A Formulation of Second-Order Matching

Djamel Ouassim Ait-Moussa

August 17, 2025

Higher-order unification is the problem of solving semantic equations between arbitrary programs, understood as λ-terms. While higher-order unification is undecidable in general (Huet, 1976), decidable special cases have been extensively investigated (Huet and Lang, 1978; Miller, 1992; Dowek, 2001). Among them, *second-order matching* has been proposed as a generalization of the first-order pattern-matching construct found in functional languages to second-order terms, that is, to syntax trees with variable binding.

In this report, we revisit the Huet-Lang algorithm (Huet and Lang, 1978) for solving second-order matching problems. We reformulate their algorithm in modern notation, using a non-deterministic system of inference rules to express the available canonical solutions to a given matching problem. This allows us to specify the algorithm in a concise way, emphasizing its core steps and simplifying its correctness proof.

1 Introduction

Unification problems predate the study of programming languages, and were introduced in the context of propositional logic (Herbrand, 1930). A unification problem is an equation of the form $t_1 = t_2$ where t_1 and t_2 are syntactic objects called *terms* that contain *variables* representing unknowns. Solving a unification problem means finding values for some or all of the variables in order to make the left-hand side and the right-hand side of the equation syntactically equal. The action of replacing each variable with its chosen value is called *substitution*.

1.1 First-order syntax

Terms can represent objects of different natures. An example of a term in propositional logic would be

$$t_1 = (A \implies (B \wedge A \implies D)) \vee C$$

where A, B, C, D are constant propositions. Terms can also represent algebraic expressions, for example

$$u_1 = ((3 + 5) \times 7) - 12,$$

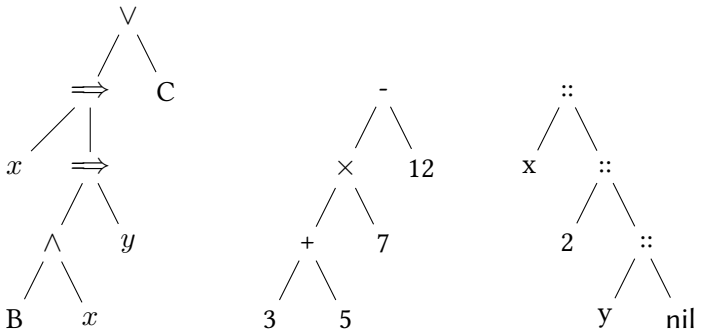

Figure 1: Terms t_2, u_2 and v_2 seen as trees

or inductive data structures that can be found in functional programming languages, such as

$$v_1 = 1 :: (2 :: (3 :: (4 :: \mathsf{nil}))),$$

which represents the list usually denoted by $[1; 2; 3; 4]$.

The three example terms above are *closed*, meaning that they contain no variable. In almost all situations where we reason about syntactic objects, the need to refer to "unknown" or "undetermined" subterms arises, and it is answered by adding variables to the terms. We give three example terms with variables.

$$\begin{aligned} t_2 &= (x \implies (B \wedge x \implies y)) \vee C \\ u_2 &= (x \times 7) - 12 \\ v_2 &= x :: (2 :: y) \end{aligned}$$

We give meaning to the concept of a variable through the operation of substitution. If we replace each occurence of x by the term A and each occurence of y by the term D in t_2, we get $(A \implies (B \wedge A \implies D)) \vee C$ which corresponds to the term t_1 ; in that case we write $t_2 = t_1[x \mapsto A, y \mapsto D]$. We see that we also have $u_2 = u_1[x \mapsto 3 + 5]$ and $v_2 = v_1[x \mapsto 1, y \mapsto (3 :: (4 :: \mathsf{nil}))]$.

It can be useful to see terms as trees, where internal nodes are operator symbols taking their arguments as children, and leaves are either variables or constant operators. Figure 1 shows what the terms defined above would look like.

Looking at terms of trees, we can see that there is no syntactical difference between terms of different languages and they are fully distinguished by which operators are used. We can therefore study problems of unification and matching in a uniform way by abstracting over the set of operators in use in a given language. We call this set the *signature* of this language. The languages of t_2 and u_2 are characterized respectively by the signatures Σ_t and Σ_u defined as follows.

$$\begin{aligned} \Sigma_t &= \{ \vee : (2), \wedge : (2), \implies : (2), \neg : (1) \} \\ \Sigma_u &= \{ + : (2), \times : (2), - : (2), / : (2) \} \end{aligned}$$

We assigned a number, called *arity*, to each operator in each language, which represent the number of argument the operator has. This notion of arity is insufficient if we want to define a signature for the term v_2, because there are two distinct types of terms in its language: numbers and lists. In that case, we need a way to specify the fact that the operator :: only accepts a number as its first argument, and a list as its second argument, and that it outputs a list as a result. If we denote the type of lists by $\mathbb{L}$ and the type of numbers by $\mathbb{N}$, we can define a

$$\Sigma_v = \{ (::) : \mathbb{N} \times \mathbb{L} \to \mathbb{L} \}.$$

We have just (informally) defined what is called *first-order abstract syntax*, the setting where first-order unification is studied. For the purpose of strudying second-order matching, we will introduce, later, an even more refined notion of arity that allows us to work with terms binding variables (sections 1.4 and 2.2).

1.2 First-order unification

Let us present first-order unification using the signature of simple types that is used in ML-like functional programming languagues,

$$\Sigma = \{\mathsf{N}(1), \to (2)\},$$

where N is the type of integers, and $x \to y$ denotes the type of functions from x to y. ML-like languages support generic types, that is types that are not closed (i.e. that contain type variables). So a typical program might, at a given time, have in its context a function f_1 of type $x \to x$, and a term f_2 of type y. When encountering the expression $(f_1 f_2)$, the type-checker needs to make sure that the type of f_2 (i.e. y) and the type of the argument of f_1 (i.e. x) are the same. It therefore needs to *unify* the two type expressions, or deem the program ill typed if it is impossible to unify them. In this example, a possible substitution is $[x \mapsto \mathsf{N}, y \mapsto \mathsf{N}]$ which makes f_1 of type $\mathsf{N} \to \mathsf{N}$ and f_2 of type N. But another possible substitution is $[x \mapsto y]$, which is *more general* that the first one because it lets the type checker avoid commiting to a choice for the variable y : it can decide it is indeed N later, or make another choice for it depending on future information. In this situation we say that the second substitution is *more general* than the first one.

We can ask two questions: if an equation has a solution, is there a always a "best" substitution in the sense that it is more general than any other solution substitution? And if the answer is yes, can we algorithmically find this most general substitution? Robinson (1965) proved that the existence of a solution is decidable, and that any solvable first-order unification problem has a computable most general solution.

1.3 First-order matching

Matching is a special case of unification, where in an equation $t = u$, only t may contain variables. In other words, u is closed and we are looking for a substitutionσ such that $t[\sigma] = u$.

Since first-order unification is decidable, so is first-order matching. But we are more interesed in the latter as an expressive construct in functional programming languages : pattern matching.

(a) Seeing w as a first-order term

(b) Incorporating variable binding in w's representation

Figure 2: Representing the term w by a tree

It allows programmers to deconstruct terms of a sum type (which are, in their simplest form, first-order terms over a user-defined signature) in an expressive way. For example, if we write

$$\begin{aligned} &\texttt{match}\ 1 :: (2 :: (3 :: (4 :: \mathsf{nil}))) \ \texttt{with} \\ &\quad \mid\ x :: (2 :: y) \longrightarrow x :: y \\ &\quad \mid\ x \longrightarrow x, \end{aligned}$$

we obtain the solution substitution $[x \mapsto 1, y \mapsto (3 :: 4 :: \mathsf{nil})]$, and therefore this expression evaluates to the list $[1; 3; 4]$.

1.4 Second-order syntax

We would like to extend the pattern matching construct to not only work on inductive terms of a sum type, but also on terms representing programs. However, first-order abstract syntax is no expressive enough to represent programs, as it lacks the essential notion of variable binding. For example, consider the following program in a language having only anonymous functions, applications and additions as constructs.

$$w = (\texttt{fun}\ x.\ x + x)y$$

If we want to look at it as a first-order term, we might define the signature

$$\Sigma_{\mathsf{bad}} = \{\ \texttt{fun} : \mathsf{T} \times \mathsf{T} \to \mathsf{T}, \texttt{app} : \mathsf{T} \times \mathsf{T} \to \mathsf{T}\ \}, + : \mathsf{T} \times \mathsf{T} \to \mathsf{T}\ \}$$

where T is the unique type of terms. We would then represent the term w by the tree in fig. 2a. We see already see two problems in our approach. First, our term seems to have two variables, x and y. But we did not intend for x to be an unknown in our term, it is *bound* in the argument of the $\texttt{fun}$ operator and we do not want to be able to substitute it. In fact, we want to be able to say that $w = \texttt{fun}\ z.\ z + z$ for any variable z. Following this observation, we see that what we considered as the first argument to the $\texttt{fun}$ operator, the variable "x", is not a subterm, but only an indicator that the only subterm of $\texttt{fun}$, $x + x$, binds the variable x.

We therefore need to enrich our signatures with a construct allowing us to specify that an operator's argument binds a certain number of variables on one hand, and enrich our syntax with a generic binding construct on the other hand. A good signature for our small language here would be

$$\Sigma_{\text{good}} = \{\ \texttt{fun} : (\mathsf{T} \to \mathsf{T}) \to \mathsf{T}, \texttt{app} : \mathsf{T} \times \mathsf{T} \to \mathsf{T}\ \}, + : \mathsf{T} \times \mathsf{T} \to \mathsf{T}\ \}.$$

The operator `fun` now has only one argument of type $\mathsf{T} \times \mathsf{T}$, meaning that it represents a term of type T in which a variable of type T is bound. The term w can then be seen as the tree in section 1.4, in which the node (λx) represents binding. A variable x appearing in a second-order tree can now be in one of two cases: it either has a close ancestor node λx, in which case we say it is *bound* by that node, or it has no ancestor of that form and we say it is *free* in the term represented by that tree. The substitution we define for second-order terms only replaces free variables, and does not have access to bound variables. We present these concepts more formally in section 2.

