E-Unification for Second-Order Abstract Syntax

Nikolai Kudasov
Innopolis University, Tatarstan Republic, Russia

Abstract
Higher-order unification (HOU) concerns unification of (extensions of) λ-calculus and can be seen as an instance of equational unification (E-unification) modulo βη-equivalence of λ-terms. We study equational unification of terms in languages with arbitrary variable binding constructions modulo arbitrary second-order equational theories. Abstract syntax with general variable binding and parametrised metavariables allows us to work with arbitrary binders without committing to λ-calculus or use inconvenient and error-prone term encodings, leading to a more flexible framework. In this paper, we introduce E-unification for second-order abstract syntax and describe a unification procedure for such problems, merging ideas from both full HOU and general E-unification. We prove that the procedure is sound and complete.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases E-unification, higher-order unification, second-order abstract syntax

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.10

Acknowledgements I thank Nikolay Shilov for encouragement and fruitful discussions about the contents of the paper. I also thank Alexander Chichigin, Violetta Sim, Timur Fuyzrahmanov, Alexey Stepanov, and Dale Miller for discussions of early versions of this work. I thank Alexander Chichigin, Timur Fuyzrahmanov, and Ikechi Ndukwe for proofreading the paper. Finally, I thank the anonymous reviewers of FSCD-2023 for their valuable and detailed comments.

1 Introduction

Higher-order unification (HOU) is a process of solving symbolic equations with functions. Consider the following equation in untyped λ-calculus that we want to solve for M:

\[ M \ g \ (\lambda z. z \ a) \ ? \ g \ a \] (1)

A solution to this problem (called a unifier) is the substitution \( \theta = [M \mapsto \lambda x. \lambda y. y \ x] \).

Indeed, applying \( \theta \) to the equation we get \( \beta \)-equivalent terms on both sides:

\[ \theta(M \ g \ (\lambda z. z \ a)) = (\lambda x. \lambda y. y \ x) \ g \ (\lambda z. z \ a) \ \equiv \beta \ (\lambda y. y \ g) \ (\lambda z. z \ a) \ \equiv \beta \ (\lambda z. z \ a) \ \equiv \beta \ g \ a \ = \theta(g \ a) \]

Higher-order unification has many applications, including type checking [22] and automatic theorem proving in higher-order logics [24]. In general, HOU is undecidable [12] and searching for a unifier can be rather expensive without non-trivial optimizations. For some problems, a decidable fragment is sufficient to solve for. For instance, Miller’s higher-order pattern unification [23] and its variations [13, 37] are often used for dependent type inference.

Traditionally, HOU algorithms consider only one binder – λ-abstraction. A common justification is an appeal to Higher-Order Abstract Syntax (HOAS) [27]:

It is well-known that λ-abstraction is general enough to represent quantification in formulae, abstraction in functional programs, and many other variable-binding constructs [27]. (Nipkow and Prehofer [25, Section 1])
However, HOAS has received some criticism from both programming language implementors and formalisation researchers, who argue that HOAS and its variants [2, 36] have some practical issues [19, 5], such as being hard to work under binders, having issues with general recursion [18], and lacking a formal foundation [9].

Fiore and Szamozvancev [9] argue that existing developments for formalising, reasoning about, and implementing languages with variable bindings 'offer some relief, however at the expense of inconvenient and error-prone term encodings and lack of formal foundations'. Instead, they suggest to consider second-order abstract syntax [10], that is, abstract syntax with variable binding and parametrised metavariables. Indeed, Fiore and Szamozvancev [9] use second-order abstract syntax to generate metatheory in Agda for languages with variable bindings.

In this paper, we develop a mechanisation of equational reasoning for second-order abstract syntax. We take inspiration in existing developments for HOU and E-unification. Although we cannot directly reuse all HOU ideas that rely heavily on the syntax of λ-calculus, we are still able to adapt many of them, since second-order abstract syntax provides parametrised metavariables which are similar to flex terms in HOU.

1.1 Related Work

To the best of our knowledge, there does not exist a mechanisation of equational reasoning for second-order abstract syntax. Thus, we compare our approach with existing HOU algorithms that encompass equational reasoning. Snyder’s higher-order E-unification [28] extends HOU with first-order equational theories. Nipkow and Prehofer [25] study higher-order rewriting and (higher-order) equational reasoning. As mentioned, these rely on λ-abstraction and a HOAS-like encoding to work with other binding constructions. In contrast, we work with arbitrary binding constructions modulo a second-order equational theory.

Dowek, Hardin, and Kirchner [8] present higher-order unification as first-order E-unification in λσ-calculus (a variant of λ-calculus with explicit substitutions) modulo βη-reduction. Their idea is to use explicit substitutions and de Bruijn indices so that metavariable substitution cannot result in name capture and reduces to grafting (first-order substitution). In this way, algorithms for first-order E-unification (such as narrowing) can be applied. Kirchner and Ringeissen [17] develop that approach for higher-order equational unification with first-order axioms. In our work, parametrised metavariables act in a similar way to metavariables with explicit substitutions in λσ-calculus. While it should be possible to encode second-order equations as first-order equations in σ-calculus (with explicit substitution, but without λ-abstraction and application), it appears that this approach requires us to also encode rules of our unification procedure.

As some equational theories can be formulated as term rewriting systems, a line of research combining rewrite systems and type systems exists, stemming from the work of Tannen [33], which extends simply-typed λ-calculus with higher-order rewrite rules. Similar extensions exist for the Calculus of Constructions [1, 35, 29, 30] and λΠ-calculus [7]. Cockx, Tabareau, and Winterhalter [6] introduce Rewriting Type Theory (RTT) which is an extension of Martin-Löf Type Theory with (first-order) rewrite rules. Chrząszcz and Walukiewicz-Chrząszcz [3] discuss how to extend Coq with rewrite rules. Cockx [4] reports on a practical extension of Agda with higher-order non-linear rewrite rules, based on the same ideas as RTT [6]. Rewriting is especially useful in proof assistants that compare types (and terms) through normalisation by evaluation (NbE) rather than higher-order unification. Contrary to type theories extended with rewrite rules, our approach relies on simply-typed syntax, but allows for an arbitrary second-order equational theory, enabling unification even in the absence of a confluent rewriting system.
Kudasov [20] implements higher-order (pre)unification and dependent type inference in Haskell for an intrinsically scoped syntax using so-called free scoped monads to generate the syntax of the object language from a data type describing node types. Such a definition is essentially a simplified presentation of second-order abstract syntax. Kudasov’s pre-unification procedure contains several heuristics, however no soundness or completeness results are given in the preprint.

1.2 Contribution

The primary contribution of this paper is the introduction of $E$-unification for second-order abstract syntax and a sound and complete unification procedure. The rest of the paper is structured as follows:
- In Section 2, we briefly revisit second-order abstract syntax, equational logic, and term rewriting à la Fiore and Hur [10].
- In Section 3, we generalise traditional $E$-unification concepts of an $E$-unification problem and an $E$-unifier for a set of second-order equations $E$.
- In Section 4, we define the unification procedure that enumerates solutions for any given $E$-unification problem and prove it sound.
- In Section 5, we prove completeness of our unification procedure, taking inspiration from existing research on $E$-unification and HOU.
- Finally, we discuss some potential pragmatic changes for a practical implementation as well as limitations of our approach in Section 6.

2 Second-Order Abstract Syntax

In this section, we recall second-order abstract syntax, second-order equational logic, and second-order term rewriting of Fiore and Hur [10].

2.1 Second-Order Terms

We start by recalling a notion of second-order signature, which essentially contains information about the syntactic constructions (potentially with variable bindings) of the object language.

A second-order signature [10, Section 2] $\Sigma = (T, O, | - |)$ is specified by a set of types $T$, a set of operators $O$, and an arity function $| - | : O \rightarrow (T^* \times T^*)^* \times T$. For an operator $F \in O$, we write $F : (\sigma_1, \tau_1, \ldots, \sigma_n, \tau_n) \rightarrow \tau$ when $|F| = ((\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n), \tau)$. Intuitively, this means that an operator $F$ takes $n$ arguments each of which binds $n_i = |\sigma_i|$ variables of types $\sigma_{i,1}, \ldots, \sigma_{i,n_i}$ in a term of type $\tau_i$.

For the rest of the paper, we assume an ambient signature $\Sigma$, unless otherwise stated.

A typing context [10, Section 2] $\Theta | \Gamma$ consists of metavariable typings $\Theta$ and variable typings $\Gamma$. Metavariable typings are parametrised types: a metavariable of type $[\sigma_1, \ldots, \sigma_n] \tau$, when parametrised by terms of type $\sigma_1, \ldots, \sigma_n$, will yield a term of type $\tau$. We will write a centered dot ($\cdot$) for the empty (meta)variable context.

For example, this context has a metavariable $M$ with two parameters and variables $x, y$:
$\Theta | \Gamma = (M : [\sigma, \sigma \Rightarrow \tau] \tau | x : \sigma \Rightarrow \tau, y : \sigma)$.  

