

Modular Abstract Syntax Trees (MAST): Substitution Tensors with Second-class Sorts

Marcelo P. Fiore^{a,1} Ohad Kammar^{b,2} Georg Moser^{c,3} Sam Staton^{d,4}

^a *Department of Computer Science and Technology
University of Cambridge
England*

^b *Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
Scotland*

^c *Theoretical Computer Science
Department of Computer Science
University of Innsbruck
Austria*

^d *Department of Computer Science
University of Oxford
England*

Abstract

We adapt Fiore, Plotkin, and Turi’s treatment of abstract syntax with binding, substitution, and holes to account for languages with second-class sorts. These situations include programming calculi such as the Call-by-Value λ -calculus (CBV) and Levy’s Call-by-Push-Value (CBPV). Prohibiting second-class sorts from appearing in variable contexts changes the characterisation of the abstract syntax from monoids in monoidal categories to actions in actegories. We reproduce much of the development through bicategorical arguments. We apply the resulting theory by proving substitution lemmata for varieties of CBV.

Keywords: denotational semantics, presheaves, abstract syntax, binding, actegories, monoidal actions, skew monoidal categories, bicategories, substitution lemma, holes, metaprogramming.

1 Introduction

Fiore, Plotkin, and Turi’s [39] mathematical foundations for abstract syntax with binding and substitution possess several unique properties. It is based on Goguen et al.’s [46] initial algebra semantics, and as such provides an abstract interface that supports modularity and extensibility. It accommodates both syntactic representations and their semantic models in one semantic setting. Its multi-sorted extension supports intrinsically-typed representation: the abstract syntax also encodes simply-typed constraints, ensuring only

¹ Email: marcelo.fiore@cl.cam.ac.uk

² Email: ohad.kammar@ed.ac.uk

³ Email: georg.moser@uibk.ac.at

⁴ Email: sam.staton@cs.ox.ac.uk

well-typed syntax trees are expressed. It supports context-aware *holes*, called *metavariables*. Despite its mathematical sophistication, it lends itself to formalisation and computational representation [8,9,31,25]. This approach is robust to generalisation, e.g. to polymorphic [37], and dependently-typed [33] settings.

This work concerns another such generalisation: support for second-class sorts [11], as employed by common calculi such as the Call-by-Value λ -calculus and Levy’s Call-by-Push-Value [68,67]. Typically we consider separate syntax for values and for computations, but variables in the language only stand for values, leaving the sorts of computations ‘second-class’ w.r.t. substitution. Supporting the syntactic needs of these calculi is essential for the applicability of this theory to formalisation and computational representation of programming languages. After all, common programming languages and their syntactic representations are overwhelmingly Call-by-Value.

We make these ideas and motivation precise through a comprehensive case-study of a Call-by-Value λ -calculus (CBV) with records/products, variants/coproducts, and a simple inductive type of natural numbers, with various flavours of recursion: structural, unbounded, and general recursion. Fig 1 presents the raw terms of this calculus. Each type A has *two* sorts of terms associated with it: values V and unrestricted terms M . We can embed values into unrestricted terms through the term constructor `val`, and use them wherever we may use a term. Values enjoy a first-class status w.r.t. binding and substitution: we only have value variables and may only substitute values for variables. This distinction partitions the abstract syntax into first-class sorts for values and second-class sorts for computations. Our contribution is to modify the classical theory to allow this distinction between these two classes of sorts.

As a concrete yard-stick for this new theory, consider fragments of the calculus. For example, a fragment involving only functions and records, but not natural numbers. Each such fragment makes a different set of semantic demands on its class of models. For example, functions require certain exponentials, natural numbers require a parameterised natural number object, recursion requires parameterised fixed-points, etc. So long as we consider a fragment in isolation, we can simply aggregate all the required structure into one definition, define the semantic interpretation function, and establish its basic properties directly. However, once we consider multiple fragments simultaneously, we quickly need a modular representation of the syntax, its semantics, and their properties. A concrete example of a property that typically requires tedious reproof is the semantic substitution lemma, which states that the semantics of a substituted term is the semantics of the term composed with the semantics of the substitution: $\llbracket M[\theta] \rrbracket = \llbracket M \rrbracket \circ \llbracket \theta \rrbracket$. Without a modular theory encompassing both syntax and semantics, proving this lemma requires a separate tedious inductive argument for each fragment. The theory we develop here will allow us to prove it modularly for each fragment of the calculus, and combine these results together to $2^7 = 128$ different substitution lemmata for 128 different languages in Lemma 7.4.

This problem is a semantic counterpart to Wadler’s Expression Problem⁵. We do not purport to solve the Expression Problem, which concern the design and implementation of abstract syntax and its evaluator in an existing programming language. We review the immediate literature and draw some connections in the Related Work Sec. 9. Our approach is closest to Swierstra’s à la carte solution to the Expression Problem. Like him, we use coproducts of signature functors [85].

The classical theory phrases the universal property of capture-avoiding substitution over the abstract syntax as follows. The abstract syntax is a presheaf indexed by sorts and simply-typed contexts. Both the syntax and its semantic models form algebras for signature functors over this presheaf category, which allow them to be aware of binding constructs. Both the syntax and its semantic models also form monoids w.r.t. a monoidal tensor product whose unit is the presheaf of variables. This tensor product is the input to capture-avoiding substitution, pairing a value with its closure, subject to the structural properties of substitution. Tensoring with a *pointed* presheaf then provides closures that may contain syntactic or semantic representations of variables. The algebra structure and the substitution monoid must be compatible. Phrasing this compatibility requires a *pointed strength* for the signature functor. It is this strength that yields the theory its modularity. It explains how to propagate variable-aware closures under each syntactic construct, independently from all the other language constructs and their semantics.

⁵ Philip Wadler, *The Expression Problem*, Java Genericity mailing list, 1998:
<https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt> .

$A, B, C ::=$	type	$V, W ::=$	value
β	base	x	variable
$ A \rightarrow B$	function	$ \lambda x : A. M$	function abstraction
$ (C_i : A_i i \in I)$	record (I finite)	$ (C_i : V_i i \in I)$	record constructor
$ \{C_i : A_i i \in I\}$	variant (I finite)	$ A. C_i V$	variant constructor
$ \mathbb{N}$	natural number	$ n$	number literal
$M, N, K, L ::=$			term
	$\text{val } V$		value
$ \text{let } x_1 = M_1; \dots; x_n = M_n \text{ in } N$			sequencing
$ M @ N$			function application
$ (C_1 : M_1, \dots, C_n : M_n)$			record constructor
$ \text{case } M \text{ of } (C_1 x_1, \dots, C_n x_n) \Rightarrow N$			record pattern match
$ A. C_i M$			variant constructor
$ \text{case } M \text{ of } \{C_i x_i \Rightarrow M_i i \in I\} N$			variant pattern match
$ \text{unroll } M \text{roll } M$			number (de)constructor
$ \text{fold } M \text{ by } \{x \Rightarrow N\}$			bounded iteration
$ \text{for } i = M \text{ do } N$			unbounded iteration
$ \text{let rec } f_1 \Gamma_1 : A_1 = M_1; \dots; f_n \Gamma_n : A_n = M_n \text{ in } N$			recursion

Fig. 1. Syntax of CBV

Contribution

Generalising the theory to account for second-class sorts involves refining these ingredients further. We generalise compatible monoids to compatible *actions*. In this sense, we extend Fiore and Turi’s [40] treatment of *value-passing* process calculus (cf. Related Work Sec. 9). The terms of first-class sorts and their semantics still form a monoid, allowing us to collapse towers of substitutions into first-class terms or denotations. This monoid *acts* on terms of second-class sorts and their semantics, but those no longer form a monoid, but an *action*. Just as monoidal categories are the natural categorical setting for monoids, *actegories* are the natural categorical setting for actions. We dub the resulting theory *Modular Abstract Syntax Trees* (MAST). This simple change propagates throughout the theory and requires reformulating it from start to finish. Fortunately, the following relatively recent development makes our work straightforward.

Bicategorical foundations. Fiore, Gambino, Hyland and Winskel [35] characterised the substitution tensor product as 1-cell composition in a Kleisli bicategory over *profunctors*. While technical, this bicategory captures the structure of the tensor as it operates on heterogeneously sorted structures: one set of sorts for syntactic classes, and one set of sorts for the context of bound variables. In the same way that monoidal categories come from one object bicategories, actegories come from a choice of two objects in a bicategory. This observation avoids the many calculations direct treatments incur.

Evaluation. To evaluate MAST, we study fragments of the CBV calculus in Fig 1. MAST’s modularity allows us to formulate succinctly their syntax and semantics, deriving 128 substitution lemmata.

Skew monoidal structure. In an earlier version of this work, we observed that the presheaves indexed by both kinds of sorts and by contexts of first-class sorts have a substitution tensor with an associator and unitors. Unfortunately, the left unit is not invertible. This failure means the tensor is *left-skew* monoidal [87]. Moreover, it is *right-unital* and *associative*, meaning both the right-unit and associator are invertible. Adapting the theory is straightforward, but requires redeveloping it completely. Fiore and Szamozvancev’s [31,32] recent skew monoidal theory of abstract syntax has been influential to our

work. Skewness in their development has a different source. Nonetheless, we reused their results verbatim thanks to our shared categorical abstractions. While our recourse to actegories and actions avoids the need for skew monoidal structure, we can connect the two approaches. We prove that sufficiently well-behaved actegories induce skew monoidal categories, and moreover the category of actions in the actegory is isomorphic to the category of monoids in the skew monoidal category. When working concretely, the skew structure provides a uniform list of proof-obligations, avoiding the need to split the structure into the components of its first-class and second-class sorts.

Alternative. Instead of re-developing the theory, in practice one can take a concrete semantic structure \mathbf{M} with second-class sorts, and complete it to obtain a model $\overline{\mathbf{M}}$ of the classical theory, in which the second-order sorts can appear in contexts. We are not interested in this alternative. First, it leads to unnecessarily complicated models (i.e., the hypothetical $\overline{\mathbf{M}}$) which contain formal/syntactic extensions needed to account for variables of second-class sorts that programs will never exhibit. Second, we would expect proponents of this alternative to prove this kind of completion is always possible. Doing so would require to either empirically complete all known models, or, more satisfyingly, define a completion construction $\mathbf{M} \mapsto \overline{\mathbf{M}}$. In this work, we define models for second-class sorts, i.e., the structure for \mathbf{M} , and moreover show they have the same abstract theory as those with first-class sorts. This development lays the foundation for such wholesale completion construction $\mathbf{M} \mapsto \overline{\mathbf{M}}$, which we leave to further work.

Paper structure. While the theory is technical, applying it concretely involves few self-contained ingredients. Sections 2–6 cover these ingredients in a tutorial style. We accompany each concept with motivating explanations and concrete examples (more than 25 examples overall). This development culminates in the Special Representation Thm 6.4 which characterises the presheaf of abstract syntax with holes in terms of \mathbf{O} -actions. This theorem enables us to prove dozens of substitution lemmata. Sec. 7 reports on a case study: extensions to CBV. Sec. 8 provides a detailed technical outline of the main ingredients in the bicategorical development and presents the General Representation Thm 8.3 for \mathbf{O} -actions. Thm 8.3 implies the Special Representation Thm 6.4 directly, but abstracts away from the concrete syntactic details and technicalities of presheaves using skew monoidal products and actions. Sec. 9 surveys closely related work. Sec. 10 concludes. We prepared an accompanying technical report [38], which is not published as part of this article. It contains the same contents of this article expanded with Appendices A–D, which include further technical details. Appendix A provides omitted proofs concerning our case study. Appendix B expands on our bicategorical development, including: the full definition of a bicategory; how to obtain monoidal actions from bicategories; and the existence of the closed structure for substitution tensors. Appendix C provides the technical details behind our development of monoidal categories, including: actegories, strong functors; and connections to skew monoidal categories. Appendix D gives the full proof for the General Representation Thm 8.3 for MAST, based on parameterised initiality as outlined in Szamozvancev's thesis [86]. We will refer to these Appendices throughout this article.

2 Heterogeneous sorting systems and structures

The first component in the theory specifies the available sorts of syntactic classes of interest. The first-class sorts, unlike the second-class sorts, can appear in contexts and binding positions. Formally, a *heterogeneous sorting system* $\mathbf{R} = (\mathbf{Fst}_\mathbf{R}, \mathbf{Snd}_\mathbf{R}, \mathbf{Sort}_\mathbf{R}, \mathbf{fst}_\mathbf{R}, \mathbf{snd}_\mathbf{R})$ consists of:

- a set $\mathbf{Fst}_\mathbf{R}$ whose elements are the *first-class sorts*;
- a set $\mathbf{Snd}_\mathbf{R}$ whose elements are the *second-class sorts*;
- a coproduct diagram: $\mathbf{Fst}_\mathbf{R} \xrightarrow{\mathbf{fst}_\mathbf{R}} \mathbf{Sort}_\mathbf{R} \xleftarrow{\mathbf{snd}_\mathbf{R}} \mathbf{Snd}_\mathbf{R}$

We denote by $b = \coprod_{i \in I} (c_i : a_i)$ the coproduct diagram with apex b and injections: $(c_i : a_i \rightarrow b)_{i \in I}$. So, for a sorting system \mathbf{R} , we have $\mathbf{Sort}_\mathbf{R} = (\mathbf{fst}_\mathbf{R} : \mathbf{Fst}_\mathbf{R}) \amalg (\mathbf{snd}_\mathbf{R} : \mathbf{Snd}_\mathbf{R})$. We call the apex $\mathbf{Sort}_\mathbf{R}$ the *total* set of sorts. We omit the sorting system \mathbf{R} and write \mathbf{Fst} , \mathbf{Snd} , and \mathbf{Sort} when \mathbf{R} is unambiguous. When we don't specify a set for the apex, we will use the disjoint union and its injections as the apex. A *homogeneous* sorting system is a sorting system which only has first class sorts: $\mathbf{Snd}_\mathbf{R} = \emptyset$ so that $\mathbf{fst}_\mathbf{R} : \mathbf{Fst}_\mathbf{R} \cong \mathbf{Sort}_\mathbf{R}$. In this way, MAST generalises the classical theory from homogeneous sorting systems to heterogeneous ones.

Example 2.1 The homogeneous sorting system CBN for the *call-by-name* λ -calculus has the simple types

as first-class sorts and no second-class sorts, i.e., $\text{Sort}_{\text{CBN}} := \text{Fst}_{\text{CBN}} := \text{SimpleType}$. The sorting system CBN for the *call-by-value* λ -calculus has both a first-class sort for *A-values* and a second-class sort for *A-computations* for each simple type $A \in \text{SimpleType}$:

$$\text{Sort}_{\text{CBV}} := \{A, \text{comp } A \mid A \in \text{SimpleType}\} = (\text{val} := (\lambda A. A) : \text{SimpleType}) \amalg (\text{comp} : \text{SimpleType})$$

Here, the total set of sorts includes two versions of each simple type A : an untagged version, representing the sort of *A-values*, and a tagged version $\text{comp } A$ representing the sort of *A-computations*. We take the inclusion $\text{val } A := A$ as the left coproduct injection, and the tagging function comp as the right injection.

The sorting systems CBN and CBV codify that in CBV only values may be substituted for variables, and CBN does not make this distinction and all expressions can be substituted for variables.

Example 2.2 Each sorting system \mathbf{R} restricts to a homogeneous system $\mathbf{R}|_{\text{fst}} := (\text{Fst}_{\mathbf{R}}, \emptyset)$.

Every set of first-class sorts determines a category of contexts and renamings in the following way. Given a set S , we write $S_{\vdash} := \text{List } S$ for the set of finite sequences $\Gamma \in S_{\vdash}$ with elements in S which we call the *S-sorted contexts*. We will need to refer to positions in a given context. For example, the third position in the context $\Gamma := [A, A \rightarrow B, B]$ has sort B . To simplify such references, we will label these positions with distinct meta-level variable names, and omit the brackets when other symbols delimit the context unambiguously. For example, we will write $\Gamma := x : A, f : A \rightarrow B, y : B$, and refer to the third position by y . This presentation makes it seem as if we use a nominal representation of binding, whereas in fact we use a nameless (i.e., de Bruijn [28]) representation, indexing context positions by ordinals. We thus refer to the set $\mathbb{V}\Gamma$ of positions in a given context Γ as the set of its *variables*, and will use the meta-level labels to refer to its elements. We write $(x : a) \in \Gamma$ when the element a appears in position $x \in \mathbb{V}\Gamma$.

A *renaming* $\rho : \Gamma \rightarrow \Delta$ is a function $-[\rho] : \mathbb{V}\Gamma \leftarrow \mathbb{V}\Delta$ satisfying $(y[\rho] : s) \in \Gamma \iff (y : s) \in \Delta$. The choice of direction (from Γ to Δ) is a matter of taste. One mnemonic for our choice is the fictional typing judgement $\Gamma \vdash \rho : \Delta$. Renamings compose as functions in opposite order with the identity function acting as the identity renaming, and collect into a category S_{\vdash} of *contexts* and *renamings* between them.

Example 2.3 The following is a renaming from the context to itself:

$$\rho : \Gamma := [x : \beta_1, f : \beta_1 \rightarrow \beta_2, g : \beta_1 \rightarrow \beta_2, y : \beta_1] \rightarrow \Gamma \quad x[\rho] := x \quad f[\rho] := g \quad g[\rho] := f \quad y[\rho] := x$$

Thus a renaming may permute the order, e.g., permuting the variables in positions f and g , identify some variables, e.g., x and y , or weaken the context, e.g., y is not in the image.

Example 2.4 Each category S_{\vdash} has chosen finite products. The terminal object is the empty context $\mathbb{1} := []$ with the unique renaming $(_) : \Gamma \rightarrow \mathbb{1}$ being the empty function $\mathbb{V}\Gamma \leftarrow \mathbb{V}[]$. The product of two contexts is their concatenation, with the left/right projection sending the i^{th} position from the left/right to the i^{th} position to the left/right. Letting $\Gamma := [x_1 : s_1, \dots, x_n : s_n]$ and $\Delta := [x_{n+1} : s_{n+1}, \dots, x_{n+m} : s_{n+m}]$:

$$\Gamma \# \Delta := [x_1 : s_1, \dots, x_{n+m} : s_{n+m}] \quad \Gamma \xleftarrow{x_i[\pi_1] := x_i} \Gamma \# \Delta \xrightarrow{x_{n+i} := x_i[\pi_2]} \Delta$$

Let \mathbf{R} be a sorting system. Consider the set $\text{Sort}_{\mathbf{R}}$ as a discrete category. An \mathbf{R} -*structure* P is a presheaf $P \in \mathbf{PSh}(\text{Sort}_{\mathbf{R}} \times (\text{Fst}_{\mathbf{R}})_{\vdash})$, i.e., a functor $P : \text{Sort}_{\mathbf{R}} \times (\text{Fst}_{\mathbf{R}})_{\vdash}^{\text{op}} \rightarrow \mathbf{Set}$ indexed by sorts and contexts and contravariant in renamings. If I is a set, an I -*family* F is a presheaf over the discrete category induced by I , and an \mathbf{R} -*family* is a family over $\text{Sort}_{\mathbf{R}} \times (\text{Fst}_{\mathbf{R}})_{\vdash}$. Thus an \mathbf{R} -*family* F assigns to each sort $s \in \text{Sort}_{\mathbf{R}}$ and context $\Gamma \in (\text{Fst}_{\mathbf{R}})_{\vdash}$ a set $F_s \Gamma$. So an \mathbf{R} -*structure* P is an \mathbf{R} -*family* equipped with a functorial action:

$$-[\rho]_P : P_s \Gamma \leftarrow P_s \Delta \quad p[\text{id}]_P = p \quad (q[\sigma])[\rho] = q[\sigma \circ \rho] \quad (s \in \text{Sort}, p \in P_s \Gamma, q \in P_s \Delta, \text{ and } \Gamma \xrightarrow{\rho} \Delta \xrightarrow{\sigma})$$

Let **R-Struct** be the category of \mathbf{R} -structures and natural transformations $\alpha : P \rightarrow Q$ between them. This category is our central semantic domain. Through it we will define constructions and universal properties for most other concepts. Each \mathbf{R} -*structure* P amounts to two presheaves:

$$P|_{\text{fst}} \in \mathbf{PSh}(\text{Fst} \times \text{Fst}_{\vdash}^{\text{op}}) \quad P|_{\text{snd}} \in \mathbf{PSh}(\text{Snd} \times \text{Fst}_{\vdash}^{\text{op}})$$

which we call the *first-class* and *second-class fragments* of P , respectively. In a homogeneous sorting system, the second-class fragment of every presheaf trivialises to the empty functor from the empty category to **Set**, and we will identify such presheaves with their first-class fragment. In this way, MAST presheaves generalise the homogeneous presheaves of the classical theory.⁶

Example 2.5 Let $\mathbf{Sort} = \mathbf{Fst}$ be a *homogeneous* sorting system, i.e., a set of first-class sorts. The presheaf of *variables* $\mathbb{V} \in \mathbf{Fst}\text{-}\mathbf{Struct}$ is given by $\mathbb{V}_s\Gamma := \{x|(x : s) \in \Gamma\}$ equipped with renaming $x[\rho]_{\mathbb{V}} := x[\rho]$.

