Abstract clones for abstract syntax

Nathanael Arkor
University of Cambridge, UK

Dylan McDermott
Reykjavik University, Iceland

Abstract

We give a formal treatment of simple type theories, such as the simply-typed λ-calculus, using the framework of abstract clones. Abstract clones traditionally describe first-order structures, but by equipping them with additional algebraic structure, one can further axiomatize second-order, variable-binding operators. This provides a syntax-independent representation of simple type theories. We describe multisorted second-order presentations, such as the presentation of the simply-typed λ-calculus, and their clone-theoretic algebras; free algebras on clones abstractly describe the syntax of simple type theories quotiented by equations such as β- and η-equality. We give a construction of free algebras and derive a corresponding induction principle, which facilitates syntax-independent proofs of properties such as adequacy and normalization for simple type theories. Working only with clones avoids some of the complexities inherent in presheaf-based frameworks for abstract syntax.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Equational logic and rewriting; Theory of computation → Higher order logic; Theory of computation → Proof theory

Keywords and phrases simple type theories, abstract clones, second-order abstract syntax, substitution, variable binding, presentations, free algebras, induction, logical relations

Funding *Dylan McDermott*: Icelandic Research Fund project grant № 196323-053

1 Introduction

The abstract concept of type theory is crucial in the study of programming languages. However, while it is generally appreciated that the concrete syntax associated to a type theory is peripheral to its fundamental structure, conventional techniques for working with type theories and proving properties thereof are predominantly syntactic. The primary reason for this incongruity is that, though abstract frameworks for defining and reasoning about general classes of type theories have been developed (e.g. [14, 13, 5, 12, 21, 2, 3, 19], there called *second-order abstract syntax*), the mathematical prerequisites are significant and often appear unapproachable to those without a firm category theoretic background. This is regrettable, because these general techniques alleviate much of the rote associated to syntactic proofs, such as those for adequacy, normalization, and the admissibility of substitution.

It so happens that there exists in the mathematical folklore an approach that is particularly well-suited to capturing the essential structure of simple type theories and yet requires essentially no experience with category theory to employ fruitfully: this is the formalism of *abstract clones* (often simply called *clones*) with algebraic structure. The structure of an abstract clone captures the notion of a context-indexed family of terms, closed under variable projection and substitution; equipping clones with algebraic structure permits the expression of variable-binding operators, like the λ-abstraction operator familiar from λ-calculi. It is known amongst cognoscenti that abstract clones might be employed for this purpose: for instance, Fiore, Plotkin, and Turi [16] proved that abstract clones are equivalent to their notion of *substitution monoids*, which represent families of (unityped) terms with an associated capture-avoiding substitution operation; later, Fiore and Mahmoud [32, 15] proved that abstract clones with algebraic structure are equivalent to the Σ-*monoids* of Fiore et

al., which extend substitution monoids with second-order (i.e. variable binding) algebraic structure. In a separate line of inquiry, Hyland [24] uses abstract clones with algebraic structure to give a modern treatment of the unityped λ-calculus. However, it does not appear that abstract clones have previously been expressly proposed for the study of simple type theories (in fact, the definition of a typed abstract clone with algebraic structure is absent from the literature).

Here, we give an exposition of the use of abstract clones with algebraic structure in defining simple type theories and proving various of their properties. After setting up the relevant definitions (Section 2), we describe how simple type theories can be modelled by algebras of second-order presentations (Section 3). We then show that free algebras exist, giving an abstract description of the syntax of the type theory (Section 4). We derive an induction principle [30] that enables abstract reasoning about the syntax (Section 5), and show that this is powerful enough to prove non-trivial properties of type theories, in particular using logical relations (Section 6). We also compare the clone-theoretic framework to other approaches (Section 7). Though we do not expect our treatment to be surprising to experts familiar with prior categorical developments, it is an important perspective in the understanding of simple type theories and deserves explication.

Though we occasionally make reference to category theory throughout the paper, knowledge of category theory is not necessary to understand the content.

2 Abstract clones and first-order presentations

A typed (or *multisorted*) *abstract clone* [36], henceforth simply *clone*, encapsulates the structure of terms in simple contexts, closed under variables and substitution. Informally, for each context $x_1 : A_1, \dots, x_n : A_n$ and type B, where A_1 to A_n are types (or *sorts*), a clone $\mathbf{X}$ specifies a set of terms $X(A_1, \dots, A_n; B)$, each element of which is considered a term of type B in the context $x_1 : A_1, \dots, x_n : A_n$. It also specifies terms var_i representing the projection of the variable x_i from the context, and functions $\mathsf{subst}_{\Gamma;A_n,\dots,A_n;B} : X(A_1, \dots, A_n; B) \times X(\Gamma; A_1) \times \cdots \times X(\Gamma; A_n) \to X(\Gamma; B)$ representing simultaneous substitution:

$$\begin{aligned} t \in X(A_1, \dots, A_n; B) \quad &\textit{represents} \quad x_1 : A_1, \dots, x_n : A_n \vdash t : B \\ \mathsf{var}_i^{(A_1,\dots,A_n)} \in X(A_1, \dots, A_n; A_i) \quad &\textit{represents} \quad x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i \\ \mathsf{subst}_{\Gamma;A_1,\dots,A_n;B}(t, u_1, \dots, u_n) \quad &\textit{represents} \quad \Gamma \vdash t\{x_1 \mapsto u_1, \dots, x_n \mapsto u_n\} : B \end{aligned}$$

The clone $\mathbf{X}$ is required to satisfy laws expressing that (1) substituting variables for themselves does nothing; (2) applying a substitution to a variable results in the term corresponding to that variable in the substitution; and (3) substitution is associative.

