logic we consider, recall that most unification problems in first-order logic can be considered to be simple, quantified formulas involving only equations of the form

$$\exists x_1 \ldots \exists x_m [t_1 = t'_1 \wedge \cdots \wedge t_n = t'_n].$$

Proving such formulas is well known to be decidable. When designing proof search procedures for intuitionistic logic, a more general form of unification is usually

Affiliation and address for both authors: Inria Saclay & LIX/Ecole Polytechnique, 1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France
Contact author: Dale Miller <dale.miller@inria.fr>

considered. These are represented by formulas of the form

$$\mathcal{Q}x_1 \dots \mathcal{Q}x_m[t_1 = t'_1 \wedge \cdots \wedge t_n = t'_n],$$

where $\mathcal{Q}_i$ is either $\forall$ or $\exists$. In intuitionistic and higher-order logics, Skolemization is sometimes unsound or undesirable, so it is not generally possible to reduce all such mixed quantifier prefixes to only existential quantifiers [23]. In any case, the provability (in classical and intuitionistic logic) of such first-order, mixed prefix formulas is still decidable. In this paper, we allow an equation to be replaced with, for example, an implication between equations. If we have two unequal terms, for example, z (zero) and $(s\ z)$ (one), then the inequality $t \neq t'$ can be written as $t = t' \supset z = (s\ z)$. As we shall show, this simple extension yields formulas for which provability is undecidable. Thus, systems that automate proof search involving such equality cannot expect to solve all unification problems completely prior to making other choices in theorem proving.

2 First-order quantification and term equality

By first-order logic *without* equality, we mean the usual notion of logic in which we have *logical connectives*, such as conjunction, disjunction, implication, negation, true, false, and universal and existential quantifiers. We shall also assume that we can form first-order terms (over some specific signature). Finally, *atomic formulas* are built by applying, for example, an n-ary predicate symbol P to a list $t_1, \dots, t_n$ of first-order terms to get the formula $P(t_1, \dots, t_n)$. Note that predicate symbols are *non-logical* symbols. In sequent calculi presentations of first-order logic, there are introduction rules for logical connectives: there are no introduction rules for predicates. Church [5] and Turing [30] independently proved that it is undecidable to determine if a first-order logic formula is provable.

The usual approach to adding equality to first-order logic is to first introduce a binary predicate to denote equality and then to axiomatize that predicate with the rules for equivalence (the axioms of reflexivity, symmetry, and transitivity) and congruence (for every constructor[1] of terms): see, for example, the textbook [9]. Sometimes, inequality is also axiomatized: see, for example, the Clark completion approach to capturing negation-as-failure [6].

We shall take a different but popular approach to adding term equality to first-order logic by introducing equality as a *logical connective* for which our sequent proof system will have a left and right introduction rule, following Girard [13] and Schroeder-Heister [29]. The De Morgan dual of equality, namely inequality, is also captured in this proof-theoretic setting. Note that when equality is a logical connective, a formula of the form $t = t'$ is not atomic since its top-level symbol is not a non-logical symbol. Equality in this paper will be interpreted as strict, syntactic equality. The main novelty of the approach to equality here is its treatment of equality on *open* terms. Here, free variables in terms are *eigenvariables* within sequent calculus proofs: such variables were used by Gentzen to connect certain occurrences of quantified expressions with their open and immediate subformula, as illustrated by the $\forall_R$ rule given later in this section.

[1] A constructor is also called a *function symbol*.

We shall assume that we have exactly one primitive type ι, and that the bound variables in all existential and universal quantifiers have this type. Non-primitive types are used to specify the arity of constructors: the type $\iota \to \cdots \to \iota \to \iota$, with $n+1$ occurrences of ι, is the type of a constructor of arity $n \geq 0$. We shall take as fixed, a signature Σ_0 of first-order constructors. For example, in Section 4, Σ_0 is the set $\{s\colon \iota \to \iota, p\colon \iota \to \iota \to \iota, k\colon \iota \to \iota \to \iota \to \iota\}$.

Sequents are written as the triple $\Sigma :\ \Gamma \vdash \Delta$ where Σ is a set of *eigenvariables* and Γ and Δ are multisets of formulas whose free variables must be members of Σ. We shall write $\Sigma \Vdash t\colon \iota$ to denote the fact that t has type ι using the constants in Σ_0 and the variables in Σ. The rules for first-order quantifiers are

$$\frac{\Sigma \Vdash t : \iota \qquad \Sigma :\ [t/x]B, \Gamma \vdash \Delta}{\Sigma :\ \forall x.B, \Gamma \vdash \Delta}\ \forall_L \qquad \frac{x\colon \iota, \Sigma :\ \Gamma \vdash B, \Delta}{\Sigma :\ \Gamma \vdash \forall x.B, \Delta}\ \forall_R$$

$$\frac{x\colon \iota, \Sigma :\ B, \Gamma \vdash \Delta}{\Sigma :\ \exists x.B, \Gamma \vdash \Delta}\ \exists_L \qquad \frac{\Sigma \Vdash t : \iota \qquad \Sigma :\ \Gamma \vdash [t/x]B, \Delta}{\Sigma :\ \Gamma \vdash \exists x.B(x), \Delta}\ \exists_R$$

In the $\forall_R$ and $\exists_L$ rules, the eigenvariable x is assumed to not be in Σ and, as a consequence, it is not free in the concluding sequent. We shall assume that the names of quantifiers can be α-converted as needed and that substitution, denoted as $[t/x]B$, is capture-avoiding. As a result, we will generally assume that the name of a universally bound variable is the same as the name of the eigenvariable that instantiates it in the $\forall_R$ and $\exists_L$ rules.

