A Saturation-Based Unification Algorithm for Higher-Order Rational Patterns

ZHIBO CHEN and FRANK PFENNING, Carnegie Mellon University, Pittsburgh, PA, USA

Higher-order unification has been shown to be undecidable. Miller discovered the pattern fragment and subsequently showed that higher-order pattern unification is decidable and has most general unifiers. We extend the algorithm to higher-order rational terms (a.k.a. regular Böhm trees, a form of cyclic λ-terms) and show that pattern unification on higher-order rational terms is decidable and has most general unifiers. We prove the soundness and completeness of the algorithm.

CCS Concepts: • **Theory of computation** → **Proof theory**; **Equational logic and rewriting**;

Additional Key Words and Phrases: Pattern Unification, Rational Terms, Regular Böhm Trees

ACM Reference format:
Zhibo Chen and Frank Pfenning. 2024. A Saturation-Based Unification Algorithm for Higher-Order Rational Patterns. *ACM Trans. Comput. Logic* 26, 1, Article 4 (December 2024), 33 pages.
https://doi.org/10.1145/3704265

1 Introduction

Unification is the backbone of logic programming [Miller, 1991] and is also used in type reconstruction in the implementation and coverage checking of dependent type theories [Pfenning and Schürmann, 1999; Schürmann and Pfenning, 2003]. Given a list of equations with unification metavariables, unification is the problem of finding substitutions for the unification metavariables such that the equations hold true. Often, one is interested in finding most general unifiers. Consider the following equation as an example, where unification metavariables are written in capital letters:

$$\lambda x.\, \lambda y.\, \lambda z.\, F\,x\,y \doteq \lambda x.\, \lambda y.\, \lambda z.\, G\,y\,z.$$

The most general unifier in this case is $F = \lambda v.\, \lambda w.\, H\,v$ and $G = \lambda u.\, \lambda v.\, H\,v$ for some fresh unification metavariable H.

The attempt to develop a formal representation for circular proof systems [Brotherston and Simpson, 2011; Fortier and Santocanale, 2013] has led to the development of CoLF [Chen and Pfenning, 2023], a logical framework with higher-order rational (circular) terms. Type checking and reconstruction in CoLF involves unification on inherently circular terms. We also foresee applications of our unification algorithms in the context of cyclic logic and process calculi [Derakhshan and Pfenning, 2019], for example, to implement transformations of recursive (cyclic) processes or to perform type inference in the presence of recursive (cyclic) types.

Authors' Contact Information: Zhibo Chen (corresponding author), Carnegie Mellon University, Pittsburgh, PA, USA; e-mail: zhiboc@andrew.cmu.edu; Frank Pfenning, Carnegie Mellon University, Pittsburgh, PA, USA; e-mail: fp@cs.cmu.edu.

ACM 1557-945X/2024/12-ART4
https://doi.org/10.1145/3704265

In this article, we provide a unification algorithm on higher-order rational terms in the sense of the type theory CoLF [Chen and Pfenning, 2023], where two λ-terms are equal if their infinite unfoldings as rational trees are equal. The higher-order rational terms we are considering are also called $\bot$-free regular Böhm trees by Huet [1998]. Our work is distinguished from recent works on nominal unification in λ-calculus with recursive let [Schmidt-Schauß et al., 2022] (a.k.a. cyclic λ-calculus), in that the notion of equality in their work is much weaker than ours. Their equality is based on alpha-equivalence and permutation of order of declaration within the recursive let construct, but our equality is based on the infinite tree equalities generated from circular terms. For instance, given two recursive definitions $\underline{r} =_d c\,\underline{r}$ and $\underline{s} =_d c\,(c\,\underline{s})$, our algorithm considers $\underline{r}$ and $\underline{s}$ to be equal, whereas the algorithm by Schmidt-Schauß et al. distinguishes these two terms.

We only consider the case of unification problems between simply typed higher-order rational terms, and in particular, we treat validity as a separate issue and thus do not distinguish between type and cotype [Chen and Pfenning, 2023]. For instance, when encountering the unification problem $F = \mathsf{succ}\ F$, supposedly with the type of natural numbers, our algorithm is happy to come up with the solution F being an infinite stack of succ's, and disregards the fact the circular terms $F = \underline{r}$, $\underline{r} =_d \mathsf{succ}\ \underline{r}$ are not valid. In an implementation, validity checking can be a separate procedure from unification. The approach is to build validity checking into unification, such that $F = \mathsf{succ}\ F$ of natural number type has no unifier due to the failure of the occurs check, because validity requires every term of natural number type to be a finite term. We did not take the combined approach as we find separating two issues leads to a cleaner presentation of the unification algorithm.

One of the simplest examples of higher-order unification on higher-order rational terms is the following one, where F is of simple type $(* \to *) \to *$:

$$\lambda x.\, x\,(F\,x) \doteq \lambda x.\, F\,x.$$

If we were to consider this problem in the setting of noncyclic unification, there would be no unifier due to the failure of the occurs check. However, our cyclic-unification algorithm will successfully find the unifier $F = \lambda x.\, \underline{r}\,x$ where $\underline{r} =_d \lambda x.\, x\,(\underline{r}\,x)$. The symbol $\underline{r}$ is a recursion constant that unfolds to the infinitary term, $\lambda x.\, x\,(x\,(x\,(x\,\ldots)))$, an abstraction that binds x and its body is an infinite stack of x's.

We first present the algorithm for first-order rational unification in a new way (Section 2) and then extend the algorithm to include higher-order patterns (Section 3). In each case, we first define the unification problem (Sections 2.1 and 3.1) and give a preprocessing algorithm that transforms an arbitrary unification problem into a flattened form (Sections 2.2 and 3.2). Then, we give a saturation-based algorithm that operates on the flattened form (Sections 2.4 and 3.4). The saturation rules are complemented with examples showing how the algorithm operates on concrete problems. Finally, a proof of the correctness of the algorithm is given (Sections 2.6 and 3.7). Examples use the syntax of Twelf [Pfenning and Schürmann, 1999] extended with cyclic terms of CoLF [Chen and Pfenning, 2023].

2 First-Order Rational Unification

First-order rational unification [Jaffar, 1984] arises directly out of first-order unification [Robinson, 1965], but without occurs check. We give a new presentation of the algorithm based on saturation [Ganzinger, 1996; Pfenning, 2006], that mimics the structure of the higher-order case in Section 3. In Jaffar's algorithm, the unification is presented as transformations on equivalence classes of terms (containing variables) into a solved form, where solutions can be read directly. Our algorithm shares the essential idea as his algorithm, but we presented it very differently. The primary motivation for a different presentation is to make the later presentation of the higher-order case easier to follow. The circular terms in Jaffar's algorithm are created implicitly by the presence of a variable and its recursive definition in the same equivalence class, whereas we use explicit recursive definitions and explicit equations between terms.

2.1 Problem Formulation

We present the definition of a unification problem in this section, and then present in Section 2.2 a flattened form of the unification problem that the algorithm and the proofs assume, obtainable by preprocessing. With three syntactic entities, *constructors* (written c, d, or e) and *unification metavariables* (written in capital letters E, F, G, H), *recursion constants* (written in underlined letters $\underline{r}, \underline{s}, \underline{t}$) [Chen and Pfenning, 2023], possibly with subscripts, a first-order *concrete unification context* Δ_c is a system of equations $T \doteq T'$ together with definitions for recursion constants that may occur in T. In term $c\, T_1 \, \dots \, T_n$, c is the *head* and $T_1 \, \dots \, T_n$ are the *arguments* of c. The grammar is shown as follows. It enforces that recursive definitions are required to be contractive: $\underline{r} =_d T$ means the head of T must be a constructor:

$$\begin{array}{ll} \text{Concrete Unification Contexts} & \Delta_c ::= [\,] \mid \Delta_c, T_1 \doteq T_2 \mid \Delta_c, \underline{r} =_d c\, T_1 \, \dots \, T_n \\ \text{Terms} & T ::= c\, T_1 \, \dots \, T_n \mid H \mid \underline{r}. \end{array}$$

We now define the infinitary denotation of T in a context Δ_c by depth k observations of M. First, define $M_\perp$ to be first-order terms with the symbol $\perp$, and contractive and recursive unification metavariables (defined later in Section 2.2):

$$M_\perp ::= c\,(M_\perp)_1 \, \dots \, (M_\perp)_n \mid H^{\blacksquare} \mid H^{\bigcirc} \mid \perp.$$

We define *definitional expansion* up to depth k of a term T into $M_\perp$ as the function $\mathsf{exp}^{\Delta_c}_{(k)}(T) = M_\perp$, defined by lexicographic induction on (k, T). Since the parameter Δ_c remains unchanged throughout, we omit writing it to reduce visual clutter if it is not referenced:

$$\begin{aligned} &\mathsf{exp}_{(0)}(T) = \perp \\ &\mathsf{exp}_{(k+1)}(c\, T_1 \, \dots \, T_n) = c\,(\mathsf{exp}_{(k)}(M_1)) \, \dots \, (\mathsf{exp}_{(k)}(M_n)) \\ &\mathsf{exp}_{(k+1)}(H) = H^{\bigcirc} \\ &\mathsf{exp}^{\Delta_c}_{(k+1)}(\underline{r}) = \mathsf{exp}_{(k+1)}(c\, T_1 \, \dots \, T_n) \text{ if } \underline{r} =_d c\, T_1 \, \dots \, T_n \in \Delta_c. \end{aligned}$$

As an example, given a signature for conatural numbers and their simple equality, we are asked to find which number's double cosuccessor is `omega`.

```
conat : cotype.
cozero : conat.
cosucc : conat -> conat.

omega : conat = cosucc omega.
?- omega = (cosucc (cosucc H)).
```

We may formulate the problem as follows, where H is a fresh unification metavariable standing for the answer to our query:

$$\Delta_c = \{\underline{omega} =_d cosucc\ \underline{omega}, cosucc\,(cosucc\ H) \doteq \underline{omega}\}.$$

We will not define unifiers for the concrete unification context, but the definition would look similar to the one for the unification context after the preprocessing phase defined next. Eventually, we will find the following unifier for Δ_c:

$$\Gamma_c = \{H \doteq \underline{omega}, \underline{omega} =_d cosucc\ \underline{omega}\}.$$

Notice that given two terms in a concrete context, their equality of definitional expansion up to depth ω is decidable [Chen and Pfenning, 2023; Huet, 1998]. The core idea is to carry out structural comparisons of the terms and memorizing intermediate equalities. This comparison terminates because the rationality of the terms ensures that only a finite number of intermediate equalities are possible.

2.2 Preprocessing

Terms are now divided into recursive terms and contractive terms. A *contractive term* is a term with a constructor as its head, and a *recursive term* is a term with a recursion constant as its head. The purpose of preprocessing is to put recursive definitions into shallow forms that are one level deep, meaning that the arguments to a constructor must not be contractive terms and can only be recursive terms. That means a term $c\,(d\,e)$ must be written down using recursive definitions: $c\,\underline{r}, \underline{r} =_d d\,\underline{s}, \underline{s} =_d e$. This greatly simplifies the termination proof of the unification algorithm here and for the higher-order case, which we eventually wish to develop. Similarly, unification metavariables are divided into *recursive unification metavariables* (with superscript $^{\bigcirc}$), which may unify with only recursion constants, and *contractive unification metavariables* (with superscript $^{\blacksquare}$) which may only unify contractive terms. We use the lower case letter m to denote either $\blacksquare$ or $\bigcirc$ and write H^m to indicate a unification metavariable H that is either contractive or recursive. We also include a special symbol contra for contradictory unification contexts that do not have a unifier. The unification context now only permits equations between two recursive terms or two contractive terms. The grammar is as follows:

$$\begin{array}{ll}
\text{Unification Contexts} & \Delta, \Gamma ::= [\,] \mid \Delta, U_1 \doteq U_2 \mid \Delta, N_1 \doteq N_2 \mid \Delta, \underline{r} =_d U \mid \Delta, \mathsf{contra} \\
\text{Contractive Terms} & U ::= c\, N_1\, \ldots\, N_n \mid H^{\blacksquare} \\
\text{Recursive Terms} & N ::= \underline{r} \mid H^{\bigcirc}.
\end{array}$$

As an example, we would like the concrete unification context Δ_c defined in the previous section

$$\Delta_c = \{\underline{omega} =_d cosucc\ \underline{omega}, cosucc\,(cosucc\, H) \doteq \underline{omega}\}$$

to be processed to the following unification context Δ:

$$\Delta = \{\underline{omega} =_d cosucc\ \underline{omega}, \underline{s} =_d cosucc\, \underline{r}, \underline{r} =_d cosucc\, H^{\bigcirc}, \underline{s} \doteq \underline{omega}\}.$$

We define a preprocessing translation from concrete unification contexts to unification contexts. We write $\Delta_c \rhd \Delta$ translation of Δ_c to Δ, $T \rhd^{\blacksquare} U \diamond \Delta$ for translating a term T into a contractive term U with a new context Δ, and $T \rhd^{\bigcirc} N \diamond \Delta$ for translating a term T into a recursive term N with a new context Δ. We treat a unification context as an unordered list and may write Δ_1, Δ_2 to join two contexts Δ_1 and Δ_2 with disjoint sets of recursion constants. If the set of recursion constants of Δ_1 is not disjoint from the set of recursion constants of Δ_2, we may consistently rename recursion constants in Δ_2 such that Δ_1, Δ_2 is always defined.

$\boxed{\Delta_c \rhd \Delta}$

$$\frac{}{[\,] \rhd [\,]}(1) \qquad \frac{\Delta_c \rhd \Delta_1 \qquad T_1 \rhd^{\bigcirc} N_1 \diamond \Delta_2 \qquad T_2 \rhd^{\bigcirc} N_2 \diamond \Delta_3}{\Delta_c, T_1 \doteq T_2 \rhd \Delta_1, \Delta_2, \Delta_3, N_1 \doteq N_2}(2)$$

$$\frac{\Delta_c \rhd \Delta_1 \qquad c\, T_1\, \ldots\, T_n \rhd^{\blacksquare} U \diamond \Delta_2}{\Delta_c, \underline{r} =_d c\, T_1\, \ldots\, T_n \rhd \Delta_1, \Delta_2, \underline{r} =_d U}(3)$$

$$\boxed{T \triangleright^{\bigcirc} N \diamond \Delta}$$

$$\frac{c\,T_1\ \ldots\ T_n \triangleright^{\blacksquare} U \diamond \Delta}{c\,T_1\ \ldots\ T_n \triangleright^{\bigcirc} \underline{r} \diamond (\Delta, \underline{r} =_d U)}(\underline{r}\text{ fresh})(4) \qquad \frac{}{H \triangleright H^{\bigcirc} \diamond [\,]}(5) \qquad \frac{}{\underline{r} \triangleright \underline{r} \diamond [\,]}(6)$$

$$\boxed{T \triangleright^{\blacksquare} U \diamond \Delta}$$

$$\frac{\forall_{i,1\leq i\leq n}.T_i \triangleright^{\bigcirc} N_i \diamond \Delta_i}{c\,T_1\ \ldots\ T_n \triangleright^{\blacksquare} c\,N_1\ \ldots\ N_n \diamond (\Delta_1, \ldots, \Delta_n)}(7) \qquad \text{(No rules for } T = H \text{ or } T = \underline{r}\text{)}$$

The informal intuition of the transformation rules is that we make transform both sides of a unification equation into recursive terms and transform the body of a recursive definition to a contractive term. Recursion constants are already recursive terms. To transform any term with a constructor head to a contractive term, we transform the arguments into recursive terms (rule (7)). To transform any nonrecursive term into a recursive term, create a recursive definition that mimics the term (rule (4)).

We define the definitional expansion at depth k for a recursive or a contractive term mutually recursively, $\mathsf{exp}^{\Delta}_{(k)}(U) = M_{\perp}$ and $\mathsf{exp}^{\Delta}_{(k)}(N) = M_{\perp}$. We take the liberty to omit writing Δ if it remains unchanged throughout and is not referenced:

$$\begin{aligned}
&\mathsf{exp}_{(0)}(U) = \perp\\
&\mathsf{exp}_{(k+1)}(H^{\blacksquare}) = H^{\blacksquare}\\
&\mathsf{exp}_{(k+1)}(c\,N_1\ \ldots\ N_n) = c\,(\mathsf{exp}_{(k)}(N_1))\ \ldots\ (\mathsf{exp}_{(k)}(N_n))\\
&\mathsf{exp}_{(0)}(N) = \perp\\
&\mathsf{exp}_{(k+1)}(H^{\bigcirc}) = H^{\bigcirc}\\
&\mathsf{exp}^{\Delta}_{(k+1)}(\underline{r}) = \mathsf{exp}^{\Delta}_{(k+1)}(U) \text{ if } \underline{r} =_d U \in \Delta.
\end{aligned}$$

The translation preserves the definitional expansion of arbitrary depth.

Theorem 2.1. *We have*

(1) *If* $T \triangleright^{\blacksquare} U \diamond \Delta_2$ *and* $\mathsf{exp}^{\Delta_c}_{(k)}(\underline{s}) = \mathsf{exp}^{\Delta_1}_{(k)}(\underline{s})$ *for all* s *occurring in* T, *then* $\mathsf{exp}^{\Delta_c}_{(k)}(T) = \mathsf{exp}^{\Delta_1,\Delta_2}_{(k)}(U)$.
(2) *If* $T \triangleright^{\bigcirc} N \diamond \Delta_2$ *and* $\mathsf{exp}^{\Delta_c}_{(k)}(\underline{s}) = \mathsf{exp}^{\Delta_1}_{(k)}(\underline{s})$ *for all* s *occurring in* T, *then* $\mathsf{exp}^{\Delta_c}_{(k)}(T) = \mathsf{exp}^{\Delta_1,\Delta_2}_{(k)}(N)$.

Proof. Simultaneous induction on (k, T), where (2) may appeal to (1) without a decrease in size. We show the case for rules (4) and (7) as examples.

Rule (4): The premise $c\,T_1\ \ldots\ T_n \triangleright^{\blacksquare} U \diamond \Delta$ implies, by the induction hypothesis, that $\mathsf{exp}^{\Delta_c}_{(k)}$ $(c\,T_1\ \ldots T_n) = \mathsf{exp}^{\Delta_1,\Delta}_{(k)}(U)$. We have

$$\begin{aligned}
&\mathsf{exp}^{\Delta_c}_{(k)}(c\,T_1\ \ldots\ T_n)\\
&= \mathsf{exp}^{\Delta_1,\Delta}_{(k)}(U) && \text{(by the induction hypothesis)}\\
&= \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d U}_{(k)}(U) && \text{(because } \underline{r} \text{ is fresh)}\\
&= \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d U}_{(k)}(\underline{r}) && \text{(by definition).}
\end{aligned}$$

Rule (7): When $k = 0$, the result is trivial. When $k > 0$, the premise $T_i \triangleright^{\bigcirc} N_i \diamond \Delta'_i$ implies, by the induction hypothesis, that $\mathsf{exp}^{\Delta_c}_{(k-1)}(T_i) = \mathsf{exp}^{\Delta_1,\Delta'_i}_{(k-1)}(N_i)$. Let $\Delta' = \Delta'_1, \ldots, \Delta'_n$, we have $\mathsf{exp}^{\Delta_c}_{(k-1)}(N_i) = \mathsf{exp}^{\Delta_1,\Delta'_i}_{(k-1)}(N_i) = \mathsf{exp}^{\Delta_1,\Delta'}_{(k-1)}(N_i)$, because each Δ'_i may only contain fresh recursion constants. We have

$$\begin{aligned}
&\mathsf{exp}^{\Delta_c}_{(k)}(c\,T_1\,\ldots\,T_n) && \\
&= c\,(\mathsf{exp}^{\Delta_c}_{(k-1)}(T_1))\,\ldots\,(\mathsf{exp}^{\Delta_c}_{(k-1)}(T_n)) && \text{(by definition)} \\
&= c\,(\mathsf{exp}^{\Delta_1,\Delta'}_{(k-1)}(N_1))\,\ldots\,(\mathsf{exp}^{\Delta_1,\Delta'}_{(k-1)}(N_n)) && \text{(by the induction hypothesis)} \\
&= \mathsf{exp}^{\Delta_1,\Delta'}_{(k)}(c\,N_1\,\ldots\,N_n) && \text{(by definition).}
\end{aligned}$$

□

Corollary 2.2. *If $\Delta_c \triangleright \Delta$, then every equation in Δ_c corresponds to an equation in Δ with equal definitional denotation, and every recursive definition in Δ_c corresponds to a recursive definition in Δ.*

Proof. Directly by structural induction over $\Delta_c \triangleright \Delta$. □

It is worth noting that we have assumed all concrete unification metavariables H are recursive, in the sense that they may unify with a recursion constant. In practice, implementations may want to use the preprocessed forms directly. The concrete form and the translation procedure merely serve as a mechanism to parse the user's input and as a formal explanation of the flattened definitions.

