Undecidability of Higher-Order Unification Formalised in Coq

Simon Spies
Saarland University
Saarland Informatics Campus, Saarbrücken, Germany
spies@ps.uni-saarland.de

Yannick Forster
Saarland University
Saarland Informatics Campus, Saarbrücken, Germany
forster@ps.uni-saarland.de

Abstract

We formalise undecidability results concerning higher-order unification in the simply-typed λ-calculus with β-conversion in Coq. We prove the undecidability of general higher-order unification by reduction from Hilbert's tenth problem, the solvability of Diophantine equations, following a proof by Dowek. We sharpen the result by establishing the undecidability of second-order and third-order unification following proofs by Goldfarb and Huet, respectively.

Goldfarb's proof for second-order unification is by reduction from Hilbert's tenth problem. Huet's original proof uses the Post correspondence problem (PCP) to show the undecidability of third-order unification. We simplify and formalise his proof as a reduction from modified PCP. We also verify a decision procedure for first-order unification.

All proofs are carried out in the setting of synthetic undecidability and rely on Coq's built-in notion of computation.

CCS Concepts • **Theory of computation** → **Lambda calculus**; **Type theory**.

Keywords higher order unification, synthetic undecidability, Coq

ACM Reference Format:
Simon Spies and Yannick Forster. 2020. Undecidability of Higher-Order Unification Formalised in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '20), January 20–21, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3372885.3373832

1 Introduction

Higher-order unification in the simply-typed λ-calculus is the problem of finding a substitution making two given typed terms convertible. We speak of *higher-order* unification because substitution may insert abstractions $\lambda x.s$ for variables.

While first-order unification was conceived as a method to implement automated resolution in classical logic, higher-order unification has a variety of applications nowadays. For instance, it is used as the foundation of programming languages such as λ-Prolog, in automated deduction, and in type-inference procedures for dependent-type theory. In particular, higher-order unification naturally arises when working in proof assistants based on type theory. With a right recursive definition of +, the proposition $\forall n.\ 0 + n = n$ can be proven using an application of the induction principle $\forall P.\ P(0) \rightarrow (\forall n.\ P(n) \rightarrow P(n + 1)) \rightarrow \forall n.\ P(n)$. Higher-order unification can be used to infer $P := \lambda n.0 + n = n$ to make $\forall n.\ 0 + n = n$ convertible to the conclusion $\forall n.\ P(n)$. Similarly, it allows for programming with implicit arguments and automated proof search.

In contrast to first-order unification, higher-order unification does not establish syntactic equality but equality up to conversion. This allows for more complicated solutions. For example, $\forall n.\ 0+n = n$ can also be proven with an application of the adapted induction principle $\forall P.\ P(1) \rightarrow (\forall n.\ P(n + 1) \rightarrow P(n + 2)) \rightarrow \forall n.\ P(n + 1)$ where higher-order unification may infer the predicate $P := \lambda n.0 + \text{pred}\ n = \text{pred}\ n$ assuming $\text{pred}(n + 1)$ is convertible to n.

The decision problem concerning higher-order unification can be analysed in general or at fixed orders. For instance, *second-order* unification only mentions terms with free variables of second-order type. In contrast, *third-order* unification is concerned with terms where variables have a type of at most order three.

In 1965, first-order unification was shown to be decidable by Robinson [1965] and even has linear decision algorithms [Martelli and Montanari 1976; Paterson and Wegman 1978]. In 1972, Huet [1972, 1973] and Lucchesi [1972] independently showed that third-order unification is undecidable, thereby establishing the undecidability of higher-order unification in general. In both cases the proof is by reduction from Post's correspondence problem [1946]. In 1981, Goldfarb [1981] proved that even second-order unification is undecidable by reducing from the solvability of Diophantine equations, more commonly known as Hilbert's tenth problem [Davis 1973; Matijasevivc 1970].

In this paper, we formalise decidability results concerning first- and higher-order unification for the Curry-style simply-typed λ-calculus with unrestricted β-conversion in the constructive type theory of Coq. Explicitly, we formalise the undecidability of second- and third-order unification (and thus higher-order unification in general) by simplifying the constructions of Goldfarb and Huet. We implement a verified decider for first-order unification in the λ-calculus and give enumerability proofs for all considered problems.

Definitions, lemmas, and theorems in the PDF-version of this document are hyperlinked with the accompanying Coq development; the statements are marked by the symbol 🐓.

We contribute our mechanisation to the Coq library of undecidable problems, available at

https://github.com/uds-psl/coq-library-undecidability/

Synthetic Undecidability Decidability results are commonly mechanised in Coq by calling a problem $\mathbf{P}$ represented by a predicate over a type X decidable if $\text{dec}\ \mathbf{P} := \exists f : X \to \mathbb{B}. \forall x.\ \mathbf{P}(x) \leftrightarrow f x = \text{true}$. This standard synthetic definition of decidability differs from traditional definitions omitting the condition that f is computable, justified by the fact that every function definable in constructive type theory is always computable in any model of computation.

The mechanisation of synthetic undecidability, introduced by Forster et al. [2019a], has been studied in a line of recent work, now integrated in a Coq library of undecidable problems [Forster et al. 2020b]. Synthetically, the negative notion of *un*decidability has to be defined carefully: simply defining it as $\text{dec}\ \mathbf{P} \to \bot$ is not sufficient due to the axiomatic freedom common to constructive type theories like Coq's. In Coq, the decidability of every problem $\mathbf{P}$ can be consistently assumed and thus $\text{dec}\ \mathbf{P} \to \bot$ is not provable for any $\mathbf{P}$. For instance, if we take **Halt** as a formalisation of the halting problem for Turing machines, dec **Halt** is an independent statement, meaning that neither dec **Halt** $\to \bot$ nor dec **Halt** can be proved in axiom-free Coq.

To define synthetic undecidability we make use of the non-provability of dec **Halt** and call a problem $\mathbf{P}$ synthetically undecidable if $\text{dec}\ \mathbf{P} \to$ dec **Halt** can be proved. Since the conclusion is not provable in axiom-free Coq, a proof of the implication must have started from an unprovable assumption. Furthermore, via the Curry Howard isomorphism the implication can be turned into a function in a concrete model of computation which transforms deciders for $\mathbf{P}$ to deciders of **Halt** — ensuring that $\mathbf{P}$ cannot be decidable in any Turing-complete model of computation.

On paper, undecidability proofs for a problem $\mathbf{P}$ are rarely carried out by appealing to the definition of decidability, but rather by giving a chain of computable many-one reductions from an undecidable problem $\mathbf{Q}$ to $\mathbf{P}$. In Coq, we can define $\mathbf{Q} \preceq \mathbf{P}$ as $\exists f. \forall x.\ \mathbf{P}(x) \leftrightarrow \mathbf{Q}(f x)$ and say that $\mathbf{Q}$ many-one reduces to $\mathbf{P}$. Similar to before, the computability of f can be omitted in the synthetic definition. Since many-one reductions transport decidability, we consider a reduction **Halt** $\preceq \mathbf{P}$ an undecidability proof for $\mathbf{P}$, because $\text{dec}\ \mathbf{P} \to$ dec **Halt** follows.

We build this work on top of the reductions **Halt** $\preceq$ **MPCP** mechanised by Forster, Heiter, and Smolka [2018] and the reduction **Halt** $\preceq$ **H10** mechanised by Larchey-Wendling and Forster [2019].

Contribution We contribute the first formalisation of undecidability for any unification problem. All our problems are presented as unification in a simply-typed, Curry-style λ-calculus allowing β-conversion in any context. We formalise the undecidability of higher-order unification in general following Dowek [2001], by reduction from Hilberts tenth problem **H10**. By simplifying the proof by Huet [1973], we formalise undecidability of third-order unification, by reduction from the modified Post correspondence problem **MPCP**. Lastly, by embedding the proof by Goldfarb [1981] into the full λ-calculus, we formalise the undecidability of second-order unification, by reduction from **H10**. Furthermore, we establish enumerability of all these problems and the decidability of first-order unification.

Overview In Section 2 we introduce the λ-calculus we use, define higher-order unification $\mathbf{U}$, and prove undecidability and enumerability for $\mathbf{U}$. In Section 3 we define nth-order unification $\mathbf{U}_n$ and show that $\mathbf{U}_n \preceq \mathbf{U}$. In Section 4 we present a simplified account of Huet's [1973] undecidability proof for $\mathbf{U}_3$ and in Section 5 a simplified account of Goldfarb's [1981] undecidability proof for $\mathbf{U}_2$, adapted to our setting. In Section 6 we explain how constants can be added or removed to the λ-calculus in the context of unification problems. In Section 7 we present a decidability proof for $\mathbf{U}_1$, before we give comments on the formalisation in Section 8, and conclude in Section 9.

Preliminaries We write $\mathcal{L}(X)$ for the type of lists over X. Given a list A, we write $x \in A$, if x is contained in the list A. We write $B \subseteq A$ if this is the case for every $x \in B$. We write $\overleftarrow{A}$ for the reverse list, $|A|$ for the length of A, a^n for the list which contains a n-times, and $[f x \mid x \in A]$ for the list obtained by applying f to every element of A. $+\!\!+$ denotes list concatenation and :: the cons operation on lists.

2 Higher-Order Unification

In this work, we consider a Curry-style[1] simply-typed λ-calculus (STLC) with unrestricted β-reduction. For a discrete type of constants C we define *terms*, *types*, and *typing contexts* by

$$\begin{aligned} s, t, u, v &::= x \mid c \mid \lambda x.s \mid s\ t && (x : \mathbb{N}, c : C) \\ A, B &::= \alpha \mid A \to B && (\alpha : \mathbb{N}) \\ \Gamma, \Delta, \Sigma &::= x_1 : A_1, \ldots, x_n : A_n \end{aligned}$$

[1]Meaning that abstractions are not annotated with a type.

We refer to both variables and constants as *atoms*, denoted by the letter a. We write vars s for the list of free variables which occur in s and dom Γ for the variables that occur in Γ. $\Gamma \cup \Delta$ denotes the context which contains all bindings of Γ and Δ. For a list of types $L = [A_1, \ldots, A_n]$, we write $L \to B$ for the function type $A_1 \to \cdots \to A_n \to B$.

Substitution In the formalisation, we use a De Bruijn [1972] representation of terms which we generate using Autosubst 2 [Stark et al. 2019]. For the sake of readability, we use named syntax on paper and follow Barendregt's variable convention [1984], assuming that the free variables occurring in terms are always distinct from bound variables.

Both on paper and in the formalisation we use capture-avoiding *parallel substitutions*, denoted by σ, τ. A parallel substitution is an infinite mapping from variables (i.e. natural numbers) to terms. We denote *applying the parallel substitution* σ to all free variables in the term s by $s[\sigma]$. The substitution $\sigma[x := s]$ denotes the extension of σ by binding x to the term s and $\sigma[\tau]$ the composition of σ and τ. We write s/x for the single-point substitution id$[x := s]$ where id $x := x$.

Operational Semantics We write $s > t$, if s can be *reduced* to t in a single β-reduction step and $s >^* t$, if s can be reduced to t in finitely many steps. We say s is *convertible* to t, written $s \equiv t$, if s and t are related in the equivalence closure of $>$. A term s is called *normal*, if it cannot be reduced any further. The small step semantics $>$ is confluent, normal forms are unique, and convertibility is compatible with substitutions and the term structure:

Lemma 2.1.

1. *If $s >^* t$ and $s >^* u$, then $t >^* v$ and $u >^* v$ for some v.*
2. *If $s >^* t$ and $s >^* u$ for normal t and u, then $t = u$.*
3. *$>$, $>^*$, $\equiv$ are compatible with the term structure and substitution.*

Simple Typing We equip our calculus with a Curry-style *simple type system*. We write $\Gamma \vdash s : A$, if s can be assigned the type A under the typing context Γ. The type of constants is given by a signature function Ω.

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash c : \Omega c} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\ t : B}$$

$$\frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B}$$

We lift *typing to substitutions* by $\Delta \vdash \sigma : \Gamma := \forall (x : A) \in \Gamma.\ \Delta \vdash \sigma x : A$. Substitutivity and preservation can be shown:

Lemma 2.2. *Let $\Gamma \vdash s : A$.*

1. *If $\Delta \vdash \sigma : \Gamma$, then $\Delta \vdash s[\sigma] : A$.*
2. *If $s > t$, then $\Gamma \vdash t : A$.*
3. *If $s >^* t$, then $\Gamma \vdash t : A$.*

Normalisation It is well known that in the simply-typed λ-calculus, all well-typed terms are weakly normalising:

Lemma 2.3. *If $\Gamma \vdash s : A$, then there exists some normal term t such that $s >^* t$.*

We introduce an *evaluation function* ξ which for a well-typed term $\Gamma \vdash s : A$ computes its normal form as follows: Following the standard technique by Takahashi [1989], in the proof of confluence (lemma 2.1), we define a *parallel reduction function* ς. We use ς to define a *step-indexed evaluation function* $\xi_n\ s$ where $\xi_n\ s$ returns the normal form of s if it can be reached by applying ς n-times. lemma 2.3 allows the computation of a sufficient step index n, which can be used in ξ to compute the normal form using ξ_n.