The introduction rules for first-order equality are the following.

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

The eq_L rule is rewritten as two rules: the proviso $\dagger$ requires that t and t' are not unifiable and the proviso $\ddagger$ requires that t and t' are unifiable with the 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 adding all those variables that are free in some term in the range of θ. Note that when the proviso $\ddagger$ holds, the premise of eq_L will have one fewer occurrences of logical connectives but it may have more complex term structures.

It is easy to prove that equality is, for example, an equivalence relation by using these inference rules. For example, by remembering that the standard sequent calculus rule for introducing $\supset$ on the right is given as

$$\frac{\Sigma :\ \Gamma, B \vdash C}{\Sigma :\ \Gamma \vdash B \supset C}\ \supset_R,$$

we can prove the transitivity of equality as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x\colon \iota, w\colon \iota :\ \cdot \vdash x = x}\ eq_R}{x\colon \iota, w\colon \iota :\ x = w \vdash x = w}\ eq_L,\ \text{replace } w \text{ with } x}{x\colon \iota, y\colon \iota, w\colon \iota :\ x = y, y = w \vdash x = w}\ eq_L,\ \text{replace } y \text{ with } x}{x\colon \iota, y\colon \iota, w\colon \iota :\ \cdot \vdash x = y \supset y = w \supset x = w}\ \supset_R \times 2}{\cdot :\ \cdot \vdash \forall x \forall y \forall w.(x = y \supset y = w \supset x = w)}\ \forall_r \times 3$$

Besides being an equivalence relation, equality is also a congruence. This property of our equality is easily established for all constructors. For example, let Σ_0

contain the constructors $z\colon \iota$ and $s\colon \iota \to \iota$ (encoding zero and successor). The following proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x\colon \iota\,;\, \cdot \vdash x = x}\, eq_R}{x\colon \iota, y\colon \iota\,;\, (s\ x) = (s\ y) \vdash x = y}\, eq_L,\ \text{replace } y \text{ with } x}{x\colon \iota, y\colon \iota\,;\, \cdot \vdash (s\ x) = (s\ y) \supset x = y}\, \supset_R}{\cdot\,;\, \cdot \vdash \forall x.\forall y.(s\ x) = (s\ y) \supset x = y}\, \forall_R$$

proves that successor is injective. We can also directly proved that zero is not a successor.

$$\dfrac{\dfrac{\dfrac{}{x\colon \iota\,;\, (s\ x) = z \vdash \bot}\, eq_L}{x\colon \iota\,;\, \cdot \vdash (s\ x) = z \supset \bot}\, \supset_R}{\cdot\,;\, \cdot \vdash \forall x.(s\ x) = z \supset \bot}\, \forall_R$$

Note that the above application of the eq_L rule is justified by the fact that the terms z and $(s\ x)$ do not unify. Thus, the failure of unification leads to a successful proof. (Here, we can replace the symbol $\bot$ for false with, say, $z = (s\ z)$.) As a consequence of these observations, this proof system proves all of Peano's axioms except for the one regarding induction.

For another example, abbreviate the terms z, $(s\ z)$, $(s\ (s\ z))$, $(s\ (s\ (s\ z)))$, etc by $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, $\mathbf{3}$, etc. Let the sets $A = \{\mathbf{0}, \mathbf{1}\}$ and $B = \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$ be encoded as

$$\{x \mid x = \mathbf{0} \vee x = \mathbf{1}\} \quad \text{and} \quad \{x \mid x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2}\}.$$

By assuming the usual left and right introduction rules for $\vee$, namely,

$$\dfrac{\Sigma\,:\, \Gamma, B \vdash E \qquad \Sigma\,:\, \Gamma, C \vdash E}{\Sigma\,:\, \Gamma, B \vee C \vdash E}\, \vee_L \quad \text{and} \quad \dfrac{\Sigma\,:\, \Gamma \vdash B_i}{\Sigma\,:\, \Gamma \vdash B_1 \vee B_2}\, \vee_R\ (i = 1, 2),$$

we can prove that A is a subset of B: that is, $\forall x.\, x \in A \supset x \in B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\,;\, \cdot \vdash \mathbf{0} = \mathbf{0}}}{\cdot\,;\, \cdot \vdash \mathbf{0} = \mathbf{0} \vee \mathbf{0} = \mathbf{1} \vee \mathbf{0} = \mathbf{2}}}{x\colon \iota\,;\, x = \mathbf{0} \vdash x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2}} \qquad \dfrac{\dfrac{\dfrac{}{\cdot\,;\, \cdot \vdash \mathbf{1} = \mathbf{1}}}{\cdot\,;\, \cdot \vdash \mathbf{1} = \mathbf{0} \vee \mathbf{1} = \mathbf{1} \vee \mathbf{1} = \mathbf{2}}}{x\colon \iota\,;\, x = \mathbf{1} \vdash x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2}}}{\dfrac{x\colon \iota\,;\, x = \mathbf{0} \vee x = \mathbf{1} \vdash x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2}}{\cdot\,;\, \cdot \vdash \forall x.(x = \mathbf{0} \vee x = \mathbf{1}) \supset (x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2})}}$$

Here, the doubled horizontal line indicates that more than one inference rule is applied. We leave it as an exercise to the reader to prove the negation of the converse of this formula, namely, $\exists x.(x \in B \wedge (x \in A \supset \bot))$. If one also adds to this logic a fixed point operator and the inference rules for unfolding such fixed points, it is possible to capture arbitrary Horn clause specifications as well as several model checking style queries [3,15].

