Binder Aware Recursion over Well-Scoped de Bruijn Syntax

Jonas Kaiser
Saarland University
Saarbrücken, Germany
jkaiser@ps.uni-saarland.de

Steven Schäfer
Saarland University
Saarbrücken, Germany
schaefer@ps.uni-saarland.de

Kathrin Stark
Saarland University
Saarbrücken, Germany
kstark@ps.uni-saarland.de

Abstract

The de Bruijn representation of syntax with binding is commonly used, but flawed when it comes to recursion. As the structural recursion principle associated to an inductive type of expressions is unaware of the binding discipline, each recursive definition requires a separate proof of compatibility with variable instantiation. We solve this problem by extending Allais' notion of syntax traversals to obtain a framework for instantiation-compatible recursion. The framework is general enough to handle multivariate, potentially mutually recursive syntactic systems.

With our framework we define variable renaming and instantiation, syntax directed typing and certain unary logical relations for System F. These definitons lead to concise proofs of type preservation, as well as weak and strong normalisation.

Our framework is designed to serve as the theoretical foundation of future versions of the Autosubst Coq library. All developments and case studies are formalised in the Coq proof assistant.

CCS Concepts • **Theory of computation** → *Automated reasoning*; *Type theory*; *Operational semantics*;

Keywords well-scoped de Bruijn representation, recursion principle, parallel substitutions, System F

ACM Reference Format:
Jonas Kaiser, Steven Schäfer, and Kathrin Stark. 2018. Binder Aware Recursion over Well-Scoped de Bruijn Syntax. In *Proceedings of 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3167098

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CPP'18, January 8–9, 2018, Los Angeles, CA, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5586-5/18/01…$15.00
https://doi.org/10.1145/3167098

1 Introduction

The formal treatment of syntactic systems à la de Bruijn [7] consists of two parts: a first-order, nameless representation of syntactic expressions that encodes variables as numerical indices and, second, a *binding discipline* which gives meaning to this encoding. The intended notion of variable instantiation is a recursively defined syntactic operation that applies a *parallel substitution*, that is, a functional mapping from indices to expressions, to an expression. In this design, α-equivalence reduces to plain syntactic equality and essential substitution lemmas can be obtained through basic equational reasoning [1, 22]. For these reasons, de Bruijn syntax has been used repeatedly and successfully in the mechanisation of programming language metatheory in proof assistants like Coq [14, 23].

There is, however, one major drawback: the structural recursion principle associated with the abstract datatype (ADT) of syntactic expressions fails to respect the binding discipline. As such, the recursor is mostly useless when it comes to syntactic operations which are expected to be compatible with variable renaming and instantiation. These include renaming and instantiation itself, syntax translations, logical relations and syntax directed inference systems. With the basic recursor, compatibility has to be established manually for each recursive definition.

Our goal is a framework that, given the de Bruijn representation of a syntacic system, yields the right notion of a recursor. It should, in particular, allow for the definition of syntactic operations that are compatible with renaming and instantiation by construction. A first step towards a solution is Allais et al.'s observation that most syntactic binder-respecting operations recursively traverse expressions in a very similar way [3], in particular with respect to the handling of binders. The common definitional structures are captured in Allais' work with the notion of an *abstract syntax traversal*. A traversal consists of semantic constructors that mirror their syntactic counterparts of a *well-scoped de Bruijn expression type* [2, 6]. *Evaluation* with respect to a traversal is then an abstract way of assigning, to each syntactic variable and expression, a denotation taken from an indexed family of types. Allais et al. give uniform definitions of renaming, instantiation, a CPS-transformation and the structures underlying various normalisation-by-evaluation proofs in terms of syntax traversals.

We extend Allais' results about structural commonalities to the study of properties and theorems that can be established generically for traversals and related structures. We start with a straightforward proof that Allais' traversals are compatible with the renaming of variables by construction. Compatibility with instantiation is more involved, though. Since traversals use distinct denotation domains for variables and expressions, they cannot express the desired equality. We identify a *lifting operation* that coalesces the two domains and thus allows us to formulate instantiation compatibility. Further, instantiation can be expressed as the lifting of the renaming traversal which clarifies the connection between the two operations.

The proof of instantiation compatibility relies on certain naturality conditions on the semantic constructors. We refer to a traversal that satisfies these conditions as a *model*. We further identify a subclass of traversals, called *simple traversals*, that omit all syntactic scope annotations. Simple traversals are always fully natural and therefore models. Many recursive definitions, with the notable exceptions of renaming and instantiation, are captured by this subclass.

We also propose to extend traversals and models to multivariate syntactic systems, i.e. syntactic systems that consist of several, potentially mutually recursive, syntactic sorts which each may contain their own class of variables. While evaluation with respect to a univariate model requires a single environment for the single class of free variables, we now require vectors of environments. When we consider multivariate instantiation in terms of such a traversal, it becomes clear that instantiation with vectors of parallel substitutions is the natural form of instantiation for multivariate syntactic systems. This observation supports the ideas presented in [14].

To demonstrate our framework, we include various case studies that concern the metatheory of System F. We use traversals to define typing and certain unary logical relations which are by construction compatible with instantiation. We then exploit these properties to obtain elegant proofs of preservation of typing and weak and strong normalisation.

A further application considers the renaming and instantiation traversals themselves. Application of the compatibility lemma yields compositionality of substitutions for free. This compositionality constitutes an integral part of the equational theory of the Autosubst Coq library [22, 23]. The remaining parts of the Autosubst equational theory are obtained in a simplified manner through our framework as well.

We conjecture that our framework is fully automatable. The denotation-parametric type of traversals, the naturality conditions for models, the concrete traversal structures for renaming and instantiation and the generic compatibility proofs and embedding lemmas are all fully determined by the underlying syntactic system. It is therefore conceivable to automatically generate these constructions from concise system descriptions, like for example the HOAS specifications proposed in [14]. We plan to incorporate this approach into a future version of the Autosubst library.

In summary, our contributions are as follows:

- A proof that abstract syntax traversals over well-scoped de Bruijn syntax à la Allais respect renaming by construction (Theorem 3.5).
- The idea that traversals support a notion of lifting (Section 3.3), as exemplified by the lifting of renaming to instantiation (Definition 3.10), and a generic embedding lemma that connects the two (Theorem 3.12).
- We identify a class of traversals, called models, that exhibit certain naturality conditions (Section 3.4), together with a proof that such models are by definition compatible with instantiation (Theorem 3.15).
- An extension of the framework to multivariate, potentially mutually recursive syntactic systems (Section 5).
- Elegant proofs of weak normalisation (Section 9) and preservation of typing (Section 7) for a call-by-value variant of System F.
- A proof of strong normalisation (Section 10) of a standard presentation of System F.

Accompanying Coq development. The development of the framework and the case studies are formalised in the Coq proof assistant. The Coq development is available at http://www.ps.uni-saarland.de/extras/cpp18-rec.

2 Preliminaries

Throughout this work we use a call-by value variant of System F (F_{CBV}) as our example syntactic system. F_{CBV} concisely showcases the complications that arise from potentially mutual dependencies between various sorts of a given system. While its sort of *types* is univariate and has a single binding constructor, the sorts of *terms* and *values* are multivariate, containing both type and value variables with two corresponding binders. In particular, the absence of term variables implies that β-reduction can only substitute values, which makes this a call-by-value system.

In the following we briefly introduce F_{CBV} together with the basic building blocks and notational conventions used in this work. We use a notion of well-scoped de Bruijn syntax [2], that is we use $\mathbb{N}$-indexed inductive type families to represent syntactic sorts. The syntax of F_{CBV} with full scope annotations is shown in Figure 1. The scopes, written as superscripts, are exclusive upper bounds on the freely occurring variables of certain sorts. All our constructions are designed to track and respect scoping but for the sake of readability we usually only annotate the types, not the expressions and operations.

Following de Bruijn [7], we encode *variables as indices* taken from a finite k-element type I^k with the index $x \in I^k$ referencing the xth enclosing binder of the corresponding

$$
\begin{array}{rcll}
A^k, B^k \in \mathsf{ty}^k & ::= & x^k_{\mathsf{ty}} \mid A^k \to B^k \mid \forall.\, A^{k+1} & x \in I^k \\
s^{k,l}, t^{k,l} \in \mathsf{tm}^{k,l} & ::= & s^{k,l}\, t^{k,l} \mid s^{k,l}\, A^k \mid v^{k,l} & \\
u^{k,l}, v^{k,l} \in \mathsf{vl}^{k,l} & ::= & x^{k,l}_{\mathsf{vl}} \mid \lambda A^k.\, s^{k,l+1} \mid \Lambda.\, s^{k+1,l} & x \in I^l
\end{array}
$$

Figure 1. Well-Scoped Syntax of $\mathrm{F}_{\mathrm{CBV}}$.

scope, counting upwards and starting from 0. Where necessary, we use a sort subscript to distinguish a syntactic variable, $x^k_{\mathsf{ty}} \in \mathsf{ty}^k$, from the underlying element of the finite type $x \in I^k$. We express the type variable constructor explicitly as id_{ty}, and similarly for the value variables.

Renamings $\xi, \zeta : I^n \to I^m$ map variables into variables and we write $A\langle\xi\rangle$ for the recursive operation that applies ξ simultaneously to all free variables of A. For *injective* renamings we write $I^n \hookrightarrow I^m$. As an example, consider the *shift* injection $\uparrow^n : I^n \hookrightarrow I^{n+1}$ which often occurs in conjunction with the element 0^{n+1} denoting the index not in the image of $\uparrow^n$.

We further require *type substitutions* $\sigma : I^n \to \mathsf{ty}^k$ and *value substitutions* $\tau : I^m \to \mathsf{vl}^{k,l}$ which, similarly to renamings, are applied in parallel to types, terms and values, written $A[\sigma]$, $s[\sigma, \tau]$ and $v[\sigma, \tau]$ respectively.

The free indices of an expression of scope k are interpreted in environments $\rho, \theta : I^k \to V^m$, where $V : \mathbb{N} \to \mathrm{Type}$ is an indexed family of types. Finite types I and the sort ty are both examples of such type families.

With respect to substitutions and environments we borrow two basic operations from the σ-calculus [1]. Composition, written $\xi \circ \sigma$, corresponds to forward function composition. The second operation is extension, also referred to as cons and written $A^n \cdot \sigma$. Given $\sigma : I^k \to \mathsf{ty}^n$, we obtain a substitution $A^n \cdot \sigma : I^{k+1} \to \mathsf{ty}^n$ which maps 0^{k+1} to A^n, and $(\uparrow\ x)^{k+1}$ to $\sigma\, x$. The equational theory that governs these operations for the univariate case, is discussed in detail in [1, 22]. First steps towards a similar theory for the multivariate setting are taken in [14].