Lemma 2.4. *If $\Gamma \vdash s : A$, then $s >^* \xi s$ and ξs is normal.*

Reasoning about the normal forms of terms in the context of unification frequently requires an analysis of normal forms:

Lemma 2.5. *If s is normal, then $s = \lambda x_1 \cdots x_n.a\ t_1\ \cdots\ t_m$ for some n, atom a, and finite number of normal $t_1, \ldots, t_m$.*

2.1 Unification

Given two well-typed terms, *higher-order unification* is the problem of finding a well-typed substitution such that under the substitution both terms are convertible. Formally, an *instance of higher-order unification* is a dependent tuple $(\Gamma, s, t, A, H_s, H_t)$ where H_s and H_t are proofs of $\Gamma \vdash s : A$ and $\Gamma \vdash t : A$ respectively. On paper, we omit H_s, H_t and write $\Gamma \vdash s \overset{?}{=} t : A$ instead.

Definition 2.6.

$$\mathbf{U}(\Gamma \vdash s \overset{?}{=} t : A) := \exists \Delta \sigma.\ \Delta \vdash \sigma : \Gamma \text{ and } s[\sigma] \equiv t[\sigma]$$

Note that when proving unifiability of a higher-order unification instance, the substitution is allowed to contain open terms. As a consequence, the equation $\Gamma \vdash \lambda xy.f\ x \overset{?}{=} \lambda xy.f\ y : \alpha$ in the context $\Gamma = (f : \alpha \to \alpha)$ can be unified by the substitution $\lambda\_.z/f$ in the context $\Delta = (z : \alpha)$. If unification was not allowed to introduce fresh variables, this equation could be unified if and only if the type α is inhabited. If α is an empty type and σ was only allowed to insert closed terms, then the two terms would not be unifiable. In this case all closed terms of type $\alpha \to \alpha$ are convertible to identity functions. Hence, after substitution the left term is convertible to $\lambda xy.x$ and the right term to $\lambda xy.y$ which in turn are clearly not convertible. Thus, disallowing fresh variables would force unification to perform an emptiness check in cases such as the above equation which would make it trivially undecidable in dependent type theories.

Normalisation Using that every well-typed term is weakly normalising, we show that every equation $\Gamma \vdash s \overset{?}{=} t : A$ is unifiable if and only if it is unifiable by a substitution which only inserts normal terms.

Definition 2.7. $\mathbf{NU}(\Gamma \vdash s \overset{?}{=} t : A) := \exists \Delta\sigma. \Delta \vdash \sigma : \Gamma$ *and* $s[\sigma] \equiv t[\sigma]$ *and* $\forall x.$ normal (σx).

Lemma 2.8.

1. *If* $\Delta \vdash \sigma : \Gamma$*, then we can compute a substitution* τ *such that* $\Delta \vdash \tau : \Gamma$*,* $\sigma x >^* \tau x$ *for all* x*, and* τx *is normal for all* $x \in \operatorname{dom} \Gamma$.
2. $\mathbf{U}(\Gamma \vdash s \overset{?}{=} t : A)$ *iff* $\mathbf{NU}(\Gamma \vdash s \overset{?}{=} t : A)$.

Proof. 1. Pick $\tau x := \xi(\sigma x)$ if $(x : A) \in \Gamma$ and $\tau x := \sigma x$ otherwise, where ξ is the evaluation function defined before.
2. The backward direction is trivial. For the forward direction we use 1. to obtain $\Delta \vdash \tau' : \Gamma$. We modify τ' to a substitution τ inserting only normal terms: $\tau x := \tau' x$ if $x \in \operatorname{vars} s$ or $x \in \operatorname{vars} t$ and $\tau x := x$ otherwise. □

2.2 System Unification

Dowek [2001] defines unification as a problem over multiple equations which have to be unified with the same substitution. Following Snyder and Gallier [1989], we call multiple equations a *system of equations*. A single equation is a pair of terms, written $s \overset{?}{=} t$, and a system of equations is a list of single equations, denoted by the letter E. We lift typing to systems of equations as follows:

$$\frac{}{\Gamma \vdash [\,] : [\,]} \qquad \frac{\Gamma \vdash s : A \quad \Gamma \vdash t : A \quad \Gamma \vdash E : L}{\Gamma \vdash (s \overset{?}{=} t :: E) : (A :: L)}$$

A *system unification instance* is a dependent tuple (Γ, E, L, H) where H is a proof of $\Gamma \vdash E : L$. On paper, we also write $\Gamma \vdash E : L$ for the sake of readability.

Definition 2.9.

$\mathbf{SU}(\Gamma \vdash E : L) := \exists \Delta\sigma. \Delta \vdash \sigma : \Gamma$ *and* $\forall s \overset{?}{=} t \in E.\ s[\sigma] \equiv t[\sigma]$

We show that higher-order unification and system unification are interreducible. To prove $\mathbf{SU} \preceq \mathbf{U}$ we exploit that bound variables can neither be replaced nor reduced. For example, given the equations $s_1 \overset{?}{=} t_1$ and $s_2 \overset{?}{=} t_2$, we produce the equation $\lambda h.h\ s_1 s_2 \overset{?}{=} \lambda h.h\ t_1 t_2$, which is unifiable iff the original equations are unifiable by the same substitution.

Theorem 2.10. $\mathbf{U} \preceq \mathbf{SU}$ *and* $\mathbf{SU} \preceq \mathbf{U}$.

Analogously to lemma 2.8, we define a normal version of system unification and prove it equivalent:

Definition 2.11. $\mathbf{NSU}(\Gamma \vdash E : L) := \exists \Delta\sigma. \Delta \vdash \sigma : \Gamma$ *and* $\forall s \overset{?}{=} t \in E.\ s[\sigma] \equiv t[\sigma]$ *and* $\forall x.$normal (σx)

Lemma 2.12. $\mathbf{NSU}(\Gamma \vdash E : L)$ *iff* $\mathbf{SU}(\Gamma \vdash E : L)$

2.3 Undecidability of Higher-Order Unification

Goldfarb [1981] reduces Hilbert's tenth problem **H10** to second-order unification. As the proof is quite intricate, Dowek [2001] motivates the main idea by reducing **H10** to higher-order unification in general. While the latter is not as strong as Goldfarb's original result, its proof is considerably simpler. In the following, we formalise a modified version of Dowek's argument and use it as a stepping stone to establish the undecidability of second-order unification in Section 5. Note that by theorem 2.10 it suffices to reduce **H10** to system unification.

In its original formulation, Hilbert's tenth problem asks whether a given Diophantine equation, an equation over integers involving only addition, multiplication and constants, has a solution. Larchey-Wendling and Forster [2019] mechanise the synthetic undecidability of **H10**. In the following, we use their formulation of the problem based on *elementary Diophantine equations* (EDEs for short):

$$d ::= x \doteq 1 \mid x + y \doteq z \mid x \cdot y \doteq z \quad (x, y, z : \mathbb{N})$$

We denote systems of EDEs by the letter D and write vars D for the variables occuring in D. The satisfaction of an EDE d by the *variable assignment* θ is denoted by $\theta \vDash d$ and the satisfaction of a system D by $\theta \vDash D$:

$$\theta \vDash x \doteq 1 \text{ iff } \theta x = 1 \qquad \theta \vDash x + y \doteq z \text{ iff } \theta x + \theta y = \theta z$$

$$\theta \vDash x \cdot y \doteq z \text{ iff } \theta x \cdot \theta y = \theta z \qquad \theta \vDash D \text{ iff } \forall d \in D.\ \theta \vDash d$$

Hilbert's tenth problem **H10** is then formulated as the satisfiability of a system of EDEs:

Definition 2.13. $\mathbf{H10}(D) := \exists \theta.\ \theta \vDash D$

The reduction from **H10** is based on two core ideas. The first idea is to encode natural numbers, addition, and multiplication into the λ-calculus using Church numerals. The second idea is to restrict the *domain* of the unification equations to encodings of natural numbers, by introducing characteristic equations only satisfied by Church numerals.

Church numerals, defined by $[\![n]\!]_{\mathsf{cn}} := \lambda af.f^n\ a$, express iteration in the λ-calculus. Addition and multiplication can be defined by add $s\ t := \lambda af.s\ (t\ a\ f)\ f$ and mul $s\ t := \lambda af.s\ a\ (\lambda b.t\ b\ f)$.

Lemma 2.14. *Let* $[\![\mathbb{N}]\!]_{\mathsf{cn}} := \alpha \to (\alpha \to \alpha) \to \alpha$.

1. $\Gamma \vdash [\![n]\!]_{\mathsf{cn}} : [\![\mathbb{N}]\!]_{\mathsf{cn}}$
2. add $[\![m]\!]_{\mathsf{cn}}\ [\![n]\!]_{\mathsf{cn}} \equiv [\![m+n]\!]_{\mathsf{cn}}$
3. mul $[\![m]\!]_{\mathsf{cn}}\ [\![n]\!]_{\mathsf{cn}} \equiv [\![m \cdot n]\!]_{\mathsf{cn}}$.
4. *If* $[\![m]\!]_{\mathsf{cn}} \equiv [\![n]\!]_{\mathsf{cn}}$*, then* $m = n$.

In the reduction, we have to construct variable assignments from substitutions, which requires us to recover natural numbers from λ-terms:

Lemma 2.15. *For every term* s*, we can compute* n *such that* $s = [\![n]\!]_{\mathsf{cn}}$ *or prove that no such* n *exists.*

Note that Church numerals distribute with function application, meaning $f\ ([\![n]\!]_{\mathsf{cn}}\ s\ f) \equiv f^{n+1}\ s \equiv [\![n]\!]_{\mathsf{cn}}\ (f\ s)\ f$. We use this property to give a characteristic equation which can only be satisfied by Church numerals:

Lemma 2.16. *Let s be a normal term.*

$$\lambda af.f\ (s\ a\ f) \equiv \lambda af.s\ (f\ a)\ f \quad \textit{iff} \quad s = [\![n]\!]_{\mathsf{cn}} \textit{ for some } n$$

Reduction For the reduction, we need to transform a system of EDEs into a system of unification equations. We encode the individual equations by

$$\overline{x \doteq 1} := x \overset{?}{=} [\![1]\!]_{\mathsf{cn}} \qquad \overline{x + y \doteq z} := \mathsf{add}\ x\ y \overset{?}{=} z$$

$$\overline{x \cdot y \doteq z} := \mathsf{mul}\ x\ y \overset{?}{=} z$$

and restrict the domain of the unification equations through characteristic equations of the form $\mathsf{CN}\,x := \lambda af.x\ (f\ a)\ f \overset{?}{=} \lambda af.f\ (x\ a\ f)$.

Lemma 2.17. *Let $\Gamma_D := [(x : [\![\mathbb{N}]\!]_{\mathsf{cn}}) \mid x \in \mathsf{vars}\ D]$. The following rules are then admissible:*

$$\frac{x \in \mathsf{vars}\ D}{\Gamma_D \vdash \mathsf{CN}\ x : [\![\mathbb{N}]\!]_{\mathsf{cn}}} \qquad \frac{d \in D}{\Gamma_D \vdash \overline{d} : [\![\mathbb{N}]\!]_{\mathsf{cn}}}$$

In analogy to the notation $\theta \vDash d$, we define $\sigma \vDash s \overset{?}{=} t := s[\sigma] \equiv t[\sigma]$ to express that σ unifies s and t. In particular, we mean σ unifies the encoding of d, when we write $\sigma \vDash \overline{d}$. We show that the characteristic equation restricts the domain and that the encoding preserves arithmetic operations.

Lemma 2.18. *Let σx be normal.*

$$\sigma \vDash \mathsf{CN}\ x \textit{ iff } \exists n.\ \sigma x = [\![n]\!]_{\mathsf{cn}}$$

Lemma 2.19. *Let $\sigma x = [\![m]\!]_{\mathsf{cn}}$, $\sigma y = [\![n]\!]_{\mathsf{cn}}$, and $\sigma z = [\![p]\!]_{\mathsf{cn}}$.*

$$\sigma \vDash \overline{x \doteq 1} \textit{ iff } m = 1 \qquad \sigma \vDash \overline{x + y \doteq z} \textit{ iff } m + n = p$$

$$\sigma \vDash \overline{x \cdot y \doteq z} \textit{ iff } m \cdot n = p$$

Theorem 2.20. $\mathbf{H10} \preceq \mathbf{SU}$

Proof. For a system of EDEs D, we produce the system of equations:

$$f(D) = \Gamma_D \vdash E_D : [\![\mathbb{N}]\!]_{\mathsf{cn}}{}^{|E_D|}$$

where $E_D = [\mathsf{CN}\ x \mid x \in \mathsf{vars}\ D] +\!\!\!+ [\overline{d} \mid d \in D]$.