1 In literature on $E$-unification, authors use the term functional symbol instead.

2 We follow the terminology of Fiore and Hur.
variables $\sigma$  
operators $\sigma$  

\[
\begin{array}{c}
x : \tau \in \Gamma \\
\Theta | \Gamma \vdash x : \tau
\end{array}
\quad \text{for all } i = 1, \ldots, n
\begin{array}{c}
M : [\sigma_1, \ldots, \sigma_n] \tau \in \Theta \\
\Theta | \Gamma \vdash M[t_1, \ldots, t_n] : \tau
\end{array}
\]

\[
F : (\overline{\tau_1}, \ldots, \overline{\tau_n}) \rightarrow \tau
\quad \text{for all } i = 1, \ldots, n
\begin{array}{c}
\Theta | \Gamma \vdash F(\overline{\tau_i}t_1, \ldots, x_n t_n) : \tau
\end{array}
\]

\[\xi \quad \text{metavariables}\]

\[\text{Figure 1 Second-order terms in context.}\]

\section*{Definition 1 ([10, Section 2]).} A judgement for typed terms in context $\Theta \mid \Gamma \vdash t : \tau$ is defined by the rules in Figure 1. Variable substitution on terms is defined in a usual way, see [10, Section 2] for details.

Let $\Theta = (M_i : (\overline{\tau_i})_{i \in \{1, \ldots, n\}})$, and consider a term $\Theta \mid \Gamma \vdash t : \tau$, and for all $i \in \{1, \ldots, n\}$ a term in extended\(^1\) context $\Xi \mid \Gamma, \Delta, \overline{\tau_i} : \tau_i \vdash t_i : \tau_i$. Then, metavariable substitution $t[M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}}$ is defined recursively on the structure of $t$:

\[x[M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}} = x\]

\[M_k[s][M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}} = t_k[s \mapsto s[M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}}]\quad \text{when } k \in \{1, \ldots, n\} \text{ and } |s| = |\overline{x_i}|\]

\[N[s][M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}} = N[s \mapsto N[M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}}]\quad \text{when } N \not\in \{M_1, \ldots, M_n\}\]

\[F(\overline{\tau_i} s) [M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}} = F(\overline{\tau_i} s \mapsto F(\overline{\tau_i} s[M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}}))\]

We write $\Theta : \Gamma \rightarrow \Xi \mid \Gamma, \Delta$ for a substitution $\Theta = [M_i[\overline{x_i}] \mapsto t_i]_{i \in \{1, \ldots, n\}}$.

When both $\Gamma$ and $\Delta$ are empty, we write $\Theta : \Gamma \rightarrow \Xi$ as a shorthand for $\Theta : \emptyset \rightarrow \emptyset \mid \emptyset$.

For single metavariable substitutions in a larger context we will omit the metavariables that map to themselves. That is, we write $[M_k[\overline{x_i}] \mapsto t_k] : \Theta \mid \Gamma \rightarrow \Xi \mid \Gamma, \Delta$ to mean that $t_i = M_i[\overline{x_i}]$ for all $i \neq k$.

\subsection*{2.2 Second-Order Equational Logic}

We now define second-order equational presentations and rules of second-order logic, following Fiore and Hur [10, Section 5]. This provides us with tools for reasoning modulo second-order equational theories, such as $\beta\eta$-equivalence of $\lambda$-calculus.

An equational presentation [10, Section 5] is a set of axioms each of which is a pair of terms in context.

\section*{Example 2.} Terms of simply-typed $\lambda$-calculus are generated with a family of operators (for all $\sigma, \tau$) $\text{abs}^{\sigma\rightarrow\tau} : \sigma \rightarrow (\sigma \Rightarrow \tau)$ and $\text{app}^{\sigma\rightarrow\tau} : (\sigma \Rightarrow \tau, \sigma) \rightarrow \tau$. And equational presentation for simply-typed $\lambda$-calculus is given by a family of axioms:

\[M : [\sigma] \tau, N : [\sigma] \mid \Gamma \vdash \text{app}(\text{abs}(x.M[x]), N)[]) \equiv M[N][] : \tau \quad (\beta)\]

\[M : [\sigma \Rightarrow \tau] \mid \Gamma \vdash \text{abs}(x.\text{app}(M[]), x) \equiv M[] : \sigma \Rightarrow \tau \quad (\eta)\]

\(^3\) Here we slightly generalise the definition of Fiore and Hur by allowing arbitrary extension of context to $\Gamma, \Delta$ in the resulting term. This is useful in particular when $\Gamma$ is empty. See Definition 26.
We also recognise a subclass of problems in solved form, i.e. problems that have an immediate solution. For the most part, this is a straightforward generalisation of standard concepts of E-unification [11].

In this section, we formulate the equational unification problem for second-order abstract syntax, describe what constitutes a solution to such a problem and whether it is complete. We also recognise a subclass of problems in solved form, i.e. problems that have an immediate solution. For the most part, this is a straightforward generalisation of standard concepts of E-unification [11].
We write a finite set of second-order constraints in a shared metavariable context, where variable context is split into two components: a finite set of variables that we cannot substitute (bound variables) and a finite set that we need to solve for (free variables). Metavariables are always treated existentially, so we do not split metavariable context. Similarly to equational representations, we can parametrise (a set of) constraints, yielding a parametrised version of Item 2. Item 4 is equivalent to Item 3 modulo \( \Theta = \Gamma \). Thus, from now on, we will assume \( \Gamma = \Gamma \) (i.e. \( \Gamma \) is empty) for all constraints.

**Definition 3.** A second-order constraint \( \Theta \mid \Gamma \vdash s \equiv t : \tau \) is a pair of terms in a context, where variable context is split into two components: \( \Gamma = (\Gamma^1, \Gamma^2) \).

The idea is that \( \Gamma^1 \) contains variables that we need to solve for (free variables), while \( \Gamma^2 \) contains variables that we cannot substitute (bound variables). Metavariables are always treated existentially, so we do not split metavariable context. Similarly to equational representations, we can parametrise (a set of) constraints, yielding \( \Theta, \Gamma \mid \Gamma \vdash s \equiv t : \tau \).

**Example 4.** Assume \( \alpha = \sigma \Rightarrow \tau, \beta = (\sigma \Rightarrow \tau) \Rightarrow \tau \). The following are equivalent:

1. For all \( g : \alpha, a : \sigma \), find \( m : \alpha \Rightarrow \beta \Rightarrow \tau \) such that \( m \cdot g (\lambda z. z a) \equiv g a : \tau \).
2. \( \cdot \mid m : \alpha \Rightarrow \beta \Rightarrow \tau, g : \alpha, a : \sigma \vdash \text{app}(\text{app}(m, g), \text{abs}(z. \text{app}(z, a))) \equiv \text{app}(g, a) : \tau \)
3. \( M : [\alpha \Rightarrow \beta \Rightarrow \tau \mid g : \alpha, a : \sigma \vdash \text{app}(\text{app}(M), g), \text{abs}(z. \text{app}(z, a))) \equiv \text{app}(g, a) : \tau \)
4. \( M : [\alpha, \beta \Rightarrow \tau \mid g : \alpha, a : \sigma \vdash \text{app}(\text{app}(M, g), \text{abs}(z. \text{app}(z, a))) \equiv \text{app}(g, a) : \tau \)

Here, Item 2 is a direct encoding of Item 1 as a second-order constraint. Item 3 is a parametrised version of Item 2. Item 4 is equivalent to Item 3 modulo \( \beta \)-equality, witnessed by metasubstitutions \( [M] \mapsto \text{abs}(x. \text{abs}(y. \text{app}(x, y))) \) and \( [M, x, y] \mapsto \text{app}(\text{app}(M, x), y) \).

**Definition 5.** Given an equational presentation \( E \), an \( E \)-unification problem \( \Theta, \Gamma \vdash s \equiv t : \tau \) is a finite set \( S \) of second-order constraints in a shared metavariable context \( \Theta \). We present an \( E \)-unification problem as a formula of the following form:

\[
\exists (M_1 : [\alpha] \tau_1, \ldots, M_n : [\alpha] \tau_n), (\forall (z_1 : \tau_1), s_1 \equiv t_1 : \tau_1) \wedge \ldots \wedge (\forall (z_k : \tau_k), s_k \equiv t_k : \tau_k)
\]

**Definition 6.** A metavariable substitution \( \xi : \Theta \rightarrow \Xi \) is called an \( E \)-unifier for an \( E \)-unification problem \( \Theta, \Gamma \vdash s \equiv t : \tau \) if for all constraints \( \Theta \mid \Gamma \vdash s \equiv t : \tau \) in \( S \) we have

\[
\Xi \mid \Gamma_{\Xi} \vdash \xi s \equiv_{E} \xi t : \tau
\]

We write \( U_{E}(S) \) for the set of all \( E \)-unifiers for \( \Theta, S \).