Huet (1976) proved that second-order unification is undecidable, and that second-order matching is still decidable in the second-order.

1.5 Second-order matching

Second-order matching is decidable, and when a solution exists, it is computable. But second-order matching differs from the first-order case in that a solution is no longer a most general substitution, but a most general set of independent substitutions.

We give a simple example with the following matching problem.

$$\begin{aligned} \Sigma &= \{\ A : \mathsf{T}\ \} \\ t &= f(A) \\ u &= A \end{aligned}$$

The variable f here is of type $\mathsf{T} \to \mathsf{T}$: it takes an argument of type T and the result must match the term A of type T. Therefore we must replace f with a term binding one variable. One solution is the substitution

$$\sigma_1 = [f \mapsto \lambda x.x],$$

which maps f to the identity function. Another solution is the substituion

$$\sigma_2 = [f \mapsto \lambda x.A],$$

which maps f to the constant function mapping every argument to A. Both substitutions make the term t match the term u, but neither one is more general than the other.

In section 3.1, we show that the solution set still has a structure and can be charactarized in a straightforward way. In section 3.2, we present an inference system that can also be seen as an algorithm that computes the solution set of a matching problem.

2 Higher-order syntax

In this section, we formally define *higher-order abstract syntax*, a general setting in which we can study higher-order unification problems. We define higher-order terms as a special class of simply typed λ-terms, the class of *long normal form* terms, also called β-short η-long terms.

2.1 Mathematical prolegomena

Definition 2.1.1 (Valuations)**.** Let V be a set. A *V-valuation* is a pair $f = (\mathsf{dom}(f), \underline{f})$ where $\mathsf{dom}(f)$ is a finite set and $\underline{f}$ is a (total) map from $\mathsf{dom}(f)$ to V. Let A be a set. An (A, V)-valuation f is a valuation satisfying $\mathsf{dom}(f) \subseteq A$.

Given a V-valuation f and $x \in \mathsf{dom}(f)$, we write $f(x)$ for $\underline{f}(x)$. For an (A, V)-valuation f, $x \in A$ and $v \in V$, we define:

$$f, x \mapsto v \quad := \quad (\mathsf{dom}(f) \cup \{x\}, \underline{f'}),$$

where $\underline{f'} : \mathsf{dom}(f) \cup \{x\} \to V$ maps every $y \in \mathsf{dom}(f) \backslash \{x\}$ to $f(y)$ and x to v. We sometimes write $x \in f$ instead of $x \in \mathsf{dom}(f)$. We also define

$$f - x \quad := \quad (\mathsf{dom}(f) \backslash \{x\}, \underline{f}_{|\mathsf{dom}(f) \backslash \{x\}}).$$

For two (A, V)-valuations f and g, we say that $f \subseteq g$ if $\mathsf{dom}(f) \subseteq \mathsf{dom}(g)$ and for all $x \in \mathsf{dom}(f)$, $\underline{f}(x) = \underline{g}(x)$.

Definition 2.1.2. We equip ourselves with a countably infinite set $\mathbb{A}$ of *variables*.

2.2 Signatures

We start by defining simple types over a set and valuations.

Definition 2.2.1 (Simple types)**.** Let T be a set. The set $T^{\to}$ of *(simple) types over* T is defined by the following grammar.

$$T^{\to} \ni \tau, \upsilon ::= A \in T \mid \tau \to \upsilon$$

We abbreviate the type $\tau_1 \to (\tau_2 \to (\ldots (\tau_n \to A)) \ldots)$ as $\tau_1 \times \ldots \times \tau_n \to A$.
We define the order o of a type in $T^{\to}$ by induction.

$$\mathsf{o}(A) = 0$$
$$\mathsf{o}(\tau \to \upsilon) = \mathsf{max}(\mathsf{o}(\tau) + 1, \upsilon)$$

We extend this definition to any $T^{\to}$-valuation Γ by setting

$$\mathsf{o}(\Gamma) \quad := \quad \mathsf{max}_{x \in \mathsf{dom}(\Gamma)}\ \mathsf{o}(\Gamma(x)).$$

Definition 2.2.2 (Higher-order signature)**.** A *higher-order signature* is a couple $\Sigma = (T_\Sigma, \mathsf{Op}_\Sigma)$ where T_Σ is a set of *primitive types*, and Op_Σ is a $T_\Sigma^{\to}$-valuation of *operators*. We write $|\Sigma|$ for $\mathsf{dom}(\mathsf{Op}_\Sigma)$, and call the elements of this set *operator symbols*.

We assume that for any signature Σ under consideration, the sets $|\Sigma|$ and $\mathbb{A}$ are disjoint.

2.3 The λ-terms

We fix a higher-order signature Σ for the remaining of this section.

Definition 2.3.1 (Untyped λ-terms)**.** We define the set $\Lambda^{\mathsf{u}}_\Sigma$ of untyped λ-terms over Σ by the following rules.

$$\begin{array}{rr} \Lambda^{\mathsf{u}}_\Sigma \ni t, u ::= & \textbf{λ-terms over } \Sigma \\ \mid x & \text{variable } (x \in \mathbb{A}) \\ \mid F & \text{constant symbol } (F \in |\Sigma|) \\ \mid (t)u & \text{application} \\ \mid \lambda x.t & \text{abstraction} \end{array}$$

We abbreviate, when convenient, $\lambda x_1 . \ldots \lambda x_n . t$ by $\lambda x_1 \ldots x_n . t$ and $((\ldots(((t_1)t_2)t_3)\ldots)t_{n-1})t_n$ by $t_1 \ldots t_n$ or $t_1(t_2, \ldots, t_n)$. We always consider terms up to a renaming of their bound variables, and abide by the Barendregt convention.

Definition 2.3.2 (Typing contexts)**.** A *typing context*, denoted Γ or Δ, is a $(\mathbb{A}, T^{\rightarrow}_\Sigma)$-valuation.

Definition 2.3.3 (Typed λ-terms)**.** We define the typing judgment $\boxed{\Gamma \vdash t : \tau}$ inductively by the following system of inference rules.

$$\frac{x \in \Gamma}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{F \in |\Sigma|}{\Gamma \vdash F : \mathrm{Op}_\Sigma(F)}$$

$$\frac{\Gamma \vdash t : \tau \to \upsilon \qquad \Gamma \vdash u : \tau}{\Gamma \vdash (t)u : \upsilon} \qquad \frac{\Gamma, x : \tau \vdash t : \upsilon}{\Gamma \vdash \lambda x.t : \tau \to \upsilon}$$

We define the set of typed λ-terms for Σ by

$$\Lambda_\Sigma \quad := \quad \{t \in \Lambda^{\mathsf{u}}_\Sigma \mid \exists \Gamma, \exists \tau,\ \Gamma \vdash t : \tau\}.$$

And we overload the notation with

$$\begin{array}{rcl} \Lambda_\Sigma(\Gamma) & := & \{t \in \Lambda^{\mathsf{u}}_\Sigma \mid \exists \tau,\ \Gamma \vdash t : \tau\} \text{ and} \\ \Lambda_\Sigma(\Gamma, \tau) & := & \{t \in \Lambda^{\mathsf{u}}_\Sigma \mid\ \Gamma \vdash t : \tau\}. \end{array}$$

Definition 2.3.4 (Free variables)**.** Let $t \in \Lambda^{\mathsf{u}}_\Sigma$. We define the set $\mathsf{FV}(t)$ of *free variables* of the term t by induction.

$$\begin{array}{rr} \mathsf{FV}(x) = \{x\} & (x \in \mathbb{A}) \\ \mathsf{FV}(F) = \emptyset & (F \in |\Sigma|) \\ \mathsf{FV}((t)u) = \mathsf{FV}(t) \cup \mathsf{FV}(u) & \\ \mathsf{FV}(\lambda x.t) = \mathsf{FV}(t) \backslash \{x\} & \end{array}$$

Definition 2.3.5 (Simple substitution)**.** We define a *simple substitution* as an $(\mathbb{A}, \Lambda^{\mathsf{u}}_{\Sigma})$-valuation. We use the symbols σ and θ for substitutions. For $x \in \mathbb{A}$, we define $\sigma.x$ as follows.

$$\sigma.x \coloneqq \begin{cases} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma) \\ x & \text{otherwise} \end{cases}$$

We denote the set of simple subtitutions over Σ by $\mathcal{S}^{\mathsf{u}}_{\Sigma}$, and the unique substitution with an empty domain by id. For $\sigma \in \mathcal{S}^{\mathsf{u}}_{\Sigma}$, we define

$$\mathsf{FV}(\sigma) \coloneqq \bigcup_{x \in \sigma} \mathsf{FV}(\sigma(x))$$

We say that σ is closed if $\mathsf{FV}(\sigma) = \emptyset$.