It remains to prove that $\mathbf{H10}(D)$ iff $\mathbf{SU}(f(D))$.

1. Let θ be a solution for D. Pick $\Delta := \emptyset$ and $\sigma x := [\![\theta x]\!]_{\mathsf{cn}}$. Typing follows with lemma 2.14 and the unifiability with lemma 2.19.
2. By lemma 2.12 let $\Delta \vdash \sigma : \Gamma_D$ be a substitution inserting only normal terms and unifying the equations E_D. Using lemma 2.16 we define $\theta x := n$ if $\sigma x = [\![n]\!]_{\mathsf{cn}}$ and $\theta x := 0$ otherwise. We establish satisfiability using Lemmas 2.18 and 2.19. □

Enumerability We show that the problem $\mathbf{U}$ is enumerable. In general, following Forster et al. [2019a] we say that a problem P on X is enumerable if there is a function $L : \mathbb{N} \to \mathcal{L}X$ such that Px holds if and only if $\exists n.\ x \in Ln$. A type X is enumerable if there is a function $L : \mathbb{N} \to \mathcal{L}X$ such that $\forall x : X. \exists n.\ x \in Ln$.

Our strategy to enumerate $\mathbf{U}$ is to first enumerate all possible dependent tuples $\Gamma \vdash s \overset{?}{=} t : A$ along with all possible contexts Δ and substitutions σ such that $\Delta \vdash \sigma : \Gamma$ and $\forall x \notin \mathsf{dom}\ \Gamma.\ \sigma x = x$. We then obtain an enumerator for $\mathbf{U}$ by projecting out the pairs $\Gamma \vdash s \overset{?}{=} t : A$ such that $\xi(s[\sigma]) = \xi(t[\sigma])$.

Lemma 2.21. *The following are enumerable:*

1. *The types of terms, types and typing contexts.*
2. *Proofs of $\Gamma \vdash s : A$.*
3. *Dependent tuples $\Gamma \vdash s \overset{?}{=} t : A$.*
4. *Dependent tuples (Δ, σ, Γ) such that $\Delta \vdash \sigma : \Gamma$ and $\forall x \notin \mathsf{dom}\ \Gamma.\ \sigma x = x$.*

Proof. By exhibiting a function from $\mathbb{N}$ to lists of the desired type, using standard techniques from Forster et al. [2019a]. □

Theorem 2.22. $\mathbf{U}$ *is enumerable.*

Proof. By lemma 2.21 we can easily obtain an enumerator L for tuples $(\Gamma, s, t, A, H_1, H_2, \Delta, \sigma)$ such that H_1 is a proof of $\Gamma \vdash s : A$, H_2 is a proof of $\Gamma \vdash t : A$, $\Delta \vdash \sigma : \Gamma$, and $\xi(s[\sigma]) = \xi(t[\sigma])$.

We obtain an enumerator for $\mathbf{U}$ from L by discarding Δ and σ from the results of L. □

Corollary 2.23. $\mathbf{SU}$ *is enumerable.*

Proof. Immediate with Theorems 2.10 and 2.22. □

3 Nth-Order Unification

In Section 2, we established the undecidability of higher-order unification in general. In the following, we restrict the order of the variables which may occur. For example, *second*-order unification is only concerned with terms containing free variables of at most order two. *Third*-order unification on the other hand restricts the order of free variables to three.

Order Typing In general, we define the order of a type A by $\mathsf{ord}\ \alpha := 1$ and $\mathsf{ord}\ (A \to B) := \max(\mathsf{ord}\ A + 1, \mathsf{ord}\ B)$. Following Snyder and Gallier [1989], we say a term is of order n, if the types of all occurring variables are at most of order n and the types of all occurring constants are at most of order $n + 1$. We capture the fragment of all terms which are of order n by *order typing* $\Gamma \vdash_n s : A$.

$$\frac{(x : A) \in \Gamma \quad \mathsf{ord}\ A \le n}{\Gamma \vdash_n x : A} \qquad \frac{\mathsf{ord}\ (\Omega c) \le n + 1}{\Gamma \vdash_n c : \Omega c}$$

$$\frac{\Gamma \vdash_n s : A \to B \quad \Gamma \vdash_n t : A}{\Gamma \vdash_n s\ t : B} \qquad \frac{\Gamma, x : A \vdash_n s : B}{\Gamma \vdash_n \lambda x.s : A \to B}$$

Allowing constants of order $n + 1$ coincides with the usual first-order interpretation where for constants $\mathsf{g} : \alpha \to \alpha \to \alpha$ and $\mathsf{a} : \alpha$, the term $x : \alpha \vdash_1 \mathsf{g}\ \mathsf{a}\ x : \alpha$ is a first-order term.

Order typing is lifted to substitutions by $\Delta \vdash_n \sigma : \Gamma := \forall (x : A) \in \Gamma.\ \Delta \vdash_n \sigma x : A$.

For $\Gamma \vdash_n s : A$, neither the type A nor the order n is necessarily unique. Nonuniqueness is inherited from the Curry-style type system (and monotonicity of the rules). For example, the term $\lambda x.x$ can be typed as $\vdash_1 \lambda x.x : \alpha \to \alpha$ and $\vdash_2 \lambda x.x : (\alpha \to \alpha) \to (\alpha \to \alpha)$. We remark that w.r.t. unification, only the order of *free* variables is of interest, and their order is uniquely determined by the typing context.

Order typing is monotone with respect to the order and can be seen as a more fine-grained version of the standard typing judgement. Similar to standard typing, order typing is also preserved under substitution and reduction.

Lemma 3.1.

1. *If $\Gamma \vdash_n s : A$, then $\Gamma \vdash s : A$.*
2. *If $\Gamma \vdash s : A$, then $\Gamma \vdash_n s : A$ for some n.*
3. *If $\Gamma \vdash_n s : A$ and $n \le m$, then $\Gamma \vdash_m s : A$.*

All properties also hold for substitutions.

Lemma 3.2. *Let $\Gamma \vdash_n s : A$.*

1. *If $\Delta \vdash_n \sigma : \Gamma$, then $\Delta \vdash_n s[\sigma] : A$.*
2. *If $s \succ s'$, then $\Gamma \vdash_n s' : A$.*
3. *If $s \succ^* s'$, then $\Gamma \vdash_n s' : A$.*

Nth-Order Unification Compared to higher-order unification, nth-order unification restricts the order of all occurring terms to n. An instance of nth-order unification is a dependent tuple $(\Gamma, s, t, A, H_s, H_t)$ where H_s, H_t are proofs of $\Gamma \vdash_n s : A$ and $\Gamma \vdash_n t : A$ respectively. On paper, we omit H_s, H_t and write $\Gamma \vdash_n s \overset{?}{=} t : A$ instead.

Definition 3.3.

$$\mathbf{U}_n(\Gamma \vdash_n s \overset{?}{=} t : A) := \exists \Delta\sigma.\ \Delta \vdash_n \sigma : \Gamma \text{ and } s[\sigma] \equiv t[\sigma]$$

For $n = 0$, the equation $x : \beta \vdash_0 \mathsf{a} \overset{?}{=} \mathsf{a} : \alpha$ is not unifiable if a is a constant of type α and β is empty. Any substitution would have to replace x by some term of type β and order 1. If β is empty, no term without variables exists of type β. To exclude such artificial instances, we only consider the case where $n \ge 1$.

Similar to higher-order unification, we define a system version of nth-order unification, called $\mathbf{SU}_n$ and versions of both $\mathbf{U}_n$ and $\mathbf{SU}_n$ where the substitution is required to be normal on all arguments called $\mathbf{NU}_n$ and $\mathbf{NSU}_n$. We do not spell out all the details of these problems on paper.

An instance of $\mathbf{SU}_n$ is denoted by $\Gamma \vdash_n E : L$. The reduction $\mathbf{U} \preceq \mathbf{SU}$ carries over to $\mathbf{U}_n \preceq \mathbf{SU}_n$ without changes. The reduction $\mathbf{SU} \preceq \mathbf{U}$ does not. Consider the second-order system of equations $\lambda xy.x \overset{?}{=} f$ and $\lambda xy.y \overset{?}{=} g$ which can be typed as $\alpha \to \alpha \to \alpha$ in a context where f, g are of this type. The equation $\lambda h.h\ (\lambda xy.x)\ (\lambda xy.y) \overset{?}{=} \lambda h.h\ f\ g$ cannot be typed in the second-order fragment as applying the left hand sides to h makes h a third-order variable. The reduction $\mathbf{SU}_n \preceq \mathbf{U}_n$ is not needed in the following as we can prove all unification problems $\mathbf{U}_{2+k}$ undecidable without it. In the one case where it would be useful, connecting $\mathbf{H10}$ with $\mathbf{U}_2$ using theorem 5.23, we can essentially inline the reduction but have to take care of some binders. On paper the reduction exists: If $n = 1$ it is trivial because first-order unification is decidable (theorem 7.6). If $n > 1$, then in the context of the library we have a reduction $\mathbf{Halt} \preceq \mathbf{H10} \preceq \mathbf{U}_2 \preceq \mathbf{U}_n$ using corollary 5.25. Further, $\mathbf{SU}_n$ is enumerable thus there is a reduction $\mathbf{SU}_n \preceq \mathbf{Halt}$ which we do not formalise in this work. The tools for formalising said reduction are developed in [Forster and Kunze 2019; Forster et al. 2020a].

The equivalence of $\mathbf{U}_n$ and $\mathbf{NU}_n$, as well as the equivalence of $\mathbf{SU}_n$ and $\mathbf{NSU}_n$ is preserved. The proofs are analogous to their higher-order counterparts.

3.1 Conservativity

Huet [1973] proves the undecidability of higher-order unification and remarks that since his reduction only uses terms of at most order three, it can be seen as an undecidability proof for third-order unification. Formalising this argument would require the duplication of auxiliary lemmas to the setting without explicit order. Instead, in this work, we show $\mathbf{U}_n \preceq \mathbf{U}$, the *conservativity of unification.*

The key insight for the reduction is that for two terms s and t of order n one can always transform a unifying higher-order substitution into a unifying substitution of order n. This is accomplished by replacing free variables and constants which are not needed to unify s and t with terms of the same type and order one.

We denote the *arity* of A by $\mathsf{ar}(A)$ and the *target* of A, the rightmost base type of A, by $\mathsf{tar}(A)$. For example, the arity of the type $\alpha \to (\alpha \to \alpha) \to \beta$ is 2 and the target is β.

Lemma 3.4. *For the term $\mathsf{inhab}_x\ A := \lambda x_1 \dots x_{\mathsf{ar}(A)}.x$ we have $\Gamma \vdash_1 \mathsf{inhab}_x\ A : A$ provided that $(x : \mathsf{tar}(A)) \in \Gamma$.*

Lemma 3.5. *Let $\Delta \vdash \sigma : \Gamma$ with $s[\sigma] \equiv t[\sigma]$.*

1. *There exist $\Sigma \vdash \tau : \Gamma$ with $s[\tau] \equiv t[\tau]$ and $\mathsf{ord}\ \Sigma \le 1$.*
2. *There exist $\Delta \cup \Sigma \vdash \tau : \Gamma$ with $s[\tau] \equiv t[\tau]$, $\mathsf{ord}\ \Sigma \le 1$ and all constants in τ are constants from s or t.*

Proof. For 1. let $\tau' x := \mathsf{inhab}_{y_x}\ A$ if $(x : A) \in \Delta$ and $\tau' x := x$ otherwise where each y_x is a fresh variable. We pick $\Sigma := [(y_x : \mathsf{tar}(A)) \mid (x : A) \in \Delta]$ and $\tau := \sigma[\tau']$.

For 2. let C be a list of all constants in s or t and x_c a fresh variable for every $c \notin C$. We define τ by replacing every constant $c \notin C$ in σ by $\mathsf{inhab}_{x_c}\ (\Omega c)$ and pick $\Sigma := [(x_c : \mathsf{tar}(\Omega c)) \mid c \notin C, c \in \mathsf{consts}\ (\sigma x), x \in \mathsf{dom}\ \Gamma]$ where the list $\mathsf{consts}\ (\sigma x)$ contains the constants in σx. □

We recover order typing from ordinary typing by:

Lemma 3.6. *If $\mathsf{ord}\ \Gamma \le n$, $\mathsf{ord}\ A \le n+1$, and $\mathsf{ord}\ (\Omega c) \le n+1$ for all constants c in s, then:*

$$\frac{\Gamma \vdash s : A \quad \text{normal } s}{\Gamma \vdash_n s : A}$$

By definition $\Gamma \vdash_n s \overset{?}{=} t : A$ ensures that for all variables $(x : B) \in \Gamma$ which occur in s or t the type B is at most of order n. However, there may be variables of arbitrary order in Γ which do not occur in s or t. The following lemma shows that unifiability is not affected by this technicality.