Example 2.6 Let \mathbb{A}^{CBV} be the CBV-structure comprising the values and terms of the CBV λ -calculus:

$$\mathbb{A}_{\text{val } A}^{\text{CBV}} \Gamma := \{V|\Gamma \vdash V : \text{val } A\} \quad \mathbb{A}_{\text{comp } A}^{\text{CBV}} \Gamma := \{M|\Gamma \vdash M : \text{comp } A\} \quad X[\rho]_{\mathbb{A}^{\text{CBV}}} := X[\rho]$$

Its functorial action is given by the usual, syntactic, definition of renaming. Its first-class fragment supports a natural transformation from the homogeneous presheaf of variables: $\mathbf{var} : \mathbb{V} \rightarrow \mathbb{A}^{\text{CBV}}|_{\text{val}}$.

Let \mathcal{C} be a category with chosen finite products. Every functor $F : S \rightarrow \mathcal{C}$ extends to a product-preserving functor $F^{\text{Env}} : S_{\vdash} \rightarrow \mathcal{C}$ via:

$$F_{\Gamma}^{\text{Env}} := \prod_{(x:s) \in \Gamma} F_s \quad F_{\rho}^{\text{Env}} := \left(F_{\Gamma}^{\text{Env}} \xrightarrow{\pi_{x[\rho]}} F_s \right)_{(x:s) \in \Delta} : F_{\Gamma}^{\text{Env}} \rightarrow F_{\Delta}^{\text{Env}} \quad (\rho : \Gamma \rightarrow \Delta)$$

The product-preservation condition provides the *concatenation* operation $(\#) : P_{\Gamma}^{\text{Env}} \times P_{\Delta}^{\text{Env}} \xrightarrow{\cong} P_{\Gamma+\Delta}^{\text{Env}}$. The typical case is $\mathcal{C} := \mathbf{PSh}(\mathbf{Fst}_{\mathbf{R}})_{\vdash}$. By considering each homogeneous presheaf $P \in \mathbf{Fst}\text{-}\mathbf{Struct}$ as a functor $\mathbf{Fst}_{\mathbf{R}} \rightarrow \mathbf{PSh}(\mathbf{Fst}_{\mathbf{R}})_{\vdash}$, we form the functor $P^{\text{Env}} \in (\mathbf{Fst}_{\mathbf{R}})_{\vdash} \rightarrow \mathbf{PSh}(\mathbf{Fst}_{\mathbf{R}})_{\vdash}$. We call the elements $e \in P_{\Gamma}^{\text{Env}} \Delta$, the *P-valued Γ -environments* in context Δ . They are are Γ -indexed tuples of $P\Delta$ elements of the appropriate sorts, and can represent both semantic and syntactic substitutions. As with coproducts, we write $b = \prod_{i \in I} (c_i : a_i)$ for the product diagram $(b \xrightarrow{c_i} a_i)_{i \in I}$, and $(c_i : u_i)_{i \in I}$ for the tuple whose i -th component is u_i .

Example 2.7 The following environment is in $(\mathbb{A}^{\text{CBV}})_{[a:\beta, b:\beta, c:\beta \rightarrow \beta]}^{\text{Env}}[x : \beta, f : \beta \rightarrow \beta]$:

$$(a : x, b : x, c : \lambda z : \beta. f @ (f @ z))$$

Remark 2.8 We chose a nameless representation for contexts since it simplifies some concrete aspects in the development. As in the classical theory, this choice is not essential. For example, we can represent contexts nominally as a list pairing variable names and sorts. To define \mathbb{V} , we must disambiguate variables with the same concrete name. These different choices give equivalent small categories of contexts and therefore equivalent categories of presheaves. All of our concepts are defined by universal properties, and so the same development can be carried out in any of those representations.

3 Signature combinators

One defining feature of the classical theory is how it deconstructs signatures with binding [2,75] into smaller components. Formally, and following categorical logic and Goguen’s initial algebra approach to semantics, we use endofunctors $\mathbf{O} : \mathbf{R}\text{-}\mathbf{Struct} \rightarrow \mathbf{R}\text{-}\mathbf{Struct}$ to represent signatures. Each element $\mathbf{op} \in (\mathbf{O}X)_s\Gamma$ represents a language constructs of sort s . The presheaf X represents the sub-terms \mathbf{op} may use and the context Γ represents the free variables in scope. Representing signatures with endofunctors enables some degree of modularity. For example, the coproduct of signature functors $\mathbf{O}_1 \amalg \mathbf{O}_2$ gives terms in which we can take operators from either \mathbf{O}_1 or \mathbf{O}_2 . This decomposition allows us to study each operator on its own and combine their properties modularly. In doing so, we abstract from sorting systems and their categories of structures. Thus we define these combinators on more abstract presheaf categories.

⁶ The other trivialising case, $\mathbf{Sort}_{\mathbf{R}} := \emptyset \amalg \mathbf{Snd}$, yields $\mathbf{R}\text{-}\mathbf{Struct} \cong \mathbf{Set}^{\mathbf{Snd}}$, and so MAST generalises multi-sorted algebra, without accounting for equational presentations and their varieties. We will not make use of this fact further.

Application, restriction, and extension

We often want to project out one or more subterms, or only define a language construct in a specific collection of sorts. For example, in the simplest case we project out or define in a single sort:

$$\begin{array}{lll}
 (@s_0) : \mathbf{R-Struct} \rightarrow \mathbf{PSh}(\mathbf{Fst}_R)_\vdash & X @ s_0 := X_{s_0} & (s_0 \in \mathbf{Sort}_R) \\
 \Downarrow_{s_0} : \mathbf{R-Struct} \leftarrow \mathbf{PSh}(\mathbf{Fst}_R)_\vdash & (\Downarrow Y)_s := \begin{cases} s = s_0 : & Y \\ & \text{otherwise: } \emptyset \end{cases} &
 \end{array}$$

Example 3.1 The CBV inclusion of A -values into A -computations is: $\mathbf{ValOp}_A X := \Downarrow_{\mathbf{comp}_A}(X @ \mathbf{val} A)$.

In a more general form of these combinators we restrict to, and extend along, a function $f : I \rightarrow J$ between sets, for any small category \mathbb{A} , obtaining an adjoint pair of combinators $\Downarrow_f \dashv (|_f)$:

$$\begin{array}{lll}
 (|_f) : \mathbf{PSh}(J \times \mathbb{A}) \rightarrow \mathbf{PSh}(I \times \mathbb{A}) & (X|_f)_i := X_{f_i} & (f : I \rightarrow J \text{ in } \mathbf{Set}) \\
 \Downarrow_f : \mathbf{PSh}(J \times \mathbb{A}) \leftarrow \mathbf{PSh}(I \times \mathbb{A}) & (\Downarrow_f Y)_j := \coprod_{i \in f^{-1}[j]} Y_i &
 \end{array}$$

Example 3.2 Taking $\mathbf{fst} : \mathbf{Fst}_R \rightarrow \mathbf{Sort}_R$ recovers the combinator $(|_{\mathbf{fst}})$. We recover the combinators $(@s_0)$ and \Downarrow_{s_0} using the constant function $s_0 := (\lambda r. s_0) : \mathbb{1} \rightarrow \mathbf{Sort}_R$.

Products and coproducts

As in categorical logic and initial algebra semantics, the bread-and-butter combinators are products and coproducts. Products allow us to express n -ary syntactic constructs. Coproducts allow us to combine signatures into larger signatures. We will use both in many different settings, and so define them in their utmost generality as $\prod_I, \coprod_I : \mathcal{C}^I \rightarrow \mathcal{C}$, where \mathcal{C} has I -indexed products or coproducts, as appropriate.

Example 3.3 We combine the value inclusions in one functor: $\mathbf{ValOp} X := \coprod_{A \in \mathbf{SimpleType}} (\mathbf{val}_A : \mathbf{ValOp}_A X)$.

Example 3.4 Application in CBV has the signature:

$$\mathbf{AppOp} X := \coprod_{A, B \in \mathbf{SimpleType}} \left((@) : \Downarrow_{\mathbf{comp}_B} ((X @ \mathbf{comp}(A \rightarrow B)) \times (X @ \mathbf{comp} A)) \right)$$

Scope shift

We use the following scope shift operation to express operators that bind variables:

$$\Gamma \triangleright : \mathbf{R-Struct} \rightarrow \mathbf{R-Struct} \quad (\Gamma \triangleright X)_s \Delta := X_s(\Delta \# \Gamma) \quad (\Gamma \in (\mathbf{Fst}_R)_\vdash)$$

Aside. In the classical single-sorted and homogeneous theory, scope shift by one variable is presheaf exponentiation by the presheaf of variables. In our setting, $(\Gamma \triangleright) = (\mathbf{y}_\Gamma \multimap)$, where $\mathbf{y} : (\mathbf{Fst}_R)_\vdash \rightarrow \mathbf{PSh}(\mathbf{Fst}_R)_\vdash$ is the Yoneda embedding and $(G \multimap) : \mathbf{R-Struct} \rightarrow \mathbf{R-Struct}$, for a single-sorted presheaf $G \in \mathbf{PSh}((\mathbf{Fst}_R)_\vdash)$, is the right adjoint to the functor $(G \odot) : \mathbf{R-Struct} \rightarrow \mathbf{R-Struct}$ given by: $(G \odot P)_s \Gamma := G\Gamma \times P_s \Gamma$.

Example 3.5 For abstraction: $\mathbf{LamOp} X := \coprod_{A, B \in \mathbf{SimpleType}} \left((\lambda x : A.) : \Downarrow_{A \rightarrow B} [x : A] \triangleright X @ \mathbf{comp} B \right)$.

Combining these examples, we have the full CBV signature: $\mathbf{CvOps} := \mathbf{LamOp} \amalg \mathbf{ValOp} \amalg \mathbf{AppOp}$. The presheaf of syntax is then the initial algebra $\mathbb{A}^{\text{CBV}} = \mu X. ((\mathbf{CvOps} X) \amalg \mathbb{V})$. Using the same methodology, one can mechanically translate, e.g., Aczel's [2,75] *binding signatures* which express a wide class of syntax.

4 Substitution tensors

The substitution tensor imposes semantic invariants on the input for syntactic or semantic substitution operations, which will be of the form $-[-]_P : P \otimes P|_{\text{fst}} \rightarrow P$ where P is a \mathbf{R} -structure standing for the abstract syntax or its semantics. Let R, S, T be sets, and $P \in \mathbf{PSh}(R \times S_{\vdash})$, $Q \in \mathbf{PSh}(S \times T_{\vdash})$ be two heterogeneous presheaves. Their *heterogeneous substitution tensor* is the heterogeneous presheaf:

$$P \otimes Q \in \mathbf{PSh}(R \times T_{\vdash}) \quad (P \otimes Q)_{\Gamma} := \int^{\Delta \in S_{\vdash}} P_{\Delta} \times Q_{\Delta}^{\text{Env}} \Gamma := \left(\coprod_{\Delta \in S_{\vdash}} P_{\Delta} \times Q_{\Delta}^{\text{Env}} \Gamma \right) / (\sim)$$

This definition involves the Q -environment functor $Q^{\text{Env}} : S \rightarrow \mathbf{PSh}(T_{\vdash})$ given by $Q^{\text{Env}} := \prod_{(x:s) \in \Delta} P_s \Gamma$. The coend's quotienting relation (\sim) is the least equivalence relation generated by relating the triples:

$$(\Delta_1, t[\rho]_P, e) \sim (\Delta_2, t, e_{-[\rho]}) \quad (\rho : \Delta_1 \rightarrow \Delta_2, t \in P_s \Delta_2, e \in Q_{\Delta_1}^{\text{Env}} \Gamma)$$

As we explained in the introduction, these identification represent invariants for substitution:

Example 4.1 Consider the syntax presheaf \mathbb{A}^{CBV} . For brevity, we use the vernacular $(k y)$, rather than elaborate $((\text{val } k) @ (\text{val } y))$, syntax. Consider the following equivalences in $\mathbb{A}^{\text{CBV}} \otimes \mathbb{A}^{\text{CBV}}$:

- Assigning the same value to different variables (f, g below) relates to renaming the two variables to one ($f, g \mapsto h$). Formally, taking $f[\rho] := h, g[\rho] := h, t := \lambda x. f(g x)$, and $e := (f : k y, g : k y)$ witnesses:

$$[\lambda x. f(g x), (f : k y, g : k y)]_{[f,g:\beta \rightarrow \beta]} = [\lambda x. h(h x), (h : k y)]_{[h:\beta \rightarrow \beta]} \in (\mathbb{A}^{\text{CBV}} \otimes \mathbb{A}^{\text{CBV}})_{\text{val}(\beta \rightarrow \beta)}[k : \beta \rightarrow \beta, y : \beta]$$

- Weakening the context by unused variables relates to projecting only the used variables. Formally, taking $z[\rho] := z, t := \lambda x. z$, and $e := (f : \lambda x. x, z : y)$ witnesses:

$$[\lambda x. z, (f : \lambda x. x, z : y)]_{[f:\beta \rightarrow \beta, z:\beta]} = [\lambda x. z, (z : y)]_{[z:\beta]} \in (\mathbb{A}^{\text{CBV}} \otimes \mathbb{A}^{\text{CBV}})_{\text{val}(\beta \rightarrow \beta)}[y : \beta]$$

- Permuting the environment relates to permuting the variables. Formally, taking $f[\rho] := g, g[\rho] := f, t := \lambda x. g(f x)$, and $e := (f : \lambda x. x, g : k)$ witnesses:

$$[\lambda x. f(g x), (f : \lambda x. x, g : k)]_{[f,g:\beta \rightarrow \beta]} = [\lambda x. g(f x), (f : k, g : \lambda x. x)]_{[f,g:\beta \rightarrow \beta]} \in (\mathbb{A}^{\text{CBV}} \otimes \mathbb{A}^{\text{CBV}})_{\text{val}(\beta \rightarrow \beta)}[k : \beta \rightarrow \beta]$$

These examples are representative in the following sense. We can represent every renaming $\rho : \Delta_0 \rightarrow \Delta_2$ as the composition: $\rho : \Delta_0 \xrightarrow{i} \Delta_1 \xrightarrow{\tau} \Delta_2$, where: $i : \Delta_0 \rightarrow \Delta_1$ is a renaming with a surjective action on variables, and τ is a *thinning*, a renaming with an injective and relative-order-preserving action on variables. Permuting the variables and then repeatedly identifying some, but not necessarily all, adjacent variables of the same sort generates all renamings with surjective actions. Repeatedly thinning out a variable generates all thinnings. Therefore (\sim) is the smallest equivalence relation that contains the following three identifications (cf. Ex. 4.1):

- Identifying two variables vs. environments containing the same value in their positions.
- Weakening by a thinning vs. projecting according to a thinning.
- Permuting two variables in the term vs. permuting the values in their positions.

The substitution tensor product has the following *left/right unitors* and *associator* maps:

$$\begin{array}{lll} \mathbf{e} : \mathbb{V} \otimes P \xrightarrow{\cong} P & \mathbf{r} : P \otimes \mathbb{V} \xrightarrow{\cong} P & \mathbf{a} : (P \otimes Q) \otimes L \xrightarrow{\cong} P \otimes (Q \otimes L) \\ \mathbf{e}[x, e]_{\Delta} := e_x & \mathbf{r}[p, \rho]_{\Delta} := p[\rho] & \mathbf{a}[[p, q]_{\Delta_1}, e]_{\Delta_2} := \left[p, \left([q_x, e]_{\Delta_2} \right)_{(x:s) \in \Delta_1} \right]_{\Delta_1} \end{array}$$

While these *mediators* satisfy familiar-looking pentagon and triangle laws, they are best understood as part of a bicategory whose 0-cells are small categories, 1-cells are generalised heterogeneous preshaves, and 2-cells are natural transformations between them. Fiore, Gambino, Hyland and Winskel [35] introduced and investigated this bicategorical perspective in §8.1.