Definition 2.3.6 (Applying a substitution)**.** Let $\sigma \in \mathcal{S}^{\mathsf{u}}_{\Sigma}$. We define $-[\sigma] : \Lambda^{\mathsf{u}}_{\Sigma} \to \Lambda^{\mathsf{u}}_{\Sigma}$ by induction.

$$\begin{aligned} x[\sigma] &= \sigma.x \\ F[\sigma] &= F \\ (t)u[\sigma] &= (t[\sigma])u[\sigma] \\ \lambda x.t[\sigma] &= \lambda y.t[\sigma, x : y] \qquad \text{where } y \text{ does not appear in } t \text{ nor in } \sigma \end{aligned}$$

Definition 2.3.7 (Well-typed substitution)**.** We define the set of well typed substitutions with respect to an output context Δ and an input context Γ as follows.

$$\mathcal{S}_{\Sigma}(\Delta, \Gamma) \quad \coloneqq \quad \{\sigma \in \mathcal{S}^{\mathsf{u}}_{\Sigma} \mid \forall x \in \mathbb{A},\ \Delta \vdash \sigma.x : \Gamma(x)\}$$

Lemma 2.3.8 (Relation between free variables and typing contexts)**.** *If* $\Gamma \vdash t : \tau$*, then* $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$*. And if* $\sigma \in \mathcal{S}_{\Sigma}(\Delta, \Gamma)$*, then* $\mathsf{FV}(\sigma) \subseteq \mathsf{dom}(\Delta)$*.*

Proof. Straightforward by induction on the typing judgement. □

Proposition 2.3.9. *Let* $t \in \Lambda_{\Sigma}(\Delta, \tau)$ *and* $\sigma \in \mathcal{S}_{\Sigma}(\Delta, \Gamma)$*. We have* $\Gamma \vdash t[\sigma] : \tau$*.*
In other words, we have

$$-[=] : \Lambda_{\Sigma}(\Gamma, \tau) \times \mathcal{S}_{\Sigma}(\Delta, \Gamma) \to \Lambda_{\Sigma}(\Delta, \tau)$$

for all $\tau \in T^{\to}_{\Sigma}$ *and all contexts* Γ *and* Δ*.*

Proof. This is shown by a routine induction on the term t. A detailed proof can be found in many works treating λ-calculus, see for example the lemma 12.18 in the book of Hindley and Seldin (1986). □

Definition 2.3.10 (Composing substitutions)**.** Let $\sigma_1, \sigma_2 \in \mathcal{S}_{\Sigma}$. We define

$$\sigma_2 \circ \sigma_1 \quad \coloneqq \quad (\mathsf{dom}(\sigma_1) \cup \mathsf{dom}(\sigma_2), x \mapsto (\sigma_1.x)[\sigma_2])$$

Proposition 2.3.11. *Let* $\sigma_1, \sigma_2 \in \mathcal{S}_{\Sigma}$ *and* $t \in \Lambda^{\mathsf{u}}_{\Sigma}$*. We have* $t[\sigma_2 \circ \sigma_1] = t[\sigma_1][\sigma_2]$

Proof. The proof for this proposition is not trivial, and can be found in detail in the book of Krivine (1993) □

Proposition 2.3.12. *Let* $\sigma_1 \in \mathcal{S}_\Sigma(\Delta, \Gamma)$ *and* $\sigma_2 \in \mathcal{S}_\Sigma(\Theta, \Delta)$. *We have* $\sigma_2 \circ \sigma_1 \in \mathcal{S}_\Sigma(\Theta, (\Delta, \Gamma))$

Proof. Straightforward by induction on the term t. □

2.4 The λ-calculus

Definition 2.4.1 (One-hole context)**.** We define the set $\mathcal{C}_\Sigma$ of *one-hole contexts* by induction.

$$
\begin{array}{rll}
\mathcal{C}_\Sigma \ni K ::= & & \textbf{one-hole contexts over } \Sigma \\
& \mid \Box & \text{empty context} \\
& \mid (t)K \mid (K)t & \text{application } (t \in \Lambda^{\mathsf{u}}_\Sigma) \\
& \mid \lambda x.K & \text{abstraction}
\end{array}
$$

We use the same notations that we introduced for λ-terms, and we define by induction the operation $C[t]$ of filling a context C with a term t and a judgment $\boxed{(\Box : \Delta \vdash \tau) \mid \Gamma \vdash\ K : \upsilon}$ for well typed one-hole contexts as follows.

$$
\begin{aligned}
\Box[t] &= t \\
(t)K[u] &= (t)K[u] \\
(K)t[u] &= (K[u])t \\
\lambda x.K &= \lambda x.K[u]
\end{aligned}
$$

$$
\frac{}{(\Box : \Gamma \vdash \upsilon) \mid \Gamma \vdash\ \Box : \upsilon}
\qquad
\frac{(\Box : \Delta \vdash \upsilon) \mid \Gamma, x : \tau_1 \vdash\ K : \tau_2}{(\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ \lambda x.K : \tau_1 \to \tau_2}
$$

$$
\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad (\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ K : \tau_1}{(\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ (t)K : \tau_2}
$$

$$
\frac{(\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ K : \tau_1 \to \tau_2 \qquad \Gamma \vdash t : \tau_1}{(\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ (K)t : \tau_2}
$$

As for terms and substitutions, we denote the set of one-hole contexts that are of type τ given a context Γ and a hole of type υ having access to an additional context Δ by $\mathcal{C}_\Sigma(\Gamma, \tau, \Delta, \upsilon)$.

Lemma 2.4.2. *The following rule is admissible.*

$$
\frac{(\Box : \Delta \vdash \upsilon) \mid \Gamma \vdash\ E : \tau \qquad \Gamma, \Delta \vdash t : \upsilon}{\Gamma \vdash E[t] : \tau}
$$

Proof. Straightforward by induction on the derivation of the one-hole context typing judgment. □

Definition 2.4.3 ($\beta\eta$ reduction)**.** We define the set of *atomic β-reductions* and *atomic η-reductions* as follows.

$$\rightsquigarrow_\beta \coloneqq \{((\lambda x.t)u, t[x := u]) \mid t, u \in \Lambda^{\mathsf{u}}_\Sigma\}$$
$$\rightsquigarrow_\eta \coloneqq \{(\lambda x.(t)x, t) \mid t \in \Lambda^{\mathsf{u}}_\Sigma\}$$

We then define the contextual closure of these two atomic relations under one-hole contexts in the following way.

$$\to_\beta \coloneqq \{(E[t], E[t']) \mid t \rightsquigarrow_\beta t', E \in \mathcal{C}_\Sigma\}$$
$$\to_\eta \coloneqq \{(E[t], E[t'] \mid t \rightsquigarrow_\eta t', E \in \mathcal{C}_\Sigma\}$$

Finally, we define $\to_{\beta\eta} \quad \coloneqq \quad \to_\beta \cup \to_\eta$ and for $r \in \{\beta, \eta, \beta\eta\}$, we define the relation $\to_r^*$ as the reflexive-transitive closure of $\to_r$ and the relation $\equiv_r$ as the symmetric, reflexive, transitive closure of $\to_r$. For any two terms t and u, we say that they are r-equivalent when we have $t \equiv_r u$, and we say that t r-reduces to u (or that u is an r-reduct of t) when we have $t \to_r^* u$. We say that a term t is r-normal if t has no r-reduct.

Theorem 2.4.4 (Subject reduction)**.** *If $\Gamma \vdash t : \tau$ and $t \to_{\beta\eta} u$, then $\Gamma \vdash u : \tau$*

Proof. We start by showing that $\rightsquigarrow_\beta$ and $\rightsquigarrow_\eta$ are compatible with the type system using theorem 2.3.9. We then get the result by applying theorem 2.4.2. □

Theorem 2.4.5 ($\beta\eta$-normal form)**.** *$\beta\eta$-reduction is confluent and strongly normalizable for typed terms. This means that for every well-typed term $t \in \Lambda_\Sigma$, there exists a unique term $\mathsf{Nf(t)}$ such that $t \to_{\beta\eta}^* \mathsf{Nf(t)}$ and $\mathsf{Nf(t)}$ is $\beta\eta$-normal.*

Proof. By combining theorems 12.35, 7.14 and 7.16 of the book of Hindley and Seldin (1986). □

Definition 2.4.6. We define the set $\Lambda^{\mathsf{n}}_\Sigma$ of normal terms by

$$\Lambda^{\mathsf{n}}_\Sigma \coloneqq \{\mathsf{Nf(t)} \mid t \in \Lambda_\Sigma\}$$

2.5 Higher-order syntax terms

Proposition 2.5.1. *Let $t \in \Lambda^{\mathsf{n}}_\Sigma(\Gamma, \tau)$.*

1. *There exist unique $n, m \geq 0$ and $x_1, \ldots, x_n \in \mathbb{A}$ and $u_1, \ldots, u_k \in \Lambda^{\mathsf{n}}_\Sigma$ and $a \in \Sigma \cup \mathbb{A}$ such that*
$$t = \lambda x_1 \ldots x_n.a(u_1, \ldots, u_k).$$
2. *τ is in the form $\tau_1 \to \ldots \to \tau_n \to \upsilon$ for some $\tau_1, \ldots, \tau_n, \upsilon \in T^{\to}_\Sigma$.*

Proof. The uniqueness is obtained by inverting the rules for constructing terms in addition to reasoning modulo renaming of bound variables. To prove existence, we start by observing that by an immediate induction, we get $t = \lambda x_1 \ldots x_n.u(u_1, \ldots, u_k)$ for some $u \in \Lambda^{\mathsf{u}}_\Sigma$ that is not in the form $(\cdot)\cdot$. If for some $i \in \{1, \ldots k\}$ and $u' \in \Lambda_\Sigma$ we have $u_i \to_{\beta\eta}^* u'$, then we would have $t \to_{\beta\eta}^* \lambda x_1 \ldots x_n.u(u_1, \ldots, u_{i-1}, u', u_{i+1}, \ldots, u_k)$ which is not possible since t is $\beta\eta$-normal

; therefore $\forall i \in \{1, \dots, k\}, u_i \in \Lambda^{\mathsf{n}}_{\Sigma}$. To conclude, we only have to show that u is not in the form $\lambda x.u'$. It can't be in this form because otherwise t would admit the atomic β-reduction $u \rightsquigarrow_{\beta} u'[x := u_1]$.
We prove the second point by inverting n times the typing judgment for t. □

Definition 2.5.2 (Long normal form)**.** Let $t = \lambda x_1 \dots x_n.a(u_1, \dots, u_k) \in \Lambda^{\mathsf{n}}_{\Sigma}(\Gamma, \tau)$, and $\tau_1 \to \dots \to \tau_m \to A := \tau$. By theorem 2.5.1, $m \geq n$. We define the long normal form of t with respect to Γ and τ, denoted $\mathsf{N}_{\Gamma,\tau}(t)$, by

$$\mathsf{N}_{\Gamma,\tau}(t) \coloneqq \lambda x_1 \dots x_n x_{n+1} \dots x_m.a(\mathsf{N}_{\Gamma,\tau}(u_1), \dots, \mathsf{N}_{\Gamma,\tau}(u_k), x_{n+1}, \dots, x_m),$$

where $x_{n+1}, \dots, x_m$ are not free in t and different from $x_1, \dots, x_n$.
For $t \in \Lambda_{\Sigma}(\Gamma, \tau)$, we define

$$\mathsf{N}_{\Gamma,\tau}(t) \coloneqq \mathsf{N}_{\Gamma,\tau}(\mathsf{Nf}(\mathsf{t}))$$

In this form, we call $(x_1, \dots, x_m)$ the *binder* of t, and we call a the *head* of t.

