



Substitution for Non-Wellfounded Syntax with Binders Through Monoidal Categories

Ralph Matthes   

IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France

Kobe Wullaert   

Delft University of Technology, The Netherlands

Benedikt Ahrens   

Delft University of Technology, The Netherlands

University of Birmingham, United Kingdom

Abstract

We describe a generic construction of non-wellfounded syntax involving variable binding and its monadic substitution operation.

Our construction of the syntax and its substitution takes place in category theory, notably by using monoidal categories and strong functors between them. A language is specified by a multi-sorted binding signature, say Σ . First, we provide sufficient criteria for Σ to generate a language of possibly infinite terms, through ω -continuity. Second, we construct a monadic substitution operation for the language generated by Σ . A cornerstone in this construction is a mild generalization of the notion of heterogeneous substitution systems developed by Matthes and Uustalu; such a system encapsulates the necessary corecursion scheme for implementing substitution.

The results are formalized in the Coq proof assistant, through the UniMath library of univalent mathematics.

2012 ACM Subject Classification Theory of computation \rightarrow Type theory; Theory of computation \rightarrow Logic and verification

Keywords and phrases Non-wellfounded syntax, Substitution, Monoidal categories, Actegories, Tensorial strength, Proof assistant Coq, UniMath library

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.25

Related Version *Full Version*: <https://arxiv.org/abs/2308.05485> [23]

Acknowledgements We thank Henning Basold for pointing us to the work on completely iterative algebras, leading to a simpler proof of Theorem 4. We also thank Thomas Lamiaux for valuable comments on a draft of this paper. We gratefully acknowledge the work by the Coq development team in providing the Coq proof assistant and surrounding infrastructure, as well as their support in keeping UniMath compatible with Coq. Not least, we thank the anonymous FSCD reviewers for their thoughtful feedback on our submission.

1 Introduction

1.1 General Motivation for Non-Wellfounded Syntax With Binders

Non-wellfounded syntax with binders appears in its purest form in the coinductive reading of untyped λ -calculus. *Potentially* non-wellfounded λ -terms still consist of variables, λ -abstractions and applications only, but the construction process with these constructors can go on forever. Such construction processes can be described through functional programming, and the host programming language then serves as a meta-language for the description of those infinitary λ -terms. Instead of taking a programming perspective, one can also ask if a possibly circular definition of such a non-wellfounded term is well-formed, in the sense



© Ralph Matthes, Kobe Wullaert, and Benedikt Ahrens;
licensed under Creative Commons License CC-BY 4.0

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).

Editor: Jakob Rehof; Article No. 25; pp. 25:1–25:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that it uniquely determines such a structure. Naturally, uniqueness is understood up to bisimilarity, i. e., two such non-wellfounded λ -terms are considered equal if their infinite unfoldings have the same labels (indicating the applied constructor) in the same order on each level, starting at the root. The presence of variable binding presents the extra challenge of having to consider this bisimilarity modulo renaming of bound variables, i. e., α -equivalence – a challenge that is amplified by the possibility of having an infinite number of bindings in a non-wellfounded λ -term. In this paper, we either work on an abstract level that does not reveal this challenge, or we resort to a representation using nested datatypes that is a form of de Bruijn representation with well-scopedness guaranteed by the typing system (see, e. g., [6]), and therefore α -equivalence is just not needed.

There are many uses of coinductive untyped λ -terms (such as Böhm trees), and coinductive readings of term structures with binding (also with simple types, e. g., as in an automata-theoretic analysis by Melliès [24]) have a counterpart in infinitary rewriting.

1.2 A Motivational Application Scenario

We have an application scenario in mind for which the “static” part, i. e., the well-typed syntax itself, is important, even without the aforementioned “dynamics” of infinitary rewriting. It is more complicated than just λ -calculus, notably by the presence of embedded inductive types. This in particular motivates our search for *datatype-generic* constructions for a wide range of non-wellfounded simply-typed syntax.

The application scenario is as follows: We want to represent the entire search space for inhabitants in simply-typed λ -calculus (STLC) by a potentially non-wellfounded term of a suitable calculus. The inhabitation problem itself is the following question: “Given a context Γ and a type A of STLC, is there a term t of STLC such that $\Gamma \vdash t : A$?”. Taking into account the entire search space means including infinite runs that arise in a (naive) search loop. And the term of the “suitable calculus” should again have type A in context Γ but represent the search space and not only be a single inhabitant. This calculus is informally given by the following grammar:

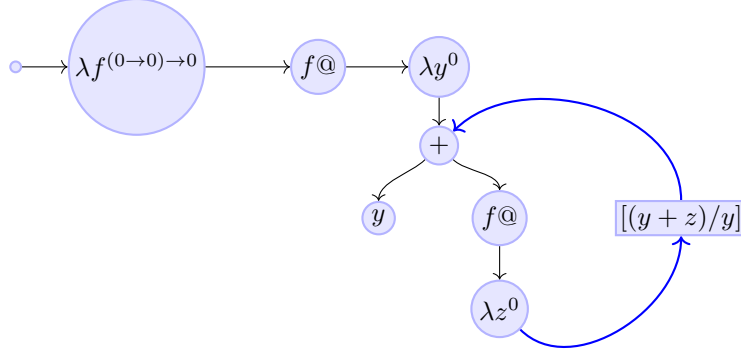
$$\begin{array}{ll} \text{(terms)} & N ::=_{co} \lambda x^A.N \mid E_1 + \cdots + E_n \\ \text{(elimination alternatives)} & E ::=_{co} x\langle N_1, \dots, N_k \rangle \end{array}$$

with one constructor for each $n, k \geq 0$, hence we have sums with any finite number of summands and tuples with any finite number of arguments. We write x in place of $x\langle \rangle$ – this captures $k = 0$. The elimination alternatives resemble the neutral terms of λ -calculus of the form $xN_1 \dots N_k$ – we are only searching for inhabitants in normal form. They have this name because they correspond to repeated implication elimination (as expressed by STLC typing) and they are summands in $E_1 + \cdots + E_n$ that indicate a finite choice between those “alternative” n summands. Search for normal forms in STLC only has finitely many options at each choice point, even though, e. g., there are infinitely many inhabitants of the type of Church numerals.

The elements of the syntactic category of terms are also called “forests”. The index *co* means that the grammar is read coinductively. There are two clauses that embed (finite) lists into the codata type. It therefore presents at least the challenges of non-wellfounded “rose trees”, i. e., finitely-branching unlabeled trees without a bound on the branching width. The scenario comes from [27, Section 3.2], and we plan to study it with our formalization. The typing rules for these expressions are given in Figure 1, with Γ ranging over finite(!) typing contexts. They are the usual implication introduction and a vectorized implication elimination (down to atomic types p), and the rule for typing alternatives (of the same

$$\frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x^A. N : A \rightarrow B}^{co} \quad \frac{(x : \vec{B} \rightarrow p) \in \Gamma \quad \forall i \leq k, \Gamma \vdash N_i : B_i}{\Gamma \vdash x \langle N_i \rangle_{i \leq k} : p}^{co} \quad \frac{\forall i \leq n, \Gamma \vdash E_i : p}{\Gamma \vdash \sum_{i \leq n} E_i : p}^{co}$$

■ **Figure 1** Coinductive typing rules for simple types in the application scenario.



■ **Figure 2** Forest representation of all inhabitants of THREE.

atomic type) – and all rules are read coinductively, indicated by the *co* mark. A well-typed such term hence *locally* conforms to intuitionistic implicative logic. For illustration, we give a well-typed forest in graphical form – the much easier example of Church numerals is found in Appendix A. Let **THREE** $\equiv ((0 \rightarrow 0) \rightarrow 0) \rightarrow 0$ for an atom 0. This is the simplest type of rank (i. e., nesting depth) 3. We define a closed forest of type **THREE** in Figure 2 [27, Example 16]. $f@$ is short for $f \langle N \rangle$ with N given by where the arrow points to. The “decontraction operation” in the back link resides on the meta-level and is specific to the summation in this example grammar: $[(y + z)/y]$ (written $[y : 0, z : 0/y : 0]$ in the cited paper) is decontraction and says that every occurrence of y has to be replaced by a sum once with y and once with z in place of the original y . This forest representation can be seen as a formal approach to the informal concept of “inhabitation machines” [8, pp. 34–38]. All the inhabitants of **THREE** can be read off this forest: omitting types, they are of the form $\lambda f.f \langle \lambda y_1.f \langle \lambda y_2.f \langle \dots \langle \lambda y_n.y_i \rangle \dots \rangle \rangle \rangle$, with $1 \leq i \leq n$. The individual inhabitants are wellfounded, but the forests representing the entire search spaces (for all simple types) are obtained coinductively in [27]. We use a generic construction of syntax such as the forests of this scenario that is based on category theory.

1.3 Context and Overview of this Paper

For *wellfounded* languages with variable binding, categorical semantics are given in [17]. The importance of monoidal structure for the modelling of substitution is emphasized there; many of the constructions are given on the level of monoidal categories, and are later instantiated to a suitable category of contexts. A very extensive overview of work on substitution for wellfounded syntax with binders, comparing [17] and subsequent work by the same and other authors, is given by Lamiaux and Ahrens [21]. A categorical semantics of *non-wellfounded* syntax with binding appeared in [22] involving the first author. That work is set concretely in endofunctor categories instead of general monoidal categories.

In that context, the present paper makes the following contributions. Firstly, the new definitions and results of the present paper lift the approach of [22] to the abstraction level of monoidal categories – to reach the same abstraction level as [17]. Secondly, we provide a full type-theoretic formalization of the results on the abstract level. Using both of these contributions, by (non-trivial) instantiation, we get a tool chain from multi-sorted binding signatures to certified monadic substitution for non-wellfounded syntax and thus the non-wellfounded counterpart to the tool chain described in [5] involving two of the present authors. Such a tool chain is absent from [22] even on the informal level (neither multi-sorted binding signatures nor unsorted binding signatures are considered). Our approach is now general enough so that the monoidal category underlying [17] and its ramifications can also be studied concerning non-wellfounded syntax and substitution for it.