This approach to the treatment of quantification and term equality has been shown to satisfy cut-elimination in classical, intuitionistic, and linear logic settings [13,29]. Given that unification is used to introduce equations, this approach to term equality has been extended to terms containing bindings by replacing first-order unification with higher-order unification [8,21,25]. The logic and proof search underlying the Abella theorem prover [2] and the Bedwyr model checker [3] incorporate these particular extensions.

$$\frac{\Sigma:\ B \vdash C}{\Sigma:\ \cdot \vdash B \supset C}\ \supset_R \qquad \frac{\Sigma:\ \cdot \vdash B \qquad \Sigma:\ \cdot \vdash C}{\Sigma:\ \cdot \vdash B \wedge C}\ \wedge_R$$

$$\frac{x\colon \iota, \Sigma:\ \cdot \vdash B}{\Sigma:\ \cdot \vdash \forall x.B}\ \forall_R \qquad \frac{\Sigma \Vdash t : \iota \qquad \Sigma:\ \cdot \vdash [t/x]B}{\Sigma:\ \cdot \vdash \exists x.B(x)}\ \exists_R$$

$$\frac{}{\Sigma:\ t = t' \vdash E}\ eq_L\dagger \qquad \frac{\theta\Sigma:\ \cdot \vdash \theta E}{\Sigma:\ t = t' \vdash E}\ eq_L\ddagger \qquad \frac{}{\Sigma:\ \cdot \vdash t = t}\ eq_R$$

The provisos † and ‡ are the same as presented in Section 2. In the $\forall_R$ rule, we assume that the bound variable x is not a member of Σ.

Fig. 1 The intuitionistic proof system $\mathcal{I}$

In Gentzen's original sequent systems LK and LJ [11], eigenvariables were essentially scoped constants in the sense that in a cut-free proof, once an eigenvariable is introduced into a proof (reading inference rules from conclusion to premises), that eigenvariable did not vary. Eigenvariables were substituted for only during cut-elimination. While those systems of Gentzen did not involve unification in the proof system, proof search algorithms, such as those described for logic programming [22,27], involve mixing unification and introduction rules. In particular, unification is used to determine the terms to use for instantiating the quantifiers in the $\forall_L$ and $\exists_r$ rules. The treatment of the left-introduction rule for equality changes those basic principles: unification is now part of the description of an inference rule and eigenvariables can vary over the course of building a cut-free proof.

As we have mentioned previously, this approach to equality stands in contrast to other approaches where equality is a non-logical predicate that is axiomatized as an equivalence relation and a congruence. One point of comparison is that in the setting used here, we have an immediate specification of both equality and inequality: however, incorporating additional equality rules need to be done separately (as is done for the rules of λ-conversion in [21]). In contrast, when equality is a non-logical predicate given an explicit axiomatization, inequality needs to be addressed separately while additional equations (such as commutativity) can be accommodated as just additional axioms.

3 A subset of the logic

From the perspective of searching for sequent calculus proofs, the source of undecidability in first-order logic usually comes from the contraction rule. For example, in intuitionistic logic, a universally quantified assumption might need to be instantiated multiple times. In classical logic, Herbrand's theorem tells us that an existentially quantified formula might need to be instantiated multiple times to form a tautologous disjunction. In general, there is no *a priori* bound on the number of instances of some formulas that might be needed to complete a proof.

It might seem surprising then that provability in the logic we study in this paper is also undecidable even though our proof system is linear in the sense of linear logic [12] (i.e., there are no contraction and weakening rules). The goal of this paper is to prove this result.

In order to strengthen our undecidability result, we consider only formulas defined by the following recursive definition.

$$\begin{array}{rl} \Phi & ::= \Phi \wedge \Phi \mid \exists x.\Phi \mid \forall x.\Phi \mid \Psi \\ \Psi & ::= t_1 = t_1' \supset \cdots \supset t_n = t_n' \supset t_0 = t_0' \quad (n \geq 0) \end{array}$$

The proof system $\mathcal{I}$ for these logical formulas is given in Figure 1. When we say that the formula Φ is provable, we mean that the sequent $\cdot : \cdot \vdash \Phi$ has a proof: that is, we consider proving Φ with no initial collection of eigenvariables and with no formulas on the left. We state here several facts about the $\mathcal{I}$ proof system.

1. If a sequent has a formula on the left of the sequent arrow, that formula must be an equation, and the formula on the right must be a Ψ-formula. Also, this sequent must be the conclusion of one of the eq_L rules. Right introduction rules can only be applied to sequents with an empty left-hand side.
2. When reading proofs from conclusion to premises, the only rule that is non-deterministic is the $\exists_R$ since the choice of t is unconstrained. In general, there can be an infinite number of terms t such that $\Sigma \Vdash t : \iota$ (depending on Σ_0).
3. If the signature Σ_0 contains enough constructors to build a (closed) term, say, t, then the logical constant for true can be defined to be $t = t$ (in linear logic, this yields the multiplicative true). If there are at least two different Σ_0-terms t and t' then the logical false formula can be defined to be $t = t'$ (in linear logic, this yields the additive false).
4. Let $\mathbf{LJ}^{=}$ be Gentzen's **LJ** proof system in which the rules eq_R and eq_L from Figure 1 have been added. A Φ-formula is provable in $\mathbf{LJ}^{=}$ if and only if it is provable in this proof system.
5. Let $\mathbf{LL}^{=}$ be version of a two-sided proof system for linear logic **LL** in which the rules eq_R and eq_L from Figure 1 have been added. Since $\mathcal{I}$ does not contain the structural rules of weakening or contraction (as defined by Gentzen [11]), it follows that provability in $\mathbf{LJ}^{=}$ and $\mathbf{LL}^{=}$ coincides for Φ-formulas.