3 A Framework for Recursion over Syntax with Binding

In this section we extend the syntax traversals of Allais et al. [3] into a general mechanism for defining structurally recursive functions on the types of $\mathrm{F}_{\mathrm{CBV}}$ while respecting instantiation.

After recalling the idea of a syntax traversal with minor variations, we focus on renamings. We prove that renaming is functorial and that each traversal is compatible with renaming. We then generalise the relation between renaming and instantiation into a *lifting* construction for traversals. This allows us to state precisely in what sense evaluation is compatible with instantiation. Finally, we refine the notion of syntax traversal into the notion of a syntactic model[1]. As key property, we prove that evaluating a term with respect to a model is compatible with instantiation.

3.1 Type Traversals

The notion of a syntax traversal was introduced by Allais et al. [3], as an analog for structural recursion on syntactic types with binders. Traversals consist of a traversal structure that contains all data necessary to define a recursive function and an evaluation function that performs the recursion. Throughout this section we consider ty-traversals, that is traversals for the types of $\mathrm{F}_{\mathrm{CBV}}$. When the underlying sort is clear from the context we drop the prefix. The signature $\mathbb{T}^V_D$ of traversal structures is parametric in two indexed type families $V, D : \mathbb{N} \to \mathrm{Type}$ which we refer to as *denotation domains* for indices and, respectively, type expressions.

Definition 3.1. A ty-*traversal* $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$ for the types of $\mathrm{F}_{\mathrm{CBV}}$ consists of semantic counterparts to the syntactic constructors.

$$
\begin{array}{rcl}
\mathbb{V} & : & \forall n.\, V^n \to D^n \\
\mathbb{A} & : & \forall n.\, D^n \to D^n \to D^n \\
\mathbb{Q} & : & \forall m.\, (\forall n.\, (I^m \hookrightarrow I^n) \to V^n \to D^n) \to D^m
\end{array}
$$

Since the main arguments of the semantic constructors fully determine the scope parameters, we keep them implicit and never explicitly abstract or instantiate them.

To evaluate a type with respect to a traversal, indices have to be lifted into different scopes. We hence need additional structure on the index denotations V.

Definition 3.2. A *graded domain* consists of a domain $V : \mathbb{N} \to \mathrm{Type}$ together with a thinning operation

$$
\_ \rhd_V \_ \;:\; \forall m\, n.\, V^m \to (I^m \hookrightarrow I^n) \to V^n.
$$

We omit the subscript on the thinning operation, when the domain is clear from the context. When the image of an environment ρ is graded, we lift the thinning operation to ρ and write $\rho \rhd \xi$ for $\rho \circ (\_ \rhd \xi)$. A graded domain is *compositional* if thinning respects compositions $x \rhd \xi \rhd \zeta = x \rhd \xi \circ \zeta$. It is *functorial*, if it is compositional and thinning respects identities, $x \rhd \mathsf{id} = x$. A graded domain is *injective* if thinning $\_ \rhd \xi$ is injective for all ξ.

The family I of finite types with thinning $x \rhd \xi = \xi\, x$, as well as any constant family $\lambda n.\, T$ for a fixed type T are examples of injective, functorial graded domains. Every compositional, injective graded domain is automatically functorial.

Definition 3.3. Let $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$ be a ty-traversal where V is a graded domain. Evaluation $[\![\_]\!]^{\mathsf{T}}_{\_}$ with respect

[1] So called, since it is almost a presheaf model.

to T is defined recursively as follows.

$$\begin{array}{lcl} [\![\_]\!]^{\mathsf{T}}_{\_} & : & \mathsf{ty}^m \to (I^m \to V^n) \to D^n \\ \\ [\![x_{\mathsf{ty}}]\!]^{\mathsf{T}}_{\rho} & = & \mathbb{V}\,(\rho\,x) \\ [\![A \to B]\!]^{\mathsf{T}}_{\rho} & = & \mathbb{A}\,[\![A]\!]^{\mathsf{T}}_{\rho}\,[\![B]\!]^{\mathsf{T}}_{\rho} \\ [\![\forall.\ A]\!]^{\mathsf{T}}_{\rho} & = & \mathbb{Q}(\lambda \xi\, v.[\![A]\!]^{\mathsf{T}}_{v\,\cdot(\rho \triangleright_V \xi)}) \end{array}$$

3.2 Renaming

We recall the definition of renaming as a traversal from [3].

Definition 3.4 (Renaming)**.** We define the *renaming traversal* $\mathsf{R}_{\mathsf{ty}} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^I_{\mathsf{ty}}$ with

$$\begin{array}{lcl} \mathbb{V} & = & \mathsf{id}_{ty} \\ \mathbb{A} & = & \_ \to \_ \\ \mathbb{Q} & = & \lambda\, F.\ (\forall.\ F\ \uparrow\ 0). \end{array}$$

Renaming of variables according to ξ is evaluation with respect to R_{ty} and environment ξ:

$$A\langle\xi\rangle := [\![A]\!]^{\mathsf{R}_{\mathsf{ty}}}_{\xi}$$

This definition of renaming satisfies the usual recursive equations of renaming:

$$\begin{array}{rcl} x_{\mathsf{ty}}\langle\xi\rangle & = & (\xi\,x)_{\mathsf{ty}} \\ (A \to B)\langle\xi\rangle & = & A\langle\xi\rangle \to B\langle\xi\rangle \\ (\forall.\ A)\langle\xi\rangle & = & \forall.\ A\langle 0 \cdot \xi \triangleright \uparrow\rangle \end{array}$$

We now prove, that all traversals are compatible with this definition of renaming.

Theorem 3.5 (Compatibility with Renaming)**.** *For an arbitrary* ty*-traversal* $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$*, type* $A : \mathsf{ty}^k$*, renaming* $\xi : I^k \to I^m$ *and environment* $\rho : I^m \to V^n$ *we have*

$$[\![A\langle\xi\rangle]\!]^{\mathsf{T}}_{\rho} = [\![A]\!]^{\mathsf{T}}_{\xi\,\circ\,\rho}.$$

Proof. By induction on A. The cases for variables and function types are trivial. The case of quantification $(\forall.\ A)$ follows from the inductive hypothesis and simple equational reasoning.

$$\begin{array}{rcl} [\![(\forall.\ A)\langle\xi\rangle]\!]^{\mathsf{T}}_{\rho} & = & \mathbb{Q}(\lambda\,\zeta\,v.\ [\![A\langle 0 \cdot \xi \triangleright \uparrow\rangle]\!]^{\mathsf{T}}_{v\,\cdot\,\rho \triangleright \zeta}) \\ & = & \mathbb{Q}(\lambda\,\zeta\,v.\ [\![A]\!]^{\mathsf{T}}_{(0\,\cdot\,\xi \triangleright \uparrow)\,\circ\,(v\,\cdot\,\rho \triangleright \zeta)}) \\ & = & \mathbb{Q}(\lambda\,\zeta\,v.\ [\![A]\!]^{\mathsf{T}}_{v\,\cdot((\xi\,\circ\,\rho) \triangleright \zeta)}) \\ & = & [\![\forall.\ A]\!]^{\mathsf{T}}_{\xi\,\circ\,\rho} \end{array}$$

□

Note that types form a graded domain with renaming as thinning. With Theorem 3.5 it follows that renaming is compatible with composition and hence thinning on types is compositional.

Corollary 3.6. $A\langle\xi\rangle\langle\zeta\rangle = A\langle\xi\,\circ\,\zeta\rangle$

Additionally, types are an injective graded domain and therefore functorial.

Lemma 3.7. *Renaming* $\_\langle\xi\rangle$ *under an injection* ξ *is injective.*

Proof. By induction on types. □

Corollary 3.8. $A\langle\mathsf{id}\rangle = A$

3.3 Lifting Traversals

There is an asymmetry in the definition of traversals which allows us to distinguish between semantic indices and semantic expressions. As argued in [3], this asymmetry is crucial to defining e.g. the renaming traversal. If both V and D have the structure of a graded domain then an asymmetric traversal can be lifted into a symmetric one.

Definition 3.9. Let $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$ be a ty-traversal for graded domains V, D. We define the *lifted traversal*

$$\overline{\mathsf{T}} := (\mathsf{id}, \mathbb{A}, \overline{\mathbb{Q}}) : \mathbb{T}^D_D$$

with

$$\overline{\mathbb{Q}} := \lambda\,F.\ \mathbb{Q}\,(\lambda\,\xi\,v.\ F\,\xi\,(\mathbb{V}\,v)).$$

A traversal is *uniform* if it is equal to its own lifting, $\mathsf{T} = \overline{\mathsf{T}}$. In particular, $\overline{\mathsf{T}}$ is always uniform since lifting is idempotent.

Since types form a graded domain with renaming as thinning, we can define instantiation as the lifting of renaming.

Definition 3.10. We define the *instantiation* of a type $A : \mathsf{ty}^m$ under a substitution $\sigma : I^m \to \mathsf{ty}^n$ as follows.

$$A[\sigma] := [\![A]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}}}_{\sigma}$$

Instantiation then satisfies the usual equations.

$$\begin{array}{rcl} x_{\mathsf{ty}}[\sigma] & = & \sigma\,x \\ (A \to B)[\sigma] & = & A[\sigma] \to B[\sigma] \\ (\forall.\ A)[\sigma] & = & \forall.\ A[0_{\mathsf{ty}} \cdot (\sigma\,\circ\,\_\langle\uparrow\rangle)] \end{array}$$

Under mild conditions, a traversal embeds into its lifting.

Definition 3.11. A semantic variable constructor $\mathbb{V} : \forall n.\ V^n \to D^n$ for two graded domains V and D is *natural* if it is compatible with thinning.

$$(\mathbb{V}\,x) \triangleright_D \xi = \mathbb{V}(x \triangleright_V \xi)$$

Theorem 3.12 (Embedding Lemma)**.** *Let* $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$ *be a* ty*-traversal where* V *and* D *are graded domains and* $\mathbb{V}$ *is natural. Then evaluation with respect to* T *is a special case of evaluation with respect to* $\overline{\mathsf{T}}$*.*

$$[\![A]\!]^{\mathsf{T}}_{\rho} = [\![A]\!]^{\overline{\mathsf{T}}}_{\rho\,\circ\,\mathbb{V}}$$

Proof. By induction on A. In the case of quantification we use that $\mathbb{V}$ is natural. □

3.4 Type Models

In Section 3.2, we proved that evaluation with respect to an arbitrary traversal is compatible with renaming. In this section, we extend traversals to the notion of a model by requiring every semantic constructor to be natural. Evaluation with respect to a model always respects instantiation.