In more detail, we construct non-wellfounded syntax from final coalgebras in suitable functor categories, hence based on category theory. Variable binding is modelled through the use of nested datatypes, as, e.g., in [10, 11]. The structure map of the final coalgebra is, of course, an isomorphism with its inverse providing an algebra structure. This excludes the existence of exotic terms. We benefit from that abstract view in order to construct a monadic substitution operation similar to [7], i. e., a meta-level operation that is not specific to the application scenario presented above (in contrast to the decontraction operation it features). The qualifier “monadic” implies that the generic approach includes proving the monad laws. For the case of wellfounded syntax, this is well-established in the literature (see, e. g., [7]). For the non-wellfounded case but without types, this has also been done before [20].

Further previewing our technical contributions, our approach is to generalize the notion of heterogeneous substitution systems [22] from endofunctor categories to monoidal categories. Those systems (abbreviated HSS) were meant as a tool to construct monadic substitution both for wellfounded and non-wellfounded syntax – by a common abstraction that serves as a pivotal structure between initial algebras and final coalgebras, respectively, on the input side and the substitution monad as output. We call the generalization *monoidal heterogeneous substitution systems* (MHSS). All these ingredients have been considered before for the sake of representation of wellfounded syntax [16, Section I.1.2][19, Section 5.2.1]. Section 3 is the core contribution of this paper, making the step to non-wellfounded syntax on the more abstract level of monoidal categories. The results in that section demonstrate the pivotal role of MHSS: from a final coalgebra, a MHSS is constructed, and from a MHSS, a monoid is constructed, which abstracts away from monadic substitution. Section 4 applies the results of Section 3 to the endofunctor scenario – which is hardwired into the definitions in [22].

1.4 Synopsis

The remainder of this paper is structured as follows. In Section 2 we review some prerequisites from category theory that are used later in the paper. Section 3 presents the construction of non-wellfounded syntax with substitution on the level of monoidal categories. As promised above, it generalizes both to monoidal categories and from wellfounded to non-wellfounded syntax – sloppily construable as “pushout” of these two directions. Section 4 applies the results of Section 3 to the endofunctor scenario, capturing simply-typed non-wellfounded syntax (with binding) generically. The appendix contains technical complements. However, for lack of space, Appendix B and Appendix D are only present in the full version [23].

All of the definitions and results presented in this paper (except for the motivational application scenario in Section 1.2) are formalized and computer-checked in UniMath [29], a library of univalent mathematics based on the computer proof assistant Coq [28]. Throughout this paper, definitions and results are annotated by Coq identifiers for the corresponding

definitions and results in our library. These identifiers are hyperlinks leading to an HTML version of the proof code; for instance, clicking on `monoidal` brings you to the definition of monoidal category. The formalization is not the main topic of this paper (a discussion of some formalization aspects is found in Appendix E); we use it mainly to relieve ourselves from the burden of writing out lengthy and uninteresting proofs, and the reader from the burden of reading them. Instead, we restrict ourselves to pointing the reader to interesting aspects of proofs and constructions, and aim to convey the intuition behind – and useful applications of – our work.

2 Preliminaries

We assume working knowledge of category theory and mostly only point to specific choices of notation. We write $a : \mathcal{C}$ to indicate that a is an object in category \mathcal{C} ; we write $f : a \rightarrow b$ to indicate that f is a morphism from a to b in \mathcal{C} . Following the choice adopted for the UniMath library, composition is written in “diagrammatic” order, i. e., the composite of $f : a \rightarrow b$ and $g : b \rightarrow c$ is denoted $f \cdot g : a \rightarrow c$. We will make use of a category `Set` of sets; objects of this category are called “small” sets.

2.1 Monoidal Categories and Actegories

In this section we briefly review the notions of monoidal category and of actegory.

A monoidal category is given by a six-tuple $(\mathcal{C}, \otimes, I, \lambda, \rho, \alpha)$ where \mathcal{C} is a category, $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$, $I : \mathcal{C}$, $\lambda = (\lambda_x)_{x:\mathcal{C}}$ (the *left unitor*) with $\lambda_x : I \otimes x \rightarrow x$, $\rho = (\rho_x)_{x:\mathcal{C}}$ (the *right unitor*) with $\rho_x : x \otimes I \rightarrow x$ and $\alpha = (\alpha_{x,y,z})_{x,y,z:\mathcal{C}}$ (the *associator*) with $\alpha_{x,y,z} : (x \otimes y) \otimes z \rightarrow x \otimes (y \otimes z)$. The unitors and the associator are required to be natural isomorphisms, and are furthermore subject to coherence laws called the “triangle law” and the “pentagon law”, recalled in Appendix B that is only available in the full version [23]. We will use the letter \mathcal{V} to indicate the first component of a monoidal category and, by slight abuse of language, we even call \mathcal{V} a monoidal category when the other components are left implicit. We can also just mention $(\mathcal{C}, \otimes, I)$ or $(\mathcal{V}, \otimes, I)$.

For the proper understanding of the strength notion and for the construction process of a strength in our application scenario, we use actions of monoidal categories on categories, called *actegories*. For the naming of concepts, we vaguely follow [14].

Given a monoidal category \mathcal{V} , a (left) \mathcal{V} -actegory is given by a quadruple $(\mathcal{C}, \odot, \lambda, \text{act})$ where \mathcal{C} is a category, $\odot : \mathcal{V} \times \mathcal{C} \rightarrow \mathcal{C}$ (the *action*), $\lambda = (\lambda_x)_{x:\mathcal{C}}$ (the *unitor*) with $\lambda_x : I \odot x \rightarrow x$, and $\text{act} = (\text{act}_{v,w,x})_{v,w:\mathcal{V},x:\mathcal{C}}$ (the *actor*) with $\text{act}_{v,w,x} : (v \otimes w) \odot x \rightarrow v \odot (w \odot x)$. The unitor and the actor are required to be natural isomorphisms, and are furthermore required to satisfy coherence laws analogous to the ones for monoidal categories, called also “triangle law” and “pentagon law” that are found in Appendix B (only in [23]). We consider it as important that actegories are a kind of widening of the concept of monoidal categories, in the following sense: Given a monoidal category $(\mathcal{V}, \otimes, I, \lambda, \rho, \alpha)$, $(\mathcal{V}, \otimes, \lambda, \alpha)$ is a \mathcal{V} -actegory, and it is called the actegory *with the canonical self-action* (cf. `actegory_with_canonical_self_action`).

We furthermore consider *strong monoidal functors* from monoidal category $(\mathcal{C}, \otimes, I)$ to monoidal category $(\mathcal{D}, \otimes', I')$. Such a functor is given by a triple (F, ϵ, μ) , where $F : \mathcal{C} \rightarrow \mathcal{D}$ (the *underlying functor*), $\epsilon : I' \rightarrow FI$ (preservation of unit) and $\mu = (\mu_{x,y})_{x,y:\mathcal{C}}$ (preservation of tensor) with $\mu_{x,y} : Fx \otimes' Fy \rightarrow F(x \otimes y)$. Here, we assume μ to be a natural transformation, and ϵ and μ to be isomorphisms (so as to be “strong”), as well as the three well-known (lax) laws of preservation of left and right unitality and associativity. By abuse of notation, we even call F a strong monoidal functor when the other components are left implicit.

$$\begin{array}{ccc}
 I \odot' Fx & \xrightarrow{\ell_{I,x}} & F(I \odot x) \\
 \searrow \lambda'_{Fx} & & \swarrow F\lambda_x \\
 & & Fx \\
 & & \swarrow \text{act}'_{v,w,Fx} \\
 & & (v \otimes w) \odot' Fx \\
 & & \xrightarrow{\ell_{v \otimes w, x}} & F((v \otimes w) \odot x) \\
 & & \downarrow \text{F act}'_{v,w,x} & \downarrow \\
 & & v \odot' (w \odot' Fx) & F(v \odot (w \odot x)) \\
 & \searrow 1_v \odot' \ell_{w,x} & & \swarrow \ell_{v,w \odot x} \\
 & & v \odot' F(w \odot x) &
 \end{array}$$

■ **Figure 3** Preservation of the unitor and the actor in a linear functor.

For the purposes of syntax representation, we only consider the lax form of morphisms between actegories, over a common monoidal category \mathcal{V} . Given a monoidal category $(\mathcal{V}, \otimes, I)$, a lax linear functor from actegory $(\mathcal{C}, \odot, \lambda, \text{act})$ to actegory $(\mathcal{D}, \odot', \lambda', \text{act}')$ is given by a pair (F, ℓ) , where $F : \mathcal{C} \rightarrow \mathcal{D}$ (the *underlying functor*) and $\ell = (\ell_{v,x})_{v:\mathcal{V}, x:\mathcal{C}}$ (the *lineator*) with $\ell_{v,x} : v \odot' Fx \rightarrow F(v \odot x)$. We require the lineator to be a natural transformation (not necessarily an isomorphism), and furthermore require it to satisfy two laws of preservation of the unitor and the actor, see Figure 3. Currying away the second index to ℓ and using that λ and act are isomorphisms, these laws uniquely determine ℓ_I and give a formula to calculate $\ell_{v \otimes w}$ from ℓ_v and ℓ_w . (On this level of generality of the description, this is not different from the situation for the μ component of a strong monoidal functor.)

2.2 Pointed Strength

Pointed strength is best understood through actegories; this is sketched in [16, Section I.1.2]), and our presentation here has the same main ingredients (using reindexing and a coslice category, see below). This abstract view is helpful for creating libraries of functors with pointed strength, as will be visible in Section 4. Hur uses the notion of pointed strength extensively but only spells it out concretely [19, Section 5.2.1].