All of these facts follow directly from the following observation: If we interpret the logical symbols $\supset$ and $\wedge$ as the linear logic connectives $\multimap$ and &, respectively, then a Φ-formula is provable in linear logic if and only if it is provable in this proof system. This result follows immediately from the completeness of the focused proof system for μMALL given in [1] since μMALL contains both multiplicative additive linear logic and the equality used here.

If one eliminates the presence of implications in Φ-formulas, then we are left with essentially the same formulas that have been used in [23] to develop an approach to *unification under a mixed prefix*. As in that paper, we shall explicitly deal with quantifier alternation and will not attempt to simplify it using Skolemization.

4 Encoding some arithmetic

For the rest of this paper, we shall fix the set of sorts S to be $\{\iota\}$ and the signature Σ_0 to be

$$\{s\colon \iota \to \iota,\ p\colon \iota \to \iota \to \iota,\ k\colon \iota \to \iota \to \iota \to \iota\}.$$

We shall use the constructor s as the successor function when representing natural numbers. Note, however, that we do not admit a constructor for zero in this

signature. In fact, it is the case that there are no Σ_0-terms of type ι, a fact that plays an important role in our examples and main result. The constructor p will be used in Section 5 for building lists of terms, while the constructor k will be used to build lists of pairs of terms in Section 6.

Consider attempting a proof of a sequent of the form $\cdot : \cdot \vdash \forall a \forall b \exists X.B(a, b, X)$. An attempt to prove this sequent results in an attempt to prove

$$a\colon \iota, b\colon \iota : \cdot \vdash B(a, b, t),$$

where t is a term built from the variables $\{a\colon \iota, b\colon \iota\}$ and the constructors in Σ_0. If the only member of Σ_0 used in constructing t is the successor function s then t is either of the form $(s^n a)$ or $(s^n b)$, where $n \geq 0$ is the number of iterations of the successor constructor. A term of the first kind is called the *a-nat* for n and a term of the second kind is called the *b-nat* for n.

The following proposition allows us to use provability to constrain the instantiation of an existential quantifier to being either an *a-nat* or a *b-nat*.

Proposition 1 *The formula*

$$\forall a \exists X \forall b \exists Y.[(a = b \supset X = Y) \wedge (a = (s\ b) \supset X = (s\ Y))] \tag{1}$$

has many proofs and these can be put into a one-to-one correspondence with the natural numbers. In particular, if $n \geq 0$ then there is a proof of this formula in which X is instantiated with the a-nat for n and Y is instantiated with the b-nat for n. Conversely, if this formula has a proof then the instantiations for X and Y in that proof are the a-nat and b-nat, respectively, for some $n \geq 0$.

While the proof of this proposition is straightforward, we provide some of its details. Let $n \geq 0$. It is straightforward to show that

$$[a = b \supset (s^n a) = (s^n b)] \wedge [a = (s\ b) \supset (s^n a) = (s^{n+1} b)]$$

has a (unique) proof. Conversely, assume that (1) has a proof and let T and R be the instantiations for X and Y, respectively. Note that the only difference between T and R is the occurrences of the eigenvariables a and b within them: they share all occurrences of the other constructors. Let θ be the substitution that replaces a with $(s\ b)$. The term R must contain an occurrence of b since otherwise $T\theta$ and $R\theta$ would be equal, contradicting the provability of the second conjunct in (1). We now argue that the constructors p and k are not present in either T or R. Assume, on the contrary, that T (and, hence, R) has an occurrence of p and let $l \geq 0$ be the distance that occurrence has to the root of T (and in R). However, that distance remains l in $T\theta$ but is $l + 1$ in $(s\ (R\theta))$, which means that these two terms are not equal, a contradiction. By a similar argument, the constructor k cannot occur in T and R. Hence, the only constructor these terms can contain is s. Thus, T is an *a-nat*, R is a *b-nat*, and they both encode the same natural number.

The following proposition shows how it is possible to capture the addition of natural numbers via provability. The proof of this proposition is immediate and not given.

Proposition 2 *Let n_1, n_2, and n_3 be natural numbers and let N_1 be the a-nat for n_1, N_2 be the b-nat for n_2, and N_3 be the a-nat for n_3. Then, $n_1 + n_2 = n_3$ holds if and only if $\forall a \forall b.[b = N_1 \supset N_2 = N_3]$ is provable.*

It is possible to characterize the multiplication of natural numbers as well, but, as we shall see, that is significantly more difficult.

5 Constraining instantiations

The order in which variables are quantified within a formula provides some restrictions on the terms used in substitutions within a proof. For example, the quantifier prefix in Proposition 1, namely, $\forall a \exists X \forall b \exists Y$ forbids the substitution instance for X to contain the eigenvariable b. It is possible to specify much more general restrictions on substitutions than those that arise from quantification. For example, we might find a need to restrict the terms that instantiate Y in proofs so that they do not contain occurrences of a.