Let \mathbf{R} be a heterogeneous sorting system. The substitution tensor then provides a two-argument functor $(\otimes) : \mathbf{R}\text{-Struct} \times \mathbf{Fst}_{\mathbf{R}}\text{-Struct} \rightarrow \mathbf{R}\text{-Struct}$. We understand it abstractly through actions. Formally, a *monoidal action* $(\mathcal{V}, \mathcal{A})$ consists of:

- A monoidal category $\mathcal{V} = (\underline{\mathcal{V}}, \otimes, \mathbb{I}, \mathbf{a}, \boldsymbol{\ell}, \mathbf{r})$, we will typically write $\mathbf{r}' := \mathbf{r}^{-1}$; and
- A \mathcal{V} -actegory $\mathcal{A} = (\underline{\mathcal{A}}, \otimes, \mathbf{a}, \mathbf{r})$, i.e., a functor and isomorphisms, natural in $a \in \underline{\mathcal{A}}$ and $x, y \in \underline{\mathcal{V}}$:

$$(\otimes) : \mathcal{A} \times \mathcal{V} \rightarrow \mathcal{A} \quad \mathbf{a}_{a,x,y} : (a \otimes x) \otimes y \rightarrow a \otimes (x \otimes y) \quad \mathbf{r}_a : a \otimes \mathbb{I} \rightarrow a$$

satisfying the following equations:

$$\begin{array}{ccc} \begin{array}{c} \text{action} \\ \text{pentagon} \\ = \end{array} & \begin{array}{c} (x \otimes a) \otimes (b \otimes c) \\ \text{a} \nearrow \quad \text{a} \searrow \\ ((x \otimes a) \otimes b) \otimes c \\ \text{a} \otimes \text{id} \swarrow \quad \text{id} \otimes \text{a} \searrow \\ (x \otimes (a \otimes b)) \otimes c \quad x \otimes ((a \otimes b) \otimes c) \\ \text{a} \nearrow \quad \text{a} \searrow \\ a \end{array} & \begin{array}{c} (x \otimes \mathbb{I}) \otimes a \xrightarrow{\text{a}} x \otimes (\mathbb{I} \otimes a) \\ \text{action} \\ \text{triangle} \\ = \end{array} \\ & \begin{array}{c} \text{r} \otimes \text{id} \swarrow \quad \text{id} \otimes \boldsymbol{\ell} \searrow \\ x \otimes a \end{array} & \end{array}$$

Take $\mathbf{R}\text{-Struct}_{\text{snd}} := \mathbf{PSh}(\mathbf{Snd}_{\mathbf{R}} \times (\mathbf{Fst}_{\mathbf{R}})_{\vdash})$ as the heterogeneous structures with second-class sorts only:

Theorem 4.2 *Each heterogeneous sorting system \mathbf{R} gives a monoidal action $(\mathbf{Fst}_{\mathbf{R}}\text{-Struct}, \mathbf{R}\text{-Struct}_{\text{snd}})$:*

- *The monoidal category of homogeneous $\mathbf{Fst}_{\mathbf{R}}$ -structures equipped with substitution tensors, the presheaf of variables, and their associator and unitors:*

$$\mathbf{Fst}\text{-Struct} = (\mathbf{PSh}(\mathbf{Fst}_{\mathbf{R}} \times (\mathbf{Fst}_{\mathbf{R}})_{\vdash}), (\otimes), \mathbb{V}, \mathbf{a}, \boldsymbol{\ell}, \mathbf{r})$$

- *The $\mathbf{Fst}_{\mathbf{R}}$ -actegory of the second-class-sorted structures equipped with substitution tensors and mediators:*

$$\mathbf{R}\text{-Struct}_{\text{snd}} = (\mathbf{PSh}(\mathbf{Snd}_{\mathbf{R}} \times (\mathbf{Fst}_{\mathbf{R}})_{\vdash}), (\otimes), \mathbf{a}, \mathbf{r})$$

The theorem is a direct consequence of the bicategorical development [35] (cf. Prop. 8.1). The monoidal category is the one from the classical theory. Identifying the actegory structure is the technical innovation in MAST. The projections into the first-class and second-class components form an isomorphism of categories $((|_{\text{fst}}, |_{\text{snd}}) : \mathbf{R}\text{-Struct} \xrightarrow{\cong} \mathbf{Fst}\text{-Struct} \times \mathbf{R}\text{-Struct}_{\text{snd}})$. We will use its inverse in the sequel:

$$\langle\langle -, - \rangle\rangle : \mathbf{Fst}\text{-Struct} \times \mathbf{R}\text{-Struct}_{\text{snd}} \xrightarrow{\cong} \mathbf{R}\text{-Struct} \quad \langle\langle P, Q \rangle\rangle|_{\text{fst}} = P \quad \langle\langle P, Q \rangle\rangle|_{\text{snd}} = Q \quad L = \langle\langle L|_{\text{fst}}, L|_{\text{snd}} \rangle\rangle$$

Since each monoidal category acts on itself, and actegories have componentwise products, we can and will extend the action, through this isomorphism, to an action of the homogeneous structures on all \mathbf{R} -structures: $(\otimes) : \mathbf{R}\text{-Struct} \times \mathbf{Fst}_{\mathbf{R}} \rightarrow \mathbf{R}\text{-Struct}$, satisfying: $(P \otimes Q)|_{\text{fst}} = P|_{\text{fst}} \otimes Q$. However, the Representation Thm 6.4 in the sequel needs the action from Thm 4.2, not the extended action (\otimes) .

We will use the following two well-known constructions to combine actegories compositionally:

Example 4.3 Every monoidal category \mathcal{V} is a \mathcal{V} -actegory through its own monoidal tensor, i.e., taking: $(a \otimes b) := (a \otimes b)$. The actegory axioms are a subset of the monoidal ones.

Example 4.4 Given a sequence of \mathcal{V} -actegories $(\mathcal{A}_i, (\otimes^i), \mathbb{V}^i, \mathbf{a}^i, \mathbf{r}^i)_i$, we define the *product \mathcal{V} -actegory* componentwise as follows, validating the axioms componentwise:

$$(\otimes) : (\prod_{i \in I} \mathcal{A}_i) \times \mathcal{C} \rightarrow \prod_{i \in I} \mathcal{A}_i \quad (x \otimes a) := (x_i \otimes^i a)_i \quad \mathbf{a} := (\mathbf{a}^i)_i \quad \mathbf{r} := (\mathbf{r}^i)_i$$

Let $(\mathcal{V}, \mathcal{A})$ be a monoidal action. A $(\mathcal{V}, \mathcal{A})$ -action (\mathbf{M}, \mathbf{A}) consists of:

- A \mathcal{V} -monoid \mathbf{M} , i.e., a triple $(\underline{\mathbf{M}}, \bullet, \mathbf{e})$ consisting of an object $\underline{\mathbf{M}} \in \mathcal{V}$ and two morphisms $\mathbb{I} \xrightarrow{\mathbf{e}} \underline{\mathbf{M}} \xleftarrow{\bullet} \underline{\mathbf{M}} \otimes \underline{\mathbf{M}}$ satisfying the following three equations:

- An \mathbf{M} -action \mathbf{A} in \mathcal{A} , i.e., a pair $(\underline{\mathbf{A}}, \bullet)$ consisting of an object $\underline{\mathbf{A}} \in \mathcal{A}$ and a morphism $\bullet : \underline{\mathbf{A}} \otimes \underline{\mathbf{M}} \rightarrow \underline{\mathbf{A}}$ satisfying the following two equations:

Generalising the classical theory, we use monoids and actions with respect to the substitution tensors to axiomatise syntactic and semantic capture-avoiding substitution operations. We therefore use the more suggestive notation $\mathbf{M} = (\underline{\mathbf{M}}, -[-], \mathbf{var})$ for monoids in **Fst-Struct** and $\mathbf{A} = (\underline{\mathbf{A}}, -[-])$ for their actions in **R-Struct_{nd}**, which we call *substitution actions*.

Example 4.5 The first-class fragment of the syntax presheaf \mathbb{A}^{CBV} has a CBV-monoid structure given by capture-avoiding simultaneous substitution on values $[V, \theta]_\Gamma \mapsto V[\theta]$. The inclusion of variables in values is the map $\mathbf{var} : \mathbb{V}_A \xrightarrow{x \mapsto x} (\mathbb{A}^{\text{CBV}})_A$. This monoid acts on the second-class fragment through capture-avoiding substitution on computations $[M, \theta]_\Gamma \mapsto M[\theta]$. The five axioms become the familiar substitution lemmata:

$$x[\theta] = \theta_x \quad (V[\theta])[\sigma] = V[\theta[\sigma]] \quad V[\text{id}] = V \quad (M[\theta])[\sigma] = M[\theta[\sigma]] \quad M[\text{id}] = M$$

We will derive capture-avoiding substitution and its meta-theory from the Representation Thm 6.4.

Example 4.6 Let \mathcal{C} be a Cartesian-closed category with chosen finite products \prod and exponentials b^a . Every choice of interpretation $[\beta]$ for the base types equips \mathcal{C} with a CBV-action structure by first extending the interpretation to types and contexts, and then defining the carrier presheaf by:

$$[\mathbf{A} \rightarrow \mathbf{B}] := [\mathbf{B}]^A \quad [\Gamma] := \prod_{(x:A) \in \Gamma} [\mathbf{A}] \quad \underline{\mathbf{M}}_A \Gamma := \mathcal{C}([\Gamma], [\mathbf{A}]) \quad \underline{\mathbf{A}}_{\text{comp}}_A \Gamma := \mathcal{C}([\Gamma], [\mathbf{A}])$$

Define the unit by projection $(\mathbf{var}_{B;\Gamma} y) : [\Gamma] = \prod_{(x:A) \in \Gamma} [\mathbf{A}] \xrightarrow{\pi_y} [\mathbf{B}]$; and substitution by composition:

$$([\Delta] \xrightarrow{f} [\mathbf{A}]) ([\Gamma] \xrightarrow{\theta_y} [\mathbf{B}])_{(y:B) \in \Delta} := \left([\Gamma] \xrightarrow{(\theta_y)_{(y:B) \in \Delta}} \prod_{(y:B) \in \Delta} [\mathbf{B}] = [\Delta] \xrightarrow{f} [\mathbf{A}] \right)$$

We explore the richer strong-monad models for CBV in our case study in Sec. 7.

The isomorphism $\langle\langle -, - \rangle\rangle : \mathbf{Fst-Struct} \times \mathbf{R-Struct}_{\text{nd}} \xrightarrow{\cong} \mathbf{R-Struct}$ lets us re-package substitution actions in terms of **R**-structures. A *substitution structure* $\mathbf{S} = (\underline{\mathbf{S}}, -[-]_{\mathbf{S}}, \mathbf{var}_{\mathbf{S}})$ is an **R**-structure $\underline{\mathbf{S}}$ equipped with morphisms $\mathbf{var}_{\mathbf{S}} : \mathbb{V} \rightarrow \underline{\mathbf{S}}_{\text{fst}}$ and $-[-] : \underline{\mathbf{S}} \otimes \underline{\mathbf{S}}_{\text{fst}} \rightarrow \underline{\mathbf{S}}$ forming a monoid $(\mathbf{S}|_{\text{fst}}, \mathbf{var}, (-|_{\text{fst}})[-]_{\mathbf{S}})$ and an

action of it $(\mathbf{S}_{\text{snd}}, (-|_{\text{snd}})[-]_{\mathbf{S}})$. I.e., this monoid and its action form a substitution action. Thus we will speak of substitution actions when we refer to a pair of structures—a homogeneous monoid and its action on a heterogeneous structure over second-class sorts. We will speak of substitution structures when we want to emphasise the combined heterogeneous structure over both first-class and second-class sorts.

5 Signature functors

So far we have specified functors \mathbf{O} , their algebras, and actions. We can equip the abstract syntax with this structure, and its denotational semantics with this structure. We can even characterise the abstract syntax as an initial algebra $(\mu X.(\mathbf{O}X) \amalg \langle\!\langle \mathbb{V}, \emptyset \rangle\!\rangle, \text{roll})$ for the signature, and the denotational semantics as the unique $(\mathbf{O} \amalg \langle\!\langle \mathbb{V}, \emptyset \rangle\!\rangle)$ -algebra homomorphism. This characterisation, however, does not account for neither the syntactic nor semantic substitution lemmata. The missing ingredient is a structural map, a tensorial strength for the signature functor. The strength specifies how to avoid unintended capture when moving under each operator in the signature. More precisely, it specifies how to change an environment containing the free variables in scope when we propagate it to the arguments of each language construct. It is this information, phrased w.r.t. arbitrary environments, that gives MAST much of its modularity. The overall signature functor is a coproduct of signature functors, each with their own strength. Each strength determines how to substitute through each language construct independently from the other language constructs. To define the strengths, the classical theory uses pointed tensors and their actions, which we adapt to heterogeneous structures in this section.

To describe scope changes, the classical theory uses a point-free way to consider those homogeneous prehseaves $P \in \mathbf{Fst}\text{-}\mathbf{Struct}$ that can, moreover, encode variables $\llbracket x : s \rrbracket_P \in P_s[x : s]$. Since we have that $\mathbb{V}_s = \mathbf{y}_{[x : s]} \in \mathbf{PSh}(\mathbf{Fst}_R)$, the Yoneda lemma represents $(\llbracket x : s \rrbracket_P)_{s \in \mathbf{Fst}_R}$ via a natural transformation:

$$\mathbf{var} : \mathbb{V} \rightarrow P \quad \mathbf{var}_{s; \Gamma} x := \llbracket x : s \rrbracket_P \left[[x : s] \xleftarrow{\pi_x} \Gamma \right] \in P_s \Gamma \quad \llbracket x : s \rrbracket_P := \mathbf{var}_{s; [x : s]} x$$

In MAST we follow the same technique. We start by organising these presheaves into a monoidal category of their own. Let $\mathcal{V} = (\underline{\mathcal{V}}, (\otimes), \mathbb{I}, \mathbf{a}, \boldsymbol{\ell}, \mathbf{r})$ be a monoidal category. Recall the *coslice* category $\underline{\mathcal{V}}^\bullet := \mathbb{I}/\underline{\mathcal{V}}$:

- *pointed objects* A : pairs $(\underline{A}, \mathbf{var}_A : \mathbb{I} \rightarrow A)$ consisting of an object \underline{A} in $\underline{\mathcal{V}}$ and a $\underline{\mathcal{V}}$ -arrow $\mathbf{var}_A : \mathbb{I} \rightarrow A$ called the *point*; and
- arrows $f : A \rightarrow B$ are point-preserving $\underline{\mathcal{V}}$ -morphisms $f : \underline{A} \rightarrow \underline{B}$ (cf. diagram on right).

$$\begin{array}{ccc} \mathbf{var}_A & \nearrow & \underline{A} \\ \mathbb{I} & = & \downarrow f \\ & \searrow & \mathbf{var}_B \quad \underline{B} \end{array}$$

The forgetful functor $\underline{-} : \underline{\mathcal{V}}^\bullet \rightarrow \underline{\mathcal{V}}$ sending pointed objects and morphisms to their underlying objects and morphisms is faithful. The tensor product (\otimes) lifts along $\underline{-}$ to the following *pointed* tensor:

$$(\otimes^\bullet) : \underline{\mathcal{V}}^\bullet \times \underline{\mathcal{V}}^\bullet \rightarrow \underline{\mathcal{V}}^\bullet \quad \underline{A} \otimes^\bullet \underline{B} : \underline{A} \otimes \underline{B} \quad \mathbf{var}_{A \otimes^\bullet B} := \mathbb{I} \xrightarrow{\mathbf{r}'} \mathbb{I} \otimes \mathbb{I} \xrightarrow{\mathbf{var}_A \otimes \mathbf{var}_B} \underline{A} \otimes \underline{B} \quad (A \xrightarrow{f} A') \otimes^\bullet (B \xrightarrow{g} B') := f \otimes g$$

This definition for $f \otimes^\bullet g$ relies on a simple point-preservation argument (cf. §C.1).

The initial pointed object is given by \mathbb{I} equipped with its identity: $\mathbb{I}^\bullet := (\mathbb{I}, \text{id}_{\mathbb{I}}) \in \underline{\mathcal{V}}^\bullet$. The unique point-preserving morphism to any pointed object A is given by the point: $[] := \mathbf{var}_A : \mathbb{I}^\bullet \rightarrow A$.

The pointed objects inherit the monoidal structure from \mathcal{V} (see §C.1 for the proof):

Proposition 5.1 (Fiore [33]) *Every monoidal category \mathcal{V} yields a monoidal category of pointed objects $\mathcal{V}^\bullet := (\underline{\mathcal{V}}^\bullet, (\otimes^\bullet), \mathbb{I}^\bullet, \mathbf{a}, \boldsymbol{\ell}, \mathbf{r})$. I.e., the mediators preserve points, hence lift to \mathcal{V}^\bullet .*

Example 5.2 Let $A \in \mathbf{R}\text{-}\mathbf{Struct}^\bullet$ be a pointed structure. Its point \mathbf{var}_A is uniquely determined, through the Yoneda lemma, by the tuple of variable interpretations $(\llbracket x : s \rrbracket_A := (\mathbf{var}_A)_{s; [x : s]} x)_{s \in \mathbf{Fst}}$. Given any other pointed structure B , the variable interpretation for $A \otimes^\bullet B$ is $\llbracket x : s \rrbracket_{A \otimes^\bullet B} = [\llbracket x \rrbracket_A, (x : \llbracket x \rrbracket_B)]_{[x : s]}$.

Example 5.3 If \mathcal{A} is a \mathcal{V} -actegory, then the monoidal category of pointed objects \mathcal{V}^\bullet acts on \mathcal{A} by: $(a \otimes_\bullet A) := (a \otimes \underline{A})$. The \mathcal{V}^\bullet -actegory axioms hold straightforwardly. We denote this \mathcal{V}^\bullet -actegory by \mathcal{A}_\bullet . In particular, we have an $\mathbf{R}\text{-}\mathbf{Struct}^\bullet$ -actegory $\mathbf{R}\text{-}\mathbf{Struct}$, given by the \mathbf{R} -structures.

Let \mathcal{A}, \mathcal{B} be two \mathcal{V} -actegories, and consider any functor $F : \mathcal{A} \rightarrow \mathcal{B}$. A *tensorial strength* for F from \mathcal{A} to \mathcal{B} is a natural transformation $\text{str} : (Fa) \otimes_{\mathcal{A}} b \rightarrow F(a \otimes b)$ satisfying the two axioms:

$$\begin{array}{c}
 \begin{array}{ccc}
 \begin{array}{c} (Fx) \otimes (a \otimes b) \\ \text{str} \end{array} & \xrightarrow{\quad \text{strength pentagon} \quad} & F(x \otimes (a \otimes b)) \\
 \begin{array}{c} a \nearrow \\ ((Fx) \otimes a) \otimes b \\ \text{str} \otimes \text{id} \end{array} & = & \begin{array}{c} \nearrow Fx \\ F \mathbf{a} \end{array} \\
 \begin{array}{c} (F(x \otimes a)) \otimes b \\ \text{str} \end{array} & \xrightarrow{\quad \text{str} \quad} & F((x \otimes a) \otimes b)
 \end{array} \\
 \begin{array}{c} \begin{array}{ccc} & Fx & \\ \mathbf{r} & \nearrow & \searrow F\mathbf{r} \\ (Fx) \otimes \mathbb{I} & \xrightarrow{\quad \text{str} \quad} & F(x \otimes \mathbb{I}) \end{array} \\
 \begin{array}{c} \text{strength triangle} \\ = \end{array}
 \end{array}
 \end{array}$$

A *strong functor* $F : \mathcal{A} \rightarrow \mathcal{B}$ is a functor $\underline{F} : \underline{\mathcal{A}} \rightarrow \underline{\mathcal{B}}$ equipped with a strength, $\text{str}_{\underline{F}}$, for \underline{F} from \mathcal{A} to \mathcal{B} .

The strong functors w.r.t. the pointed monoidal structure are central to the modularity of both the classical theory and MAST. As we will soon see, the strength of the signature functor will provide the structure that allows us to propagate syntactic and semantic substitutions through each term constructor.

Definition 5.4 An **R-signature functor** is an **R-Struct**[•]-strong functor: $\mathbf{O} : \mathbf{R-Struct}_+ \rightarrow \mathbf{R-Struct}_+$.

Each signature functor explains, through its strength, how to propagate substitutions to its subterms:

Example 5.5 The strength for the abstraction signature (Ex. 3.5):

$$\begin{array}{c}
 \text{LamOp} := \left\{ (\lambda x : A.) : \underset{A \rightarrow B}{\underset{\text{def}}{\leftrightarrow}} [x : A] \triangleright X @ A \mid A, B \in \text{SimpleType} \right\} \\
 \text{str}_{P, A; B \rightarrow C, \Gamma}^{\text{LamOp}} [\lambda x : A.t, e]_{\Delta} := \lambda x : B. \left[t, e \left[\Gamma \xleftarrow{\pi_1} \Gamma \# [x : B] \right] \# (\text{var}_A)_{\Gamma \# [x : B]} x \right]_{\Delta \# [x : B]}
 \end{array}$$

Thus, to propagate the environment e under the binder, we *weaken* it. The strengths for the other CBV signature functors from Exs. 3.3 and 3.4 propagate the environment as it is:

$$\text{str}^{\text{ValOp}} [\text{val } t, e]_{\Delta} := \text{val } [t, e]_{\Delta} \quad \text{str}^{\text{AppOp}} [t_1 @ t_2, e]_{\Delta} := [t_1, e]_{\Delta} @ [t_2, e]_{\Delta}$$

These rules are identical to the syntactic rules one may use to define capture-avoiding substitution, but crucially the environment e is taken from any pointed presheaf, not just the presheaf of abstract syntax. It is this additional generality that allows the classical theory and MAST to combine signatures modularly, as well as account for both syntactic and semantic substitution.