▶ **Notation 1.** *We fix a set S of types (sorts). We denote by S^* the free monoid on S, i.e. lists of elements of S. Conceptually, contexts $x_1 : A_1, \dots, x_n : A_n$ are given by elements $[A_1, \dots, A_n] \in S^*$, since variable names carry no information. We write $\diamond \in S^*$ for the empty context, and Γ, Ξ for the concatenation of $\Gamma \in S^*$ and $\Xi \in S^*$. For contexts $\Gamma, \Delta \in S^*$, where $\Delta = [A_1, \dots, A_n]$, we define $X(\Gamma; \Delta) = \prod_{i \le n} X(\Gamma; A_i)$. We call the elements $\boldsymbol{\sigma} \in X(\Gamma; \Delta)$* substitutions*; a substitution is therefore a tuple $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)$ of terms $\sigma_i \in X(\Gamma; A_i)$.*

▶ **Definition 2.** *An S-sorted* clone $\mathbf{X} = (X, \mathsf{var}, \mathsf{subst})$ *consists of*

- *for each context $\Gamma \in S^*$ and sort $A \in S$, a set $X(\Gamma; A)$ of* terms*;*
- *for each context $\Gamma \in S^*$, a tuple $\mathsf{var}^{(\Gamma)} \in X(\Gamma; \Gamma)$ of* variables*;*
- *for each pair of contexts $\Gamma, \Delta \in S^*$ and sort $A \in S$, a* substitution function $\mathsf{subst}_{\Gamma;\Delta;A} : X(\Delta; A) \times X(\Gamma; \Delta) \to X(\Gamma; A)$*, which we write as $t[\boldsymbol{\sigma}] = \mathsf{subst}_{\Gamma;\Delta;A}(t, \boldsymbol{\sigma})$;*

such that

$$\mathsf{var}_i^{(A_1,\dots,A_n)}[\boldsymbol{\sigma}] = \sigma_i \qquad \text{for each } \boldsymbol{\sigma} \in X(\Gamma; A_1, \dots, A_n) \text{ and } i \leq n \tag{1}$$

$$t[\mathsf{var}^{(\Gamma)}] = t \qquad \text{for each } t \in X(\Gamma; A) \tag{2}$$

$$t[\sigma'_1[\boldsymbol{\sigma}], \dots, \sigma'_m[\boldsymbol{\sigma}]] = (t[\boldsymbol{\sigma}'])[\boldsymbol{\sigma}] \quad \text{for each } t \in X(\Xi; A),\ \boldsymbol{\sigma}' \in X(\Delta; \Xi),\ \boldsymbol{\sigma} \in X(\Gamma; \Delta) \tag{3}$$

A clone homomorphism $f : \mathbf{X} \to \mathbf{X}'$ *consists of a function* $f_{\Gamma;B} : X(\Gamma; B) \to X'(\Gamma; B)$ *for each context* $\Gamma \in S^*$ *and sort* $B \in S$*, such that the following hold, where* $\Delta = [A_1, \dots, A_n] \in S^*$*:*

$$f_{\Delta;A_i}(\mathsf{var}_i^{(\Delta)}) = \mathsf{var}_i^{(\Delta)'} \qquad \text{for each } i \leq n$$

$$f_{\Gamma;B}(t[\boldsymbol{\sigma}]) = (f_{\Delta;B}(t))[f_{\Gamma;A_1}(\sigma_1), \dots, f_{\Gamma;A_n}(\sigma_n)]' \quad \text{for each } t \in X(\Delta; B),\ \boldsymbol{\sigma} \in X(\Gamma; \Delta)$$

We write $\mathbf{Clone}(S)$ *for the category of* S*-sorted clones and homomorphisms.*

We extend every clone homomorphism $f : \mathbf{X} \to \mathbf{X}'$ to act on substitutions as follows, where $\Delta = [A_1, \dots, A_n] \in S^*$:

$$f_{\Gamma;\Delta} : X(\Gamma; \Delta) \to X'(\Gamma; \Delta) \qquad f_{\Gamma;\Delta}(\boldsymbol{\sigma}) = (f_{\Gamma;A_1}(\sigma_1), \dots, f_{\Gamma;A_n}(\sigma_n))$$

▶ **Example 3.** We denote by $\mathbf{Var}_S$ the S-sorted *clone of variables*, whose family of terms is given by $\mathsf{Var}_S(A_1, \dots, A_n; B) = \{i \mid A_i = B\}$; whose variables are given by $\mathsf{var}_i^{(\Gamma)} = i$; and whose substitution is given by $i[\boldsymbol{\sigma}] = \sigma_i$. $\mathbf{Var}_S$ is the initial object in $\mathbf{Clone}(S)$: for any S-sorted clone $\mathbf{X}$, there is a unique homomorphism $\triangleright : \mathbf{Var}_S \to \mathbf{X}$ given by $\triangleright_{\Gamma;B}(i) = \mathsf{var}_i^{(\Gamma)}$.