We take the flattened unification context as the "canonical representation" for a unification problem from now on, and we may use the syntax category M for either U or N. We use $\mathsf{defs}(\Delta)$ and $\mathsf{eqs}(\Delta)$ to denote the list of recursive definitions and equations of Δ, respectively. Definitional expansion exp does not depend on unification equations but only on recursive definitions, and thus, we have $\mathsf{exp}^{\Delta}_{(k)}(M) = \mathsf{exp}^{\mathsf{defs}(\Delta)}_{(k)}(M)$, for all Δ, k, and M.

2.3 Term Equality and Unifiers

Two terms are equal in a unification context if they have the same definitional expansion, i.e., given $M \doteq M'$ in Δ, we say that M is equal to M' (and thus the equation $M \doteq M'$ *holds*) if $\mathsf{exp}^{\Delta}_{(k)}(M) = \mathsf{exp}^{\Delta}_{(k)}(M')$ for all k. We say that a unification context is *contradiction-free* if contra is not present in the context.

A (simultaneous) substitution is usually understood as a mapping from unification metavariables to terms. In the case of circular terms, the substitutions may carry recursive definitions. We choose to define substitutions as unification contexts of special forms, where the left-hand sides of all unification equations are unification metavariables, and the corresponding right-hand sides are their values. We write Γ for substitutions and Δ for ordinary unification contexts. A *substitution* is a contradiction-free unification context where the left-hand side of each unification equation is a unique unification metavariable. The set of unification metavariables that occur on the left-hand sides of a substitution Γ is called the *domain* of the substitution and is written $\mathsf{dom}(\Gamma)$. If a substitution contains an equation $H^m \doteq M$, we say that M is the *value* of H^m in Γ. Two substitutions are equal if they have the same domain, and the definitional expansions of the values of each unification metavariable in their domain are equal, i.e., $\Gamma = \Gamma'$ if $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$, and for all $H^m \in \mathsf{dom}(\Gamma)$, $\mathsf{exp}^{\Gamma}_{(k)}(H^m[\Gamma]) = \mathsf{exp}^{\Gamma'}_{(k)}(H^m[\Gamma])$, where $H^m[\Gamma]$ is the value of H^m in Γ, obtained by the substitution operation that will be defined.

As an example, Γ and Γ' below are substitutions with the domain $\{H^{\bigcirc}\}$:

$$\begin{aligned}
\Gamma &= \{H^{\bigcirc} \doteq \underline{omega}, && \underline{omega} =_d cosucc\ \underline{omega}\} \\
\Gamma' &= \{H^{\bigcirc} \doteq \underline{s}, && \underline{omega} =_d cosucc\ \underline{omega}, \underline{s} =_d cosucc\ \underline{omega}\}.
\end{aligned}$$

Moreover, $\Gamma = \Gamma'$ because the expansions of every unification metavariable in the domain are equal: $H^{\bigcirc}$ expands to $cosucc\,(cosucc\ldots)$.

We emphasize that in a substitution, unification metavariables occurring on the right-hand sides of unification equations and in recursive definitions are free. Thus, the substitution Γ'' below has

$H^{\bigcirc}$ in the recursive definition free, and Γ'' is not equal to Γ defined above:

$$\Gamma'' = \{H^{\bigcirc} \doteq \underline{\mathit{omega}}, \underline{\mathit{omega}} =_d \mathit{cosucc}\ H^{\bigcirc}\}.$$

We write $U[\Gamma]$ and $N[\Gamma]$ for applying the substitution to terms. They are defined in obvious ways:

$$\begin{array}{l}
(c\, N_1\ \ldots\ N_n)[\Gamma] = c\,(N_1[\Gamma])\ \ldots\ (N_n[\Gamma]) \\
(H^{\blacksquare})[\Gamma] = \begin{cases} U' & \text{if } H^{\blacksquare} \doteq U' \in \mathsf{eqs}(\Gamma) \\ H^{\blacksquare} & \text{otherwise} \end{cases} \\
(\underline{r})[\Gamma] = \underline{r} \\
(H^{\bigcirc})[\Gamma] = \begin{cases} N' & \text{if } H^{\bigcirc} \doteq N' \in \mathsf{eqs}(\Gamma) \\ H^{\bigcirc} & \text{otherwise.} \end{cases}
\end{array}$$

The application of substitution Γ to a *unification context* Δ is denoted $\Delta[\Gamma]$, which replaces occurrences of unification metavariables in Γ by their values in Δ, while combining all recursive definitions and performing recursion constant renaming as necessary:

$$\Delta[\Gamma] = \mathsf{defs}(\Gamma), \{M[\Gamma] \doteq M'[\Gamma] \mid M \doteq M' \in \mathsf{eqs}(\Delta)\}, \{\underline{r} =_d U[\Gamma] \mid \underline{r} =_d U \in \mathsf{defs}(\Delta)\}.$$

The application of a substitution Γ_2 to another *substitution* is Γ_1 is denoted $\Gamma_1[\Gamma_2]$, and it replaces the occurrences of unification metavariables in the right-hand sides and recursive definitions of Γ_1 by their values in Γ_2 and combine all recursive definitions, performing recursion constant renaming as necessary:

$$\Gamma_1[\Gamma_2] = \mathsf{defs}(\Gamma_2), \{H^m \doteq M'[\Gamma_2] \mid H^m \doteq M' \in \mathsf{eqs}(\Gamma_1)\}, \{\underline{r} =_d U[\Gamma_2] \mid \underline{r} =_d U \in \mathsf{defs}(\Gamma_1)\}.$$

The composition of substitutions is denoted $\Gamma_1 \circ \Gamma_2$ (applying Γ_1 and then applying Γ_2), and is defined to be $\Gamma_1[\Gamma_2]$ plus any additional substitutions in Γ_2:

$$\Gamma_1 \circ \Gamma_2 = (\Gamma_1[\Gamma_2]), \{H^m \doteq M \mid H^m \doteq M \in \mathsf{eqs}(\Gamma_2) \wedge H^m \notin \mathsf{dom}(\Gamma_1)\}.$$

Let $UV(\Delta)$ denote the set of all unification metavariables that occur in Δ, a *unifier* for a contradiction-free unification context Δ is a substitution Γ such that $UV(\Delta) = \mathsf{dom}(\Gamma)$, and every equation in $\Delta[\Gamma]$ holds. A unification context Δ with $\mathsf{contra} \in \Delta$ has no unifiers. A unifier Γ_1 is *more general* than another unifier Γ_2 if there is a substitution Γ' such that $\Gamma_1 \circ \Gamma' = \Gamma_2$.

As an example, given Γ and Δ defined below, Γ is a unifier of Δ, because every equation in $\Delta[\Gamma]$ holds. Notice that when carrying out the substitution, the duplicate recursion constant $\underline{\mathit{omega}}$ in Γ is renamed to $\underline{t}$. The major changes are highlighted in blue.

$$\begin{array}{l}
\Gamma = \{ \\
\quad H^{\bigcirc} \doteq \underline{\mathit{omega}}, \\
\\
\\
\quad \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}} \\
\}
\end{array}$$

$$\begin{array}{l}
\Delta = \{ \\
\quad \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, \\
\quad \underline{s} =_d \mathit{cosucc}\ \underline{r}, \\
\quad \underline{r} =_d \mathit{cosucc}\ H^{\bigcirc}, \\
\\
\quad \underline{s} \doteq \underline{\mathit{omega}} \\
\}
\end{array}$$

$$\begin{array}{l}
\Delta[\Gamma] = \{ \\
\quad \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, \\
\quad \underline{s} =_d \mathit{cosucc}\ \underline{r}, \\
\quad \underline{r} =_d \mathit{cosucc}\ \underline{t}, \\
\quad \underline{t} =_d \mathit{cosucc}\ \underline{t}, \\
\quad \underline{s} \doteq \underline{\mathit{omega}} \\
\}
\end{array}$$

2.4 The Unification Algorithm

We saturate the unification context Δ using the rules defined below. If all the premises of a rule are present in the context, we add the rule's conclusion to the context. The algorithm terminates when no new equations or recursive definitions can be added to the context. The goal of the rules is to ensure that in a saturated unification context, either contra is present, indicating there is no unifier, or there is an equation between each unification metavariable and its value in a unifier.

Structural Rules:

(SIMP-F)
$$\frac{c\,N_1\ \ldots\ N_n \doteq d\,N'_1\ \ldots\ N'_n}{\mathsf{contra}}\,(c \neq d)$$

(SIMP)
$$\frac{c\,N_1\ \ldots N_n \doteq c\,N'_1\ \ldots N'_n}{N_1 \doteq N'_1, \ldots, N_n \doteq N'_n}$$

Expansion, Symmetry, and Transitivity

(R-EXP)
$$\frac{\underline{r} \doteq \underline{s} \qquad \underline{r} =_d U_1 \qquad \underline{s} =_d U_2}{U_1 \doteq U_2}$$

(U-SYM)
$$\frac{U \doteq U'}{U' \doteq U}$$

(U-TRANS)
$$\frac{U_1 \doteq U_2 \qquad U_2 \doteq U_3}{U_1 \doteq U_3}$$

(N-SYM)
$$\frac{N \doteq N'}{N' \doteq N}$$

(N-TRANS)
$$\frac{N_1 \doteq N_2 \qquad N_2 \doteq N_3}{N_1 \doteq N_3}$$

We give an example of the ways the algorithm operates on our previous example. We label each equation with a number and use Δ_i to refer to the set of Equations and Definitions $(1)-(i)$. For example, our example Δ is denoted Δ_4, consisting of Equations and Definitions $(1)-(4)$. At each step, we show some additional equations and definitions and the ways they are obtained. We only show the first few important steps and the rest will be only symmetry and transitivity:

(1) $\underline{omega} =_d cosucc\ \underline{omega}$ given
(2) $\underline{s} =_d cosucc\,\underline{r}$
(3) $\underline{r} =_d cosucc\,H^{\circ}$
(4) $\underline{s} \doteq \underline{omega}$
(5) $cosucc\ \underline{r} \doteq cosucc\ \underline{omega}$ by Rule (R-EXP) on (4), (2), and (1)
(6) $\underline{r} \doteq \underline{omega}$ by Rule (SIMP) on (5)
(7) $cosucc\ H^{\circ} \doteq cosucc\ \underline{omega}$ by Rule (R-EXP) on (6), (3), and (1)
(8) $H^{\circ} \doteq \underline{omega}$ by Rule (SIMP) on (7)
(9) $\ldots$ by Rules (U-SYM)(U-TRANS)(N-SYM)(N-TRANS)

2.5 Saturated Unification Contexts

We now describe how a unifier may be constructed from a saturated contradiction-free unification context. Given a unification context Δ, we say that a unification metavariable $H^{\blacksquare}$ is *resolved* if there is an equation of the form $H^{\blacksquare} \doteq c\,N_1\ \ldots\ N_n$ or $c\,N_1\ \ldots\ N_n \doteq H^{\blacksquare}$, and $c\,N_1\ \ldots\ N_n$ is called a resolution of $H^{\blacksquare}$. Similarly, we say that a unification metavariable H° is *resolved* if there is an equation of the form $H^{\circ} \doteq \underline{r}$ or $\underline{r} \doteq H^{\circ}$, and $\underline{r}$ is called a resolution of H°. In a unification context, every unification metavariable is either resolved or unresolved. There may be multiple resolutions for each resolved unification metavariable; we pick a unique resolution for each unification metavariable. The choice of resolution is not important, because every resolution will be equal modulo definitional expansion in a saturated contradiction-free context. Unresolved unification metavariables form an equivalence class equated by $\doteq$, and we pick a unique representative variable for each class. We construct the substitution $\Gamma = \mathsf{unif}(\Delta)$ for a contradiction-free context Δ as follows:

(1) Start with Γ containing all recursive definitions of Δ.
(2) For each resolved unification metavariable in $UV(\Delta)$, add to Γ the unification metavariable and its resolution.
(3) For each unresolved unification metavariable in $UV(\Delta)$, add to Γ the unification metavariable and the representative unification metavariable for its equivalence class.
(4) Replace the occurrences of resolved unification metavariables in the right-hand sides and recursive definitions of Γ with their resolutions, and replace the occurrences of unresolved unification metavariables in the right-hand sides and recursive definitions of Γ with their representative unification metavariables. Repeat this step until all unification metavariables in the right-hand sides and recursive definitions are representative unification metavariables for some equivalence class of unresolved unification metavariables.

We will later show that if Δ is a saturated contradiction-free unification context, then $\Gamma = \mathsf{unif}(\Delta)$ is a unifier for Δ. As an example, we show how the unifier for Δ_8 (Equations and Definitions (1)–(8) defined above) can be constructed. The main differences in each step are highlighted in blue:

(1) Initialize Γ_1 to all recursive definitions of Δ_8.
(2) Since $H^{\bigcirc}$ is resolved, we add its resolution to get Γ_2.
(3) There is no unresolved unification metavariable, we skip step (3).
(4) Replace occurrences of resolved unification metavariables with their resolutions to get Γ_3.

$$\begin{array}{l|l|l}
\Gamma_1 = \{ & \Gamma_2 = \{ & \Gamma_3 = \{ \\
\ \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, & \ \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, & \ \underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, \\
\ \underline{s} =_d \mathit{cosucc}\, \underline{r}, & \ \underline{s} =_d \mathit{cosucc}\, \underline{r}, & \ \underline{s} =_d \mathit{cosucc}\, \underline{r}, \\
\ \underline{r} =_d \mathit{cosucc}\, H^{\bigcirc} & \ \underline{r} =_d \mathit{cosucc}\, H^{\bigcirc}, & \ \underline{r} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, \\
\} & \ H^{\bigcirc} \doteq \underline{\mathit{omega}} & \ H^{\bigcirc} \doteq \underline{\mathit{omega}} \\
 & \} & \}
\end{array}$$

(5) Note that we may remove unused recursive definitions (s, r) to get an equivalent substitution Γ_4:
$\Gamma_4 = \{\underline{\mathit{omega}} =_d \mathit{cosucc}\ \underline{\mathit{omega}}, H^{\bigcirc} \doteq \underline{\mathit{omega}}\}$.

2.6 Correctness of the Algorithm

We want to show that given a unification context Δ_1, it has a finitary saturation sequence

$$\Delta_1 \longrightarrow \Delta_2 \longrightarrow \cdots \longrightarrow \Delta_n,$$

where Δ_n is a saturated unification context that has $\mathsf{unif}(\Delta_n)$ as its most general unifier. Moreover, unifiers are preserved between Δ_i and Δ_{i+1}. Then, the most general unifiers of Δ_n are the most general unifiers of Δ_1. Concretely, we want to show three things:

(1) *Correspondence.* At each step of the algorithm, the most general unifier of the context before corresponds to the most general unifier of the context after (Theorem 2.5).
(2) *Termination.* Any unification context always saturates in a finite number of steps (Theorem 2.6).
(3) *Correctness.* The unifier for a saturated unification context $\mathsf{unif}(\Delta_n)$ is actually the most general unifier (Theorem 2.7).

Lemma 2.3. *Let Δ' be a unification context, and let Δ have the same set of recursive definitions and unification metavariables, but fewer unification equations than Δ', i.e.,* $\mathsf{eqs}(\Delta) \subseteq \mathsf{eqs}(\Delta')$, $\mathsf{defs}(\Delta) = \mathsf{defs}(\Delta')$*,* $UV(\Delta) = UV(\Delta')$*, then any unifier of Δ' is a unifier of Δ.*

Proof. Let Γ be a unifier of Δ', all unification equations of $\Delta'[\Gamma]$ hold. Take any $M \doteq M' \in \Delta$, we know $\mathsf{exp}_{(k)}^{\Delta'[\Gamma]}(M[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta'[\Gamma]}(M'[\Gamma])$, we have $UV(\Delta) = UV(\Delta')$, and it suffices to show $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M'[\Gamma])$ by showing $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta'[\Gamma]}(M[\Gamma])$. But since definitional expansions only depend on recursive definitions, we have

$$\begin{array}{ll} \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M[\Gamma]) & \\ = \mathsf{exp}_{(k)}^{\mathsf{defs}(\Delta[\Gamma])}(M[\Gamma]) & \text{(expansion only definition only depends on definitions of } \Delta[\Gamma]) \\ = \mathsf{exp}_{(k)}^{\mathsf{defs}(\Delta'[\Gamma])}(M[\Gamma]) & (M \text{ can only depend on recursion constants occurring in } \Delta) \\ = \mathsf{exp}_{(k)}^{\Delta'[\Gamma]}(M[\Gamma]) & \text{(expansion only definition only depends on definitions of } \Delta'[\Gamma]). \quad \square \end{array}$$

Lemma 2.4. *If Γ is a unifier for Δ, then Γ is a unifier for Δ' where Δ' has all recursive definitions of Δ and additional true equations $M \doteq M'$ in the sense that $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M'[\Gamma])$.*

Proof. Because definitional expansion depends only on recursive definitions but not unification equations, we have $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M) = \mathsf{exp}_{(k)}^{\Delta'[\Gamma]}(M)$ for all k and M. □

Theorem 2.5 (Correspondence). *If Δ' is obtained from Δ by applying one of the rules, then the unifiers of Δ' and the unifiers of Δ coincide.*

Proof. We analyze each rule.

Case (SIMP-F), both Δ and Δ' have no unifiers.

Case (SIMP), it's easy to check that any unifier Γ of Δ' is a unifier of Δ by Lemma 2.3. Now suppose Γ is a unifier of Δ, we want to show that Γ is a unifier for Δ'. The additional equations $M_i \doteq M_i'$ in Δ' satisfy $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M_i[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(M_i'[\Gamma])$, and the rest follows by Lemma 2.4.