Lemma 3.7. *If $s[\sigma] \equiv t[\sigma]$ and for all $(x : B) \in \Gamma$ where x is a free variable of s or t, $\Delta \vdash_n \sigma x : B$, then $s[\tau] \equiv t[\tau]$ for some Σ and $\Sigma \vdash_n \tau : \Gamma$.*

Lemma 3.8. *Let $\Gamma \vdash_n s \overset{?}{=} t : A$. For every Δ and substitution $\Delta \vdash \sigma : \Gamma$ with $s[\sigma] \equiv t[\sigma]$, there exists Σ and a substitution $\Sigma \vdash_n \tau : \Gamma$ with $s[\tau] \equiv t[\tau]$.*

Proof. Let $\Delta \vdash \sigma : \Gamma$ be a unifying substitution. Let Δ_1 and τ_1 be obtained from lemma 3.5 1. for σ and Δ_2 and τ_2 from lemma 3.5 2. for τ_1.

That is, we have $\Delta_1 \cup \Delta_2 \vdash \tau_2 : \Gamma$ with ord $(\Delta_1 \cup \Delta_2) \le 1$ and all constants in τ_2 are constants from s or t. The claim follows by normalising τ_2 using lemma 2.8 and Lemmas 3.6 and 3.7. □

Corollary 3.9. *Let $n \le m$.*

1. $\mathbf{U}_n \le \mathbf{U}_m$ *and* $\mathbf{U}_n \le \mathbf{U}$.
2. $\mathbf{SU}_n \le \mathbf{SU}_m$ *and* $\mathbf{SU}_n \le \mathbf{SU}$.

Proof. 1. follows from Lemmas 3.1 and 3.8. The proof of 2. is analogous. □

Conservativity allows us to lift enumerability from higher-order unification to the nth-order fragments.

Corollary 3.10. $\mathbf{U}_n$ *and* $\mathbf{SU}_n$ *are enumerable.*

4 Third-Order Unification

Huet [1973] proves the undecidability of higher-order unification by reduction from Post's correspondence problem [1946] **PCP**. We adapt Huet's proof to our setting and simplify it by reducing the *modified* Post correspondence problem [Hopcroft et al. 2006] to third-order unification. The synthetic undecidability of **MPCP** was established by Forster, Heiter, and Smolka [2018].

Definition 4.1. *We define the modified Post correspondence problem* **MPCP** *over a dedicated card l_0/r_0 and a list of cards $l_1/r_1, \dots, l_n/r_n$ where each l_i, r_i is a binary string.*

$$\mathbf{MPCP}(l_0/r_0, l_1/r_1, \dots, l_n/r_n) := \exists k.\exists i_1 \dots i_k. l_0 l_{i_1} \cdots l_{i_k} = r_0 r_{i_1} \cdots r_{i_k}$$

We denote Booleans by the letter b and binary strings by l, r and w. Binary strings are encoded as

$$\overline{b_1 \cdots b_n} := \lambda x. \overline{b_1}\ (\cdots (\overline{b_n}\ x) \cdots) \text{ with } \overline{1} := u_1, \overline{0} := u_0$$

for fixed variables u_1, u_0. For encoded strings, concatenation can be implemented simply as composition and encoded strings are of type $\alpha \to \alpha$ provided u_1, u_0 are of this type:

Lemma 4.2. *Let $\Gamma \vdash_3 u_1 : \alpha \to \alpha$ and $\Gamma \vdash_3 u_0 : \alpha \to \alpha$. We then have $\Gamma \vdash_3 \overline{b} : \alpha \to \alpha$ and $\Gamma \vdash_3 \overline{w} : \alpha \to \alpha$.*

Proof. Using the λ-typing rule and an induction on w. □

Lemma 4.3. $\overline{w_1 \cdots w_n}\ s \equiv \overline{w_1}\ (\cdots (\overline{w_n}\ s) \cdots)$

Proof. By induction on n. The inductive step follows with $\overline{w}\ (\overline{w'}\ s) \equiv \overline{ww'}\ s$, which is shown by induction on w. □

The encoding is invertible, provided its arguments do not encode strings.

Lemma 4.4. *If $\overline{w}\ s \equiv \overline{w'}\ t$, then $w = w'$ and $s \equiv t$ provided neither s nor t is convertible to $u_0\ s'$ or $u_1\ s'$ for any s'.*

Proof. By induction on w. □

We now define and verify the many-one reduction from **MPCP** to $\mathbf{U}_3$. For a dedicated card l_0/r_0 and a list of cards $l_1/r_1, \dots, l_n/r_n$, we construct $\hat{\Gamma} \vdash_3 \hat{s} \overset{?}{=} \hat{t} : \hat{A}$ where

$$\hat{\Gamma} := (x_f : (\alpha \to \alpha)^{n+1} \to \alpha) \qquad \hat{A} := (\alpha \to \alpha)^2 \to \alpha$$
$$\hat{s} := \lambda u_0 u_1. \overline{l_0}\ (x_f\ \overline{l_0} \cdots \overline{l_n}) \qquad \hat{t} := \lambda u_0 u_1. \overline{r_0}\ (x_f\ \overline{r_0} \cdots \overline{r_n})$$

We introduce a decomposition lemma to obtain sequences of indices from terms:

Lemma 4.5. *For any term s and variables $x_1, \dots, x_n$, we have $s = x_{i_1}\ (\cdots (x_{i_k}\ s') \cdots)$ for some $i_1, \dots, i_k$ and term s' such that s' is not of the shape $x_i\ s''$ for any $1 \le i \le n$ and s''.*

For an empty base type α, Reynolds's notion of parametricity [1983] suggests that a function f of type $(\alpha \to \alpha)^n \to \alpha \to \alpha$ fulfills $f\ g_1 \cdots g_n\ t \equiv g_{i_1}\ (\cdots (g_{i_k}\ t))$ for a fixed sequence $i_1, \dots, i_k$ depending solely on f. Since we are using possibly nonempty base types and open substitutions, we cannot use such a general parametricity statement. Instead, we remove the argument t and focus on terms of type $\Gamma \vdash_3 s : (\alpha \to \alpha)^n \to \alpha$.

Lemma 4.6. *For all $\Gamma \vdash_3 s : (\alpha \to \alpha)^n \to \alpha$ with u_1, u_0 not free in s, there is a sequence $i_1, \dots, i_k$ such that for all $w_1 \dots w_n$*

$$s\ \overline{w_1} \cdots \overline{w_n} \equiv \overline{w_{i_1}}\ (\cdots (\overline{w_{i_k}}\ t) \cdots)$$

for some t which is not convertible to $u_0\ s'$ or $u_1\ s'$ for all s'.

Proof. We analyse the normal form v of s. By lemma 2.5, v must be of the shape $\lambda x_1 \cdots x_m.v'$ where $v' = a\ s_1 \cdots s_p$ for atom a. We proceed by case analysis on m.

1. If $m > n$, we have a contradiction since in this case v cannot be of type $(\alpha \to \alpha)^n \to \alpha$.
2. If $n = m$, then by lemma 4.5 let k be the largest number such that $v' = x_{i_1}\ (\cdots (x_{i_k}\ v'') \cdots)$ for some sequence $i_1, \dots, i_k$ and term v''. Thus, we obtain the equivalence

$$v\ \overline{w_1} \cdots \overline{w_n} \equiv \overline{w_{i_1}}\ (\cdots (\overline{w_{i_k}}\ t) \cdots) \text{ for a certain term } t.$$

Since v and therefore v'' contains neither u_1 nor u_0 and v'' is not an application of any of the x_i's, we can conclude that the resulting term t is not convertible to $u_0\ s'$ or $u_1\ s'$ for any s'.

3. If $m < n$, then we proceed with a case analysis on a.
   a. If $a = c$ for some constant c, then $v\ \overline{w_1} \cdots \overline{w_n}$ reduces to a term t with c as its applicative head. With $k := 0$ we have $v\ \overline{w_1} \cdots \overline{w_n} \equiv t$ and thus the claim follows.
   b. If $a = x$ for some x, then the type of x, in this case $(\alpha \to \alpha)^{n-m} \to \alpha$, enforces that $x \neq x_i$ for all i since every x_i is of type $\alpha \to \alpha$. As a consequence, x is a free variable and behaves similarly to a constant. Analogous to 3a we have $v\ \overline{w_1} \cdots \overline{w_n} \equiv t$ for a sufficient t. □

Theorem 4.7. **MPCP** $\preceq$ **U**$_3$

Proof. We prove **MPCP**$(l_0/r_0, l_1/r_1, \dots, l_n/r_n)$ if and only if **U**$_3(\hat{\Gamma} \vdash_3 \hat{s} \overset{?}{=} \hat{t} : \hat{A})$. The forward direction is straightforward. Given $i_1, \dots, i_k$ such that $l_0 l_{i_1} \cdots l_{i_k} = r_0 r_{i_1} \cdots r_{i_k}$, we pick σ as $\sigma x_f := \lambda x_0 x_1 \cdots x_n . x_{i_1}\ (\cdots (x_{i_k}\ z) \cdots)$ and $\sigma x := x$ otherwise, in context $\Delta := (z : \alpha)$. Using lemma 4.3, we have:

$$\begin{aligned} s[\sigma] &\equiv \lambda u_0 u_1 . \overline{l_0}\ (\overline{l_{i_1}}\ (\cdots (\overline{l_{i_k}}\ z) \cdots)) \\ &\equiv \lambda u_0 u_1 . \overline{l_0 l_{i_1} \cdots l_{i_k}}\ z \\ &= \lambda u_0 u_1 . \overline{r_0 r_{i_1} \cdots r_{i_k}}\ z \\ &\equiv \lambda u_0 u_1 . \overline{r_0}\ (\overline{r_{i_1}}\ (\cdots (\overline{r_{i_k}}\ z) \cdots)) \\ &\equiv t[\sigma]. \end{aligned}$$

For the other direction, assume $\Delta \vdash_3 \sigma : \Gamma$ and $s[\sigma] \equiv t[\sigma]$. This entails $\overline{l_0}\ (\sigma x_f\ \overline{l_0} \cdots \overline{l_n}) \equiv \overline{r_0}\ (\sigma x_f\ \overline{r_0} \cdots \overline{r_n})$. Thus, by Lemmas 4.3 and 4.6 we have $\overline{l_0 l_{i_1} \cdots l_{i_k}}\ t' \equiv \overline{r_0 r_{i_1} \cdots r_{i_k}}\ t''$ for some t', t'' such that neither t' nor t'' is convertible to $u_0\ s'$ or $u_1\ s'$ for any s'. The claim then follows with lemma 4.4. □

5 Second-Order Unification

Goldfarb [1981] proves the undecidability of second-order unification for the applicative fragment of the λ-calculus with hereditary substitutions. We adapt his proof to our Curry-style simply-typed λ-calculus, simplify his construction, and give an intuitive explanation. We use a single equation to encode multiplication whereas Goldfarb's reduction relies on two.

The structure of the proof is similar to the undecidability proof of higher-order unification in Section 2. Instead of Church numerals, Goldfarb uses an encoding of natural numbers based on constants $\mathsf{g} : \alpha \to \alpha \to \alpha$ and $\mathsf{a} : \alpha$, which we refer to as *Goldfarb numerals*. The Goldfarb numeral $[\![n]\!]_{\mathrm{gn}} := \lambda a . \mathsf{S}^n\ a$ can be derived from the Church numeral $[\![n]\!]_{\mathrm{cn}} = \lambda a f . f^n\ a$ by fixing $\mathsf{S} := \mathsf{g}\ \mathsf{a}$ for f.

Lemma 5.1. $\Gamma \vdash_2 [\![n]\!]_{\mathrm{gn}} : \alpha \to \alpha$

We call the reduced form $\mathsf{S}^n\ t$ of the application of a Goldfarb literal $[\![n]\!]_{\mathrm{gn}}$ to a term t an *applied Goldfarb numeral.* In the following, applied Goldfarb numerals allow us to reason about syntactic equality whereas Goldfarb numerals would force us to reason about convertibility.

Lemma 5.2.

$$\mathsf{S}^0\ t = t \qquad \mathsf{S}^{n+1}\ t = \mathsf{S}\ (\mathsf{S}^n\ t) \qquad \mathsf{S}^{m+n}\ t = \mathsf{S}^m\ (\mathsf{S}^n\ t)$$

Lemma 5.3. *Let s, t be atoms. If $\mathsf{S}^m\ s = \mathsf{S}^n\ t$, then $m = n$ and $s = t$.*

Lemma 5.4. *Let s be normal. We can compute some n such that $s\ \mathsf{a} \equiv \mathsf{S}^n\ \mathsf{a}$, or prove that no such n exists.*

Proof. With the reduction function ς, we obtain the normal form v of $s\ \mathsf{a}$ and check whether $v = \mathsf{S}^n\ \mathsf{a}$ for some n. □

Multiplication We now encode constants, addition and multiplication equations of **H10** using Goldfarb numerals. The equations constants and addition are straightforward adaptions of the equations for Church numerals. Multiplication for Church numerals, however, is implemented by instantiating the second argument f of a numeral with a function. Since Goldfarb numerals fix this argument to S, this construction cannot be reused. Instead, we use a relational encoding of multiplication in the λ-calculus in terms of conversion equations.