Definition 2.5.3 (Higher-order syntax terms)**.** We define the set $\mathcal{T}_{\Sigma}$ of higher-order terms over Σ by

$$\mathcal{T}_{\Sigma} \quad \coloneqq \quad \prod_{\Gamma,\tau} \{\mathsf{N}_{\Gamma,\tau}(t) \mid t \in \Lambda_{\Sigma}(\Gamma, \tau)\}$$

Proposition 2.5.4 (Inductive definition of higher-order terms)**.** *The set $\mathcal{T}_{\Sigma}$ of higher-order terms over Σ is charactarized by the judgement $\boxed{\Sigma \mid \Gamma \vdash^{\mathsf{n}} t : \tau}$ which is defined as follows.*

$$\frac{\Sigma \mid \Gamma, x : \tau \vdash^{\mathsf{n}} t : \upsilon}{\Sigma \mid \Gamma \vdash^{\mathsf{n}} \lambda x.t : \tau \to \upsilon} \qquad \frac{\Sigma \cup \Gamma \ni a : \tau_1 \to \dots \to \tau_n \to A \qquad (\Sigma \mid \Gamma \vdash^{\mathsf{n}} u_i : \tau_i)_{1 \leq i \leq n}}{\Sigma \mid \Gamma \vdash^{\mathsf{n}} a(u_1, \dots, u_n) : A}$$

Proof. Result of theorem 2.5.1. □

Definition 2.5.5 (Structural order)**.** We define $\preceq : \ \mathcal{T}_{\Sigma} \times \mathcal{T}_{\Sigma}$ as the order induced by the inductive structure of higher-order terms.

Definition 2.5.6 (Normal substitution)**.** Let $\sigma \in \mathcal{S}_{\Sigma}(\Delta, \Gamma)$. We say that σ is normal if for all $x \in \mathsf{dom}(\sigma)$, we have $\sigma(x) \in \mathcal{T}_{\Sigma}(\Gamma, \Delta(x))$. We denote

$$\mathcal{S}^{\mathsf{n}}_{\Sigma}(\Delta, \Gamma) \coloneqq \{\sigma \in \mathcal{S}_{\Sigma}(\Delta, \Gamma) \mid \ \sigma \text{ is normal } \}$$

Definition 2.5.7 (Action of a normal substitution)**.** Let $\sigma \in \mathcal{S}^{\mathsf{n}}_{\Sigma}(\Delta, \Gamma)$ and $t \in \mathcal{T}_{\Sigma}(\Delta, \tau)$. We define the action of σ on t, denoted $t[\sigma]_{\mathsf{h}}$, as the long normal form of $t[\sigma]$.

$$t[\sigma]_{\mathsf{h}} \coloneqq \mathsf{N}_{\Gamma,\tau}(t[\sigma])$$

The notation $[-]_{\mathsf{h}}$ for the normal form of a substitution is justified by the following proposition, which describes *hereditary substitution*, an algorithm for substitution that ensures normalization of the resulting term on the fly.

Proposition 2.5.8 (definition of the action of a normal substitution)**.** *The action of a normal substitution is characterized by the following recursive definition.*

$$\begin{aligned} \lambda x.t[\sigma]_{\mathsf{h}} &= \lambda y.t[\sigma, x := y]_{\mathsf{h}} && \textit{(y fresh in t and } \sigma\textit{)} \\ a(u_1, \dots, u_n)[\sigma]_{\mathsf{h}} &= t[\sigma, x_1 := u_1, \dots, x_n := u_n]_{\mathsf{h}} && \textit{when } \sigma(a) = \lambda x_1 \dots x_n.t \\ a(u_1, \dots, u_n)[\sigma]_{\mathsf{h}} &= a(u_1[\sigma]_{\mathsf{h}}, \dots, u_n[\sigma]_{\mathsf{h}}) && \textit{when } a \notin \mathsf{dom}(\sigma) \end{aligned}$$

Proof. See the article of Keller and Altenkirch (2011). □

3 Matching problem

3.1 The problem

We start by presenting the matching problem, and exhibiting the structure of its solutions, and defining a notion of *canonical* solution that can be precisely described.

Definition 3.1.1 (Matching)**.** Let $t_1 \in \mathcal{T}_\Sigma(\Gamma, \tau)$ and $t_2 \in \mathcal{T}_\Sigma(\emptyset, \tau)$. We say that t_2 *matches* t_1 if there exists $\Gamma' \subseteq \Gamma$ and $\sigma \in \mathcal{S}^{\mathsf{n}}_\Sigma(\emptyset, \Gamma')$ such that $t_1[\sigma]_{\mathsf{h}} = t_2$. In that case we write $t_1 \leq^\tau t_2$ and say that σ is a *filter* for the pair (t_1, t_2).

Definition 3.1.2 (Matching problem)**.** A *matching problem* is a pair $P = (\Gamma, |P|)$ where Γ is a context and $|P|$ is a finite set of triplets (t, u, τ) such that $\Gamma \vdash t : \tau$ and $\cdot \vdash u : \tau$. We denote each $(t, u, \tau) \in |P|$ as $t \leq^\tau_? u$.
A substitution $\sigma \in \mathcal{S}_\Sigma$ is a filter for the problem P if it is a filter for each pair in $|P|$.

Definition 3.1.3 (Substitution preorder)**.** Let $\sigma_1, \sigma_2 \in \mathcal{S}_\Sigma$. We write that σ_1 is *more general* than σ_2 if there exists $\theta \in \mathcal{S}_\Sigma$ such that $\sigma_2 = \theta \circ \sigma_1$. In that case, we write $\sigma_1 \leq \sigma_2$.

Proposition 3.1.4. *The relation $\leq$ is a preorder for the set $\mathcal{S}_\Sigma$. We denote $[\![\sigma]\!]$ the equivalence class of a substitution σ with respect to the equivalence relation induced by $\leq$.*

Proof. Using theorem 2.3.11, we have that substitution composition is associative and admits id as a neutral element. We therefore respectively get that $\leq$ is transitive and reflexive. □

Proposition 3.1.5 (Triangular-form for closed substitutions)**.** *Let $\sigma \in \mathcal{S}_\Sigma(\emptyset, \Gamma)$ with $\mathsf{dom}(\sigma) = \{x_1, \dots, x_n\}$. We have*

$$\sigma \quad = \quad \{x_1 \mapsto \sigma(x_1)\} \circ \dots \circ \{x_n \mapsto \sigma(x_n)\}.$$

Proof. This is a direct consequence of the definition of composition combined with the fact that for any closed term $t \in \Lambda^{\mathsf{u}}_\Sigma$ and any substitution $\sigma \in \mathcal{S}_\Sigma$, $t[\sigma] = t$. □

Definition 3.1.6. Let P be a matching problem. We denote by $\mathcal{F}(P)$ the set of all filters for P. We say that a set $S \subseteq \mathcal{F}(P)$ is:

1. *complete* if for all filters $\sigma \in \mathcal{F}(P)$, there exists a filter $\sigma' \in S$ such that $\sigma' \leq \sigma$;
2. *minimal* if for all distinct pairs of filters $\sigma_1, \sigma_2 \in S$, we have $\sigma_1 \not\leq \sigma_2$.

Definition 3.1.7 (Solution to a matching problem)**.** A *solution* to a matching problem P is a set $S \subseteq \mathcal{F}(P)$ that is complete and minimal.

Proposition 3.1.8. *Let P be a matching problem with two solutions S_1, S_2. We have $[\![S_1]\!] = [\![S_2]\!]$.*

Proof. Since S_1 and S_2 have symmetrical roles, it is sufficient to prove that$[\![S_1]\!] \subseteq [\![S_2]\!]$. Let $\sigma_1 \in S_1$. Since S_2 is complete, there exists $\sigma_2 \in S_2$ such that $\sigma_2 \leq \sigma_1$. Since S_1 is complete there exists $\sigma_1' \in S_1, \sigma_1'$. By transitivity we get $\sigma_1 \leq \sigma_1'$, but S_1 is minimal, so $\sigma_1 = \sigma_1'$. Finally, we have $\sigma_1 \leq \sigma_2 \leq \sigma_1$, therefore $[\![\sigma_1]\!] = [\![\sigma_2]\!]$. □

Definition 3.1.9 (Canonical filter)**.** Let $t \in \mathcal{T}_\Sigma(\Gamma, \tau)$, $u \in \mathcal{T}_\Sigma(\emptyset, \tau)$ and σ a filter for the pair (t, u). For each $x \in \mathsf{dom}(\sigma)$, we say that x is (t, u, σ)*-essential* if

$$t[\sigma - x]_\mathsf{h} \neq u.$$

Given a matching problem P and a filter σ for P, σ is *canonical* with respect to P if

$$\forall x \in \mathsf{dom}(\sigma),\ \exists t \leq^\tau_? u \in |P|, \quad x \text{ is } (t, u, \sigma)\text{-essential}$$

Proposition 3.1.10 (Canonical filter map)**.** *Let P be a matching problem. For every filter σ for P, there exists a unique canonical filter, which we denote $\mathsf{c}_P(\sigma)$, such that $\mathsf{c}_P(\sigma) \leq \sigma$. Additionally, $\mathsf{c}_P(\sigma) \subseteq \sigma$.*

Proof. We proceed by well-founded induction on the length of $\mathsf{dom}(\sigma)$. We reason by cases on the canonicity of σ.