**Example 7.** Consider unification problem \( \Theta, S \) for the simply-typed \( \lambda \)-calculus:

\[
\Theta = M : [\alpha \Rightarrow \tau, (\sigma \Rightarrow \tau) \Rightarrow \tau]r
\]

\[
S = \{ \Theta \mid g : \sigma \Rightarrow \tau, y : \sigma \vdash M[g, \text{app}(x, y)] \equiv \text{app}(g, y) : \tau \}
\]

Substitution \( [M[z_1, z_2] \mapsto \text{app}(z_2, z_1)] : \Theta \rightarrow \cdot \) is an \( E \)-unifier for \( \langle \Theta, S \rangle \).
3.1 Unification Problems in Solved Form

Here, we recognise a class of trivial unification problems. The idea is that a constraint that looks like a metavariable substitution can be uniquely unified. A unification problem can be unified as long as substitutions for constraints are sufficiently disjoint. More precisely:

Definition 8. An E-unification problem \( ⟨Θ, S⟩ \) is in solved form when S consists only of constraints of the form \( Θ, M : [σ]τ \mid ΓV ⊢ M[σ] \equiv t : τ \) such that

1. \( σ \mid ΓV \subseteq ΓV \) (parameters of M are distinct variables from \( ΓV \))
2. \( Θ \mid ΓV \vdash t : τ \) \( (M\) and variables not occurring in \( Γ \) do not occur in \( t \))
3. all constraints have distinct metavariables on the left hand side

Example 9. Let \( Θ = (M : [σ, σ]σ) \). Then

1. \( \{Θ \mid x : σ, y : σ \vdash M[y, x] \equiv \text{app}(\text{abs}(z.x), y) : σ\} \) is in solved form;
2. \( \{Θ \mid x : σ, y : σ \vdash M[x, x] \equiv \text{app}(\text{abs}(z.x), y) : σ\} \) is not in solved form, since parameters of \( M \) are not distinct variables and also since variable \( y \) occurs on the right hand side, but does not occur in parameters of \( M \);
3. \( \{Θ \mid f : σ \Rightarrow σ, y : σ \vdash M[y, \text{app}(f, y)] \equiv \text{app}(f, y) : σ\} \) is not in solved form, since second parameter of \( M \) is not a variable.

Proposition 10. An E-unification problem \( ⟨Θ, S⟩ \) in solved form has an E-unifier.

Proof. Assume \( Θ = \{Θ \mid Γ \vdash M[σ] \equiv t_i : τ_i\} \). Let \( ξ_S = [M[σ] \Rightarrow t_i]_{i \in \{1, ..., n\}} \). Note that \( ξ_S \) is a well formed metasubstitution since, by assumption, each \( Γ \) is a sequence of distinct variables, \( t_i \) does not reference other variables or \( M_i \), and each metavariable \( M_i \) is mapped only once in \( ξ_S \). Applying \( ξ_S \) to each constraint we get trivial constraints, which are satisfied by reflexivity: \( Θ \mid Γ \vdash t_i ≡ t_i : τ_i \). Thus, \( ξ_S \) is an E-unifier for \( ⟨Θ, S⟩ \).

Later, we will refer to the E-unifier constructed in the proof of Proposition 10 as \( ξ_S \).

3.2 Comparing E-unifiers

In general, a unification problem may have multiple unifiers. Here, we generalise the usual notion of comparing E-unifiers [11] to the second-order abstract syntax using the subsumption order, leading to a straightforward generalisation of the ideas of the most general unifier and a complete set of unifiers. We do not consider generalising essential unifiers [14, 32] or homeomorphic embedding [31], although these might constitute a prospective future work.

Definition 11. Two metavariable substitutions \( θ, ξ : Θ \rightarrow Ξ \) are said to be equal modulo E (notated \( θ ≡_E ξ \)) if for all metavariables \( M : [σ]τ \in Θ \), any context \( Γ \), and any terms \( θ \mid Γ \vdash t_i : σ_i \) for all \( i \in \{1, ..., n\} \) we have

\[ Ξ \mid Γ \vdash M[t_1, \ldots, t_n] \equiv E Ξ[M[t_1, \ldots, t_n]] : τ \]

We say that \( θ \) is more general modulo E than \( ξ \) (notated \( θ \preceq_E ξ \)) when there exists a substitution \( η : Ξ \rightarrow Ξ \) such that \( η \circ θ \equiv_E ξ \).

Empty substitution is more general than any substitution. A more interesting example may be found in λ-calculus. Let

\[ θ_1 = [M[x, y] \mapsto \text{app}(N[x], y)] \]
\[ θ_2 = [M[x, y] \mapsto \text{app}(\text{abs}(z.x), y), N[x] \mapsto \text{abs}(z.x)] \]
\[ θ_3 = [M[x, y] \mapsto x, N[x] \mapsto \text{abs}(z.x)] \]

Then, \( θ_1 \preceq_E θ_2, θ_2 \equiv_E θ_3 \), and \( θ_1 \preceq_E θ_3 \) (witnessed by \( [N[x] \mapsto \text{abs}(z.x)] \circ θ_1 \equiv_E θ_3 \)).
Proposition 12. If \( \theta \equiv_E \xi \) then for any \( E \)-unification problem \((\Theta, S)\) we have \( \theta \in U_E(S) \) iff \( \xi \in U_E(S) \).

Proof. For each constraint \( \Theta \models \frac{\tau}{s \rightarrow t} \) by induction on the structure of \( s \) and \( t \) it is straightforward to show that \( \exists \frac{\theta \in \Theta \models \xi \equiv \xi}{\theta \equiv \xi \in U_E(S)} \).

Corollary 13. If \( \theta \preceq_E \xi \) and \( \theta \in U_E(S) \) then \( \xi \in U_E(S) \).

Not all substitutions can be compared. Consider untyped lambda calculus with \( * \) being the type of any term. Let \( \Theta = (M : [*,*]* \) and

\[
\begin{align*}
\theta &= [M[z1, z2] \mapsto app(z2, z1)] \\
\xi &= [M[z1, z2] \mapsto app(z1, app(z2, abs(z, z)))]
\end{align*}
\]

None of these substitutions is more general modulo equational theory \( E \) of untyped \( \lambda \)-calculus than the other. At the same time, both are \( E \)-unifiers for the problem

\[
\exists (M : [*,*]* \). \forall (g : *, y : *). M[g, abs(x, app(x, y))] \equiv app(g, y) : *
\]

3.3 Complete Sets of \( E \)-unifiers

While there is sometimes more than one solution to an \( E \)-unification problem, we may often hope to collect several sufficiently general unifiers into a single set:

Definition 14. Given an \( E \)-unification problem \((\Theta, S)\), a (minimal) complete set of \( E \)-unifiers for \((\Theta, S)\) (notated \( CSU_E(S) \)) is a subset of \( U_E(S) \) such that

1. (completeness) for any \( \eta \in U_E(S) \) there exists \( \theta \in CSU_E(S) \) such that \( \theta \preceq_E \eta \);
2. (minimality) for any \( \theta, \xi \in CSU_E(S) \) if \( \theta \preceq_E \xi \) then \( \theta = \xi \).

We reserve the notation \( CSU_E(S) \) to refer to minimal complete sets of \( E \)-unifiers (i.e. satisfying both conditions).

Example 15. The \( E \)-unification problem \((\Theta, S)\) in untyped \( \lambda \)-calculus has an infinite \( CSU_E(S) \):

\[
\langle \Theta, S \rangle = \exists (M : [*,*]* \). \forall (g : *, y : *). M[g, abs(x, app(x, y))] \equiv app(g, y) : *
\]

\[
CSU_E(S) = \{ [M[z1, z2] \mapsto app(z2, z1)], [M[z1, z2] \mapsto app(z1, app(z2, abs(z, x))]), [M[z1, z2] \mapsto app(app(z2, abs(x, abs(f, app(f, x))), z1)), \ldots \}
\]

Proposition 16. For any two minimal complete sets of \( E \)-unifiers \( CSU^1_E(S) \) and \( CSU^2_E(S) \), there exists a bijection \( f : CSU^1_E(S) \leftrightarrow CSU^2_E(S) \) such that

\[
\forall \theta \in CSU^1_E(S). \quad \theta \equiv_E f(\theta)
\]

Thus, \( CSU_E(S) \) is unique up to a bijection modulo \( E \), so from now on we will refer to the complete set of \( E \)-unifiers.

Definition 17. When the complete set of \( E \)-unifiers \( CSU_E(S) \) is a singleton set, then we refer to its element as the most general \( E \)-unifier of \( S \) (notated \( mgu_E(S) \)).
Example 18. Consider this $E$-unification problem $\langle \Theta, S \rangle$ in simply-typed $\lambda$-calculus:

$$\exists (m : [\sigma \Rightarrow \tau, (\sigma \Rightarrow \tau) \Rightarrow \tau]). \forall (g : \sigma \Rightarrow \tau, y : \sigma). M[g, \text{abs}(x.\text{app}(x, y))] \triangleright app(g, y) : \tau$$

For this problem the most general $E$-unifier exists: $\text{mgu}_E(S) = [m[z_1, z_2] \mapsto \text{app}(z_2, z_1)]$.

This example differs from Example 15 as here we work in simply-typed lambda calculus.

Proposition 19. If $\langle \Theta, S \rangle$ is an $E$-unification problem in solved form, then $\text{mgu}_E(S) \equiv E \xi_S$.

Proof. It is enough to check that for any $E$-unifier $\theta \in U_E(S)$ we have $\xi_S \triangleq E \theta$. Observe that $\theta \equiv E \theta \circ \xi_S$ since for any constraint $(\Theta | \Gamma \vdash M[\pi] \triangleright t : \tau) \in S$ such that $M : [\pi] \tau \in \Theta$, any context $\Gamma$, and any terms $\Theta | \Gamma \vdash t_i : \sigma_i$ (for all $i \in \{1, \ldots, n\}$) we have

$$\Xi | \Gamma \vdash \theta M[\tilde{t}] \equiv E \theta t_i[\pi \mapsto \tilde{t}] \equiv E \theta (\xi_S M[\pi])[\pi \mapsto \tilde{t}] \equiv E \theta (\xi_S M[\tilde{t}]) : \tau$$

4 Unification Procedure

In this section, we introduce a unification procedure to solve arbitrary $E$-unification problems over second-order abstract syntax. We show that the procedure is sound at the end of this section, and we devote Section 5 for the completeness result.