It is straightforward to show the transformations in the previous example are strengths from first principles. However, this fact follows compositionally. We will spend the remainder of this section building up the machinery to derive this strength compositionally.

Example 5.6 Recall the *scope shift* combinator $(\triangleright \Gamma) : \mathbf{R-Struct} \rightarrow \mathbf{R-Struct}$ that we use to describe binding constructs, such as abstraction. We define its strength $\text{str}_{P, A}^{\triangleright \Gamma} : (\Gamma \triangleright P) \otimes_{\mathcal{A}} A \rightarrow \Gamma \triangleright (P \otimes_{\mathcal{A}} A)$ by:

$$\begin{array}{c}
 \text{str}_{s; \Xi}^{\triangleright \Gamma} [t \in P_s(\Delta \# \Gamma), e \in \underline{A}_{\Delta}^{\text{Env}} \Xi]_{\Delta} := \\
 \left[t \in P_s(\Delta \# \Gamma), \left(e_x \left[\Xi \xleftarrow{\pi_1} \Xi \# \Gamma \right] \right)_{(x: s) \in \Delta} \# \left(\llbracket y \rrbracket_A \left[[y : r] \xleftarrow{\pi_{y[r_2]}} \Xi \# \Gamma \right] \right)_{(y: r) \in \Gamma} \in \underline{A}_{\Delta \# \Gamma}^{\text{Env}} (\Xi \# \Gamma) \right]_{\Delta \# \Gamma}
 \end{array}$$

I.e., in order to propagate the environment under a scope-shift, we extend it by the environment that sends every variable $(y : r) \in \Delta$ to its representation $\llbracket y \rrbracket$ in A , suitably weakened into the context $\Xi \# \Gamma$.

When the operators in a signature do not bind variables themselves but are merely compatible with binding, we can exhibit a strength w.r.t. the simpler action (\otimes) thanks to the following result, which appears implicitly in Fiore's [33] work (see §C.2 for the straightforward proof):

Lemma 5.7 Let \mathcal{V} be a monoidal category; \mathcal{A}, \mathcal{B} two \mathcal{V} -actegories; and $F : \mathcal{A} \rightarrow \mathcal{B}$ a strong functor. Then the following morphisms exhibit \underline{F} as a strong functor: $\underline{F}_* : \mathcal{A}_* \rightarrow \mathcal{B}_*$.

$$\text{str}_{x,a}^F : (\underline{F}x) \otimes a = (\underline{F}x) \otimes \underline{a} \xrightarrow{\text{str}_{x,\underline{a}}^F} \underline{F}(x \otimes a) = \underline{F}(x \otimes a)$$

The next few examples all use this lemma to derive strengths.

Example 5.8 Recall the restriction combinator $(|_I) : \mathbf{R-Struct} \rightarrow \mathbf{PSh}(I \times \mathbf{Fst}_\vdash)$ and its specialisation that projects out a single sort $(@s_0) : \mathbf{R-Struct} \rightarrow \mathbf{PSh}(\mathbf{Fst}_\vdash)$ (where $s_0 \in \mathbf{Sort}$) that we use it to signify a sub-term of sort s_0 . Define $\text{str}^{|_I} : P|_I \otimes Q \rightarrow (P \otimes Q)|_I$ by $\text{str}_{s,\Gamma}^{|_I} \left[t \in P_{s_0} \Delta, e \in Q_\Delta^{\text{Env}} \Gamma \right]_\Delta := [t, e]_\Delta$, and derive a strength for the projection $\text{str}^{@s_0} : (P @ s_0) \otimes Q \rightarrow (P \otimes Q) @ s_0$ from it. Similarly, recall the *extension* combinator $\wp_I : \mathbf{PSh} I \times (\mathbf{Fst}_\mathbf{R})_\vdash \rightarrow \mathbf{R-Struct}$ that lets us construct nodes at a specific subset of sorts $I \subseteq \mathbf{SortR}$. Then $((\wp_I P) \otimes Q)_s = \emptyset$ when $s \notin I$, and so we can define the strength $\text{str}_{P;Q}^{\wp_I} : (\wp_P) \otimes Q \rightarrow \wp_{P \otimes Q}$ vacuously in those sorts:

$$\text{str}_{P;Q;s}^{\wp_I} : ((\wp_P) \otimes Q)_s = \emptyset \xrightarrow{!} \wp_I(P \otimes Q)_s \quad \text{str}_{P;Q;r}^{\wp_I} : ((\wp_P) \otimes Q)_r = (P \otimes Q)_r = \wp_I(P \otimes Q)_r \quad (s \notin I \ni r)$$

Example 5.9 Let $(\mathcal{A}_i)_{i \in I}$ be a family of \mathcal{V} -actegories. The projection functors $\pi_j : \prod_{i \in I} \mathcal{A}_i \rightarrow \mathcal{A}_j$ have the identity as a strength $\text{str}_{(x_i)_i, a}^{\pi_j} = \text{id}_{x_i \otimes a}$ since: $(\pi_j(x_i)_i) \otimes_i a = x_j \otimes a = \pi_j(x_i)_i \otimes a$. For a \mathcal{V} -actegory \mathcal{B} and a family of strong functors $F_i : \mathcal{B} \rightarrow \mathcal{A}_i$, the tupling functor $(\underline{F}_i)_i : \mathcal{B} \rightarrow \prod_i \mathcal{A}_i$ has the following $\prod_i \mathcal{A}_i$ -morphism as strength: $\text{str}_{(x_i)_i, a}^{(F_i)_i} := \left((F_i x) \otimes a \xrightarrow{\text{str}_{x,a}^{F_i}} F_i(x \otimes a) \right)_i$.

Example 5.10 Recall the I -ary sums $\coprod_I : \mathcal{C}^I \rightarrow \mathcal{C}$ and J -ary products $\prod_J : \mathcal{C}^J \rightarrow \mathcal{C}$ that let us alternate between I -indexed operator nodes and include J -ary branching factor in categories with I -coproducts/ J -products. The product has strength, and coproducts, if they distribute over (\otimes) , also have a strength:

$$\text{str}^{\prod_J} : \left(\prod_{j \in J} x_j \right) \otimes a \xrightarrow{(\pi_j \otimes \text{id})_{j \in J}} \prod_{j \in J} (x_j \otimes a) \quad \text{str}^{\coprod_I} : \left(\coprod_{i \in I} x_i \right) \otimes a \xrightarrow{[(i : t) \otimes \text{id}]_{i \in I}^{-1}} \coprod_{i \in I} (x_i \otimes a)$$

When $\mathcal{C} = \mathbf{R-Struct}$, the substitution tensor (\otimes) distributes over coproducts (cf. Prop. 8.2), and so the following pointwise formulae describe these strengths:

$$\text{str}_{P,Q}^{\prod_J} [t, e]_\Delta := ([t_j, e]_\Delta)_{j \in J} \quad \text{str}_{P,Q}^{\coprod_I} [(i : t), e]_\Delta := (i : [t, e]_\Delta)$$

As is well-known, strong functors compose, and their combined strength is given by (see Lemma C.2):

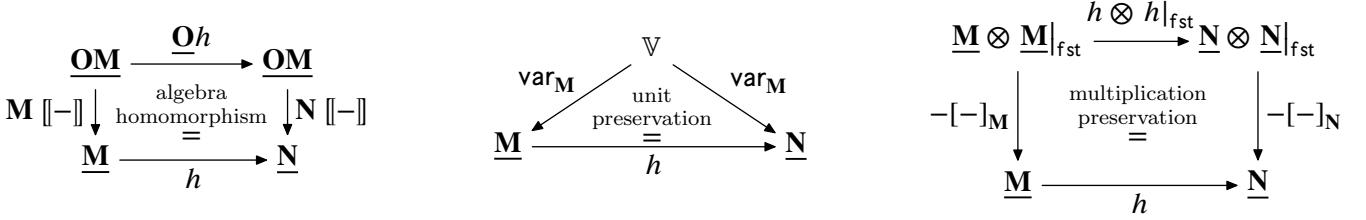
$$\text{str}_{x,a}^{G \circ F} : (\underline{G}F x) \otimes a \xrightarrow{\text{str}_x^G} \underline{G}(\underline{F}x \otimes a) \xrightarrow{\underline{G} \text{str}_a^F} \underline{G}F(x \otimes a)$$

This fact and the signature combinator covers all our examples. E.g., cf. the strength from Ex. 5.5.

6 Compatible actions and structures

We define the substitution structure of interpretations of the syntax that ensures they are compatible with the operations in the signature, culminating in the Special Representation Thm 6.4, concluding the tutorial. Actions also give additional perspectives on the compatibility condition (§6.1).

Definition 6.1 Let \mathbf{O} be an \mathbf{R} -signature functor, $\mathbf{M} = (\underline{\mathbf{M}}, -[-]_{\mathbf{M}}, \text{var}_{\mathbf{M}})$ a substitution structure. We say that an \mathbf{O} -algebra $\llbracket - \rrbracket : \underline{\mathbf{OM}} \rightarrow \underline{\mathbf{M}}$ is *compatible* with \mathbf{M} , when:

Fig. 2. Definition of an \mathbf{O} -substitution structure homomorphism $h : \mathbf{M} \rightarrow \mathbf{N}$

$$\begin{array}{c}
 \mathbf{str} \nearrow \mathbf{O}(\mathbf{M} \otimes \mathbf{M}) \\
 (\mathbf{OM}) \otimes \mathbf{var}_M \xrightarrow{\text{compatibility}} \mathbf{O}(-[-]_M) \\
 \mathbf{[-]} \otimes \mathbf{id} \searrow = \mathbf{OM} \\
 \mathbf{M} \otimes \mathbf{var}_M \xrightarrow{-[-]_M} \mathbf{M}
 \end{array}$$

An \mathbf{O} -compatible substitution structure $\mathbf{M} = (\underline{\mathbf{M}}, -[-]_M, \mathbf{var}_M, \mathbf{[-]}_M)$, or \mathbf{O} -structure for short, consists of a structure $\mathbf{M} = (\underline{\mathbf{M}}, -[-]_M, \mathbf{var}_M)$ and an \mathbf{O} -algebra $\mathbf{[-]}_M : \mathbf{OM} \rightarrow \underline{\mathbf{M}}$, compatible with \mathbf{M} .

Example 6.2 Recall the substitution structure \mathbf{M} for CBV in a Cartesian-closed category \mathcal{C} (Ex. 4.6). The interpretation of the CBV λ -calculus equips it with a \mathbf{LamOp} -, \mathbf{ValOp} -, and \mathbf{AppOp} -algebra structures:

$$\begin{aligned}
 \llbracket \lambda x : A. \rrbracket_M \left(\llbracket \Gamma, x : A \rrbracket \xrightarrow{f} \llbracket B \rrbracket \right) &:= \left(\llbracket \Gamma \rrbracket \xrightarrow{\text{curry } f} \llbracket B \rrbracket^{[A]} \right) & \llbracket \text{val} \rrbracket_M \left(\llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket A \rrbracket \right) &:= f \\
 \llbracket \left(\llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket B \rrbracket^A \right) @ \left(\llbracket \Gamma \rrbracket \xrightarrow{a} \llbracket B \rrbracket \right) \rrbracket_M &:= \left(\llbracket \Gamma \rrbracket \xrightarrow{(f, a)} \llbracket B \rrbracket^{[A]} \times \llbracket A \rrbracket \xrightarrow{\text{eval}} \llbracket B \rrbracket \right)
 \end{aligned}$$

The compatibility axiom for \mathbf{LamOp} amounts to the following equation, for each $\theta \in C(\llbracket \Gamma \rrbracket, \llbracket \Delta \rrbracket) \cong \mathbf{M}_\Delta^{\text{Env}} \Gamma$:

$$\text{curry}(\llbracket \lambda x : A. \rrbracket f) \circ (\theta_y)_y = \text{curry} \left(\llbracket \Gamma \rrbracket \times \llbracket A \rrbracket \xrightarrow{(\theta_y)_y \times \text{id}} \llbracket \Delta \rrbracket \xrightarrow{f} \llbracket B \rrbracket \right)$$

It follows from the naturality curry . The compatibility axioms for \mathbf{ValOp} and \mathbf{AppOp} hold immediately.

We combine compatibility conditions from fragments to the whole CBV language (cf. §C.3 for the proof):

Lemma 6.3 *Let \mathbf{M} be a substitution structure and $(\mathbf{O}_i)_{i \in I}$ and $(\mathbf{[-]}_i : \mathbf{O}_i \mathbf{M} \rightarrow \underline{\mathbf{M}})_{i \in I}$ be families of \mathbf{R} -signature functors and algebras for them. The cotupled algebra: $[\mathbf{[-]}_i]_{i \in I} : \coprod_{i \in I} \mathbf{O}_i \mathbf{M} \rightarrow \underline{\mathbf{M}}$ is compatible with \mathbf{M} iff every algebra \mathbf{O}_i is compatible with \mathbf{M} .*

An \mathbf{O} -substitution structure *homomorphism* $h : \mathbf{M} \rightarrow \mathbf{N}$ is a morphism $h : \underline{\mathbf{M}} \rightarrow \underline{\mathbf{N}}$ that is both an \mathbf{O} -homomorphism and an action homomorphism in the sense of Fig 2. Let $\mathbf{H} \in \mathbf{R-Struct}$ be an \mathbf{R} -structure, whose elements we think of as holes. An \mathbf{O} -structure over \mathbf{H} is a pair $(\mathbf{M}, \mathbf{[-]}_{\text{meta}})$ consisting of a \mathbf{O} -structure \mathbf{M} and a morphism $\mathbf{?} : \mathbf{H} \rightarrow \underline{\mathbf{M}}$, which we will call the *metavariable interpretation map*. A morphism of \mathbf{O} -structures $h : (\mathbf{M}, \mathbf{?}) \rightarrow (\mathbf{N}, \mathbf{?})$ over \mathbf{H} is an \mathbf{O} -structure homomorphism $h : \mathbf{M} \rightarrow \mathbf{N}$ preserving the metavariable interpretation. We let $\mathbb{J} := \langle \mathbb{V}, \emptyset \rangle$ be the $\mathbf{R-Struct}$ -structure of variables: its first-class fragment is the presheaf of variables, and its second-class sets are empty.

Theorem 6.4 (special representation) *Let \mathbf{O} be an \mathbf{R} -signature functor, and \mathbf{H} an \mathbf{R} -structure. Consider any initial algebra $\mathbb{S}^{\mathbf{O}} \mathbf{H} = \langle \langle \mathbb{S}_{\text{fst}}^{\mathbf{O}} \mathbf{H}, \mathbb{S}_{\text{snd}}^{\mathbf{O}} \rangle \rangle := \mu X. (\mathbf{O} X) \amalg \mathbb{J} \amalg (\mathbf{H} \otimes X|_{\text{fst}})$ given by:*

$$\mathbf{[-]} : \mathbf{O}(\mathbb{S}^{\mathbf{O}} \mathbf{H}) \rightarrow \mathbb{S}^{\mathbf{O}} \mathbf{H} \quad \mathbf{var} : \mathbb{V} \rightarrow \mathbb{S}_{\text{fst}}^{\mathbf{O}} \mathbf{H} \quad \mathbf{?} -[-] : \mathbf{H} \otimes \mathbb{S}_{\text{fst}}^{\mathbf{O}} \mathbf{H} \rightarrow \mathbb{S}^{\mathbf{O}} \mathbf{H}$$

There is a unique morphism $-[-] : \$^0\mathbf{H} \otimes \$^0_{\text{fst}}\mathbf{H} \rightarrow \$^0\mathbf{H}$, called simultaneous substitution, satisfying:

$$\begin{array}{c}
 \begin{array}{ccc}
 \text{str}_{\$H, \text{var}}: \underline{O}(\$H \otimes \$_{\text{fst}}H) \xrightarrow{\quad} \underline{O}(-[-]) & & \mathbb{I} \otimes \$_{\text{fst}}H \xrightarrow{\quad} & \mathbf{H} \otimes (\$_{\text{fst}}H \otimes \$_{\text{fst}}H) \xrightarrow{\text{id} \otimes (-[-])} \mathbf{H} \otimes \$_{\text{fst}}H \\
 \begin{array}{c} \nearrow \text{op} \\ \text{case} \end{array} & \begin{array}{c} \nearrow \text{var} \otimes \text{id} \\ \text{case} \end{array} & \begin{array}{c} \nearrow \vartheta \\ \text{case} \end{array} & \begin{array}{c} \nearrow \mathbf{a} \\ \text{metavariable} \\ \text{case} \end{array} \\
 (\underline{O}H) \otimes \$_{\text{fst}}H \xrightarrow{\quad} \underline{O}H & \$_{\text{fst}}H \otimes \$_{\text{fst}}H \xrightarrow{\quad} \$_{\text{fst}}H & (\mathbf{H} \otimes \$_{\text{fst}}H) \otimes \$_{\text{fst}}H \xrightarrow{\quad} \mathbf{H} \otimes \$_{\text{fst}}H \\
 \begin{array}{c} \nearrow [-] \otimes \text{id} \\ \text{case} \end{array} & \begin{array}{c} \nearrow [-] \\ \text{case} \end{array} & \begin{array}{c} \nearrow (?[-]) \otimes \text{id} \\ \text{case} \end{array} & \begin{array}{c} \nearrow ?[-] \\ \text{case} \end{array} \\
 \$H \otimes \$_{\text{fst}}H \xrightarrow{-[-]} \$H & \$_{\text{fst}}H \otimes \$_{\text{fst}}H \xrightarrow{-[-]} \$_{\text{fst}}H & (\mathbf{H} \otimes \$_{\text{fst}}H) \otimes \$_{\text{fst}}H \xrightarrow{-[-]} \$H \otimes \$_{\text{fst}}H & \$H \otimes \$_{\text{fst}}H \xrightarrow{-[-]} \$H
 \end{array}
 \end{array}$$

Equipping $\mathbf{F}_0\mathbf{H} := (\$^0\mathbf{H}, -[-], \text{var}, \llbracket - \rrbracket)$ with $? : \mathbf{H} \xrightarrow{r'} \mathbf{H} \otimes \mathbb{V} \xrightarrow{\text{id} \otimes \text{var}} \mathbf{H} \otimes \$^0\mathbf{H} \xrightarrow{?-[-]} \$^0\mathbf{H}$, yields the free \mathbf{O} -substitution structure over \mathbf{H} . We call morphisms $\theta : \mathbf{H}_1 \rightarrow \$^0\mathbf{H}_2$ simple metavariable substitutions, and we call their Kleisli extension $-[\theta] : \$^0\mathbf{H}_1 \rightarrow \$^0\mathbf{H}_2$ metavariable substitution by θ .

This theorem lets us prove substitution lemmata wholesale (cf. Lemma 7.4), concluding the tutorial.

6.1 On pointed strengths and compatibility

The compatibility condition Def 6.1 is technical, involving all MAST components. It deserves to, but often does not, take the centre spot in accounts of this theory. In concrete examples, one can see how these components contribute to the modularity of the theory. Each syntactic construct, specified through a signature functor \mathbf{O}_i , carries its own strength. Each model exhibits a substitution structure, as well as an interpretation for each of these functors through an \mathbf{O}_i -algebra structure. Compatibility expresses a semantic condition on the interaction between semantic substitution and the semantics of each \mathbf{O}_i in isolation, without referring to the rest of the syntax. We will now give a few more results, brought about thanks to the action-based perspective, that also explain the abstract role strength and compatibility have.