▶ **Example 4.** The terms of any universal algebra [8] form a *monosorted* clone (i.e. an S-sorted clone for which S is a singleton $\{*\}$). The sets of terms, along with the variables and substitution function, exactly match the classical notions. For instance, monoids form a clone $\mathbf{Mon}$, where $\mathrm{Mon}(\underbrace{*, \dots, *}_{n}; *)$ is the free monoid on n elements.

▶ **Example 5.** Let Ty be the set of sorts freely generated by a base type $\mathsf{b} \in \mathsf{Ty}$ and function types $(A \Rightarrow B) \in \mathsf{Ty}$ for $A, B \in \mathsf{Ty}$ (precisely, Ty is the free magma on $\{\mathsf{b}\}$). The terms of the simply typed λ-calculus (STLC) form a Ty-sorted clone $\boldsymbol{\Lambda}$. Consider terms generated by the following rules:

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A} \qquad \frac{\Gamma \vdash f : A \Rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{app}\, f\, a : B} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \Rightarrow B}$$

(We write app to distinguish application of λ-terms from application of mathematical functions. We also use named variables for readability, identifying α-equivalent terms.) Capture-avoiding simultaneous substitution $t\{x_i \mapsto u_i\}_i$ of terms is defined in the usual way by recursion on t:

$$x_j\{x_i \mapsto u_i\}_i = u_j \qquad (\mathsf{app}\, f\, a)\{x_i \mapsto u_i\}_i = \mathsf{app}\,(f\{x_i \mapsto u_i\}_i)\,(a\{x_i \mapsto u_i\}_i)$$

$$(\lambda x : A.\, t)\{x_i \mapsto u_i\}_i = \lambda y : A.\, (t\{x_1 \mapsto u_1, \dots, x_n \mapsto u_n, x \mapsto y\})$$

The clone $\boldsymbol{\Lambda}$ has sets of terms $\Lambda(A_1, \dots, A_n; B) = \{x_1 : A_1, \dots, x_n : A_n \vdash t : B\}$, variables $\mathsf{var}_i^{(\Gamma)} = x_i$, and substitution $t[\boldsymbol{\sigma}] = t\{x_i \mapsto \sigma_i\}_i$.

There is a related Ty-sorted clone $\boldsymbol{\Lambda}_{\beta\eta}$ of STLC terms up to $\beta\eta$-equality, defined by quotienting the sets of terms associated to $\boldsymbol{\Lambda}$ by the equivalence relation $\approx_{\beta\eta}$, where $\Gamma \vdash t \approx_{\beta\eta} t' : A$ is the congruence relation generated by the following rules:

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{app}\,(\lambda x : A.\, t)\, u \approx_{\beta\eta} t\{x \mapsto u\} : B}\ (\beta) \qquad \frac{\Gamma \vdash t : A \Rightarrow B}{\Gamma \vdash t \approx_{\beta\eta} \lambda x : A.\, \mathsf{app}\, t\, x : A \Rightarrow B}\ (\eta)$$

▶ Remark 6. We shall only consider abstract clones with *sets* of types. However, as illustrated by the previous example, the types in a simple type theory often have algebraic structure themselves. By considering only the underlying set of types, the algebraic structure is forgotten. This simplifies the development, at the cost of some loss of expressivity. By specifying a (monosorted) clone of types, rather than a set, one recovers exactly the *simple type theories* of Arkor and Fiore [5].

$\mathbf{Clone}(S)$ is a cartesian category, permitting us to combine clones pointwise. The terminal object $\mathbf{1}$ is the unique clone in which every set of terms is a singleton. The binary product $\mathbf{X}_1 \times \mathbf{X}_2$ has sets of terms given by products of sets $(X_1 \times X_2)(\Gamma; A) = X_1(\Gamma; A) \times X_2(\Gamma; A)$, variables $\mathsf{var}_i^{(\Gamma)} = (\mathsf{var}_i^{(\Gamma)}, \mathsf{var}_i^{(\Gamma)})$, and substitution $(t_1, t_2)[(\sigma_{11}, \sigma_{21}), \ldots, (\sigma_{1n}, \sigma_{2n})] = (t_1[\boldsymbol{\sigma}_1], t_2[\boldsymbol{\sigma}_2])$.

▶ Remark 7. S-sorted abstract clones form a *variety* in the sense of universal algebra; this means that $\mathbf{Clone}(S)$ is the category of models for a (multisorted) algebraic theory. Such categories are well-behaved, and several of the properties we mention throughout the paper (such as being cartesian) follow abstractly from this observation. We often choose to be more explicit for ease of comprehension, but make note where this abstract perspective is helpful.