Our unification procedure has features inspired by classical $E$-unification and HOU algorithms. For the equational part, we took inspiration from the complete sets of transformations for general (first-order) $E$-unification of Gallier and Snyder [11]. For unification of metavariables, we took inspiration from Huet’s higher-order pre-unification [15] and Jensen and Pietrzykowski’s procedure [16]. Some key insights from the recent work by Vukmirovic, Bentkamp, and Nummelin [34] give us the opportunity to improve the algorithm further, however, we are not attempting to achieve an efficient $E$-unification for second-order abstract syntax in this paper.

Note that we cannot directly reuse HOU ideas in our procedure, since we do not have full $\lambda$-calculus at our disposal. Instead we only have parametrised metavariables $M[t_1, \ldots, t_n]$ which are analogous to applications of variables in HOU ($m t_1 \ldots t_n$). Still, we can adapt some ideas if they do not rely on normalisation or specific syntax of $\lambda$-calculus. For other ideas, we introduce simpler, yet more general versions. This allows us to preserve completeness, perhaps, sacrificing some efficiency, making the search space larger. While we believe it is possible to optimise our procedure to have virtually the same running time for unification problems in $\lambda$-calculus as HOU algorithms mentioned above, we leave such optimisations for future work.

To produce the unification procedure we follow and generalise some of the common steps that can be found in literature on HOU and first-order $E$-unification:

1. Classify substitutions that will constitute partial solutions for certain classes of constraints.

The idea is that an overall solution will emerge as a composition of partial solutions.

2. Define transition rules that make small steps towards a solution.

3. Determine when to stop (succeed or fail).

4. If possible, organize rules in a proper order, yielding a unification procedure.

4.1 Bindings

Now we define different elementary substitutions that will serve as partial solutions for some constraints in our unification procedure. Here, we generalise a list of bindings collected by Vukmirovic, Bentkamp, and Nummelin [34]. From that list, Huet-style projection (also
known as partial binding in HOU literature) is not used. Instead, imitation for axioms and JP-style projection bindings cover all substitutions that can be generated by Huet-style projection bindings\(^4\). We also use a simplified version of iteration binding here, again, since it generates all necessary bindings when considered together with generalised imitation binding.

\textbf{Definition 20.} We define the following types of bindings $\zeta$:

- **JP-style projection** for $M$. If $M : [\sigma_1, \ldots, \sigma_k]^\tau$ and $\sigma_1 = \tau$ then
  
  \[ \zeta = [M[z_i] \mapsto z_i] \text{ is a JP-style projection binding} \]

- **Imitation** for $M$. If $M : [\sigma_1, \ldots, \sigma_k]^\tau$, $F : (\overline{\alpha_1, \beta_1}, \ldots, \overline{\alpha_n, \beta_n}) \rightarrow \tau$ and $M_i : [\sigma_1, \ldots, \sigma_k, \overline{\alpha_i, \beta_i}]$ for all $i$,
  
  \[ \zeta = [M[z_i] \mapsto F(\overline{\alpha_i, M_i[z_i]}, \ldots, \overline{\alpha_n, M_n[z_i]}))] \text{ is an imitation binding} \]

- **Elimination** for $M$. If $M : [\sigma_1, \ldots, \sigma_k]^\tau$ and $1 \leq j_1 < j_2 < \ldots < j_{n-1} < j_n \leq k$ such that
  
  \[ E : [\sigma_1, \ldots, \sigma_j]^\tau \text{ then} \]
  
  \[ \zeta = [M[z_i] \mapsto E[z_{j_1}, \ldots, z_{j_n}]] \text{ is a (parameter) elimination binding} \]

- **Identification** of $M$ and $N$. If $M : [\sigma_1, \ldots, \sigma_k]^\tau$, $N : [\nu_1, \ldots, \nu_l]^\tau$, $I : [\sigma_1, \ldots, \sigma_k, \nu_1, \ldots, \nu_l]^\tau$, $M_i : [\sigma_1, \ldots, \sigma_k, \nu_i]$ for all $i \in \{1, \ldots, l\}$, and $N_j : [\nu_1, \ldots, \nu_l][\sigma_j]$ for all $j \in \{1, \ldots, k\}$ then
  
  \[ \zeta = [M[z_i] \mapsto I(\overline{\sigma_i, M_i[z_i]}, \ldots, \overline{\sigma_k, M_k[z_i]})), N[\overline{\nu_1, \ldots, \nu_l}] \mapsto I(\overline{\sigma_1, N_1[\overline{\nu_1, \ldots, \nu_l}]}, \ldots, \overline{\sigma_k, N_k[\overline{\nu_1, \ldots, \nu_l]})] \text{ is an identification binding} \]

- **Iteration** for $M$. If $M : [\sigma_1, \ldots, \sigma_k]^\tau$, $F : (\overline{\alpha_1, \beta_1}, \ldots, \overline{\alpha_n, \beta_n}) \rightarrow \tau$, $H : [\sigma_1, \ldots, \sigma_k, \gamma]^\tau$, and $M_i : [\sigma_1, \ldots, \sigma_k, \overline{\alpha_i, \beta_i}]$ for all $i$, then
  
  \[ \zeta = [M[z_i] \mapsto H(\overline{\sigma_i, F(\overline{\alpha_i, M_i[z_i]}, \ldots, \overline{\alpha_n, M_n[z_i]})))] \text{ is an iteration binding} \]

The iteration bindings allow to combine parameters of a metavariable in arbitrary ways. This is also particularly influenced by the fact that the type $\gamma$ used in the bindings may be arbitrary. This type of bindings introduce arbitrary branching in the procedure below, so should be used with care in pragmatic implementations. Intuitively, we emphasize two distinct use cases for the iteration bindings:

1. To extract a new term from one or more parameters by application of an axiom. In this case, we use iteration, where the root of one of the sides of an axiom is used as an operator $F$.
2. To introduce new variables in scope. In this case, any operator that introduces at least one variable into scope is used in an iteration. This case is important for the completeness of the procedure. See Example 32.

\subsection*{4.2 Transition Rules}

We will write each transition rule of the unification procedure in the form $(\Theta \mid \Gamma \vdash \gamma : \tau) \xrightarrow{\delta} (\Xi, S)$, where $\Theta : \Theta \rightarrow \Xi$ is a metavariable substitution and $S$ is a new set of constraints that is supposed to replace $\gamma : \tau$. We will often write $S$ instead of $(\Xi, S)$ when $\Xi$ is understood from context.

We will now go over the rules that will constitute the $E$-unification procedure when put in proper order. The first two rules are straightforward.

\textbf{Definition 21 (delete).} If a constraint has the same term on both sides, we can delete it:

\[ (\Theta \mid \Gamma \vdash t \equiv t : \tau) \xrightarrow{id} \emptyset \]

\(^4\) Note, that Huet-style projection cannot be formulated in pure second-order abstract syntax as it explicitly relies on $\text{abs}$ and $\text{app}$. Thus, in $E$-unification we can recover such projections only by using axioms in some form. Kudakov [20] implements a heuristic that resembles a generalisation of Huet-style projections. We leave proper generalisations for future work.
We say (metavariables are fresh)

Let \( F : (\overline{\sigma}, \tau_1, \ldots, \overline{\sigma}, \tau_n) \to \tau \), then we can decompose a constraint with \( F \) on both sides into a set of constraints for each pair of (scoped) subterms:

\[
(\Theta | \Gamma \vdash F(\overline{\tau}) \equiv F(\overline{x}) : \tau) \quad \text{id} \quad (\Theta | \Gamma, \overline{x} : \overline{\tau} \vdash t_i \equiv s_i : \tau_i)_{i \in \{1, \ldots, n\}}
\]

Let \( M : [\sigma_1, \ldots, \sigma_n] \tau \), then we can decompose a constraint with \( M \) on both sides into a set of constraints for each pair of parameters:

\[
(\Theta | \Gamma \vdash M[\overline{s}] : \tau) \quad \text{id} \quad (\Theta | \Gamma \vdash t_i \equiv s_i : \sigma_i)_{i \in \{1, \ldots, n\}}
\]

Example 23.

\[
\begin{align*}
\Theta | \Gamma &= (M : [\sigma] [\sigma] \Rightarrow \sigma, f : \sigma \Rightarrow \sigma) \\
\{\Theta | \Gamma \vdash \text{abs}(x.\text{app}(M[x], x)) \equiv \text{app}(f, x)\} \\
&\text{id}\{\Theta | \Gamma, x : \sigma \vdash \text{app}(M[x], x) \equiv \text{app}(f, x)\} \\
&\quad \text{decompose} \\
&\text{id}\{\Theta | \Gamma, x : \sigma \vdash M[x] \equiv f, \ \Theta | \Gamma, x : \sigma \vdash x \equiv x\} \\
&\quad \text{decompose} \\
&\text{id}\{\Theta | \Gamma, x : \sigma \vdash M[x] \equiv f\} \\
&\quad \text{(delete)}
\end{align*}
\]

The next two rules are second-order versions of imitate and project rules used in many HOU algorithms. The idea is that a metavariable can either imitate the other side of the constraint, or simply project one of its parameters:

Definition 24 (imitate). For constraints with a metavariable \( M : [\overline{\sigma}] \tau \) and an operator \( F : (\overline{\sigma}, \tau_1, \ldots, \overline{\sigma}, \tau_n) \to \tau \) we can imitate the operator side using an imitation binding (metavariables \( \overline{T} \) are fresh):

\[
(\Theta | \Gamma \vdash M[\overline{s}] \equiv F(\overline{\tau}) : \tau) \quad \text{[\text{imitate}]} \quad (\Theta | \Gamma \vdash F(\overline{T}[\overline{s}, \overline{x}]) \equiv F(\overline{T}) : \tau)
\]

Note that (imitate) can be followed up by an application of the (decompose) rule.