First, we relate substitution actions with the pointed monoidal structure. The difference between monoids in a monoidal category \mathcal{V} and monoids for its associated pointed monoidal category \mathcal{V}^\bullet is the point of the monoid and the point-preservation of the unit and multiplication. This difference is irrelevant:

Proposition 6.5 *Let \mathcal{V} be a monoidal category. The forgetful functor $\underline{-} : \mathcal{V}^\bullet \rightarrow \mathcal{V}$ lifts to an isomorphism between their categories of monoids:*

$$\mathbf{Monoid} \mathcal{V}^\bullet \xrightleftharpoons[\underline{-}]{\quad} \mathbf{Monoid} \mathcal{V} \quad \underline{\mathbf{M}} := (\underline{\mathbf{M}}, \underline{-[-]_M}, \underline{\text{var}}) \quad \underline{h} := h \quad \mathbf{M}^\bullet := ((\mathbf{M}, \text{var}), -[-], \text{var}) \quad h^\bullet := h$$

Given a \mathcal{V} -actegory \mathcal{A} , the categories of \mathbf{M} -actions in \mathcal{A} and \mathbf{M}^\bullet -actions in \mathcal{A}_\bullet are also isomorphic.

The proof in §C.3 is straightforward. For monoids, we show that the pointed structure of a pointed monoid must be its unit, and validating the point preservation axioms. The proof for actions is immediate.

Next, we explicate the abstract role the pointed strength serves, proved by direct calculation (cf. §C.3):

Proposition 6.6 *Let \mathcal{V} be a monoidal category, \mathcal{A} be \mathcal{V} -actegory, and $\mathbf{O} : \mathcal{A}_\bullet \rightarrow \mathcal{A}_\bullet$ a pointed-strong functor. For every \mathcal{V} -monoid, \mathbf{O} lifts to the following functor over \mathbf{M} -actions:*

$$\mathbf{O}_M : \mathbf{M}\text{-Action } \mathcal{A} \rightarrow \mathbf{M}\text{-Action } \mathcal{A} \quad \mathbf{O}_M A := \underline{\mathbf{O}A} := \underline{\mathbf{O}A} \quad -[-]_{\mathbf{O}_M A} : \underline{\mathbf{O}A} \otimes \mathbf{M} \xrightarrow{\text{str}_{\Delta, \text{var}}} \underline{\mathbf{O}}(\underline{A} \otimes \underline{M}) \xrightarrow{-[-]_A} \underline{A}$$

Putting these two results together gives a new perspective on the pointed strength and on compatible algebras. The monoidal category of homogeneous structures **Fst-Struct** acts on the category of heterogeneous structures **R-Struct**, and so **(Fst-Struct)**[•] acts on **R-Struct**_•. Given a substitution structure **S**, the following process produces an **(Fst-Struct)**[•]-action on **S** in **R-Struct**_•. Moving to the first-, and second-, order fragments, we get a substitution action $(\mathbf{S}|_{\text{fst}}, \mathbf{S}|_{\text{snd}})$. The monoid $\mathbf{S}|_{\text{fst}}$ acts on itself and on $\mathbf{S}|_{\text{snd}}$,

and therefore the monoid $\mathbf{S|}_{\text{fst}}^*$ acts on both $\mathbf{S|}_{\text{fst}}$ in $\mathbf{Fst}\text{-}\mathbf{Struct}$, and $\mathbf{S|}_{\text{snd}}$ in $\mathbf{R}\text{-}\mathbf{Struct}_{\text{snd}}$. Pairing these up as gives an $\mathbf{S|}_{\text{fst}}^*$ -action on $(\mathbf{S|}_{\text{fst}}, \mathbf{S|}_{\text{snd}})$ in $\mathbf{Fst}\text{-}\mathbf{Struct} \times \mathbf{R}\text{-}\mathbf{Struct}_{\text{snd}}$. Combining them through the isomorphism $\langle\langle -, - \rangle\rangle : \mathbf{Fst}\text{-}\mathbf{Struct} \times \mathbf{R}\text{-}\mathbf{Struct}_{\text{snd}} \xrightarrow{\cong} \mathbf{R}\text{-}\mathbf{Struct}$ exhibits an $\mathbf{S|}_{\text{fst}}^*$ -action on \mathbf{S} in $\mathbf{R}\text{-}\mathbf{Struct}$. Let $\llbracket [-] : \mathbf{OS} \rightarrow \mathbf{S}$ be any \mathbf{O} -algebra structure. It is compatible precisely when it is an $\mathbf{S|}_{\text{fst}}^*$ -action homomorphism $\llbracket [-] : \mathbf{OS|}_{\text{fst}} \mathbf{S} \rightarrow \mathbf{S}$, recasting compatibility as substitution-preservation in a precise way.

7 Case study: the Call-by-Value λ -calculus

We use algebraic signatures to modularly describe extensions to the CBV type system, and signature functors to modularly describe extensions to its terms. Starting with a semantics based on strong monads, we extend a basic calculus with sequential composition, functions, products, coproducts, an inductive datatype of natural numbers, iteration, and recursion. These require 7 checks that each additional bunch of semantic definitions is well-defined and compatible with the substitution structures in the corresponding algebra. In return, the MAST theory lets us deduce $2^7 = 128$ different substitution lemmata, for each language fragment. Note that not all models can interpret each fragment, and so one cannot deduce the substitution lemma for a fragment from the lemma for the full language. The development is similar in spirit to Swierstra's à la carte methodology [85,41], but it also provides semantic substitution lemmata. We use this opportunity to also summarise the standard denotational semantics for these features.

7.1 The full calculus

Fig 1 presents the abstract syntax of all the features we will consider without explicating their typing judgements nor their binding structure. Fig 3 presents the typing judgements. Our base calculus includes a construct for sequencing, which evaluates the intermediate results in order, binding them to variables. Extending the calculus with records adds tuples of labelled fields, which we eliminate with a pattern matching construct. Extending the calculus with variants adds tagged sums, which we eliminate with a pattern matching construct. Extending the calculus with natural numbers adds natural number literals as values, the iso-recursive constructor **roll** and deconstructor **unroll**, and a bounded iteration eliminator. We further extend the calculus with unbounded iteration **for** $i = M$ **do** N , which initialises i to M , and then iterates N until it is **done**. Finally, we extend the calculus with higher-order recursion through the **let rec** construct. It extends the body's (N) context with n mutually-recursive functions f_1, \dots, f_n . Here we assume some predefined mapping from contexts Γ and tuple-types (Γ) , identifying each position $(x : A) \in \Gamma$ with a label x . For example, sending its position to its corresponding numeral.

7.2 Simple types à la carte

To develop these calculi and their models fully modularly, we first need to treat their sets of types modularly. We will use the classical à la carte methodology, for initial algebra semantics in \mathbf{Set} , to mix and choose the collection of simple types we work with in each case. We will typically work with respect to an ordinary signature functor $\mathbf{L} : \mathbf{Set} \rightarrow \mathbf{Set}$ that has an initial algebra $\mathbf{L}(\mu\mathbf{L}) \rightarrow \mu\mathbf{L}$. This functor specifies the signature for the simple types given by $\mathbf{Type}_{\mathbf{L}} := \mu\mathbf{L}$. For any set \mathbf{Type} , whether inductively given by such a signature functor or not, define the sorting system $\mathbf{CBV}_{\mathbf{Type}}$ as the coproduct diagram:

$$\mathbf{Fst}_{\mathbf{CBV}_{\mathbf{Type}}} := \mathbf{Type} \subseteq \mathbf{Sort}_{\mathbf{CBV}_{\mathbf{Type}}} := \{A, \text{comp } A \mid A \in \mathbf{Type}\} \xleftarrow{\text{comp}} \mathbf{Type} =: \mathbf{Snd}_{\mathbf{CBV}_{\mathbf{Type}}}$$

We will also work with respect to a MAST $\mathbf{CBV}_{\mathbf{Type}}$ -signature functor \mathbf{O} , and define an \mathbf{O} -monoid, which by the Special Representation Thm 6.4 satisfies the Substitution Lemma 7.4.

Example 7.1 In the simplest case, all we have are base types. The signature functor for types is the constant functor $\underline{\mathbf{Base}} : \mathbf{Set} \rightarrow \mathbf{Set}$, $\underline{\mathbf{Base}} X := \mathbf{Base}$. It has the identity function $\mathbf{id} : \mathbf{Base} \rightarrow \mathbf{Base}$ as its initial algebra, thus $\mathbf{Base} = \mu \underline{\mathbf{Base}} =: \mathbf{Type}$. Summarising, the set of types is the set of base types.

$$\begin{array}{c}
\frac{(x:A) \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash V : A}{\Gamma \vdash \text{val } V : \text{comp } A} \quad \frac{\Gamma, x_1 : A_1, \dots, x_n : A_n \vdash N : \text{comp } B \quad \text{for all } i < n : \Gamma, x_1 : A_1, \dots, x_i : A_i \vdash M_{i+1} : \text{comp } A_{i+1}}{\Gamma \vdash \text{let } x_1 = M_1; \dots; x_n = M_n \text{ in } N : \text{comp } B} \\
\\
\frac{\Gamma, x : A \vdash M : \text{comp } B}{\Gamma \vdash \lambda x : A. M : A \rightarrow B} \quad \frac{\Gamma \vdash M : \text{comp } (A \rightarrow B) \quad \Gamma \vdash N : \text{comp } A}{\Gamma \vdash M @ N : \text{comp } B} \\
\\
\frac{\text{for all } 1 \leq i \leq n : \Gamma \vdash M_i : A_i}{\Gamma \vdash (C_1 : M_1, \dots, C_n : M_n) : \text{comp } (C_1 : A_1, \dots, C_n : A_n)} \quad \frac{\Gamma \vdash M : \text{comp } (C_1 : A_1, \dots, C_n : A_n) \quad \Gamma, x_1 : A_1, \dots, x_n : A_n \vdash N : \text{comp } B}{\Gamma \vdash \text{case } M \text{ of } (C_1 x_1, \dots, C_n x_n) \Rightarrow N : \text{comp } B} \\
\\
\frac{A = \llbracket C_i : A_i \mid i \in I \rrbracket \quad \Gamma \vdash M : \text{comp } A_i}{\Gamma \vdash A. C_i M : \text{comp } A} \quad \frac{\Gamma \vdash M : \llbracket C_i : A_i \mid i \in I \rrbracket \quad \text{for all } 1 \leq i \leq n : \Gamma, x_i : A_i \vdash M_i : \text{comp } B}{\Gamma \vdash \text{case } M \text{ of } \{ C_i x_i \Rightarrow M_i \mid i \in I \} N : \text{comp } B} \\
\\
\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \text{unroll } M : \llbracket 0 : (_) \text{, } (1+) : \mathbb{N} \rrbracket} \quad \frac{\Gamma \vdash M : \text{comp } \llbracket 0 : (_), (1+) : \mathbb{N} \rrbracket}{\Gamma \vdash \text{roll } M : \text{comp } \mathbb{N}} \quad \frac{\Gamma \vdash M : \text{comp } \mathbb{N} \quad \Gamma, x : \llbracket 0 : (_), (1+) : A \rrbracket \vdash N : \text{comp } A}{\Gamma \vdash \text{fold } M \text{ by } \{ x \Rightarrow N \} : \text{comp } A} \\
\\
\frac{\Gamma \vdash M : \text{comp } A \quad \Gamma, i : A \vdash N : \text{comp } \llbracket \text{done} : B, \text{continue} : A \rrbracket}{\Gamma \vdash \text{for } i = M \text{ do } N : \text{comp } B} \quad \frac{\overbrace{\Gamma, f_1 : (G_1) \rightarrow A_1, \dots, f_n : (G_n) \rightarrow A_n}^{\Delta \text{ is}} \vdash N : B \quad \text{for all } 1 \leq i \leq n : \Delta + \Gamma_n \vdash M_n : A_n}{\Gamma \vdash \text{let } \text{rec } f_1 \Gamma_1 : A_1 = M_1; \dots; f_n \Gamma_n : A_n = M_n \text{ in } N : B}
\end{array}$$

Fig. 3. Type system of CBV, omitting analogous rules for value records and variants.

Example 7.2 To accommodate function types, take the functor $\text{FunTy} := X \mapsto X \times X : \text{Set} \rightarrow \text{Set}$. Then:

$$\underline{\text{Base}} \amalg \text{FunTy } X \cong \{\beta \mid \beta \in \text{Base}\} \amalg \{(\rightarrow) : X \times X\}$$

Its initial algebra is Type , with $\llbracket - \rrbracket : (\underline{\text{Base}} \amalg \text{FunTy})\text{Type} \rightarrow \text{Type}$ where $\llbracket \beta \rrbracket := \beta$; $\llbracket (\rightarrow) \rrbracket (A, B) := A \rightarrow B$.

Example 7.3 Next, we deal with records and variants uniformly. Let X be a set. A *row* in X is a function $(C_i \mapsto x_i)_{i \in I}$ from a finite set of *field/constructor labels* $\{C_i \mid i \in I\}$ to X . Letting Label be the set of field labels, define $\text{Row} : \text{Set} \rightarrow \text{Set}$ by $\text{Row } X := \coprod_{I \subseteq \text{finLabel}} X^I$. The functors for record and variant types are: RecordTy , $\text{VariantTy} := \text{Row} : \text{Set} \rightarrow \text{Set}$.

The collection of types $\text{CBVType}_{\text{Base}}$ defined inductively in Fig 1 is the initial algebra for the functor:

$$\text{FullCBV} := \underline{\text{Base}} \amalg ((\rightarrow) : \text{FunTy}) \amalg ((-) : \text{RecordTy}) \amalg (\llbracket - \rrbracket : \text{VariantTy}) \amalg (\mathbb{N} : \text{NatTy}) : \text{Set} \rightarrow \text{Set}$$

We want to work with more than $2^4 = 16$ collections of types, given by various restrictions of this collection, depending on the typing needs of each language fragment. As a running example, consider the term former for higher-order recursive definitions. For each function signature $f[x_1 : A_1, \dots, x_n : A_n] : B$, we will need to identify the function type $(x_1 : A_1, \dots, x_n : A_n) \rightarrow B$. In fragments that contain both records and all function types, this construct will need to identify the compound type $(x_1 : A_1, \dots, x_n : A_n) \rightarrow B$. But in fragments that only contain higher-order recursion without records, we will fuse function application with record creation. To allow such flexibility, we use the following concepts. We define a *typing need* to be a signature functor $\mathbf{R} : \text{Set} \rightarrow \text{Set}$. For higher-order recursion, we use the typing need $\text{RecNeed } X := X \vdash \times X$. A *fulfillment* of a typing need \mathbf{R} in a set Type is then a relation $(\models) : \mathbf{R}(\text{Type}) \rightarrow \text{Type}$, which we will write as $k \models A : \mathbf{R}$ for every $k \in \mathbf{R}(\text{Type})$ and $A \in \text{Type}$ satisfying $(k, A) \in (\models)$. A fulfillment explains which types satisfy the typing need. For example, the fulfillment of the typing need for higher-order recursion RecNeed in the set of types for the full calculus is given by the relation defined by:

$$([x_1 : A_1, \dots, x_n : A_n], B) \models ((x_1 : A_1, \dots, x_n : A_n) \rightarrow B) : \text{RecNeed} \quad (A_1, \dots, A_n, B \in \text{CBVType}_{\text{Base}})$$

Note that we apply two type constructors in this fulfillment: the function type constructor (\rightarrow) and the record type constructor $(-)$. In the fulfillment for fragments that include higher-order recursion but not

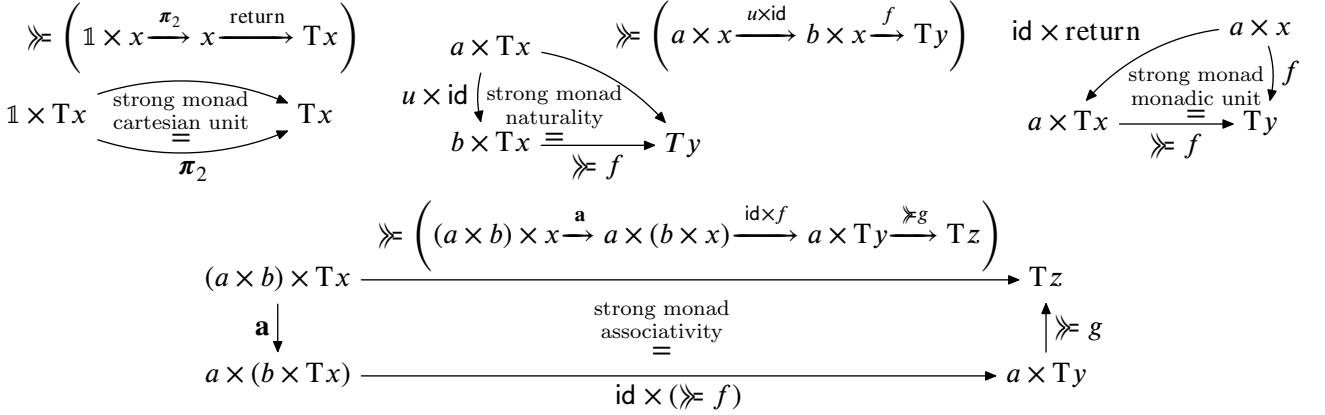


Fig. 4. Strong monad laws

records, we will fulfill this need by the single type-constructor for n -ary functions ($\text{(-)} \rightarrow \text{-}$) that these fragments include. We will use the following typing needs, and only them, in our case study:

- When we need the unit type, we impose the typing need $\text{EmptyRecord} := 1 := \{\star\}$, fulfilled by the empty record type, when the fragment includes records, and by the unit type otherwise: $\star \models \text{(-)} : \text{EmptyRecord}$.
- When we need to deconstruct natural numbers, we impose the typing need:

$$\text{NatConstVariant} := (\mathbb{N} : 1) \amalg (\text{Maybe} : \text{Id})$$

We will fulfill it in fragments with the natural numbers and bounded iteration over them by specifying the natural number type and an option type for each type:

$$\mathbb{N} \star \models \mathbb{N} : \text{NatConstVariant} \quad \text{Maybe} A \models \{0 : \text{(-)}, (1+) : A\} : \text{NatConstVariant} \quad (A \in \text{Type})$$

When the fragment doesn't include all variant or record types, we will ensure it includes the single type-constructor $\{0 : \text{(-)}, (1+) : \text{-}\}$.

- To type the bodies of while-loops, we impose the need $\text{NatConstVariant} := (\text{done} : \text{Id}, \text{continue} : \text{Id})$ fulfilled by a binary variant type:

$$(\text{done} : A, \text{continue} : B) \models \{\text{done} : A, \text{continue} : B\} : \text{NatConstVariant} \quad (A, B \in \text{Type})$$

- To type recursive function definitions, we impose $\text{RecNeed} X := X \times X$, our running example.

7.3 The substitution structures

Let Type be a set whose elements represent types. A *strong-monad model* $(\mathcal{C}, \llbracket - \rrbracket, T)$ consists of:

- A locally-small Cartesian category \mathcal{C} with chosen finite products.
- An interpretation function $\llbracket - \rrbracket : \text{Type} \rightarrow \mathcal{C}$.
- A strong monad [61] T over \mathcal{C} , i.e., an assignment of:
 - an object Tx to every $x \in \mathcal{C}$;
 - a morphism $\text{return}_x : x \rightarrow Tx$ to every $x \in \mathcal{C}$;
 - a morphism $\approx_{a,x,y} f : a \times Tx \rightarrow Ty$ to every $a, x, y \in \mathcal{C}$ and $f : a \times x \rightarrow Ty$;

satisfying the four equations [70] in Fig 4, w.r.t the cartesian monoidal structure $(\mathcal{C}, (\times), 1, \mathbf{a}, \ell, \mathbf{r})$.