2.1 Substitution and context extension

We briefly consider the structure of substitutions $\boldsymbol{\sigma}$ in S-sorted clones $\mathbf{X}$, in particular to define various substitutions that we use below, and to characterize context extension in clones. If $\boldsymbol{\sigma} \in X(\Gamma; \Delta)$ and $\boldsymbol{\sigma}' \in X(\Delta; \Xi)$ are substitutions, then their *composition* $(\boldsymbol{\sigma}' \circ \boldsymbol{\sigma}) \in X(\Gamma; \Xi)$ is the substitution $(\sigma'_1[\boldsymbol{\sigma}], \ldots, \sigma'_m[\boldsymbol{\sigma}])$, where m is the length of Ξ. The three equations in the definition of a clone (Definition 2) equivalently state (1 & 2) that var is the (left- and right-) unit for composition $(\mathsf{var}^{(\Delta)} \circ \boldsymbol{\sigma} = \boldsymbol{\sigma} = \boldsymbol{\sigma} \circ \mathsf{var}^{(\Gamma)})$; and (3) that composition is associative $(\boldsymbol{\sigma}'' \circ (\boldsymbol{\sigma}' \circ \boldsymbol{\sigma}) = (\boldsymbol{\sigma}'' \circ \boldsymbol{\sigma}') \circ \boldsymbol{\sigma})$. In fact, this perspective underlies the connection between abstract clones and cartesian multicategories (which may be considered categories whose morphisms have multiple inputs, corresponding to each of the variables in a context): we elaborate on this connection in Section 7.

We call the substitutions $\boldsymbol{\rho} \in \mathsf{Var}_S(\Gamma; \Delta)$ *variable renamings*. This is justified by observing that $\boldsymbol{\rho}$ selects a variable in the context Δ for each variable in Γ. If $t \in X(\Delta; A)$ is a term in some clone $\mathbf{X}$, then $t[\triangleright\boldsymbol{\rho}] \in X(\Gamma; A)$ corresponds to the term in which the variables in t have been renamed according to $\boldsymbol{\rho}$. A special case of renaming is *weakening* $\mathsf{wk}_\Xi^{(\Gamma)} = (1, \ldots, n) \in \mathsf{Var}_S(\Gamma, \Xi; \Gamma)$. Using weakening and composition, we may define the *lifting* of a substitution $\boldsymbol{\sigma} \in X(\Gamma; \Delta)$ to a larger context:

$$\mathsf{lift}_\Xi(\boldsymbol{\sigma}) = (\boldsymbol{\sigma} \circ (\triangleright\mathsf{wk}_\Xi^{(\Gamma)}), \triangleright(n+1, \ldots, n+m)) \in X(\Gamma, \Xi; \Delta, \Xi)$$

where n is the length of Γ and m is the length of Ξ.

Context extension induces the following operation on clones. Given an S-sorted clone $\mathbf{X}$ and context $\Xi \in S^*$, we let $\Uparrow^\Xi \mathbf{X}$ be the S-sorted clone with terms $(\Uparrow^\Xi X)(\Gamma; A) = X(\Gamma, \Xi; A)$, variables $(\mathsf{var}_i^{(\Gamma,\Xi)})_{i \le n} \in X(\Gamma, \Xi; \Gamma)$, and substitution $t[\boldsymbol{\sigma}, \triangleright(n+1, \ldots, n+m)] \in X(\Gamma, \Xi; A)$ for $t \in X(\Delta, \Xi; A)$ and $\boldsymbol{\sigma} \in X(\Gamma, \Xi; \Delta)$, where n is the length of Γ and m is the length of Ξ. This satisfies a universal property as follows. Weakening forms a homomorphism $\mathsf{weaken}_{\mathbf{X}}^{(\Xi)} : \mathbf{X} \to \Uparrow^\Xi \mathbf{X}$ that sends $t \in X(\Gamma; A)$ to $t[\triangleright\mathsf{wk}_\Xi^{(\Gamma)}] \in X(\Gamma, \Xi; A)$. Then, for every homomorphism $g : \Uparrow^\Xi \mathbf{X} \to \mathbf{Y}$, we obtain a homomorphism $g \circ \mathsf{weaken}_{\mathbf{X}}^{(\Xi)} : \mathbf{X} \to \mathbf{Y}$ and a substitution $g_{\diamond;\Xi}(\mathsf{var}^{(\Xi)}) \in Y(\diamond; \Xi)$. Together, these uniquely determine g: to give a homomorphism g is just to give a homomorphism $\mathbf{X} \to \mathbf{Y}$ and a closed term σ_i for each extra variable from Ξ. (From the perspective of algebraic theories, context extension $\Uparrow^\Xi \mathbf{X}$ corresponds to the construction of the *polynomial* [28] or *simple slice category* [26] over Ξ.)