Definition 3.13 (Model)**.** Let V, D be graded domains. A ty-traversal $\mathsf{T} = (\mathbb{V}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^V_D$ is a model if all operations are natural in the following sense.

- $\mathbb{V}$ is natural in the sense of Definition 3.11.
- $(\mathbb{A}\, A\, B) \triangleright_D \xi = \mathbb{A}\, (A \triangleright_D \xi)\, (B \triangleright_D \xi)$
- $\mathbb{Q}$ is natural for all natural F. We say that F is natural if $(F\, \zeta\, x) \triangleright_D \xi = F\, (\zeta \circ \xi)\, (x \triangleright_V \xi)$ for all ξ, ζ, x and $\mathbb{Q}$ is natural for F if

$$(\mathbb{Q}\, F) \triangleright_D \xi = \mathbb{Q}\, (F \circ (\xi \circ \_)).$$

The naturality conditions in the definition of a model are chosen to obtain a naturality statement for evaluation, which is essential for compatibility with instantiation.

Lemma 3.14. *For all models* $\mathsf{T} : \mathbb{T}^V_D$*, where* V, D *are graded domains and* V *is compositional, we have*

$$[\![A]\!]^{\mathsf{T}}_{\rho} \triangleright_D \xi = [\![A]\!]^{\mathsf{T}}_{\rho \triangleright_V \xi}.$$

Proof. By induction on A, using the corresponding naturality conditions in the definition of model. □

From Lemma 3.14 we obtain compatibility with instantiation.

Theorem 3.15 (Compatibility with instantiation)**.** *For all models* $\mathsf{T} : \mathbb{T}^V_D$*, where* V, D *are graded domains and* V *is compositional, we have*

$$[\![A[\sigma]]\!]^{\mathsf{T}}_{\rho} = [\![A]\!]^{\overline{\mathsf{T}}}_{[\![\sigma]\!]^{\mathsf{T}}_{\rho}}$$

where evaluation lifts point-wise to substitutions: $[\![\sigma]\!]^{\mathsf{T}}_{\rho} = \lambda n. [\![\sigma\, n]\!]^{\mathsf{T}}_{\rho}$.

Proof. By induction on A. In the case of quantification, we use Lemma 3.14. □

At this point we also note a frequently useful lemma, namely that the lifting of a model is itself a model.

Lemma 3.16. *For every model* T*, the traversal* $\overline{\mathsf{T}}$ *is a model.*

4 Case Study: Instantiation on Types

As a first example of our framework, we consider the instantiation operation on types. Since instantiation can be represented as a traversal it is automatically compatible with renaming (Theorem 3.5).

Corollary 4.1. $A\langle\xi\rangle[\sigma] = A[\xi \circ \sigma]$

Furthermore, it is easy to see that the renaming traversal is a model. Together with Lemma 3.16 we obtain the following theorems about instantiation.

Lemma 4.2. *The renaming traversal* R_{ty} *is a model.*

Proof. The cases of type variables and function types hold by definition. For quantification, we can assume that we are given a function F such that $(F\, \zeta\, x)\langle\xi\rangle = F\, (\zeta \circ \xi)\, (\xi x)$ holds and we have to show that $\mathbb{Q}$ is natural for F. This follows by equational reasoning.

$$\begin{aligned}(\mathbb{Q}\, F)\langle\xi\rangle &= (\forall.\ F \uparrow 0)\langle\xi\rangle \\ &= \forall.\ (F \uparrow 0)\langle 0_{\mathsf{ty}} \cdot \xi \circ \uparrow\rangle \\ &= \forall.\ F\, (\xi \circ \uparrow)\, 0 \\ &= \mathbb{Q}\, (\lambda\zeta.\ F(\xi \circ \zeta))\end{aligned}$$ □

Since the instantiation traversal corresponds to lifted renaming, renaming can be directly shown to be a special case of instantiation. Moreover, the core substitution lemmas of Autosubst hold.

Corollary 4.3 (Renaming as instantiation)**.**

$$A\langle\xi\rangle = A[\xi \circ \mathsf{id}_{ty}]$$

Proof. By the embedding theorem (Theorem 3.12). □

Corollary 4.4. $A[\mathsf{id}_{ty}] = A$

Proof. By Corollary 3.8, Corollary 4.3 and the embedding theorem (Theorem 3.12). □

Corollary 4.5. $A[\sigma][\sigma'] = A[\sigma \circ \_[\sigma']]$

Proof. By Theorem 3.15. □

In light of Corollary 4.4, we observe that Theorem 3.12 can be seen as a special case of Theorem 3.15 with slightly weaker assumptions.

5 Multivariate Traversals and Models

The definitions of traversals and models presented in Section 3 only apply to syntactic systems with a single sort and a single class of variables. We illustrate the generalisation to mutually recurive and multivariate sorts by extending our framework from the types of System F to its terms and values.

Both type and value variables appear, and we thus require semantic domains for type indices V_{ty} and value indices V_{vl}. Additionally, due to the mutual recursion between terms and values, the traversals for terms and values have to be defined as a single traversal structure with semantic domains for both term expressions D_{tm} and value expressions D_{vl}. Finally, terms and values contain types, which means that we also require a semantic domain for types D_{ty}.

The generalised definitions may look daunting at first, but the extensions are mostly mechanical. Well-scoped syntax is tremendously helpful when piecing together the various definitions. In the following we replicate the results of Section 3 without reiterating the corresponding motivations.

Definition 5.1. For given $V_{\mathsf{ty}}, V_{\mathsf{vl}}, D_{\mathsf{ty}} : \mathbb{N} \to \mathsf{Type}$ and $D_{\mathsf{vl}}, D_{\mathsf{tm}} : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}$, a tm/vl-*traversal*

$$\mathsf{E} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathbb{V}_{\mathsf{vl}}, \mathbb{L}_{\mathsf{tm}}, \mathbb{L}_{\mathsf{ty}}) : \mathbb{T}^{V_{\mathsf{ty}}, V_{\mathsf{vl}}}_{D_{\mathsf{ty}}, D_{\mathsf{tm}}, D_{\mathsf{vl}}}$$

$$
\begin{aligned}
\mathbb{I} &: \forall m\, n.\ D_{\mathsf{vl}}^{m,n} \to D_{\mathsf{tm}}^{m,n} \\
\mathbb{A}_{\mathsf{tm}} &: \forall m\, n.\ D_{\mathsf{tm}}^{m,n} \to D_{\mathsf{tm}}^{m,n} \to D_{\mathsf{tm}}^{m,n} \\
\mathbb{A}_{\mathsf{ty}} &: \forall m\, n.\ D_{\mathsf{tm}}^{m,n} \to D_{\mathsf{ty}}^{m} \to D_{\mathsf{tm}}^{m,n} \\
\mathbb{V}_{\mathsf{vl}} &: \forall m\, n.\ V_{\mathsf{vl}}^{m,n} \to D_{\mathsf{vl}}^{m,n} \\
\mathbb{L}_{\mathsf{tm}} &: \forall m\, n.\ D_{\mathsf{ty}}^{m} \to (D_{\mathsf{tm}}^{V_{\mathsf{vl}}})^{m,n} \to D_{\mathsf{vl}}^{m,n} \\
\mathbb{L}_{\mathsf{ty}} &: \forall m\, n.\ (D_{\mathsf{tm}}^{\hat{V}_{\mathsf{ty}}})^{m,n} \to D_{\mathsf{vl}}^{m,n}
\end{aligned}
$$

where

$$
\begin{aligned}
\hat{V}_{\mathsf{ty}}^{m,n} &:= V_{\mathsf{ty}}^{m} \\
(P^{Q})^{m,n} &:= \forall m'\, n'.\ (I^{m} \hookrightarrow I^{m'}) \to (I^{n} \hookrightarrow I^{n'}) \to Q^{m',n'} \to P^{m',n'}
\end{aligned}
$$

$$
\begin{aligned}
[\![s\ t]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{A}_{\mathsf{tm}}\, [\![s]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}}\, [\![t]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} \\
[\![s\ A]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{A}_{\mathsf{ty}}\, [\![s]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}}\, [\![A]\!]_{\rho}^{\mathsf{T}} \\
[\![v]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{I}\, [\![v]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} \\
\\
[\![x_{\mathsf{vl}}]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{V}_{\mathsf{vl}}(\theta\, x) \\
[\![\lambda A.\ s]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{L}_{\mathsf{tm}}\, [\![A]\!]_{\rho}^{\mathsf{T}}\, (\lambda \xi\, \zeta\, v.\ [\![s]\!]_{\rho \triangleright\, \xi,\, v\, \cdot (\theta \triangleright (\xi,\zeta))}^{\mathsf{T},\mathsf{E}}) \\
[\![\Lambda.\ s]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} &:= \mathbb{L}_{\mathsf{ty}}\, (\lambda \xi\, \zeta\, v.\ [\![s]\!]_{v\, \cdot (\rho \triangleright\, \xi),\, \theta \triangleright (\xi,\zeta)}^{\mathsf{T},\mathsf{E}})
\end{aligned}
$$

Figure 2. Semantic constructors for terms and values of $\mathrm{F}_{\mathrm{CBV}}$ (left) and definition of evaluation (right).

for terms and values of $\mathrm{F}_{\mathrm{CBV}}$ consists of semantic counterparts to the syntactic constructors (see Figure 2 for the corresponding types).

Given a traversal structure E for terms and values as well as a traversal structure T for types, we define evaluation with respect to T, E by mutual recursion for both terms and values. As for types, this definition requires a notion of thinning on semantic (value) indices. Since the types differ we have to adapt the notion of graded domains to this setting. This adaption is completely mechanical (cf. Definition 3.2).

Definition 5.2. A *bigraded domain* consists of a family of types $V : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}$ together with a thinning operation

$$\_ \triangleright (\_, \_) \ :\ V^{m,n} \to (I^{m} \hookrightarrow I^{m'}) \to (I^{n} \hookrightarrow I^{n'}) \to V^{m',n'}.$$

We again lift the thinning operation to environments ρ and write $\rho \triangleright (\xi, \zeta)$ for $\rho \circ (\_ \triangleright (\xi, \zeta))$.