The rest of the cases are similar to Case (SIMP). □

Theorem 2.6 (Termination). *The saturation algorithm always terminates.*

Proof. We observe that all terms in an equation are built up from recursion constants, unification metavariables, and constants, and all terms have finite depth (due to the grammar) and finite width (the maximum width is preserved by the algorithm). There can be only finitely many equations given a bounded number of recursion constants, constructors, and unification metavariables, and there are no rules that create additional recursion constants, constructors, or unification metavariables. □

Theorem 2.7 (Correctness of Unifiers). *Given any saturated contradiction-free unification context Δ, let $\Gamma = \mathsf{unif}(\Delta)$, then Γ is a unifier for Δ. Moreover, it is the most general unifier.*

Proof. The proof is broken into two parts. The first part is to show that Γ is a unifier, and the second part is to show that Γ is the most general unifier.

(Part 1) To show Γ is a unifier, we need to show that $\mathsf{dom}(\Gamma) = UV(\Delta)$, which is true by definition, and that every equation in $\Delta[\Gamma]$ holds. We show the following two claims simultaneously by induction on k, where claim (2) may refer to claim (1) without decreasing k:

(1) For all $U_1 \doteq U_2$ in Δ, $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_1[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_2[\Gamma])$.
(2) For all $N_1 \doteq N_2$ in Δ, $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(N_1[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(N_2[\Gamma])$.

Both claims are trivial when $k = 0$. Consider the case when $k > 0$, we show (1) and (2) by case analysis on the structure of $U_1 \doteq U_2$ and $N_1 \doteq N_2$:

(a) Both U_1 and U_2 have constructors as their heads. Since $\mathsf{contra} \notin \Delta$, they must have identical constructor heads. Now let $U_1 = c\, N_1 \ \ldots \ N_n$ and $U_2 = c\, N_1' \ \ldots \ N_n'$. Since Δ is saturated, we have $N_i \doteq N_i'$ for all $1 \leq i \leq n$. The result then follows from the fact that each N_i and N_i' have equal definitional expansion up to depth $k-1$ by induction hypothesis.

(b) Both U_1 and U_2 are contractive unification metavariables. Due to saturation, either both are unresolved, and the result follows because they would be in the same equivalence class and thus have the same representative unification metavariable, or both are resolved. If they have a unique resolution U, then we have $\mathsf{exp}_{(k)}(U_1) = \mathsf{exp}_{(k)}(U) = \mathsf{exp}_{(k)}(U_2)$. If they have multiple resolutions and one of the resolutions is U, saturation guarantees that there is an equation between every resolution. Rule (SIMP) ensures that the equations between children of the head constructors are in Δ, and the two terms would be equal by IH, using a similar technique as the case (a).

(c) One of U_1 and U_2 is a unification metavariable, and the other has a constructor as its head. Obviously this is a resolution equation, and it suffices to show that all other resolutions have equal definitional expansions up to depth k, which follows from saturation and the case (a).

(d) Both N_1 and N_2 are recursion constants. Let $N_1 = \underline{r}$, where $\underline{r} =_d U_1 \in \Delta$ and $N_2 = \underline{s}$, where $\underline{s} =_d U_2 \in \Delta$. Since Δ is saturated, $U_1 \doteq U_2 \in \Delta$, and by IH (i.e., case (a) above), $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_1[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_2[\Gamma])$, and

$$\begin{aligned}
&\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(\underline{r}[\Gamma]) && \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(\underline{r}) && \text{(since } \underline{r}[\Gamma] = \underline{r}\text{)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_1[\Gamma]) && \text{(by the definition of } \Delta[\Gamma] \text{ and definitional expansion)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_2[\Gamma]) && \text{(shown)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(\underline{s}) && \text{(by the definition of } \Delta[\Gamma] \text{ and definitional expansion)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(\underline{s}[\Gamma]) && \text{(since } \underline{s}[\Gamma] = \underline{s}\text{).}
\end{aligned}$$

(e) The case when either or both of N_1 and N_2 are recursive unification metavariables are exactly analogous to cases (b) and (c).

(Part 2) To show Γ is the most general unifier, given any other unifier Γ_2 of Δ, it suffices to construct a unifier Γ_1 such that $\Gamma \circ \Gamma_1 = \Gamma_2$. But the construction of Γ_1 is easy: Γ_2 must map resolved unification metavariables analogously as Γ (otherwise a contradiction will arise), and it may choose to map equivalence classes of unresolved unification metavariables freely. Γ_1 simply records how unresolved unification metavariables are mapped in Γ_2. □

3 Higher-Order Pattern Unification

In this section, we extend the algorithm to a higher-order setting. In particular, both recursive definitions and unification metavariables are of higher-order types: They may be applied to pattern variables. The main technical challenge of the higher-order case is to handle the scoping of variables in the presence of recursive definitions. We copy Miller's higher-order pattern unification algorithm [Miller, 1991] for nonrecursive cases and handle the recursive definitions by delegating the scoping to unification metavariables (i.e., rule (PRUNE) in Section 3.4). In terms of presentation, recursion constants and unification metavariables are always applied to pattern variables. We need to update the definitions to take variable renaming into account.

3.1 Problem Formulation

We now give a similar development by allowing recursion constants and unification metavariables to carry patterns [Miller, 1991]. A *pattern* is a list of pairwise distinct bound variables (written x, y, or z), and the *pattern restriction* ensures that a recursion constant or a unification metavariable may only be applied to a pattern. Here's an example of a higher-order pattern unification problem (without recursive definitions):

$$\lambda x.\,\lambda y.\,\lambda z.\,c\,(F\,x\,y) \doteq \lambda x.\,\lambda y.\,\lambda z.\,c\,(G\,y\,z).$$

A variable may not appear free in a unifier. For instance, the substitution $F\,x\,y = x, G\,y\,z = x$ (i.e., $F = \lambda x.\,\lambda y.\,x, G = \lambda y.\,\lambda z.\,x$) is not a unifier because x is free in the substitution of G but the substitution $F\,x\,y = d, G\,y\,z = d$ is a unifier.

Regular Böhm trees [Huet, 1998], subsequently termed higher-order rational terms, provide a natural model for higher-order terms. As with the first-order case, our use of a context containing recursive definitions for recursion constants follows the design of CoLF [Chen and Pfenning, 2023]. While CoLF allows repetitions of bound variables in the arguments to recursion constants, we disallow them in the setting of unification to ensure that most general unifiers exist. This is not a restriction in practice, because any recursive definition with repetitive arguments can be rewritten to definitions within the pattern fragment, as observed by Huet [1998]. For example, if we have nonpattern appears as arguments to a recursion constant $\underline{r}\,x\,x$, with $\underline{r} =_d \lambda y.\,\lambda z.\,T$, we can always create a fresh recursion constant $\underline{t}$ that mimics $\underline{r}$, i.e., $\underline{t}\,x$ with $\underline{t} =_d \lambda w.\,[w, w/y, z]T$. We assume that every unification metavariable, recursion constant, constructor, or variable is assigned a simple type, and terms are always written in β-normal-η-long forms, except that arguments to recursion constants and unification metavariables are written in nonexpanded forms. We also assume that λ-bound variables may undergo α-renaming. Here's the grammar for the unification problem in concrete syntax:

$$\begin{array}{ll}
\text{Concrete Unification Contexts} & \Delta_c ::= [\,] \mid \Delta_c, T_1 \doteq T_2 \mid \Delta_c, \underline{r} =_d \lambda x_1.\ldots.\lambda x_l.\,h\,T_1\,\ldots\,T_n \\
\text{Terms} & T ::= \lambda x_1.\ldots.\lambda x_l.\,h\,T_1\,\ldots\,T_n \mid \lambda x_1.\ldots.\lambda x_l.\,H\,y_1\,\ldots\,y_n \\
 & \quad\mid \lambda x_1.\ldots.\lambda x_l.\,\underline{r}\,y_1\,\ldots\,y_n \\
\text{Constructor or Variable Heads} & h ::= c \mid x.
\end{array}$$

To avoid visual clutter when writing down a list of terms, we adopt the following conventions of using overlines to represent a list of terms:

1. A list of variables $\bar{x}$ means $x_1, \ldots, x_l$ that are pairwise distinct.
2. A list of variables appearing in a binder position means iterative abstractions. For example, $\lambda\bar{x}$ means $\lambda x_1.\ldots.\lambda x_l$.
3. A list of variables in an application means iterative applications. For example, $c\,\bar{x}$ means $c\,x_1\,\ldots\,x_n$. Similarly, a list of terms in an application position means iterative applications. For example, $h\,\overline{N}$ means $h\,N_1\,\ldots\,N_n$.
4. The notation $[\bar{y}/\bar{x}]M$ denotes the simultaneous renaming of variables, substituting $\bar{y}$ for $\bar{x}$ in M.

With the new abbreviation notation, the grammar for the concrete syntax for a unification problem may be written as the following:

$$\begin{array}{ll}
\text{Concrete Unification Contexts} & \Delta_c ::= [\,] \mid \Delta_c, T_1 \doteq T_2 \mid \Delta_c, \underline{r} =_d \lambda\bar{x}.\,h\,\overline{T} \\
\text{Terms} & T ::= \lambda\bar{x}.\,h\,\overline{T} \mid \lambda\bar{x}.\,H\,\bar{y} \mid \lambda\bar{x}.\,\underline{r}\,\bar{y} \\
\text{Constructor or Variable Heads} & h ::= c \mid x.
\end{array}$$

The grammar enforces that the definition for a recursion constant is required to be contractive: It has a variable or a constructor for its head. We use $FV(T)$ to denote the set of free variables in T. We require all recursive definitions to be closed in the sense that $r =_d \lambda \bar{x}.\, h\,\overline{T} \in \Delta_c$ implies that $FV(h\,\overline{T}) \subseteq \bar{x}$.

As with the first-order case, we define the infinitary denotation of T in a context Δ_c by depth k observations of M. Now $M_\perp$ includes λ-bindings and variables:

$$M_\perp ::= \lambda\bar{x}.\, y\,\overline{M_\perp} \mid \lambda\bar{x}.\, c\,\overline{M_\perp} \mid \lambda\bar{x}.\, H^{\blacksquare}\,\overline{M_\perp} \mid \lambda\bar{x}.\, H^{\bigcirc}\,\overline{M_\perp} \mid \perp.$$

We define *definitional expansion* up to depth k of a term T into $M_\perp$ as the function $\mathsf{exp}_{(k)}^{\Delta_c}(T) = M_\perp$, defined by lexicographic induction on (k, T). We omit Δ_c to reduce visual clutter if it is not referenced:

$$\begin{aligned}
&\mathsf{exp}_{(0)}(T) = \perp\\
&\mathsf{exp}_{(k+1)}(\lambda\bar{x}.\, h\, T_1\ \ldots\ T_n) = \lambda\bar{x}.\, h\,(\mathsf{exp}_{(k)}(T_1))\ \ldots\ (\mathsf{exp}_{(k)}(T_n))\\
&\mathsf{exp}_{(k+1)}(\lambda\bar{x}.\, H\,\bar{y}) = \lambda\bar{x}.\, H^{\bigcirc}\bar{y}\\
&\mathsf{exp}_{(k+1)}^{\Delta_c}(\lambda\bar{x}.\, \underline{r}\,\bar{y}) = \mathsf{exp}_{(k+1)}^{\Delta_c}(\lambda\bar{x}.\, [\bar{y}/\bar{z}](h\,\overline{T}))\ \text{if}\ \underline{r} =_d \lambda\bar{z}.\, h\,\overline{T} \in \Delta_c.
\end{aligned}$$

As an example, we use an encoding of stream processors `sp` [Abel and Pientka, 2016; Danielsson and Altenkirch, 2010; Ghani et al., 2009]. At each step, a stream processor may choose to consume an input element (`get`) or produce an output element (`put`) and may do so indefinitely.[1] The use of λ-bindings due to the typing of get ensures that a stream can only produce elements that it has consumed:

```
sp : cotype.
element : type.
get: (element -> sp) -> sp.
put: element -> sp -> sp.
```

We may define stream processors `odd` or `even` that return only the odd-indexed or even-indexed elements, where the index starts from 0. We write λ-abstractions in square brackets, following the convention of CoLF [Chen and Pfenning, 2023]:

```
odd : sp = get ([x] even).
even : sp = get ([x] put x odd).
```

We may use unification to determine what is the behavior of the stream processor `S` after reading two elements of the input, if it behaves the same as `odd`:

```
?- get ([x] get ([y] S x y)) = odd.
```

The problem may be posed as the following concrete unification context Δ_c, which will be used as a running example:

$$\Delta_c = \{get\,(\lambda x.\ get\,(\lambda y.\, S\,x\,y)) \doteq \underline{odd}, \underline{odd} =_d get\,(\lambda x.\ \underline{even}), \underline{even} =_d get\,(\lambda x.\ put\ x\ \underline{odd})\}.$$

Eventually, we will find the following most general unifier, written in the concrete syntax:

$$\Gamma_c = \{S\,x\,y \doteq \underline{r_3}\,y, \underline{odd} =_d get\,(\lambda x.\ get\,(\lambda y.\, \underline{r_3}\,y)), \underline{r_3} =_d \lambda w.\ put\ w\,\underline{odd}\}.$$

[1] Stream processors were used to illustrate the semantics of mixed-induction and coinduction, but here we consider stream processors to be purely coinductive. Thus, we are happy to accept stream processors that keep consuming inputs without producing an output.

3.2 Preprocessing

As with the first-order case, we preprocess the unification problem Δ_c such that every recursive definition will only be one level deep. In the higher-order case, this processing is similar to Huet's treatment of regular Böhm trees [Huet, 1998]. Terms are divided into contractive terms U, which have constructors (c, d, e), bound variables (x, y, z), or contractive unification metavariables $(E^{\blacksquare}, F^{\blacksquare}, G^{\blacksquare}, H^{\blacksquare})$ as their heads, and recursive terms N, which have either recursion constants $(\underline{r}, \underline{s}, \underline{t})$ or recursive unification metavariables $(E^{\bigcirc}, F^{\bigcirc}, G^{\bigcirc}, H^{\bigcirc})$ as their heads. It is still the case that the terms are always written in β-normal-η-long forms, with the exception that arguments to recursion constants and unification metavariables are written in nonexpanded forms. The grammar for terms in their preprocessed form is summarized as follows:

$$\begin{array}{ll} \text{Unification Contexts} & \Delta ::= [\,] \mid \Delta, U_1 \doteq U_2 \mid \Delta, N_1 \doteq N_2 \mid \Delta, \underline{r} =_d U \mid \Delta, \mathsf{contra} \\ \text{Contractive Terms} & U ::= \lambda \bar{x}.\, y\, \overline{N} \mid \lambda \bar{x}.\, c\, \overline{N} \mid \lambda \bar{x}.\, H^{\blacksquare}\, \bar{y} \\ \text{Recursive Terms} & N ::= \lambda \bar{x}.\, \underline{r}\, \bar{y} \mid \lambda \bar{x}.\, H^{\bigcirc}\, \bar{y}. \end{array}$$

We use the letter h to denote either constructors c or variables x, but not unification metavariables. We use $FV(U)$ or $FV(N)$ to denote the set of free variables in U or N, and may use the syntax category M to denote either U or N. We also require all recursive definitions to be closed in the sense that $\underline{r} =_d U \in \Delta$ implies $FV(U) = \emptyset$. As with the first-order case, $UV(\Delta)$ denotes the set of unification metavariables in Δ. $\mathsf{defs}(\Delta)$ and $\mathsf{eqs}(\Delta)$ denote the list of recursive definitions and equations of Δ, respectively. Δ_1, Δ_2 denotes the union of two contexts Δ_1 and Δ_2, consistently renaming recursion constants in Δ_2 if necessary. Δ is *contradiction-free* if $\mathsf{contra} \notin \Delta$.

As with first-order terms, we use the judgments $\Delta_c \rhd \Delta$, $T \rhd^{\blacksquare} U \diamond \Delta$, $T \rhd^{\bigcirc} N \diamond \Delta$ for translating from concrete syntax into unification contexts, contractive terms, and recursive terms. They are defined as follows:

$$\boxed{\Delta_c \rhd \Delta}$$

$$\frac{}{[\,] \rhd [\,]}(1) \qquad \frac{\Delta_c \rhd \Delta_1 \qquad T_1 \rhd^{\bigcirc} N_1 \diamond \Delta_2 \qquad T_2 \rhd^{\bigcirc} N_2 \diamond \Delta_3}{\Delta_c, T_1 \doteq T_2 \rhd \Delta_1, \Delta_2, \Delta_3, N_1 \doteq N_2}(2)$$

$$\frac{\Delta_c \rhd \Delta_1 \qquad \lambda \bar{x}.\, h\, \overline{T} \rhd^{\blacksquare} U \diamond \Delta_2}{\Delta_c, \underline{r} =_d \lambda \bar{x}.\, h\, \overline{T} \rhd \Delta_1, \Delta_2, \underline{r} =_d U}(3)$$

$$\boxed{T \rhd^{\bigcirc} N \diamond \Delta}$$

$$\frac{h\, \overline{T} \rhd^{\blacksquare} U \diamond \Delta \qquad \bar{z} = FV(h\, \overline{T})}{\lambda \bar{x}.\, h\, \overline{T} \rhd^{\bigcirc} \lambda \bar{x}.\, \underline{r}\, \bar{z} \diamond (\Delta, \underline{r} =_d \lambda \bar{z}.\, U)}(\underline{r}\ \text{fresh})(4) \qquad \frac{}{\lambda \bar{x}.\, H\, \bar{y} \rhd \lambda \bar{x}.\, H^{\bigcirc}\, \bar{y} \diamond [\,]}(5)$$

$$\frac{}{\lambda \bar{x}.\, \underline{r}\, \bar{y} \rhd \lambda \bar{x}.\, \underline{r}\, \bar{y} \diamond [\,]}(6)$$

$$\boxed{T \rhd^{\blacksquare} U \diamond \Delta}$$

$$\frac{\forall_{i, 1 \le i \le n}. T_i \rhd^{\bigcirc} N_i \diamond \Delta_i}{\lambda \bar{x}.\, h\, T_1\, \ldots\, T_n \rhd^{\blacksquare} \lambda \bar{x}.\, h\, N_1\, \ldots\, N_n \diamond (\Delta_1, \ldots, \Delta_n)}(7) \qquad (\text{No rules for } T = \lambda \bar{x}.\, H\, \bar{y} \text{ or } T = \lambda \bar{x}.\, \underline{r}\, \bar{y})$$

In rule (4), we ensure that the body of a recursive definition is always closed by abstracting over all free variables when creating a recursive definition.