- If σ is already canonical, we only have to prove uniqueness. Let σ' be a canonical filter such that $\sigma' \leq \sigma$. By definition, there exists $\theta \in \mathcal{S}_\Sigma$ such that $\sigma = \theta \circ \sigma'$. Note that since σ and σ' are closed, for all $x \in \mathsf{dom}(\sigma)$, either $x \in \mathsf{dom}(\sigma')$ and $\sigma(x) = \sigma'(x)$, or $x \in \mathsf{dom}(\theta) \backslash \mathsf{dom}(\sigma')$ and $\sigma(x) = \theta(x)$. We reason by cases.
  - If the set $\mathsf{dom}(\theta) \backslash \mathsf{dom}(\sigma')$ is empty, we must have $\sigma' = \sigma$, concluding the proof.
  - Otherwise, let $x \in \mathsf{dom}(\theta) \backslash \mathsf{dom}(\sigma')$. Using theorem 2.3.11, we have, for all $t \leq^\tau_? u \in |P|$:

    $$t[\sigma - x]_\mathsf{h} = t[(\theta - x) \circ \sigma']_\mathsf{h} = t[\sigma']_\mathsf{h}[\theta - x]_\mathsf{h} = u[\theta - x]_\mathsf{h} = u$$

    hence x is not (t, u, σ)-essential, which is contradictory because σ is canonical.
- Otherwise, there exists $x \in \mathsf{dom}(\sigma)$ such that for all $t \leq^\tau_? u \in |P|$, $t[\sigma - x]_\mathsf{h} = u$. We set $\sigma' \coloneqq \sigma - x$. By induction hypothesis, there exists a canonical filter $\mathsf{c}_P(\sigma')$ for P such that $\mathsf{c}_P(\sigma') \leq \sigma'$. Using theorem 3.1.5, we have $\sigma = \{x \mapsto \sigma(x)\} \circ \sigma'$, so $\sigma \leq \sigma'$ and by transitivity $\mathsf{c}_P(\sigma') \leq \sigma$. We set $\mathsf{c}_P(\sigma) \coloneqq \mathsf{c}_P(\sigma')$. By induction hypothesis, we have $\mathsf{c}_P(\sigma') \subseteq \sigma'$; and by definition of σ', we have $\sigma' \subseteq \sigma$. Therefore $\mathsf{c}_P(\sigma) = \mathsf{c}_P(\sigma') \subseteq \sigma$. We finally show that there is no other canonical filter by using the same reasoning as the first step of this proof. □

Corollary 3.1.11 (Existence of a canonical solution)**.** *Let P be a matching problem. The set*

$$\mathcal{F}_{\mathsf{C}}(P) \coloneqq \{\mathsf{c}(\sigma) \mid \sigma \in \mathcal{F}(P)\}$$

of canonical filters for P is a solution to the problem $\mathcal{F}_{\mathsf{C}}(P)$.

Proof. By theorem 3.1.10, $\mathcal{F}_{\mathsf{C}}(P)$ is complete and minimal. □

3.2 Matching algorithm

Starting here, we restrict ourselves to a signature Σ of the second order and to contexts Γ of the first order.

Definition 3.2.1 (Second-order matching algorithm)**.** We define the judgement

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

by the inductive system of rules that follows.

$$\textsc{Introduce}\ \frac{\Gamma \mid \Delta, x : A \vdash t \leq^{A}_{?} u \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t \leq^{\tau \to A}_{?} \lambda x.u \dashv \sigma}$$

$$\textsc{Simplify}\ \frac{\Sigma \cup \Delta \ni a : \tau_1 \times \ldots \tau_n \to A \qquad (\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_{\mathsf{h}} \leq^{\tau_i}_{?} u_i \dashv \sigma_i)_{1 \leq i \leq n}}{\Gamma \mid \Delta \vdash a(t_1, \ldots, t_n) \leq^{A}_{?} a(u_1, \ldots, u_n) \dashv \sigma_n \circ \ldots \circ \sigma_1}$$

$$\textsc{Project}\ \frac{1 \leq i \leq n \qquad \Gamma \mid \Delta \vdash f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_{\mathsf{h}} \leq^{\tau}_{?} u \dashv \sigma}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^{A}_{?} u \dashv \sigma \circ \{f \mapsto \lambda x_1 \ldots x_n.x_i\}}$$

$$\textsc{Imitate}\ \frac{\begin{array}{c}(\upsilon_i =: \upsilon_i^1 \times \ldots \times \upsilon_i^{k_i} \to B_i)_{1 \leq i \leq n} \\ \theta \coloneqq \{f \mapsto \lambda x_1 \ldots x_n.F(\overrightarrow{\lambda y_1, \ldots, y_{k_i}.f_i(x_1, \ldots, x_n, y_1, \ldots, y_{k_i})})\} \\ \Sigma \ni F : \upsilon_1 \times \ldots \times \upsilon_m \to B \\ \Gamma, f_1 : \_, \ldots, f_m : \_ \mid \Delta \vdash f(t_1, \ldots, t_n)[\theta]_{\mathsf{h}} \leq^{A}_{?} F(u_1, \ldots, u_m) \dashv \sigma\end{array}}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^{A}_{?} F(u_1, \ldots, u_m) \dashv (\sigma \circ \theta) - f_1, \ldots, f_m}$$

If we can derive the judgement $\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma$, we claim that σ is a canonical substitution for the problem $t \leq^{\tau}_{?} u$ under the context Γ and with bound variables in Δ.

We only consider judgements where we have $\Delta \cup \Gamma = \emptyset$. This can always be ensured because the only rule extending Δ is the Introduce rule, which enriches Δ with a bound variable, therefore we can always rename the bound variable to be disjoint from the domain of Γ.

Definition 3.2.2. Let $t \in \mathcal{T}_{\Sigma}((\Gamma, \Delta), \tau)$ and $u \in \mathcal{T}_{\Sigma}(\Delta, \tau)$. We say that a substitution $\sigma \in \mathcal{S}_{\Sigma}(\emptyset, \Gamma)$ is is Δ-*canonical* with respect to the problem $\{t \leq^{\tau}_{?} u\}$ if for every enumeration $x_1, \ldots x_n$ of $\mathsf{dom}(\Delta)$, σ is canonical with respect to

$$\lambda x_1 \ldots x_n.t \leq^{\Delta(x_1) \times \cdots \times \Delta(x_n) \to \tau}_{?} \lambda x_1 \ldots x_n.u.$$

It is clear that σ is a canonical filter (in the sense of theorem 3.1.9) with respect to $\{t \leq^{\tau}_{?} u\}$ if and only if it is $\emptyset$-canonical with respect to the same problem.

Theorem 3.2.3 (Soundness). *If $\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma$, then σ is a Δ-canonical filter with respect to the problem $\{t \leq^{\tau}_{?} u\}$.*

Proof. We prove it by induction on the derivation of the judgement.

- Case $\dfrac{\Gamma \mid \Delta, x : A \vdash t' \leq^{A}_{?} u' \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t' \leq^{\tau' \to A}_{?} \lambda x.u' \dashv \sigma}$: by the Barendregt variable convention, we may assume that x is fresh enough, so that

$$(\lambda x.t')[\sigma]_{\mathsf{h}} = \lambda x.t'[\sigma, x \mapsto x]_{\mathsf{h}} = \lambda x.t'[\sigma]_{\mathsf{h}}.$$

  By induction hypothesis, $t'[\sigma]_{\mathsf{h}} = u'$, so

$$t = (\lambda x.t')[\sigma]_{\mathsf{h}} = \lambda x.u' = u.$$

  And for $y \in \mathsf{dom}(\sigma)$, by induction hypothesis, $t'[\sigma - y]_{\mathsf{h}} \neq u'$. Therefore we have

$$t[\sigma - y]_{\mathsf{h}} = \lambda x.t'[\sigma - y]_{\mathsf{h}} \neq \lambda x.u' = u.$$

  So σ is a Δ-canonical filter with respect to $t \leq^{\tau}_{?} u$.

- Case $\dfrac{\Sigma \cup \Delta \ni a : \tau_1 \times \ldots \tau_n \to A \qquad (\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_{\mathsf{h}} \leq^{\tau_i}_{?} u_i \dashv \sigma_i)_{1 \leq i \leq n}}{\Gamma \mid \Delta \vdash a(t_1, \ldots, t_n) \leq^{A}_{?} a(u_1, \ldots, u_n) \dashv \sigma_n \circ \ldots \circ \sigma_1}$:
  We show that σ is a filter for the pair $(t \leq^{\tau}_{?} u)$:

$$\begin{aligned}
t[\sigma]_{\mathsf{h}} &= a(t_1, \ldots, t_n)[\sigma_n \circ \ldots \circ \sigma_1]_{\mathsf{h}} \\
&= a(t_1[\sigma_n \circ \ldots \circ \sigma_1]_{\mathsf{h}}, \ldots, t_n[\sigma_n \circ \ldots \circ \sigma_1]_{\mathsf{h}}) \\
&= a(t_1[\sigma_1]_{\mathsf{h}}[\sigma_n \circ \ldots \circ \sigma_2]_{\mathsf{h}}, t_2[\sigma_2 \circ \sigma_1]_{\mathsf{h}}[\sigma_n \circ \ldots \circ \sigma_3]_{\mathsf{h}}, \ldots, t_n[\sigma_n \circ \ldots \circ \sigma_1]_{\mathsf{h}}) && \text{theorem 2.3.11} \\
&= a(u_1[\sigma_n \circ \ldots \circ \sigma_2]_{\mathsf{h}}, u_2[\sigma_n \circ \ldots \circ \sigma_3]_{\mathsf{h}}, \ldots, u_n) && \text{by IH} \\
&= a(u_1, \ldots, u_n). && (u_i)_i \text{ are closed}
\end{aligned}$$

  In order to prove that σ is Δ-canonical, we need to show that the $(\sigma_i)_{1 \leq i \leq n}$ have disjoint domains. Let $i \in \{2, \ldots, n\}$. By induction hypothesis, σ_i is Δ-canonical for the problem $t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_{\mathsf{h}} \leq^{\tau}_{?} u_i$, therefore for any $x \in \mathsf{dom}(\sigma_i)$, x is free in $t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_{\mathsf{h}}$ and in particular, since $\sigma_1, \ldots, \sigma_{i-1}$ are closed, $\sigma_{i-1} \circ \ldots \circ \sigma_1$ is also closed and x must not be defined in it. In other words, $x \notin \mathsf{dom}(\sigma_{i-1} \circ \ldots \circ \sigma_1) = \mathsf{dom}(\sigma_1 \cup \ldots \cup \sigma_n)$. We now show that σ is Δ-canonical. Suppose by contradiction that there exists $x \in \sigma$ such that $t[\sigma - x]_{\mathsf{h}} = u$. We then have for any $i \in \{1, \ldots, n\}, t_i[\sigma - x]_{\mathsf{h}} = u_i$. Since the $(\sigma_i)_{1 \leq i \leq n}$ have disjoint domains, there exists $i \in \{1 \leq i \leq n\}$ such that $x \in \sigma_i$

$$\sigma - x = \sigma_n \circ \ldots \circ (\sigma_i - x) \circ \ldots \circ \sigma_1.$$