A bigraded domain is *compositional*, if thinning respects compositions $x \triangleright (\xi, \zeta) \triangleright (\xi', \zeta') = x \triangleright (\xi \circ \xi', \zeta \circ \zeta')$, *functorial*, if it is compositional and thinning respects identities $x \triangleright (\mathsf{id}, \mathsf{id}) = x$, and *injective* if thinning $\_ \triangleright (\xi, \zeta)$ is injective for all ξ, ζ.

Definition 5.3. Let $\mathsf{T} = (\mathbb{V}_{\mathsf{ty}}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}_{D_{\mathsf{ty}}}^{V_{\mathsf{ty}}}$ be a ty-traversal and $\mathsf{E} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathbb{V}_{\mathsf{vl}}, \mathbb{L}_{\mathsf{tm}}, \mathbb{L}_{\mathsf{ty}}) : \mathbb{T}_{D_{\mathsf{ty}}, D_{\mathsf{tm}}, D_{\mathsf{vl}}}^{V_{\mathsf{ty}}, V_{\mathsf{vl}}}$ be a tm/vl-traversal, where V_{ty} and V_{vl} are a graded and bigraded domains respectively. Evaluation $[\![\_]\!]_{\_,\_}^{\mathsf{T},\mathsf{E}}$ with respect to T, E is defined by mutual recursion.

$$
\begin{aligned}
[\![\_]\!]_{\_,\_}^{\mathsf{T},\mathsf{E}} &: \mathsf{tm}^{m,n} \to (I^{m} \to V_{\mathsf{ty}}^{m'}) \to (I^{n} \to V_{\mathsf{vl}}^{n'}) \to D_{\mathsf{tm}}^{m',n'} \\
[\![\_]\!]_{\_,\_}^{\mathsf{T},\mathsf{E}} &: \mathsf{vl}^{m,n} \to (I^{m} \to V_{\mathsf{ty}}^{m'}) \to (I^{n} \to V_{\mathsf{vl}}^{n'}) \to D_{\mathsf{vl}}^{m',n'}
\end{aligned}
$$

The defining equations are given in Figure 2. The two evaluation functions can be disambiguated by the type of their first argument.

The following aspects are worth pointing out. First, whenever we reach a variable we have to project the correct component of the vector of environments, e.g.

$$[\![x_{\mathsf{vl}}]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}} = \mathbb{V}_{\mathsf{vl}}(\theta\, x)$$

for value variables.

Second, when a given subexpression is of a different sort, we have to select the correct evaluation function, traversal structures and environments. Take for example $[\![s\ A]\!]_{\rho,\theta}^{T,E}$ where the correct environment for instantiating the subterm A is ρ. The change of evaluation functions also occurs on the embedding of values into terms $[\![v]\!]_{\rho,\theta}^{\mathsf{T},\mathsf{E}}$, where we use v to both denote the actual value, as well as its embedding into the sort of terms.

Third, and most interesting, the traversal of binders changes the interpretation of indices in scope. We have to adjust the full environment which is more involved than in the single-sorted setting. The component that corresponds to the sort of the binder we just traversed, say ρ, is modified almost as before. While the index 0^{k+1} is mapped to the newly bound value v as usual, we have to ensure that $\uparrow x$ is mapped to $\rho\, x$ and then adjusted according to the scope change of the vector renaming (ξ, ζ). The postcomposed adjustment may have to be cast to the appropriate subvector. Consider for example the type component, where (ξ, ζ) is simplified to ξ.

As before, certain naturality conditions on traversals are necessary to obtain compatibility with instantiation.

Definition 5.4. A tm/vl-traversal is a model if every semantic constructor is natural with respect to the graded and bigraded domains $V_{\mathsf{ty}}, D_{\mathsf{ty}}, V_{\mathsf{vl}}, D_{\mathsf{vl}}, D_{\mathsf{tm}}$. For example, for $\mathbb{I}$ we must have $(\mathbb{I}\, v) \triangleright_{D_{\mathsf{tm}}} \xi = \mathbb{I}(v \triangleright_{D_{\mathsf{vl}}} \xi)$.

We omit a complete listing of the naturality conditions for space reasons.

Finally the construction of lifting and renaming proceeds as before.

Definition 5.5. Let $\mathsf{T} = (\mathbb{V}_{\mathsf{ty}}, \mathbb{A}, \mathbb{Q}) : \mathbb{T}^{V_{\mathsf{ty}}}_{D_{\mathsf{ty}}}$ be a ty-traversal and $\mathsf{E} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathbb{V}_{\mathsf{vl}}, \mathbb{L}_{\mathsf{tm}}, \mathbb{L}_{\mathsf{ty}}) : \mathbb{T}^{V_{\mathsf{ty}}, V_{\mathsf{vl}}}_{D_{\mathsf{ty}}, D_{\mathsf{tm}}, D_{\mathsf{vl}}}$ be a tm/vl-traversal, where $V_{\mathsf{ty}}, V_{\mathsf{vl}}, D_{\mathsf{ty}}, D_{\mathsf{tm}}, D_{\mathsf{vl}}$ are graded and bigraded domains respectively.

We define the *lifted traversal*

$$\overline{\mathsf{E}} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathsf{id}, \mathbb{L}'_{\mathsf{tm}}, \mathbb{L}'_{\mathsf{ty}}) : \mathbb{T}^{D_{\mathsf{ty}}, D_{\mathsf{vl}}}_{D_{\mathsf{ty}}, D_{\mathsf{tm}}, D_{\mathsf{vl}}}$$

where $\mathbb{L}'_{\mathsf{tm}}, \mathbb{L}'_{\mathsf{ty}}$ are defined as follows.

$$\begin{aligned} \mathbb{L}'_{\mathsf{tm}} &:= \lambda A F.\, \mathbb{L}_{\mathsf{tm}}\, A\, (\lambda \xi\, \zeta\, v.\, F\, \xi\, \zeta\, (\mathbb{V}_{\mathsf{vl}}\, v)) \\ \mathbb{L}'_{\mathsf{ty}} &:= \lambda F.\, \mathbb{L}_{\mathsf{ty}} (\lambda \xi\, \zeta\, v.\, F\, \xi\, \zeta\, (\mathbb{V}_{\mathsf{ty}}\, v)) \end{aligned}$$

The renaming traversal simply uses the syntactic constructors as semantic constructors.

Definition 5.6 (Renaming)**.** We define the *renaming traversal* $\mathsf{R}_{\mathsf{vl}} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathbb{V}_{\mathsf{vl}}, \mathbb{L}_{\mathsf{tm}}, \mathbb{L}_{\mathsf{ty}})$ where $V_{\mathsf{ty}} = I$, $D_{\mathsf{ty}} = \mathsf{ty}$, $V_{\mathsf{vl}}\, m\, n = I^n$, $D_{\mathsf{vl}} = \mathsf{vl}$, and $D_{\mathsf{tm}} = \mathsf{tm}$.

$$\begin{aligned} \mathbb{V}_{\mathsf{vl}}\, x &:= x & \mathbb{A}_{\mathsf{tm}}\, s\, t &:= s\, t \\ \mathbb{A}_{\mathsf{ty}}\, s\, A &:= s\, A & \mathbb{I}\, v &:= v \\ \mathbb{L}_{\mathsf{tm}}\, A\, F &:= \lambda A.\, F\, \mathsf{id} \uparrow 0 & \mathbb{L}_{\mathsf{ty}}\, F &:= \Lambda.\, F \uparrow \mathsf{id}\, 0 \end{aligned}$$

As before, we define renaming as evaluation with the renaming traversal.

$$s\langle \xi, \zeta \rangle := [\![s]\!]^{\mathsf{R}_{\mathsf{ty}}, \mathsf{R}_{\mathsf{vl}}}_{\xi, \zeta}$$

Terms and values form a bigraded domain under renaming and we define instantiation as evaluation with the lifted renaming traversal.

$$s[\sigma, \tau] := [\![s]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}}, \overline{\mathsf{R}_{\mathsf{vl}}}}_{\sigma, \tau}$$

In particular, instantiation under a type substitution σ and a value substitution τ satisfies the following equations.

$$\begin{aligned} x[\sigma, \tau] &= \tau\, v \\ (s\, t)[\sigma, \tau] &= s[\sigma, \tau]\, t[\sigma, \tau] \\ (s\, A)[\sigma, \tau] &= s[\sigma, \tau]\, A[\sigma] \\ (\lambda A.\, s)[\sigma, \tau] &= \lambda A[\sigma].\, s[\sigma, 0 \cdot (\tau \triangleright (\mathsf{id}, \uparrow)] \\ (\Lambda.\, s)[\sigma, \tau] &= \Lambda.\, s[0 \cdot (\sigma \triangleright \uparrow), \tau \triangleright (\uparrow, \mathsf{id})] \end{aligned}$$

This reproduces the definition of instantiation for $\mathrm{F}_{\mathrm{CBV}}$ from [14]. We proceed as before and show that every traversal is compatible with renaming.

Theorem 5.7. *Let* T *and* E *be a* ty*-traversal and a* tm/vl*-traversal respectively. For all terms* s*, renamings* ξ, ζ *and environments* ρ, ρ' *we have*

$$[\![s \triangleright (\xi, \zeta)]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'} = [\![s]\!]^{\mathsf{T}, \mathsf{E}}_{\xi \circ \rho, \zeta \circ \rho'}.$$

If the semantic variable constructors are natural, we obtain another embedding lemma.

Theorem 5.8. *Let* T *be a* ty*-traversal with the semantic variable constructor* $\mathbb{V}_{\mathsf{ty}}$ *and* E *a* tm/vl*-traversal with the semantic variable constructor* $\mathbb{V}_{\mathsf{vl}}$ *such that both* $\mathbb{V}_{\mathsf{ty}}$ *and* $\mathbb{V}_{\mathsf{vl}}$ *are natural. For all terms and values* s *and environments* ρ, ρ' *we have*

$$[\![s]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'} = [\![s]\!]^{\overline{\mathsf{T}}, \overline{\mathsf{E}}}_{\rho \circ \mathbb{V}_{\mathsf{ty}}, \rho \circ \mathbb{V}_{\mathsf{vl}}}.$$

Finally, we obtain compatibility with instantiation for models as desired.

Theorem 5.9. *Let* T *and* E *be a type model and a term and value model respectively, for which the graded domain of types* D_{ty} *and values* D_{vl} *are compositional. For all terms and values* s*, type substitutions* σ*, value substitutions* τ *and environments* ρ, ρ' *we have*

$$[\![s[\sigma, \tau]]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'} = [\![s]\!]^{\overline{\mathsf{T}}, \overline{\mathsf{E}}}_{[\![\sigma]\!]^{\mathsf{T}}_{\rho}, [\![\tau]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'}}$$

where $[\![\tau]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'} = \lambda n. [\![\tau\, n]\!]^{\mathsf{T}, \mathsf{E}}_{\rho, \rho'}$.