Definition 25 (project). For constraints with a metavariable \( M : [\overline{\sigma}] \tau \) and a term \( u : \tau \), if \( \sigma_i = \tau \) then we can produce a JP-style projection binding for the parameter at position \( i \):

\[
(\Theta | \Gamma \vdash M[\overline{s}] \equiv u : \tau) \quad \text{[\text{project}]} \quad (\Theta | \Gamma \vdash s_i \equiv u : \tau)
\]

The next rule is concerned with matching one side of a constraint against one side of an axiom. When matching with an axiom, we need to instantiate it to the particular use (indeed, an axiom serves as a schema!). However, it is not sufficient to simply map metavariables of the axiom into fresh metavariables of corresponding types. Since we are instantiating axiom for a particular constraint which may have a non-empty \( \Gamma \), it is important to add all those variables to each of the fresh metavariables:

Definition 26. Let \( \Gamma = (\overline{\tau} : \overline{\tau}) \) and \( \xi : \Xi \vdash \cdot \rightarrow \Theta | \Gamma \).

We say \( \xi \) instantiates the axiom \( \Xi \) \( \vdash l \equiv r \) in context \( \Theta | \Gamma \) if

1. for any \( (M_i : [\sigma] u) \in \Xi, \xi \) maps \( M_i[l] \) to \( N_i[l, \overline{x}] \);
2. \( N_i = N_j \) iff \( i = j \) for all \( i, j \).

---

5 This is different to \( E \)-unification with first-order axioms, where metavariables do not carry their own context and can be unified with an arbitrary variable later.
Example 27. Let \( \xi = [M[z] \mapsto M_1[z, g, y], N[z] \mapsto N_1[g, y]] \). Then, \( \xi \) instantiates the axiom

\[
M : [\sigma | \tau], N : [\sigma | \tau] \vdash \text{app}(x, M[x]), N[[]) \equiv M[[N[[]) : \tau
\]

in context \( M_1 : [\sigma, \sigma \Rightarrow \tau, \sigma | \tau], N_1 : [\sigma \Rightarrow \tau, \sigma | \sigma \Rightarrow \tau, y : \sigma] \).

Definition 28 (mutate). For constraints where one of the sides matches\(^6\) an axiom in \( E \):

\[
\Xi \mid \vdash l \equiv r : \tau
\]

We rewrite the corresponding side (here, \( \xi \) instantiates the axiom in context \( \Theta | \Gamma_F \)).

\[
(\Theta | \Gamma_F \vdash t \equiv s : \tau) \vdash \{\Theta | \Gamma_F \vdash t \equiv \xi : \tau\} \cup \{\Theta | \Gamma_F \vdash \xi : r \equiv s : \tau\}
\]

In general, we may rewrite in both directions. However, it may be pragmatic to choose a single direction to some of the axioms (e.g. \( \beta\eta \)-reductions), while keeping others bidirectional (e.g. commutativity and associativity axioms). Note that, unlike previous rules, the (mutate) rule can lead to infinite transition sequences.

The remaining rules deal with constraints with metavariables on both sides. One rule attempts to unify distinct metavariables:

Definition 29 (identify). When a constraint consists of a pair of distinct metavariables \( M : [\sigma_1, \ldots, \sigma_k] | \tau \) and \( N : [\gamma_1, \ldots, \gamma_l] | \tau \), we can use an identification binding (metavariables \( i, M, N \) are fresh):

\[
(\Theta | \Gamma_F \vdash M[\theta] \equiv N[\tau]) \vdash \{[\theta[1], \ldots, \theta[l]] \vdash \theta[1] \equiv N[\theta[l]]\}
\]

Another rule attempts to unify identical metavariables with distinct lists of parameters:

Definition 30 (eliminate). When a constraint has the same metavariable \( M : [\sigma_1, \ldots, \sigma_n] | \tau \) on both sides and there is a sequence \( (j_k)_{k=1}^n \) such that \( s_{j_k} = t_{j_k} \) for all \( k \in \{1, \ldots, n\} \), then we can eliminate every other parameter and leave the remaining terms identical (metavariables \( E \) is fresh):

\[
(\Theta | \Gamma_F \vdash M[\theta] \equiv M[\tau]) \vdash [\theta[1] \mapsto [z_{j_1}, \ldots, z_{j_n}]] \rightarrow \emptyset
\]

The idea of the final rule is to extend a list of parameters with some combination of those that exist already. For example, consider constraint \( \forall x, y, z. M[pair(x, y), z] \equiv N[x, z] \). It is clear, that if we can work with a pair of \( x \) and \( y \), then we can work with them individually, since we can extract \( x \) using \( \text{fst} \) and \( y \) using \( \text{snd} \). Thus, a substitution \( [y[p, z] \mapsto M_1[p, z, \text{fst}(p)]] \) would result in a new constraint \( \forall x, y, z. M_1[pair(x, y), z, \text{fst}(pair(x, y))] \equiv N[x, z] \). This one can now be solved by applying (identify), (eliminate), and (decompose) rules that will lead us to \( \forall x, y, z. \text{fst}(pair(x, y)) \equiv : \sigma \) which will be processed using (mutate) rule.

Definition 31 (iterate). When a constraint consists of a pair of (possibly, identical) metavariables \( M : [\sigma_1, \ldots, \sigma_k] | \tau \) and \( N : [\gamma_1, \ldots, \gamma_l] | \tau \), we can use an iteration binding (metavariables \( H, K \) are fresh):

\[
(\Theta | \Gamma_F \vdash M[\theta] \equiv N[\tau]) \vdash \{\Theta | \Gamma_F \vdash H[\theta, F(\theta)] \equiv N[\theta]\}
\]

We check that the roots of terms match. Technically, we do not have to perform this check and apply (mutate) rule for any axiom (non-deterministically), since full matching will be performed by the unification procedure.
The following example demonstrates the importance of iteration by an arbitrary operator to introduce variables into scope:

**Example 32.** Consider a unification problem for simply-typed λ-calculus:

\[ Ξ : [σ ⇒ σ ⇒ τ](σ ⇒ τ) \]

\[ ∀ f : σ ⇒ σ ⇒ σ ⇒ τ, \]

\[ M[λx.λy.f x y x x] \vdash M[λx.λy.f y y y y] : σ ⇒ τ \]

It has the following E-unifier: \( ζ = [M[g] \mapsto λz.g z z] \). To construct this unifier from bindings, we start with iteration binding \([M[g] \mapsto [g, λz.M[1, [g, z]]]]\), introducing the lambda abstraction, which is followed by a projection \([λ[g, r] ⇒ r] \), which is followed by another iteration (to introduce application), and so on.

Finally, we compile all transition rules into the unification procedure:

**Definition 33.** The E-unification procedure over an equational presentation \( E \) is defined by repeatedly applying the following transitions (non-deterministically) until a stop:

1. If no constraints are left, then stop (**succeed**).
2. If possible, apply (**delete**) rule.
3. If possible, apply (**mutate**) or (**decompose**) rule (**non-det.**).
4. If there is a constraint consisting of two non-metavariabes and none of the above transitions apply, stop (**fail**).
5. If there is a constraint \( M[\vdash F(\ldots)] \), apply (**imitate**) or (**project**) rules (**non-det.**).
6. If there is a constraint \( M[\vdash x] \), apply (**project**) rules (**non-det.**).
7. If possible, apply (**identify**), (**eliminate**), or (**iterate**) rules (**non-det.**).
8. If none of the rules above are applicable, then stop (**fail**).

Many HOU algorithms [23, 21] implement a rule (typically called **eliminate**) that allows to eliminate metavariables, when a corresponding constraint is in solved form. Such a rule is not necessary here, as it is covered by a combination of (**imitate**), (**decompose**), (**delete**), (**identify**), and (**eliminate**) rules. However, it simplifies presentation of examples and also serves as a practical optimisation, so we include it as an optional rule:

**Definition 34 (eliminate**\(^*\)).** When a constraint \( C = (Θ | Γ, V ⊢ M[\vdash u]) \) is in solved form, we can eliminate it with a corresponding unifier \( ξ(C) = [M[\vdash u] ⇒ u] \):

\[ (Θ | Γ, V ⊢ M[\vdash u] ⇒ u) \xrightarrow{[u[τ]⇒u]} Ξ \]

The (**eliminate**\(^*\)) rule should have the same priority as (**delete**) in the procedure.

**Lemma 35.** In the procedure defined in Definition 33, each step is sound. That is, if \( S \xrightarrow{θ} S′ \) is a single-step transition that the procedure takes and \( ξ ∈ U_E(S′) \) then \( ξ \circ θ ∈ U_E(S) \).