Given monoidal categories \mathcal{W} and \mathcal{V} , a strong monoidal functor $F : \mathcal{W} \rightarrow \mathcal{V}$ and a \mathcal{V} -actegory $(\mathcal{C}, \odot, \lambda, \text{act})$, one can canonically construct a \mathcal{W} -actegory $(\mathcal{C}, \odot', \lambda', \text{act}')$ over the same base category – the *reindexing of the \mathcal{V} -actegory along F* . (It seems that it would suffice that F is an oplax monoidal functor instead of a strong one.) On objects, the action \odot' is constructed as $w \odot' x := Fw \odot x$. We will not spell out the details here; our formalization of this construction is given in `reindexed_actegory`.

We need reindexed actegories for one specific situation: the actegory with the canonical pointed action. We assume a monoidal category \mathcal{V} with unit I and construct the monoidal category of “monoidal-pointed objects”: the underlying category is the coslice category I/\mathcal{V} whose objects are pairs (v, pv) with $v : \mathcal{V}$ and $pv : I \rightarrow v$ (“a point for v ”), and the monoidal category can be easily constructed. Just for the record: $I^{ptd} := (I, 1_I)$ is the unit, and the tensor is defined on objects as $(v, pv) \otimes^{ptd} (w, pw) := (v \otimes w, \lambda_I^{-1} \cdot (pv \otimes pw))$.

Given a monoidal category \mathcal{V} , the actegory \mathcal{V}^{ptd} that we call the *actegory with the canonical pointed action of \mathcal{V}* is obtained by reindexing: in the definition above, we take $\mathcal{W} := I/\mathcal{V}$, \mathcal{V} as given, F the forgetful functor that forgets the points (and is strong monoidal), and as \mathcal{V} -actegory the actegory with the canonical self-action of \mathcal{V} introduced above. It follows that in \mathcal{V}^{ptd} , the monoidal-pointed objects of \mathcal{V} act on the objects of \mathcal{V} .

Given a monoidal category \mathcal{V} and an endofunctor F on \mathcal{V} , a *pointed tensorial strength* for F is a θ so that (F, θ) is a lax endomorphism of actegory \mathcal{V}^{ptd} . In other words, θ is the lineator (following the literature, we use θ for this specific use of lineators) in the situation where source and target action are \mathcal{V}^{ptd} .

$$\begin{array}{ccc}
I \otimes Fv & \xrightarrow{\theta_{I ptd, v}} & F(I \otimes v) \\
\lambda_{Fv} \searrow & & \swarrow F\lambda_v \\
& Fv & \\
(v \otimes w) \otimes Fx & \xrightarrow{\theta_{(v, pv) \otimes ptd_{(w, pw), x}}} & F((v \otimes w) \otimes x) \\
\downarrow \alpha_{v, w, Fx} & & \downarrow F\alpha_{v, w, x} \\
v \otimes (w \otimes Fx) & & F(v \otimes (w \otimes x)) \\
\downarrow 1_v \otimes \theta_{(w, pw), x} & & \downarrow \theta_{(v, pv), w \otimes x} \\
v \otimes F(w \otimes x) & &
\end{array}$$

■ **Figure 4** Preservation of the unitor and the actor for pointed tensorial strength.

In order to allow for an easy comparison with the literature, we spell out the lineator laws for pointed tensorial strength θ (besides the requirement of naturality in both arguments): the components are $\theta_{(v, pv), x} : v \otimes Fx \rightarrow F(v \otimes x)$, and the preservation rules are given in Figure 4. The differences with [19, Section 5.2.1] are all of presentational nature, most notably that we use left actegories while Hur has the monoidal-pointed objects as second parameter of his “pointed strength” st.

3 Monoid Structure on Non-Wellfounded Syntax

In this section, we construct a well-behaved substitution operation on non-wellfounded syntax. We do so on the level of monoidal categories, using the new notion of “monoidal heterogeneous substitution system”. Already mentioned in the introduction, this notion will have a pivotal role in this section: as an intermediate step between a given final coalgebra and the monoid representing substitution on that final coalgebra. The carrier of these structures is one object t of the given monoidal category \mathcal{V} , so t is the representation of all terms as a whole, thus abstracting away from context/scope and typing details. (In Section 4, \mathcal{V} will be instantiated to an endofunctor category, so that such a t will be a functor whose argument is interpreted as a typing context.)

► **Definition 1** (Monoidal heterogeneous substitution system, **mhss**). *Let \mathcal{V} be a monoidal category with unit I , tensor \otimes and right unitor ρ and H an endofunctor on \mathcal{V} with a pointed tensorial strength θ for H . We consider triples (t, η, τ) with $t : \mathcal{V}$ (the “terms”), $\eta : I \rightarrow t$ (representing the injection of variables into terms) and $\tau : Ht \rightarrow t$ (the H -algebra representing the domain-specific constructors). Hence (t, η) is a monoidal-pointed object. (t, η, τ) is a monoidal heterogeneous substitution system (MHSS) for (\mathcal{V}, H, θ) if, for all (z, e, f) with $z : \mathcal{V}$, $e : I \rightarrow z$ and $f : z \rightarrow t$, there is a unique morphism $h : z \otimes t \rightarrow t$ such that the following diagram commutes:*

$$\begin{array}{ccccc}
z \otimes I & \xrightarrow{1_z \otimes \eta} & z \otimes t & \xleftarrow{1_z \otimes \tau} & z \otimes Ht \\
\downarrow \rho_z & & \downarrow h & & \downarrow \theta_{(z, e), t} \\
& & & & H(z \otimes t) \\
& & & & \downarrow Hh \\
z & \xrightarrow{f} & t & \xleftarrow{\tau} & Ht
\end{array} \tag{1}$$

The uniquely existing morphism h is denoted as $(\llbracket f \rrbracket)_{(z, e)}$.

Notice that for the considered triples (z, e, f) , (z, e) is a monoidal-pointed object. The morphism f is just a \mathcal{V} -morphism and not a “monoidal-pointed” morphism from (z, e) to (t, η) , see Remark 2. The left unitor and the associator of the monoidal category do not enter this definition directly but through the laws governing θ (cf. Figure 4).

As seen on the right-hand side of Equation (1), the strength θ is an operation that serves to prepare the arguments that are fed into the “structurally recursive call” Hh , before applying the domain-specific constructors bundled in τ . In other words, h mostly follows a homomorphic pattern, except for the rearrangement required by variable binding – implicitly expressed in functor H – that is taken care of by θ .

This notion of MHSS is not a recursion scheme specifically for the carrier of an initial algebra (for the functor $I + H-$). The present notion of MHSS only formulates the “desideratum”, not sufficient conditions for its fulfillment. What makes MHSS suitable for dealing with coinductive syntax as well (when t, η, τ come from a final coalgebra) is the deliberate restriction of the target type of h to t . (This is already part of the notion of heterogeneous substitution system (HSS) [22], see Remark 2). There is also the restriction to τ as the H -algebra in the arrow on the bottom of the diagram, hence the limitation to a notion of substitution and not some general recursive pattern.

► Remark 2. *Monoidal* heterogeneous substitution systems “almost” generalize the notion of HSS of [22] from the specific situation where an endofunctor category $[\mathcal{C}, \mathcal{C}]$ is considered to the (unrestricted) monoidal category \mathcal{V} . The ingredients and stipulations of [22, Definition 5] are an instance of our notion of MHSS as soon as \mathcal{C} has binary coproducts – so as to be able to speak about an $(\text{Id} + H-)$ -algebra – modulo the following:

1. The order of the arguments of strength θ is inverted.
2. We consider all \mathcal{V} -morphisms f and not only morphisms f between the monoidal-pointed objects (z, e) and (t, η) , satisfying $\eta = e \cdot f$. In our diagram, that is Equation (1), f is just the \mathcal{V} -morphism. However, in the diagram in [22, Definition 5], f is written although Uf is meant, with U the forgetful functor from pointed endofunctors to endofunctors forgetting the points.¹

For the representation of substitution for non-wellfounded syntax, we will need abstract counterparts to the construction of a monad out of a HSS [22, Theorem 10] and the construction of a HSS from a final coalgebra [22, Theorem 17]. It is fair to say that these results carry over to MHSS without difficulty. We sketch the counterpart to the former construction in Section 3.1 and detail a different path to obtaining the latter in Section 3.2.

3.1 Construction of a Monoid From a MHSS

Let $(\mathcal{V}, \otimes, I, \lambda, \rho, \alpha)$ be a monoidal category. A \mathcal{V} -monoid is given by a triple (v, η, μ) where $v : \mathcal{V}$, $\eta : I \rightarrow v$ (the “unit” of the monoid) and $\mu : v \otimes v \rightarrow v$ (monoid “multiplication”), such that the left and right unit laws and the associative law hold. We recall the laws in Appendix B (only in [23]).

Let furthermore H be an endofunctor on \mathcal{V} with a pointed tensorial strength θ for H . An (H, θ) -monoid [17] is a quadruple (v, η, μ, τ) with (v, η, μ) a \mathcal{V} -monoid and $\tau : Hv \rightarrow v$ (thus (v, τ) is an H -algebra), such that the following diagram commutes:

¹ This second difference appears to be a *conceptual* simplification, and has been formalized in 2022 for endofunctor categories as a “simplified notion of HSS”, serving as a test bed for our definition of MHSS, cf. `SubstitutionSystems.SimplifiedHSS.SubstitutionSystems.v`.

$$\begin{array}{ccc}
v \otimes Hv & \xrightarrow{\theta_{(v,\eta),v}} & H(v \otimes v) & \xrightarrow{H\mu} & Hv \\
1_v \otimes \tau \downarrow & & & & \downarrow \tau \\
v \otimes v & \xrightarrow{\mu} & & & v
\end{array} \tag{2}$$

The condition expressed by the diagram is the starting point for the parameterization process that ends in the definition of MHSS.