As an example, we show how the unification problem Δ_c in the previous section is translated into Δ. Notice that the left-hand side of the unification equation $get\,(\lambda x.\ get\,(\lambda y.\ S\ x\,y))$ is moved into a recursive definition $\underline{r_1}$ according to the definition, and the body of *even* is extracted to $\underline{r_3}$:

$$\begin{array}{l}
\Delta_c = \{ \\
\ get\,(\lambda x.\ get\,(\lambda y. S\,x\,y)) \doteq \underline{odd}, \\
\\
\\
\ \underline{odd} =_d get\,(\lambda x.\ \underline{even}), \\
\ \underline{even} =_d get\,(\lambda x.\ put\ x\,\underline{odd}) \\
\}
\end{array}$$

$$\begin{array}{l}
\Delta = \{ \\
\ \underline{r_1} \doteq \underline{odd}, \\
\ \underline{r_1} =_d get\,(\lambda x.\ \underline{r_2}\,x), \\
\ \underline{r_2} =_d \lambda x.\ get\,(\lambda y. S^{\circ}\,x\,y), \\
\ \underline{odd} =_d get\,(\lambda x.\ \underline{even}), \\
\ \underline{even} =_d get\,(\lambda x.\ \underline{r_3}\,x), \\
\ \underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\,x)\ \underline{odd}, \\
\ \underline{r_4} =_d \lambda x. x \\
\}
\end{array}$$

As with the first-order case, we define the definitional expansion at depth k for a recursive or a contractive term mutually recursively, $\mathsf{exp}^{\Delta}_{(k)}(U) = M_{\perp}$ and $\mathsf{exp}^{\Delta}_{(k)}(N) = M_{\perp}$. We also take the liberty to omit writing Δ if it remains unchanged throughout and is not referenced:

$$\begin{aligned}
&\mathsf{exp}_{(0)}(U) = \perp \\
&\mathsf{exp}_{(k+1)}(\lambda\bar{x}. H^{\blacksquare}\ \bar{y}) = \lambda\bar{x}. H^{\blacksquare}\bar{y} \\
&\mathsf{exp}_{(k+1)}(\lambda\bar{x}. h\,T_1\ \ldots\ T_n) = \lambda\bar{x}. h\,(\mathsf{exp}_{(k)}(T_1))\ \ldots\ (\mathsf{exp}_{(k)}(T_n)) \\
&\mathsf{exp}_{(0)}(N) = \perp \\
&\mathsf{exp}_{(k+1)}(\lambda\bar{x}. H^{\circ}\ \bar{y}) = \lambda\bar{x}. H^{\circ}\bar{y} \\
&\mathsf{exp}^{\Delta}_{(k+1)}(\lambda\bar{x}. \underline{r}\,\bar{y}) = \mathsf{exp}^{\Delta}_{(k+1)}(\lambda\bar{x}.\ [\bar{y}/\bar{z}](h\,\overline{T}))\ \mathit{if}\ \underline{r} =_d \lambda\bar{z}. h\,\overline{T} \in \Delta.
\end{aligned}$$

We show that the translation preserves the definitional expansion of arbitrary depth.

Theorem 3.1. *We have*

(1) *If* $T \rhd^{\blacksquare} U \diamond \Delta_2$ *and* $\mathsf{exp}^{\Delta_c}_{(k)}(\lambda\bar{x}. \underline{s}\,\bar{y}) = \mathsf{exp}^{\Delta_1}_{(k)}(\lambda\bar{x}. \underline{s}\,\bar{y})$ *for all* $\underline{s}$ *occurring in* T, *then* $\mathsf{exp}^{\Delta_c}_{(k)}(T) = \mathsf{exp}^{\Delta_1,\Delta_2}_{(k)}(U)$.

(2) *If* $T \rhd^{\circ} N \diamond \Delta_2$ *and* $\mathsf{exp}^{\Delta_c}_{(k)}(\lambda\bar{x}. \underline{s}\,\bar{y}) = \mathsf{exp}^{\Delta_1}_{(k)}(\lambda\bar{x}. \underline{s}\,\bar{y})$ *for all* $\underline{s}$ *occurring in* T, *then* $\mathsf{exp}^{\Delta_c}_{(k)}(T) = \mathsf{exp}^{\Delta_1,\Delta_2}_{(k)}(N)$.

Proof. Simultaneous induction on (k, T), where (2) may appeal to (1) without a decrease in size.

We show the case for rule (4) as an example. The premise $h\,\overline{T} \rhd^{\blacksquare} U \diamond \Delta$ implies, by the induction hypothesis, that $\mathsf{exp}^{\Delta_c}_{(k)}(h\,\overline{T}) = \mathsf{exp}^{\Delta_1,\Delta}_{(k)}(U)$. Let $\bar{z} = FV(h\,\overline{T})$, we want to show that $\mathsf{exp}^{\Delta_c}_{(k)}(\lambda\bar{x}. h\,\overline{T}) = \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d\lambda\bar{z}.\,U}_{(k)}(\lambda\bar{x}. \underline{r}\,\bar{z})$. Observe that both $\mathsf{exp}^{\Delta_c}_{(k)}$ and $\mathsf{exp}^{\Delta}_{(k)}$ commute with λ-abstractions, and that $\mathsf{exp}^{\Delta}_{(k)}$ is fixed by the recursive definitions for recursion constants that occur in the argument, i.e.,

$\mathsf{exp}^{\Delta_1,\Delta}_{(k)}(U) = \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d\lambda\bar{z}.\,U}_{(k)}(U)$. We have

$$\begin{array}{ll}
\mathsf{exp}^{\Delta_c}_{(k)}(\lambda\bar{x}.\,h\,\overline{T}) & \\
= \lambda\bar{x}.\ \mathsf{exp}^{\Delta_c}_{(k)}(h\,\overline{T}) & \text{(by definition)} \\
= \lambda\bar{x}.\ \mathsf{exp}^{\Delta_1,\Delta}_{(k)}(U) & \text{(by the induction hypothesis)} \\
= \lambda\bar{x}.\ \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d\lambda\bar{z}.U}_{(k)}(U) & \text{(because } \underline{r} \text{ is fresh)} \\
= \lambda\bar{x}.\ \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d\lambda\bar{z}.\,U}_{(k)}(\underline{r}\,\bar{z}) & \text{(by definition)} \\
= \mathsf{exp}^{\Delta_1,\Delta,\underline{r}=_d\lambda\bar{z}.\,U}_{(k)}(\lambda\bar{x}.\,\underline{r}\,\bar{z}) & \text{(by definition).}
\end{array}$$

□

Corollary 3.2. *If $\Delta_c \rhd \Delta$, then every equation in Δ_c corresponds to an equation in Δ with an equal definitional denotation, and every recursive definition in Δ_c corresponds to a recursive definition in Δ.*

Proof. Directly by structural induction over $\Delta_c \rhd \Delta$. □

3.3 Term Equality and Unifiers

The core ideas for term equality, substitution, and unifiers for the higher-order case are similar to the first-order case. The main technical difference from the first-order case is that $H^m\,\bar{x} \doteq U$ is to be interpreted as H^m standing for $\lambda\bar{x}.\,U$. We will only repeat the most essential definitions.

Informally two terms are equal if they have the same definitional expansion. Formally, M is *equal* to M' in a context Δ (i.e., $M \doteq M'$ holds) if for all k, $\mathsf{exp}^{\Delta}_{(k)}(M) = \mathsf{exp}^{\Delta}_{(k)}(M')$.

There are two kinds of substitutions in the higher-order case, substitutions for ordinary variables and substitutions for unification metavariables. Due to the pattern restriction, the only substitutions for ordinary variables are simultaneous variable renaming that we have seen, and are written in the notation $[\bar{y}/\bar{x}]M$. Substitutions for unification metavariables remain a special form of unification contexts, but they are now higher-order.

A *substitution* is a contradiction-free unification context where the left-hand side of each unification equation is a unique unification metavariable followed by a list of bound variables, which is a superset of the free variables occurring on the right-hand side of that equation. Intuitively, the variables following a unification metavariable serve as λ-binders for its value (on the right-hand side). All unification equations in a substitution are of the form $H^{\blacksquare}\,\bar{x} \doteq U$ or $H^{\bigcirc}\,\bar{x} \doteq N$, where $FV(U) \subseteq \bar{x}$ and $FV(N) \subseteq \bar{x}$. The equation $H^{\blacksquare}\,\bar{x} \doteq U$ or $H^{\bigcirc}\,\bar{x} \doteq N$ is called a *substitution equation* for $H^{\bigcirc}$ or $H^{\blacksquare}$ in Γ. The intuitive meaning of a substitution equation $H^m\,\bar{x} \doteq M$ is that H^m "stands for" $\lambda\bar{x}.\,M$. Since terms are all written in their η-long-form, U or N on the right-hand side of the unification equation should not contain top-level λ-bindings. The set of unification metavariables that occur on the left-hand sides of the unification equations in a substitution Γ is called the *domain* of the substitution and is denoted $\mathsf{dom}(\Gamma)$. Two substitutions Γ and Γ' are equal if $\mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma')$, and for all $H^m \in \mathsf{dom}(\Gamma)$, $\mathsf{exp}^{\Gamma}_{(k)}((\lambda\bar{x}.\,H^m\,\bar{x})[\Gamma]) = \mathsf{exp}^{\Gamma'}_{(k)}((\lambda\bar{x}.\,H^m\bar{x})[\Gamma])$, for all k, where $\lambda\bar{x}.\,H^m\,\bar{x}$ is the η-long-form of H^m according to its simple type.

As an example, the Γ, Γ', and Γ'' below are all equal substitutions with the domain $\{S^{\bigcirc}\}$. The main differences are highlighted in blue.

$$\begin{array}{l|l|l}
\Gamma = \{ & \Gamma' = \{ & \Gamma'' = \{ \\
\ S^{\bigcirc}\,z\,w \doteq r_3\ w, & \ S^{\bigcirc}\,y\,u \doteq \underline{r_3}\,u, & \ S^{\bigcirc}\,w\,z \doteq \underline{r_1}\,z, \\
\ \underline{odd} =_d get\,(\lambda x.\,\underline{even}), & \ \underline{odd} =_d get\,(\lambda x.\,\underline{even}), & \ \underline{odd} =_d get\,(\lambda x.\,\underline{even}), \\
\ \underline{even} =_d get\,(\lambda x.\,\underline{r_3}\,x), & \ \underline{even} =_d get\,(\lambda x.\,\underline{r_3}\,x), & \ \underline{even} =_d get\,(\lambda x.\,\underline{r_1}\,x), \\
\ \underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\,x)\ \underline{odd}, & \ \underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\,x)\ \underline{odd}, & \ \underline{r_1} =_d \lambda x.\ put\,(\underline{r_5}\,x)\ \underline{odd}, \\
\ \underline{r_4} =_d \lambda x.\,x & \ \underline{r_4} =_d \lambda x.\,x & \ \underline{r_5} =_d \lambda x.\,x \\
\} & \} & \}
\end{array}$$

We write $U[\Gamma]$ and $N[\Gamma]$ for applying the substitution (for unification metavariables) to terms. They are defined as follows:

$$
\begin{aligned}
(\lambda\bar{x}.h\, N_1\, \dots\, N_n)[\Gamma] &= \lambda\bar{x}.\, h\,(N_1[\Gamma])\, \dots\, (N_n[\Gamma])\\
(\lambda\bar{x}.\, H^{\blacksquare}\, \bar{y})[\Gamma] &= \begin{cases} \lambda\bar{x}.\, [\bar{y}/\bar{z}]U'. & \text{if } H^{\blacksquare}\, \bar{z} \doteq U' \in \mathsf{eqs}(\Gamma)\\ \lambda\bar{x}.\, H^{\blacksquare}\, \bar{y} & \text{otherwise}\end{cases}\\
(\lambda\bar{x}.\, \underline{r}\, \bar{y})[\Gamma] &= \lambda\bar{x}.\, \underline{r}\, \bar{y}\\
(\lambda\bar{x}.\, H^{\bigcirc}\, \bar{y})[\Gamma] &= \begin{cases} \lambda\bar{x}.\, [\bar{y}/\bar{z}]N' & \text{if } H^{\bigcirc}\, \bar{z} \doteq N' \in \mathsf{eqs}(\Gamma)\\ \lambda\bar{x}.\, H^{\bigcirc}\, \bar{y} & \text{otherwise.}\end{cases}
\end{aligned}
$$

It is worth noting that the substitution commutes with λ-abstractions so that $(\lambda\bar{x}.\, M)[\Gamma] = \lambda\bar{x}.\, (M[\Gamma])$. Substitution also commutes with simultaneous variable renaming so that $([\bar{y}/\bar{x}]M)[\Gamma] = [\bar{y}/\bar{x}](M[\Gamma])$. We can show both claims by induction on the structure of M, and the intuition is that substitutions are "closed" substitutions for unification metavariables.

The application of a substitution Γ to a unification context Δ, the application of a substitution Γ_2 to another substitution Γ_1, and the composition of substitutions $\Gamma_1 \circ \Gamma_2$ (apply Γ_1 and then Γ_2) are analogous counterparts of their first-order definitions:

$$
\begin{aligned}
\Delta[\Gamma] &= \mathsf{defs}(\Gamma), \{M[\Gamma] \doteq M'[\Gamma] \mid M \doteq M' \in \mathsf{eqs}(\Delta)\}, \{\underline{r} =_d U[\Gamma] \mid \underline{r} =_d U \in \mathsf{defs}(\Delta)\}\\
\Gamma_1[\Gamma_2] &= \mathsf{defs}(\Gamma_2), \{H^m\, \bar{x} \doteq M'[\Gamma_2] \mid H^m\, \bar{x} \doteq M' \in \mathsf{eqs}(\Gamma_1)\}, \{\underline{r} =_d U[\Gamma_2] \mid \underline{r} =_d U \in \mathsf{defs}(\Gamma_1)\}\\
\Gamma_1 \circ \Gamma_2 &= (\Gamma_1[\Gamma_2]), \{H^m\, \bar{x} \doteq M \mid H^m\, \bar{x} \doteq M \in \mathsf{eqs}(\Gamma_2) \wedge H^m \notin \mathsf{dom}(\Gamma_1)\}.
\end{aligned}
$$

We repeat the definition of unifiers and the most general unifier for the higher-order case. A *unifier* for a contradiction-free unification context Δ is a substitution Γ such that $UV(\Delta) = \mathsf{dom}(\Gamma)$, and every equation in $\Delta[\Gamma]$ holds. A unification context Δ with $\mathsf{contra} \in \Delta$ has no unifiers. A unifier Γ_1 is *more general* than another unifier Γ_2 if there is a substitution Γ' such that $\Gamma_1 \circ \Gamma' = \Gamma_2$.

As an example, given Γ and Δ defined below, Γ is a unifier of Δ, because every equation in $\Delta[\Gamma]$ holds. The main changes are highlighted in blue. Notice that when carrying out the substitution, the duplicate recursion constants in Γ are renamed by adding a prime (′) sign.

$$
\begin{aligned}
&\Gamma = \{\\
&\quad S^{\bigcirc}\, z\, w \doteq \underline{r_3}\, w,\\
&\quad \underline{odd} =_d get\,(\lambda x.\, \underline{even}),\\
&\quad \underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x),\\
&\quad \underline{r_3} =_d \lambda x.\, put\,(\underline{r_4}\, x)\, \underline{odd},\\
&\quad \underline{r_4} =_d \lambda x.\, x\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\Delta = \{\\
&\quad \underline{r_1} \doteq \underline{odd},\\
&\quad \underline{r_1} =_d get\,(\lambda x.\, \underline{r_2}\, x),\\
&\quad \underline{r_2} =_d \lambda x.\, get\,(\lambda y.\, S^{\bigcirc}\, x\, y),\\
&\quad \underline{odd} =_d get\,(\lambda x.\, \underline{even}),\\
&\quad \underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x),\\
&\quad \underline{r_3} =_d \lambda x.\, put\,(\underline{r_4}\, x)\, \underline{odd},\\
&\quad \underline{r_4} =_d \lambda x.\, x\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\Delta[\Gamma] = \{\\
&\quad \underline{r_1} \doteq \underline{odd},\\
&\quad \underline{r_1} =_d get\,(\lambda x.\, \underline{r_2}\, x),\\
&\quad \underline{r_2} =_d \lambda x.\, get\,(\lambda y.\, \underline{r_3{}'}\, y),\\
&\quad \underline{odd} =_d get\,(\lambda x.\, \underline{even}),\\
&\quad \underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x),\\
&\quad \underline{r_3} =_d \lambda x.\, put\,(\underline{r_4}\, x)\, \underline{odd},\\
&\quad \underline{r_4} =_d \lambda x.\, x\\
&\quad \underline{odd'} =_d get\,(\lambda x.\, \underline{even'}),\\
&\quad \underline{even'} =_d get\,(\lambda x.\, \underline{r_3{}'}\, x),\\
&\quad \underline{r_3{}'} =_d \lambda x.\, put\,(\underline{r_4{}'}\, x)\, \underline{odd'},\\
&\quad \underline{r_4{}'} =_d \lambda x.\, x\\
&\}
\end{aligned}
$$

3.4 The Algorithm

We now present the saturation-based algorithm for higher-order rational terms.

In the first-order case, a unification metavariable is resolved if there is an equation between the unification metavariable and either a recursion constant or a term with a constructor head. In the higher-order case, we have to consider the binding structure, the resolution must only have free variables that appear in the arguments to the unification metavariable. Also, a unification metavariable may be resolved by another unification metavariable that has strictly fewer arguments. A contractive unification metavariable $H^{\blacksquare}$ is *resolved* if (a) there exists a unification equation $H^{\blacksquare}\,\bar{y} \doteq h\,\bar{z}$ (or its symmetry) and $\bar{z} \subseteq \bar{y}$, or (b) there exists a unification equation $H^{\blacksquare}\,\bar{y} \doteq G^{\blacksquare}\,\bar{w}$ (or its symmetry) with $\bar{w} \subsetneq \bar{y}$.[2] $H^{\blacksquare}$ is unresolved otherwise, and is denoted by the judgment $H^{\blacksquare}$ $\mathsf{unresolved}$. Similarly, a contractive unification metavariable $H^{\bigcirc}$ is *resolved* if there exists an equation $H^{\bigcirc}\,\bar{y} \doteq \underline{r}\,\bar{z}$ (or its symmetry) and $\bar{z} \subseteq \bar{y}$, or there exists an equation $H^{\bigcirc}\,\bar{y} \doteq G^{\bigcirc}\,\bar{w}$ (or its symmetry) with $\bar{w} \subsetneq \bar{y}$. $H^{\bigcirc}$ is unresolved otherwise, and is denoted by the judgment $H^{\bigcirc}$ $\mathsf{unresolved}$.

We saturate the unification context Δ using the rules defined in Figure 1. The saturation rules preserve the definition of all recursion constants. Once saturated, a unifier can be constructed easily. The presence of the symbol contra in a unification context indicates that the unification context has no unifiers.

We use the concept of a parameter to ensure the termination of the saturation-based algorithm. The parameters are indicated by bracketed existential quantifiers $(\exists X)$ where X is a parameter that stands for a variable, a recursion constant, a unification metavariable, or a list of those. The new equations or definitions under the existential quantification subsume any instantiation of the equation or definition [McLaughlin and Pfenning, 2009]. The parameters ensure freshness and nonredundancy: When new variables, recursion constants, or unification metavariables are introduced by one of the rules (freshness), the existential quantification ensures that the rule applies (thus the conclusion equations or definitions are created) only if there does not exist any instantiation of the conclusion equations or definitions in the unification context (nonredundancy). For example, $(\exists x, y)\ c\,x \doteq d\,y$ means that we pick globally fresh variables x, y, ensure that there is no equation of the form $c\,z \doteq d\,w$ in the unification context for any variable z and w, as this is a renaming of $c\,x = d\,y$, and then add the equation $c\,x \doteq d\,y$ to the unification context.