As an exercise, consider proving a formula of the form $\forall x \forall y \exists X.B(x, y, X)$ in which we wish to have a proof if and only if the substitution term for X contains only the constructor k and the eigenvariable y (or, equivalently, such terms do not contain s, p, and the eigenvariable x). One approach to formalizing this is the following. Let $z_1, \ldots, z_5$, $\star$, and $\diamond$ be seven variables of type ι and consider the following four sets of equations.

$$\begin{array}{ll} \mathcal{E}_1 = \{z_1 = x,\ z_2 = (k \star \star \star)\} & \widehat{\mathcal{E}_1} = \mathcal{E}_1 \cup \{z_3 = \star,\ z_4 = \diamond,\ z_5 = \diamond\} \\ \mathcal{E}_2 = \{z_1 = \star,\ z_2 = \star\} & \widehat{\mathcal{E}_2} = \mathcal{E}_2 \cup \{z_3 = \star,\ z_4 = \diamond,\ z_5 = (p\ \star\ \diamond)\} \end{array}$$

Let us denote by $\overset{1}{=}$ and $\overset{2}{=}$ the equality between terms that use the equations in $\mathcal{E}_1$ and in $\mathcal{E}_2$, respectively. Consider the following sequence of alternating equations.

$$\begin{array}{c} (k\ (k\ x\ x\ x)\ x\ x) \overset{1}{=} (k\ (k\ z_1\ z_1\ z_1)\ z_1\ z_1) \overset{2}{=} (k\ (k\ \star\ \star\ \star)\ \star\ \star) \\ \overset{1}{=} (k\ z_2\ \star\ \star) \overset{2}{=} (k\ \star\ \star\ \star) \overset{2}{=} z_2 \overset{1}{=} \star \end{array}$$

Thus, the properly constrained term $(k\ (k\ x\ x\ x)\ x\ x)$ rewrites in this alternating fashion to the variable $\star$. It is not difficult to see that any term rewrites in this way to $\star$ if and only if that term satisfies our desired restriction.

Extending this example further, we now use the pairing constructor p in order to collect these various terms into the following list structures.

$$\begin{array}{l} L_1 = (p\quad (k\ (k\ x\ x\ x)\ x\ x)\quad (p\ \ (k\ (k\ \star\ \star\ \star)\ \star\ \star)\ (p\ (k\ \star\ \star\ \star)\ z_5))) \\ L_2 = (p\ (k\ (k\ z_1\ z_1\ z_1)\ z_1\ z_1)\ (p\qquad (k\ z_2\ \star\ \star)\qquad (p\qquad z_2\qquad z_4))) \end{array}$$

It is now easy to check that L_1 and L_2 are equal using the equations in $\widehat{\mathcal{E}_1}$ and that L_1 and $(p\ (k\ (k\ x\ x\ x)\ x\ x)\ L_2)$ are equal using the equations in $\widehat{\mathcal{E}_2}$.

We can now fully provide the formula that describes $B(x, y, X)$ requested above. In particular, the formula

$$\begin{array}{l} \forall z_1 \forall z_2 \forall z_3 \forall z_4 \forall z_5 \forall \star \forall \diamond \exists L_1 \exists L_2 [\\ \quad (z_1 = x \supset z_2 = (k\ \star\ \star\ \star) \supset z_3 = \star \supset z_4 = \diamond \supset z_5 = \diamond \qquad \supset L_1 = L_2) \wedge \\ \quad (z_1 = \star \supset z_2 = \star \qquad\qquad \supset z_3 = \star \supset z_4 = \diamond \supset z_5 = (p\ \star\ \diamond) \supset L_1 = (p\ X\ L_2))] \end{array}$$

is provable if and only if X satisfies the constraint that permits only occurrences of the constructor k and the variable x. Notice that the existential quantification on L_1 and L_2 essentially encodes the entire rewriting computation of the term instantiating X into the $\star$ variable.

In the next section, where we will encode the multiplication of natural numbers, we shall need a similar restriction on terms.

6 Encoding multiplication

In the previous section, we described a rewriting-style approach to showing that a substitution satisfies a certain constraint or not. Since the logic we are using does not allow for any iteration or recursion, the only way to capture computations that might take many steps (depending on input values) is to have the entire trace of a computation be encoded into a substitution term and then to use a formula to do a check of its correctness as a computational trace. We can characterize multiplication in a similar fashion by encoding traces that capture the interaction of addition.

For example, the list of pairs $[\langle 6,3\rangle, \langle 4,2\rangle, \langle 2,1\rangle, \langle 0,0\rangle]$ is evidence that 6 is the result of multiplying 2 times 3: here, when moving to the left, the second component of a pair is incremented by one while the first component is incremented by two. To encode such traces, we use the k constructor as follows: in the above example, if W encodes $[\langle 4,2\rangle, \langle 2,1\rangle, \langle 0,0\rangle]$ then

$$\Big(k\ \Big(s\ (s\ (s\ (s\ (s\ (s\ v_1))))))\Big)\ \Big(s\ (s\ (s\ v_2))\Big)\ W\Big)$$

encodes the full list of pairs. Here, v_1 and v_2 are some appropriately chosen eigenvariables.