► **Theorem 3** (Construction of a monoid from MHSS, `mhss_monoid`, `mhss_to_sigma_monoid`). *We assume the parameters \mathcal{V} , H and θ of a MHSS. Let (t, η, τ) be a MHSS, and let $\mu := (1_t)_{(t,\eta)}$ the uniquely existing morphism for $(t, \eta, 1_t)$. Then (t, η, μ, τ) is an (H, θ) -monoid.*

This generalizes [17, Proposition 3.5] from the construction of an initial (H, θ) -monoid (under extra sufficient conditions) to the construction of an (H, θ) -monoid from a suitable H -algebra (without further conditions). On the other hand, it lifts [22, Theorem 10] from the abstraction level of endofunctor categories to that of monoidal categories. For the proof of Theorem 3, we can precisely follow the organization of the proof of [22, Theorem 10]. The absence of the pointedness requirement for f in the definition of MHSS gives rise to an inessential simplification. The defining diagram of an (H, θ) -monoid is just the right-hand side of Equation (1) for the instance used to define μ . So, we have to establish the monoid laws, for which we only give an overview (cf. `mhss_monoid` for the formalization). We define $\mu^{(0)} := \eta : I \rightarrow t$ and $\mu^{(1)} := (\eta)_{I^{ptd}} : I \otimes t \rightarrow t$. The morphism λ_t satisfies its defining diagram, hence $\mu^{(1)} = \lambda_t$ by uniqueness. The right unit law of a monoid is just the left-hand side of the defining diagram of μ . The left unit law of a monoid asks for λ_v to be equal to a morphism m ; since $\lambda_t = \mu^{(1)}$, it suffices to show that m satisfies the defining diagram of $\mu^{(1)}$. Now, define $\mu^{(2)} := \mu$. The morphism $\mu^{(2)}$ is even a morphism of the monoidal-pointed objects $(t, \eta) \otimes^{ptd} (t, \eta)$ and (t, η) ; the proof uses that $\lambda_I = \rho_I$, which holds generally. Define $\mu^{(3)} := (\mu^{(2)})_{(t,\eta) \otimes^{ptd} (t,\eta)} : (t \otimes t) \otimes t \rightarrow t$. The associative law of a monoid can now be dealt with by showing that both sides of that equation satisfy the defining diagram of $\mu^{(3)}$ and are hence equal by uniqueness. The reasoning in both cases is just the monoidal generalization of the first two items of [22, p. 168].

3.2 Construction of a MHSS From a Final Coalgebra

In this section, we assume the parameters \mathcal{V} , H and θ of a MHSS. We require binary coproducts in the underlying category of \mathcal{V} (and use `inl` and `inr` without indices for the left and right injection into the coproduct). We also assume a final coalgebra (t, out) of the functor $(I + H-)$, i. e., $t : \mathcal{V}$ and $\text{out} : t \rightarrow I + Ht$. By Lambek’s theorem, `out` is an isomorphism, with inverse $\text{out}^{-1} : I + Ht \rightarrow t$ that can be written as $\text{out}^{-1} = [\eta, \tau]$ with $\eta : I \rightarrow t$ and $\tau : Ht \rightarrow t$. We also require that binary coproducts distribute over the tensor of \mathcal{V} in its second argument; this means that, for all v, w_1 and w_2 , the morphism $[1_v \otimes \text{inl}, 1_v \otimes \text{inr}]$ from $v \otimes w_1 + v \otimes w_2$ to $v \otimes (w_1 + w_2)$ has an inverse. We call that inverse δ for “distributor”, without specifying its arguments.

► **Theorem 4** (Construction of MHSS from final coalgebra, `final_coalg_to_mhss_alt`). *The triple (t, η, τ) is a MHSS for (\mathcal{V}, H, θ) .*

For clarity, we deviate from the proof of [22, Theorem 17] for HSS, which uses primitive corecursion. We instead use that out^{-1} is a completely iterative algebra (abbreviated as “cia”), which follows from `out` being a final coalgebra [25]. In particular, we will only use the definition of `cia` and not more general corecursion schemes implied by that property.

$$\begin{array}{ccc}
 z \otimes t & \xrightarrow{\text{eqm}} & (I + H-)(z \otimes t) + t \\
 \downarrow 1_z \otimes \text{out} & & \uparrow [f \cdot \text{inr}, \text{inr} \cdot \text{inl}] \\
 z \otimes (I + Ht) & & \\
 \downarrow \delta & & \\
 z \otimes I + z \otimes Ht & \xrightarrow{\rho_z + \theta_{(z,e),t}} & z + H(z \otimes t)
 \end{array}$$

■ **Figure 5** Definition of eqm as composition of four morphisms.

Given an endofunctor F on a category \mathcal{C} with binary coproducts, an F -algebra (c, α) is called a *cia* iff for every $x : \mathcal{C}$ and every morphism $e : x \rightarrow Fx + c$ (“a flat equation morphism”), there is a unique morphism $h : x \rightarrow c$ that is a “solution” of e in c in the sense that the following diagram commutes:

$$\begin{array}{ccc}
 x & \xrightarrow{e} & Fx + c \\
 h \downarrow & & \downarrow Fh + 1_c \\
 c & \xleftarrow{[\alpha, 1_c]} & Fc + c
 \end{array}$$

This generalizes the intuition when \mathcal{C} is **Set**: the elements of x are the unknowns, and e either requires a structure in F over the unknowns or directly assigns a value in c . A morphism h is a solution if, in the first case, applying h inside the structure and then assembling the structure through α yields the value of h again.

To prove Theorem 4, we apply the *cia* scheme for $F := I + H-$ and $\alpha := \text{out}^{-1}$. Given a triple (z, e, f) , the following are equivalent:

1. $h : z \otimes t \rightarrow t$ satisfies the defining diagram for $(f)_{(z,e)}$
2. h satisfies the defining diagram of a solution for the flat equation morphism $\text{eqm} : z \otimes t \rightarrow (I + H-)(z \otimes t) + t$ defined in Figure 5.

The details are found in Appendix C. In a nutshell, the two defining diagrams can be massaged so that the equivalence can be seen for each path in the diagrams individually.

On a general note, there is a whole arsenal of categorical corecursion schemes. For MHSS (and hence for the representation of substitution in the section to come), we picked the method of completely iterative algebras. Working with these tools from category theory is an alternative to intuitive “guarded” definitions and reasoning with observation depths. This alternative is suitable for formalization, for which the present paper is further evidence.

4 Non-Wellfounded Syntax for Multi-Sorted Binding Signatures

In this section, we start from the notion of multi-sorted binding signature (reviewed in Section 4.1). Exploiting the high-level results of Section 3 and thus showing their usefulness, we are going to construct the non-wellfounded syntax specified by such a signature, together with a well-behaved – monadic – substitution operation on the terms of that syntax.

Our work builds upon previous work [5] involving two of the present authors. There, categorical semantics of languages of *wellfounded* terms is developed, and a construction of the syntax generated by a multi-sorted binding signature is given. In this section, we construct *non-wellfounded* syntax based on that very same notion of multi-sorted binding signatures. Given such a signature, the existence of the generated syntax is guaranteed by ω -continuity of the associated signature functor – while, for wellfounded terms, [5] establishes ω -cocontinuity to construct the syntax. For a modular proof of ω -continuity, we decompose

the construction of the associated signature functor slightly differently. Extensionally, we arrive at the same functor, but the formalization of that proof is somewhat intricate. We therefore suggest the new construction of the signature functor as the one to work with also in the wellfounded case. This makes good sense if one wants to consider the embedding of wellfounded syntax into non-wellfounded syntax for the same multi-sorted binding signature, and it is doable since we also formalized a proof of ω -cocontinuity. But there is also an advantage on the conceptual side: the building blocks of the signature functor are all endofunctors, unlike previously [5]. A second difference with the previous work is that, for the strength construction, we systematically refer to results that reside on the abstract level of monoidal categories. (Aspects of steps 1, 2, and 4 below are also described in [5, Section 2]; we discuss them here again for the sake of being self-contained. The items 5-7 are concretized in Section 4.4.)

1. We describe simply-typed syntax with variable binding (of finitely many sorted variables in each constructor argument) as a multi-sorted binding signature, see Section 4.1.
2. Given a multi-sorted binding signature, we construct a signature functor H (deviating from [5] for technical reasons), see Section 4.2.
3. We prove ω -continuity of $(\text{Id} + H-)$ and construct the coinductive syntax as the inverse of a final coalgebra thereof, see Section 4.3.
4. We construct a “lax lineator” between actions expressing pointed tensorial strength of H , see Section 4.4 including a discussion of the case of simply-typed λ -calculus and references to the appendix for more details on the general case (only in [23]).
5. We construct a MHSS (Definition 1) for (H, θ) by applying Theorem 4.
6. We construct an (H, θ) -monoid by applying Theorem 3.
7. Finally we interpret the obtained monoid as monad (hence as monadic substitution) since, during the entire section, we are instantiating the monoidal category to the endofunctors.

4.1 Multi-Sorted Binding Signatures: Motivation and Definition

We want to construct syntax of non-wellfounded terms that feature variable binding and have a simple notion of typing. Such type systems can be specified using “multi-sorted binding signatures”; this notion was used, in particular, in [5], but appears in almost any literature about initial semantics for multi-sorted syntax. The prime example is simply-typed λ -calculus (STLC), whose extension to non-wellfounded well-typed terms is an instance of our construction. We study this example in some detail in Example 5 before reviewing multi-sorted binding signatures in Definition 6.

► **Example 5** (Non-wellfounded simply-typed lambda-calculus). We are now rephrasing [5, Example 2.2 and Example 2.10]. We assume the types of simply-typed λ -calculus to form a small set S that is closed under a binary operation $\Rightarrow: S \rightarrow S \rightarrow S$. The elements of S are called *sorts*, so as to distinguish them from the types of our ambient type theory. We model syntax over a base category \mathcal{C} (with initial object \perp , terminal object \top , and binary products and binary coproducts), not necessarily the category Set ; however, we motivate the notions for the special case where \mathcal{C} is Set . Let \mathcal{C}^S be the functor category $[S, \mathcal{C}]$ where S is viewed as a discrete category. In the case when \mathcal{C} is Set , objects of this category are simply functions $\xi: S \rightarrow \text{Set}$, and we generally use letter ξ to indicate objects of \mathcal{C}^S . They represent the typing contexts since ξs represents the totality of variables of sort s .