We extend the notation of using overlines to denote lists of unification metavariables (possibly with arguments) and operations on them. A list of unification metavariables is written $\overline{G^{\bigcirc}}$ or $\overline{G^{\blacksquare}}$. Also, we write $\overline{G^{\bigcirc}\,\overline{x}}$ to denote the list of applications where each unification metavariable is applied to $\overline{x}$. For example, $c\,\overline{G^{\bigcirc}\,\overline{x}}$ denotes $c\ (G^{\bigcirc}_1\,x_1\ \ldots\ x_m)\ \ldots\ (G^{\bigcirc}_n\,x_1\ \ldots\ x_m)$. $h\,\eta\mathsf{exp}(\overline{G^{\bigcirc}\,\overline{x}})$ denotes a term with head h whose arguments are the result of applying top-level η-expansions to all terms in $\overline{G^{\bigcirc}\,\overline{x}}$ according to the simple type of h. For example, $c\,\eta\mathsf{exp}(\overline{G^{\bigcirc}\,\overline{x}})$[3] denotes a term of the form $c\ (\lambda\overline{w}_1.\,G^{\bigcirc}_1\,\overline{x}\,\overline{w}_1)\ \ldots\ (\lambda\overline{w}_n.\,G^{\bigcirc}_n\,\overline{x}\,\overline{w}_n)$, i.e.,

$$c\ (\lambda w_{1,1}.\ldots\lambda w_{1,l_1}.\,G^{\bigcirc}_1\,x_1\ \ldots\ x_m\,w_{1,1}\ \ldots\ w_{1,l_1})\ \ldots\ (\lambda w_{n,1}.\ldots\lambda w_{n,l_n}.\,G^{\bigcirc}_n\,x_1\ \ldots\ x_m\,w_{n,1}\ \ldots\ w_{n,l_n}).$$

Rule (U-INST)(N-INST) instantiate λ-abstractions, but only when there are no instantiations that are currently present in the context. Rule (IMIT) is called imitation by Huet [1975] because $H^{\blacksquare}$ mimics the behavior of the term $c\,\overline{N}$ on the right. Rules (PROJ-F)(PROJ) are called projections because $H^{\blacksquare}$ projects its ith argument its head. Rule (PRUNE) prunes the variables that are not in common from both $H^{\bigcirc}$ and $\underline{r}$, by creating a recursive definition $\underline{t}$, whose body $G^{\blacksquare}\,\bar{w}$ can only use variables $\bar{w}$ that are common to $H^{\bigcirc}$ and $\underline{r}$. Rules (FF-D)(FF-S)[4] also serve a similar effect of pruning by removing the extra variables from the arguments to unification metavariables that cannot be used. It resolves H^m, or G^m, or both, by equating them to a common unification metavariable F^m

[2] $\bar{w} \subsetneq \bar{y}$ means $\bar{w}$ is a proper subset of $\bar{y}$. For example, $H^{\blacksquare}\ y \doteq G^{\blacksquare}\ y$ is not a resolution equation, but $H^{\blacksquare}\,y \doteq G^{\blacksquare}$ is, while either equation may appear as a substitution equation for $H^{\blacksquare}$ in a unifier.

[3] We should remark that the non-η-expanded version $c\,\overline{G^{\bigcirc}\,\overline{x}}$ should not appear in the unification context, since we write everything in the η-long form (except the arguments to recursion constants and unification metavariables). Thus, $h\,\eta\mathsf{exp}(\overline{G^{\bigcirc}\,\overline{x}})$ always appears in a conclusion where $\overline{G^{\bigcirc}}$ is fresh (bound by the existential quantifier $(\exists\overline{G^{\bigcirc}})$).

[4] (FF-D) suggests flexible-flexible pairs with different unification metavariables and (FF-S) suggests flexible-flexible pairs with the same unification metavariable. See Huet [1975] for the distinction between flexible and rigid terms.

Structural Rules

(U-INST)
$$\frac{\lambda \bar{x}.\, U \doteq \lambda \bar{x}.\, U'}{(\exists \bar{x})\, U \doteq U'}$$

(N-INST)
$$\frac{\lambda \bar{x}.\, N \doteq \lambda \bar{x}.\, N'}{(\exists \bar{x})\, N \doteq N'}$$

(SIMP-F1)
$$\frac{x\,\overline{N} \doteq c\,\overline{N'}}{\text{contra}}$$

(SIMP-F2)
$$\frac{x\,\overline{N} \doteq y\,\overline{N'}}{\text{contra}}\,(x \neq y)$$

(SIMP-F3)
$$\frac{c\,\overline{N} \doteq d\,\overline{N'}}{\text{contra}}\,(c \neq d)$$

(SIMP)
$$\frac{h\,N_1\,\ldots N_n \doteq h\,N'_1\,\ldots N'_n}{N_1 \doteq N'_1, \ldots, N_n \doteq N'_n}$$

(PROJ-F)
$$\frac{H^{\blacksquare}\,\bar{y} \doteq z_i\,\overline{N} \qquad z_i \notin \bar{y}}{\text{contra}}$$

Resolution Rules

(IMIT)
$$\frac{H^{\blacksquare}\,\bar{y} \doteq c\,\overline{N} \qquad H^{\blacksquare}\ \text{unresolved}}{(\exists \overline{G^{\bigcirc}})\, H^{\blacksquare}\,\bar{y} \doteq c\,\eta\text{exp}(\overline{G^{\bigcirc}\,\bar{y}})}$$

(PROJ)
$$\frac{H^{\blacksquare}\,\bar{y} \doteq y_i\,\overline{N} \qquad y_i \in \bar{y} \qquad H^{\blacksquare}\ \text{unresolved}}{(\exists \overline{G^{\bigcirc}})\, H^{\blacksquare}\,\bar{y} \doteq y_i\,\eta\text{exp}(\overline{G^{\bigcirc}\,\bar{y}})}$$

(PRUNE)
$$\frac{H^{\bigcirc}\,\bar{y} \doteq \underline{r}\,\bar{x} \qquad \bar{x} \not\subseteq \bar{y} \qquad \bar{w} = \bar{x} \cap \bar{y} \qquad H^{\bigcirc}\ \text{unresolved}}{(\exists t)(\exists G^{\blacksquare})\, H^{\bigcirc}\,\bar{y} \doteq \underline{t}\,\bar{w}, \underline{r}\,\bar{x} \doteq \underline{t}\,\bar{w}, \underline{t} =_d \lambda \bar{w}.G^{\blacksquare}\,\bar{w}}$$

(FF-D)
$$\frac{\begin{array}{c} G^m\,\bar{x} \doteq H^m\,\bar{y} \\ G^m \neq H^m \qquad \bar{z} = \bar{x} \cap \bar{y} \qquad \bar{x} \not\subseteq \bar{y} \wedge \bar{y} \not\subseteq \bar{x} \qquad H^m\ \text{unresolved} \vee G^m\ \text{unresolved} \end{array}}{(\exists F^m)\, G^m\,\bar{x} \doteq F^m\,\bar{z}, H^m\,\bar{y} \doteq F^m\,\bar{z}}\,(m \in \{\blacksquare, \bigcirc\})$$

(FF-S)
$$\frac{\begin{array}{c} G^m\,\bar{x} \doteq G^m\,\bar{y} \\ \bar{x} = x_1\,\ldots\,x_n \qquad \bar{y} = y_1\,\ldots\,y_n \qquad \bar{z} = \cup_i \{x_i \mid x_i = y_i\} \qquad \bar{x} \neq \bar{y} \qquad G^m\ \text{unresolved} \end{array}}{(\exists F^m)\, G^m\,\bar{x} \doteq F^m\,\bar{z}, G^m\,\bar{y} \doteq F^m\,\bar{z}}\,(m \in \{\blacksquare, \bigcirc\})$$

Expansion, Consistency, Symmetry, and Transitivity

(REC-EXP)
$$\frac{\underline{r}\,\bar{x} \doteq \underline{s}\,\bar{y} \qquad \underline{r} =_d \lambda \bar{z}.\, U_1 \qquad \underline{s} =_d \lambda \bar{w}.\, U_2}{[\bar{x}/\bar{z}]U_1 \doteq [\bar{y}/\bar{w}]U_2}$$

(U-AGREE)
$$\frac{H^{\blacksquare}\,\bar{x} \doteq U_1 \qquad H^{\blacksquare}\,\bar{y} \doteq U_2 \qquad FV(U_1) \subseteq \bar{x} \qquad FV(U_2) \subseteq \bar{y}}{U_1 \doteq [\bar{x}/\bar{y}]U_2}$$

(N-AGREE)
$$\frac{H^{\bigcirc}\,\bar{x} \doteq N_1 \qquad H^{\bigcirc}\,\bar{y} \doteq N_2 \qquad FV(N_1) \subseteq \bar{x} \qquad FV(N_2) \subseteq \bar{y}}{N_1 \doteq [\bar{x}/\bar{y}]N_2}$$

(U-SYM)
$$\frac{U \doteq U'}{U' \doteq U}$$

(N-SYM)
$$\frac{N \doteq N'}{N' \doteq N}$$

(U-TRANS)
$$\frac{U_1 \doteq U_2 \qquad U_2 \doteq U_3}{U_1 \doteq U_3}$$

(N-TRANS)
$$\frac{N_1 \doteq N_2 \qquad N_2 \doteq N_3}{N_1 \doteq N_3}$$

Fig. 1. Unification rules.

whose arguments $\bar{z}$ is a strict subset of both $\bar{x}$ and $\bar{y}$. Rule (REC-EXP) unfolds recursive definitions and compares them for equality. Rules (U-AGREE)(N-AGREE) ensure that any two resolutions of a unification metavariable are consistent.

We give an example of how the algorithm operates on the stream processor unification context, now denoted Δ_7 (Equations and Definitions (1)–(7)). At each step, we show some additional equations or definitions and the ways they are obtained. We omit the final uninteresting steps when only symmetry and transitivity rules can be applied:

(1) $\underline{r_1} \doteq \underline{odd}$ given
(2) $\underline{r_1} =_d get\,(\lambda x.\, \underline{r_2}\, x)$
(3) $\underline{r_2} =_d \lambda x.\ get\,(\lambda y.\, S^{\bigcirc}\, x\, y)$
(4) $\underline{odd} =_d get\,(\lambda x.\, \underline{even})$
(5) $\underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x)$
(6) $\underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\, x)\, \underline{odd}$
(7) $\underline{r_4} =_d \lambda x.\, x$
(8) $get\,(\lambda x.\, \underline{r_2}\, x) \doteq get\,(\lambda x.\, \underline{even})$ by Rule (REC-EXP) on (1), (2), and (4)
(9) $\lambda x.\, \underline{r_2}\, x \doteq \lambda x.\, \underline{even}$ by Rule (SIMP) on (8)
(10) $\underline{r_2}\, z \doteq \underline{even}$ by Rule (N-INST) on (9), and we verify that there does not exist any equation $(\exists x)\ \underline{r_2}\, x \doteq \underline{even}$ in the context Δ_9.
(11) $get\,(\lambda y.\, S^{\bigcirc}\, z\, y) \doteq get\,(\lambda x.\, \underline{r_3}\, x)$ by Rule (REC-EXP) on (10), (3), and (5)
(12) $\lambda y.\, S^{\bigcirc}\, z\, y \doteq \lambda x.\, \underline{r_3}\, x$ by Rule (SIMP) on (11)
(13) $S^{\bigcirc}\, z\, w \doteq \underline{r_3}\, w$ by Rule (N-INST) on (12), and we verify that there does not exist any equation $(\exists x)\ S^{\bigcirc}\, z\, x \doteq \underline{r_3}\, x$ in the context Δ_{12}
(14) ... by Rules (U-SYM)(N-SYM)(U-TRANS)(N-TRANS) ...

The above example used only the structural rules and expansion rules. More examples will be given in Section 3.6.

3.5 Saturated Unification Contexts

We now describe how a substitution $\Gamma = \mathsf{unif}(\Delta)$ can be constructed from a contradiction-free unification context Δ. The construction in the higher-order case takes the pattern variables following a unification metavariable into account. We will later show that if Δ is a saturated unification context, then $\mathsf{unif}(\Delta)$ is the most general unifier for Δ.

Given any unification context, every unification metavariable $H^{\blacksquare}$ or $H^{\bigcirc}$ is either resolved or unresolved. Suppose $H^{\blacksquare}$ or $H^{\bigcirc}$ is resolved:

(1) If $H^{\blacksquare}$ is resolved, then there exists an equation $H^{\blacksquare}\ \bar{y} \doteq h\,\bar{z}$ or $H^{\blacksquare}\ \bar{y} \doteq G^{\blacksquare}\ \bar{w}$, with $FV(h\,\bar{z}) \subseteq \bar{y}$ and $\bar{w} \subsetneq \bar{y}$. The equation $H^{\blacksquare}\ \bar{y} \doteq h\,\bar{z}$ or $H^{\blacksquare}\ \bar{y} \doteq G^{\blacksquare}\ \bar{w}$ is called a *resolution equation* for $H^{\blacksquare}$.
(2) If $H^{\bigcirc}$ is resolved, then there exists an equation $H^{\bigcirc}\ \bar{y} \doteq \underline{r}\,\bar{z}$ or $H^{\bigcirc}\ \bar{y} \doteq G^{\bigcirc}\ \bar{w}$, with $FV(\underline{r}\,\bar{z}) \subseteq \bar{y}$ and $\bar{w} \subsetneq \bar{y}$. The equation $H^{\bigcirc}\ \bar{y} \doteq \underline{r}\,\bar{z}$ or $H^{\bigcirc}\ \bar{y} \doteq G^{\bigcirc}\ \bar{w}$ is called a *resolution equation* for $H^{\bigcirc}$.

There may be multiple such equations, but we consistently pick a resolution equation (called *the* resolution equation, or simply the *resolution*[5]) for each unification metavariable:

(1) Further, the operation of *replacing occurrences of a (resolved) contractive unification metavariable $H^{\blacksquare}$ by its resolution in a term M* is defined to be M with occurrences of $H^{\blacksquare}\ \bar{x}$ replaced by $[\bar{x}/\bar{y}](h\,\bar{z})$ or $[\bar{x}/\bar{y}](G^{\blacksquare}\ \bar{w})$ according to the resolution equation $H^{\blacksquare}\ \bar{y} \doteq h\,\bar{z}$ or $H^{\blacksquare}\ \bar{y} \doteq G^{\blacksquare}\ \bar{w}$.

[5] In the higher-order case, resolutions for unification metavariables are in the form of equations, whereas in the first-order case, the resolutions are terms.

(2) Similarly, the operation of *replacing occurrences of a (resolved) recursive unification metavariable $H^{\bigcirc}$ by its resolution in a term M* is defined to be M with occurrences of $H^{\bigcirc}\,\bar{x}$ replaced by $[\bar{x}/\bar{y}](\underline{r}\,\bar{z})$ or $[\bar{x}/\bar{y}](G^{\bigcirc}\,\bar{w})$ according to the resolution equation $H^{\bigcirc}\,\bar{y} \doteq \underline{r}\,\bar{z}$ or $H^{\bigcirc}\,\bar{y} \doteq G^{\bigcirc}\,\bar{w}$.

Now suppose H^m ($m \in \{\blacksquare, \bigcirc\}$) is unresolved. Unresolved unification metavariables form equivalence classes related by $\doteq$. We pick a *representative unification metavariable* for each equivalence class, such that if G^m is the representative unification metavariable for the equivalence class of H^m, there is an equation $H^m\,\bar{y} \doteq G^m\,\bar{z}$, where $\bar{z}$ is a permutation of $\bar{y}$. This equation is called the *representative equation* for H^m. We pick the representative equation in such a way that the right-hand sides $G^m\,\bar{z}$ of all representative equations are equal for all unresolved unification metavariables in the same equivalence class, i.e., for any other unification metavariable F^m in the same equivalence class as H^m, $F^m\,\bar{w} \doteq G^m\,\bar{z}$ is picked (and $F^m\,\bar{w}' \doteq G^m\,\bar{z}'$ with $\bar{z}' \neq \bar{z}$ is not picked). The operation of *replacing occurrences of a (unresolved) unification metavariable H^m by its representative unification metavariable G^m in a term M* is defined to be M with occurrences of $H^m\,\bar{x}$ replaced by $[\bar{x}/\bar{y}](G^m\,\bar{z})$.

We construct the substitution $\Gamma = \mathsf{unif}(\Delta)$ for a contradiction-free context Δ as follows. The construction is very similar to the first-order case, except that now the replacement of unification metavariables uses variable renaming:

(1) Start with Γ containing all recursive definitions of Δ.
(2) For each resolved unification metavariable in $UV(\Delta)$, add to Γ the resolution equation of that unification metavariable.
(3) For each unresolved unification metavariable in $UV(\Delta)$, add to Γ the representative equation of that unification metavariable.
(4) Replace the occurrences of resolved unification metavariables in the right-hand sides and recursive definitions of Γ with their resolutions, and replace the occurrences of unresolved unification metavariables in the right-hand sides and recursive definitions of Γ with their representatives. Repeat this step until all unification metavariables in the right-hand sides and recursive definitions are representative unification metavariables for some equivalence class of unresolved unification metavariables.

As an example, we show how the unifier for Δ_{13} (Equations and Definitions (1)–(13) defined above in Section 3.4), $\Gamma = \mathsf{unif}(\Delta_{13})$ can be constructed. Major changes in each step are highlighted in blue:

(1) Initialize Γ_1 to all recursive definitions of Δ_{13}.
(2) Since $H^{\bigcirc}$ is resolved, we add its resolution equation to get Γ_2.
(3) There are no unresolved unification metavariables, we skip step (3).
(4) Replace occurrences of resolved unification metavariables with their resolutions to get Γ_3.

$$\begin{aligned}
&\Gamma_1 = \{\\
&\quad \underline{r_1} =_d \mathit{get}\,(\lambda x.\,\underline{r_2}\,x),\\
&\quad \underline{r_2} =_d \lambda x.\ \mathit{get}\,(\lambda y.\,S^{\bigcirc}\,x\,y),\\
&\quad \underline{\mathit{odd}} =_d \mathit{get}\,(\lambda x.\,\underline{\mathit{even}}),\\
&\quad \underline{\mathit{even}} =_d \mathit{get}\,(\lambda x.\,\underline{r_3}\,x),\\
&\quad \underline{r_3} =_d \lambda x.\ \mathit{put}\,(\underline{r_4}\,x)\ \underline{\mathit{odd}},\\
&\quad \underline{r_4} =_d \lambda x.\,x\\
&\}
\end{aligned}$$

$$\begin{aligned}
&\Gamma_2 = \{\\
&\quad S^{\bigcirc}\,z\,w \doteq \underline{r_3}\,w,\\
&\quad \underline{r_1} =_d \mathit{get}\,(\lambda x.\,\underline{r_2}\,x),\\
&\quad \underline{r_2} =_d \lambda x.\ \mathit{get}\,(\lambda y.\,S^{\bigcirc}\,x\,y),\\
&\quad \underline{\mathit{odd}} =_d \mathit{get}\,(\lambda x.\,\underline{\mathit{even}}),\\
&\quad \underline{\mathit{even}} =_d \mathit{get}\,(\lambda x.\,\underline{r_3}\,x),\\
&\quad \underline{r_3} =_d \lambda x.\ \mathit{put}\,(\underline{r_4}\,x)\ \underline{\mathit{odd}},\\
&\quad \underline{r_4} =_d \lambda x.\,x\\
&\}
\end{aligned}$$

$$\begin{aligned}
&\Gamma_3 = \{\\
&\quad S^{\bigcirc}\,z\,w \doteq \underline{r_3}\,w,\\
&\quad \underline{r_1} =_d \mathit{get}\,(\lambda x.\,\underline{r_2}\,x),\\
&\quad \underline{r_2} =_d \lambda x.\ \mathit{get}\,(\lambda y.\,\underline{r_3}\,y),\\
&\quad \underline{\mathit{odd}} =_d \mathit{get}\,(\lambda x.\,\underline{\mathit{even}}),\\
&\quad \underline{\mathit{even}} =_d \mathit{get}\,(\lambda x.\,\underline{r_3}\,x),\\
&\quad \underline{r_3} =_d \lambda x.\ \mathit{put}\,(\underline{r_4}\,x)\ \underline{\mathit{odd}},\\
&\quad \underline{r_4} =_d \lambda x.\,x\\
&\}
\end{aligned}$$