By induction hypothesis, $t_i[\sigma_i \circ \ldots \circ \sigma_1]_\mathsf{h}$ is closed so the only possible free variable in $t_i[(\sigma_i - x) \circ \ldots \circ \sigma_1]_\mathsf{h}$ is x and since for any $j \neq i$, $x \notin \sigma_j$ we have:

$$\begin{aligned}
u_i &= t_i[\sigma_n \circ \ldots \circ (\sigma_i - x) \circ \ldots \sigma_1]_\mathsf{h} \\
&= t_i[(\sigma_i - x) \circ \ldots \circ \sigma_1]_\mathsf{h}[\sigma_n \circ \ldots \circ \sigma_{i+1}]_\mathsf{h} && \text{theorem 2.3.11} \\
&= t_i[(\sigma_i - x) \circ \ldots \circ \sigma_1]_\mathsf{h} && (x \notin \sigma_j)_{j>i} \text{ et } \mathrm{FV}(t) \subseteq \{x\} \\
&= t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h}[\sigma_i - x]_\mathsf{h} && \text{theorem 2.3.11}
\end{aligned}$$

So σ_i is not Δ-canonical for $t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq^\tau_? u_i$, which contradicts the induction hypothesis.

- Case $\dfrac{1 \leq i \leq n \qquad \Gamma \mid \Delta \vdash f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h} \leq^\tau_? u \dashv \sigma'}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^A_? u \dashv \sigma' \circ \{f \mapsto \lambda x_1 \ldots x_n.x_i\}}$:
The fact that σ is a filter is straightforward:

$$\begin{aligned}
t[\sigma]_\mathsf{h} &= f(t_1, \ldots, t_n)[\sigma' \circ \{f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h} \\
&= f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h}[\sigma']_\mathsf{h} && \text{theorem 2.3.11} \\
&= u && \text{induction hypothesis}
\end{aligned}$$

f is trivially (t, u, σ)-essential:

$$t[\sigma - f]_\mathsf{h} = f(t_1, \ldots, t_n)[\sigma - f]_\mathsf{h} = f(t_1[\sigma - f]_\mathsf{h}, \ldots, t_n[\sigma - f]_\mathsf{h}) \neq u,$$

so we only have to show that all $x \in \sigma'$ are (t, u, σ)-essential. Let $x \in \sigma'$. By hypothesis, σ' is Δ-canonical for a term that does not have f as a free variable so $x \neq f$, therefore we have:

$$\begin{aligned}
t[\sigma - x]_\mathsf{h} &= f(t_1, \ldots, t_n)[(\sigma' - x) \circ \{f \mapsto \lambda x_1 \ldots x_n.x_i\}]_\mathsf{h} \\
&= f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h}[\sigma' - x]_\mathsf{h} && \text{theorem 2.3.11} \\
&\neq u && \text{induction hypothesis}
\end{aligned}$$

- Case $\dfrac{\begin{array}{c}\ldots \\ \Gamma, f_1 : \_, \ldots, f_m : \_ \mid \Delta \vdash f(t_1, \ldots, t_n)[\theta]_\mathsf{h} \leq^A_? F(u_1, \ldots, u_m) \dashv \sigma\end{array}}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^A_? F(u_1, \ldots, u_m) \dashv (\sigma \circ \theta) - f_i}$: the proof for this case is analogous to that of the PROJECT case, with the straightforward additional step showing that the $(f_i)_i$ are (and are the only) non-$(t, u, \sigma \circ \theta$-essential variables.

□

3.3 Completeness

Lemma 3.3.1. *Let $\sigma_1 \in \mathcal{S}_\Sigma(\emptyset, \Gamma_1)$ and $\sigma_2 \in \mathcal{S}_\Sigma(\emptyset, \Gamma_2)$. If $\sigma_2 \subseteq \sigma_1$, then*

$$\sigma_2 \circ \sigma_1 = \sigma_1 \circ \sigma_2 = \sigma_1.$$

Proof. By definition of substitution composition, we have

$$\mathsf{dom}(\sigma_2 \circ \sigma_1) = \mathsf{dom}(\sigma_1 \circ \sigma_2) = \mathsf{dom}(\sigma_2) \cup \mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma_1).$$

And for $x \in \mathsf{dom}(\sigma_1)$,

$$(\sigma_2 \circ \sigma_1)(x) = (\sigma_1.x)[\sigma_2]_\mathsf{h} = \sigma_1(x)[\sigma_2]_\mathsf{h} = \sigma_1(x)$$

where the last step equality follows from $\sigma_1(x)$ being close, and

$$(\sigma_1 \circ \sigma_2)(x) = (\sigma_2.x)[\sigma_1]_\mathsf{h} = \begin{cases} \sigma_2(x)[\sigma_1]_\mathsf{h} = \sigma_2(x) = \sigma_1(x) & \text{if } x \in \sigma_2 \text{ (because } \sigma_2 \subseteq \sigma_1) \\ x[\sigma_1]_\mathsf{h} = \sigma_1(x) & \text{otherwise.} \end{cases}$$

□

Theorem 3.3.2 (Completeness). *Let $t \in \mathcal{T}_\Sigma((\Gamma, \Delta), \tau)$, $u \in \mathcal{T}_\Sigma(\Delta, \tau)$ and $\sigma \in \mathcal{S}_\Sigma(\emptyset, \Gamma)$. If σ is a Δ-canonical filter with respect to $\{t \leq^\tau_? u\}$, then we have $\Gamma \mid \Delta \vdash t \leq^\tau_? u \dashv \sigma$*

Proof. We prove it by well-founded induction on the pair (u, t) using the lexicographic order $\prec \times_\mathcal{L} \prec$ of the structural order $\prec$ on higher-order terms. We reason by cases on the form of t.

- Case $t = \lambda x.t'$: Assuming, without loss of generality, that x is fresh enough, we have

  $$u = t[\sigma]_\mathsf{h} = (\lambda x.t')[\sigma]_\mathsf{h} = \lambda x.t'[\sigma]_\mathsf{h}.$$

  And for $y \in \sigma$, since σ is Δ-canonical we have

  $$\begin{aligned} t[\sigma]_\mathsf{h} &\neq t[\sigma - y]_\mathsf{h} \\ &= \lambda x.t'[\sigma - y]_\mathsf{h}. \end{aligned}$$

  By combining the two equations above, we get $\lambda x.t'[\sigma]_\mathsf{h} \neq \lambda x.t'[\sigma - y]_\mathsf{h}$. Which implies that $t'[\sigma]_\mathsf{h} \neq t'[\sigma - y]_\mathsf{h}$. By inverting the typing judgement for t, we have that $\Gamma, \Delta x : \tau_1 \vdash t' : \tau_2$ and $\tau = \tau_1 \to \tau_2$ for some $\tau_1, \tau_2 \in T^{\to}_\Sigma$. Therefore, σ is $(\Delta, x : \tau_1)$-canonical with respect to the problem $t' \leq^{\tau_2}_? t'[\sigma]_\mathsf{h}$. Since $u = \lambda x.t'[\sigma]_\mathsf{h}$, we have $(t', t'[\sigma]_\mathsf{h}) \prec \times_\mathcal{L} \prec (t, u)$; by induction hypothesis, we have $\Gamma \mid \Delta, x : \tau_1 \vdash t' \leq^{\tau_2}_? t'[\sigma]_\mathsf{h} \dashv \sigma$, and can therefore apply the rule Introduce in conjunction with the hypothesis that $(\lambda x.t')[\sigma]_\mathsf{h} = u$.

  $$\frac{\Gamma \mid \Delta, x : \tau_1 \vdash t' \leq^{\tau_2}_? t'[\sigma]_\mathsf{h} \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t' \leq^{\tau_1 \to \tau_2}_? u \dashv \sigma}$$

- Case $t = a(t_1, \ldots, t_n)$ with $\Sigma \cup \Delta \ni a : \tau_1 \times \ldots \times \tau_n \to A$: we have $\tau = A$ and

  $$u = t[\sigma]_\mathsf{h} = a(t_1[\sigma]_\mathsf{h}, \ldots, t_n[\sigma]_\mathsf{h}).$$

  We denote $(u_i \coloneqq t_i[\sigma]_\mathsf{h})_{1 \leq i \leq n}$ and we define substitutions $(\sigma_i)_{1 \leq i \leq n}$ by induction.

  $$\begin{aligned} \sigma_1 &= \mathsf{c}_{t_1 \leq^{\tau_1}_? u_1}(\sigma) \\ \sigma_{i+1} &= \mathsf{c}_{t_{i+1}[\sigma_i \circ \ldots \circ \sigma_1]_\mathsf{h} \leq^{\tau_{i+1}}_? u_{i+1}}(\sigma) \qquad\qquad 1 \leq i < n \end{aligned}$$

  These substitutions are well-defined: we show by induction that for all $i \in \{1, \ldots, n\}$, $\sigma_i \subseteq \sigma$ and σ is a filter for $t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq^{\tau_i}_? u_i$.

- Case 0: by hypothesis, σ is a filter for $t_1 \leq_?^{\tau_1} u_1$, and by a combination of the definition of $\mathsf{c}_{t_1 \leq_?^{\tau_1} u_1}$ and theorem 3.1.10, $\sigma_1 \subseteq \sigma$.
- Case $i+1$: By induction hypothesis,

$$\mathsf{dom}(\sigma_i \circ \ldots \circ \sigma_1) = \mathsf{dom}(\sigma_i) \cup \ldots \cup \mathsf{dom}(\sigma_1) \subseteq \mathsf{dom}(\sigma).$$

  Which allows us to have

$$\begin{aligned} t_{i+1}[\sigma_i \circ \ldots \circ \sigma_1]_\mathsf{h}[\sigma]_\mathsf{h} &= t_{i+1}[(\sigma_i \circ \ldots \circ \sigma_1) \circ \sigma]_\mathsf{h} && \text{theorem 2.3.11} \\ &= t_{i+1}[\sigma]_\mathsf{h} && \text{theorem 3.3.1} \\ &= u_{i+1}. \end{aligned}$$

  σ is therefore a filter, making σ_{i+1} well-defined ; and by $theorem$ 3.1.10 and by definition of σ_{i+1}, $\sigma_{i+1} \subseteq \sigma$.