**Proof.** It is sufficient to show that each step is sound with respect to the constraint it acts upon. That is, we consider the step \( \{C\} \xrightarrow{θ} S'' \) such that \( C ∈ S \) and \( S'' ⊆ S' \). By assumption \( ξ ∈ U_E(S') \) and thus also \( ξ ∈ U_E(S'') \). Note that for any constraint \( D ∈ (S′ −\{C\}) \) we have a corresponding constraint \( D' ∈ (S' − S'') \) such that \( D = θD' \). Since \( ξ \) unifies \( D' \) it follows that \( ξ \circ θ \) unifies \( D \). Thus, it is enough for us to show that \( ξ \circ θ \) unifies \( U_E(\{C\}) \).

We now go over the list of possible steps:
10:14  E-Unification for Second-Order Abstract Syntax

- **(delete):** it is clear that any substitution unifies $C$;
- **(decompose):** since $\xi$ unifies all subterm pairs in $S''$, it also unifies $C$;
- **(imitate), (project), (identify), (eliminate), (iterate):** all of these rules simply make a decision on how to substitute some metavariables (choose $\theta$) and immediately apply that substitution. So, $S'' = \{\theta C\}$ and since $\xi$ unifies $S'$ then $\xi \circ \theta$ unifies $S$.
- **(mutate):** let $C = (\Theta | \Gamma \vdash s = t : \tau)$ and we mutate according to axiom $(\Xi | \vdash l \equiv r : \tau) \in E$ with substitution $\zeta$ instantiating this axiom. By assumption, $\xi$ unifies both $s \equiv \zeta l$ and $\zeta r \equiv t$. Also, $\Theta | \Gamma \vdash l \equiv E \zeta r : \tau$. In this rule, $\theta = id$, and so we can show that $\xi \circ \theta = \xi$ unifies $s \equiv t$. $\xi s \equiv E \xi (\zeta l) \equiv E \xi (\zeta r) \equiv E \xi t$  

\[ \triangleright \]

**Theorem 36.** The procedure defined in Definition 33 is sound. That is, if $S \xrightarrow{\theta_1} S_1 \xrightarrow{\theta_2} \ldots \xrightarrow{\theta_n} \emptyset$ is a path produced by the procedure, then $\theta_1 \circ \theta_2 \circ \ldots \circ \theta_n \in U_E(S)$.

**Proof.** Direct corollary of Lemma 35.  

\[ \triangleright \]

## 5 Proof of Completeness

In this section we prove our main theorem, showing that our unification procedure is complete.

We start with a definition of mixed operators:

\[ \triangleright \]

**Definition 37.** We say that an operator $F : (\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n) \to \gamma$ is mixed iff $\alpha_i$ is empty and $\alpha_j$ is not empty for some $i$ and $j$.

Dealing with mixed operators can be very non-trivial. In the following theorem, we assume that all operators either introduce scopes in all subterms, or in none. That is, for each operator $F : (\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n) \to \gamma$, either $|\alpha_i| = 0$ for all $i$ or $|\alpha_i| > 0$ for all $i$. The assumption is justified since we can always encode a mixed operator as a combination of non-mixed operators. For example, $let(t_1, x, t_2)$ can be encoded as $let(t_1, block(x, t_2))$.

\[ \triangleright \]

**Theorem 38.** Assuming no mixed operators are used, the procedure described in Definition 33 is complete, meaning that all paths from a root to all (success) leaves in the search tree constructed by the procedure, form a complete (but not necessarily minimal) set of $E$-unifiers. More specifically, let $E$ be an equational presentation and $(\Theta, S)$ be an $E$-unification problem. Then for any $E$-unifier $\theta \in U_E(S)$ there exists a path $S \xrightarrow{\xi} \emptyset$ such that $\xi \preceq_E \theta$.

\[ \triangleright \]

**Remark 39.** The unification procedure may produce redundant unifiers. For example, consider the following unification problem in simply-typed $\lambda$-calculus:

\[ \exists M : [\sigma, \sigma] \tau, N : [\sigma] (\sigma \Rightarrow \tau). \forall x : \sigma. M[x, x] \rightarrow app(N[x], x) \]

Depending on whether we start with the (imitate) rule or the (mutate) first, we can arrive at the following unifiers:

\[ \theta_1 = [M[z_1, z_2] \mapsto app(N[z_1], z_2)] \]
\[ \theta_2 = [M[z_1, z_2] \mapsto t[z_2, z_1], N[z_1] \mapsto abs(z, t[z_1, z])] \]

It is clear that $\theta_1 \neq \theta_2$, but $\theta_1 \preceq_E \theta_2$ (witnessed by $[N[z_1] \mapsto abs(z, t[z_1, z])]$). Hence, the set of $E$-unifiers produced for this unification problem is not minimal (by Definition 14).

Our proof is essentially a combination of the two approaches: one by Gallier and Snyder in their proof of completeness for general (first-order) $E$-unification [11], and another one by Jensen and Pietrzykowski (JP) [16], refined by Vukmirovic, Bentkamp, and Nummelin.
(VBN) [34] for full higher-order unification. In particular, we need to reuse some of the ideas from the latter when dealing with parametrised metavariables. However, we cannot reuse the idea of JP’s \( \omega \)-simplicity or VBN’s base-simplicity, as those are dependent crucially on the \( \eta \)-long terms, \( \lambda \)-abstraction, function application, which are not accessible to us in a general second-order abstract syntax. Instead, we re-use the ideas of Gallier and Snyder to understand when it is okay to decompose terms. To understand when to apply (iterate) rule, we also look at the rewrite sequence instead of \( \omega \)-simplicity of terms.

The main idea of the proof is to take the unification problem \( S \) together with its \( E \)-unifier \( \theta \) and then choose one of the rules of the procedure guided by \( \theta \). Applying a rule updates constraints and the remaining substitution is also updated. To show that this process terminates, we introduce a measure that strictly decreases with each rule application.

**Definition 40.** Let \( \theta \in U_E(S) \). Then, define the measure on pairs \( \langle S, \theta \rangle \) as the lexicographic comparison of

1. sum of lengths of the rewriting sequences \( \theta s \Leftarrow E \theta t \) for all \( s \subseteq t \) of \( S \);
2. total number of operators used in \( \theta \);
3. total number of metavariables used in \( \theta \);
4. sum of sizes of terms in \( S \).

We denote the quadruple above as \( \text{ord}(S, \theta) \).

The following definition helps us understand when we should apply the (project) rule:

**Definition 41.** A metavariable \( M : [\overline{\sigma}]r \) is projective at \( j \) relative to \( \theta \) if \( \theta M[\overline{\tau}] = z_j \).

One of the crucial points in the proof is to understand whether we can apply (identify) or (eliminate) rules for constraints with two metavariables on both sides. The following lemma provides precise conditions for this, allowing for (identify) when metavariables are distinct or (eliminate) when they are equal.

**Lemma 42.** Let \( s = M[\overline{\tau}] \) and \( t = N[\overline{\tau}] \) such that \( \zeta s \Leftarrow E \zeta t \). Let \( s_1, \ldots, s_n \) be the subterms of \( \zeta s \), \( t_1, \ldots, t_n \) the subterms of \( \zeta t \) such that the rewriting sequence \( \zeta s \Leftarrow E \zeta t \) corresponds to the union of independent rewritings \( s_i \Leftarrow E t_i \) for all \( i \in \{1, \ldots, n\} \). If for all \( i \) we have either that \( s_i \) is a subterm of an occurrence of \( \zeta u_j \), or that \( t_i \) is a subterm of an occurrence of \( \zeta v_j \), then there exist terms \( w, \overline{w}, \overline{v} \) such that

1. \( \overline{w}[\overline{y} \mapsto \overline{v}] \equiv_E \overline{\pi} \) and \( \overline{\Sigma} \overline{y} \mapsto \overline{v} \equiv_E \overline{\tau} \);
2. \( \zeta M[\overline{\tau}] = w[\overline{\pi} \mapsto \overline{w}] \) and \( \zeta N[\overline{\tau}] = w[\overline{\tau} \mapsto \overline{w}] \).

**Proof.** We define an auxiliary family of terms \( \Theta : \overline{\tau} : \overline{\pi} : \overline{y} : \overline{v} \mapsto : \overline{\tau} : \overline{r} : \overline{\sigma} \) for pairs of terms \( \Theta = \overline{\sigma} : \overline{\tau} : \overline{\pi} : \overline{y} : \overline{v} \mapsto = \overline{\tau} : \overline{r} : \overline{\sigma} \) such that \( \overline{r} \) is a subterm of \( \zeta M[\overline{\tau}] \) and \( \overline{r} \) is a subterm of \( \zeta N[\overline{\tau}] \) satisfying \( l[\overline{\pi} \mapsto \overline{\tau}] \equiv_E r[\overline{\tau} \mapsto \overline{\pi}] \). We define \( w'(l, r) \) inductively on the structure of \( l \) and \( r \), maintaining \( l \equiv_E w'(l, r)[\overline{\tau} \mapsto \overline{v}] \) and \( r \equiv_E w'(l, r)[\overline{\tau} \mapsto \overline{v}] \):