In the following, we first give an intuitive, more abstract explanation of the proof idea before we give the details for the full construction. The idea behind the relational encoding can be understood by looking at how the equation $m \cdot n = p$ can be encoded as an equation $\mathcal{M}_{m,n,p}\ M$ over finite sequences $M : \mathcal{L}(\mathbb{N} \times \mathbb{N})$. The same ideas then apply to encode the equation $m \cdot n = p$ as a conversion equation $\mathcal{G}_{m,n,p}\ G$ over functions G in the λ-calculus.

To compute the product of m and n by hand, one can start with the pair $(0, 0)$ and iteratively apply $\mathsf{step}_n(a, k) := (a + n, k + 1)$ until the counter, i.e. the second component reaches m. This process generates the sequence

$$(0, 0), (n, 1), (2 \cdot n, 2), \dots, (m \cdot n, m).$$

The pair (p, m) occurs in this sequence if and only if $m \cdot n = p$.

We represent finite sequences as lists and define $T_m := [t_0, \dots, t_{m-1}]$ where $t_k := (k \cdot n, k)$. Note that the sequence above is exactly T_{m+1}. The iteration function step_n is lifted to lists by $\mathsf{step}_n\ M := [\mathsf{step}_n\ (a, k) \mid (a, k) \in M]$. In the following, we show that $m \cdot n = p$ *and* $M = T_m$ *iff* $\mathcal{M}_{m,n,p}\ M$ for the equation

$$\mathcal{M}_{m,n,p}\ M := M +\!\!+ [(p, m)] = t_0 :: \mathsf{step}_n\ M.$$

The forward direction of this equivalence is straightforward. By construction, application of step_n yields the next pair of the sequence and if $p = m \cdot n$, then T_m is always a solution:

Lemma 5.5. $\mathsf{step}_n\ t_k = t_{k+1}$

Lemma 5.6. $\mathcal{M}_{m,n,m \cdot n}\ T_m$

Proof. Since $(m \cdot n, m) = t_m$, we have $T_m +\!\!+ [(m \cdot n, m)] = [t_0, t_1, \dots, t_m] = t_0 :: [\mathsf{step}_n\ t_0, \dots, \mathsf{step}_n\ t_{m-1}] = t_0 :: \mathsf{step}_n\ T_m$, where the second equality is by lemma 5.5. □

We turn to the backwards direction of the equivalence. Note that the equation $\mathcal{M}_{m,n,p}\ M$ can be seen a description of M in terms of its head and its tail. We show that this suffices to enforce an iterative structure and to characterise the multiplication sequence.

Lemma 5.7. *If $\mathcal{M}_{m,n,p}\ M$, then $M = T_l$ for some l.*

Proof. We generalise the statement: *If $M +\!\!+ [x] = (a, b) :: \mathsf{step}_n\ M$, then $M +\!\!+ [x] = [(a, b), (a + n, b + 1), \ldots, (a + |M| \cdot n, b + |M|)]$* by induction on M with lemma 5.5. The claim follows with $x = (p, m)$ and $a = b = 0$. □

Lemma 5.8. *If $\mathcal{M}_{m,n,p}\ T_l$, then $m = l$ and $m \cdot n = p$.*

Proof. By lemma 5.6 we have $\mathcal{M}_{l,n,l\cdot n}\ T_l$, i.e. we know that $T_l +\!\!+ [(l \cdot n, l)] = t_0 :: \mathsf{step}_n\ T_l$. By the assumption $\mathcal{M}_{m,n,p}\ T_l$, we know $t_0 :: \mathsf{step}_n\ T_l = T_l +\!\!+ [(p, m)]$ and thus $T_l +\!\!+ [(l \cdot n, l)] = T_l +\!\!+ [(p, m)]$. By injectivity we have $m = l$ and $m \cdot n = p$. □

Corollary 5.9. *$m \cdot n = p$ and $M = T_m$ iff $\mathcal{M}_{m,n,p}\ M$.*

Proof. Follows from Lemmas 5.6 to 5.8. □

In the remainder of this section, we mirror the above proof in the λ-calculus. To this end, we define the counterpart to $\mathcal{M}_{m,n,p}\ M$, a relation $\mathcal{G}_{m,n,p}\ G$ over second-order terms G. Because our calculus does not have means to computationally distinguish $\mathsf{g}\ s\ t$ from e.g. a, we cannot implement the step_n function over sequences or the concatenation of lists as second-order λ-terms. Instead, we require G to be a function and realise step_n and concatenation by choosing appropriate arguments for G. Note that Goldfarb numerals are of second order, so functions taking them as arguments will be of third order. To work with second-order terms only, we use two fresh variables a, b and applied Goldfarb numerals such as $\mathsf{S}^n\ a$ instead of Goldfarb numerals. For such applied numerals, addition cannot be realised by application. Instead, we rely on substitution: Addition of m can be realised by the substitution $\mathsf{S}^m\ a/a$, since $(\mathsf{S}^n\ a)[(\mathsf{S}^m\ a)/a] = \mathsf{S}^{n+m}\ a$.

We encode lists as $[] := \mathsf{a}$ and $s :: t := \mathsf{g}\ s\ t$ and pairs as $\widehat{(m, n)} := \langle \mathsf{S}^m\ a, \mathsf{S}^n\ b\rangle$ where $\langle s, t\rangle := \mathsf{g}\ s\ t$. By construction $\hat{t}_k = \langle \mathsf{S}^{k\cdot n}\ a, \mathsf{S}^k\ b\rangle$. In the following, we use list notation both for lists of pairs and encoded lists. We realise $\mathcal{M}_{m,n,p}\ M$ in the λ-calculus as

$$\mathcal{G}_{m,n,p}\ G := \quad G\ [\widehat{(p, m)}]\ a\ b \equiv \hat{t}_0 :: G\ []\ (\mathsf{S}^n\ a)\ (\mathsf{S}\ b)$$

where G may not contain a or b as free variables. In the formalisation, this is achieved by renaming both variables in G.

In the following, we show that *$m \cdot n = p$ and $G = \hat{T}_m$ iff $\mathcal{G}_{m,n,p}\ G$* for a λ-term $\hat{T}_m$ encoding T_m. Note that the straightforward encoding of T_m, i.e. $[\hat{t}_0, \ldots, \hat{t}_{m-1}]$ is not a function and therefore cannot satisfy the equation. Instead, we define $\hat{T}_m := \lambda rab.\hat{t}_0 :: \ldots :: \hat{t}_{m-1} :: r$.

We again start with the forwards direction:

Lemma 5.10. *Let $\sigma^{\mathsf{id}}_{p,q}$ and σ^{st} be defined by*

	a	b	r
$\sigma^{\mathsf{id}}_{p,q}$	a	b	$[\widehat{(p, q)}]$
σ^{st}	$\mathsf{S}^n\ a$	$\mathsf{S}\ b$	$[]$.

and id othw. Then $\hat{T}_m\ [\widehat{(p, q)}]\ a\ b \equiv (\hat{t}_0 :: \ldots :: \hat{t}_{m-1} :: r)[\sigma^{\mathsf{id}}_{p,q}]$ and $\hat{T}_m\ []\ (\mathsf{S}^n\ a)\ (\mathsf{S}\ b) \equiv (\hat{t}_0 :: \ldots :: \hat{t}_{m-1} :: r)[\sigma^{\mathsf{st}}]$.

Note that $\sigma^{\mathsf{id}}_{p,q}$ leaves elements of the multiplication sequence unchanged, whereas σ^{st} realises the step_n function. We show the counterpart of lemma 5.5 which suffices to recover lemma 5.6.

Lemma 5.11. *$\hat{t}_k[\sigma^{\mathsf{id}}_{p,q}] = \hat{t}_k$ and $\hat{t}_k[\sigma^{\mathsf{st}}] = \hat{t}_{k+1}$*

Corollary 5.12. *$\hat{T}_m\ [\widehat{(p, q)}]\ a\ b \equiv [\hat{t}_0, \ldots, \hat{t}_{m-1}, \widehat{(p, q)}]$ and $\hat{T}_m\ []\ (\mathsf{S}^n\ a)\ (\mathsf{S}\ b) \equiv [\hat{t}_1, \ldots, \hat{t}_m]$*

Proof. Immediate from Lemmas 5.10 and 5.11. □

Corollary 5.13. *$\mathcal{G}_{m,n,m\cdot n}\ \hat{T}_m$*

Proof. By corollary 5.12 it follows that $\hat{T}_m\ [\widehat{(m \cdot n, m)}]\ a\ b \equiv [\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_m] \equiv \hat{t}_0 :: [\hat{t}_1, \ldots, \hat{t}_m] \equiv \hat{t}_0 :: \hat{T}_m\ []\ (\mathsf{S}^n\ a)\ (\mathsf{S}\ b)$. □

The backwards direction of the equivalence, i.e. the counterpart of lemma 5.7 is more complicated in the λ-calculus. The combination of the following three lemmas suffices to recover the result. First, we show that terms satisfying the equation have to be functions of the shape $\lambda rab.G'$ where G' satisfies a substitution equation. Then, we show that the substitution $\sigma^{\mathsf{id}}_{p,q}$ is invertible for the elements of the sequence $\hat{t}_k$. Lastly, we show how from the substitution equation, one can recover the multiplication sequence.

Lemma 5.14. *Let G be normal. If $\mathcal{G}_{m,n,p}\ G$, then $G = \lambda rab.G'$ for some G' with $G'[\sigma^{\mathsf{id}}_{p,m}] = \hat{t}_0 :: G'[\sigma^{\mathsf{st}}]$.*

Proof. The claim $G = \lambda rab.G'$ follows by normal form analysis on G. By conversion, we obtain $G'[\sigma^{\mathsf{id}}_{p,m}] \equiv \hat{t}_0 :: G'[\sigma^{\mathsf{st}}]$. The terms $G'[\sigma^{\mathsf{id}}_{p,m}]$ and $\hat{t}_0 :: G'[\sigma^{\mathsf{st}}]$ are normal since inserting λ-free terms into normal terms does not create new redicies. Equality follows with unique normal forms (lemma 2.1). □

Lemma 5.15.

1. *If $s[\sigma^{\mathsf{id}}_{p,q}] = a$, then $s = a$.*
2. *If $s[\sigma^{\mathsf{id}}_{p,q}] = b$, then $s = b$.*
3. *If $s[\sigma^{\mathsf{id}}_{p,q}] = \mathsf{S}^k\ t$, then $s = \mathsf{S}^k\ t'$ and $t'[\sigma^{\mathsf{id}}_{p,q}] = t$ for some t'.*
4. *If $s[\sigma^{\mathsf{id}}_{p,q}] = \hat{t}_k$, then $s = \hat{t}_k$.*

Proof. 1. and 2. are immediate from the definition of $\sigma^{\mathsf{id}}_{p,q}$. 3. follows by induction on k. 4. follows from 1. - 3. □

Lemma 5.16.

1. *If $s[\sigma^{\mathrm{id}}_{p,q}] = \hat{t}_k :: s[\sigma^{\mathrm{st}}]$, then $s = r$ or $s = \hat{t}_k :: s'$ for some s'.*
2. *If $s[\sigma^{\mathrm{id}}_{p,q}] = \hat{t}_k :: s[\sigma^{\mathrm{st}}]$, then $s = \hat{t}_k :: \ldots :: \hat{t}_{k+l-1} :: r$ for some l.*

Proof. 1. By case analysis on s and lemma 5.15.
2. Follows by size induction on s. By 1. we have $s = r$ or $s = \hat{t}_k :: s'$. If $s = r$ the claim is immediate. For $s = \hat{t}_k :: s'$, we have $s'[\sigma^{\mathrm{id}}_{p,q}] = \hat{t}_k[\sigma^{\mathrm{st}}] :: s'[\sigma^{\mathrm{st}}]$. The claim follows with lemma 5.11 and the inductive hypothesis. □

Corollary 5.17. *Let G be normal. If $\mathcal{G}_{m,n,p}\ G$, then $G = \hat{T}_l$ for some l.*

Proof. By lemma 5.14 we have $G = \lambda rab.G'$ for some G' with $G'[\sigma^{\mathrm{id}}_{p,m}] = \hat{t}_0 :: G'[\sigma^{\mathrm{st}}]$. The claim follows with lemma 5.16 with $k = 0$. □

We can now recover lemma 5.8.