▶ **Lemma 8.** *For each clone homomorphism $f : \mathbf{X} \to \mathbf{Y}$ and substitution $\boldsymbol{\sigma} \in Y(\diamond; \Xi)$, there is a unique homomorphism $g : \Uparrow^{\Xi}\mathbf{X} \to \mathbf{Y}$ such that $g \circ \mathsf{weaken}_{\mathbf{X}}^{(\Xi)} = f$ and $g_{\diamond;\Xi}(\mathsf{var}^{(\Xi)}) = \boldsymbol{\sigma}$.*

Proof. Suppose g is such a homomorphism. Then, for each $t \in X(\Gamma, \Xi; A)$, we have $g_{\Gamma;A}(t) = (g_{\Gamma,\Xi;A}(\mathsf{weaken}_{\mathbf{X}}^{(\Xi)}(t)))[\mathsf{var}^{(\Gamma)}, (g_{\diamond;\Xi}(\mathsf{var}^{(\Xi)})) \circ \triangleright \mathsf{wk}_{\Gamma}^{(\diamond)}] = (f_{\Gamma,\Xi;A}(t))[\mathsf{var}^{(\Gamma)}, \boldsymbol{\sigma} \circ \triangleright \mathsf{wk}_{\Gamma}^{(\diamond)}]$, where the first equality uses preservation of variables and substitution by g, and the second uses the assumptions on g. Hence, g is unique when it exists. For existence, define $g_{\Gamma;A}(t) = (f_{\Gamma,\Xi;A}(t))[\mathsf{var}^{(\Gamma)}, \boldsymbol{\sigma} \circ \triangleright \mathsf{wk}_{\Gamma}^{(\diamond)}]$. ◀

Substitutions $\boldsymbol{\sigma} \in Y(\diamond; \Xi)$ are in natural bijection with homomorphisms $\Uparrow^{\Xi}\mathsf{Var}_S \to \mathbf{Y}$, and so Lemma 8 equivalently states that $\Uparrow^{\Xi}\mathbf{X}$ is the coproduct of $\mathbf{X}$ and $\Uparrow^{\Xi}\mathsf{Var}_S$. (This contrasts with presheaf-based frameworks [16, 22], in which context extension is exponentiation.)

2.2 First-order presentations

Clones describe collections of terms closed under variable projection and substitution. We will frequently be interested in clones equipped with extra structure, so as, for example, to interpret the operations of a given type theory. *Presentations* permit the axiomatization of clones that interpret various operations, subject to sets of axioms; while the *algebras* for a given presentation are exactly those clones that satisfy the axiomatization. Later, we will see how clones may be freely generated from presentations, allowing one to define a clone simply by specifying its generating operators and axioms.

Our treatment of first-order presentations is the classical notion of presentation for multisorted universal algebra [9, 17].

▶ **Definition 9.** *An S-sorted* first-order signature Σ *consists of a set $\Sigma(\Gamma; B)$ for each $(\Gamma; B) \in S^* \times S$. We call the elements $\mathsf{o} \in \Sigma(\Gamma; B)$ the $(\Gamma; B)$-*ary operators. Terms over Σ *are generated by the following rules:*

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A} \qquad \frac{\Gamma \vdash t_1 : A_1 \quad \cdots \quad \Gamma \vdash t_n : A_n}{\Gamma \vdash \mathsf{o}(t_1, \ldots, t_n) : B} \ (\mathsf{o} \in \Sigma(A_1, \ldots, A_n; B))$$

*An $(A_1, \ldots, A_n; B)$-*ary term t *over Σ is a term $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$, and an $(\Gamma; B)$-*ary equation *over Σ is a pair (t, u) of $(\Gamma; B)$-ary terms. An S-*sorted first-order presentation *$\boldsymbol{\Sigma} = (\Sigma, E)$ consists of an S-sorted first-order signature Σ and, for each $(\Gamma; B) \in S^* \times S$, a set $E(\Gamma; B)$ of $(\Gamma; B)$-ary equations.*

▶ Remark 10. Observe that the operators of a signature correspond to terms in the logic specified below (namely, first-order equational logic). In particular, a $(\Gamma; B)$-ary operator o, where $\Gamma = [A_1, \ldots, A_n] \in S^*$, may be thought of either as a function $\mathsf{o} : A_1, \ldots, A_n \to B$, or as a term $x_1 : A_1, \ldots, x_n : A_n \vdash \mathsf{o} : B$. These perspectives are complementary, and mirror the practice in categorical logic of representing terms by morphisms.

▶ **Definition 11.** *If $\Gamma \vdash u_i : A_i$ for $i \leq n$ and $x_1 : A_1, \ldots x_n : A_n \vdash t : B$ are terms over an S-sorted first-order signature Σ, their substitution $\Gamma \vdash t\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\} : B$ is defined by recursion on t in the usual way. The* equational logic *over an S-sorted first-order presentation $\boldsymbol{\Sigma} = (\Sigma, E)$ consists of the following rules for the congruence of $\approx$ under operations and substitution, together with reflexivity, symmetry and transitivity of $\approx$:*

$$\frac{\Gamma \vdash t_1 \approx u_1 : A_1 \quad \cdots \quad \Gamma \vdash t_n \approx u_n : A_n}{\Gamma \vdash \mathsf{o}(t_1, \ldots, t_n) \approx \mathsf{o}(u_1, \ldots, u_n) : B} \ (\mathsf{o} \in \Sigma(A_1, \ldots, A_n; B))$$

$$\frac{\Gamma \vdash t'_1 \approx u'_1 : A_1 \quad \cdots \quad \Gamma \vdash t'_n \approx u'_n : A_n}{\Gamma \vdash t\{x_i \mapsto t'_i\}_i \approx u\{x_i \mapsto u'_i\}_i : B} \ ((t, u) \in E(A_1, \ldots, A_n; B))$$