In order to check that such a list of pairs correctly captures the trace of the computation of a multiplication operation, we can use a second trace and two conjunctive implications that allows us to check these two traces under two different substitutions. In particular, consider the following two terms.

$$Z = (k\ \ (s\ (s\ (s\ (s\ w_3))))\ (s\ (s\ w_4))\ (k\ \ (s\ (s\ w_3))\ (s\ w_4)\ (k\ \ w_3\ w_4\ w_6)))$$
$$Z' = (k\ \ (s\ (s\ (s\ (s\ w_1))))\ (s\ (s\ w_2))\ (k\ \ (s\ (s\ w_1))\ (s\ w_2)\ (k\ \ w_1\ w_2\ w_5)))$$

Note that these terms are the same modulo the renaming of free variables, and they both encode the list of pairs $[\langle 4,2\rangle, \langle 2,1\rangle, \langle 0,0\rangle]$. Now consider the following two sets of equations where $\bullet$ is a new variable.

$$\mathcal{E}_3 = \{w_1 = a,\ w_2 = b,\ w_3 = a,\quad w_4 = b,\quad\ \ w_5 = \bullet,\ w_6 = \bullet\}$$
$$\mathcal{E}_4 = \{w_1 = a,\ w_2 = b,\ w_3 = X_1,\ w_4 = (s\ b),\ w_5 = \bullet,\ w_6 = (k\ a\ b\ \bullet)\}$$

Using the equations in $\mathcal{E}_3$, it is immediate that Z and Z' are equal. If we use the equations in $\mathcal{E}_4$ instead, then Z is equal to $(k\ X_3\ Y_2\ Z')$. The fact that these two qualified equalities hold establishes that Z is a correct trace of a multiplication calculation.

Given this discussion, consider the provability of the formula in Figure 2. Proofs of this formula capture multiplication if and only if we can also guarantee the following additional restrictions.

1. The substitutions for $\exists X_1 \exists X_2 \exists X_3$ are *a-nat* n_1, n_2, n_3.
2. The substitution for $\exists Y_2$ is the *b-nat* for n_2.
3. The substitution for Z does not contain the eigenvariables w_1 and w_2 and the substitution for Z' does not contain the eigenvariables w_3 and w_4.

If these additional constraints hold, then we can conclude that $n_1 \times n_2 = n_3$. Proposition 1 can be used to produce formulas that solve the first and second of these restrictions. In fact, Figure 3 contains exactly such a formula. We now use

$$\begin{array}{l}\forall a \exists X_1 \exists X_2 \exists X_3 \forall b \exists Y_2 \forall w_1 \forall w_2 \forall w_3 \forall w_4 \forall w_5 \forall w_6 \forall \bullet \exists Z \exists Z' [\\ \quad (w_1 = a \supset w_2 = b \supset w_3 = a \quad \supset w_4 = b \quad \supset w_5 = \bullet \supset w_6 = \bullet \quad \supset Z = Z') \wedge \\ \quad (w_1 = a \supset w_2 = b \supset w_3 = X_1 \supset w_4 = (s\ b) \supset w_5 = \bullet \supset w_6 = (k\ a\ b\ \bullet) \supset \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad Z = (k\ X_3\ Y_2\ Z'))]\end{array}$$

Fig. 2 Partial specification of multiplication

$$\begin{array}{l}\exists Y_1.[(a = b \supset X_1 = Y_1) \wedge (a = (s\ b) \supset X_1 = (s\ Y_1))] \wedge \\ \qquad [(a = b \supset X_2 = Y_2) \wedge (a = (s\ b) \supset X_2 = (s\ Y_2))] \wedge \\ \exists Y_3.[(a = b \supset X_3 = Y_3) \wedge (a = (s\ b) \supset X_3 = (s\ Y_3))]\end{array}$$

Fig. 3 Specify that X_1, X_2, X_3 denote *a-nat* and that Y_1, Y_2, Y_3 are the corresponding *b-nat*.

$$\begin{array}{l}\forall u_1 \forall u_2 \forall u_3 \forall u_4 \forall u_5 \forall u_6 \forall u_7 \forall u_8 \forall u_9 \forall u_{10} \forall u_{11} \forall u_{12} \forall \star \forall \diamond \exists J_1 \exists J_2 \exists K_1 \exists K_2 [\\ \quad (\bigwedge \widehat{\mathcal{E}}_5 \supset J_1 = J_2) \ \wedge \ (\bigwedge \widehat{\mathcal{E}}_6 \supset J_1 = (p\ Z\ J_2)) \wedge \\ \quad (\bigwedge \widehat{\mathcal{E}}_7 \supset K_1 = K_2) \ \wedge \ (\bigwedge \widehat{\mathcal{E}}_8 \supset K_1 = (p\ Z'\ K_2))]\end{array}$$

Fig. 4 Specify restrictions on Z and Z'

the encoding technique illustrated in Section 5 to find a formula that solves the third restriction.