Therefore, for all $i \in \{1, \ldots, n\}$, by definition of σ_i,

$$\sigma_i \text{ is a } \Delta\text{-canonical filter for } t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq_?^{\tau_i} u_i.$$

For all $i \in \{1, \ldots, n\}$, $u_i \prec u$ so $(u_i, t_i[\ldots]) \prec \times_{\mathcal{L}} \prec (u, t)$; by induction hypothesis, we get

$$(\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq_?^{\tau_i} u_i \dashv \sigma_i)_{1 \leq i \leq n}.$$

This allows us to apply the SIMPLIFY rule and obtain

$$\Gamma \mid \Delta \vdash a(t_1, \ldots, t_n) \leq_?^A a(u_1, \ldots, u_n) \dashv \sigma_n \circ \ldots \circ \sigma_1.$$

Since $t = a(t_1, \ldots, t_n)$ and $u = a(u_1, \ldots, u_n)$, all we need to do is to show that $\sigma = \sigma_n \circ \ldots \circ \sigma_1$. Using the soundness theorem, we have that $\sigma_n \circ \ldots \circ \sigma_1$ is Δ-canonical for the problem. But by hypothesis, σ is also Δ-canonical. By theorem 3.1.10, the two are equal.

- Case $t = f(t_1, \ldots, t_n)$ with $\Gamma \ni f : \tau_1 \times \ldots \times \tau_n \to A$: in that case, $\tau = A$. Let $a(u_1, \ldots, u_m)$ be the form of u, and $\upsilon_1 \times \ldots \times \upsilon_m \to A$ be the type of a. Since $u \in \mathcal{T}_\Sigma(\Delta, A)$ where A is a primitive type, its binder is empty and its head a is either a function symbol in Σ or a variable in Δ. Since the variable f is the head of t, it is necessarily (t, u, σ)-essential, therefore by theorem 3.1.5 there exists some σ' such that $f \notin \sigma'$ and

$$\sigma = \sigma' \circ \{f \mapsto \sigma(f)\}.$$

There exist $(x_i)_{1 \leq i \leq n} \in \mathbb{A}^n$, $r \geq 0$, $(\kappa_i)_{1 \leq i \leq r} \in T_\Sigma^{\to}$, $(v_i \in \mathcal{T}_\Sigma(\Delta', \kappa_i))_{1 \leq i \leq r}$ and $\Sigma \cup \Delta' \ni b : \kappa_1 \times \ldots \times \kappa_r \to A$ such that

$$\sigma(f) = \lambda x_1 \ldots x_n . b(v_1, \ldots, v_r),$$

where $\Delta' \coloneqq (x_1 : \tau_1, \ldots, x_n : \tau_n)$. Without loss of generality, we may assume that the $(x_i)_i$ do not appear in Γ or Δ.

We have

$$\begin{aligned} a(u_1, \dots, u_n) &= u \\ &= t[\sigma]_{\mathsf{h}} \\ &= f(t_1, \dots, t_n)[\sigma' \circ (f \mapsto \sigma(f))]_{\mathsf{h}} \\ &= f(t_1, \dots, t_n)[f \mapsto \lambda x_1, \dots, x_n.b(v_1, \dots, v_r)]_{\mathsf{h}}[\sigma']_{\mathsf{h}} \\ &= b(v_1, \dots, v_r)[\sigma', x_1 \mapsto t_1[\sigma]_{\mathsf{h}}, \dots, x_n \mapsto t_n[\sigma]_{\mathsf{h}}]_{\mathsf{h}}. \end{aligned}$$

Let's abbreviate $(\sigma', (x_i \mapsto t_i[\sigma]_{\mathsf{h}})_{1 \leq i \leq n})$ as σ''. We separate two cases based on the nature of b.

1. If $b \in \Delta'$ (i.e. there exists $1 \leq i \leq n$ such that $b = x_i$), we necessarily have $r = 0$ since f is of order 1. We get in that case

$$u = b(v_1, \dots, v_n)[\sigma'']_{\mathsf{h}} = x_i[\sigma'']_{\mathsf{h}} = t_i[\sigma]_{\mathsf{h}},$$

therefore σ' is a filter for the problem $t_i \leq^{\tau_i}_{?} u_i$. It is also Δ-canonical because if we had some $y \in \sigma'$ such that $u = t_i[\sigma' - y]_{\mathsf{h}}$, we would also have $u = t[\sigma - y]_{\mathsf{h}}$ which contradicts the fact that σ is Δ-canonical for $t \leq^{\tau}_{?} u$. We have $t_i \prec t$ so $(t_i, u) \prec \times_{\mathcal{L}} \prec (t, u)$, we can therefore apply our induction hypothesis to obtain

$$\Gamma \mid \Delta \vdash t_i \leq^{A}_{?} u \dashv \sigma'.$$

Since $t_i = f(t_1, \dots, t_n)[f \mapsto \lambda x_1 \dots x_n.x_i]_{\mathsf{h}}$, we can apply the Project rule to get

$$\Gamma \mid \Delta \vdash t \leq^{A}_{?} u \dashv \sigma' \circ (f \mapsto \lambda x_1 \dots x_n.x_i)$$

2. Otherwise, $b \in \Sigma$. We write F instead of b to emphasize on the fact that it is an operator symbol. In that case:

$$a(u_1, \dots, u_n) = F(v_1, \dots, v_n)[\sigma'']_{\mathsf{h}} = F(v_1[\sigma'']_{\mathsf{h}}, \dots, v_r[\sigma'']_{\mathsf{h}}),$$

so $a = F$, $r = m$ and $(u_i = v_i[\sigma'']_{\mathsf{h}})_{1 \leq i \leq m}$. In addition to that, since the $(v_i)_i$ only have $x_1, \dots, x_n$ as free variables, we have for all $i \in \{1, \dots, m\}$,

$$u_i = v_i[\sigma'']_{\mathsf{h}} = v_i[\sigma, x_1 \mapsto t_1[\sigma], \dots, x_n \mapsto t_n[\sigma]]_{\mathsf{h}} = v_i[x_1 \mapsto t_1[\sigma], \dots, x_n \mapsto t_n[\sigma]]_{\mathsf{h}}.$$

For each $i \in \{1, \dots, n\}$, we define a fresh variable $f_i \in \mathbb{A}$ and its associated type: $v'_i \coloneqq \tau_1 \to \dots \to \tau_n \to v_i$. Morally, we want each f_i to imitate the term v_i which has access to the free variables $x_1, \dots, x_n$ of respective types $\tau_1, \dots, \tau_n$.
We set

$$\begin{aligned} \Gamma' &\coloneqq (\Gamma - f), f_1 : v'_1, \dots, f_m : v'_m, \\ \theta &\coloneqq f \mapsto F(\lambda x_1 \dots x_n.\mathsf{N}_{\Gamma', v'_1}(f_1(x_1, \dots, x_n)), \dots, \mathsf{N}_{\Gamma', v'_n}(f_n(x_1, \dots, x_n))), \\ t' &\coloneqq t[\theta]_{\mathsf{h}}. \end{aligned}$$

We observe that $\theta \in \mathcal{S}_\Sigma((\Gamma', \Delta), (\Gamma, \Delta))$, so by theorem 2.4.4, we have $\Gamma', \Delta \vdash t' : A$ For $i \in \{1, \dots, m\}$, we define

$$\rho_i \coloneqq (f_i \mapsto \lambda x_1, \dots, x_n.v_i),$$

and

$$\rho \coloneqq \rho_1 \circ \dots \circ \rho_m$$

Let $i \in \{1, \dots, m\}$, and let $\lambda z_i^1 \dots z_i^{k_i}.v_i'$ be the long normal form of v_i where $v_i^1 \times \dots \times v_i^{k_i} \to B = v_i$ for some types $v_i^1, \dots, v_i^{k_i}, B$. We have

$$\begin{aligned}
&\mathsf{N}_{\Gamma', v_i}(f_i(t_1[\theta]_\mathsf{h}, \dots, t_n[\theta]_\mathsf{h}))[\sigma' \circ \rho]_\mathsf{h} \\
&= (\lambda z_i^1 \dots z_i^{k_i}.f_i(t_1[\theta]_\mathsf{h}, \dots, t_n[\theta]_\mathsf{h}, z_i^1, \dots, z_i^{k_i}))[\sigma' \circ \rho]_\mathsf{h} \\
&= \lambda z_i^1 \dots z_i^{k_i}.v_i'[x_1 \mapsto t_1[\rho \circ \theta]_\mathsf{h}, \dots, x_n \mapsto t_n[\rho \circ \theta]_\mathsf{h}, z_i^1 \mapsto z_i^1, \dots, z_i^{k_i} \mapsto z_i^{k_i}]_\mathsf{h}[\sigma']_\mathsf{h} \\
&= \lambda z_i^1 \dots z_i^{k_i}.v_i'[x_1 \mapsto t_1[\rho \circ \theta]_\mathsf{h}, \dots, x_n \mapsto t_n[\rho \circ \theta]_\mathsf{h}]_\mathsf{h}[\sigma']_\mathsf{h}.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\rho \circ \theta &= \rho, f \mapsto F(v_1, \dots, v_m) \\
&= \rho, f \mapsto \sigma(f).
\end{aligned}$$