Each such strong-monad model induces a substitution structure \mathbf{M} in CBV_{Type} -structures. First, let:

$$\llbracket \Gamma \rrbracket := \llbracket - \rrbracket_{\Gamma}^{\text{Env}} := \prod_{(x:A) \in \Gamma} \llbracket A \rrbracket \quad \llbracket \Gamma \xrightarrow{\rho} \Delta \rrbracket := \llbracket - \rrbracket_{\rho}^{\text{Env}} : \llbracket \Gamma \rrbracket \xrightarrow{(\pi_y)_{(y:B) \in \Delta}} \llbracket \Delta \rrbracket$$

I.e., interpret contexts as products and their renamings as tupled projections. The CBV_{Type} -structure \mathbf{M} :

$$\llbracket \text{comp } A \rrbracket := \text{T} \llbracket A \rrbracket \quad \underline{\mathbf{M}}_s \Gamma := \mathcal{C}(\llbracket \Gamma \rrbracket, \llbracket s \rrbracket) \quad \underline{\mathbf{M}}_s (\Gamma \xleftarrow{\rho} \Delta) : \left(\llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket s \rrbracket \right) \mapsto \left(\llbracket \Delta \rrbracket \xrightarrow{\llbracket \rho \rrbracket} \llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket s \rrbracket \right)$$

I.e., $\underline{\mathbf{M}}_A \Gamma := \mathcal{C}(\llbracket \Gamma \rrbracket, \llbracket A \rrbracket)$ and $\underline{\mathbf{M}}_{\text{comp}_A} \Gamma := \mathcal{C}(\llbracket \Gamma \rrbracket, \text{T} \llbracket A \rrbracket)$ so the semantics of computations are Kleisli arrows for the monad T . The monoid's unit interprets variables by projecting the appropriate component:

$$\text{var} : \mathbb{V} \rightarrow \underline{\mathbf{M}}|_{\text{fst}} \quad \text{var}_{B,\Gamma} y := \left(\llbracket \Gamma \rrbracket = \left(\prod_{(x:A) \in \Gamma} \llbracket A \rrbracket \right) \xrightarrow{\pi_y} \llbracket B \rrbracket \right)$$

To define substitution, we use the isomorphism which internalises tupling:

$$\overline{(-)} := (\pi_y \circ (-))_{(y:B) \in \Delta} : \mathcal{C}(\llbracket \Gamma \rrbracket, \llbracket \Delta \rrbracket) \xrightarrow{\cong} \left(\prod_{(y:B) \in \Delta} \mathcal{C}(\llbracket \Gamma \rrbracket, \llbracket B \rrbracket) \right) = \underline{\mathbf{M}}_{\Delta}^{\text{Env}} \Gamma$$

We express the functorial action of $\underline{\mathbf{M}}^{\text{Env}}$ through this isomorphism as follows:

$$\underline{\mathbf{M}}_{\Delta_1 \xrightarrow{\rho} \Delta_2, \Gamma}^{\text{Env}} : \left(\overline{\llbracket \Gamma \rrbracket \xrightarrow{\theta} \llbracket \Delta_1 \rrbracket} \right) \mapsto \left(\overline{\llbracket \Gamma \rrbracket \xrightarrow{\theta} \llbracket \Delta_1 \rrbracket \xrightarrow{\llbracket \rho \rrbracket} \llbracket \Delta_2 \rrbracket} \right)$$

We then define substitution $-[-]_{\mathbf{M}} : \mathbf{M} \otimes \mathbf{M} \rightarrow \mathbf{M}$ by pre-composition:

$$\left(\llbracket \Delta \rrbracket \xrightarrow{f} \llbracket s \rrbracket \right) \left[\overline{\llbracket \Gamma \rrbracket \xrightarrow{\theta} \llbracket \Delta \rrbracket} \right]_{\mathbf{M}, s, \Gamma} := \left(\llbracket \Gamma \rrbracket \xrightarrow{\theta} \llbracket \Delta \rrbracket \xrightarrow{f} \llbracket s \rrbracket \right)$$

Appendix A details the proof that this definition forms a substitution structure.

7.4 The CBV customisation menu

We will now consider some fragments of the full CBV calculus. Fig 5 list the constructs in each fragment, which fragments of the type system they require, and what model structure they need. We will treat the base, sequential, and functional fragments. The other fragments admit a similar treatment. We define each fragment, and then explain how to combine fragments together into one calculus and its model class of interest. Together, these combinations describe $2^7 = 128$ different calculi, and their denotational semantics. Thanks to MAST and the Special Representation Thm 6.4, the substitution operation and denotational semantics for each combination satisfy a substitution lemma. Formally, let \mathbf{Ext} be the set of 7 extensions listed in the ‘name’ column. For each fragment $\varepsilon \subseteq \mathbf{Ext}$, we specify:

- a simple signature $\mathbf{L}_{\varepsilon} : \mathbf{Set} \rightarrow \mathbf{Set}$, which we define á la carte as $\mathbf{L}_{\varepsilon} := \underline{\mathbf{Base}} \amalg \coprod_{\text{ext} \in \varepsilon} \mathbf{L}_{\varepsilon}^{\text{ext}}$, inducing the set $\mathbf{Type}_{\varepsilon} := \mu \mathbf{L}_{\varepsilon}$ and sorting system $\text{CBV}_{\varepsilon} := \text{CBV}_{\text{Type}_{\varepsilon}}$;
- a fulfillment $(\mathbf{F}_{\varepsilon}^{\text{ext}}) : \mathbf{R}_{\text{ext}} \mathbf{Type}_{\varepsilon} \nrightarrow \mathbf{Type}_{\varepsilon}$ for each typing need $\mathbf{R}_{\text{ext}} : \mathbf{Set} \rightarrow \mathbf{Set}$ and extension $\text{ext} \in \varepsilon$;
- a CBV_{Type}-signature functor $\mathbf{O}_{\varepsilon} := \underline{\mathbf{Base}} \amalg \coprod_{\text{ext} \in \varepsilon} \mathbf{O}_{\varepsilon}^{\text{ext}}$ describing the syntactic constructs in this fragment;
- additional structure or properties we require of the substitution structure \mathbf{M} for a strong-monad model $(\mathcal{C}, \llbracket - \rrbracket, \text{T})$. These requirements impose structure and properties of the type interpretation function $\llbracket - \rrbracket : \mathbf{Type}_{\varepsilon} \rightarrow \mathcal{C}$, typically via universal properties involving categorical structures over the model.

name	syntactic constructs	typing needs	additional model needs
base	returning a value: <code>val</code>		strong monad over a Cartesian category
sequential functions	sequencing: <code>let</code> abstraction and application $(\lambda x. : A), (@)$	function (\rightarrow)	Kleisli exponentials
records	constructors and pattern match $(C_1 : -, \dots, C_n : -)$ <code>case - of</code> $(C_1 x_1, \dots, C_n x_n) \Rightarrow -$	record $\{C_i : - \mid i \in I\}$	
variants	constructors and pattern match $A.C_i -$, <code>case - of</code> $\{C_i x_i \Rightarrow - \mid i \in I\}$	variant $\{C_i : - \mid i \in I\}$	distributive category
natural numbers	the zero and successor constructors, literals, empty record, (de)constructors, and bounded iteration, the pattern matching $0, (1+), n, (_)$, <code>unroll</code> , <code>roll</code> <code>fold - by</code> $\{x \Rightarrow -\}$ <code>case - of</code> $\{0 \Rightarrow -, 1+x \Rightarrow -\}$	naturals, empty record, and the variants \mathbb{N} $\{0 : (_),\}$ $\{(1+) : -\}$	binary coproducts distributed over by the products, and a natural numbers object
while	the constructors, unbounded iteration <code>done</code> , <code>continue</code> , <code>for</code> $i = -$ <code>do</code> $-$	the variants $\{done : -,$ $\{continue : -\}$	binary coproducts, distributive products, and the monad has a complete Elgot structure [5,4,3,29,16,48]
recursion	relevant record constructor, function application, recursion $(-) , (@), let rec$	the functions $(\{x_i : - \mid i \in I\} \rightarrow)$	uniform parameterised monadic fixed-points [51,81], Kleisli exponentials

Fig. 5. A customisation menu of CBV fragments

$$\begin{aligned}
\text{str}_{P,A,\Gamma}^{\mathbf{O}_\epsilon^{\text{seq}}} &= \left[\begin{array}{l} \text{let } x_0 : A_0 = (p_0 \in P_{\text{comp } A_0} \Delta) \\ x_1 : A_1 = (p_1 \in P_{\text{comp } A_1} (\Delta, x_0 : A_0)) \\ \vdots \\ x_n : A_n = (p_n \in P_{\text{comp } A_n} (\Delta, x_0 : A_0, \dots, x_{n-1} : A_{n-1})) \\ \text{in } (q \in P_{\text{comp } B} (\Delta, x_0 : A_0, \dots, x_n : A_n)), \theta \in A_\Delta^{\text{Env } \Gamma} \end{array} \right]_\Delta \\
&:= \left(\begin{array}{l} \text{let } x_0 : A_0 = [p_0, \theta]_\Delta \\ x_1 : A_1 = [p_1, \theta \# (x_0 : \text{var}_A x_0)]_{\Delta, x_0 : A_0} \\ \vdots \\ x_n : A_n = [p_n, \theta \# (x_0 : \text{var}_A x_0, \dots, x_{n-1} : \text{var}_A x_{n-1})]_{\Delta, x_0 : A_0, \dots, x_{n-1} : A_{n-1}} \\ \text{in } [q \in P_{\text{comp } B}, \theta \# (x_0 : \text{var}_A x_0, \dots, x_n : \text{var}_A x_n)]_{\Delta, x_0 : A_0, \dots, x_n : A_n} \end{array} \right)
\end{aligned}$$

Fig. 6. Pointed strength for the sequential signature functor

We then fix such a substitution structure \mathbf{M} , and further specify:

- a compatible \mathbf{O} -algebra for \mathbf{M} making it an \mathbf{O} -structure.

Base fragment

All our fragments include the base fragment: the signature functor \mathbf{L}_ϵ includes the set of base types and so Type_ϵ includes them. There are no typing need for the base fragment, and it contributes no tuples to the fulfillment relation. The base calculus has, for each type $A \in \text{Type}_\epsilon$, one operator coercing A -values to A -computations. Its binding signature functor and its derived strength are:

$$\text{Base } X := \coprod_{A \in \text{Type}_\epsilon} (\text{val}_A : \underset{\text{comp } A}{\rightsquigarrow} (X @ A)) \quad \text{str}_{P,A,\Gamma}^{\text{Base}} [\text{val}_A (p \in P_A \Delta), \theta \in A_\Delta^{\text{Env } \Gamma}] := \text{val}_A [p, \theta]$$

The base requirement of a model is for it to be a strong-monad model $\mathbf{M} := (\mathcal{C}, \llbracket - \rrbracket, \text{T})$. Define its Base -algebra structure as follows, and see calculation (A.1) in the appendix for the compatibility condition:

$$\llbracket \text{val}_A \llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket A \rrbracket \rrbracket_{\mathbf{M}} := \left(\llbracket \Gamma \rrbracket \xrightarrow{f} \llbracket A \rrbracket \xrightarrow{\text{return}} \text{T} \llbracket A \rrbracket \right)$$

Sequential fragment

This fragment does not extend the language with new types nor does it impose any typing requirements, and so $\mathbf{L}_\epsilon^{\text{seq}} := \mathbf{R}_\epsilon^{\text{seq}} := \emptyset$. The fulfillment relation is then the empty relation $(\mathbb{F}_{\text{frag}}^{\text{seq}}) : \mathbf{R}_\epsilon^{\text{seq}} \text{Type}_\epsilon := \emptyset \rightsquigarrow \text{Type}_\epsilon$.

The sequential fragment has, for every non-empty context $[x_0 : A_0, \dots, x_n : A_n] \in (\text{Type}_\epsilon)_\vdash$ and type $B \in \text{Type}_\epsilon$, one sequencing operator $\text{let } x_0 = M_0; \dots; x_n = M_n \text{ in } N$:

$$\mathbf{O}_\epsilon^{\text{seq}} X := \coprod_{n \in \mathbb{N}} \coprod_{[x_0 : A_0, \dots, x_n : A_n] \in (\text{Type}_\epsilon)_\vdash} \coprod_{B \in \text{Type}_\epsilon} \left(\begin{array}{l} (\text{let } x_0 : A_0 = _ ; \dots ; x_n : A_n = _ \text{ in } _ : B) : \\ \underset{\text{comp } B}{\rightsquigarrow} \left(\begin{array}{l} \left(\prod_{i=0}^n [x_0 : A_1, \dots, x_{i-1} : A_{i-1}] \triangleright X @ \text{comp } A_i \right) \\ \times ([x_0 : A_1, \dots, x_n : A_n] \triangleright X @ \text{comp } B) \end{array} \right) \end{array} \right)$$

with its induced tensorial strength given in Fig 6. Given a strong-monad model $\mathbf{M} := (\mathcal{C}, \llbracket - \rrbracket, \text{T})$, we require no additional semantic structure of it. We use the monadic bind to interpret the let construct, and to ease dealing with the intermediate results bound to x_1, \dots, x_n , we use the following derived semantic

$$\begin{aligned} \mathbf{str}_{P,C,A \rightarrow B,\Gamma}^{\mathbf{O}_\epsilon^{\mathbf{fun}}} [\lambda x : A. (p \in P_{\mathbf{comp} B}(\Delta, x : A)), \theta \in C_\Delta^{\mathbf{Env}} \Gamma]_\Delta &:= \lambda x : A. [p, (\theta, x : \mathbf{var} x)]_{\Delta, x : A} \\ \mathbf{str}_{P,C,B,\Gamma}^{\mathbf{Fun}} [(p \in P_{\mathbf{comp} A \rightarrow B} \Delta) @ (q \in P_{\mathbf{comp} A} \Delta), \theta \in C_\Delta^{\mathbf{Env}} \Gamma]_\Delta &:= [p, \theta]_\Delta @ [q, \theta]_\Delta \end{aligned}$$

Fig. 7. Pointed strength for the functional-fragment signature functor

structure. Consider the *Cartesian strength* of the monad:

$$\mathbf{str}_{a,b} : a \times \mathbf{T} b \xrightarrow{\mathbf{F}_{a,b,a \times b} \mathbf{id}} \mathbf{T}(a \times b)$$

We use it to define, for every $f : a \times x \rightarrow \mathbf{Ty}$ a morphism that keeps the intermediate result:

$$\ll f : a \times x \xrightarrow{(\pi_2, f)} x \times \mathbf{Ty} \xrightarrow{\mathbf{str}} \mathbf{T}(x \times y) \quad \mathbf{F} \ll f : a \times \mathbf{T} x \xrightarrow{\mathbf{F}(\ll f)} \mathbf{T}(x \times y)$$

Define the $\mathbf{O}_\epsilon^{\mathbf{seq}}$ -algebra structure as follows. Given A_0, \dots, A_n, B , let:

$$\Delta := [x_0 : A_0, \dots, x_n : A_n] \quad \Delta_i := [x_0 : A_0, \dots, x_{i-1} : A_{i-1}] \quad \text{for all } i = 1, \dots, n$$

and then define, suppressing canonical isomorphisms such as $\Delta_i \times A_i \cong \Delta_{i+1}$:

$$\left[\begin{array}{l} \mathbf{let} x_0 : A_0 = \llbracket \Gamma \rrbracket \xrightarrow{f_0} \mathbf{T} \llbracket A_0 \rrbracket \\ x_1 : A_1 = \llbracket \Gamma \rrbracket \times \llbracket \Delta_1 \rrbracket \xrightarrow{f_1} \mathbf{T} \llbracket A_1 \rrbracket \\ \vdots \\ x_n : A_n = \llbracket \Gamma \rrbracket \times \llbracket \Delta_n \rrbracket \xrightarrow{f_n} \mathbf{T} \llbracket A_n \rrbracket \mathbf{in} \llbracket \Gamma \# \Delta \rrbracket \xrightarrow{g} \mathbf{T} \llbracket B \rrbracket : B \end{array} \right] := \\ \llbracket \Gamma \rrbracket \xrightarrow{(\mathbf{id}, f_0)} \llbracket \Gamma \rrbracket \times \mathbf{T} \llbracket \Delta_1 \rrbracket \xrightarrow{(\mathbf{id}, \ll f_1)} \llbracket \Gamma \rrbracket \times \mathbf{T} \llbracket \Delta_2 \rrbracket \rightarrow \dots \rightarrow \llbracket \Gamma \rrbracket \times \mathbf{T} \llbracket \Delta_n \rrbracket \xrightarrow{(\mathbf{id}, \ll f_n)} \llbracket \Gamma \rrbracket \times \mathbf{T} \llbracket \Delta \rrbracket \xrightarrow{\mathbf{F} g} \mathbf{T} \llbracket B \rrbracket$$

We prove that this $\mathbf{O}_\epsilon^{\mathbf{seq}}$ -algebra is compatible with substitution in §A.3.

Functional fragment

For this fragment we extend the set of types—á la carte—using the following simple signature functor for this fragment to add function types: $\mathbf{L}_\epsilon^{\mathbf{fun}} X := \{x \rightarrow y \mid x, y \in X\} \cong X \times X$. We impose no typing requirements for this extension ($\mathbf{R}_{\mathbf{fun}} = \emptyset$) and the fulfillment relation for these fragments is empty ($(\mathbf{F}_\epsilon^{\mathbf{fun}}) : \mathbf{R}_{\mathbf{fun}} \mathbf{Type}_{frag} = \emptyset \nrightarrow \mathbf{Type}_\epsilon$). These fragments extend the base language with function abstraction and application, which we add using the following signature functor:

$$\mathbf{O}_\epsilon^{\mathbf{fun}} X := \coprod_{A, B \in \mathbf{Type}_L} \left((\lambda x : A. : \underset{A \rightarrow B}{\Downarrow} [x : A] \triangleright X @ \mathbf{comp} B) \amalg ((@) : \underset{B}{\Downarrow} (X @ \mathbf{comp}(A \rightarrow B)) \times (X @ \mathbf{comp} A)) \right)$$

Its derived strength is given in Fig 7. We require models \mathbf{M} for fragments with $\mathbf{fun} \in \epsilon$ must be equipped with a choice of *Klesli exponentials* $((\mathbf{Ty})^x, \mathbf{eval} : (\mathbf{Ty})^x \times x \rightarrow \mathbf{Ty})$, for every pair of objects $x, y \in \mathcal{C}$. Define the $\mathbf{O}_\epsilon^{\mathbf{fun}}$ -algebra structure by:

$$\begin{aligned} \llbracket \lambda x : A. \left(\llbracket \Gamma \rrbracket \times \llbracket A \rrbracket \xrightarrow{f} \mathbf{T} \llbracket B \rrbracket \right) \rrbracket &:= \left(\llbracket \Gamma \rrbracket \xrightarrow{\mathbf{curry} f} \llbracket A \rightarrow B \rrbracket \right) \\ \llbracket \left(\llbracket \Gamma \rrbracket \xrightarrow{f} \mathbf{T} \llbracket A \rightarrow B \rrbracket \right) @ \left(\llbracket \Gamma \rrbracket \xrightarrow{a} \mathbf{T} \llbracket A \rrbracket \right) \rrbracket &:= \\ \llbracket \Gamma \rrbracket \xrightarrow{(\mathbf{id}, f)} \llbracket \Gamma \rrbracket \times \mathbf{T} \llbracket A \rightarrow B \rrbracket &\xrightarrow{\mathbf{F} \left(\llbracket \Gamma \rrbracket \times \llbracket A \rightarrow B \rrbracket \xrightarrow{\pi_1} \llbracket \Gamma \rrbracket \xrightarrow{a} \mathbf{T} \llbracket A \rrbracket \right)} \mathbf{T} (\llbracket A \rightarrow B \rrbracket \times \llbracket A \rrbracket) \xrightarrow{\mathbf{F} \mathbf{eval}} \mathbf{T} \llbracket B \rrbracket \end{aligned}$$

We prove that this $\mathbf{O}_\epsilon^{\text{seq}}$ -algebra is compatible with substitution in §A.4.