(5) Note that we may remove unused recursive definitions (r_1, r_2) to get an equivalent substitution Γ_4:
$\Gamma_4 = \{S^{\circ}\, z\, w \doteq \underline{r_3}\, w, \underline{odd} =_d get\,(\lambda x.\, \underline{even}),$
$\underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x), \underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\, x)\, \underline{odd}, \underline{r_4} =_d \lambda x.\, x\}.$

Note that the final substitution Γ_4 is equivalent to the following substitution Γ_c, written in the concrete syntax without flattened definitions. We can easily check that this is a unifier for the concrete unification context Δ_c in Section 3.1:
$\Gamma_c = \{S^{\circ}\, z\, w \doteq \underline{r_3}\, w, \underline{odd} =_d get\,(\lambda x.\ get\,(\lambda y.\, \underline{r_3}\, y)), \underline{r_3} =_d \lambda w.\ put\ w\ \underline{odd}\}.$

3.6 Additional Examples

Recall the definition for odd and even in Section 3.1:

```
odd : sp = get ([x] even).
even : sp = get ([x] put x odd).
```

We have seen an example of how unification can figure out the behavior of the stream processor S after reading two elements of the input, if it behaves the same as odd:

```
?- get ([x] get ([y] S x y)) = odd.
```

$$\Delta_c = \{get\,(\lambda x.\ get\,(\lambda y.\, S\, x\, y)) \doteq \underline{odd}, \underline{odd} =_d get\,(\lambda x.\, \underline{even}), \underline{even} =_d get\,(\lambda x.\ put\ x\ \underline{odd})\}.$$

3.6.1 An Example with No Solution. In the above example, S may depend on both numbers x and y that read from the input stream. We may restrict x to only use the number at index 0, by omitting y from the argument of S:

```
?- get ([x] get ([y] S x)) = odd.
```

$$\Delta_c = \{get\,(\lambda x.\ get\,(\lambda y.\, S\, x)) \doteq \underline{odd}, \underline{odd} =_d get\,(\lambda x.\, \underline{even}), \underline{even} =_d get\,(\lambda x.\ put\ x\ \underline{odd})\}.$$

The odd stream processor outputs an element at index 1, but S doesn't have access to y. This unification problem does not have a solution, and the algorithm eventually adds the symbol contra to the unification context. The first six steps (Equations and Definitions (1)–(13)) are similar to the previous example with changes marked in blue, but now Equation (13) is no longer a resolution equation for S°. Note that $\Delta_c \rhd \Delta_7$ (Equations and Definitions (1)–(7)):

(1) $\underline{r_1} \doteq \underline{odd}$ given
(2) $\underline{r_1} =_d get\,(\lambda x.\, \underline{r_2}\, x)$
(3) $\underline{r_2} =_d \lambda x.\ get\,(\lambda y.\, S^{\circ}\, x)$
(4) $\underline{odd} =_d get\,(\lambda x.\, \underline{even})$
(5) $\underline{even} =_d get\,(\lambda x.\, \underline{r_3}\, x)$
(6) $\underline{r_3} =_d \lambda x.\ put\,(\underline{r_4}\, x)\, \underline{odd}$
(7) $\underline{r_4} =_d \lambda x.\, x$
(8) $get\,(\lambda x.\, \underline{r_2}\, x) \doteq get\,(\lambda x.\, \underline{even})$ by Rule (REC-EXP) on (1), (2), and (4)
(9) $\lambda x.\, \underline{r_2}\, x \doteq \lambda x.\, \underline{even}$ by Rule (SIMP) on (8)
(10) $\underline{r_2}\, z \doteq \underline{even}$ by Rule (N-INST) on (9), and we verify that there does not exist any equation $(\exists x)\underline{r_2}\, x \doteq \underline{even}$ in the context Δ_9.
(11) $get\,(\lambda y.\, S^{\circ}\, z) \doteq get\,(\lambda x.\, \underline{r_3}\, x)$ by Rule (REC-EXP) on (10), (3), and (5)
(12) $\lambda y.\, S^{\circ}\, z \doteq \lambda x.\, \underline{r_3}\, x$ by Rule (SIMP) on (11)
(13) $S^{\circ}\, z \doteq \underline{r_3}\, w$ by Rule (N-INST) on (12), and we verify that there does not

exist any equation $(\exists x)\, S^{\bigcirc}\, z \doteq \underline{r_3}\, x$ in the context Δ_{12}.

(14) $S^{\bigcirc}\, z \doteq \underline{t}$ by Rule (PRUNE) on (13)
(15) $\underline{r_3}\, w \doteq \underline{t}$
(16) $\underline{t} =_d G^{\blacksquare}$
(17) $\mathit{put}\ (\underline{r_4}\, w)\ \underline{\mathit{odd}} \doteq G^{\blacksquare}$ by Rule (REC-EXP) on (15), (6), and (16)
(18) $G^{\blacksquare} \doteq \mathit{put}\ (\underline{r_4}\, w)\ \underline{\mathit{odd}}$ by Rule (U-SYM) on (17)
(19) $G^{\blacksquare} \doteq \mathit{put}\, F^{\bigcirc} H^{\bigcirc}$ by Rule (IMIT) on (17)
(20) $\mathit{put}\ (\underline{r_4}\, w)\ \underline{\mathit{odd}} \doteq \mathit{put}\, F^{\bigcirc} H^{\bigcirc}$ by Rule (U-TRANS) on (17) and (19)
(21) $\underline{r_4}\, w \doteq F^{\bigcirc}$ by Rule (SIMP) on (20)
(22) $\underline{\mathit{odd}} \doteq H^{\bigcirc}$
(23) $F^{\bigcirc} \doteq \underline{r_4}\, w$ by Rule (N-SYM) on (21)
(24) $F^{\bigcirc} \doteq \underline{s}$ by Rule (PRUNE) on (23)
(25) $\underline{r_4}\, w \doteq \underline{s}$
(26) $\underline{s} =_d F^{\blacksquare}$
(27) $w \doteq F^{\blacksquare}$ by Rule (REC-EXP) on (25), (7), and (26)
(28) $F^{\blacksquare} \doteq w$ by Rule (N-SYM) on (27)
(29) contra by Rule (PROJ-F) on (28).

We now consider some more problems that do not involve *odd* or *even*.

3.6.2 A Stream Processor that Keeps Producing Elements. For example, we may ask, what is a stream H that outputs the given element and continues as itself:

```
?- [x] put x (H x) = [x] H x.
```

We should have the following result, which says that `H` produces an argument and continues as itself:

```
H = [x] put x (H x).
```

Indeed, the algorithm is able to find this solution. Our unification algorithm correctly finds a recursive definition for $H^{\bigcirc}$, as seen below, with $\Delta_c \rhd \Delta_3$ (Equations and Definitions (1)–(3)):

$$\Delta_c = \{\lambda x.\ \mathit{put}\ x\, (H\, x) \doteq \lambda x. H\, x\}$$

(1) $\lambda x.\, \underline{r_1}\, x \doteq \lambda x. H^{\bigcirc}\, x$ given
(2) $\underline{r_1} =_d \lambda x.\ \mathit{put}\ (\underline{r_2}\, x)\ (H^{\bigcirc}\, x)$
(3) $\underline{r_2} =_d \lambda x. x$
(4) $\underline{r_1}\, z \doteq H^{\bigcirc} z$ by Rule (N-INST) on (1)
(5) $\ldots$ $\ldots$

$$\mathsf{unif}(\Delta_4) = \{H^{\bigcirc} z \doteq \underline{r_1}\, z, \underline{r_1} =_d \lambda x.\ \mathit{put}\ (\underline{r_2}\, x)\ (\underline{r_1}\, x), \underline{r_2} =_d \lambda x. x\}.$$

Note that this problem has no unifier under Miller's higher-order pattern unification algorithm [Miller, 1991] due to the failure of the occurs check on `H`.

3.6.3 A Stream Processor that Keeps Consuming Elements. Dually, consider a stream processor S that reads an element and continues as itself, with $\Delta_c \rhd \Delta_2$:

```
?- [x] get ([y] S y) = [x] S x.
```

The intuitive solution is that S reads an element and continues as itself:

```
S = [x] get ([y] S y).
```

The algorithm is able to find this solution:

$$\Delta_c = \{\lambda x.\ get\,(\lambda y.\, S\, y) \doteq \lambda x.\, S\, x\}$$

$$\begin{array}{ll}
(1)\ \lambda x.\, \underline{r_1}\, x \doteq \lambda x.\, S^{\bigcirc}\, x & \text{given} \\
(2)\ \underline{r_1} =_d \lambda x.\ get\,(\lambda y.\, S^{\bigcirc}\, y) & \\
(3)\ \underline{r_1}\, z \doteq S^{\bigcirc} z & \text{by Rule (N-INST) on (1)} \\
(4)\ \dots & \dots
\end{array}$$

$$\mathsf{unif}(\Delta_3) = \{S^{\bigcirc}\, z \doteq \underline{r_1}\, z, \underline{r_1} =_d \lambda x.\ get\,(\lambda y.\, \underline{r_1}\, y)\}.$$

Here, the definition r_1 never uses its argument. Our unification algorithm will not actively prune the arguments to recursion constants unless triggered by a unification equation like in rule (PRUNE).

3.6.4 Another Stream Processor that Keeps Consuming Elements. In the previous example, S discards its arguments but passes the read element to itself. One interesting question to ask would be, what if it discards the read element and passes the input argument to itself. Will the two stream processors be equal?

```
?- [x] get ([y] S y) = [x] S x. [x] get ([y] S x) = [x] S x.
```

We expect S not to be able to use any arguments and instead should discard all arguments. Informally, we expect S to be equivalent to H, which discards all arguments and just keeps reading the input stream:

```
S = [x] H. H = get ([y] H).
```

The algorithm is able to find this solution. This example highlights the use of Rules (PRUNE)(N-AGREE), during steps (9), (15), and (16):

$$\Delta_c = \{\lambda x.\ get\,(\lambda y.\, S\, y) \doteq \lambda x.\, S\, x, \lambda x.\ get\,(\lambda y.\, S\, x) \doteq \lambda x.\, S\, x\}$$

$$\begin{array}{ll}
(1)\ \lambda x.\, \underline{r_1}\, x \doteq \lambda x.\, S^{\bigcirc}\, x & \text{given} \\
(2)\ \underline{r_1} =_d \lambda x.\ get\,(\lambda y.\, S^{\bigcirc}\, y) & \\
(3)\ \lambda x.\, \underline{r_2}\, x \doteq \lambda x.\, S^{\bigcirc}\, x & \\
(4)\ \underline{r_2} =_d \lambda x.\ get\,(\lambda y.\, S^{\bigcirc}\, x) & \\
(5)\ \underline{r_1}\, z \doteq S^{\bigcirc}\, z & \text{by Rule (N-INST) on (1)} \\
(6)\ \underline{r_2}\, w \doteq S^{\bigcirc}\, w & \text{by Rule (N-INST) on (3)} \\
(7)\ S^{\bigcirc}\, z \doteq \underline{r_1}\, z & \text{by Rule (N-SYM) on (5)} \\
(8)\ S^{\bigcirc}\, w \doteq \underline{r_2}\, w & \text{by Rule (N-SYM) on (6)} \\
(9)\ \underline{r_1}\, z \doteq \underline{r_2}\, z & \text{by Rule (N-AGREE) on (7)} \\
(10)\ get\,(\lambda y.\, S^{\bigcirc}\, y) \doteq get\,(\lambda y.\, S^{\bigcirc}\, z) & \text{by Rule (REC-EXP) on (9), (2) and (4)} \\
(11)\ \lambda y.\, S^{\bigcirc}\, y \doteq \lambda y.\, S^{\bigcirc}\, z & \text{by Rule (SIMP) on (10)} \\
(12)\ S^{\bigcirc}\, y \doteq S^{\bigcirc}\, z & \text{by Rule (N-INST) on (11)} \\
(13)\ S^{\bigcirc}\, y \doteq H^{\bigcirc} & \text{by Rule (FF-S) on (12)} \\
(14)\ S^{\bigcirc}\, z \doteq H^{\bigcirc} & \\
(15)\ \underline{r_1}\, z \doteq H^{\bigcirc} & \text{by Rule (N-AGREE) on (7) and (14)} \\
(16)\ \underline{r_1}\, z \doteq \underline{t} & \text{by Rule (PRUNE) on (15)} \\
(17)\ H^{\bigcirc} \doteq \underline{t} & \\
(18)\ \underline{t} =_d G^{\blacksquare} & \\
(19)\ get\,(\lambda y.\, S^{\bigcirc}\, y) \doteq G^{\blacksquare} & \text{by Rule (REC-EXP) on (16), (2) and (18)}
\end{array}$$

$(20)\ G^{\blacksquare} \doteq get\,(\lambda y.\, S^{\bigcirc}\, y)$ by Rule (U-SYM) on (19)
$(21)\ S^{\bigcirc}\, y \doteq \underline{t}$ by Rule (N-TRANS) on (13) and (17)
$(22)\ \ldots$ . . .

We have

$$\mathsf{unif}(\Delta_{20}) = \{S^{\bigcirc}\, z \doteq \underline{t}, \underline{t} =_d get\,(\lambda y.\, \underline{t})\}.$$

We explain the construction of $\mathsf{unif}(\Delta_{20})$, and only show the most relevant equations and definitions:

$\Gamma_1 = \{\underline{t} =_d G^{\blacksquare}, \ldots\}$ Start with all the recursive definitions of Δ_{20}.
$\Gamma_1 = \{S^{\bigcirc}\, z \doteq \underline{t}, \underline{t} =_d G^{\blacksquare}, \ldots\}$ Add the resolution equation for $S^{\bigcirc}$.
$\Gamma_1 = \{S^{\bigcirc}\, z \doteq \underline{t}, \underline{t} =_d get\,(\lambda y.\, S^{\bigcirc}\, z), \ldots\}$ Replace resolved unification metavariable $G^{\bigcirc}$ with its resolution.
$\Gamma_1 = \{S^{\bigcirc}\, z \doteq \underline{t}, \underline{t} =_d get\,(\lambda y.\, t), \ldots\}$ Replace resolved unification metavariable $S^{\bigcirc}$ with its resolution.

3.6.5 *Variable Dependency.*

Finally, consider the following unification problem which illustrates how pattern variables following different unification metavariables restrict how the variables can be used:

```
?- get ([x] get ([y] H x)) = get ([x] get ([y] S y)).
```

After reading two input elements, continuation `H` may use the first element, and continuation `S` may use the second element, but `H` and `S` have to be equal. Intuitively, `H` and `S` will be equal to some stream processor `F` that does not use any arguments:

```
H = [x] F, S = [x] F
```

Indeed, unification correctly finds that neither `H` nor `S` can use their argument:

$$\Delta_c = \{get\,(\lambda x.\ get\,(\lambda y.\, H\, x)) \doteq get\,(\lambda x.\ get\,(\lambda y.\, S\, y))\}$$

$(1)\ \underline{r_1} \doteq \underline{r_3}$ given
$(2)\ \underline{r_1} =_d get\,(\lambda x.\, \underline{r_2}\, x)$
$(3)\ \underline{r_2} =_d \lambda x.\ get\,(\lambda y.\, H^{\bigcirc}\, x)$
$(4)\ \underline{r_3} =_d get\,(\lambda x.\, \underline{r_4}\, x)$
$(5)\ \underline{r_4} =_d \lambda x.\ get\,(\lambda y.\, S^{\bigcirc}\, y)$
$(6)\ get\,(\lambda x.\, \underline{r_2}\, x) \doteq get\,(\lambda x.\, \underline{r_4}\, x)$ by Rule (REC-EXP) on (1), (2) and (4)
$(7)\ \lambda x.\, \underline{r_2}\, x \doteq \lambda x.\, \underline{r_4}\, x$ by Rule (SIMP) on (6)
$(8)\ \underline{r_2}\, z \doteq \underline{r_4}\, z$ by Rule (N-INST) on (7)
$(9)\ get\,(\lambda y.\, H^{\bigcirc}\, z) \doteq get\,(\lambda y.\, S^{\bigcirc}\, y)$ by Rule (REC-EXP) on (8), (3) and (5)
$(10)\ \lambda y.\, H^{\bigcirc}\, z \doteq \lambda y.\, S^{\bigcirc}\, y$ by Rule (SIMP) on (9)
$(11)\ H^{\bigcirc}\, z \doteq S^{\bigcirc}\, w$ by Rule (N-INST) on (10)
$(12)\ H^{\bigcirc}\, z \doteq F^{\bigcirc}$ by Rule (FF-D) on (11)
$(13)\ S^{\bigcirc}\, w \doteq F^{\bigcirc}$
$(14)\ \ldots$ . . .

Now the unifier for Δ_{13} is

$$\mathsf{unif}(\Delta_{13}) = \{H^{\bigcirc}\, z \doteq F^{\bigcirc}, S^{\bigcirc}\, w \doteq F^{\bigcirc}, F^{\bigcirc} \doteq F^{\bigcirc}\}.$$

3.7 Correctness of the Algorithm

As with the first-order case, we show the following three properties of the algorithm to establish its correctness:

(1) *Correspondence.* At each step of the algorithm, the most general unifier of the context before corresponds to the most general unifier of the context after (Theorem 3.8). Some steps in the algorithm create additional unification metavariables, and as a result, this correspondence needs to take the difference in the domains of the unifiers into account. The correspondence proof shares the essential ideas of Miller [1991] and Huet [1975]'s proofs in that terms are inspected one level at a time, and the dependencies of unification metavariables on pattern arguments are implicitly kept track of.

(2) *Termination.* Any unification context always saturates in a finite number of steps (Theorem 3.10). This proof is much more involved due to the presence of pattern variables.

(3) *Correctness.* The unifier for a saturated unification context $\mathsf{unif}(\Delta)$ is actually the most general unifier for Δ (Theorem 3.11). The main complexity in this case is to handle the pattern variables following a unification metavariable.

Lemma 3.3. *Given unification contexts Δ and Δ', $\mathsf{eqs}(\Delta) \subseteq \mathsf{eqs}(\Delta')$, $\mathsf{defs}(\Delta) = \mathsf{defs}(\Delta')$, $UV(\Delta) = UV(\Delta')$, then any unifier of Δ' is a unifier of Δ.*

Proof. The proof is essentially the same as the proof for Lemma 2.3, by observing that definitional expansion does not depend on unification equations but only on recursive definitions. □

Lemma 3.4. *If Γ is a unifier for Δ, then Γ is a unifier for Δ' where Δ' has all recursive definitions of Δ and additional true equations $M \doteq M'$ in the sense that $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M[\Gamma]) = \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M'[\Gamma])$.*

Proof. Because definitional expansion depends only on recursive definitions, we have $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M) = \mathsf{exp}^{\Delta'[\Gamma]}_{(k)}(M)$ for all k and M. □

Let $\Gamma'|_{UV(\Delta)}$ be the substitution Γ' with the domain restricted to $UV(\Delta)$. That is, $\Gamma'|_{UV(\Delta)}$ is obtained from Γ' by removing all substitution equations of Γ' if the unification metavariable on the left-hand side is not in $UV(\Delta)$.

We state a similar lemma to Lemma 3.4 but now the unifier of Δ' can have a larger domain than $UV(\Delta)$.