1. if \( l = x_i \), or \( r = x_i \), then \( l = r \) and \( w'(l, r) = x_i \);
2. if \( l = z_k \), then \( w'(l, r) = z_k \) and \( \eta_k' = r \);
3. if \( r = y_k \), then \( w'(l, r) = y_k \) and \( \eta_k' = l \);
4. if \( l = F(\overline{\pi}, \overline{p}) \) and \( r = F(\overline{\pi}, \overline{q}) \) then \( w'(l, r) = F(\overline{\pi}, w'(p, q)) \); note that \( w'(p_i, q_i) \) is well-defined for all \( i \), since \( l[\overline{\pi} \mapsto \overline{\tau}] \equiv_E r[\overline{\tau} \mapsto \overline{\pi}] \) implies component-wise equality \( p_i[\overline{\pi} \mapsto \overline{\tau}] \equiv_E q_i[\overline{\tau} \mapsto \overline{\pi}] \) for all \( i \). This is true, since otherwise we are rewriting (at root) both \( l \) and \( r \), but \( l[\overline{\pi} \mapsto \overline{\tau}] \) or \( r[\overline{\tau} \mapsto \overline{\pi}] \) corresponds to a parameter occurrence \( \zeta u_j \) or \( \zeta v_j \) in terms \( s \) or \( t \) correspondingly.
5. if \( l = M[\overline{\pi}] \) and \( r = M[\overline{\tau}] \) then \( w'(l, r) = M[w'(p, q)] \); here, \( w'(p_i, q_i) \) is well-defined for all \( i \), similarly to the previous case.
If \( w_k \) (or \( v_k \)) has not been defined for some \( k \), it means that a corresponding parameter is not essential and can be eliminated. We set\(^7\) such \( u_k \) to be a fresh metavariable \( \xi_k \). We set \( w = w'([\xi t], [\zeta N]) \). By construction, \( \zeta t = w' \rightarrow v' \) and \( \zeta N = w \rightarrow v' \).

**Corollary 43.** Let \( s = M[\pi] \) and \( t = M[\pi] \) such that \( \zeta s \leftrightarrow \zeta t \). Let \( s_1, \ldots, s_n \) be the subterms of \( s \), \( t_1, \ldots, t_n \) the subterms of \( t \) such that the rewriting sequence \( \zeta s \leftrightarrow \zeta t \) corresponds to the union of independent rewritings \( s_i \leftrightarrow t_i \) for all \( i \in \{1, \ldots, n\} \). If for all \( i \) we either have that \( s_i \) is a subterm of an occurrence of \( \zeta u_j \), or that \( t_i \) is a subterm of an occurrence of \( \zeta v_j \), then, there exists a sequence \( 1 \leq j_1 < \ldots < j_k \leq n \) such that and \( FV(\zeta M[\pi]) = \{z_{j_1}, \ldots, z_{j_k}\} \) and \( u_{j_i} \equiv \mathbf{E} v_{j_i} \) for all \( i \).

The following lemma will help us generalize solutions in Item 2(e)iii of the proof below.

**Lemma 44.** Let \( \Xi | \mathfrak{r} : \mathfrak{a} \vdash w : \sigma \) be a subterm of \( \Xi | \mathfrak{r} : \mathfrak{a} \vdash t : \tau \). If \( t \) does not contain mixed operators, then there exists a substitution \( \zeta w,t = [h[z] \mapsto \epsilon] \) and a collection of terms \( \Xi | \mathfrak{r} : \mathfrak{a} \vdash \zeta w,t : \tau \), such that each \( s_i \) is a subterm of \( t \) and \( \zeta w,t h[w,\epsilon] = t \).

**Proof.** Note that \( w \) and \( \mathfrak{a} \) are subterms of \( t \) and are not under binders (since they have the same variable context). Then, by induction on the structure of \( t \):
1. If \( t = w \), then \( \zeta w,t = [h[z] \mapsto \epsilon] \);
2. If \( t = F(\mathfrak{r} t_1, \ldots, \mathfrak{r} t_n) \) then, since \( t \) does not contain mixed operators, \( \mathfrak{r} t \) is empty for all \( i \). Now, if \( w \) is a subterm of \( t_i \) and \( \zeta w,t_i = [h, z, y_1, \ldots, y_k] \mapsto h \) then \( \zeta = [h[z, y_1, \ldots, y_k]] \mapsto F(y_1, \ldots, y_{i-1}, h, y_{i+1}, \ldots, y_k) \).
3. If \( t = N[t_1, \ldots, t_n] \) such that \( w \) is a subterm of \( t_i \) and \( \zeta w,t_i = [h, z, y_1, \ldots, y_k] \mapsto h \) then \( \zeta = [h[z, y_1, \ldots, y_k]] \mapsto N(y_1, \ldots, y_{i-1}, h, y_{i+1}, \ldots, y_k) \).

Note that case of \( t \equiv x \) is impossible unless \( t = w \) (case 1).

We are now ready to prove Theorem 38.

**Proof of Theorem 38.** Let \( S_0 = S \) and \( \theta_0 = \rho \circ \theta \), where \( \rho \) is some renaming substitution such that every metavariable occurring in \( S_0 \) does not occur in \( S_0 \). Note that \( \theta_0 \) is an \( E \)-unifier of \( S \), since \( \theta \) is by assumption.

We now inductively define \( S_i, \xi_i, \) and \( \theta_i \) until we reach some \( i \) such that \( S_i = \varnothing \). We ensure that \( \text{ord}(S_i, \theta_i) \) decreases with every step, so that such sequence of steps would always terminate. We maintain the following invariants for each step:
1. \( \langle S_i, \theta_i \rangle \xrightarrow{\xi_i} \langle S_{i+1}, \theta_{i+1} \rangle \) where \( S_i \xrightarrow{\xi_i} S_{i+1} \) by some rule of the unification procedure;
2. \( \text{ord}(S_{i+1}, \theta_{i+1}) < \text{ord}(S_i, \theta_i) \);
3. \( \theta_i \in \text{U}_E(S_i) \);
4. \( \theta_{i_0} \equiv \mathbf{E} \circ \xi_{i_0} \circ \ldots \circ \xi_0 \);
5. every free variable occurring in \( \theta_i S_i \) does not occur in \( S_i \);

If \( S_i \neq \varnothing \), then let \( \forall \mathfrak{r} : \mathfrak{a} \vdash t : \tau \) be a constraint in \( S_i \). We consider two major cases with respect to the rewriting sequence \( \theta_i \vdash S_i \) from \( E \) such that \( \zeta = \mathbf{E} \).

1. **The rewriting sequence contains a root rewrite.** More precisely, there exists a sequence \( u_0 \mapsto u_1 \mapsto u_n \) such that for some \( u_j \) such that \( u_j \mapsto u_{j+1} \) is a direct application of a rewrite using an axiom. This means that \( s \) and \( t \) can be unified by a direct use of an axiom. More specifically, there exists an instantiation \( \zeta \) of an axiom \( \Xi | \mathfrak{r} \vdash \theta_i \) from \( E \) such that \( \zeta \equiv \mathbf{E} \).

---

\(^7\) alternatively, we could have adjusted the statement to only mention subsets of variables \( \mathfrak{s} \) and \( \mathfrak{y} \) that are used at least once.
\[ s \overset{\gamma}{\rightarrow} u_j \text{ and } u_{j+1} \overset{\gamma}{\rightarrow} t. \] Thus, we can apply \textbf{(mutate)} rule. The measure decreases since the rewrite sequence \( s \overset{\gamma}{\rightarrow}_E t \) is now split into two sequences \( s \overset{\gamma}{\rightarrow} u_j \) and \( u_{j+1} \overset{\gamma}{\rightarrow}_E t \) such that sum of lengths of new sequences is exactly one less than the length of the original sequence.

2. Rewriting sequence is empty or does not contain a root rewrite. This means that rewrites may only happen in subterms.

a. If \( s = x \) and \( t = y \) where \( x \) and \( y \) are variables, then \( x = y \) and we can apply \textbf{(delete)} rule, with \( \xi_i = \text{id} \) and \( \theta_{i+1} = \theta_i \). The measure is reduced since the total size of constraints is reduced, while the rewriting sequences and the remaining substitution remain the same.

b. If \( s = F(\overline{x}u) \) and \( t = F(\overline{x}v) \), then \( \theta_i s = F(\overline{x}\theta_i u) \) and \( \theta_i t = F(\overline{x}\theta_i v) \). Since there are no root rewrites, \( \theta_i \) unifies each pair \( u_j = v_j \) in corresponding extended contexts, so we can apply \textbf{(decompose)} rule with \( \xi_i = \text{id} \) and \( \theta_{i+1} = \theta_i \). Note that the chain of rewrites may be split into several chains, but the total sum of lengths remains the same. Second component of the measure also remains unchanged. We reduce the third component of the measure, since the total size of terms in the unification problem decreases, the sum of chains of rewrites is unchanged.

c. If \( s = M[\overline{m}] \) and \( M \) is projective at \( j \) relative to \( \theta_i \) then we can apply \textbf{(project)} rule with \( \xi_i = [M[\overline{m}] \rightarrow z_j] \). Note that the chain of rewrites remains unchanged and \( \xi_i \) does not take any operators away from \( \theta_{i+1} \) (which is a restriction of \( \theta_i \) to metavariables other than \( m \)). We reduce the measure by reducing the total size of terms in the unification problem.

d. If \( s = M[\overline{m}] \) where \( M \) is not projective relative to \( \theta_i \) and \( \theta_i s = F(\overline{x}u) \) and \( t = F(\overline{x}v) \), then \( \theta_i \) unifies each pair \( u_i = v_i \) in corresponding extended contexts and we can apply \textbf{(imitate)} rule with \( \xi_i = [M[\overline{m}] \rightarrow F(\overline{x}T[\overline{x}, \overline{m}])] \). Let \( \theta_i M[\overline{m}] = F(\overline{x}w) \), then \( \theta_{i+1} \) is constructed from \( \theta_i \) by removing mapping for \( M \) and adding mappings \([T_j[\overline{x}, \overline{m}] \rightarrow w_j]\) for all \( j \). The chain of rewrites is unchanged and the measure decreases since we reduce the number of operators in \( \theta_{i+1} \).