So since $\mathsf{FV}(v_i') \cap \mathsf{dom}(\sigma') = \mathsf{FV}(t_i) \cap \mathsf{dom}(\rho) = \emptyset$ we get:

$$\begin{aligned}
&\mathsf{N}_{\Gamma', v_i}(f_i(t_1[\theta]_\mathsf{h}, \dots, t_n[\theta]_\mathsf{h}))[\sigma' \circ \rho]_\mathsf{h} \\
&= \lambda z_i^1 \dots z_i^{k_i}.v_i'[\sigma', x_1 \mapsto t_1[(\rho, f \mapsto \theta(f)) \circ \sigma']_\mathsf{h}, \dots, x_n \mapsto t_n[(\rho, f \mapsto \theta(f)) \circ \sigma']_\mathsf{h}]_\mathsf{h} \\
&= \lambda z_i^1 \dots z_i^{k_i}.v_i'[\sigma', x_1 \mapsto t_1[\rho \circ \sigma]_\mathsf{h}, \dots, x_n \mapsto t_n[\rho \circ \sigma]_\mathsf{h}]_\mathsf{h} \\
&= \lambda z_1 \dots z_k.v_i'[x_1 \mapsto t_1[\rho \circ \sigma]_\mathsf{h}, \dots, x_n \mapsto t_n[\rho \circ \sigma]_\mathsf{h}]_\mathsf{h} \\
&= \lambda z_i^1 \dots z_i^{k_i}.v_i'[x_1 \mapsto t_1[\sigma]_\mathsf{h}, \dots, x_n \mapsto t_n[\sigma]_\mathsf{h}]_\mathsf{h} \\
&= u_i.
\end{aligned}$$

Finally, if we denote for all i the term $\lambda z_1^1 \dots z_1^{k_i}.f_i(t_1[\theta]_\mathsf{h}, \dots, t_n[\theta]_\mathsf{h})$ by f_i', we have

$$\begin{aligned}
t'[\sigma' \circ \rho]_\mathsf{h} &= t[\theta]_\mathsf{h}[\sigma' \circ \rho]_\mathsf{h} \\
&= F(f_1', \dots, f_m')[\sigma' \circ \rho]_\mathsf{h} \\
&= F(f_1'[\sigma' \circ \rho]_\mathsf{h}, \dots, f_m'[\sigma' \circ \rho]_\mathsf{h}) \\
&= F(u_1, \dots, u_m) \\
&= u
\end{aligned}$$

Therefore, $\sigma' \circ \rho$ is a filter for the problem $t' \leq_?^A u$. It is also Δ-canonical: all the $(f_i)_i$ are essential or else they remain free in t'. And for $y \in \sigma'$, if we assume by contradiction that $t'[(\sigma' - y) \circ \rho]_\mathsf{h} = u$, we can redo the three calculations above and show that $t[\sigma - y]_\mathsf{h} = u$ which is absurd because σ is canonical for the pair

$t \leq^A_? u$.
We have the canonical pair $\sigma' \circ \rho$ for the problem $t' \leq^A_? u$. Since the left-hand side is u itself, we can use the result of the second case of this proof , and arrive immediately at

$$\Gamma' \mid \Delta \vdash t' \leq^A_? u \dashv \sigma' \circ \rho,$$

which can be rewritten as

$$(\Gamma - f), f_1 : \upsilon'_1, \ldots, f_m : \upsilon'_m \mid \Delta \vdash t[\theta]_{\mathsf{h}} \leq^A_? u \dashv \sigma' \circ \rho.$$

We can now apply the Imitate and we get

$$\Gamma \mid \Delta \vdash t \leq^A_? u \dashv ((\sigma' \circ \rho) \circ \theta) - f_1, \ldots, f_m.$$

To conclude, let us simplify this substitution:

$$\begin{aligned}((\sigma' \circ \rho) \circ \theta) - f_1, \ldots, f_m &= (\sigma' \circ (\rho \circ \theta)) - f_1, \ldots, f_m \\ &= (\sigma' \circ (\rho, f \mapsto \sigma(f))) - f_1, \ldots, f_m \\ &= (\sigma \circ \rho) - f_1, \ldots, f_m \\ &= (\sigma \circ f_1 \mapsto \rho(f_1) \circ \ldots \circ f_m \mapsto \rho(f_m)) - f_1, \ldots, f_m \\ &= \sigma\end{aligned}$$

□

3.4 Implementation

Theorems 3.2.3 and 3.3.2 show that the inference system we defined completely covers the problem of second-order unification. Here, we make the observation that this inference system describes, in fact, an algorithm.

Given a judgement $\Gamma \mid \Delta \vdash t \leq^\tau_? u \dashv \sigma$, we choose to see the five first arguments $(\Gamma, \Delta, t, \tau, u)$ as the input, and σ as the output. With this dichotomy, we see that each one of the four inference rules we defined has the property that the inputs of the premise can be trivially computed using the inputs of the result, and the output of the result can be computed trivially using the outputs of the premise. This, combined with the fact that for a given input tuple, there are only finitely many applicable inference rules, means that we have a non-deterministic algorithm that, assuming it terminates, gives us all canonical substitutions for a given problem, hence the solution to that problem. Therefore, to be convinced that we have an algorithm that computes the solution of any second-order matching problem, we only have to show that our algorithm always terminates.

Lemma 3.4.1 (Termination)**.** *Let $(\Gamma, \Delta, t, \tau, u, \sigma)$, be the data of a judgement. There exists no infinite sequence of upwards derivations using the inference rules defined in theorem 3.2.1 and starting with $\Gamma \mid \Delta \vdash t \leq^\tau_? u \dashv \sigma$.*

Proof. We prove it by induction on the pair (u, t), using the lexicographic product $\prec \times_{\mathcal{L}} \prec$ of the structural order $\prec$ on higher-order terms. We proceed by cases on the derivation of the judgement.

- Case $\dfrac{\Gamma \mid \Delta, x : A \vdash t' \leq^A_? u' \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t' \leq^{\tau' \to A}_? \lambda x.u' \dashv \sigma}$: by induction hypothesis, since $u' \prec \lambda x.u' = u$, the judgement $\Gamma \mid \Delta, x : A \vdash t \leq^A_? u' \dashv \sigma$ is derived in a finite number of steps.

- Case $\dfrac{\Sigma \cup \Delta \ni a : \tau_1 \times \ldots \tau_n \to A \qquad (\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq^{\tau_i}_? u_i \dashv \sigma_i)_{1 \leq i \leq n}}{\Gamma \mid \Delta \vdash a(t_1, \ldots, t_n) \leq^A_? a(u_1, \ldots, u_n) \dashv \sigma_n \circ \ldots \circ \sigma_1}$: for all $i \in \{1, \ldots\}$, by induction hypothesis, since $u_i \prec a(u_1, \ldots, u_n) = u$, the judgement $\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_\mathsf{h} \leq^{\tau_i}_? u_i \dashv \sigma_i$ is derived in a finite number of steps.

- Case $\dfrac{1 \leq i \leq n \qquad \Gamma \mid \Delta \vdash f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h} \leq^\tau_? u \dashv \sigma'}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^A_? u \dashv \sigma' \circ \{f \mapsto \lambda x_1 \ldots x_n.x_i\}}$: we have $f(t_1, \ldots, t_n)[f \mapsto \lambda x_1 \ldots x_n.x_i]_\mathsf{h} = t_i$. By induction hypothesis, since $t_i \prec t$, the judgement $\Gamma \mid \Delta \vdash t_i \leq^\tau_? u \dashv \sigma$ is derived in a finite number of steps.

- Case $\dfrac{\begin{array}{c}\ldots \\ \Gamma, f_1 : \_, \ldots, f_m : \_ \mid \Delta \vdash f(t_1, \ldots, t_n)[\theta]_\mathsf{h} \leq^A_? F(u_1, \ldots, u_m) \dashv \sigma\end{array}}{\Gamma, f : \tau_1 \times \ldots \times \tau_n \to A \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^A_? F(u_1, \ldots, u_m) \dashv (\sigma \circ \theta) - f_i}$: in that case, we have

  $$f(t_1, \ldots, t_n)[\theta]_\mathsf{h} = F(f_1(\ldots), \ldots, f_m(\ldots)).$$

  Since F is rigid, by inversion of the judgement

  $$\Gamma \mid \Delta \vdash F(f_1(\ldots), \ldots, f_m(\ldots)) \leq^A_? F(u_1, \ldots, u_m) \dashv \sigma,$$

  the only applicable rule is the SIMPLIFY rule, which gets us to m judgements of the form

  $$\Gamma \mid \Delta \vdash f_i(\ldots) \leq^{\cdot\cdot}_? u_i \dashv \sigma$$

  which are all derived in a finite number of steps by induction hypothesis. □

References

Dowek, Gilles (Dec. 2001). "Higher-Order Unification and Matching". In: *Handbook of Automated Reasoning* 2. DOI: 10.1016/B978-044450813-3/50018-7.

Herbrand, Jacques (1930). *Recherches sur la théorie de la démonstration*. fre. URL: http://eudml.org/doc/192791.

Hindley, J. and Jonathan Seldin (Sept. 1986). *Introduction to Combinators and Lambda-Calculus*. ISBN: 9780511809835. DOI: 10.1017/CBO9780511809835.

Huet, Gérard (Jan. 1976). "Résolution d'Équations dans des Langages d'ordre 1,2,...,ω." PhD thesis. URL: https://gallium.inria.fr/~huet/PUBLIC/Huet1976.pdf.

Huet, Gérard and Bernard Lang (1978). "Proving and applying program transformations expressed with second-order patterns". In: *Acta Informatica* 11.1. ISSN: 1432-0525. DOI: 10.1007/bf00264598. URL: http://dx.doi.org/10.1007/bf00264598.

Keller, Chantal and Thorsten Altenkirch (May 2011). “Normalization by hereditary substitutions”. In.

Krivine, J.L. (1993). *Lambda-calculus, Types and Models*. Computers and their applications. Ellis Horwood. ISBN: 9780130624079. URL: https://books.google.fr/books?id=brWEAAAAIAAJ.

Miller, Dale (1992). “Unification under a mixed prefix”. In: *Journal of Symbolic Computation* 14.4, pp. 321–358. ISSN: 0747-7171. URL: https://www.sciencedirect.com/science/article/pii/074771719290011R.

Robinson, J. A. (Jan. 1965). “A Machine-Oriented Logic Based on the Resolution Principle”. In: *J. ACM* 12.1, pp. 23–41. ISSN: 0004-5411. DOI: 10.1145/321250.321253. URL: https://doi.org/10.1145/321250.321253.