6 Case Study: Instantiation Laws for $\mathrm{F}_{\mathrm{CBV}}$

Analogous to Section 4, we consider the instantiation operation on terms and values.

Using Theorem 5.7, we show that renaming on $\mathrm{F}_{\mathrm{CBV}}$ terms and values is compatible with instantiation.

Corollary 6.1. *For all terms and values* s *we have*

$$s\langle \xi, \zeta \rangle[\sigma, \tau] = s[\xi \circ \sigma, \zeta \circ \tau].$$

For the remainder we note that the renaming traversal is natural, and therefore a model. The proof is analogous to Lemma 4.2.

Lemma 6.2. *The renaming traversal* R_{vl} *is a model.*

Using Lemma 6.2 we instantiate the theorems from Section 5 for term and value instantiation.

Corollary 6.3. $s\langle \xi, \zeta \rangle = s[\xi \circ \mathsf{id}_{ty}, \zeta \circ \mathsf{id}_{vl}]$

Proof. By Theorem 5.8. □

Corollary 6.4. $s[\mathsf{id}_{ty}, \mathsf{id}_{vl}] = s$

Proof. By Corollary 6.3 with functoriality of renaming. □

Corollary 6.5. $s[\sigma, \tau][\sigma', \tau'] = s[\sigma \circ \_[\sigma'], \tau \circ \_[\sigma', \tau']]$

Proof. By Theorem 5.9. □

These theorems are the main substitution lemmas needed for the equational theory of Autosubst 2 [14].

7 Case Study: System F Type Preservation

In Figure 3 we present the typing judgment of $\mathrm{F}_{\mathrm{CBV}}$ as an inductive type. Formally, the type system consists of two mutually inductive predicates.

$$\begin{aligned} \_ \vdash \_ : \_ &: (I^n \to \mathsf{ty}^m) \to \mathsf{tm}^{m,n} \to \mathsf{ty}^m \to \mathsf{Prop} \\ \_ \vdash^v \_ : \_ &: (I^n \to \mathsf{ty}^m) \to \mathsf{vl}^{m,n} \to \mathsf{ty}^m \to \mathsf{Prop} \end{aligned}$$

$$\frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\,t : B} \qquad \frac{\Gamma \vdash s : \forall.\ A}{\Gamma \vdash s\,B : A[B \cdot \mathrm{id}_{ty}]} \qquad \frac{\Gamma \Vdash^{v} v : A}{\Gamma \vdash v : A}$$

$$\frac{}{\Gamma \Vdash^{v} x_{\mathsf{vl}} : \Gamma_x} \qquad \frac{A \cdot \Gamma \vdash s : B}{\Gamma \Vdash^{v} \lambda A.\ s : A \to B} \qquad \frac{\Gamma \rhd \uparrow \vdash s : A}{\Gamma \Vdash^{v} \Lambda.\ s : \forall.\ A}$$

$$\frac{s \Downarrow \lambda A.\ b \quad t \Downarrow u \quad b[\mathrm{id}_{ty}, u \cdot \mathrm{id}_{vl}] \Downarrow v}{s\,t \Downarrow v} \qquad \frac{s \Downarrow \Lambda.\ b \quad b[A \cdot \mathrm{id}_{ty}, \mathrm{id}_{vl}] \Downarrow v}{s\,A \Downarrow v} \qquad \frac{}{v \Downarrow v}$$

Figure 3. Type System and big-step evaluation relation for $\mathrm{F}_{\mathrm{CBV}}$.

Since typing in $\mathrm{F}_{\mathrm{CBV}}$ is syntax directed we can alternatively define it recursively using a traversal. This traversal turns out to be natural. We thus obtain structural properties of the typing relation.

Definition 7.1. We define the typing traversal

$$\mathsf{J} = (\mathbb{I}, \mathbb{A}_{\mathsf{tm}}, \mathbb{A}_{\mathsf{ty}}, \mathbb{V}_{\mathsf{vl}}, \mathbb{L}_{\mathsf{tm}}, \mathbb{L}_{\mathsf{ty}}) : \mathbb{T}^{\mathsf{ty},V}_{\mathsf{ty},D,D}$$

where value indices are interpreted by $V^{m,n} := \mathsf{ty}^m$ and term expressions are interpreted by $D^{m,n} := \mathsf{ty}^m \to \mathsf{Prop}$ as follows

$$\begin{aligned}
\mathbb{I}\,P &:= P \\
B \in \mathbb{A}_{\mathsf{tm}}\,P\,Q &:= \exists A.\ P(A \to B) \wedge Q\,A \\
C \in \mathbb{A}_{\mathsf{ty}}\,P\,A &:= \exists B.\ P(\forall.\ B) \wedge C = B[A \cdot \mathrm{id}_{ty}] \\
B \in \mathbb{V}_{\mathsf{vl}}\,A &:= (A = B) \\
(A \to B) \in \mathbb{L}_{\mathsf{tm}}\,C\,F &:= (A = C) \wedge F\,\mathrm{id}\,\mathrm{id}\,A\,B \\
(\forall.\ A) \in \mathbb{L}_{\mathsf{ty}}\,F &:= A \in F \uparrow \mathrm{id}\,0
\end{aligned}$$

We call evaluation with the typing traversal recursive typing and use the following abbreviations.

$$\begin{aligned}
\mathrm{tp}_{\mathsf{vl}}\,\Gamma\,v\,A &= A \in [\![v]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}},\mathsf{J}}_{\mathrm{id}_{ty},\Gamma} \\
\mathrm{tp}_{\mathsf{tm}}\,\Gamma\,a\,A &= A \in [\![s]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}},\mathsf{J}}_{\mathrm{id}_{ty},\Gamma}
\end{aligned}$$

Note that the typing traversal is not uniform, since we interpret value indices as types, but value expressions as predicates on types. We note some properties of the definition of tp.

$$\begin{aligned}
\mathrm{tp}_{\mathsf{tm}}\,\Gamma\,v\,A &= \mathrm{tp}_{vl}\,\Gamma\,v\,A \\
\mathrm{tp}_{\mathsf{tm}}\,\Gamma\,(s\,t)\,B &= \exists A.\ \mathrm{tp}_{\mathsf{tm}}\,\Gamma\,s\,(A \to B) \wedge \mathrm{tp}_{\mathsf{tm}}\,\Gamma\,t\,A \\
\mathrm{tp}_{\mathsf{tm}}\,\Gamma\,(s\,A)\,C &= \exists B.\ \mathrm{tp}_{\mathsf{tm}}\,\Gamma\,s\,(\forall.\ B) \wedge C = B[A \cdot \mathrm{id}_{ty}] \\
\mathrm{tp}_{\mathsf{vl}}\,\Gamma\,x\,A &= (\Gamma\,x = A) \\
\mathrm{tp}_{\mathsf{vl}}\,\Gamma\,(\lambda A.\ s)\,(A \to B) &= \mathrm{tp}_{\mathsf{tm}}\,(A \cdot \Gamma)\,s\,B \\
\mathrm{tp}_{\mathsf{vl}}\,\Gamma\,(\Lambda.\ s)\,(\forall.\ A) &= \mathrm{tp}_{\mathsf{tm}}\,(\Gamma \rhd \uparrow)\,s\,A
\end{aligned}$$

Based on these properties, it is easy to see that tp coincides with the inductive typing relation.

Lemma 7.2. *Inductive and recursive typing coincide.*

$$\begin{aligned}
\mathrm{tp}_{\mathsf{vl}}\,\Gamma\,v\,A &= (\Gamma \vdash^{v} v : A) \\
\mathrm{tp}_{\mathsf{tm}}\,\Gamma\,s\,A &= (\Gamma \vdash s : A)
\end{aligned}$$

In the remainder of this section, we state all results in terms of the more familiar inductive typing judgment.

From the definition as a traversal we obtain both weakening and strengthening.

Corollary 7.3 (Weakening & Strengthening)**.**

$$(A \cdot \Gamma \vdash s\langle \mathrm{id}, \uparrow\rangle : B) \;\leftrightarrow\; (\Gamma \vdash s : B).$$

Proof. By Theorem 5.7. □

Lemma 7.4 (Naturality of Typing)**.** *The typing traversal is natural.*

Proof. By case analysis, using properties of renaming. Note that D is an injective bigraded domain with the action given by the image under renaming of types. Most cases are straightforward, but the application case depends crucially on the injectivity of the renaming. □

We obtain weakening and strengthening for type variables as a consequence of naturality.

Corollary 7.5 (Type Weakening and Strengthening)**.**

$$(\Gamma \rhd \uparrow \vdash s\langle \uparrow, \mathrm{id}\rangle : (A \rhd \uparrow)) \;\leftrightarrow\; (\Gamma \vdash s : A)$$

For all further consequences, we consider evaluation in the lifted typing traversal.

Definition 7.6. We write $\mathrm{Tp}\,R\,s\,A$ for evaluation in the lifted typing traversal, where $R : I^n \to \mathsf{ty}^m \to \mathsf{Prop}$ is a relation between variables and types.

$$\begin{aligned}
\mathrm{Tp}_{\mathsf{vl}}\,R\,v\,A &:= A \in [\![v]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}},\bar{\mathsf{J}}}_{\mathrm{id}_{ty},R} \\
\mathrm{Tp}_{\mathsf{tm}}\,R\,s\,A &:= A \in [\![s]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}},\bar{\mathsf{J}}}_{\mathrm{id}_{ty},R}
\end{aligned}$$

We refer to Tp as relational typing. In some instances it is useful to also vary the type substitution environment and we define

$$\mathrm{Tp}^{\sigma}\,R\,s\,A := A \in [\![s]\!]^{\overline{\mathsf{R}_{\mathsf{ty}}},\bar{\mathsf{J}}}_{\sigma,R}.$$

Apart from the cases for binders and variables, the definitions of recursive and relational typing coincide. For variables and binders, relational typing satisfies the following equations.

$$\begin{aligned}
\mathrm{Tp}_{\mathsf{vl}}\,R\,x\,A &= R\,x\,A \\
\mathrm{Tp}_{\mathsf{vl}}\,R\,(\lambda A.\ s)\,(A \to B) &= \mathrm{Tp}_{\mathsf{tm}}\,((A = _) \cdot R)\,s\,B
\end{aligned}$$

Since the typing traversal is natural, relational typing extends the ordinary typing judgment, and is compatible with instantiation in a stronger sense than what is possible for the typing judgment.