For the instantiation of Section 3, we take \mathcal{V} as the monoidal category of endofunctors of \mathcal{C}^S – with the tensor operation $X \otimes Y := X \cdot Y$ in diagrammatic order. In Definition 1, we are looking for one object T of \mathcal{V} (i. e., $T: [\mathcal{C}^S, \mathcal{C}^S]$) as representation of all the wellfounded and non-wellfounded terms, here of simply-typed λ -calculus. On objects, T assigns to $\xi: \mathcal{C}^S$

25:12 Substitution for Non-Wellfounded Syntax with Binders Through Monoidal Categories

and $s : S$ the object $T \xi s$ of \mathcal{C} , a representation of all the wellfounded and non-wellfounded terms that have sort s in the typing context ξ . The functor H of Section 3 prepares for the construction of T as a fixed point. Instead of only considering the “solution” T as an argument to H , we have to abstract over an arbitrary $X : \mathcal{V}$ as an object argument to H . We would like to take H as the pointwise coproduct of one summand for application and one for abstraction, for each pair (s, t) of sorts that parameterize the respective typing rules, i. e. $H \equiv \sum_{s, t : S} (\text{app}_{s, t} + \text{lam}_{s, t})$. Here the summands have to be endofunctors on $[\mathcal{C}^S, \mathcal{C}^S]$, and we only give the definition for objects (in all arguments) – where the defining equation is between objects of \mathcal{C} :

$$\begin{aligned} \text{app}_{s, t} X \xi u &\equiv \begin{cases} X \xi (s \Rightarrow t) \times X \xi s & \text{if } u = t \\ \perp & \text{else} \end{cases}, \\ \text{lam}_{s, t} X \xi u &\equiv \begin{cases} X \xi' t & \text{if } u = (s \Rightarrow t), \text{ with } \xi' s \equiv \top + \xi s \text{ and } \xi' u' = \xi u' \text{ for } u' \neq s \\ \perp & \text{else} \end{cases} \end{aligned}$$

These summands represent all well-sorted applications and λ -abstractions made from the material in the yet arbitrary object X of \mathcal{V} , while the variables are dealt with separately from H through the unit of \mathcal{V} (in our case the identity functor) in Section 3. The case distinctions on equality of sorts in this motivating example can be avoided following [5], and we will do so in the generic construction in Section 4.2.

We fix a small set S representing the sorts.

► **Definition 6** ([5, Definition 2.1], `MultiSortedSig`). A multi-sorted binding signature is given by a small set I together with an arity function $\text{ar} : I \rightarrow (S^* \times S)^* \times S$.

Here, we write A^* for the set of finite lists formed from elements of A . The intuition is as follows: for any $i : I$, $\text{ar}(i)$ is the signature of a term constructor. The second component of $\text{ar}(i)$ is the sort of the constructed term. The first component is a list of signatures of the arguments of that constructor. Each such signature is an element of $S^* \times S$, describing the sorts of all the (anonymous) variables bound by that argument, together with the sort of the argument itself. [5] makes no claim on originality of that definition, see the discussion there. It should be stressed that, while S and I can be infinite sets, each term constructor described by an $\text{ar}(i)$ only has finitely many arguments. Non-wellfounded syntax with these constructors is therefore still finitary in the sense that terms, when viewed as trees, are finitely branching.

► **Example 7** ([5, Example 2.2], `STLC_Sig`). Assume that S is closed under a binary operation \Rightarrow . We put into I the sort parameters of the typing rules of the term constructors of STLC. Thus, I is taken to be $(S \times S) + (S \times S)$. The left summand pertains to the application operation while the right summand describes λ -abstraction:

$$\text{ar}(\text{inl}\langle s, t \rangle) \equiv \langle \langle \langle _, _ \rangle, s \Rightarrow t \rangle, \langle _, s \rangle \rangle, t \rangle \quad \text{ar}(\text{inr}\langle s, t \rangle) \equiv \langle \langle \langle _, s \rangle, t \rangle, s \Rightarrow t \rangle$$

► **Example 8** (`UntypedForest_Sig`). We model the grammar of the untyped version of the forests described in Section 1.2 as a multi-sorted binding signature. Let $S_0 := \{v, t, e\}$ be a three-element set, having sorts for the three syntactic categories of term variables (v), terms (t) and elimination alternatives (e). The first sort seems unavoidable since the elimination alternative $x(N_1, \dots, N_k)$ only allows term variables in the head position, not arbitrary terms. The index set I represents the different forms of expressions that are parameterized in the elements of the syntactic categories: one for λ -abstraction, one for each index n for

summation, one for each index k for tupling, hence we set $I := (1 + \mathbb{N}) + \mathbb{N}$. We define $\text{ar}(\text{inl}(\text{inl}*)) := \langle \langle \langle \langle \mathbf{v} \rangle, \mathbf{t} \rangle, \mathbf{t} \rangle \rangle$, a simplified version of the second case for STLC: a term variable is being bound in a term, yielding a term. The other forms of expressions do not feature variable binding:

$$\text{ar}(\text{inl}(\text{inr } n)) := \langle \underbrace{\langle \langle \langle \mathbf{e} \rangle, \dots, \langle \mathbf{e} \rangle \rangle}_{n}, \mathbf{t} \rangle \quad \text{ar}(\text{inr } k) := \langle \langle \langle \mathbf{v} \rangle, \underbrace{\langle \langle \mathbf{t} \rangle, \dots, \langle \mathbf{t} \rangle \rangle}_k \rangle, \mathbf{e} \rangle$$

None of the arities have \mathbf{v} as second component, hence term variables will only come from a given context. Using the pipeline of Section 4.4, we can represent these untyped forests as an object T of $[\mathcal{C}^{\mathbf{S}_0}, \mathcal{C}^{\mathbf{S}_0}]$, analogously to Example 5. Again analogously to that example, objects ξ of $\mathcal{C}^{\mathbf{S}_0}$ represent contexts; in this untyped version they correspond just to a choice of names for all occurring variables. Then, because of the absence of \mathbf{v} as second component of our arities, we should have that $T\xi\mathbf{v}$ and $\xi\mathbf{v}$ are isomorphic for any $\xi : \mathcal{C}^{\mathbf{S}_0}$. In our representation, we are interested in those $\xi : \mathcal{C}^{\mathbf{S}_0}$ for which $\xi\mathbf{t}$ and $\xi\mathbf{e}$ are the initial object of \mathcal{C} – empty sets in the case of $\mathcal{C} = \mathbf{Set}$ – so that ξ only provides names for term variables. For those ξ , the untyped forests in the terms category and elimination alternatives category, of the grammar in Section 1.2, are represented as $T\xi\mathbf{t}$ and $T\xi\mathbf{e}$, respectively.

► **Example 9 (Forest_Sig).** The typed forests of Section 1.2 have as the set of sorts the set $\mathbf{S} \times \mathbf{S}_0$, with \mathbf{S} and \mathbf{S}_0 from the two previous examples. Moreover, since atomic types play a specific role in the typing rules in Figure 1, we have to assume a set \mathbf{atom} of atoms and an operation $\mathbf{atotype} : \mathbf{atom} \rightarrow \mathbf{S}$ (which should be thought of as an inclusion). The typing rule for tuples is additionally (as compared to the raw syntax) parameterized by a list of k elements of \mathbf{S} and one element of \mathbf{atom} . Accordingly, the index set for this multi-sorted binding signature is $I := (\mathbf{S} \times \mathbf{S} + \mathbf{atom} \times \mathbb{N}) + \mathbf{S}^* \times \mathbf{atom}$, using the set \mathbf{S}^* of finite \mathbf{S} -lists introduced above. We define $\text{ar}(\text{inl}(\text{inl}\langle s, t \rangle)) := \langle \langle \langle \langle \langle s, \mathbf{v} \rangle \rangle, \langle t, \mathbf{t} \rangle \rangle, \langle s \Rightarrow t, \mathbf{t} \rangle \rangle$, which combines the second case of Example 7 and the first case of Example 8. The other definitions are as follows:

$$\text{ar}(\text{inl}(\text{inr } \langle p, n \rangle)) := \langle \underbrace{\langle \langle \langle \mathbf{e} \rangle, \langle \mathbf{atotype } p, \mathbf{e} \rangle \rangle, \dots, \langle \langle \mathbf{e} \rangle, \langle \mathbf{atotype } p, \mathbf{e} \rangle \rangle \rangle}_{n}, \langle \mathbf{atotype } p, \mathbf{t} \rangle \rangle$$

$$\text{ar}(\text{inr } \langle [B_1, \dots, B_k], p \rangle) := \langle \langle \langle \mathbf{e} \rangle, \langle B, \mathbf{v} \rangle \rangle, \langle \langle \mathbf{e} \rangle, \langle B_1, \mathbf{t} \rangle \rangle, \dots, \langle \langle \mathbf{e} \rangle, \langle B_k, \mathbf{t} \rangle \rangle \rangle, \langle \mathbf{atotype } p, \mathbf{e} \rangle \rangle ,$$

with $B := B_1 \Rightarrow \dots \Rightarrow B_k \Rightarrow \mathbf{atotype } p$, parenthesized to the right.

A multi-sorted binding signature is just simple syntactic data (of a signature), so rather the description of a task to define the intended syntax – which in our case will include non-wellfounded terms. In the next section we discuss how to transform such a signature into a more “semantic” kind of signature: a functor H such that the semantics of the signature is given by $(\text{Id} + H-)$ -algebras. While we did this “by hand” for STLC in Example 5 (but limited the description just to object arguments), it should be clear that an automatic generation for more involved grammars such as Example 9 would be desirable.

4.2 (Modified) Signature Functor for Multi-Sorted Binding Signatures

Here, we associate to any multi-sorted binding signature (I, ar) a suitable functor H , such that the non-wellfounded syntax generated by (I, ar) is, in particular, an $(\text{Id} + H-)$ -(co)algebra. Definitions 10–13 provide the building blocks for building H modularly from basic constructions, following the combinatorial structure of (I, ar) as a family of pairs containing lists.