The terms over Σ form a clone $\mathbf{Term}^{\mathbf{\Sigma}} = (\mathsf{Term}^{\mathbf{\Sigma}}, \mathsf{var}, \mathsf{subst})$*, where* $\mathsf{Term}^{\mathbf{\Sigma}}(\Gamma; A)$ *is the set of* $\approx$*-equivalence classes of* $(\Gamma; A)$*-ary terms; the variables are* $\mathsf{var}_i^{(\Gamma)} = x_i$*; and substitution is* $t[\boldsymbol{\sigma}] = t\{x_i \mapsto \sigma_i\}_i$*. A clone* $\mathbf{X}$ *is* presented by $\mathbf{\Sigma}$ *when* $\mathbf{Term}^{\mathbf{\Sigma}}$ *is isomorphic to* $\mathbf{X}$ *in* $\mathbf{Clone}(S)$ *(that is, when there are homomorphisms* $\mathbf{Term}^{\mathbf{\Sigma}} \rightleftarrows \mathbf{X}$ *that are mutually inverse).*

▶ Remark 12. A clone may have many different presentations: for instance, the clone **Mon** of monoids (Example 4) may be presented by a unit and a binary multiplication operation, or by an n-ary multiplication operation for each $n \in \mathbb{N}$ (subject to suitable axioms).

▶ **Example 13.** Fix a finite set $V = \{v_1, \ldots, v_k\}$ of *values*. The Ty-sorted presentation $\mathbf{\Sigma}_V^{\mathbf{GS}}$ of *global V-valued state* has a $(\underbrace{\mathsf{b}, \ldots, \mathsf{b}}_{k}; \mathsf{b})$-ary operator get, a $(\mathsf{b}; \mathsf{b})$-ary operator put_{v_i} for each $i \leq k$, and equations

$$\begin{aligned} x : \mathsf{b} &\vdash \mathsf{get}(\mathsf{put}_{v_1}(x), \ldots, \mathsf{put}_{v_k}(x)) \approx x : \mathsf{b} \\ x_1 : \mathsf{b}, \ldots, x_k : \mathsf{b} &\vdash \mathsf{put}_{v_i}(\mathsf{get}(x_1, \ldots, x_k)) \approx \mathsf{put}_{v_i}(x_i) : \mathsf{b} && \text{for each } i \leq k \\ x : \mathsf{b} &\vdash \mathsf{put}_{v_i}(\mathsf{put}_{v_j}(x)) \approx \mathsf{put}_{v_j}(x) : \mathsf{b} && \text{for each } i, j \leq k \end{aligned}$$

Informally, the term $\mathsf{get}(t_1, \ldots, t_n)$ gets the current value v_i of the state and then continues as t_i, while the term $\mathsf{put}_{v_i}(t)$ sets the state to v_i and then continues as t. (In Example 23 below, we combine this presentation with the STLC to obtain a call-by-name calculus with global state. In call-by-name calculi, effects occur at base types, so it is only necessary to axiomatize get and put_{v_i} operators for $\mathsf{b} \in \mathsf{Ty}$, rather than for all types.) We denote by $\mathbf{GS}_V$ the clone $\mathbf{Term}^{\mathbf{\Sigma}_V^{\mathbf{GS}}}$ arising from the presentation $\mathbf{\Sigma}_V^{\mathbf{GS}}$.

3 Second-order presentations

Just as first-order presentations describe algebraic structure, second-order presentations describe binding algebraic structure [16]. Variable-binding operators are prevalent in type theory: for instance, the λ-abstraction operator of the STLC, let-in expressions in functional programming languages, and case-splitting in calculi with sum types. Second-order presentations are similar to first-order presentations, except that each operator must describe its binding structure, i.e. how many variables (and of what types) it binds in each operand. Hence, while first-order arities have the form $(A_1, \ldots, A_n; B) \in S^* \times S$, second-order arities have the form $((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B) \in (S^* \times S)^* \times S$. Operators of such an arity take n arguments of types $A_1, \ldots A_n$ and produce terms of type B: the length of the context $\Delta_i \in S^*$ is the number of variables bound by the i^{th} argument; and the argument types are given by the list Δ_i. First-order operators may be expressed as second-order operators that bind no variables.

▶ **Definition 14.** *An S-sorted* second-order signature *[16, 13] consists of a set $\Sigma(\Psi; B)$ for each $(\Psi; B) \in (S^* \times S)^* \times S$. We call the elements $\mathsf{o} \in \Sigma(\Psi; B)$ the $(\Psi; B)$-*ary operators.