From the quantification prefix in front of $\exists Z \exists Z'$, the substitution instances of these variables can have the eigenvariables $a, b, w_1, w_2, w_3, w_4, w_5, w_6, \bullet$ and the constructors s and k. However, we need to insist that the term instantiating Z does not contain occurrences of w_1 and w_2 and the term instantiating Z' does not contain w_3 and w_4. The technique described in Section 5 allows us to build a formula that can specify such constraints on the instantiations of Z and Z'. We first define the following set of equations.

$$\begin{array}{ll}\mathcal{E}_5 = \{u_1 = a, u_2 = b, & u_3 = w_1, u_4 = w_2, u_5 = w_3, u_6 = w_4, u_7 = \bullet, u_8 = (k \star\star\star), \\ & \qquad\qquad\qquad\qquad\qquad\qquad u_9 = (s \star)\} \\ \mathcal{E}_6 = \{u_1 = \star, u_2 = \star, & u_3 = w_1, u_4 = w_2, u_5 = \star, \quad u_6 = \star, \quad u_7 = \star, u_8 = \star, \\ & \qquad\qquad\qquad\qquad\qquad\qquad u_9 = \star\} \\ \mathcal{E}_7 = \{u_1 = a, u_2 = b, & u_3 = w_1, u_4 = w_2, u_5 = w_3, u_6 = w_4, u_7 = \bullet, u_8 = (k \star\star\star), \\ & \qquad\qquad\qquad\qquad\qquad\qquad u_9 = (s \star)\} \\ \mathcal{E}_8 = \{u_1 = \star, u_2 = \star, & u_3 = \star, \quad u_4 = \star, \quad u_5 = w_3, u_6 = w_4, u_7 = \star, u_8 = \star, \\ & \qquad\qquad\qquad\qquad\qquad\qquad u_9 = \star\} \\ \widehat{\mathcal{E}}_5 = \mathcal{E}_5 \cup & \{\ u_{10} = \star, u_{11} = \diamond, \ u_{12} = \diamond\} \\ \widehat{\mathcal{E}}_6 = \mathcal{E}_6 \cup & \{\ u_{10} = \star, u_{11} = \diamond, \ u_{12} = (p\ \star\ \diamond)\} \\ \widehat{\mathcal{E}}_7 = \mathcal{E}_7 \cup & \{\ u_{10} = \star, u_{11} = \diamond, \ u_{12} = \diamond\} \\ \widehat{\mathcal{E}}_8 = \mathcal{E}_8 \cup & \{\ u_{10} = \star, u_{11} = \diamond, \ u_{12} = (p\ \star\ \diamond)\}\end{array}$$

The last four of these sets of equations are used in Figure 4: in that figure, we use the notion $\bigwedge \mathcal{E} \supset t = t'$ to denote the implication that has every equation in $\mathcal{E}$ as a hypothetical assumption to $t = t'$: the order in which these assumptions are listed is not important. Multiplication of natural numbers is characterized by the following proposition.

Proposition 3 *Consider the formula Φ that results from inserting the formulas in Figure 3 and Figure 4 immediately inside the scope of $\exists Z \exists Z'$ in the formula in Figure 2. The formula Φ has an $\mathcal{I}$ proof in which the instantiations of X_1, X_2, X_3 are a-nat terms encoding n_1, n_2, n_3 if and only if $n_1 \times n_2 = n_3$.*

7 Undecidability of provability

We now prove that provability in our logic is recursively undecidable. We do this by reducing Hilbert's Tenth Problem (regarding finding solutions to Diophantine equations), which is known to be undecidable, to the problem of provability. For some background to Hilbert's Tenth Problem, see Matiyasevich's book [20] or his survey article [19].

We use the fact that Diophantine equations can be restricted to have one of three forms: $x = 1$, $x + y = w$, and $x \times y = w$, where x, y, w are variables that range over non-negative integers. Thus, we need to show how to reduce a list of such equations into a single formula that is provable if and only if there is an assignment of natural numbers to variables such that all equations are true.

Assume that $X_1, \ldots, X_n$ is the list of variables in such a list of equations. We start the construction of our associated formula by writing the quantifier prefix $\forall a \exists X_1 \cdots \exists X_n \forall b \exists Y_1 \cdots \exists Y_n$. Within the scope of these quantifiers, we would then write the conjunction

$$\bigwedge_{i=1}^{n} [(a = b \supset X_i = Y_i) \wedge (a = (s\ b) \supset X_i = (s\ Y_i))].$$

From Proposition 1, these formulas force the instantiations for X_i and Y_i (for all $i = 1, \ldots, n$) to be *a-nat* and *b-nat* terms, respectively, encoding the same natural number.

Every equation of the form $X_i = 1$ translates to the additional conjunction $X_i = (s\ a)$. Every equation of the form $X_i + X_j = X_k$ translates to the formula (taken from Proposition 2) $[b = X_i \supset Y_i = X_k]$. Finally, every equation of the form $X_i \times X_j = X_k$ translates to the formula described in Proposition 3, except that X_1, X_2, X_3 are replaced by X_i, X_j, X_k, respectively.

Once we have collected all of these conjunctions and attached the quantifier prefix, the resulting formula is provable if and only if the Diophantine equations it encodes has a solution. Thus, provability is not decidable.

8 Related work

Goldfarb [14] has used an encoding of Hilbert's Tenth Problem to proved that second-order unification is undecidable, although the details of his encoding and the one given here are different. The first author [23] used a similar reduction to prove that it is undecidable to determine if the so-called *flexible-flexible* unification problems in Huet's pre-unification algorithm [16] have closed solutions.

A different interaction between unification and proof theory arises when one addresses the question of whether or not a given classical logic formula has a proof of a given length or size. Such problems can often be reduced to various kinds of

second-order unification problems [7,17] and, as a result, some of these problems are also known to be undecidable given the undecidability of second-order unification [14]. Still other interaction between unification and proof theory arises when implementing theorem provers in first-order classical logic. For example, when *simultaneous rigid E-unification* (introduced in [10]) replaces first-order unification in theorem provers based on Herbrand's theorem, a number of undecidability results are known and collected in [32].