e. If \( s = M[\overline{m}] \) where \( M \) is not projective relative to \( \theta_i \) and \( \theta_i s \overset{\gamma}{\rightarrow}_E \theta_i t \) contains a rewrite of a subterm \( w \) in \( \theta_i s \) that is not a subterm of an occurrence of \( \theta_i u_i \) for some \( i \), then

i. If \( w \) is under a binder in \( \theta_i s \), we take the outermost operator \( F \) that binds a variable captured by \( w \) (that is, \( \theta_i s = \ldots F(\overline{m}_1, \overline{m}_2, \ldots, \overline{m}_n) \)) and apply \textbf{(iterate)} rule with \( \xi_i = [M[\overline{m}] \rightarrow M'[\overline{m}, F(\overline{m}_1, M_1[\overline{m}, \overline{m}], \ldots, M_n[\overline{m}, \overline{m}])]] \). Let \( \theta_i M[\overline{m}] = F(\overline{x}s') \), then \( \theta_{i+1} \) is defined as \( \theta_i \) with mapping of \( M \) removed and added mappings \([T_j[\overline{x}, \overline{m}] \rightarrow s_j]\) \( \in \{1, \ldots, n\} \). The chain of rewrites remains unchanged and the number of operators in \( \theta_{i+1} \) decreases by one, so the measure decreases.

ii. If \( w = F(\ldots) \) and is not under a binder, then we apply \textbf{(iterate)} rule with \( \xi_i = [M[\overline{m}] \rightarrow M'[\overline{m}, F(\overline{m}_1, M_1[\overline{m}, \overline{m}], \ldots, M_n[\overline{m}, \overline{m}])]] \). We define \( \theta_{i+1} \) and show that the measure decreases analogously to the previous case.

iii. If \( w = W[\overline{m}] \) and is not under a binder, then \( \theta_i M[\overline{m}] \) contains \( w' = W'[\overline{m}] \) as a subterm and \( v'_i[\overline{x} \mapsto \overline{m}] = v_i \) for all \( i \) (this is because \( w \) is not a subterm of any of the \( \theta_i u_i \)). Since \( w' \) is also not under binder, then by Lemma 44 and assumption of no mixed operators we have that there exist terms \( h, \overline{x} \), and a substitution \( \zeta = [W[\overline{m}, \overline{y}] \rightarrow h] \) such that \( \zeta h[\overline{m}, \overline{y}] = \theta_i M[\overline{m}] \). Set \( \theta'_i = [M[\overline{m}] \rightarrow h[\overline{m}, \overline{y}] \rightarrow \overline{m}] \). We have \( \theta_i = \zeta \circ \theta'_i \), that is \( \theta'_i \) is more general that \( \theta_i \) modulo \( E \). The rewriting sequence remains unchanged. If \( \theta_i s \) has an operator at root, then \( \theta'_i \) has fewer operators which decreases the measure. If \( \theta_i s \) has a metavariable at root, then \( \theta'_i \) has fewer metavariables which decreases the measure.
A pragmatic implementation of our procedure may enjoy the following changes. We find that these help make a reasonable compromise between completeness and performance:

1. remove (iterate) rule; this rule sacrifices completeness, but helps significantly reduce nondeterminism; the solutions lost are also often highly non-trivial and might be unwanted in certain applications such as type inference;

2. implement (eliminate*) rule;

3. split axioms $E = B \cup R$ such that $R$ constitutes a confluent and terminating term rewriting system, and introduce (normalize) rule to normalize terms (lazily) before applying any other rules except (delete) and (eliminate*);

4. introduce a limit on a number of applications of (mutate) rule;

5. introduce a limit on a number of bindings that do not decrease problem size;

6. introduce a limit on total number of bindings.

We now have a sequence $\langle S_0, \theta_0 \rangle \xrightarrow{\xi_0} \langle S_1, \theta_1 \rangle \xrightarrow{\xi_1} \ldots$. The sequence is finite since the measure $\text{ord}(S_i, \theta_i)$ strictly decreases with every step. Therefore, $\langle S, \theta \rangle \xrightarrow{\xi_0} \ldots \xrightarrow{\xi_n} \langle \emptyset, \text{id} \rangle$ and $\theta \equiv E \rho^{-1} \circ \theta_1 \circ \xi_{n-1} \circ \ldots \circ \xi_0 \equiv E \rho^{-1} \circ \xi_n \circ \ldots \circ \xi_0 \equiv E \xi_n \circ \ldots \circ \xi_0$, completing the proof. 

6 Discussion

A pragmatic implementation of our procedure may enjoy the following changes. We find that these help make a reasonable compromise between completeness and performance:

1. remove (iterate) rule; this rule sacrifices completeness, but helps significantly reduce nondeterminism; the solutions lost are also often highly non-trivial and might be unwanted in certain applications such as type inference;

2. implement (eliminate*) rule;

3. split axioms $E = B \cup R$ such that $R$ constitutes a confluent and terminating term rewriting system, and introduce (normalize) rule to normalize terms (lazily) before applying any other rules except (delete) and (eliminate*);

4. introduce a limit on a number of applications of (mutate) rule;

5. introduce a limit on a number of bindings that do not decrease problem size;

6. introduce a limit on total number of bindings.
When adapting ideas from classical \( E \)-unification and HOU, some technical difficulties arise from having binders lacking (in general) the nice syntactic properties of \( \lambda \)-calculus. These difficulties affect both the design of our unification procedure, leading to some simplifications, and the completeness proof, requiring us to find a different approach to define the measure and consider cases that do not have analogues.

In the procedure, we had to simplify whenever those ideas relied on normalisation, \( \eta \)-expansion, or specific syntax of \( \lambda \)-terms. Many HOU algorithms look at syntactic properties of terms to determine which rules to apply. In particular, HOU algorithms often distinguish flex and rigid terms \[15, 23\]. Jensen and Pietrzykowski introduce a notion of \( \omega \)-simple terms \[16\]. Vukmirovic, Bentkamp, and Nummelin \[34\] introduce notions of base-simple and solid terms. These properties crucially depend on specific normalisation properties of \( \lambda \)-calculus, which might be inaccessible in an arbitrary second-order equational theory. Thus, our procedure contains more non-determinism than might be necessary.

One notable example of such simplication is in the imitation and projection bindings. In HOU algorithms, it is common to have substitutions of the form

\[
[ M \mapsto \lambda x_1, \ldots, x_n. f (h_1 x_1 \ldots x_n) \ldots (h_k x_1 \ldots x_n)]
\]

where \( f \) can be a bound variable (one of \( x_1, \ldots, x_n \)) or a constant of type \( \sigma_1 \Rightarrow \cdots \Rightarrow \sigma_k \Rightarrow \tau \). These are called Huet-style projection or imitation bindings \[16, 34\] or partial bindings \[15, 23\]. Huet-style projections (and conditions prompting their use) are non-trivial to generalise well to arbitrary second-order abstract syntax, so we skipped them in this paper, opting out for simpler rules but larger search space.

In the completeness proof for HOU algorithms, the syntactic properties of \( \lambda \)-calculus are heavily exploited. Their inaccessibility in a general second-order equational theory has contributed to some difficulties when developing the proof of completeness in Theorem 38. Perhaps, the most challenging of all was handling of the Item 2(e)iii of the proof which requires the assumption of no mixed operators and Lemma 44. These do not appear to have an analogue in completeness proofs for HOU or first-order \( E \)-unification.

## 7 Conclusion and Future Work

We have formulated the equational unification problem for second-order abstract syntax, allowing to reason naturally about unification of terms in languages with binding constructions. Such languages include, but are not limited to higher-order systems such as \( \lambda \)-calculus, which expands potential applications to more languages. We also presented a procedure to solve such problems and our main result shows completeness of this procedure.

In future work, we will focus on optimisations and recognition of decidable fragments of \( E \)-unification over second-order equations.

One notable optimisation is splitting \( E \) into two sets \( R \uplus B \), where \( R \) is a set of directed equations, forming a confluent second-order term rewriting system, and \( B \) is a set of undirected equations (such as associativity and commutativity axioms).

Another potential optimisation stems from a generalisation of Huet-style binding (also known as partial binding), which can lead to more informed decisions on which rule to apply in the procedure, introduce Huet-style version of (project) and improve (iterate) rule, significantly reducing the search space. A version of such an optimisation has been implemented in a form of a heuristic to combine (imitate) and (project) rules by Kudasov \[20\].

There are several well-studied fragments both for \( E \)-unification and higher-order unification. For example, unification in monoidal theories is essentially solving linear equations over semirings \[26\]. In higher-order unification, there are several well-known decidable fragments
such as pattern unification [23]. Vukmirovic, Bentkamp, and Nummelin have identified some of the practically important decidable fragments as well as a new one in their recent work on efficient full higher-order unification [34]. It is interesting to see if these fragments could be generalised to second-order abstract syntax and used as oracles, possibly yielding an efficient \(E\)-unification for second-order abstract syntax as a strict generalisation of their procedure.