▶ **Example 15.** The Ty-sorted second-order signature Σ^Λ of the STLC consists of an $((\diamond; A \Rightarrow B), (\diamond; A); B)$-ary operator app and an $((A; B); (A \Rightarrow B))$-ary operator abs for each $A, B \in \mathsf{Ty}$. Thus each application operator app has two arguments, neither of which bind variables; and each λ-abstraction operator abs has one argument, which binds one variable.

Just as the axioms of first-order presentations are expressed in first-order equational logic, the axioms of second-order presentations are expressed in the second-order equational logic of Fiore and Hur [13]. Second-order equational logic extends the first-order setting with *metavariables* [1, 18, 11], which conceptually stand for parameterized placeholders for terms.

Each variable $x : A$ in first-order logic has an associated type $A \in S$; correspondingly, each metavariable $M : (A_1, \ldots, A_n; A)$ has an associated context and type (called *second-order arities* in [5]). M may be thought of as a variable parameterized by n terms of types A_1 through A_n; a nullary ($n = 0$) metavariable behaves like an ordinary variable. There are several alternative ways to describe second-order equational logic [6], but we follow Fiore and Hur [13] in associating to each term both a variable context and a metavariable context: a metavariable context Ψ is a list of context–sort pairs $(\Delta; A) \in S^* \times S$. The judgment $\Psi \mid \Delta \vdash t : A$ expresses that the term t has sort A in variable context Δ and metavariable context Ψ. Below, we write $\vec{x}$ for a list $x_1, \ldots, x_n$ of variables, $\vec{x}.\, t$ to indicate binding of the variables $\vec{x}$ in t, and write $\vec{x} : \Delta$ as an abbreviation of $x_1 : A_1, \ldots, x_n : A_n$ for $\Delta = [A_1, \ldots, A_n]$.

▶ **Definition 16.** *Suppose S is a set and Σ is an S-sorted second-order signature.* Terms over Σ *are generated by the following rules for variables, metavariables, and operators:*

$$\frac{}{\Psi \mid \Gamma, x : A, \Delta \vdash x : A}$$

$$\frac{\Psi, M : (A_1, \ldots, A_n; B), \Phi \mid \Gamma \vdash t_1 : A_1 \quad \cdots \quad \Psi, M : (A_1, \ldots, A_n; B), \Phi \mid \Gamma \vdash t_n : A_n}{\Psi, M : (A_1, \ldots, A_n; B), \Phi \mid \Gamma \vdash M(t_1, \ldots, t_n) : B}$$

$$\frac{\Psi \mid \Gamma, \vec{x}_1 : \Delta_1 \vdash t_1 : A_1 \quad \cdots \quad \Psi \mid \Gamma, \vec{x}_n : \Delta_n \vdash t_n : A_n}{\Psi \mid \Gamma \vdash \mathsf{o}((\vec{x}_1.\, t_1), \ldots, (\vec{x}_n.\, t_n)) : B} \; (\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B))$$

A $((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$-ary term over Σ is a term $M_1 : (\Delta_1, A_1), \ldots, M_n : (\Delta_n; A_n) \mid \diamond \vdash t : B$, and a $(\Psi; B)$-ary equation is a pair (t, u) of $(\Psi; B)$-ary terms. An S-sorted second-order presentation $\mathbf{\Sigma} = (\Sigma, E)$ consists of an S-sorted second-order signature Σ and, for each $(\Psi; B) \in (S^* \times S)^* \times S$, a set $E(\Psi; B)$ of $(\Psi; B)$-ary equations over Σ.

Multisorted second-order presentations may essentially be taken as a definition of *simple type theory* (modulo the subtlety regarding type operators described in Remark 6): just as the informal notion of *algebra* was formalized through the framework of universal algebra [8], so second-order presentations facilitate a precise, formal definition of simple type theory [5].

▶ **Example 17.** The operators of the signature Σ^Λ of the STLC present the following rules:

$$\frac{\Psi \mid \Gamma \vdash f : A \Rightarrow B \qquad \Psi \mid \Gamma \vdash a : A}{\Psi \mid \Gamma \vdash \mathsf{app}(f, a) : B} \qquad \frac{\Psi \mid \Gamma, x : A \vdash t : B}{\Psi \mid \Gamma \vdash \mathsf{abs}(x.\, t) : A \Rightarrow B}$$

We can then give, for each $A, B \in \mathsf{Ty}$, an $((A; B), (\diamond; A); B)$-ary equation for β-equality, and an $((\diamond; A \Rightarrow B); (A \Rightarrow B))$-ary equation for η-equality:

$$M_1 : (A; B), M_2 : (\diamond; A) \mid \diamond \vdash \quad \mathsf{app}(\mathsf{abs}(x.\, M_1(x)), M_2()) \approx M_1(M_2()) \quad : B \qquad (\beta)$$
$$M : (\diamond; A \Rightarrow B) \mid \diamond \vdash \quad \mathsf{abs}(x.\, \mathsf{app}(M(), x)) \approx M() \quad : A \Rightarrow B \qquad (\eta)$$

The signature Σ^Λ together with these equations forms the Ty-sorted second-order presentation $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$ of the STLC with $\beta\eta$-equality. Note that second-order equations permit the expression of *axiom schemata*, as axioms containing metavariables (in both the traditional and precise sense of the term "metavariable") [12, 5]. Without second-order equations, one would have to add β and η equations for each instantiation of the metavariables in the rules above.