We demonstrated the various moving parts in using MAST to define syntax and semantics à la carte. The other fragments follow a similar treatment, defining signature functors **Record**, **Variant**, **Nat**, **While**, and **Rec**, each with their typing needs and model structure and properties, as in Fig 5. We omit these details. We conclude by reaping the fruit of our labour: the standard substitution lemma for denotational semantics. While substitution lemmata are not hard to prove, they are tedious to establish formally. The Special Representation Thm 6.4 justifies omitting them from most technical developments:

Lemma 7.4 (substitution) *For every term $\Delta \vdash M : A$ and substitution $(\Gamma \vdash \theta_y : B)_{(y:B) \in \Delta}$, we have:*

$$\llbracket M[\theta] \rrbracket = \llbracket M \rrbracket \circ (\llbracket \theta_y \rrbracket)_y$$

Proof. By the homomorphism property of the denotational semantics:

$$\llbracket M[\theta] \rrbracket = \llbracket (-[-])[M, \theta]_\Delta \rrbracket = (-[-]) \left[\llbracket M \rrbracket, (\llbracket \theta_y \rrbracket)_{y \in \Delta} \right]_\Delta = \llbracket M \rrbracket \circ (\llbracket \theta_y \rrbracket)_y \quad \square$$

8 Technical development outline

Our development is relatively straightforward thanks to several abstractions: bi-categories and the right-closed actegorical structure of substitution (§8.1) and the General Representation Thm 8.3 (§8.2) which abstracts from the concrete details of presheaf categories and their tensors. We also relate this development to the pre-proceedings manuscript, which used *skew* monoidal structures (§8.3). In this section, we summarise how these abstract components intertwine, and relegate the remaining details to Appendices B–D.

8.1 Bicategorical development

Fiore, Gambino, Hyland, and Winskel [35] package the sophistication involved in the classical substitution tensor product in a bicategory we denote by **Prof**\$_\vdash\$. Its vertices/0-cells are small categories $\mathbb{A}, \mathbb{B}, \mathbb{C}, \dots$. The category S_\vdash of S -sorted contexts is the finite-product completion of S , and extends to categories. The arrows/1-cells $P : \mathbb{A} \nrightarrow \mathbb{B}$ in **Prof**\$_\vdash\$ are Kleisli profunctors $P : \mathbb{A} \rightarrow \mathbb{B}_\vdash$, i.e. presheaves $P \in \mathbf{PSh}(\mathbb{A}^{\text{op}} \times \mathbb{B}_\vdash)$, equivalently functors $P : \mathbb{A} \times \mathbb{B}_\vdash^{\text{op}} \rightarrow \mathbf{Set}$. Its faces/2-cells $\alpha : P \Rightarrow Q$ are natural transformations, with their usual vertical and horizontal composition. Arrows $P : \mathbb{A} \nrightarrow \mathbb{B}$ and $Q : \mathbb{B} \nrightarrow \mathbb{C}$ compose diagrammatically using the same formula as the substitution tensor product, and the profunctor of variables, suitably generalised, is the identity $\mathbb{V} : \mathbb{A} \nrightarrow \mathbb{A}$. One recovers the classical setting by restricting to the full sub-bicategory over a set of sorts \mathbf{R} , since a one vertex bicategory is a monoidal category.

Proposition 8.1 *Let \mathcal{B} be a bicategory. Every two 0-cells $\mathbb{A}, \mathbb{B} \in \mathcal{B}$ induce an actegory $(\mathcal{B}(\mathbb{A}, \mathbb{A}), \mathcal{B}(\mathbb{B}, \mathbb{A}))$. Its monoidal category is given by the endo-1-cells $X : \mathbb{A} \rightarrow \mathbb{A}$ and their 2-cells. It acts on the 1-cells $P : \mathbb{B} \rightarrow \mathbb{A}$ and their 2-cells through 1-cell post-composition and horizontal composition.*

The proof is in §B.2. We obtain Thm 4.2 from Prop. 8.1 by taking $\mathcal{B} := \mathbf{Prof}_\vdash$, $\mathbb{A} := \mathsf{Fst}$, and $\mathbb{B} := \mathsf{Snd}$.

8.2 The representation theorem

Adapting the classical proof for the representation theorem is straightforward thanks to its level of generality. Let $(\mathcal{V}, \mathcal{A})$ be an actegory, and $a \in \mathcal{A}$ and $x \in \mathcal{V}$. Recall that a *right exponential* of a by x is an object $a \rightsquigarrow x$ equipped with a universal morphism $\text{eval} : ((a \rightsquigarrow x) \otimes x) \rightarrow a$, i.e., for every arrow $f : b \otimes x \rightarrow a$ there is a unique arrow $\text{curry}f : b \rightarrow (a \rightsquigarrow x)$ making the diagram on the right commute. To distinguish the special case when the monoidal category acts on itself, we'll use the notation $x \leftarrow \otimes y$ for a right exponential of x by y . An actegory $(\mathcal{V}, \mathcal{A})$ is *closed* when all right-exponentials exist. In that case, (\otimes) distributes over coproducts in \mathcal{A} . We prove the following result in §B.3.

Proposition 8.2 *The actegory of \mathbf{R} -structures is right-closed: $(P|_{\text{snd}} \rightsquigarrow Q|_{\text{fst}})_s \Gamma := \int_{\Delta \in \text{Fst}_\perp} (P_s \Delta) Q_\Delta^{\text{Env} \Gamma}$.*

We prove the following generalisation of the classical theory in Appendix D. It implies the Special Representation Thm 6.4. Like its classical counterpart, it abstracts from the technical details of **R-Struct**.

Theorem 8.3 (general representation) *Let $(\mathcal{V}, \mathcal{A})$ be a closed actegory with finite coproducts. Letting $\mathcal{S} := \mathcal{V} \times \mathcal{A}$ be the product \mathcal{V} -actegory and $\mathbb{J} := (\mathbb{I}, \mathbb{0}) \in \mathcal{S}$, take any pointed-strong functor $\mathbf{O} : \mathcal{S}_* \rightarrow \mathcal{S}_*$, object $\mathbb{h} \in \mathcal{S}$, and initial algebra structure $([\![-\!]] : \mathbf{O}\mathbb{S} \rightarrow \mathbb{S}, \text{var} : \mathbb{I} \rightarrow \mathbb{S}_1, ?[-] : \mathbb{h} \otimes \mathbb{S}_1 \rightarrow \mathbb{S})$ over the object $\mathbb{S} = (\mathbb{S}_1, \mathbb{S}_2) := \mu(x_1, x_2).(\mathbf{O}(x_1, x_2) \amalg \mathbb{J} \amalg (\mathbb{h} \otimes x_1))$. There is a unique \mathcal{S} -morphism, called simultaneous substitution, $?[-] : \mathbb{S} \otimes \mathbb{S}_1 \rightarrow \mathbb{S}$ satisfying analogous equations to Thm 6.4. The free \mathbf{O} -action over \mathbb{h} is then $\mathbf{F}_\mathbf{O}\mathbb{h} := (\mathbb{S}, ?[-], \text{var}, [\![-\!]])$ equipped with the arrow $? : \mathbb{h} \xrightarrow{\mathbf{r}'} \mathbb{h} \otimes \mathbb{I} \xrightarrow{\text{id} \otimes \text{var}} \mathbb{h} \otimes \mathbb{S}_1 \xrightarrow{?[-]} \mathbb{S}$.*

8.3 Skew-monoidal categories

In an earlier version of this article, we developed MAST by recourse to the following structure:

$$(\otimes) : \mathbf{R}\text{-Struct} \times \mathbf{R}\text{-Struct} \rightarrow \mathbf{R}\text{-Struct} \quad P \otimes Q := P \otimes Q|_{\text{fst}} \quad \mathbb{J} \in \mathbf{R}\text{-Struct} \quad \mathbb{J}_s \Gamma := \{x | (x : s) \in \Gamma\}$$

This structure has been used, e.g., by Goncharov et al. [49] and by Greg Brown as the universe of syntax for CBPV. It falls short from being a monoidal category: while we can define the mediator maps, the putative left unitor $\ell : \mathbb{J} \otimes P \rightarrow P$ is not invertible in the presence of second-class sorts. For example, for each $s \in \text{Snd}$, we have $\mathbb{J}_{\text{snd}, s} \Gamma = \emptyset$, and so: $(\mathbb{J} \otimes \mathbb{1})_{\text{snd}, s} \Gamma = \int^\Delta \mathbb{J}_{\text{snd}, s} \Delta \times \mathbb{1} = \int^\Delta \emptyset \times \mathbb{1} = \emptyset \not\cong \mathbb{1}$. Therefore, even though **R-Struct** fails to be a monoidal category, this structure exhibits it as a *skew* monoidal category (cf. §C.4). Moreover, both the associator $(\mathbf{a} : (P \otimes Q) \otimes R \rightarrow P \otimes (Q \otimes R))$ and right unitor $(\mathbf{r}' : P \rightarrow P \otimes \mathbb{J})$ are invertible, in a situation we call an *associative* and *right-unital* skew monoidal category. To our surprise, the general representation theorem of the classical theory remains true even under the weaker assumptions that the left unitor is not invertible. That development took advantage of results by Fiore and Szamozvancev [32] about skew monoidal categories for the classical theory. We show how to recover the skew situation, and its special representation theorem, from our actegorical perspective.

Let $(\mathcal{V}, \mathcal{A})$ be an actegory. Assume \mathcal{A} has an initial object $\mathbb{0}$ preserved by the action $(\otimes) : \mathcal{A} \times \mathcal{V} \rightarrow \mathcal{A}$: the unique morphisms $[\!] : \mathbb{0} \rightarrow \mathbb{0} \otimes a$ are invertible for each $a \in \mathcal{V}$. Tensors of closed actegories preserve $\mathbb{0}$.

Proposition 8.4 *Let $(\mathcal{V}, \mathcal{A})$ be an actegory. If \mathcal{A} has an initial object $\mathbb{0}$ and the action $(\otimes) : \mathcal{A} \times \mathcal{V} \rightarrow \mathcal{A}$ preserves it, then $\mathcal{S} := (\mathcal{V} \times \mathcal{A}, (\otimes), \mathbb{J}, \mathbf{a}, \mathbf{r}', \ell)$ is an associative right-unital skew monoidal category, where:*

$$(\otimes) : \underline{\mathcal{S}} \times \underline{\mathcal{S}} \rightarrow \underline{\mathcal{S}} \quad (a, x) \otimes (b, y) := (a \otimes b, x \otimes b) \quad \mathbb{J} := (\mathbb{I}, \mathbb{0}) \quad \mathbf{a} := (\mathbf{a}, \mathbf{a}) \quad \mathbf{r}' := (\mathbf{r}^{-1}, \mathbf{r}^{-1}) \quad \ell := (\ell, [\!])$$

We have a carrier-preserving isomorphism between the categories of $(\mathcal{V}, \mathcal{A})$ -actions and \mathcal{S} -monoids. Moreover, if $(\mathcal{V}, \mathcal{A})$ are (right-)closed, then \mathcal{S} is right-closed.

See §C.4 for the proof. The general representation theorem for the skew setting does not follow from Prop. 8.4 and Thm 8.3. The actegorical angle is natural, however, and extends previous work [40].

9 Related work

The two POPL-Mark challenges [13,1] galvanise the programming-language community to hard problems in formalisation. Both challenges emphasise representation and manipulation of syntax with binding and substitution. We do not target mechanisation and computational realisation of abstract syntax with binding and substitution, but we note the ample work of this nature. All modern mechanisation systems support libraries or features for abstract syntax with binding [79,12,89,41,82,83,74,43,44,80, e.g.]. These generate specialised functions, lemmata, and proofs given a description of the syntax, instead of proving a general theory of syntax and substitution. We direct the reader to Allais et al.’s related work section [9] which surveys: non-de Bruijn approaches to binding [42,21,19]; alternative binding structure [91,77,20,45,50]; more work on automation [76,60]; universes of syntax and generic programming [59,58,14,64,95,24,71]. To those we add HOAS [56,73], monadic [10] and functorial [15] representations.

The presheaf approach [33,11,39,37,84] lends itself to mathematical operational semantics [88,47] and G. structural operational semantics (GSOS) [49]. Fiore and Turi [40] considered a special case of our heterogeneous situation in which the constructs for first-class sorts can be described independently from the second-class sorts in terms of actions. In our situation, we have mutual dependency between terms of first-class and second-class sorts, e.g., CBV terms include values, and functions, which are values, are abstracted terms. McBride varies the indexing category e.g., thinnings/order-preserving embeddings to implement co-de Bruijn representations [69]. In a different direction, Fiore and Saville reduce the free \mathbf{O} -monoid to the existence of certain list objects [30].

Fiore and Szamozvancev reformulate [32,86] and implement [31] the classical theory in terms of homogeneous *families*, i.e., sort-and-context indexed sets without a given functorial action. These families have an associated tensor without a quotient $(P \otimes Q)_s \Gamma := \coprod_{\Delta} P_s \Delta \times \prod_{(x:r) \in \Delta} Q_r \Gamma$. The theory compensates for the missing quotient by demanding additional axioms for substitution. Their tensor product (\otimes) is skew, and it is neither unital nor associative.

Close to the presheaf approach is the familial approach of Hirschowitz et al. [54,17,53,55], which also uses a skew tensor for technical reasons involving operational semantics and bisimilarity. Ahrens also uses families rather than presheaves [6] including an implementation [7] in UniMath [90]. The introductory text by Lamiaux and Ahrens [63] provides many connections between these approaches and related work.

Fiore, Gambino, Hyland, and Winskel’s Kleisli bicategories [35] follow their earlier work [36] generalising Joyal’s species of structure [57]. Olimpieri et al. [72,22] used these, e.g., when studying intersection type systems. More recently, Fiore, Galal, and Paquet introduce a bicategory of stable species [34]. Ahrens et al.’s aforementioned implementation [7] also uses such bicategorical ideas.

The Expression Problem concerns the design of an abstract syntax tree datatype for simply-typed expression language without binding and an implementation of an evaluator. Wadler attributes the problem to [78]. In Wadler’s formulation, the challenge is to extend the language with new term constructions without recompiling the code for the previous versions. Wadler reviewed several existing solutions, including object oriented ones by Cook [23]; Krishnamurthi et al. [62]; and Zenger and Odersky [92,93,94], before presenting a solution based on recursive generics in Java.

10 Conclusion

We have extended the classical theory of abstract syntax with binding and substitution with second-class sorts through categorical structure. This extension is straightforward thanks to the existing bicategorical perspective. Our tutorial separates the concepts needed to employ the theory from its underlying technical machinery and development. We expect similar results from an analogous case study on Levy’s CBPV. There the basic calculus comprises of returners and sequencing, and one can extend it with value products and coproducts, thunks, computation products, and functions. Thus, with MAST one could deduce $2^6 = 64$ different substitution lemmata by checking the compatibility conditions for 6 signatures. We want to develop a corresponding algorithmic theory using familial skew actions [32] and implement it. We are also interested in developing a multi-categorical perspective. It may avoid the need for quotients, arising through the tensor product and represent the multi-category [52,18,65,66,27,26]. This perspective might simplify the algorithmic theory. This work was supported by the Air Force Office of Scientific Research under award number FA9550-21-1-0038, ERC Grant BLAST, two ARIA SGAI TA1.1 grants, a Royal

Society University Research Fellowship, as well as FWF Project AUTOSARD P 36623. For the purpose of Open Access the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. We thank the chairs and editor for their patience, and the anonymous referees and these people for interesting and useful discussions and suggestions: Nathanael Arkor; Bob Atkey; Greg Brown; Vikraman Choudhury; Yotam Dvir; Nicola Gambino; Sergey Goncharov; Nick Hu; M. Codrin Iftode; Meven Lennon-Bertrand; Paul B. Levy; Cristina Matache; Justus Matthiesen; Conor McBride; Sean K. Moss; Filip Sieczkowski; Peter M. Sewell; Stelios Tsampas; Zoe Stafford; Dmitrij Szamozvancev; and Jacob Walters.

References

- [1] Abel, A., G. Allais, A. Hameer, B. Pientka, A. Momigliano, S. Schäfer and K. Stark, *Poplmark reloaded: Mechanizing proofs by logical relations*, Journal of Functional Programming **29**, page e19 (2019).
<https://doi.org/10.1017/S0956796819000170>
- [2] Aczel, P., *A general Church-Rosser theorem*. University of Manchester, Technical report, Technical report (1978).
- [3] Aczel, P., J. Adámek, S. Milius and J. Velebil, *Infinite trees and completely iterative theories: a coalgebraic view*, Theoretical Computer Science **300**, pages 1–45 (2003), ISSN 0304-3975.
[https://doi.org/https://doi.org/10.1016/S0304-3975\(02\)00728-4](https://doi.org/https://doi.org/10.1016/S0304-3975(02)00728-4)
- [4] Adámek, J., S. Milius and J. Velebil, *Elgot algebras*, Logical Methods in Computer Science **Volume 2, Issue 5**, 4 (2006), ISSN 1860-5974.
[https://doi.org/10.2168/LMCS-2\(5:4\)2006](https://doi.org/10.2168/LMCS-2(5:4)2006)
- [5] Adámek, J., S. Milius and J. Velebil, *Equational properties of iterative monads*, Information and Computation **208**, pages 1306–1348 (2010), ISSN 0890-5401. Special Issue: International Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).
<https://doi.org/https://doi.org/10.1016/j.ic.2009.10.006>
- [6] Ahrens, B., *Modules over relative monads for syntax and semantics*, Mathematical Structures in Computer Science **26**, page 3–37 (2016).
<https://doi.org/10.1017/S0960129514000103>
- [7] Ahrens, B., R. Matthes and A. Mörtberg, *Implementing a category-theoretic framework for typed abstract syntax*, in: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, page 307–323, Association for Computing Machinery, New York, NY, USA (2022), ISBN 9781450391825.
<https://doi.org/10.1145/3497775.3503678>
- [8] Allais, G., R. Atkey, J. Chapman, C. McBride and J. McKinna, *A type and scope safe universe of syntaxes with binding: their semantics and proofs*, Proc. ACM Program. Lang. **2** (2018).
<https://doi.org/10.1145/3236785>
- [9] Allais, G., R. Atkey, J. Chapman, C. McBride and J. McKinna, *A type- and scope-safe universe of syntaxes with binding: their semantics and proofs*, Journal of Functional Programming **31** (2021).
<https://doi.org/10.1017/S0956796820000076>
- [10] Altenkirch, T. and B. Reus, *Monadic presentations of lambda terms using generalized inductive types*, in: J. Flum and M. 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).
https://doi.org/10.1007/3-540-48168-0_32
- [11] Arkor, N. and D. McDermott, *Abstract Clones for Abstract Syntax*, in: N. Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:19, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021), ISBN 978-3-95977-191-7, ISSN 1868-8969.
<https://doi.org/10.4230/LIPIcs.FSCD.2021.30>
- [12] Aydemir, B., A. Charguéraud, B. C. Pierce, R. Pollack and S. Weirich, *Engineering formal metatheory*, in: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, page 3–15, Association for Computing Machinery, New York, NY, USA (2008), ISBN 9781595936899.
<https://doi.org/10.1145/1328438.1328443>

[13] Aydemir, B. E., A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich and S. Zdancewic, *Mechanized metatheory for the masses: The poplmark challenge*, in: J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics*, pages 50–65, Springer Berlin Heidelberg, Berlin, Heidelberg (2005), ISBN 978-3-540-31820-0.