Lemma 3.5. *If Γ is a unifier for Δ, then Γ' (where $\Gamma = \Gamma'|_{dom(\Gamma)}$) is a unifier for Δ' where Δ' has all recursive definitions of Δ and additional true equations $M \doteq M'$ in the sense that $\mathsf{exp}^{\Delta[\Gamma']}_{(k)}(M[\Gamma']) = \mathsf{exp}^{\Delta[\Gamma']}_{(k)}(M'[\Gamma'])$.*

Proof. Similar to the proof of Lemma 3.4. □

We show that the domain restriction preserves unifiers and their ordering.

Lemma 3.6. *Given unification contexts Δ and Δ', $\mathsf{eqs}(\Delta) \subseteq \mathsf{eqs}(\Delta')$, $\mathsf{defs}(\Delta) \subseteq \mathsf{defs}(\Delta')$, $UV(\Delta) \subseteq UV(\Delta')$, for any unifier Γ' of Δ', $\Gamma'|_{UV(\Delta)}$ is a unifier of Δ.*

Proof. This proof is similar to the proof of Lemma 2.3.

Let Γ' be a unifier of Δ', all unification equations of $\Delta'[\Gamma']$ hold. Let $\Gamma = \Gamma'|_{UV(\Delta)}$. Take any $M \doteq M' \in \Delta$, we want to show $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M[\Gamma]) = \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M'[\Gamma])$. We know $\mathsf{exp}^{\Delta'[\Gamma']}_{(k)}(M[\Gamma']) = \mathsf{exp}^{\Delta'[\Gamma']}\ 0ptk(M'[\Gamma'])$, and all recursion constants in M occur in Δ (because unification contexts have to be well-formed). WLOG, it suffices to show $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(M[\Gamma]) = \mathsf{exp}^{\Delta'[\Gamma']}_{(k)}(M[\Gamma'])$. By definition, for any M that occurs in Δ, $M[\Gamma] = M[\Gamma']$, we have $\Delta[\Gamma] = \Delta[\Gamma']$. Then, it suffices to show

$\mathsf{exp}_{(k)}^{\Delta[\Gamma']}(M[\Gamma']) = \mathsf{exp}_{(k)}^{\Delta'[\Gamma']}(M[\Gamma'])$. But since definitional expansions only depend on recursive definitions that actually occur in Δ, we have

$$\begin{aligned}
&\mathsf{exp}_{(k)}^{\Delta[\Gamma']}(M[\Gamma']) \\
&= \mathsf{exp}_{(k)}^{\mathsf{defs}(\Delta[\Gamma'])}(M[\Gamma']) && \text{(expansion only definition only depends on definitions of } \Delta[\Gamma']\text{)} \\
&= \mathsf{exp}_{(k)}^{\mathsf{defs}(\Delta'[\Gamma'])}(M[\Gamma']) && \text{(} M \text{ can only depend on recursion constants occurring in } \Delta\text{)} \\
&= \mathsf{exp}_{(k)}^{\Delta'[\Gamma']}(M[\Gamma']) && \text{(expansion only definition only depends on definitions of } \Delta'[\Gamma']\text{).}
\end{aligned}$$

□

Lemma 3.7 (Domain Restriction Preserves Unifier Ordering). *Given a substitution* $\Gamma_1 \circ \Gamma_2 = \Gamma_3$, *let* $S \subseteq \mathsf{dom}(\Gamma_1)$, *then there exists* Γ_2' *such that* $(\Gamma_1|_S) \circ \Gamma_2' = (\Gamma_3|_S)$. *Moreover,* $\Gamma_2' = \Gamma_2|_{FUV(\Gamma_1|_S)}$, *where the set of free unification metavariables of a substitution,* $FUV(\Gamma)$, *is the set of unification metavariables that occur on the right-hand sides and recursive definitions of the substitution* Γ.

Proof. For any substitution equation $H^m\, \bar{x} \doteq M \in \Gamma_1|_S$, the result of applying Γ_2 to M is the same as the result of applying $\Gamma_2|_{FUV(\Gamma_1|_S)}$ to M. □

Theorem 3.8 (Correspondence). *If* Δ *transforms into* Δ' *by applying one of the rules to some equation in* Δ, *then the set of unifiers of* Δ *coincides with the set of unifiers in* Δ' *with domains restricted to* $UV(\Delta)$. *Moreover, domain restriction preserves most general unifiers.*

Proof. The proof has two parts. First, we show that domain restriction preserves unifiers. Then, we show that the domain restriction preserves most general unifiers.

(Part 1) If Δ' contains contra then there is no unifier for Δ', and we can show in each case that there is no unifier for Δ by inspecting rules (SIMP-F1)(SIMP-F2)(SIMP-F3)(PROJ-F), and the case where contra is already present in Δ. Otherwise, assume $\mathsf{contra} \notin \Delta'$.

There are two groups of rules, rules that do not add new unification metavariables (*Group 1*), and rules that add new unification metavariables (*Group 2*).

(*Group 1*) For the rules that do not add new unification metavariables, every unifier for Δ' is also a unifier of Δ by Lemma 3.3. And every unifier of Δ is also a unifier for Δ' by Lemma 3.4. We show rules (REC-EXP)(U-AGREE) as examples for applying Lemma 3.4.

Rule (REC-EXP), given $\underline{r}\, \bar{x} \doteq \underline{s}\, \bar{y}$, $r =_d \lambda z.\, U_1$, and $s =_d \lambda w.\, U_2$ in Δ, it adds the equation $[\bar{x}/\bar{z}]U_1 \doteq [\bar{y}/\bar{w}]U_2$ in Δ'. By Lemma 3.4, it suffices to show $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(([\bar{x}/\bar{z}]U_1)[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(([\bar{y}/\bar{w}]U_2)[\Gamma])$. Since Γ is a unifier for Δ, we have $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}((\underline{r}\, \bar{x})[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}((\underline{s}\, \bar{y})[\Gamma])$, but $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}((\underline{r}\, \bar{x})[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(\underline{r}\, \bar{x}) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}([\bar{x}/\bar{z}](U_1[\Gamma]))$ and similarly for $(\underline{s}\, \bar{y})$.

Rule (U-AGREE), given $H^{\blacksquare}\, \bar{x} \doteq U_1$ and $H^{\blacksquare}\, \bar{y} \doteq U_2$ in Δ, it adds the equation $U_1 \doteq [\bar{x}/\bar{y}]U_2$ to Δ'. Let Γ be a unifier for Δ, we want to show Γ is a unifier for Δ'. By Lemma 3.4, it suffices to show $\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_1[\Gamma]) = \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(([\bar{x}/\bar{y}]U_2)[\Gamma])$. Suppose $H^{\blacksquare}\, \bar{w} \doteq U_{H^{\blacksquare}}$ is the substitution equation in Γ, and then

$$\begin{aligned}
&\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_1[\Gamma]) \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}((H^{\blacksquare}\, \bar{x})[\Gamma]) && \text{(since } \Gamma \text{ is a unifier for } \Delta \text{ and } H^{\blacksquare}\, \bar{x} \doteq U_1 \in \Delta\text{)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}([\bar{x}/\bar{w}]U_{H^{\blacksquare}}) && \text{(by the substitution equation } H^{\blacksquare}\, \bar{w} \doteq U_{H^{\blacksquare}} \in \Gamma\text{)} \\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}([\bar{x}/\bar{y}]\,[\bar{y}/\bar{w}]U_{H^{\blacksquare}}) && \text{(by the definition of variable renaming)} \\
&= [\bar{x}/\bar{y}]\ \mathsf{exp}_{(k)}^{\Delta[\Gamma]}([\bar{y}/\bar{w}]U_{H^{\blacksquare}}) && \text{(variable renaming commutes with definitional expansion)}
\end{aligned}$$

$$
\begin{aligned}
&= [\bar{x}/\bar{y}]\,\mathsf{exp}_{(k)}^{\Delta[\Gamma]}((H^{\blacksquare}\,\bar{y})[\Gamma]) && \text{(by the substitution equation } H^{\blacksquare}\,\bar{w} \doteq U_{H^{\blacksquare}} \in \Gamma\text{)}\\
&= [\bar{x}/\bar{y}]\,\mathsf{exp}_{(k)}^{\Delta[\Gamma]}(U_2[\Gamma]) && \text{(since } \Gamma \text{ is a unifier for } \Delta \text{ and } H^{\blacksquare}\,\bar{y} \doteq U_2 \in \Delta\text{)}\\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}([\bar{x}/\bar{y}](U_2[\Gamma])) && \text{(variable renaming commutes with definitional expansion)}\\
&= \mathsf{exp}_{(k)}^{\Delta[\Gamma]}(([\bar{x}/\bar{y}]U_2)[\Gamma]) && \text{(variable renaming commutes with substitution).}
\end{aligned}
$$

The rest of the cases except rules (IMIT)(PROJ)(PRUNE)(FF-D)(FF-S) are similar. It is obvious that the identity mapping (as a trivial domain restriction) preserves the most general unifiers.

(*Group 2*) For rules that add new unification metavariables (i.e., rules (IMIT)(PROJ)(PRUNE)(FF-D)(FF-S)), we show that if Γ' is a unifier for Δ', and then $\Gamma = \Gamma'|_{UV(\Delta)}$ is the unifier for Δ.

Rules (IMIT)(PROJ), we have $H^{\blacksquare}\,\bar{y} \doteq h\,N_1\,\ldots N_n \in \Delta$, and

$$H^{\blacksquare}\,\bar{y} \doteq h\,(\lambda\bar{x}_1.\,G_1^{\bigcirc}\,\bar{y}\,\bar{x}_1)\ \ldots\ (\lambda\bar{x}_n.\,G_n^{\bigcirc}\,\bar{y}\,\bar{x}_n) \in \Delta'.$$

Given Γ' is a unifier for Δ', it follows from Lemma 3.6 that Γ is a unifier for Δ. To show that the restriction $-|_{UV(\Delta)}$ preserves unifiers, it suffices to show that given any Γ that is a unifier of Δ, there exists a unique Γ' such that $\Gamma'|_{UV(\Delta)}$ is Γ. Given a unifier Γ, the substitution equation for $H^{\blacksquare}$ is $H^{\blacksquare}\,\bar{z} \doteq h\,(\lambda\bar{x}_1.\,N_1')\ \ldots (\lambda\bar{x}_n.\,N_n')$. The corresponding Γ' will map $H^{\blacksquare}$ analogously, and will have the substitution equation for $G_i^{\bigcirc}$ as $G_i^{\bigcirc}\,\bar{x}_i\,\bar{z} \doteq N_i'$. Γ' is a unifier for Δ' by Lemma 3.5. This Γ' is unique because any different mapping of $G_i^{\bigcirc}$ (modulo definitional expansion) will make the additional equation in Δ' false.

Rule (PRUNE), we have $H^{\bigcirc}\,\bar{y} \doteq \underline{r}\,\bar{x} \in \Delta$, and

$$H^{\bigcirc}\,\bar{y} \doteq \underline{t}\,\bar{w}, \underline{r}\,\bar{x} \doteq \underline{t}\,\bar{w}, \underline{t} =_d \lambda\bar{w}.\,G^{\blacksquare}\,\bar{w} \in \Delta'$$

with $\bar{w} = \bar{x} \cap \bar{y}$. Given Γ' is a unifier for Δ', it follows from Lemma 3.6 that Γ is a unifier for Δ. To show that the restriction $-|_{UV(\Delta)}$ preserves unifiers, it suffices to show that given any Γ that is a unifier of Δ, there exists a unique Γ' such that $\Gamma'|_{UV(\Delta)}$ is Γ. Given a unifier Γ, the substitution equation for $H^{\bigcirc}$ is $H^{\bigcirc}\,\bar{y} \doteq \underline{s}\,\bar{w}$ with $\underline{s} =_d \lambda\bar{w}.\,U \in \Gamma$.[6] $H^{\bigcirc}$ cannot be mapped to another recursive unification metavariable because of the equation $H^{\bigcirc}\,\bar{y} \doteq r\,\bar{x}$. The corresponding Γ' will map $H^{\bigcirc}$ analogously, and will have the substitution equation for $G^{\blacksquare}$ as $G^{\blacksquare}\,\bar{w} \doteq U$. This Γ' is unique because any different mapping of $G^{\blacksquare}$ (modulo definitional expansion) will make the additional equation in Δ' false.

Rules (FF-D)(FF-S) follows a similar argument. We elide the full development and remark that Γ' will map the additional unification metavariable F^m analogously as Γ maps G^m.

(Part 2) $-|_{UV(\Delta)}$ preserves the most general unifiers since the substitution that mediates between the most general unifier Γ' and any more specific Γ_2 is a substitution whose restriction mediates between $\Gamma'|_{UV(\Delta)}$ and $\Gamma_2|_{UV(\Delta)}$ by Lemma 3.7. □

Lemma 3.9 (Preservation of Well-Formed Unification Contexts). *The pattern restriction, β-normal-η-long forms, and typing are respected by the unification rules.*

Proof. Directly by analyzing the rules. □

The preservation of β-normal-η-long forms guarantees that the number of variables following any unification metavariable is constant throughout. We define the *width* of a unification metavariable to be the number of variables following it. For example, if $F^{\blacksquare}\,x\,y\,z$ appears in a unification context Δ, then the width of $F^{\blacksquare}$ is 3. Similarly, we define the *width* of a recursion constant to be the number

[6] In practice, the substitution equation may be $H^{\bigcirc}\,\bar{z} \doteq \underline{s}\,\bar{u}$, we can α-rename it to $H^{\bigcirc}\,\bar{y} \doteq \underline{s}\,([\bar{y}/\bar{z}]\bar{u})$. It might be the case that $([\bar{y}/\bar{z}]\bar{u}) \subsetneq \bar{w}$, with $\underline{s} =_d \lambda([\bar{y}/\bar{z}]\bar{u}).\,U$, but we can always construct another definition $\underline{q} =_d \lambda\bar{w}.\,U$ and set $H^{\bigcirc}\,\bar{y} \doteq \underline{q}\,\bar{w}$.

of variables following it. A recursion constant r is *pruned* if there exists an equation $r\,\bar{x} \doteq s\,\bar{y}$ such that $\bar{y} \subsetneq \bar{x}$, and r is *unpruned* otherwise. Note that the formal concept of *pruned* and *unpruned* recursion constants here should be understood in a different sense from the rule (PRUNE), which has been suggesting the informal meaning of removing variables from the pattern arguments.

Theorem 3.10 (Termination). *The algorithm always terminates.*

Proof. We observe that terms in the unification contexts are shallow as defined by the grammar, and all terms are well-typed. Given a bounded amount of variables, unification metavariables, and recursion constants, there can only be finitely many equations and recursive definitions in a unification context because terms are shallow that they are only one level deep. The rules that create new unification metavariables, variables, or recursion constants are rules (U-INST) (N-INST)(IMIT)(PROJ)(PRUNE)(FF-D)(FF-S), it suffices to show that these rules can only be applied finitely many times.

First we show that given a bounded amount of unification metavariables and recursion constants, the rules (U-INST)(N-INST) can only be applied finitely many times. Since everything is well-typed, the maximum depth and width for terms are bounded. Then, there are only finitely many equations modulo simultaneous variable renaming, and the subsumption $\exists\bar{x}$ in the conclusion of the rule (U-INST)(N-INST) prevents additional equations from being created that are merely variable renaming of existing equations.

Then, it suffices to show the rules (IMIT)(PROJ)(PRUNE)(FF-D)(FF-S) can only be applied finitely many times. We associate with each unification context a lexicographic multiset order $\langle A, B, C\rangle$, where A, B, C are multisets of natural numbers defined below, and show that each rule that creates new unification metavariables or recursion constants strictly decreases this order. The multiset order [Dershowitz and Manna, 1979] states that a multiset of natural numbers X is considered smaller than another multiset Y if X can be obtained from Y by removing a natural number n and adding a finite number of natural numbers that are strictly smaller than n. The order $\langle A, B, C\rangle$ is given by

(1) $A = \{\mathsf{width}(\underline{r}) \mid \underline{r} =_d U \in \Delta, \underline{r}\ \mathsf{unpruned}\}$ is the multiset of widths of all unpruned recursion constants.
(2) $B = \{\mathsf{width}(H^{\blacksquare}) \mid H^{\blacksquare} \in UV(\Delta), H^{\blacksquare}\ \mathsf{unresolved}\}$ is the multiset of widths of all unresolved contractive unification metavariables.
(3) $C = \{\mathsf{width}(H^{\bigcirc}) \mid H^{\bigcirc} \in UV(\Delta), H^{\bigcirc}\ \mathsf{unresolved}\}$ is the multiset of widths of all unresolved recursive unification metavariables.

For example, we could decrease the order $\langle A, B, C\rangle$ by resolving a contractive unification metavariable, and adding arbitrarily many recursive unification metavariables of any width.

First, it is easy to see that no rules can ever increase this order except rules (IMIT)(PROJ)(PRUNE) (FF-D)(FF-S): Once a unification metavariable is resolved, it remains resolved, and once a recursion constant is pruned, it remains pruned. Both conditions rely on the existence of certain equations and rules never remove equations from the unification context. Then we show each of the rules (IMIT)(PROJ)(PRUNE)(FF-D)(FF-S) strictly decreases the order $\langle A, B, C\rangle$.

Rule (IMIT) or (PROJ) removes one unresolved contractive unification metavariable and adds a finite number of recursive unification metavariables.

Rule (PRUNE) prunes a recursion constant and adds a contractive unification metavariable.

Each of the rules (FF-D) and (FF-S) resolves a recursive (respectively, contractive) unification metavariable and adds a recursive (respectively, contractive) unification metavariable of a smaller width. □

Theorem 3.11 (Correctness of Unifiers). *If Δ is a saturated contradiction-free unification context, and $\Gamma = \mathsf{unif}(\Delta)$ is the most general unifier for Δ.*

Proof. The proof largely follows the structure of the first-order case. We have two parts, the first part is to show that Γ is a unifier, and the second part is to show that Γ is the most general unifier.

(Part 1) To show Γ is a unifier, we need to show that $\mathsf{dom}(\Gamma) = UV(\Delta)$, which is true by definition, and that every equation in $\Delta[\Gamma]$ holds. It suffices to show the following:

(1) For all $U_1 \doteq U_2$ in Δ, $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(U_1[\Gamma]) = \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(U_2[\Gamma])$.

(2) For all $N_1 \doteq N_2$ in Δ, $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(N_1[\Gamma]) = \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(N_2[\Gamma])$.

We show the following two claims simultaneously by lexicographic induction on (k, and the structure of U or N), where claim (2) may refer to claim (1) without decreasing k. Both claims are trivial when $k = 0$. Consider the case when $k > 0$, we show (1) and (2) by case analysis on the structure of $U_1 \doteq U_2$ and $N_1 \doteq N_2$.