Lemma 5.18. *If $\mathcal{G}_{m,n,p}\ \hat{T}_l$, then $m = l$ and $m \cdot n = p$.*

Proof. By corollary 5.13 we have $\mathcal{G}_{l,n,l\cdot n}\ \hat{T}_l$.

$$\begin{aligned}
&[\hat{t}_0, \ldots, \hat{t}_{l-1}, \widehat{(l\cdot n, l)}] \\
\equiv\ & \hat{T}_l\ [\widehat{(l \cdot n, l)}]\ a\ b && \text{(corollary 5.12)}\\
\equiv\ & \hat{t}_0 :: \hat{T}_l\ []\ (\mathsf{S}^n\ a)\ (\mathsf{S}\ b) && (\mathcal{G}_{l,n,l\cdot n}\ \hat{T}_l)\\
\equiv\ & \hat{T}_l\ [\widehat{(p, m)}]\ a\ b && (\mathcal{G}_{m,n,p}\ \hat{T}_l)\\
\equiv\ & [\hat{t}_0, \ldots, \hat{t}_{l-1}, \widehat{(p, m)}] && \text{(corollary 5.12)}
\end{aligned}$$

With injectivity of ::, we have $m = l$ and $m \cdot n = p$. □

Corollary 5.19. *Let G be normal. $m \cdot n = p$ and $G = \hat{T}_m$ iff $\mathcal{G}_{m,n,p}\ G$.*

Proof. Follows from Corollaries 5.13 and 5.17 and lemma 5.18. □

Reduction In the definition of $\mathcal{G}_{m,n,p}\ G$, we require that G neither contains a nor b as free variables. When constructing unification equations for multiplication, we use a free variable for G. By turning a and b into bound variables, we can be sure that occurrences of a or b in terms inserted for G are distinct from the variables used in the previous proof. Recall that bound variables are consistently renamed in case a term containing these variables is inserted by substitution. We define:

$$\mathsf{GN}\ x := \lambda ab.x\ (\mathsf{S}\ a) \stackrel{?}{=} \lambda ab.\mathsf{S}\ (x\ a)$$

$$\overline{x \doteq 1} := \lambda ab.x\ a \stackrel{?}{=} \lambda ab.[\![1]\!]_{\mathrm{gn}}\ a$$

$$\overline{x + y \doteq z} := \lambda ab.x\ (y\ a) \stackrel{?}{=} \lambda ab.z\ a \qquad \overline{x \cdot y \doteq z} :=$$

$$\lambda ab.G_{xyz}\ [\langle z\ a, x\ b\rangle]\ a\ b \stackrel{?}{=} \lambda ab.\langle a, b\rangle :: G_{xyz}\ []\ (y\ a)\ (\mathsf{S}\ b)$$

where G_{xyz} is a fresh variable for each choice of x, y, and z. The encodings of the characteristic equation, addition and constants are straightforward adaptations of the equations for Church encodings. To simplify matters, we add the variable b in those equations as well, which allows us to type all equations as $\alpha^2 \to \alpha$:

Lemma 5.20. *Let $\Gamma_D := [(x : \alpha \to \alpha) \mid x \in \mathsf{vars}\ D]$*
$$\cup\ [(G_{xyz} : \alpha^3 \to \alpha) \mid x \cdot y \doteq z \in D].$$

$$\frac{x \in \mathsf{vars}\ D}{\Gamma_D \vdash_2 \mathsf{GN}\ x : \alpha^2 \to \alpha} \qquad \frac{d \in D}{\Gamma_D \vdash_2 \overline{d} : \alpha^2 \to \alpha}$$

Another consequence of Goldfarb numerals is that the adaption of the characteristic equation no longer axiomatises syntactic equality with a numeral. The Goldfarb numeral $[\![1]\!]_{\mathrm{gn}} = \lambda a.\mathsf{S}\ a$ is η-equivalent to S and S satisfies the characteristic equation. For our purposes, the extensional equality $s \approx [\![n]\!]_{\mathrm{gn}} := \forall t\rho.\ s[\rho]\ t \equiv \mathsf{S}^n\ t$ suffices where ρ is a renaming, a substitution which replaces variables by other variables. The renaming allows us to handle open terms in the reduction.

Lemma 5.21. *Let σx be normal for all x.*

$$\sigma \vDash \mathsf{GN}\ x \quad \textit{iff} \quad \sigma x \approx [\![n]\!]_{\mathrm{gn}} \ \textit{for some } n$$

Lemma 5.22. *Let σx be normal for all x and let $\sigma x \approx [\![m]\!]_{\mathrm{gn}}$, $\sigma y \approx [\![n]\!]_{\mathrm{gn}}$, and $\sigma z \approx [\![p]\!]_{\mathrm{gn}}$.*

1. *$m = 1$ iff $\sigma \vDash \overline{x \doteq 1}$.*
2. *$m + n = p$ iff $\sigma \vDash \overline{x + y \doteq z}$.*
3. *$m \cdot n = p$ and $\sigma G_{xyz} = \hat{T}_m$ iff $\sigma \vDash \overline{x \cdot y \doteq z}$.*

Theorem 5.23. $\mathbf{H10} \preceq \mathbf{SU}_2$

Proof. Given D, we construct $\Gamma_D \vdash_2 \overline{D} : (\alpha^2 \to \alpha)^{|\overline{D}|}$ where $\overline{D} = [\overline{d} \mid d \in D] \cup [\mathsf{GN}\ x \mid x \in \mathsf{vars}\ D]$. Typing follows with lemma 5.20.

1. For a solution $\theta \vDash D$, we pick the substitution $\sigma\ G_{xyz} := \hat{T}_{\theta x}$ where we choose θy for n and $\sigma x := [\![\theta x]\!]_{\mathrm{gn}}$. The claim then follows with Lemmas 5.21 and 5.22.
2. Let $\Gamma_D \vdash_2 \sigma : \Delta$ be a substitution with $s[\sigma] \equiv t[\sigma]$ for all $s \stackrel{?}{=} t \in \overline{D}$. Using lemma 2.12, wlog. we may assume that σ only inserts normal terms. We construct a solution θ by means of lemma 5.4. With Lemmas 5.21 and 5.22 $\theta \vDash D$ follows. □

We use $\langle\cdot,\cdot\rangle$ to combine multiple equations to one. Then, by combining all equations in $\overline{D}$, we establish the undecidability of second-order unification.

Lemma 5.24. *$\sigma \vDash \lambda ab.s_1 \stackrel{?}{=} \lambda ab.t_1$ and $\sigma \vDash \lambda ab.s_2 \stackrel{?}{=} \lambda ab.t_2$ iff. $\sigma \vDash \lambda ab.\langle s_1, s_2\rangle \stackrel{?}{=} \lambda ab.\langle t_1, t_2\rangle$*

Corollary 5.25. $\mathbf{H10} \preceq \mathbf{U}_2$

6 Constants

In Section 5, we formalise a proof of the undecidability of second-order unification. The second-order proof requires constants $\mathsf{a} : \alpha$ and $\mathsf{g} : \alpha \to \alpha \to \alpha$, while in Section 2 we assume an arbitrary type of constants C and a constant type signature Ω. Negligible as this may seem, Farmer [1988] proves that second-order unification is decidable in the monadic fragment, the fragment where all constants have at most arity one. In contrast, the proof of the undecidability of third-order unification in Section 4 is independent of the constants of the language.

In the following, we clarify the role of constants in our setting of unification. We formalise techniques for the introduction and elimination of constants without affecting unifiability. The introduction of constants may be understood as the conservativity of unification with respect to the available constants. Combined with the introduction, the elimination of constants can be used to obtain Goldfarb's strongest result [1981], the undecidability of second-order unification in any language which contains a binary function constant $\mathsf{g} : \alpha \to \alpha \to \alpha$. Furthermore, the results of this section allow us to deduce Huet's result of Section 4 from the results of Goldfarb presented in Section 5.

In the formalisation, constants are drawn from Coq types and injections denote type inclusions. On paper we use set notation for finite types and omit type injections and use subtyping instead to ease readability. For an injection $C \hookrightarrow \mathcal{D}$, we denote the subtype of $\mathcal{D}$ which contains all the elements that are not in the image of the injection by $\mathcal{D} \setminus C$. We add the superscript $\cdot^C$ to a unification problem to indicate that all constants are drawn from the type C. For example, to be precise, in Section 5 we consider the problem $\mathbf{U}_2^{\{\mathsf{a},\mathsf{g}\}}$.

6.1 Introduction of Constants

We prove that the introduction of new constants has no effect on nth-order unification. Explicitly, we assume two types $C \hookrightarrow \mathcal{D}$ of constants and corresponding signatures Ω_C and $\Omega_\mathcal{D}$ which agree with the injection, i.e. $\forall c : C.\ \Omega_C\ c = \Omega_\mathcal{D}\ c$. We prove $\mathbf{U}_n^C \preceq \mathbf{U}_n^\mathcal{D}$. The key observation in the reduction is that two terms $\Gamma \vdash_n s \overset{?}{=} t : A$ are unifiable iff they are unifiable using only constants which occur in s or t. Note that for the interesting case of this reduction, transforming a substitution with constants from $\mathcal{D}$ into a substitution with constants from C, it does not suffice to replace the constants in $\mathcal{D} \setminus C$ with variables. As the order of the type of a constant may be $n+1$ in nth-order fragment, a variable of the same type would no longer be contained in the fragment. Instead, we use the technique presented in lemma 3.4, i.e. the inhabitation of arbitrary types, if we are allowed to introduce fresh variables.

Recall that we write consts s for a finite list of the constants that occur in s. Analogous to substitution, we denote the parallel replacement of all constants according to a function κ, mapping constants to terms, by $s[\kappa]$.

Lemma 6.1. *$\succ, \succ^*, \equiv$ are compatible with constant replacement.*

We associate every constant $d : \mathcal{D}$ with a fresh variable x_d and replace constants according to the constant replacement $\kappa_{\mathsf{in}} d \coloneqq d$ if $d : C$ and $\kappa_{\mathsf{in}} d \coloneqq \mathsf{inhab}_{x_d}\ (\Omega_\mathcal{D} d)$ otherwise. We establish:

Lemma 6.2. *Let $D = [d \mid d \in \mathsf{consts}\ (\sigma x), x \in \mathsf{dom}\ \Gamma]$ and $\forall d \in \mathsf{consts}\ s.\ (d : C)$.*

1. $s[\sigma[\kappa_{\mathsf{in}}]] = s[\sigma][\kappa_{\mathsf{in}}]$
2. *If $\Delta \vdash_n \sigma : \Gamma$ and $\forall d \in D.\ (x_d : \mathsf{tar}(\Omega_\mathcal{D} d)) \in \Delta$, then $\Delta \vdash_n \sigma[\kappa_{\mathsf{in}}] : \Gamma$.*

Lemma 6.3. $\mathbf{U}_n^C \preceq \mathbf{U}_n^\mathcal{D}$

Proof. The reduction function is the identity. Let $\Gamma \vdash_n s \overset{?}{=} t : A$ be an nth-order unification instance drawing its constants from C. As $C \hookrightarrow \mathcal{D}$, the forward direction is trivial. For the converse direction, we assume some Σ and $\Sigma \vdash_n \tau : \Gamma$ where $s[\tau] \equiv t[\tau]$. Let $D = [d \mid d \in \mathsf{consts}\ (\tau x), x \in \mathsf{dom}\ \Gamma]$. Pick $\Delta \coloneqq \Sigma \cup [(x_d : \mathsf{tar}(\Omega_\mathcal{D} d)) \mid d \in D]$ and $\sigma \coloneqq \tau[\kappa_{\mathsf{in}}]$. Typing follows by lemma 6.2 and with lemma 6.1 also $s[\tau[\kappa_{\mathsf{in}}]] = s[\tau][\kappa_{\mathsf{in}}] \equiv t[\tau][\kappa_{\mathsf{in}}] = s[\tau[\kappa_{\mathsf{in}}]]$. □

6.2 Elimination of Constants

Considering nth-order unification, we prove that constants whose type is of an order strictly smaller than n can be removed without affecting unifiability. Explicitly, we prove $\mathbf{U}_n^\mathcal{D} \preceq \mathbf{U}_n^C$ if $C \hookrightarrow \mathcal{D}$, Ω_C and $\Omega_\mathcal{D}$ agree with the injection, and $\mathsf{ord}\ (\Omega_\mathcal{D} d) < n$ for all $d : \mathcal{D} \setminus C$. This allows us, for example, to obtain Goldfarb's result in a language without the constant $\mathsf{a} : \alpha$, i.e. $\mathbf{H10} \preceq \mathbf{U}_2^{\{\mathsf{g}\}}$. The decidability of monadic second-order unification entails that we cannot hope to eliminate constants with a type of order n as well. If this was the case, $\mathsf{g} : \alpha \to \alpha \to \alpha$ could be eliminated, making the problem $\mathbf{U}_2^\emptyset$ undecidable, a contradiction to the decidability of monadic second-order unification.