[14] Benton, N., C.-K. Hur, A. J. Kennedy and C. McBride, *Strongly typed term representations in coq*, *J. Autom. Reason.* **49**, page 141–159 (2012), ISSN 0168-7433.
<https://doi.org/10.1007/s10817-011-9219-0>

[15] Blanchette, J. C., L. Gheri, A. Popescu and D. Traytel, *Bindings as bounded natural functors*, *Proc. ACM Program. Lang.* **3**, pages 22:1–22:34 (2019).
<https://doi.org/10.1145/3290335>

[16] Bloom, S. L. and Z. Esik, *Iteration Theories: The Equational Logic of Iterative Processes*, Springer Publishing Company, Incorporated, 1st edition (2012), ISBN 3642780369.

[17] Borthelle, P., T. Hirschowitz and A. Lafont, *A cellular Howe theorem*, in: H. Hermanns, L. Zhang, N. Kobayashi and D. 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).
<https://doi.org/10.1145/3373718.3394738>

[18] Burroni, A., *T-catégories (catégories dans un triple)*, *Cahiers de Topologie et Géométrie Différentielle Catégoriques* **12**, pages 215–321 (1971).
<http://eudml.org/doc/91097>

[19] Charguéraud, A., *The locally nameless representation*, *Journal of automated reasoning* **49**, pages 363–408 (2012).

[20] Cheney, J., *Toward a general theory of names: binding and scope*, in: *Proceedings of the 3rd ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding*, MERLIN '05, page 33–40, Association for Computing Machinery, New York, NY, USA (2005), ISBN 1595930728.
<https://doi.org/10.1145/1088454.1088459>

[21] Chlipala, A., *Parametric higher-order abstract syntax for mechanized semantics*, in: *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ICFP '08, page 143–156, Association for Computing Machinery, New York, NY, USA (2008), ISBN 9781595939197.
<https://doi.org/10.1145/1411204.1411226>

[22] Clairambault, P., F. Olimpieri and H. Paquet, *From Thin Concurrent Games to Generalized Species of Structures*, in: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14, IEEE Computer Society, Los Alamitos, CA, USA (2023).
<https://doi.org/10.1109/LICS56636.2023.10175681>

[23] Cook, W. R., *Object-oriented programming versus abstract data types*, in: J. W. de Bakker, W. P. de Roever and G. Rozenberg, editors, *Foundations of Object-Oriented Languages*, pages 151–178, Springer Berlin Heidelberg, Berlin, Heidelberg (1991), ISBN 978-3-540-46450-1.

[24] Copello, E., N. Szasz and Á. Tasistro, *Formalization of metatheory of the lambda calculus in constructive type theory using the barendregt variable convention*, *Mathematical Structures in Computer Science* **31**, page 341–360 (2021).
<https://doi.org/10.1017/S0960129521000335>

[25] Crole, R. L., *The representational adequacy of Hybrid*, *Math. Struct. Comput. Sci.* **21**, pages 585–646 (2011).
<https://doi.org/10.1017/S0960129511000041>

[26] Cruttwell, G. S. H. and M. A. Shulman, *A unified framework for generalized multicategories* (2010). [0907.2460](https://arxiv.org/abs/0907.2460).
<https://arxiv.org/abs/0907.2460>

[27] Dawson, R., R. Paré and D. Pronk, *The span construction*, *Theory Appl. Categ.* **24**, pages No. 13, 302–377 (2010), ISSN 1201-561X.

[28] de Bruijn, N., *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem*, *Indagationes Mathematicae (Proceedings)* **75**, pages 381–392 (1972), ISSN 1385-7258.
[https://doi.org/https://doi.org/10.1016/1385-7258\(72\)90034-0](https://doi.org/https://doi.org/10.1016/1385-7258(72)90034-0)

[29] Elgot, C. C., *Monadic computation and iterative algebraic theories**the material reported on here evolved from research which was initiated at the university of bristol and supported by the science research council of great britain.*, in: H. Rose and J. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 175–230, Elsevier (1975).
[https://doi.org/https://doi.org/10.1016/S0049-237X\(08\)71949-9](https://doi.org/https://doi.org/10.1016/S0049-237X(08)71949-9)

[30] Fiore, M. and P. Saville, *List Objects with Algebraic Structure*, in: D. Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2017), ISBN 978-3-95977-047-7, ISSN 1868-8969.
<https://doi.org/10.4230/LIPIcs.FSCD.2017.16>

[31] Fiore, M. and D. Szamozvancev, *Formal metatheory of second-order abstract syntax*, Proc. ACM Program. Lang. **6** (2022).
<https://doi.org/10.1145/3498715>

[32] Fiore, M. and D. Szamozvancev, *Familial model of second-order abstract syntax* (2025). Unpublished.

[33] Fiore, M. P., *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).
<https://doi.org/10.1109/LICS.2008.38>

[34] Fiore, M. P., Z. Galal and H. Paquet, *Stabilized profunctors and stable species of structures*, Log. Methods Comput. Sci. **20** (2024).
[https://doi.org/10.46298/LMCS-20\(1:17\)2024](https://doi.org/10.46298/LMCS-20(1:17)2024)

[35] Fiore, M. P., N. Gambino, M. Hyland and G. Winskel, *Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures*, Selecta Mathematica New Series **24**, pages 2791–2830 (2018).
<https://doi.org/10.1007/s00029-017-0361-3>

[36] Fiore, M. P., N. Gambino, M. H. Hyland and G. Winskel, *The cartesian closed bicategory of generalised species of structures*, Journal of the London Mathematical Society **77**, pages 203–220 (2008).

[37] Fiore, M. P. and M. Hamana, *Multiversal polymorphic algebraic theories: Syntax, semantics, translations, and equational logic*, in: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '13*, page 520–529, IEEE Computer Society, USA (2013), ISBN 9780769550206.

[38] Fiore, M. P., O. Kammar, G. Moser and S. Staton, *Modular abstract syntax trees (mast): substitution tensors with second-class sorts* (2025). <https://arxiv.org/pdf/2511.03946v1.pdf>
<https://arxiv.org/abs/2511.03946v1>

[39] Fiore, M. P., G. D. Plotkin and D. Turi, *Abstract syntax and variable binding (extended abstract)*, in: *Proc. 14th LICS Conf.*, pages 193–202, IEEE, Computer Society Press (1999).

[40] Fiore, M. P. and D. Turi, *Semantics of name and value passing*, in: *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 93–104, IEEE Computer Society (2001).
<https://doi.org/10.1109/LICS.2001.932486>

[41] Forster, Y. and K. Stark, *Coq à la carte — a practical approach to modular syntax with binders*, in: *9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, USA*, ACM (2020).

[42] Gabbay, M. J. and A. M. Pitts, *A new approach to abstract syntax with variable binding*, Form. Asp. Comput. **13**, page 341–363 (2002), ISSN 0934-5043.
<https://doi.org/10.1007/s001650200016>

[43] Gacek, A., *A Framework for Specifying, Prototyping, and Reasoning about Computational Systems*, Ph.D. thesis, University of Minnesota (2009).

[44] Gacek, A., D. Miller and G. Nadathur, *A two-level logic approach to reasoning about computations*, Journal of Automated Reasoning **49**, pages 241–273 (2012).

[45] Ghani, N., M. Hamana, T. Uustalu and V. Vene, *Representing cyclic structures as nested datatypes* pages 173–188 (2006).

[46] Goguen, J. A., J. W. Thatcher, E. G. Wagner and J. B. Wright, *Initial algebra semantics and continuous algebras*, J. ACM **24**, page 68–95 (1977), ISSN 0004-5411.
<https://doi.org/10.1145/321992.321997>

[47] Goncharov, S., S. Milius, L. Schröder, S. Tsampas and H. Urbat, *Towards a higher-order mathematical operational semantics*, Proc. ACM Program. Lang. **7**, pages 632–658 (2023).
<https://doi.org/10.1145/3571215>

[48] Goncharov, S., C. Rauch and L. Schröder, *Unguarded recursion on coinductive assumptions*, Electronic Notes in Theoretical Computer Science **319**, pages 183–198 (2015), ISSN 1571-0661. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).
<https://doi.org/https://doi.org/10.1016/j.entcs.2015.12.012>

[49] Goncharov, S., S. Tsampas and H. Urbat, *Abstract operational methods for call-by-push-value*, Proc. ACM Program. Lang. **9**, pages 1013–1039 (2025).
<https://doi.org/10.1145/3704871>

[50] Hamana, M., *Initial algebra semantics for cyclic sharing structures*, in: P.-L. Curien, editor, *Typed Lambda Calculi and Applications*, pages 127–141, Springer Berlin Heidelberg, Berlin, Heidelberg (2009), ISBN 978-3-642-02273-9.

[51] Hasegawa, M. and Y. Kakutani, *Axioms for recursion in call-by-value*, in: F. Honsell and M. Miculan, editors, *Foundations of Software Science and Computation Structures*, pages 246–260, Springer Berlin Heidelberg, Berlin, Heidelberg (2001), ISBN 978-3-540-45315-4.

[52] Hermida, C., *Representable multicategories*, Advances in Mathematics **151**, pages 164–225 (2000), ISSN 0001-8708.
<https://doi.org/https://doi.org/10.1006/aima.1999.1877>

[53] Hirschowitz, A., T. Hirschowitz and A. Lafont, *Modules over monads and operational semantics*, in: Z. M. ndré, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29–July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPICS*, pages 12:1–12:23, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020).
<https://doi.org/10.4230/LIPICS.FSCD.2020.12>

[54] Hirschowitz, A., T. Hirschowitz, A. Lafont and M. Maggesi, *Variable binding and substitution for (nameless) dummies*, *CorR* **abs/2209.02614** (2022). [2209.02614](https://doi.org/2209.02614)
<https://doi.org/10.48550/ARXIV.2209.02614>

[55] Hirschowitz, A. and M. Maggesi, *Modules over monads and initial semantics*, *Information and Computation* **208**, pages 545–564 (2010), ISSN 0890-5401. Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007).
<https://doi.org/https://doi.org/10.1016/j.ic.2009.07.003>

[56] Hofmann, M., *Semantical analysis of higher-order abstract syntax*, in: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 204–213 (1999).
<https://doi.org/10.1109/LICS.1999.782616>

[57] Joyal, A., *Une théorie combinatoire des séries formelles*, Advances in Mathematics **42**, pages 1–82 (1981), ISSN 0001-8708.
[https://doi.org/https://doi.org/10.1016/0001-8708\(81\)90052-9](https://doi.org/https://doi.org/10.1016/0001-8708(81)90052-9)

[58] Keuchel, S., *Generic programming with binders and scope* (2011).

[59] Keuchel, S. and J. T. Jeuring, *Generic conversions of abstract syntax representations*, in: *Proceedings of the 8th ACM SIGPLAN Workshop on Generic Programming*, WGP ’12, page 57–68, Association for Computing Machinery, New York, NY, USA (2012), ISBN 9781450315760.
<https://doi.org/10.1145/2364394.2364403>

[60] Keuchel, S., S. Weirich and T. Schrijvers, *Needle & knot: Binder boilerplate tied up*, in: *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*, page 419–445, Springer-Verlag, Berlin, Heidelberg (2016), ISBN 9783662494974.
https://doi.org/10.1007/978-3-662-49498-1_17

[61] Kock, A., *Monads on symmetric monoidal closed categories*, Archiv der Mathematik **21**, pages 1–10 (1970).
<https://doi.org/10.1007/BF01220868>

[62] Krishnamurthi, S., M. Felleisen and D. P. Friedman, *Synthesizing object-oriented and functional design to promote reuse*, in: E. Jul, editor, *ECOOP’98 — Object-Oriented Programming*, pages 91–113, Springer Berlin Heidelberg, Berlin, Heidelberg (1998), ISBN 978-3-540-69064-1.

[63] Lamiaux, T. and B. Ahrens, *An introduction to different approaches to initial semantics* (2024). [2401.09366](https://arxiv.org/abs/2401.09366)
<https://arxiv.org/abs/2401.09366>

[64] Lee, G., B. C. D. S. Oliveira, S. Cho and K. Yi, *Gmeta: A generic formal metatheory framework for first-order representations*, in: H. Seidl, editor, *Programming Languages and Systems*, pages 436–455, Springer Berlin Heidelberg, Berlin, Heidelberg (2012), ISBN 978-3-642-28869-2.

[65] Leinster, T., *fc-multicategories* (1999). [math/9903004](https://arxiv.org/abs/math/9903004)
<https://arxiv.org/abs/math/9903004>

[66] Leinster, T., *Higher Operads, Higher Categories*, London Mathematical Society Lecture Note Series, Cambridge University Press (2004).

[67] Levy, P. B., *Call-by-push-value: A subsuming paradigm*, in: *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications*, TLCA ’99, page 228–242, Springer-Verlag, Berlin, Heidelberg (1999), ISBN 3540657630.

[68] Levy, P. B., *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*, Springer (2004).

[69] McBride, C., *Everybody's got to be somewhere*, in: R. Atkey and S. Lindley, editors, *Proceedings of the 7th Workshop on Mathematically Structured Functional Programming, MSFP@FSCD 2018, Oxford, UK, 8th July 2018*, volume 275 of *EPTCS*, pages 53–69 (2018).
<https://doi.org/10.4204/EPTCS.275.6>

[70] McDermott, D. and T. Uustalu, *What makes a strong monad?*, Electronic Proceedings in Theoretical Computer Science **360**, page 113–133 (2022), ISSN 2075-2180.
<https://doi.org/10.4204/eptcs.360.6>

[71] Morris, P., T. Altenkirch and C. McBride, *Exploring the regular tree types*, in: J.-C. Filliâtre, C. Paulin-Mohring and B. Werner, editors, *Types for Proofs and Programs*, pages 252–267, Springer Berlin Heidelberg, Berlin, Heidelberg (2006), ISBN 978-3-540-31429-5.

[72] Olimpieri, F., *Intersection Types and Resource Calculi in the Denotational Semantics of Lambda-Calculus*, Theses, Aix-Marseille Université (2020).
<https://theses.hal.science/tel-03123485>

[73] Pfenning, F. and C. Elliott, *Higher-order abstract syntax*, in: *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, PLDI '88, page 199–208, Association for Computing Machinery, New York, NY, USA (1988), ISBN 0897912691.
<https://doi.org/10.1145/53990.54010>

[74] Pientka, B., *Beluga: Programming with dependent types, contextual data, and contexts*, in: M. Blume, N. Kobayashi and G. Vidal, editors, *Functional and Logic Programming*, pages 1–12, Springer Berlin Heidelberg, Berlin, Heidelberg (2010), ISBN 978-3-642-12251-4.

[75] Plotkin, G. D., *An illative theory of relations*, Situation Theory and its Applications pages 133–146 (1990).

[76] Polonowski, E., *Automatically generated infrastructure for de bruijn syntaxes*, in: S. Blazy, C. Paulin-Mohring and D. Pichardie, editors, *Interactive Theorem Proving*, pages 402–417, Springer Berlin Heidelberg, Berlin, Heidelberg (2013), ISBN 978-3-642-39634-2.

[77] Poulsen, C., A. Rouvoet, A. Tolmach, R. Krebbers and E. Visser, *Intrinsically-typed definitional interpreters for imperative languages*, Proceedings of the ACM on Programming Languages **2**, pages 1–34 (2018), ISSN 2475-1421.
<https://doi.org/10.1145/3158104>

[78] Reynolds, J. C., *User-Defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction*, pages 309–317, Springer New York, New York, NY (1978), ISBN 978-1-4612-6315-9.
https://doi.org/10.1007/978-1-4612-6315-9_22

[79] Schäfer, S., G. Smolka and T. Tebbi, *Completeness and decidability of de bruijn substitution algebra in coq*, in: *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP 2015, Mumbai, India, January 15–17, 2015, pages 67–73, ACM (2015).

[80] Sewell, P., F. Z. Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar and R. Strniša, *Ott: Effective tool support for the working semanticist*, Journal of Functional Programming **20**, page 71–122 (2010).
<https://doi.org/10.1017/S0956796809990293>

[81] Simpson, A. K. and G. D. Plotkin, *Complete axioms for categorical fixed-point operators*, in: *LICS*, pages 30–41 (2000).
<http://www.computer.org/proceedings/lics/0725/07250030abs.htm>

[82] Stark, K., *Mechanising Syntax with Binders in Coq*, Ph.D. thesis, Saarland University (2020).

[83] Stark, K., S. Schäfer and J. Kaiser, *Autosubst 2: Reasoning with multi-sorted de bruijn terms and vector substitutions*, 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14–15, 2019 (2019).

[84] Sterling, J. and D. Morrison, *Syntax and semantics of abstract binding trees*, CoRR **abs/1601.06298** (2016). [1601.06298](https://arxiv.org/abs/1601.06298).
<http://arxiv.org/abs/1601.06298>

[85] Swierstra, W., *Data types à la carte*, Journal of Functional Programming **18**, pages 423–436 (2008).
<https://doi.org/10.1017/S0956796808006758>

[86] Szamozvancev, D., *Categorical models of second-order abstract syntax*, Ph.D. thesis, University of Cambridge (2025).

[87] Szlachányi, K., *Skew-monoidal categories and bialgebroids*, Advances in Mathematics **231**, pages 1694–1730 (2012), ISSN 0001-8708.
<https://doi.org/https://doi.org/10.1016/j.aim.2012.06.027>

[88] Turi, D. and G. D. Plotkin, *Towards a mathematical operational semantics*, in: *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 280–291 (1997).
<https://doi.org/10.1109/LICS.1997.614955>

[89] Urban, C., *Nominal techniques in Isabelle/HOL*, J. Autom. Reason. **40**, pages 327–356 (2008).
<https://doi.org/10.1007/S10817-008-9097-2>

[90] Voevodsky, V., B. Ahrens, D. Grayson *et al.*, *Unimath — a computer-checked library of univalent mathematics*, available at <http://unimath.org>.
<https://doi.org/10.5281/zenodo.10849216>

[91] Weirich, S., B. A. Yorgey and T. Sheard, *Binders unbound*, SIGPLAN Not. **46**, page 333–345 (2011), ISSN 0362-1340.
<https://doi.org/10.1145/2034574.2034818>

[92] Zenger, M., *Erweiterbare übersetzer*, Master’s thesis, University of Karlsruhe (1998).

[93] Zenger, M. and M. Odersky, *Extensible algebraic datatypes with defaults*, SIGPLAN Not. **36**, page 241–252 (2001), ISSN 0362-1340.
<https://doi.org/10.1145/507669.507665>

[94] Zenger, M. and M. Odersky, *Implementing extensible compilers* (2001).
<https://infoscience.epfl.ch/handle/20.500.14299/221735>

[95] Érdi, G., *Generic description of well-scoped, well-typed syntaxes* (2018). [1804.00119](https://arxiv.org/abs/1804.00119)
<https://arxiv.org/abs/1804.00119>