We now use $\mathcal{C}^{\mathbb{S}}$ generally for the functor category $[\mathbb{S}, \mathcal{C}]$, not just for the STLC example in Section 4.1. One can still think of \mathcal{C} as being **Set**, but we keep the category \mathcal{C} abstract and collect requirements on \mathcal{C} on the way – that are all fulfilled by **Set**. As mentioned in Section 4.1, we instantiate \mathcal{V} of Section 3 with the endofunctors on $\mathcal{C}^{\mathbb{S}}$ and now have to determine the endofunctor H on $[\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$.

In order to do this, we assume that \mathcal{C} has a terminal object \top , binary products, and set-indexed coproducts (including initial object \perp and binary coproducts).

► **Definition 10** ([5, Definition 2.3], `sorted_option_functor`). *Let s be a sort. The sorted option functor $\text{option}_s : \mathcal{C}^{\mathbb{S}} \rightarrow \mathcal{C}^{\mathbb{S}}$ is defined (on objects) as $\text{option}_s \xi t \equiv \coprod_{(s=t)} \top + \xi t$.*

In this definition, we form a coproduct in \mathcal{C} of \top over the type of proofs that $s = t$; i.e., we form a subsingleton. We thus avoid the use of case distinction: $\text{option}_s \xi$ is an equivalent replacement for ξ' in the definition of `lam` above [5, Remark 2.4, Remark 2.8].

► **Definition 11** ([5, Definition 2.5], `option_list`). *Given a non-empty list of sorts $\ell \equiv [s_1, \dots, s_n]$, $\text{option}^* \ell : \mathcal{C}^{\mathbb{S}} \rightarrow \mathcal{C}^{\mathbb{S}}$ is defined as $\text{option}^* \ell \equiv \text{option}_{s_1} \cdot (\text{option}_{s_2} \cdot \dots)$. For an empty list, it is $\text{option}^* [] \equiv \text{Id}$.*

► **Definition 12** ([5, Definition 2.6], `projSortToC`). *For any $s : \mathbb{S}$ the projection functor $\text{pr}_s : \mathcal{C}^{\mathbb{S}} \rightarrow \mathcal{C}$ is defined (on objects) as: $\text{pr}_s \xi \equiv \xi s$.*

► **Definition 13** ([5, Definition 2.7], `hat_functor`). *For any $s : \mathbb{S}$ we have a left adjoint to pr_s , written $\hat{s} : \mathcal{C} \rightarrow \mathcal{C}^{\mathbb{S}}$, defined on objects as $\hat{s} c t \equiv \coprod_{(s=t)} c$.*

Here, we essentially define $\hat{s} c s = c$ and $\hat{s} c t = \perp$ otherwise. As above, we avoid the case analysis for `apps,t X ξ u` in our STLC example in Section 4.1, hence do not need the matching of the constructor’s target type.

We now have all the basic building blocks to associate, to a given multi-sorted binding signature (I, ar) , a signature functor $H : [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}] \rightarrow [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$. We turn to the construction of the corresponding building blocks for H , involving a formal argument $X : [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$ to H . Intuitively, X is the unknown functor that, after applying our results of Section 3, will be set to the functor representing the coinductive sorted syntax. For the sake of motivating the modification of the definition of H compared to [5], we will now work top-down and use the letter G with upper indices (instead of F with upper indices, used there).

The final step of the construction of H is unchanged from [5]. Assume that we already have a functor $G^{(\vec{a}, t)} : [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}] \rightarrow [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$ for all $(\vec{a}, t) : (\mathbb{S}^* \times \mathbb{S})^* \times \mathbb{S}$. Then, for the multi-sorted binding signature (I, ar) , the associated signature functor $H : [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}] \rightarrow [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$ is given by the (pointwise) coproduct $HX \equiv \coprod_i :_I G^{\text{ar}(i)} X$ (`MultiSortedSigToFunctor'`).

The penultimate step constructs $G^{(\vec{a}, t)}$ (`hat_exp_functor_list'_optimized`). It assumes given functors $G^{(a, t)} : [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}] \rightarrow [\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$ for all $(a, t) : (\mathbb{S}^* \times \mathbb{S}) \times \mathbb{S}$. Given a non-empty list $\vec{a} \equiv [a_1, \dots, a_n]$, $G^{(\vec{a}, t)}$ is defined (on objects) as the iterated pointwise binary product $G^{(a, t)} X \equiv G^{(a_1, t)} X \times (G^{(a_2, t)} X \times \dots)$. The corner case $G^{([], t)}$ is given (maybe peculiarly) by $G^{([], t)} X \equiv \top_{[\mathcal{C}^{\mathbb{S}}, \mathcal{C}]}$, regardless of X , with $\top_{\mathcal{D}}$ the terminal object of \mathcal{D} . (The terminal object in functor categories is given by constantly the terminal object of the target category.) More precisely, $G^{([], t)}$ is the composition (in diagrammatic order) of first $\top_{[[\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}], [\mathcal{C}^{\mathbb{S}}, \mathcal{C}]}$ and second post-composition with \hat{t} (which is a functor from $[\mathcal{C}^{\mathbb{S}}, \mathcal{C}]$ to $[\mathcal{C}^{\mathbb{S}}, \mathcal{C}^{\mathbb{S}}]$). This view will be exploited for the construction of a pointed tensorial strength for $G^{([], t)}$.

It remains to construct $G^{(a, t)}$ (`hat_exp_functor_list'_piece`). It is a refined version of F^a in [5] since it takes into account the “target” sort t . In fact $G^{(a, t)}$, corresponds to $F^a \cdot \hat{t}$ in terms of that paper, and so the difference between our $G^{(\vec{a}, t)}$ and $F^{(\vec{a}, t)}$ in that

paper is whether the composition with \hat{t} is on each component of the product (our solution) or only on the product itself. That previous solution looks less convoluted but requires the consideration of non-endofunctors.

Let $a \equiv (\ell, s)$ with $\ell : \mathbf{S}^*$. The object part of functor $G^{(a,t)}$ is defined as $G^{(a,t)}X := \text{option}^* \ell \cdot X \cdot \text{pr}_s \cdot \hat{t}$. More precisely, $G^{(a,t)}$ is the composition (in diagrammatic order) of first the precomposition with $\text{option}^* \ell$ and second the postcomposition with $\text{pr}_s \cdot \hat{t}$. This view will be important to establish ω -continuity of $G^{(a,t)}$.

The instance for STLC can be compared to our introductory example in Section 4.1 – assuming decidability of the set of sorts, they coincide mathematically.

4.3 Existence of Final Coalgebra for Functor $\text{Id} + H -$

Given a multi-sorted binding signature, we want to apply Theorem 4 with \mathcal{V} the endofunctors on $\mathcal{C}^{\mathbf{S}}$ and the signature functor H defined in the previous section. Its first requirement is a final coalgebra of the functor $(\text{Id} + H -)$, with Id the identity functor on $\mathcal{C}^{\mathbf{S}}$ (which is the unit of \mathcal{V}). We get a final coalgebra through the dual of Adámek’s theorem on the existence of initial algebras for ω -cocontinuous functors on ω -cocomplete categories with initial object. We now require that \mathcal{C} is ω -complete (i. e., \mathcal{C} has limits of shape $0 \leftarrow 1 \leftarrow 2 \leftarrow \dots$), whence \mathcal{V} is also ω -complete. (We had already generally required a terminal object \top for \mathcal{C} , which gives one for \mathcal{V} .) We argue first that H is ω -continuous, then that $(\text{Id} + H -)$ is ω -continuous.

We analyze the building blocks of H . $G^{(a,t)}$ is defined as the composition of two functors, so it is ω -continuous if both functors are. We prove that postcomposition with $\text{pr}_s \cdot \hat{t}$ is ω -continuous (`post_comp_with_pr_and_hat_is_omega_cont`). For this, we require of \mathcal{C} that ω -limits distribute over sub-singleton coproducts. This means that the canonical morphism from the coproduct of the respective limits to the limit of the coproducts is an isomorphism. Precomposition with any fixed functor is ω -continuous, hence in particular for option_ℓ^* .

We move to ω -continuity of $G^{(\vec{a},t)}$, which we prove by induction on the length of \vec{a} . For an empty \vec{a} , this is a constant functor; and pointwise binary products preserve ω -continuity.

Moreover, we require of \mathcal{C} that ω -limits distribute over I -coproducts, a property that then also holds of \mathcal{V} . Then, ω -continuity of the $G^{\text{ar}(i)}$ carries over to their coproduct H .

As a final step, we need to show ω -continuity of $(\text{Id} + H -)$ (`is_omega_cont_Id_H`). In order to avoid still other hypotheses about distribution of ω -limits over certain colimits, we use that binary coproducts are (isomorphic) to `bool`-indexed coproducts. So, the final assumption on \mathcal{C} is that ω -limits distribute also over `bool`-coproducts (besides over sub-singleton coproducts and I -coproducts for the index set I of the multi-sorted binding signature).

For the most interesting case $\mathcal{C} = \mathbf{Set}$, all these requirements are met.

4.4 Putting Everything Together

We can now apply the general results of Section 3 for the construction of coinductive syntax with monadic substitution, as specified by a multi-sorted binding signature. We fix a set \mathbf{S} of sorts and a multi-sorted binding signature (I, ar) over \mathbf{S} . This means in particular that we take as \mathcal{V} the endofunctors on $\mathcal{C}^{\mathbf{S}}$ when instantiating the results of Section 3. We denote by $H : [\mathcal{C}^{\mathbf{S}}, \mathcal{C}^{\mathbf{S}}] \rightarrow [\mathcal{C}^{\mathbf{S}}, \mathcal{C}^{\mathbf{S}}]$ the signature functor associated to (I, ar) according to the construction in Section 4.2.