Instead of using Skolemization to simplify quantifier alternation, it is possible to use a dual operation, called *raising* [23], that allows universal quantifiers to move in over existential quantifiers. Raising results in the use of higher-order quantification. We have chosen to remain in first-order logic in this paper. The unpublished report [31], on which this paper is based, provides a similar proof of undecidability while using higher-order quantification.

There has, of course, been much work already done with solving inequalities within, say, a logic programming setting, under the topics "disunification" and "term-complementation": see, for example, the papers by Barbuti *et. al.* [4] and Maher [18]. These papers encode equality using a non-logical predicate symbol that is axiomatized using Horn clause theories that are dependent on the signature of constants over which unification and disunification are carried out. This approach has also been used in the setting of higher-order logic by Momigliano and Pfenning [26].

While proof search in our setting here is undecidable, it might well be the case that in many applications and theoretical settings, only a restricted subset of unification is needed. Consider the case of higher-order unification [16]: while such unification is undecidable, it also appears that the unification problems generated by a wide range of applications (such as found in Isabelle [28] and λProlog [24]) are, in fact, computationally simple. Eventually, it was recognized that a much smaller subset of higher-order unification, now called higher-order pattern unification, was decidable and frequently occurred in practice [22]. Our experience with Bedwyr and Abella, which implement aspects of proof search in the $\mathcal{I}$ proof system, indicates that unification is not a source of computational difficulties. In those systems, recursion is available, and, as a result, there are more direct methods for encoding operations on natural numbers. When working with the Bedwyr system, the unification subsystem generally reduces to proving Φ-formulas, and those resulting formulas are usually simple to solve. This simplicity probably arises since inference in that model checking system is dominated by *additive synthetic inference rules*, which are formally defined in [15]. In such synthetic inference rules, existential substitutions (for example, for Y in Proposition 1) cannot be tested in two different implications via different instantiations to the same eigenvariable. If we limit ourselves to these additive synthetic inference rules, the formulas used to relate *a-nat* and *b-nat* terms and to encode multiplication would not be allowed, and proof search in $\mathcal{I}$ would simplify greatly.

9 Conclusions

We have considered a small logic involving only first-order quantification, conjunction, and hypothetical judgments involving term equality. Such formulas are naturally seen as generalizations to the kind of unification problems seen in systems

performing automated proof search in intuitionistic logic. Although this appears to be a mild extension to classical, first-order unification, we show that the problem of proving such formulas is undecidable.

References

1. D. Baelde. Least and greatest fixed points in linear logic. *ACM Trans. on Computational Logic*, 13(1):2:1–2:44, Apr. 2012.
2. D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2):1–89, 2014.
3. D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conf. on Automated Deduction (CADE)*, number 4603 in LNAI, pages 391–397, New York, 2007. Springer.
4. 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.
5. A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.
6. K. L. Clark. Negation as failure. In J. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
7. W. M. Farmer. The Kreisel length-of-proof problem. *Ann. Math. Artif. Intell*, 6(1-3):27–55, 1992.
8. A. Gacek, D. Miller, and G. Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011.
9. J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
10. J. H. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *2nd Symp. on Logic in Computer Science*, pages 338–346, Washington, D.C., USA, June 1987. IEEE Computer Society Press.
11. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935. Translation of articles that appeared in 1934-35. Collected papers appeared in 1969.
12. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
13. J.-Y. Girard. A fixpoint theorem in linear logic. An email posting archived at `https://www.seas.upenn.edu/~sweirich/types/archive/1992/msg00030.html` to the linear@cs.stanford.edu mailing list, Feb. 1992.
14. W. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
15. Q. Heath and D. Miller. A proof theory for model checking. *J. of Automated Reasoning*, 63(4):857–885, 2019.
16. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
17. J. Krajícek and P. Pudlák. The number of proof lines and the size of proofs in first order logic. *Arch. Math. Log*, 27(1):69–84, 1988.
18. M. J. Maher. Complete axiomatizations of the algebras of finite rational and infinite trees. In *3nd Symp. on Logic in Computer Science*, pages 348–357, 1988.
19. Y. Matiyasevich. Hilbert's tenth problem and paradigms of computation. In S. B. Cooper, B. Löwe, and L. Torenvliet, editors, *CiE: Computing in Europe*, number 3526 in LNCS, pages 310–321. Springer, 2005.
20. Y. V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, Cambridge, Massachusetts, 1993.
21. R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
22. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.
23. D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.

24. D. Miller and G. Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.
25. D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, Oct. 2005.
26. A. Momigliano and F. Pfenning. Higher-order pattern complement and strict λ-calculus. *ACM Trans. on Computational Logic*, 4(4):493–529, Oct. 2003.
27. G. Nadathur. A proof procedure for the logic of hereditary Harrop formulas. *Journal of Automated Reasoning*, 11(1):115–145, Aug. 1993.
28. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Number 828 in Science & Business Media. Springer, 1994.
29. P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.
30. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.
31. A. Viel and D. Miller. Proof search when equality is a logical connective. Unpublished draft presented to the International Workshop on Proof-Search in Type Theories, July 2010.
32. A. Voronkov. Simultaneous rigid E-unification and other decision problems related to the Herbrand theorem. *Theor. Comput. Sci*, 224(1-2):319–352, 1999.