Similar techniques to eliminate constants have been used by Statman [1981].

The main idea behind our reduction is to replace constants by bound variables. Similar to constants, bound variables are not affected by substitution. In contrast to constants, bound variables cannot be used when inserting new terms by substitution. For example, the equation $y : \alpha \vdash_2 y \overset{?}{=} \mathsf{a} : \alpha$ is unifiable but after transforming a into a bound variable x_a, the resulting equation $y : \alpha \vdash_2 \lambda x_\mathsf{a}.y \overset{?}{=} \lambda x_\mathsf{a}.x_\mathsf{a} : \alpha \to \alpha$ is no longer unifiable. We circumvent this problem by passing the bound variables used to replace constants to every free variable. For the above example, we obtain $y : \alpha \to \alpha \vdash_2 \lambda x_\mathsf{a}.y\ x_\mathsf{a} \overset{?}{=} \lambda x_\mathsf{a}.x_\mathsf{a} : \alpha \to \alpha$ which is unifiable.

First, we show how to remove a list of constants $D = [d_1, \dots, d_m]$ with $d_1, \dots, d_m : \mathcal{D} \setminus C$, given $\mathsf{ord}\ (\Omega_\mathcal{D} d_i) < n$

for all $i = 1, \ldots, m$. We then generalise the result to the entire type $\mathcal{D} \setminus C$ with a particular choice of the d_i's. We associate every constant d_i with a fresh variable x_i and introduce the encoding ε which performs the transformation. Variables are replaced according to the substitution $\sigma_\varepsilon x := x\, x_1 \cdots x_m$ and constants according to $\kappa_\varepsilon d := d$ if $d : C$, $\kappa_\varepsilon d := x_i$ if $d = d_i$ for some i, and $\kappa_\varepsilon d := x_0$ for some fixed x_0 otherwise. The transformation ε is given by $\varepsilon s := \lambda x_1 \cdots x_m.s[\sigma_\varepsilon][\kappa_\varepsilon]$. We transform types by $\varepsilon(A) := \Omega_\mathcal{D} d_1 \to \ldots \to \Omega_\mathcal{D} d_m \to A$ and to contexts by $\varepsilon(\Gamma) := [(x : \varepsilon(A)) \mid (x : A) \in \Gamma]$. We introduce the operation $\varepsilon^{-1} s := s\, d_1 \cdots d_m$ which reverses the effect of ε to a certain extend. ε and ε^{-1} are lifted to substitutions by point wise application, i.e. $\varepsilon(\sigma)(x) := \varepsilon(\sigma x)$.

Lemma 6.4. *Let $\forall d \in$ consts $s.\ d \in D \lor d : C$ and $\forall d \in$ consts $(\sigma x).\ d \in D \lor d : C$ for all x.*

1. $\dfrac{\Gamma \vdash_n s : A}{\varepsilon(\Gamma) \vdash_n \varepsilon s : \varepsilon(A)}$ *and* $\dfrac{\Gamma \vdash_n s : \varepsilon(A)}{\Gamma \vdash_n \varepsilon^{-1} s : A}$
2. *$\equiv$ is compatible with ε and ε^{-1}.*
3. *$\varepsilon s[\varepsilon\sigma] \succ^* \varepsilon(s[\sigma])$ and $\varepsilon^{-1}((\varepsilon s)[\sigma]) \succ^* s[\varepsilon^{-1}\sigma]$.*

Lemma 6.5. *$\mathbf{U}_n^\mathcal{D} \le \mathbf{U}_n^C$ provided $C \hookrightarrow \mathcal{D}$, Ω_C and $\Omega_\mathcal{D}$ agree with the injection, and* ord $(\Omega_\mathcal{D} d) < n$ *for all $d : \mathcal{D} \setminus C$.*

Proof. We pick $D := [d \mid d \in \text{consts } s \mathbin{+\!\!+} \text{consts } t, d : \mathcal{D} \setminus C]$ and the function $\varepsilon(\Gamma \vdash_n s \overset{?}{=} t : A) := \varepsilon(\Gamma) \vdash_n \varepsilon s \overset{?}{=} \varepsilon t : \varepsilon(A)$ to show $\mathbf{U}_n^\mathcal{D}(\Gamma \vdash_n s \overset{?}{=} t : A)$ iff $\mathbf{U}_n^C(\varepsilon(\Gamma \vdash_n s \overset{?}{=} t : A))$. □

Corollary 6.6. *Let* g $: \mathcal{D}$ *with* $\Omega_\mathcal{D}\mathrm{g} = \alpha \to \alpha \to \alpha$.

$$\mathbf{H10} \le \mathbf{U}_2^{\{\mathrm{g},\mathrm{a}\}} \le \mathbf{U}_2^{\{\mathrm{g}\}} \le \mathbf{U}_2^\mathcal{D} \qquad \mathbf{U}_2^{\{\mathrm{g},\mathrm{a}\}} \le \mathbf{U}_3^\emptyset \le \mathbf{U}_3^C$$

7 First-Order Unification

We verify a decision procedure for first-order unification in the λ-calculus, following the approach used in a formalisation by Smolka and Husson [2014]. First-order unification is formalised in various proof assistants, we thus only give the proof idea for the simply typed λ-calculus.

Our proof consists of two steps: First, we implement a unification procedure for the λ-free fragment of our calculus. Second, we use normalisation and the unification procedure to decide the problem $\mathbf{U}_1$ for the whole calculus.

We say that a term is λ-free if it does not contain any abstractions. We fix a predicate free which determines whether a variable is considered as free, and should be replaced by substitutions, or as bound, and can thus not be replaced. We say that a substitution σ respects bound variables if $\sigma x = x$ for all variables x with bound x.

The proof is based on decomp E, a decomposition procedure which simplifies a system of equations by generating more, but structurally simpler equations. We first define it for a single equation.

$$\text{decomp } (s \overset{?}{=} s) = [\,] \qquad \text{decomp } (x \overset{?}{=} s) = [x \overset{?}{=} s]$$

$$\text{decomp } (s \overset{?}{=} x) = [x \overset{?}{=} s] \qquad \text{decomp } (s_1\, s_2 \overset{?}{=} t_1\, t_2) =$$

$$\text{decomp } (s_1 \overset{?}{=} t_1) \mathbin{+\!\!+} \text{decomp } (s_2 \overset{?}{=} t_2)$$

$$\text{decomp } (\_ \overset{?}{=} \_) = \emptyset \quad \text{othw.}$$

where $\emptyset$ indicates that there is no sensible decomposition. We lift the definition to systems of equations by decomp $[\,] = [\,]$ and decomp $(s \overset{?}{=} t :: E) =$ decomp $(s \overset{?}{=} t) \mathbin{+\!\!+}$ decomp E. Based on this we define a *unification relation* $E \mapsto \sigma$ which, if read operationally, directly yields a unification procedure:

$$\frac{\text{decomp } E = [\,]}{E \mapsto \text{id}}$$

$$\frac{\text{decomp } E = x \overset{?}{=} s :: E' \quad \text{free } x \quad x \notin \text{vars } s \quad \forall y \in \text{vars } s.\ \text{free } y \quad E'[x] \mapsto \sigma}{E \mapsto \sigma[x := s[\sigma]]}$$

We can prove computability, soundness and completeness of this relation:

Lemma 7.1. *For every E we can compute σ such that $E \mapsto \sigma$, or we can prove that no such σ exists.*

Lemma 7.2. *If $E \mapsto \sigma$, we have*

1. *σ unifies the equations in E.*
2. *σ is λ-free*
3. *σ respects bound variables.*
4. *if* ord $\Gamma \le 1$ *and $\Gamma \vdash_1 E : L$, then $\Delta \vdash_1 \sigma : \Gamma$.*

Lemma 7.3. *If σ respects bound variables and solves E, then there exists τ with $E \mapsto \tau$.*

We need two more observations before we are able to prove the decidability of $\mathbf{U}_1$:

Lemma 7.4. *If $\Gamma \vdash_1 s : A$, s is normal and $s \neq \lambda x.t$ for all t, then s is λ-free.*

Lemma 7.5. *If $\Gamma \vdash_n s \overset{?}{=} t : A$, then $\Delta \vdash_n \xi s \overset{?}{=} \xi t : B$ for some Δ with* ord $\Delta \le n$ *and A with* ord $A \le n + 1$. *Moreover,* $\mathbf{U}_n(\Gamma \vdash_n s \overset{?}{=} t : A)$ *iff* $\mathbf{U}_n(\Delta \vdash_n \xi s \overset{?}{=} \xi t : B)$.

Theorem 7.6. $\mathbf{U}_1$ *is decidable.*

Proof. By lemma 2.8 it suffices to decide whether there exists a unifying substitution inserting normal forms only. Let $\Gamma \vdash_1 s \overset{?}{=} t : A$. Due to lemma 7.5, we can assume wlog. that s and t are normal, ord $\Gamma \le 1$, and ord $A \le 2$. Because inserting normal terms in normal, first-order terms yields normal terms, and on such terms $\equiv$ coincides with equality, it suffices to decide the existence of σ fulfilling

$$\Delta \vdash_1 \sigma : \Gamma,\ s[\sigma] = t[\sigma], \text{ and normal } \sigma x \text{ for all } x.$$

We analyse the form of s and t using lemma 2.5, which are normal. Hence, let $s = \lambda x_1 \cdots x_k.a_1\ s_1 \cdots s_q$ and $t = \lambda x_1 \cdots x_l.a_2\ t_1 \cdots t_r$.

1. Case $k \neq l$. Wlog. let $k < l$. We show both terms cannot be unifiable. We assume that both terms are unifiable and derive a contradiction. If a_1 is a constant, then substitution does not affect a_1 and thus we can obtain an equation between an abstraction and a constant. This is clearly a contradiction. If a_1 is a variable, then the type of this variable must be a function type. A contradiction, since first-order terms cannot contain function variables.
2. Case $k = l$. With lemma 7.4 we know that $a_1\ s_1 \cdots s_q$ and $a_2\ t_1 \cdots t_r$ are λ-free. Thus, we can use the decision procedure from lemma 7.1 to decide whether $[a_1\ s_1 \cdots s_q \overset{?}{=} a_2\ t_1 \cdots t_r] \mapsto \sigma$ for some σ where there variables $x_1, \ldots, x_k$ are considered bound and all others are free. If this is the case, then lemma 7.2, soundness, guarantees that σ unifies s and t. If this is not the case, then completeness (lemma 7.3) guarantees that no σ unifies s and t.

□

8 Formalisation

The overhead generated by using a proof assistant is comparable to detailed paper proofs. Using a proof assistant has prevented us from making small errors, in particular in the multiplication proof in Section 5. The formalisation spans 7300 lines of code, with an additional 2400 lines of code that may be understood as an extension of the standard library. Of those seven thousand lines a significant amount is devoted to developing the meta theory about the λ-calculus and unification. Only about 1000 lines are needed for Goldfarb's result and amongst those a fourth is devoted to multiplication. The simplified version of Huet's result spans 400 lines.

While we use a named representation on paper to improve readability, in Coq we use a De Bruijn [1972] representation of terms. As a consequence, in the formalisation typing contexts are lists. Working with De Bruijn terms in Coq is considerably eased by the Autosubst 2 framework [Stark et al. 2019], which can generate the term syntax, renaming and a substitution operations as well as their correctness lemmas.

In lemma 2.3 we establish that well-typed terms are normalising. We define the logical relation following Schäfer [2019], which was first used by Forster et al. [2019b], to obtain a very short proof of normalisation for our simply-typed λ-calculus with constants.

The technique we use to enumerate predicates and types is based on prior work by Forster et al. [2019a]. The computability proof of the relation $E \vDash \sigma$ is using non-structural recursion, enabled in Coq by the Equations package [Sozeau and Mangin 2019].

Surprisingly challenging was the formalisation of the conservativity of unification. We did not find a previous proof of conservativity in the literature. When transforming a substitution $\Delta \vdash_m \sigma : \Gamma$ into $\Sigma \vdash_n \tau : \Gamma$ for $n \leq m$ free variables and constants of order m have to be replaced with terms of order n. For decreasing the order of constants we had to introduce the constant replacement $s[\kappa]$ and prove it compatible with reduction and typing. In addition, to recover ordertyping we need normal forms meaning after replacing constants and free variables we need to normalise the resulting terms. Normal terms are needed because a substitution of order m may contain unreduced mth-order subterms which are closed and thus remain untouched by substitution. As the proof of lemma 3.6 shows this is no longer the case if we consider normal forms. Further, formalising the techniques from Section 6 required longer proofs than we anticipated. On paper, we easily convinced ourselves that the techniques should work as expected, spelling out all details to constitute a formal proof was more difficult. While not particularly challenging, formalising the meta theory of the STLC caused considerable overhead. For example, where on paper one might just write "…" to indicate multiple arguments being applied to a function, in the formalisation we define a function for applying a list of terms $T = [t_1, \ldots, t_n]$ from the right to a term s. The non-tail-recursive definition $s\ [] := s$ and $s\ (t :: T) := (s\ T)\ t$ of this operation reverses the order of the arguments, i.e. $s\ T = s\ t_n\ \cdots\ t_1$. To improve readability, we deviate slightly from the formalisation on paper and use the canonical order $s\ t_1\ \cdots\ t_n$ whenever possible.