Corollary 7.7. $\Gamma \vdash s : A \;=\; \mathrm{Tp}\,(\Gamma\ \_ = \_)\ s\ A$

Proof. By Theorem 5.8. □

Corollary 7.8. $\mathrm{Tp}\,R\,s[\mathrm{id}_{ty}, \tau]\ A \;\leftrightarrow\; \mathrm{Tp}\,(\tau \circ \mathrm{Tp}\,R)\ s\ A$

Proof. By Theorem 5.9. □

Before we can show type preservation for $\mathrm{F}_{\mathrm{CBV}}$, we need a monotonicity lemma for relational typing. Relational typing is obviously monotone in the relational context, since it is only ever extended or used in the variable case. We further generalise the statement of monotonicity by considering types as ordered under specialisation.

Lemma 7.9 (Monotonicity)**.** *For all relations R, S and type substitutions σ such that*

$$R\,x\,A \;\to\; S\,x\,A[\sigma]$$

we have

$$\mathrm{Tp}\,R\,s\,A \;\to\; \mathrm{Tp}^{\sigma}\,S\,s\,A[\sigma].$$

Proof. By induction on s and equational reasoning. □

We now proceed to show type preservation of $\mathrm{F}_{\mathrm{CBV}}$.

Theorem 7.10. *For all terms s and values v such that $s \Downarrow v$, we have*

$$\Gamma \vdash s : A \;\to\; \Gamma \vdash^{v} v : A$$

Proof. By Corollary 7.7, it suffices to show the statement for relational typing.

$$\mathrm{Tp}_{\mathsf{tm}}\,R\,s\,A \;\to\; \mathrm{Tp}_{\mathsf{vl}}\,R\,v\,A$$

We proceed by induction on the derivation of $s \Downarrow v$. The case for values is trivial. For an application $\mathrm{Tp}_{\mathsf{tm}}\,R\,(s\,t)\,B$ we know that $\mathrm{Tp}_{\mathsf{tm}}\,R\,s\,(A \to B)$ and $\mathrm{Tp}\,R\,t\,A$ hold for some A. From the definition of evaluation, we have $s \Downarrow (\lambda A.\ b)$, $t \Downarrow u$, and $b[\mathrm{id}_{ty}, u \cdot \mathrm{id}_{vl}] \Downarrow v$. Hence by induction, we have $\mathrm{Tp}_{\mathsf{vl}}\,R\,u\,A$ and $\mathrm{Tp}_{\mathsf{vl}}\,R\,(\lambda A.\ b)\,(A \to B)$, or equivalently, $\mathrm{Tp}_{\mathsf{tm}}\,((A = \_) \cdot R)\,b\,B$.

We have to show that $\mathrm{Tp}_{\mathsf{vl}}\,R\,v\,B$ holds, and by induction it suffices to show $\mathrm{Tp}_{\mathsf{tm}}\,R\,b[\mathrm{id}_{ty}, u \cdot \mathrm{id}_{vl}]\,B$. At this point we are in a position to use Corollary 7.8. We have

$$\mathrm{Tp}\,R\,b[\mathrm{id}_{ty}, u \cdot \mathrm{id}_{vl}]\,B \;=\; \mathrm{Tp}\,(\mathrm{Tp}\,R\,u \cdot R)\,b\,B$$

and the result follows from Lemma 7.9, since we already have $\mathrm{Tp}_{\mathsf{tm}}\,((A = \_) \cdot R)\,b\,B$ and $\mathrm{Tp}\,R\,u\,A$.

The proof of type application is analogous. □

Type preservation is a well-studied problem and the proof of Theorem 7.10 usually requires a *context morphism lemma* such as the following.

Lemma 7.11 (Context Morphism Lemma)**.** *For all contexts Γ, Δ and substitutions σ, τ such that*

$$\forall x.\ \Delta \vdash \tau\,x : (\Gamma\,x)[\sigma]$$

we have

$$\Gamma \vdash s : A \;\to\; \Delta \vdash s[\sigma, \tau] : A[\sigma].$$

Proof. By Corollary 7.7 and Theorem 5.9 it suffices to show that $\mathrm{Tp}^{\sigma}\,(\tau \circ (\Delta \vdash \_ : \_))\,s\,A[\sigma]$ holds. This follows from monotonicity (Lemma 7.9), since we have $\mathrm{Tp}\,(\Gamma\_ = \_)\,s\,A$ by Corollary 7.7, and the assumption on Γ, Δ is exactly what we need to satisfy the premise of Lemma 7.9. □

From the proof of Lemma 7.11 we see that the context morphism lemma factors into monotonicity and the compatibility with instantiation for relational typing. What is striking about this development is that a proof of a context morphism lemma typically requires several inductions, since we have to follow the recursive structure of the instantiation operation [15].

It is also usually difficult to obtain strengthening results, such as the ones we obtain for typing in the left-to-right directions of Corollary 7.3 and 7.5. We obtain strengthening almost for free, simply by formulating typing as a traversal.

The left-to-right direction of Corollary 7.8 is similarly unusual and only possible with relational typing. Consider the problem of establishing a judgement $\Gamma \vdash s[\mathrm{id}_{ty}, \tau] : A$. It is not sufficient to type check s in a context compatible with τ as in the premise of Lemma 7.11. In general, such a context might fail to exist, even when the term $s[\mathrm{id}_{ty}, \tau]$ is typable. This problem vanishes for relational typing, since the relational context allows us to assign no types to some variables.

In this case, the lifting construction has identified an extension of the ordinary typing judgment which is arguably better behaved. For more evidence of this, note that the proofs of Theorem 7.10 and Lemma 7.11 start by invoking the embedding theorem and showing the result for relational typing. In particular, type preservation also holds for relational typing and there seems to be no need for a separate context morphism lemma, since relational typing is more strictly compatible with instantiation.

8 Simple Traversals and Liftings

To obtain a model (Definition 3.13), users have to provide both a traversal and various naturality proofs. For recursive functions that do not reinterpret variables after the traversal of a binder, the naturality proofs can be omitted. In this section, we present the subclass of *simple* traversals that are sufficient to handle these scenarios. Simple traversals turn out to be compatible with both renaming and instantiation without additional user-provided naturality proofs.

Definition 8.1 (Simple Traversal)**.** A *simple type traversal structure* $\mathsf{T} = (\mathbb{V}_S, \mathbb{A}_S, \mathbb{Q}_S) : \mathbb{T}^{V_S}_{D_S}$ consists of non-dependent semantic counterparts to the syntactic constructors.

$$\begin{array}{lcl} \mathbb{V}_S & : & V_S \to D_S \\ \mathbb{A}_S & : & D_S \to D_S \to D_S \\ \mathbb{Q}_S & : & (V_S \to D_S) \to D_S \end{array}$$

We can easily turn the types V_S and D_S into constant graded domains $V := \lambda\_.\ V_S$ and respectively $D := \lambda\_.\ D_S$.

This allow us to embed simple traversals into regular traversals.

Definition 8.2 (Embedding of Simple Traversals)**.** Let $\mathsf{T} = (\mathbb{V}_S, \mathbb{A}_S, \mathbb{Q}_S) : \mathbb{T}^{V_s}_{D_s}$ be a simple traversal. Then T is a special case of the regular traversal $\mathsf{T}' : \mathbb{T}^{\lambda\_.\ V_S}_{\lambda\_.\ D_S}$, where the non-binding semantic constructors simply ignore the index. The semantic constructor for quantification uses the identity renaming for rescoping:

$$\mathbb{Q} := \lambda F.\mathbb{Q}_S\ (\lambda v.F\ \mathsf{id}\ v)$$

In the remaining paper, we often leave this coercion implicit.

Most importantly, all simple traversals are models w.r.t. to the constant action on D.

Lemma 8.3. *A simple traversal is a model w.r.t. the constant action on D.*

Proof. Most conditions follow as the action has no effect on the result type. In the case of a binder, we require functional extensionality and naturality of F. □

Corollary 8.4. *Let $\mathsf{T} = (\mathbb{V}_S, \mathbb{A}_S, \mathbb{Q}_S) : \mathbb{T}^{V_s}_{D_s}$ be a simple traversal. Then $\overline{\mathsf{T}}$ is compatible with instantiation.*

See Sections 9 and 10 for examples where simple traversals simplify the proof structure significantly. The above results can be easily extended to the multivariate case.

9 Case Study: Weak Normalisation of System $\mathbf{F}_{\mathbf{CBV}}$

We apply the results of Section 8 to show weak normalisation of System F with call-by-value reduction. See Figure 3 for the definition of typing and reduction. The proof proceeds by constructing a unary logical relation over F types which can be seen as a compositional extension of the notion of weak normalisation. We build the logical relation as a simple traversal and obtain all of the necessary structural properties for free.

Definition 9.1. We define the uniform simple type traversal $\mathsf{W} = (\mathsf{id}, \mathbb{A}_R, \mathbb{Q}_R) : \mathbb{T}^{\mathcal{P}\ \mathsf{vl}^{0,0}}_{\mathcal{P}\ \mathsf{vl}^{0,0}}$ interpreting types by sets of closed values where

$$\begin{aligned}(\lambda C.\ b) \in \mathbb{A}_R\ P\ Q &:= \forall v \in P.\ b[\mathsf{id}_{ty}, v \cdot \mathsf{id}_{vl}] \in L\ Q\\ (\Lambda.\ b) \in \mathbb{Q}_R\ F &:= \forall PA.\ b[A \cdot \mathsf{id}_{ty}, \mathsf{id}_{vl}] \in L(F\ P)\\ s \in L\ P &:= \exists v.\ s \Downarrow v \wedge P\ v.\end{aligned}$$

We use the following abbreviations for W evaluation and its extension to sets of closed terms.

$$\begin{aligned}\mathcal{V}\ A\ \rho &:= [\![A]\!]^{\mathsf{W}}_{\rho}\\ \mathcal{E}\ A\ \rho &:= L(\mathcal{V}\ A\ \rho)\end{aligned}$$

By definition of evaluation we have the following recursive equations for $\mathcal{V}, \mathcal{E}$.

$$\begin{aligned}v \in \mathcal{V}\ X\ \rho &= v \in \rho\ X\\ (\lambda C.\ b) \in \mathcal{V}(A \to B)\rho &= \forall v \in \mathcal{V}\ A\ \rho.\ b[\mathsf{id}_{ty}, v \cdot \mathsf{id}_{vl}] \in \mathcal{E}\ B\ \rho\\ (\Lambda.\ b) \in \mathcal{V}(\forall.\ A)\rho &= \forall PA.\ b[A \cdot \mathsf{id}_{ty}, \mathsf{id}_{vl}] \in \mathcal{E}\ A\ (P \cdot \rho)\\ s \in \mathcal{E}\ A\ \rho &= \exists v.\ s \Downarrow v \wedge v \in \mathcal{V}\ A\ \rho\end{aligned}$$

The corresponding weakening and substitution lemmas follow from Theorem 8.4.