We aim to apply Theorem 4 to the final $(\text{Id} + H -)$ -coalgebra obtained in Section 4.3. To this end, the parameter θ of a MHSS has to be specified, i. e., we have to construct a suitable pointed tensorial strength θ for the signature functor H . To recall, θ instructs via the notion

of (H, θ) -monoid (Equation (2)) how the monoid multiplication μ acts on the constructors (embodied in the H -algebra τ). But in our situation, μ is the monad multiplication expressing substitution – it is well-known that \mathcal{V} -monoids in this case are nothing but monads on $\mathcal{C}^{\mathcal{S}}$ (cf. `monoid_to_monad_CAT`). In other words, we have to define θ so that it describes correctly the recursive behaviour of substitution (as expressed by μ).

► **Example 14** (Strength for STLC). A suitable θ for the “hand-written” signature functor H in Example 5 contains, in particular, operations from $Y \cdot (\text{app}_{s,t} X)$ to $\text{app}_{s,t}(Y \cdot X)$ and from $Y \cdot (\text{lam}_{s,t} X)$ to $\text{lam}_{s,t}(Y \cdot X)$ for any $s, t : \mathcal{S}$, endofunctor Y on $\mathcal{C}^{\mathcal{S}}$ that has a point $\eta : \text{Id} \rightarrow Y$, and endofunctor X on $\mathcal{C}^{\mathcal{S}}$. The instance relevant in Equation (2) is then with X and Y the representation T of STLC, and with η the inclusion of variables in terms T . The operations for *application* do essentially nothing since μ should just descend into the subtrees. For *abstraction*, given a $\xi : \mathcal{C}^{\mathcal{S}}$, we need to specify a function of type $T\xi + 1_s \rightarrow T(\xi + 1_s)$, where $1_s : \mathcal{C}^{\mathcal{S}}$ is defined as $1_s(t) := (t = s)$; that is, $X + 1_s$ is X extended by an element of sort s . In other words, we need to lift $T\xi : \mathcal{C}^{\mathcal{S}}$ extended by an element \star of sort s to $T\xi'$, with ξ' the extension of ξ by an element \star of sort s (as in Example 5); this will need η for \star in the input. In short, for abstraction, the strength specifies essentially the famous “lift” operation on a substitution function to avoid capture when descending under a binder.

The construction of a suitable θ for the variant of the generic H considered in [5] could have been adapted to our H , but we have preferred to give a construction that, although working on the level of endofunctors, is formed from building blocks that reside on the general level of monoidal categories. To structure this construction, we propose the notion of *relative lax commutator* that generalizes the notion of “pointed distributive law” in [4, Definition 10]. For lack of space, the whole (technical) construction is explained in Appendix D, but only in [23]. We thus take as θ the pointed tensorial strength for H described in the appendix. For STLC (Example 7), the strength can be exploited on the abstract level with base category \mathcal{C} (cf. `thetaSTLC` in the formalization). For forests (Example 8 and Example 9), we only exploited the situation with \mathcal{C} set to **Set** (cf. `thetaUntypedForest` and `thetaForest` in the formalization).

Theorem 4 provides us with a MHSS that serves as input to Theorem 3, hence we get an (H, θ) -monoid. This monoid is, in particular, a monad, our (certified) substitution monad for the non-wellfounded syntax described by the given multi-sorted binding signature. For STLC, we have formalized this on the abstract level and constructed the finite Church numerals as well as the infinite Church numeral (cf. `SubstitutionSystems.STLC_actegorical_abstractcat.v`).

To be more concrete, we can instantiate the base category \mathcal{C} to **Set** that satisfies all requirements on \mathcal{C} we made during the construction process and get a set of well-sorted non-wellfounded terms for any sort, given a supply of sets of variables for any sort, together with a substitution operation that respects sorts and satisfies the monad laws. We replayed the construction of the finite Church numerals in STLC in this concrete setting (cf. `SubstitutionSystems.STLC_actegorical.v`). For forests, we only considered the **Set** case and have instantiated the general constructions (cf. `SubstitutionSystems.UntypedForests.v` and `SubstitutionSystems.Forests.v`).

Although this not the topic of this paper, we mention that we also have adapted the results and the formalization of wellfounded syntax to the present setting: This includes the construction of a MHSS from an initial $(I + H-)$ -algebra under the proviso it has been obtained through a Mendler-style construction based on ω -cocontinuity of H (`initial_alg_to_mhss`). We further established that this MHSS gives rise to an initial (H, θ) -monoid `SigmaMonoidFromInitialAlgebraInitial`, for the given strength θ . These results are on

the level of a monoidal category, as in our Section 3. The signature functor H constructed in Section 4.2 is even ω -cocontinuous (based on the proof of the same property for the variant considered in [5]), under conditions on \mathcal{C} that are fulfilled for **Set**. Thus, $(\text{Id} + H-)$ is ω -bicontinuous, and we get a morphism of (H, θ) -monoids from the inductive to the coinductive syntax (`ind_into_coind` for the case **Set**). The above-mentioned four Coq vernacular example files in our UniMath library illustrate that, thanks to that actegorical development, the use of the formalized wellfounded and the formalized non-wellfounded syntax for those multi-sorted binding signatures can be done in parallel. For example, the finite Church numerals in STLC are developed independently of the choice for one of these two options. This conforms to the intuition that every single wellfounded term belonging to the non-wellfounded syntax already belongs to the wellfounded syntax, even though the categorical development of these structures is very different.

5 Related Work and Conclusions

We have cited throughout the paper the work we rely on or which initiated a line of thought. Here, we give additional information on other related work (that may have been also cited already in the main text). [20] also have codatatypes and define datatype-generic substitution corecursively, and they even calculate infinitary normal forms for their example of untyped λ -calculus. However, they do not consider typed systems, and the results are not presented on the abstraction level of monoidal categories. Instead, they use a concrete “nominal” presentation of syntax with binders. [6] also have codatatypes and even datatype-generic programming not only of substitution, but the work is not based on category theory (and so the approach is rather axiomatic than definitional). That work is implemented in the Agda system. [30] considers different categorical models of simply-typed wellfounded syntax. In its Chapter 5, the monoidal category corresponding to the framework of [17] is laid out in detail for simple types, and its Chapter 7 compares it with the monoidal category of endofunctors over a slice category. The latter is close to the concrete instance we are studying in Section 4, but we deal with non-wellfounded syntax. All in all, [30] has a lot on the strength construction with actegories, including for the typed case, and this for more than one concrete categorical representation, but non-wellfounded syntax is not considered. [12] have an approach to codatatypes that is definitionally based on category theory; but it is strongly tied to set theory through infinite cardinal numbers that appear in the definition of the class of “bounded natural functors” they consider. This allows them to implement the approach in the Isabelle system (based on a very small kernel). Popescu [26] compares different corecursors for syntax with variable binding in nominal style; it is partially formalized in Isabelle/HOL. [15] also translate multi-sorted binding signatures into signatures with strength. Their notion of syntax includes “meta-variables”, but they stay within the wellfounded terms and heavily use inductive families as provided by the Agda system. [13] comes with a UniMath/Coq formalization of the whole chain even from *skew* monoidal categories to an initial (H, θ) -monoid, hence for wellfounded syntax. Beware that swapping of the arguments of the tensor is not a harmless operation for skew monoidal categories, so our present definition of MHSS does not fit as a pivotal element in their development. [18] rework the approach of [17], still using pointed strength and (H, θ) -monoids. They also deal with simple types but do not consider non-wellfounded syntax.

We have presented, through the notion of monoidal heterogeneous substitution system, a tool which provides a monadic substitution operation also for non-wellfounded syntax, and this for the first time on the abstraction level of monoidal categories.

Our definitions and results unify the construction of both wellfounded and non-wellfounded syntax with substitution.

We also instantiated monoidal heterogeneous substitution systems to endofunctor categories and adapted the full chain from multi-sorted binding signatures to substitution for non-wellfounded syntax. For the sake of this instantiation, we provide modular results to prove ω -cocontinuity of signature functors and hence obtain both ω -cocontinuity and ω -continuity for the signature functors we generate from multi-sorted binding signatures.

All the results of this paper have been rigorously formalized with UniMath/Coq. For the specific category of sets (types of homotopy level 2 according to univalent foundations) as base category, the hypotheses of the construction of non-wellfounded syntax can be proved. Hence, for this base category, we have a “concrete” formalization of the tool chain, which provides in particular a formal construction in univalent foundations of non-wellfounded syntax with binding, as instructed by a multi-sorted binding signature, and its monadic substitution operation.

A question we have left open is that of equations on non-wellfounded terms, for instance, β -equivalence. We anticipate that some definitions could carry over from the wellfounded setting, like the definition of equations and reductions given in [2, 3]. The construction of suitable terminal coalgebras, however, seems to require some work.

References

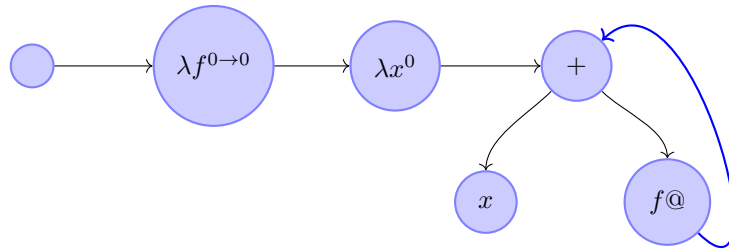
- 1 Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-wellfounded trees in homotopy type theory. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, volume 38 of *LIPICs*, pages 17–30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.TLCA.2015.17.
- 2 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPICs*, pages 6:1–6:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.FSCD.2019.6.
- 3 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Reduction monads and their signatures. *Proc. ACM Program. Lang.*, 4(POPL):31:1–31:29, 2020. doi:10.1145/3371099.
- 4 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From signatures to monads in UniMath. *J. Autom. Reason.*, 63(2):285–318, 2019. doi:10.1007/s10817-018-9474-4.
- 5 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. Implementing a category-theoretic framework for typed abstract syntax. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*, pages 307–323. ACM, 2022. doi:10.1145/3497775.3503678.
- 6 Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *J. Funct. Program.*, 31:e22, 2021. doi:10.1017/S0956796820000076.
- 7 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.
- 8 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013. URL: <http://www.cambridge.org/de/academic/subjects/mathematics/logic-categories-and-sets/lambda-calculus-types>.