We state some facts about the construction of $\mathsf{unif}(\Delta)$. These facts build intuitions that make the proof cases easier to understand. We note that if the substitution equation for $H^{\blacksquare}$ is $H^{\blacksquare}\,\bar{w} \doteq F^{\blacksquare}\,\bar{z}$ (F is necessarily unresolved by step (4) of the construction for $\Gamma = \mathsf{unif}(\Delta)$), then the substitution equation will also appear in Δ due to transitivity and the consistency of resolution rule. Also, if the substitution equation for $H^{\blacksquare}$ is $H^{\blacksquare}\,\bar{w} \doteq h\,N_1\,\ldots\,N_n$, there exists equations (not necessarily picked as resolution equations) $H^{\blacksquare}\,\bar{w} \doteq h\,N_1'\,\ldots\,N_n'$ (with $FV(h\,N_1'\,\ldots\,N_n') \subseteq \bar{w}$), and $N_1 \doteq N_1', \ldots, N_n \doteq N_n'$ in Δ, by inspecting the process of obtaining Γ and the structural rule (SIMP). For recursive unification metavariables $H^{\bigcirc}$, the substitution equations $H^{\bigcirc}\,\bar{w} \doteq F^{\bigcirc}\,\bar{z}$ and $H^{\bigcirc}\,\bar{w} \doteq \underline{r}\,\bar{z}$ will appear in Δ due to transitivity and the consistency of resolution (rule (N-AGREE)):

(a) If U_1 or U_2 contains top-level λ-abstractions, then they must have equal number of λ-abstractions due to typing and η-long forms. Due to the commutation of the definitional expansion and λ-abstractions and the commutation between substitution and λ-abstractions, the result follows by induction hypothesis. The cases (b)–(d) below show the claim when U_1 and U_2 do not have top-level λ-abstractions.

(b) Both U_1 and U_2 have constructors or variables as their heads. Since $\mathsf{contra} \notin \Delta$, they must have identical constructor or variable heads. Now let $U_1 = h\,N_1\,\ldots\,N_n$ and $U_2 = h\,N_1'\,\ldots\,N_n'$. Since Δ is saturated, we have $N_i \doteq N_i'$ for all $1 \leq i \leq n$. The result then follows from the fact that each N_i and N_i' have equal definitional expansion up to depth $k - 1$ by induction hypothesis.

(c) One of U_1 and U_2 is a unification metavariable, and the other has a constructor as its head. Without loss of generality, assume $U_1 = H^{\blacksquare}\,\bar{x}$ and $U_2 = h\,\overline{N}$. We have two cases, either $FV(h\,\overline{N}) \subseteq \bar{x}$ or not:

  (i) If $FV(h\,\overline{N}) \subseteq \bar{x}$, then this is a resolution equation. If this is the resolution equation used in Γ, then we're done. Otherwise, the resolution equation of $H^{\blacksquare}$ in Γ may be $H^{\blacksquare}\,\bar{x} \doteq h\,\overline{N'}$. The rule (U-AGREE) ensures that there is an equation between U_2 and $h\,\overline{N'}$, and the rest follows from the case (b) above.

  (ii) If $FV(h\,\overline{N}) \nsubseteq \bar{x}$, in this case either $H^{\blacksquare}$ is resolved or unresolved. $H^{\blacksquare}$ cannot be unresolved, since otherwise rule (IMIT) or (PROJ) would apply. If $H^{\blacksquare}$ is resolved, then $H^{\blacksquare}\,\bar{x} \doteq h\,\overline{N'}$ is a substitution equation, and the result follows from IH on the necessary equations between $\overline{N'}$ and $\overline{N}$.

(d) Both U_1 and U_2 have contractive unification metavariables as their heads. Let $U_1 = H^{\blacksquare}\,\bar{x}$ and $U_2 = G^{\blacksquare}\,\bar{y}$. Due to saturation, WLOG, there are three cases, both unification metavariables are unresolved, only one is unresolved, or both are resolved. We consider them one by one:

(i) Both are unresolved. If $H^\blacksquare$ is equal to $G^\blacksquare$ (they are the same unification metavariable), then $\bar{x} = \bar{y}$ (position-wise) since otherwise rule (FF-S) will apply, and it would have a resolution. Otherwise, suppose $H^\blacksquare \neq G^\blacksquare$. Since they are in the same equivalence class, for some representative unification metavariable be $F^\blacksquare$, we have the representative equations $H^\blacksquare\,\bar{w} \doteq F^\blacksquare\,\bar{z}$ and $G^\blacksquare\,\bar{u} \doteq F^\blacksquare\,\bar{z}$ in Γ. Here $\bar{w}$ and $\bar{x}$ may differ. By rule (U-AGREE), on $H^\blacksquare$, we have equations $G\,\bar{y} \doteq [\bar{x}/\bar{w}](F^\blacksquare\,\bar{z})$ and similarly $H^\blacksquare\,\bar{x} \doteq [\bar{x}/\bar{w}](F^\blacksquare\,\bar{z})$ in Δ. By symmetry and transitivity, we have $[\bar{x}/\bar{w}](F^\blacksquare\,\bar{z}) \doteq [\bar{y}/\bar{u}](F^\blacksquare\,\bar{z})$ in Δ. We have just shown that $[\bar{x}/\bar{w}](F^\blacksquare\,\bar{z})$ and $[\bar{y}/\bar{u}](F^\blacksquare\,\bar{z})$ are syntactically equal (otherwise they will be resolved by rule (FF-S)). But now $U_1[\Gamma] = (H^\blacksquare\,\bar{x})[\Gamma] = [\bar{x}/\bar{w}](F^\blacksquare\,\bar{z}) = [\bar{y}/\bar{u}](F^\blacksquare\,\bar{z}) = (G^\blacksquare\,\bar{y})[\Gamma] = U_2[\Gamma]$.

(ii) Only one of them is unresolved. WLOG, $H^\blacksquare$ is unresolved and $G^\blacksquare$ is resolved. We have $G^\blacksquare\,\bar{u} \doteq F^\blacksquare\,\bar{z}$ or $G^\blacksquare\,\bar{u} \doteq h\,\overline{N}$ in Γ. In the first case $F^\blacksquare\,\bar{z}$ is unresolved (otherwise it would have been replaced in step (4)), and then $H^\blacksquare$ and $F^\blacksquare$ are in the same equivalence class, and the rest follows from the case (d)(i) above. In the second case, we would have an equation $H^\blacksquare\,\bar{x} \doteq [\bar{y}/\bar{u}](h\,\overline{N})$. But now $H^\blacksquare$ could be resolved by rule (IMIT) or (PROJ).

(iii) Both are resolved. Suppose $H^\blacksquare\,\bar{x} \doteq U_{H^\blacksquare}$ and $G^\blacksquare\,\bar{y} \doteq U_{G^\blacksquare}$ are substitution equations in Γ. It cannot be the case that only one of $U_{H^\blacksquare}$ and $U_{G^\blacksquare}$ has unresolved unification metavariables as the head, and the other has a constructor or a variable as the head. Since transitivity and rule (U-AGREE) ensure an equation between $U_{H^\blacksquare}$ and $U_{G^\blacksquare}$, and the other would be resolved by rule (IMIT) or (PROJ). Thus, both $U_{H^\blacksquare}$ and $U_{G^\blacksquare}$ have unresolved unification metavariables as the head, or both have constructors or variables as the head. In the first case, the result follows from the case (d)(i) above. In the second case, let $U_{H^\blacksquare} = h\,\overline{N}$ and $U_{G^\blacksquare} = h\,\overline{N'}$, there are equations $U_1 \doteq h\,\overline{N''}$ and $U_2 \doteq h\,\overline{N'''}$ in Δ, with equations between $\overline{N}$ and $\overline{N''}$ (pairwise), and similarly for $\overline{N'}$ and $\overline{N'''}$. By transitivity, there is an equation $h\,\overline{N''} \doteq h\,\overline{N'''}$, and thus there are equations between $\overline{N''}$ and $\overline{N'''}$ (pairwise). Then the result follows by IH to show $\overline{N}, \overline{N'}, \overline{N''}, \overline{N'''}$ all have equal definitional expansions up to depth $k-1$.

(e) The case where N_1 or N_2 contains top-level λ-abstractions is similar to the case (a), and we show subsequently the cases when N_1 and N_2 do not contain top-level λ-abstractions.

(f) Both N_1 and N_2 have recursion constants as heads. Let $N_1 = \underline{r}\,\bar{x}$, where $\underline{r} =_d \bar{w}.\,U_1 \in \Delta$ and $N_2 = \underline{s}\,\bar{y}$, where $s =_d \lambda\bar{u}.\,U_2 \in \Delta$. Since Δ is saturated, $[\bar{x}/\bar{w}]U_1 \doteq [\bar{y}/\bar{u}]U_2 \in \Delta$. By IH, $\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(([\bar{x}/\bar{w}]U_1)[\Gamma]) = \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(([\bar{y}/\bar{u}]U_2)[\Gamma])$, and then

$$
\begin{aligned}
&\mathsf{exp}^{\Delta[\Gamma]}_{(k)}(N_1[\Gamma]) \\
&= \mathsf{exp}^{\Delta[\Gamma]}_{(k)}((\underline{r}\,\bar{x})[\Gamma]) && \text{(given)} \\
&= \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(([\bar{x}/\bar{w}]U_1)[\Gamma]) && \text{(by the definition of } \mathsf{exp}^{\Delta[\Gamma]}_{(k)}\text{)} \\
&= \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(([\bar{y}/\bar{u}]U_2)[\Gamma]) && \text{(shown by IH above)} \\
&= \mathsf{exp}^{\Delta[\Gamma]}_{(k)}((\underline{s}\,\bar{y})[\Gamma]) && \text{(by the definition of } \mathsf{exp}^{\Delta[\Gamma]}_{(k)}\text{)} \\
&= \mathsf{exp}^{\Delta[\Gamma]}_{(k)}(N_2[\Gamma]) && \text{(given).}
\end{aligned}
$$

(g) One of N_1 and N_2 is a unification metavariable, and the other has a recursion constant as its head. WLOG, assume $N_1 = H^\bigcirc\,\bar{x}$ and $N_2 = \underline{r}\,\bar{y}$. We have two cases, either $FV(r\,\bar{y}) \subseteq \bar{x}$ or not:

(i) If $FV(\underline{r}\,\bar{y}) \subseteq \bar{x}$, then this is a resolution equation. If this is the substitution equation used in Γ, then we're done. Otherwise, rule (N-AGREE) ensures that there is an equation between N_2 and the resolution equation used in Γ, and the rest follows from the case (f) above.

(ii) If $FV(\underline{r}\,\bar{y}) \not\subseteq \bar{x}$, in this case either $H^\bigcirc$ is resolved or unresolved. $H^\bigcirc$ cannot be unresolved, since otherwise rule (PRUNE) would apply. If $H^\bigcirc$ is resolved, then let $H^\bigcirc\,\bar{x} \doteq \underline{s}\,\bar{z}$ be a substitution equation. By transitivity, there is an equation between $\underline{s}\,\bar{z}$ and $\underline{r}\,\bar{y}$. By the case (f), $\mathsf{exp}_{(k)}(\underline{s}\,\bar{z}) = \mathsf{exp}_{(k)}(\underline{r}\,\bar{y})$.

(h) Both N_1 and N_2 have recursive unification metavariables as their heads. Let $N_1 = H^{\bigcirc}\,\bar{x}$ and $N_2 = G^{\bigcirc}\,\bar{y}$. Due to saturation, WLOG, there are three cases, both unification metavariables are unresolved, only one is unresolved, or both are resolved. We consider them one by one:

(i) Both are unresolved. This is exactly analogous to the case (d)(i).

(ii) Only one of them is unresolved. This is exactly analogous to the case (d)(ii), except that in the case the resolution is a recursion constant, the unresolved unification metavariables may be resolved by rule (PRUNE).

(iii) Both are resolved. Suppose $H^{\bigcirc}\,\bar{x} \doteq N_{H^{\circ}}$ and $G^{\bigcirc}\,\bar{y} \doteq N_{G^{\circ}}$ are substitution equations in Γ. By a similar reasoning as (d)(iii), both $N_{H^{\circ}}$ and $N_{G^{\circ}}$ have unresolved unification metavariables as heads, or both have recursion constants as the heads. In the first case, the equality can be established by (h)(i). In the latter case, there is an equation $N_{H^{\circ}} \doteq N_{G^{\circ}}$ due to transitivity and the rest follows by the case (f).

(Part 2) To show Γ is the most general unifier, given any other unifier Γ_2 of Δ, it suffices to construct a unifier Γ_1 such that $\Gamma \circ \Gamma_1 = \Gamma_2$. But the construction of Γ_1 is easy: Γ_2 must map resolved unification metavariables analogously as Γ (otherwise a contradiction will arise), and it may choose to map equivalence classes of unresolved unification metavariables freely. Γ_1 simply records how unresolved unification metavariables are mapped in Γ_2. □

4 Related Work

The unification algorithm for the first-order terms was first developed by Robinson [1965] as a procedure for implementing resolution. Jaffar [1984] gave an efficient unification algorithm for first-order rational trees based on the system of equations presentation of Martelli and Montanari [1982]. Huet [1975] discovered a preunification algorithm for general higher-order terms. Although general higher-order unification is undecidable and does not have most general unifiers [Huet, 1973], Miller [1991] discovered the pattern restriction that if arguments to unification metavariables are restricted to pairwise distinct bound variables, decidability and most general unifiers can be recovered. A similar idea of restricting the arguments to bound variables gives a formulation of regular Böhm trees [Huet, 1998] with decidable equality. Huet's idea later becomes the prepattern restriction in CoLF [Chen and Pfenning, 2023]. The prepattern restriction is slightly relaxed over the pattern restriction that repetition of bound variables are allowed. Our use of a signature for representing recursive definitions directly follows that of CoLF [Chen and Pfenning, 2023].

Nominal unification is an alternative way of carrying out higher-order unification [Urban et al., 2004; Urban, 2010]. It is encodable in higher-order pattern unification and higher-order pattern unification can be encoded in nominal unification. Schmidt-Schauß et al. [2022] have presented a nominal unification algorithm for a version of cyclic λ-calculi by Ariola and Blom [1997]. However, their cyclic λ-calculi has a different criterion for term equality than ours.

5 Conclusion

We have presented a saturation-based unification algorithm for finding most general unifiers for higher-order rational terms ($\perp$-free regular Böhm trees). We have shown the termination, soundness, and completeness of this algorithm. The main complexity is to arrange the conditions for applying the rules to ensure termination. We once again find Miller's pattern fragment to be fundamental in determining the most general unifiers in the presence of higher-order terms. A detailed analysis of the complexity of the algorithm and an efficient implementation will be future work.

References

Andreas Abel and Brigitte Pientka. 2016. Well-founded recursion with copatterns and sized types. *Journal of Functional Programming* 26 (2016), e2. DOI: https://doi.org/10.1017/S0956796816000022

Zena M. Ariola and Stefan Blom. Cyclic lambda calculi. 1997. In *3rd International Symposium on Theoretical Aspects of Computer Software (TACS '97).* Martín Abadi and Takayasu Ito (Eds.), Vol. 1281 of Lecture Notes in Computer Science, Springer, 77–106. DOI: https://doi.org/10.1007/BFb0014548

James Brotherston and Alex Simpson. 2011. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation* 21, 6 (2011), 1177–1216.

Zhibo Chen and Frank Pfenning. 2023. A logical framework with higher-order rational (circular) terms. In *Foundations of Software Science and Computation Structures - 26th International Conference (FoSSaCS '23), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS '23).* Orna Kupferman and Pawel Sobocinski (Eds.), Vol. 13992 of Lecture Notes in Computer Science, Springer, 68–88. DOI: https://doi.org/10.1007/978--3-031--30829-1_4

Nils Anders Danielsson and Thorsten Altenkirch. 2010. Subtyping, declaratively. In *10th International Conference on Mathematics of Program Construction (MPC '10).* Springer, 100–118.

Farzaneh Derakhshan and Frank Pfenning. 2019. Circular proofs as session-typed processes: A local validity condition. arXiv:1908.01909. Retrieved from http://arxiv.org/abs/1908.01909

Nachum Dershowitz and Zohar Manna. 1979. Proving termination with multiset orderings. *Communications of the ACM* 22, 8 (1979), 465–476. DOI: https://doi.org/10.1145/359138.359142

Jérôme Fortier and Luigi Santocanale. 2013. Cuts for circular proofs: Semantics and cut-elimination. In *22nd Annual Conference on Computer Science Logic (CSL '13).* Simona Ronchi Della Rocca (Ed.), LIPIcs 23, 248–262.

Harald Ganzinger. 1996. Saturation-based theorem proving (abstract). In *Automata, Languages and Programming, 23rd International Colloquium (ICALP '96).* Friedhelm Meyer auf der Heide and Burkhard Monien (Eds.), Vol. 1099 of Lecture Notes in Computer Science, Springer, 1–3. DOI: https://doi.org/10.1007/3-540-61440-0_113

Neil Ghani, Peter G. Hancock, and Dirk Pattinson. 2009. Representations of stream processors using nested fixed points. *Logical Methods in Computer Science* 5, 3 (2009). DOI: https://doi.org/10.2168/LMCS-5(3:9)2009

Gérard P. Huet. 1973. The undecidability of unification in third order logic. *Information Control* 22, 3 (1973), 257–267. DOI: https://doi.org/10.1016/S0019-9958(73)90301-X

Gérard P. Huet. 1975. A unification algorithm for typed lambda-calculus. *Theoretical Computer Science* 1, 1 (1975), 27–57. DOI: https://doi.org/10.1016/0304--3975(75)90011--0

Gérard P. Huet. 1998. Regular Böhm trees. *Mathematical Structures in Computer Science* 8, 6 (1998), 671–680. Retrieved from http://journals.cambridge.org/action/displayAbstract?aid=44783

Joxan Jaffar. 1984. Efficient unification over infinite terms. *New Generation Computing* 2, 3 (1984), 207–219.

Alberto Martelli and Ugo Montanari. 1982. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems* 4, 2 (1982), 258–282. DOI: https://doi.org/10.1145/357162.357169

Sean McLaughlin and Frank Pfenning. 2009. Efficient intuitionistic theorem proving with the polarized inverse method. In , *22nd International Conference on Automated Deduction (CADE '22).* Renate A. Schmidt (Ed.), Vol. 5663 of Lecture Notes in Computer Science, Springer, 230–244. DOI: https://doi.org/10.1007/978-3-642-02959-2_19

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. DOI: https://doi.org/10.1093/logcom/1.4.497

Frank Pfenning. 2006. Lecture Notes on Unification. Retrieved from http://www.cs.cmu.edu/fp/courses/lp/lectures/06-unif.pdf

Frank Pfenning and Carsten Schürmann. 1999. System description: Twelf - A meta-logical framework for deductive systems. In *16th International Conference on Automated Deduction (CADE '16).* Harald Ganzinger (Ed.), Vol. 1632 of Lecture Notes in Computer Science, Springer, 202–206.

John Alan Robinson. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12, 1 (1965), 23–41. DOI: https://doi.org/10.1145/321250.321253

Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, Mateu Villaret, and Yunus D. K. Kutz. 2022. Nominal unification and matching of higher order expressions with recursive let. *Fundamenta Informaticae* 185, 3 (2022), 247–283. DOI: https://doi.org/10.3233/FI-222110

Carsten Schürmann and Frank Pfenning. 2003. A coverage checking algorithm for LF. In *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs ('03).* David A. Basin and Burkhart Wolff (Eds.), Vol. 2758 of Lecture Notes in Computer Science, Springer, 120–135. DOI: https://doi.org/10.1007/10930755_8

Christian Urban. 2010. Nominal unification revisited. In *24th International Workshop on Unification (UNIF '10)*, 1–11. DOI: https://doi.org/10.4204/EPTCS.42.1

Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. 2004. Nominal unification. *Theoretical Computer Science* 323, 1–3 (2004), 473–497. DOI: https://doi.org/10.1016/j.tcs.2004.06.016

Received 12 December 2023; revised 31 May 2024; accepted 1 October 2024