Corollary 9.2.

$$\begin{aligned}\mathcal{V}\ A\langle\uparrow\rangle\ (P \cdot \rho) &= \mathcal{V}\ A\ \rho\\ \mathcal{V}\ A[\sigma]\ \rho &= \mathcal{V}\ A\ (\lambda i.\ \mathcal{V}(\sigma\ i)\rho)\end{aligned}$$

We proceed by extending the interpretation of types to open values and terms by quantifying over all closing substitutions.

Definition 9.3. For typing contexts $\Gamma : I^n \to ty^m$ and environments $\rho : I^m \to \mathcal{P}\ \mathsf{vl}^{0,0}$ we define

$$\sigma \in \mathcal{C}\ \Gamma\ \rho = \forall i.\ \sigma\ i \in \mathcal{V}\ (\Gamma\ i)\ \rho$$

given a type $A : ty^m$ we defined the semantic value and term typing as

$$\begin{aligned}\Gamma \vDash^v v : A &:= \forall \sigma\tau\rho.\ \sigma \in \mathcal{C}\ \Gamma\ \rho \to v[\sigma, \tau] \in \mathcal{V}\ A\ \rho\\ \Gamma \vDash s : A &:= \forall \sigma\tau\rho.\ \sigma \in \mathcal{C}\ \Gamma\ \rho \to s[\sigma, \tau] \in \mathcal{E}\ A\ \rho.\end{aligned}$$

Theorem 9.4. *Syntactic typing implies semantic typing.*

$$\begin{aligned}\Gamma \vdash^v v : A &\to \Gamma \vDash^v v : A\\ \Gamma \vdash s : A &\to \Gamma \vDash s : A\end{aligned}$$

Proof. By induction on the typing derivation. The cases for variables, application and the embedding of values into terms hold by definition. The case for lambda abstractions follows from general facts about instantiation, while the cases for type application and type abstraction require both reasoning about instantiation as well as Corollary 9.2. □

For closed terms, the premise of semantic typing is trivially satisfied and we obtain weak normalisation of well-typed terms.

Corollary 9.5. *Closed, well-typed terms evaluate to values.*

$$\vdash s : A \to \exists v.\ s \Downarrow v$$

What is striking about this proof of normalisation is that it rests entirely on our general theory of traversals and requires no specific lemmas. Once all definitions are in place, the proof of Theorem 9.4 proceeds directly, using only general facts about instantiation and evaluation.

10 Case Study: Strong Normalisation of System F

In this chapter we outline a proof of strong normalisation for System F, following Girard [12]. For technical reasons[2][3] we consider the syntax of full System F with constants.

$$A, B \in \mathsf{ty} ::= X \mid A \to B \mid \forall.\, A \qquad \text{Types}$$
$$s, t \in \mathsf{tm} ::= x \mid s\, t \mid s\, A \mid \lambda A.\, s \mid \Lambda.\, s \mid c \qquad \text{Terms}$$

We omit the discussion of the framework for System F, since it is completely analogous to the presentation for $\mathrm{F}_{\mathrm{CBV}}$ in Section 5.

We write $s \rightsquigarrow t$ for the standard single-step reduction relation of System F. The typing judgment of System F is defined by an inductive predicate $\Gamma \vdash s : A$ in exactly the same way as for $\mathrm{F}_{\mathrm{CBV}}$, omitting the distinction between value and term typing and the rule for switching between the two. We write $\mathrm{SN}\ s$ to express that the term s is strongly normalising.

The general structure of the proof remains the same as in Section 9. We define a logical relation as a simple traversal and obtain all of the usual structural properties for free. The main difference is that the definition of the logical relation is slightly more involved and that we have to quantify over terms in arbitrary scopes, instead of only considering closed values.

Definition 10.1. A term s is *neutral* iff it is not an abstraction, $s \neq \lambda A.\, b$ and $s \neq \Lambda.\, b$.

Definition 10.2 (Reducibility candidates)**.** For any fixed m, n we say that a predicate $P : \mathcal{P}\,\mathsf{tm}^{m,n}$ is *reducible* if it contains only strongly normalising terms, is closed under reduction, and is closed under expansion of neutral terms. P is reducible iff

1. $P\ s$ implies $\mathrm{SN}\ s$
2. $P\ s$, $s \rightsquigarrow t$ imply $P\ t$
3. for neutral s, we have $P\ s$ whenever $P\ t$ for all $s \rightsquigarrow t$.

We write $\mathrm{RC}^{m,n}$ for the type of reducible predicates. An environment $\rho : I^m \to \mathsf{tm}^{m',n'} \to \mathrm{Prop}$ is *admissible* if $\rho\ x$ is reducible for all x.

Definition 10.3. For any fixed m, n, we define the uniform simple type traversal $\mathrm{S} = (\mathsf{id}, \mathbb{A}_R, \mathbb{Q}_R) : \mathbb{T}^{\mathcal{P}(\mathsf{tm}^{m,n})}_{\mathcal{P}(\mathsf{tm}^{m,n})}$ interpreting types by sets of terms with a fixed number of type and value variables where

$$f \in \mathbb{A}_R\, P\, Q \;:=\; \forall s \in P.\, (f\ s) \in Q$$
$$f \in \mathbb{Q}_R\, F \;:=\; \forall (B : \mathsf{ty}^m)\, (P \in \mathrm{RC}^{m,n}).\, f\ B \in F\ P.$$

We write $\mathcal{E}\ A\ \rho$ for S evaluation.

[2] For $\mathrm{F}_{\mathrm{CBV}}$, we only have value instantiation and cannot formulate full β-reduction.

[3] Without constants the logical relation for closed terms contains no neutral terms.

The corresponding weakening and instantiation lemmas follow from the analogue of Theorem 8.4. In addition to this, we need a technical lemma to show strong normalisation. We omit the proof, as it is unrelated to any syntactic property of $\mathcal{E}$.

Lemma 10.4. *If ρ is admissible, then $\mathcal{E}\ A\ \rho$ is reducible.*

Definition 10.5. For contexts $\Gamma : I^n \to \mathsf{ty}^m$, types $A : \mathsf{ty}^m$, and terms $s : \mathsf{tm}^{m,n}$ we define the semantic typing judgment $\Gamma \vDash s : A$ as follows.

$$\Gamma \vDash s : A \;:=\; \forall m' n'\, (\rho \;:\; I^m \to \mathrm{RC}^{m',n'})\, \sigma\, \tau.$$
$$(\forall x.\, \tau\ x \in \mathcal{E}\ (\Gamma\ x)\ \rho) \;\to\; s[\sigma, \tau] \in \mathcal{E}\ A\ \rho$$

Theorem 10.6. *Syntactic typing implies semantic typing.*

$$\Gamma \vdash s : A \;\to\; \Gamma \vDash s : A$$

Proof. By induction on the typing derivation. The cases for variables and application follow from the definitions, while the case for type application makes use of Lemma 10.4 and compatibility with instantiation. In the case of abstraction and type abstraction we show closure under expansion by a nested induction using Lemma 10.4. The remainder of the proof follows using substitution lemmas analogous to Section 6, which hold for System F with nearly identical proofs. □

Corollary 10.7. *Well-typed terms are strongly normalising.*

$$\Gamma \vdash s : A \;\to\; \mathrm{SN}\ s$$

11 Implementation

The development of the paper is formalised in Coq, using the ssreflect tactic language [13]. For simplicity we use functional and propositional extensionality throughout the development, instead of setoids.

Many design decisions in this paper are directly motivated by the formalisation. For instance, we have introduced graded domains and bigraded domains as separate concepts. Instead, we could have considered both as instantiations of the same concept, either as degenerate functors, or using dependent types as functions with a variable number of arguments. However, both generalisations behave poorly in practice. Functors introduce tuple arguments in the multivariate case, and the lack of η-reduction for tuples in the Coq standard library leads to annoyances in the formalisation. Using dependent types is feasible, but leads to complicated proof goals when some types are not fully normalised.

Autoubst 2. We consider the formalisation as a prototype for a new version of Autosubst 2 [14]. As such, we have split the formalisation into two parts: the first dealing with sort-specific, but general, framework lemmas, the second with more specialised case studies. We hope to automate this first part in the future and to provide a comprehensive set of tactics to automate proofs in user code.

In the formalisation, we show various rewriting rules as part of the framework. We prove the substitution lemmas from Autosubst 2 [14], together with the rewriting rules of the σ-calculus as in [22]. Combined with the framework lemmas, this yields an effective and sufficient automation tactic for our case studies.

Using this automation, together with the notion of simple traversals, we have produced a weak normalisation proof for F_{CBV} with only 14 lines of (otherwise manual) proof script. This is the shortest normalisation proof we have ever produced using Autosubst in Coq. For comparison, the analogous proof with Autosubst 1 takes 65 lines of proof script.

We are confident that our approach can be generalised to automatically generate proofs out of second-order HOAS syntax as in [14].

Well-scopedness. Our syntax is well-scoped, i.e. syntactic sorts are represented as $\mathbb{N}$-indexed inductive type families to track an upper bound on the free variable indices. In contrast to the common assumption that this complicates the formalisation we found that it actually helped us in formulating the correct statements. A common pitfall with ordinary de Bruijn syntax is extensive arithmetic reasoning on the variable indices, which tends to guide users to rather unintuitive theorem statements and proofs. Adding scopes to the representation types of course complicates this ill-fated approach. We made the intentional choice to add in this perceived complication to deter users from suboptimal reasoning techniques. On the other hand, those primitives that work well with well-scoped syntax are incidentally those taken from the σ-calculus and they appear to be sufficient to formalise elegant de Bruijn proofs. Hence we believe, that being restricted to this set of operations is in fact a good thing and therefore part of our user interface.

In contrast, we avoid tracking object-level types, i.e. we do not adopt well-typed syntax. While this would provide even stronger guarantees, it would also lead to additional complications for certain object-level type system. For example, well-typed syntax for dependently-typed syntactic systems requires quotient inductive-inductive types [4], which are unavailable in Coq.