Nevertheless, Coq provides an ideal environment for our proofs. Its constructive nature allows a synthetic approach to undecidability, whereas tactics like setoid rewriting [Sozeau 2009] and especially the Autosubst 2 tool [Stark et al. 2019] made the formalisation of syntax and semantics straightforward.

9 Discussion

To the best of our knowledge, this paper contains the first formalisation of higher-order unification in the λ-calculus with undecidability results. In particular, we are the first to present the results by Huet and Goldfarb and first-order unification in one setting, the STLC. In the context of the Coq library of undecidable problems [Forster et al. 2020b], with Corollaries 3.9 and 5.25 we obtain the synthetic undecidability of higher-order unification $\textbf{Halt} \preceq \textbf{H10} \preceq \textbf{U}_2 \preceq \textbf{U}_{2+k} \preceq \textbf{U}$.

9.1 Related Work

Higher-order unification The undecidability proof in Section 2 is based on [Dowek 1993] which in turn presents the result by Goldfarb [1981]. In this work, we use a different equation to characterise Church numerals.

The undecidability proof of third-order unification is based on prior work by Huet [1973]. Although the result is subsumed by Goldfarb's combined with the techniques to treat constants presented in this work, we formalise Huet's result as it showcases a very different technique for establishing the undecidability of unification problems. We simplified his construction by reducing from **MPCP** instead of the Post-correspondence problem.

For comparison, we also formalised the original reduction by Huet. In his reduction, Huet must ensure that it is not possible to give a unifying substitution which corresponds to an empty sequence of cards. He excludes the empty solution by adding an additional constraint, using a technique similar to lemma 5.24. We do not need such a constraint as the empty sequence is a solution for **MPCP** (then $l_0 = r_0$ for the designated card l_0/r_0). Huet presents his result in a Church-typed setting whereas we use Curry typing throughout the work. We did not notice significant differences arising in the proofs from this decision.

The undecidability of second-order unification is based on prior work by Goldfarb [1981]. Goldfarb uses a calculus without abstractions, using hereditary substitutions for unification. We adapt his work to the STLC which for example entails proving lemma 5.14 and simplify his construction: We distinguish between pairs and lists and we only need a single equation for multiplication. Goldfarb needs two equations to ensure that the terms inserted by substitution do not contain the constants a and b. We accomplish this by using bound variables instead.

Formalised undecidability results Most of the published work on formalised undecidability results is available as part of the library of formalised undecidable problems in Coq [Forster et al. 2020b]. Amongst them are undecidability proofs for the halting problem in the call-by-value λ-calculus [Forster and Smolka 2017], for string rewriting, **PCP** and **MPCP** [Forster et al. 2018], for the halting problem of register machines, binary stack machines and provability in intuitionistic linear logic [Forster and Larchey-Wendling 2019], for various notions in first-order logic [Forster et al. 2019a], and for Hilbert's tenth problem [Larchey-Wendling and Forster 2019]. Apart from the library, there is a formalised undecidability result for provability in System F by Dudenhefner and Rehof [2018] and the undecidability of subtyping in the $D_{<:}$ calculus [Hu and Lhoták 2019] and other calculi related to dependent object types [Hu 2019], all in Coq.

An undecidability proof for Hilbert's tenth problem in Isabelle is work in progress [Bayer et al. 2019].

Working in concrete models of computation, Ramos et al. [2018] formalise the undecidability of a halting problem for a functional language in PVS, Carneiro [2018] formalises Rice's problem based on partial recursive functions in Lean, and Xu et al. [2013] formalise the undecidability of the halting problem of Turing machines in Isabelle.

Formalisation of unification To the best of our knowledge, we are the first to give a formalised presentation of higher-order unification. First-order unification (in the λ-free fragment) has been formalised in numerous proof assistants, amongst them Coq [Rouyer 1992] and Isabelle [Coen 1992]. Nominal unification has been mechanised in Isabelle by Urban et al. [2004] and in HOL4 by Kumar and Norrish [2010].

9.2 Future Work

Farmer [1988] proves the decidability of monadic second-order unification. A natural extension of this work is the formalisation of its decision procedure, determining the status of all $\mathbf{U}_n^C$ problems formally.

In Section 2, we present a semi-decision procedure for higher-order unification. A more efficient semi-decision procedure was introduced by Huet [1975], it would be interesting to mechanise its correctness proof.

Furthermore, there are several variants of unification with undecidability proofs that can be formalised, including E-unification [Degtyarev and Voronkov 1996]. Other problems like pattern unification are known to be decidable [Miller 1991], but lack a formalised proof. Several known undecidability results build on the undecidability of unification problems, like typability in the $\lambda\Pi$-calculus [Dowek 1993] (which is shown undecidable by adapting Huet's proof) or type inference in Curry-style System F_ω [Urzyczyn 1997] (shown undecidable by reduction from $\mathbf{U}_2$). Lastly, it would be a major challenge to bring the undecidability proof of semi-unification [Kfoury et al. 1993], which works by reduction from the mortality problem of Turing machines, into a shape suitable for mechanisation.

Acknowledgments

We would like to thank Dominik Kirst for constructive feedback on how to present the material, Gert Smolka for valuable guidance throughout the project, and the anonymous reviewers for very helpful hints and suggestions.

References

Henk P. Barendregt. 1984. *The Lambda Calculus: Its Syntax and Semantics* (2nd revised ed.). North-Holland.

Jonas Bayer, Marco David, Abhik Pal, Benedikt Stock, and Dierk Schleicher. 2019. The DPRM Theorem in Isabelle (Short Paper). In *10th International Conference on Interactive Theorem Proving (ITP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Mario Carneiro. 2018. Formalizing computability theory via partial recursive functions. *arXiv preprint arXiv:1810.08380* (2018).

Martin David Coen. 1992. *Interactive program derivation*. Technical Report. University of Cambridge, Computer Laboratory.

Martin Davis. 1973. Hilbert's Tenth Problem is Unsolvable. *The American Mathematical Monthly* 80, 3 (1973), 233–269.

Nicolaas Govert De Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Indagationes Mathematicae (Proceedings)*, Vol. 75. Elsevier, 381–392.

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

Gilles Dowek. 1993. The undecidability of typability in the lambda-pi-calculus. In *International Conference on Typed Lambda Calculi and Applications*. Springer, 139–145.

Gilles Dowek. 2001. Higher-Order Unification and Matching. *Handbook of automated reasoning* 2 (2001), 1009–1062.

Andrej Dudenhefner and Jakob Rehof. 2018. A Simpler Undecidability Proof for System F Inhabitation. *TYPES 2018* (2018).

William M Farmer. 1988. A unification algorithm for second-order monadic terms. *Annals of Pure and applied Logic* 39, 2 (1988), 131–174.

Yannick Forster, Edith Heiter, and Gert Smolka. 2018. Verification of PCP-related computational reductions in Coq. In *International Conference on Interactive Theorem Proving*. Springer, 253–269.

Yannick Forster, Dominik Kirst, and Gert Smolka. 2019a. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 38–51.

Yannick Forster and Fabian Kunze. 2019. A Certifying Extraction with Time Bounds from Coq to Call-By-Value Lambda Calculus. In *10th International Conference on Interactive Theorem Proving (ITP 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, John Harrison, John O'Leary, and Andrew Tolmach (Eds.), Vol. 141. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 17:1–17:19. https://doi.org/10.4230/LIPIcs.ITP.2019.17

Yannick Forster, Fabian Kunze, and Maximilian Wuttke. 2020a. Verified Programming of Turing Machines in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM.

Yannick Forster and Dominique Larchey-Wendling. 2019. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 104–117.

Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. 2020b. A Coq Library of Undecidable Problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*. https://github.com/uds-psl/coq-library-undecidability

Yannick Forster, Steven Schäfer, Simon Spies, and Kathrin Stark. 2019b. Call-by-push-value in Coq: operational, equational, and denotational theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 118–131.

Yannick Forster and Gert Smolka. 2017. Weak call-by-value lambda calculus as a model of computation in Coq. In *International Conference on Interactive Theorem Proving*. Springer, 189–206.

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

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Addison-Wesley.

Jason Hu and Ondvrej Lhoták. 2019. Undecidability of $D_{<:}$ and Its Decidable Fragments. *arXiv preprint arXiv:1908.05294* (2019).

Zhong Sheng Hu. 2019. Decidability and Algorithmic Analysis of Dependent Object Types (DOT). http://hdl.handle.net/10012/14964

Gerard Pierre Huet. 1972. Constrained resolution: a complete method for higher-order logic. (1972).

Gérard P Huet. 1973. The undecidability of unification in third order logic. *Information and control* 22, 3 (1973), 257–267.

Gerard P. Huet. 1975. A unification algorithm for typed λ-calculus. *Theoretical Computer Science* 1, 1 (1975), 27–57.

Assaf J Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. 1993. The undecidability of the semi-unification problem. *Information and Computation* 102, 1 (1993), 83–101.

Ramana Kumar and Michael Norrish. 2010. (Nominal) unification by recursive descent with triangular substitutions. In *International Conference on Interactive Theorem Proving*. Springer, 51–66.

Dominique Larchey-Wendling and Yannick Forster. 2019. Hilbert's Tenth Problem in Coq. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, Dortmund, Germany*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 27:1–27:20.

CL Lucchesi. 1972. The undecidability of the unification problem for third order languages. *Report CSRR* 2059 (1972), 129–198.

Alberto Martelli and Ugo Montanari. 1976. *Unification in linear time and space: A structured presentation*. Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche.

Yuri V. Matijasevivc. 1970. Enumerable sets are Diophantine. In *Soviet Mathematics: Doklady*, Vol. 11. 354–357.

Dale Miller. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of logic and computation* 1, 4 (1991), 497–536.

Michael S Paterson and Mark N Wegman. 1978. Linear unification. *J. Comput. System Sci.* 16, 2 (1978), 158–167.

Emil L Post. 1946. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.* 52, 4 (1946), 264–268.

Thiago Mendoncca Ferreira Ramos, César Muñoz, Mauricio Ayala-Rincón, Mariano Moscato, Aaron Dutle, and Anthony Narkawicz. 2018. Formalization of the Undecidability of the Halting Problem for a Functional Language. In *International Workshop on Logic, Language, Information, and Computation*. Springer, 196–209.

John C Reynolds. 1983. Types, abstraction and parametric polymorphism. (1983).

John Alan Robinson. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)* 12, 1 (1965), 23–41.

Joseph Rouyer. 1992. *Développement de l'algorithme d'unification dans le calcul des constructions avec types inductifs*. Ph.D. Dissertation. INRIA.

Steven Schäfer. 2019. *Engineering Formal Systems in Constructive Type Theory*. Ph.D. Dissertation. Saarland University. https://www.ps.uni-saarland.de/~schaefer/thesis/

Gert Smolka and Adrien Husson. 2014. Introduction to Computational Logic. (2014). https://courses.ps.uni-saarland.de/icl/2/Resources Unification.

Wayne Snyder and Jean H Gallier. 1989. Higher order unification revisited: Complete sets of transformations. *Technical Reports (CIS)* (1989), 778.

Matthieu Sozeau. 2009. A New Look at Generalized Rewriting in Type Theory. *Journal of Formalized Reasoning* 2, 1 (2009).

Matthieu Sozeau and Cyprien Mangin. 2019. Equations Reloaded: High-Level Dependently-Typed Functional Programming and Proving in Coq. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019).

Kathrin Stark, Steven Schäfer, and Jonas Kaiser. 2019. Autosubst 2: reasoning with multi-sorted De Bruijn terms and vector substitutions. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 166–180.

Richard Statman. 1981. On the existence of closed terms in the typed λ calculus II: Transformations of unification problems. *Theoretical Computer Science* 15, 3 (1981), 329–338.

Masako Takahashi. 1989. Parallel reductions in λ-calculus. *Journal of Symbolic Computation* 7, 2 (1989), 113–123.

Christian Urban, Andrew M Pitts, and Murdoch J Gabbay. 2004. Nominal unification. *Theoretical Computer Science* 323, 1-3 (2004), 473–497.

Pawel Urzyczyn. 1997. Type Reconstruction in F Ω. *Mathematical. Structures in Comp. Sci.* 7, 4 (Aug. 1997), 329–358. https://doi.org/10.1017/S0960129597002302

Jian Xu, Xingyuan Zhang, and Christian Urban. 2013. Mechanising Turing Machines and Computability Theory in Isabelle/HOL. In *ITP (LNCS)*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.), Vol. 7998. Springer, 147–162.