- 9 Henning Basold. *Mixed Inductive-Coinductive Reasoning—Types, Programs and Logic*. PhD thesis, Radboud University, Nijmegen, The Netherlands, 2018. URL: <https://repository.ubn.ru.nl/handle/2066/190323>.
- 10 Richard Bird and Lambert Meertens. Nested Datatypes. In Johan Jeuring, editor, *Mathematics of Program Construction, MPC'98, Proceedings*, volume 1422 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 1998.
- 11 Richard S. Bird and Ross Paterson. De Bruijn Notation as a Nested Datatype. *J. Funct. Program.*, 9(1):77–91, 1999. URL: <http://journals.cambridge.org/action/displayAbstract?aid=44239>, doi:10.1017/S0956796899003366.
- 12 Jasmin Christian Blanchette, Lorenzo Gheri, Andrei Popescu, and Dmitriy Traytel. Bindings as bounded natural functors. *Proc. ACM Program. Lang.*, 3(POPL):22:1–22:34, 2019. doi:10.1145/3290335.
- 13 Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 273–286. ACM, 2020. doi:10.1145/3373718.3394738.
- 14 Matteo Capucci and Bruno Gavranović. Actegories for the working amthematician, 2022. doi:10.48550/arXiv.2203.16351.
- 15 Marcelo Fiore and Dmitriy Szamozvancev. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. doi:10.1145/3498715.
- 16 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 57–68. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.38.
- 17 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782615.
- 18 André Hirschowitz, Tom Hirschowitz, Ambroise Lafont, and Marco Maggesi. Variable binding and substitution for (nameless) dummies. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 389–408. Springer, 2022. doi:10.1007/978-3-030-99253-8_20.
- 19 Chung-Kil Hur. *Categorical equational systems : algebraic models and equational reasoning*. PhD thesis, University of Cambridge, UK, 2010. URL: <http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.608664>.
- 20 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:20)2013.
- 21 Thomas Lamiaux and Benedikt Ahrens. An introduction to different approaches to initial semantics, 2024. arXiv:2401.09366.
- 22 Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theoretical Computer Science*, 327(1-2):155–174, 2004. doi:10.1016/j.tcs.2004.07.025.
- 23 Ralph Matthes, Kobe Wullaert, and Benedikt Ahrens. Substitution for non-wellfounded syntax with binders through monoidal categories. *CoRR*, abs/2308.05485, 2023. doi:10.48550/arXiv.2308.05485, minimally version 3.
- 24 Paul-André Melliès. Higher-order parity automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005077.

- 25 Stefan Milius. Completely iterative algebras and completely iterative monads. *Inf. Comput.*, 196(1):1–41, 2005. doi:10.1016/j.ic.2004.05.003.
- 26 Andrei Popescu. Nominal recursors as epi-recursors. *Proc. ACM Program. Lang.*, 8(POPL):425–456, 2024. doi:10.1145/3632857.
- 27 José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search through typed lambda-calculi. *Ann. Pure Appl. Log.*, 172(10):103026, 2021. doi:10.1016/j.apal.2021.103026.
- 28 The Coq Development Team. The Coq proof assistant, version 8.17, 2023. URL: <https://zenodo.org/record/8161141>.
- 29 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at <http://unimath.github.io/UniMath/>, 2021.
- 30 Julianna Zsidó. *Typed Abstract Syntax*. PhD thesis, University of Nice Sophia Antipolis, 2010. URL: <http://tel.archives-ouvertes.fr/tel-00535944/>.

A An Easy Example for the Application Scenario

To ease the understanding of the example given in Section 1, we also show the one of Church numerals.



■ **Figure 6** Forest representation of all Church numerals, including infinity.

Let 0 be a base type. We define a closed forest of type $(0 \rightarrow 0) \rightarrow 0 \rightarrow 0$ in Figure 6 [27, Example 5]. $f@$ is short for $f\langle N \rangle$ with N given by where the arrow points to. The back link (in blue and thick) forms a cycle that does not go through a λ -abstraction. Hence, we get a rational tree (with only a finite number of non-isomorphic subtrees). This is a representation of all Church numerals, including infinity, and they are all the “solutions” (including the non-wellfounded ones) for the search for inhabitants in long normal form of the type $(0 \rightarrow 0) \rightarrow 0 \rightarrow 0$. Thus, by infinite unfolding of the just binary choice, an infinite number of finite solutions and even one infinite solution are obtained. The latter is the only infinite Church numeral, obtained by looping with f . In naive proof search, this is at least a potential outcome, and one may want to analyze this phenomenon.

B Recalling Some Notions of the Category-theoretical Background

For reasons of space limitations, this appendix is only available in [23].

C Proof of Theorem 4

This short appendix completes the proof of Theorem 4 in Section 3.2. It also discusses why the result on final coalgebras is in some sense easier than its counterpart for initial algebras.

$$\begin{array}{ccccc}
 z \otimes t & \xleftarrow{1_z \otimes \text{out}^{-1}} & z \otimes (I + Ht) & \xleftarrow{[1_z \otimes \text{inl}, 1_z \otimes \text{inr}]} & z \otimes I + z \otimes Ht \\
 \downarrow h & & & & \downarrow \rho_z + \theta_{(z,e),t} \\
 & & & & z + H(z \otimes t) \\
 & & & & \downarrow [f \cdot \text{inr}, \text{inr} \cdot \text{inl}] \\
 & & & & (I + H(z \otimes t)) + t \\
 & & & & \downarrow (I+H-)h + 1_t \\
 t & \xleftarrow{[\text{out}^{-1}, 1_t]} & & & (I + Ht) + t
 \end{array}$$

■ **Figure 7** Diagram characterizing a solution h for eqm.

$$\begin{array}{ccc}
 z \otimes t & \xleftarrow{[1_z \otimes \eta, 1_z \otimes \tau]} & z \otimes I + z \otimes Ht \\
 \downarrow h & & \downarrow \rho_z + \theta_{(z,e),t} \\
 & & z + H(z \otimes t) \\
 & & \downarrow 1_z + Hh \\
 t & \xleftarrow{[f, \tau]} & z + Ht
 \end{array}$$

■ **Figure 8** Diagram characterizing morphism h that should be $(f)_{(z,e)}$.

The diagram describing a solution of eqm is given in Figure 7. Of course, the diagram governing $(f)_{(z,e)}$ can be brought into a single equation over coproducts, as seen in Figure 8. The chain of morphisms from $z \otimes I + z \otimes Ht$ to t in both diagrams – Figure 7 and Figure 8 – is identical on the path to the left, as well as on the path to the right.

We remark that Theorem 4 (and also its proof) is slicker than the case of wellfounded syntax studied in [5] (however, concretely for endofunctor categories) where extra requirements beyond being an initial algebra come into play so as to guarantee the applicability of a categorical Mendler-style recursion scheme. This difference can be motivated as follows: substitution for functor T is represented by a monadic multiplication operation of type $T \cdot T \rightarrow T$ (with $T \cdot T$ self-composition of T). In the non-wellfounded case, this has the support T of the final coalgebra as target, which is suitable for using finality. However, in the wellfounded case, the source type is $T \cdot T$ and not just T that would be the basis for using initiality.

D Pointed Tensorial Strength for the Signature Functor

For reasons of space limitations, this appendix is only available in [23].

E On the Formalization

Most of the definitions and results presented in this paper are formalized and computer-checked in UniMath [29], a library of univalent mathematics based on the computer proof assistant Coq [28]. An exception is the application scenario in Section 1; its formalization is ongoing work. For this application, we can only offer the instantiation of the general constructions of this paper but not yet the inhabitation analysis alluded to in Figure 2. Our HTML documentation is derived from commit 7432fee of the UniMath library. Proof-checking and creation of the HTML documentation can easily be reproduced at home by following the UniMath compilation instructions – do try this at home!

Concerning coinductive definitions, `Coq` features a built-in mechanism for specifying coinductive types (via the keyword `CoInductive`) and for defining functions by corecursion. However, the `UniMath` library departs from standard use of `Coq` in that such declarations of coinductive datatypes are not part of the language used in `UniMath`. Furthermore, definitions by corecursion in `Coq` face numerous issues with guardedness, in particular with so-called “mixed inductive-coinductive” declarations [9] – declarations where the coinductive type makes use of a parameterized inductive type whose parameter is built with the coinductive type. The coinductive calculus of our application scenario (see Section 1), with its lists of alternatives and arguments, falls into that class. In the formalization of the contents of this paper, we therefore construct coinductive datatypes from other type constructors, rather than postulating (a class of) coinductive datatypes using meta-theoretic devices. Our approach is thus comparable to the one employed for working with coinductive datatypes in the Isabelle system [12] and of the construction of indexed M-types in univalent foundations [1]; in all of these cases, a major goal is to keep the “trusted code base” small.

We now discuss some design choices we made in the formalization. When formalizing mathematics in a formal system, some design choices need to be made that are not of mathematical significance: different choices lead to (trivially) equivalent mathematical concepts. Nevertheless, making the right choices can be crucial for the maintainability and usability of the formal library. An example of such a choice is the following. In Section 2.1, we said that a monoidal category is given by a six-tuple, with the tensor component a bifunctor. However, for the sake of our formalization, we have chosen a different but equivalent format to present the tensor operation that we are calling “whiskered”. Here, the object mapping of \otimes is replaced by its curried version, and the morphism mapping is replaced by two families of endofunctors on \mathcal{C} that represent the morphism mappings with one of the arguments fixed to the identity morphism – thus the “whiskerings” of that bifunctor. The whiskered definition avoids functors on cartesian products of categories. Such functors do not behave well in practice: the inference of the implicit object arguments – which are pairs of objects – to the functorial action on morphisms often fails, and thus these arguments need to be given explicitly. This would make the formalization cumbersome – which is why we adopted the whiskered format for our work and hence do not suffer from those problems. A third alternative to the traditional definition and the whiskered definition would be a currying of the tensor, to be a functor into a functor category. However, this definition would not provide a clean separation between data and properties – another prerequisite for a library that scales well, in our experience.