Limitations. In this work we consider syntactic systems with at most two sorts of variables, and correspondingly only introduce the notions of graded and bigraded domains. Gradedness can be generalised to the n-ary setting, extrapolating from the ideas shown in Section 5. These changes are mechanical, but rather technical and without further insight.

Our automation for traversals relies on its syntax-directed structure. For example, non-syntax directed type systems cannot be written as a traversal.

12 Related Work

There is a wealth of literature on the topics of syntax with binders and associated recursors (see [5, 8, 9] for an overview). In the following, we briefly summarise those which are most relevant to our approach.

Syntax traversals. The idea of syntax traversals goes back at least to McBride [18] and Pouillard and Pottier [21]. The former paper directly inspired the already mentioned [3]. In [21], compatibility with *renaming* is declared one main motivation. This result is obtained via parametric methods and the notion of world-polymorphic functions, of which traversals are one instance.

All in all, this paper is strongly inspired by and heavily indebted to the elegant representation of syntax traversals of Allais et al. [3]. In contrast to [3], we fold the notion of thinning into the abstract evaluation with respect to graded denotation domains. Allais gives a direct formulation of instantiation as a traversal , while we obtain it via a general notion of traversal lifting, thereby clarifying the connection of the two operations. In addition, lifting plays an integral part in our proof that natural traversals are generally compatible with instantiation.

Further differences are our ability to handle multivariate, mutually inductive sorts and the restriction of thinnings to injective renamings. The latter allows us to show naturality of the type system traversal J of F_{CBV}. Several other interesting traversals in [3] illustrate the beauty of the underlying principle.

Automation. Autosubst 2 [14] is a library for de Bruijn syntax in Coq that lifts the ideas of the σ-calculus [1] to the multivariate setting. It takes a concise second-order HOAS system description and generates de Bruijn syntax, capture-avoiding instantiation based on vectors of parallel substitutions and a decision procedure for assumption-free substitution lemmas. The decision procedure currently handles instances $s = t$ where s and t consist of terms, renamings, and instantiations. Here we suggest an extension: our goal is to handle instances $s = t$ where s and t may involve arbitrary structure-preserving traversals. Our framework yields the foundation for this. It allows us to derive the existing structures of Autosubst 2 as special instances of more general results such as the embedding and compatibility theorems. The incorporation of our framework into Autosubst 2 is planned for the near future.

Automating type systems. With Needle & Knot [16, 17], Keuchel et al. have introduced the first system which can automate large parts of type preservation proofs in Coq. This is done by elaborating a specification language containing relations parameterised over a type with binders into de Bruijn representations. We have shown that for syntax directed type systems (Section 7), we obtain large parts of this automation by formulating type systems as models.

It might be interesting to explore if our framework can be extended to more general type systems. Currently, every type system defined with a traversal must necessarily satisfy strengthening which can be limiting.

Presheaf models. A functorial graded domain can naturally be seen as a presheaf over a category of (injective) cartesian contexts [10]. We obtain a presheaf model if we embed proofs of naturality into each operation of a traversal. For example, a model for types yields morphisms for variables ($\mathbb{V} : \mathrm{Hom}\, V\, D$), arrows ($\mathbb{A} : \mathrm{Hom}\, (D{\times}D)\, D$) and quantifiers ($\mathbb{Q} : \mathrm{Hom}\, D^V\, D$).

This is remarkably similar to Fiore et al.'s [10] notion of an algebra for a binding signature. More precisely, Fiore et al. consider models with $V = I$ specifically, and show that this choice has strong universality properties. Note that only our case study on renaming fits directly into this schema.

It is striking that we have been lead so closely to a categorical approach to recursion for syntax with binding, even though we started our investigation from first principles. Compared to an ordinary presheaf model, our framework exhibits the following differences.

- We consider "unnatural" transformations in traversals, since this leads to better reduction behavior when we implement evaluation in Coq.
- We only require compositionality, not functoriality, to show compatibility with instantiation and naturality of evaluation. All other results do not need any structural properties of thinning.
- We separate traversals and naturality conditions and only require the graded domain structure on semantic indices for evaluation. As the graded domain structure on terms is renaming, this is crucial to define renaming within the same framework.

Programming with binders. In [11, 20], the authors formalise properties about shallow embeddings of many-sorted, possibly infinitary syntax in Isabelle. An inductively defined pretype of a universal algebra for bindings is introduced and a quotient up to α-equivalence ensures the intended interpretation of binding. The quotient construction requires a notion of "goodness" and conditions on the type of variables. Note that this quotient construction is already built into the de Bruijn representation of syntax. The framework is based on freshness and single-point substitution. Though a wealth of theorems are proven, no equational theory is recognisable.

Two definitions of models are suggested: while semantic domains seem to coincide with our notion of simple models, freshness-substitution models show parallels to our definition of a model. However, in contrast to our approach, freshness-substitution models ask the user to provide a significant amount of structure.

Pientka [19] takes a different approach to programming with binders, by changing the underlying type theory to include a notion of contexts, which are related to instantiation. In this approach, recursive functions can be written using higher-order pattern matching and executed by elaborating into a de Bruijn representation. It might be possible to elaborate some subset of these "contextual" definitions in terms of traversals and models, in order to avoid any changes to the underlying type theory. However, many open questions remain and we delegate a thorough exploration of these concepts to future work.

13 Conclusion

We have introduced a framework that systematically derives a binder aware recursor for the scope-safe de Bruijn encoding of non-trivial syntactic systems. The design extends current ideas of [3] via naturality to syntactic models, introduces a notion of traversal lifting, admits multivariate expression sorts and allows for mutual dependencies between multiple sorts.

At the same time, this construction supplements the Autosubst framework with a theoretical foundation for increased proof power. We are currently in the process of incorporating our framework into the implementation of Autosubst 2 [14]. We are confident that this paves the way towards better automation and moves us one step closer to elegant mechanisable metatheory in Coq.

We also consider a much more detailed comparison of the various proposed ways to formally deal with syntactic systems that involve binding. The goal would be to establish a solid theoretical foundation for the general mechanisation and automation of metatheory.

Acknowledgments

We would like to thank the anonymous reviewers whose comments and suggestions helped improve and clarify this paper.

References

[1] Martin Abadi, Luca Cardelli, P-L Curien, and J-J Lévy. 1991. Explicit substitutions. *Journal of functional programming* 1, 4 (1991), 375–416.

[2] Robin Adams. 2004. Formalized Metatheory with Terms Represented by an Indexed Family of Types. In *Proceedings of the 2004 International Conference on Types for Proofs and Programs (TYPES'04)*. 1–16. https://doi.org/10.1007/11617990_1

[3] Guillaume Allais, James Chapman, Conor McBride, and James McKinna. 2017. Type-and-scope Safe Programs and Their Proofs. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*. ACM, 195–207.

[4] Thorsten Altenkirch and Ambrus Kaposi. 2017. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science* Volume 13, Issue 4 (Oct. 2017). https://doi.org/10.23638/LMCS-13(4:1)2017

[5] Brian E Aydemir, Aaron Bohannon, Matthew Fairbairn, J Nathan Foster, Benjamin C Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey

Washburn, Stephanie Weirich, and Steve Zdancewic. 2005. Mechanized metatheory for the masses: The POPLmark challenge. In *TPHOLs*, Vol. 3603. Springer, 50–65.

[6] Richard S Bird and Ross Paterson. 1999. De Bruijn notation as a nested datatype. *Journal of functional programming* 9, 1 (1999), 77–91.

[7] Nicolaas Govert de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)* 75, 5 (1972), 381 – 392.

[8] Amy Felty and Alberto Momigliano. 2008. Hybrid: A definitional two-level approach to reasoning with higher-order abstract syntax. *arXiv preprint arXiv:0811.4367* (2008).

[9] Amy P Felty, Alberto Momigliano, and Brigitte Pientka. 2015. The Next 700 Challenge Problems for Reasoning with Higher-Order Abstract Syntax Representations. *Journal of Automated Reasoning* 55, 4 (2015), 307–372.

[10] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding (Extended Abstract). In *Proc. 14th LICS Conf.* IEEE, Computer Society Press, 193–202.

[11] Lorenzo Gheri and Andrei Popescu. 2017. A formalized general theory of syntax with bindings. In *International Conference on Interactive Theorem Proving*. Springer, 241–261.

[12] Jean-Yves Girard, Yves Lafont, and Paul Taylor. 1989. *Proofs and types*. Vol. 7. Cambridge University Press Cambridge.

[13] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. 2016. *A Small Scale Reflection Extension for the Coq system*. Research Report RR-6455. Inria Saclay Ile de France. https://hal.inria.fr/inria-00258384

[14] Jonas Kaiser, Steven Schäfer, and Kathrin Stark. 2017. Autosubst 2: Towards Reasoning with Multi-Sorted De Bruijn Terms and Vector Substitutions. In *Proceedings of the Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP '17)*. ACM, New York, NY, USA, 10–14. https://doi.org/10.1145/3130261.3130263

[15] Jonas Kaiser, Tobias Tebbi, and Gert Smolka. 2017. Equivalence of System F and $\lambda 2$ in Coq based on Context Morphism Lemmas. In *Proceedings of CPP 2017*. ACM.

[16] Steven Keuchel, Tom Schrijvers, and Stephanie Weirich. 2017. *Needle & Knot: Boilerplate bound tighter*. Technical Report.

[17] Steven Keuchel, Stephanie Weirich, and Tom Schrijvers. 2016. Needle & Knot: Binder boilerplate tied up. In *European Symposium on Programming Languages and Systems*. Springer, 419–445.

[18] Conor McBride. 2005. Type-preserving renaming and substitution. (2005).

[19] Brigitte Pientka. 2008. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *ACM SIGPLAN Notices*, Vol. 43. ACM, 371–382.

[20] Andrei Popescu and Elsa L Gunter. 2011. Recursion principles for syntax with bindings and substitution. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 346–358.

[21] Nicolas Pouillard and François Pottier. 2010. A fresh look at programming with names and binders. In *ACM Sigplan Notices*, Vol. 45. ACM, 217–228.

[22] Steven Schäfer, Gert Smolka, and Tobias Tebbi. 2015. 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*. Springer-Verlag, Berlin, Heidelberg, 67–73. https://doi.org/10.1145/2676724.2693163

[23] Steven Schäfer, Tobias Tebbi, and Gert Smolka. 2015. Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions. In *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings (Lecture Notes in Computer Science)*, Christian Urban and Xingyuan Zhang (Eds.), Vol. 9236. Springer, 359–374. https://doi.org/10.1007/978-3-319-22102-1_24