▶ **Definition 18.** *If $(\Psi \mid \Gamma \vdash u_i : A_i)_i$ and $\Psi \mid x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ are terms over an S-sorted second-order signature Σ, then their* substitution $\Psi \mid \Gamma \vdash t\{x_i \mapsto u_i\}_i : B$ *is defined by recursion on t:*

$$x_j\{x_i \mapsto u_i\}_i = u_j \qquad M(t_1, \ldots t_m)\{x_i \mapsto u_i\}_i = M(t_1\{x_i \mapsto u_i\}_i, \ldots, t_m\{x_i \mapsto u_i\}_i)$$
$$\mathsf{o}((\vec{y_1}.\, t_1), \ldots, \mathsf{o}(\vec{y_k}.\, t_k))\{x_i \mapsto u_i\}_i = \mathsf{o}((\vec{y_1}.\, t_1\{x_i \mapsto u_i\}_i), \ldots, (\vec{y_k}.\, t_k\{x_i \mapsto u_i\}_i))$$

(On the right-hand side of the definition on operators, the terms t_i are weakened, and we omit from the substitution variables that are mapped to themselves.) If instead we have terms $(\Psi \mid \Gamma, \vec{x}_i : \Delta_i \vdash u_i : A_i)_i$ and $M_1 : (\Delta_1; A_1), \ldots, M_n : (\Delta_n; A_n) \mid \Gamma' \vdash t : B$ then their metasubstitution *$\Psi \mid \Gamma, \Gamma' \vdash t\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i : B$ is defined using ordinary substitution by recursion on t:*

$$x\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i = x \qquad M_j(t_1, \ldots, t_m)\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i = u_j\{x_{jk} \mapsto t_k\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i\}_k$$
$$\mathsf{o}((\vec{y_1}.\, t_1), \ldots, (\vec{y_k}.\, t_k))\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i = \mathsf{o}((\vec{y_1}.\, t_1\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i), \ldots, (\vec{y_k}.\, t_k\{M_i \mapsto (\vec{x}_i.\, u_i)\}_i))$$

3.1 Algebras

The algebras for a presentation are the abstract clones interpreting each of the operations of the signature, subject to the axioms of the presentation. In other words, a presentation is a specification of structure, while the algebras are the realizations, or models, of that structure. For instance, in the first-order setting, the algebras for the presentation of monoids form (set-theoretic) monoids.

▶ **Definition 19.** *An* algebra $(\mathbf{X}, [\![-]\!])$ *for an S-sorted second-order signature Σ (called "presentation clones" in [32]) consists of an S-sorted clone $\mathbf{X}$ and, for each context Γ and $((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$-ary operator o, a function $[\![\mathsf{o}]\!]_\Gamma : \prod_i X(\Gamma, \Delta_i; A_i) \to X(\Gamma; B)$ such that, for all substitutions $\boldsymbol{\sigma} \in X(\Xi; \Gamma)$ and tuples of terms $(t_i \in X(\Gamma, \Delta_i; A_i))_i$,*

$$([\![\mathsf{o}]\!]_\Gamma(t_1, \ldots, t_n))[\boldsymbol{\sigma}] = [\![\mathsf{o}]\!]_\Xi(t_1[\mathsf{lift}_{\Delta_1}\boldsymbol{\sigma}], \ldots, t_n[\mathsf{lift}_{\Delta_n}\boldsymbol{\sigma}])$$

A homomorphism *$f : (\mathbf{X}, [\![-]\!]) \to (\mathbf{X}', [\![-]\!]')$ of Σ-algebras is a homomorphism $f : \mathbf{X} \to \mathbf{X}'$ of clones such that, for all $\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$ and $(t_i \in X(\Gamma, \Delta_i; A_i))_i$,*

$$f_{\Gamma;B}([\![\mathsf{o}]\!]_\Gamma(t_1, \ldots, t_n)) \;=\; [\![\mathsf{o}]\!]'_\Gamma(f_{\Gamma,\Delta_1;A_1}t_1, \ldots, f_{\Gamma,\Delta_n;A_n}t_n)$$

The interpretation of operators in a Σ-algebra $(\mathbf{X}, [\![-]\!])$ extends to an interpretation $[\![t]\!]_\Gamma : \prod_i X(\Gamma, \Delta_i; A_i) \to X(\Gamma, \Xi; B)$ of each term $M_1 : (\Delta_1; A_1), \ldots, M_n : (\Delta_n; A_n) \mid \vec{x} : \Xi \vdash t : B$ as follows (where n is the length of Γ):

$$[\![x_i]\!]_\Gamma(\boldsymbol{\sigma}) = \mathsf{var}_{n+i}^{(\Gamma,\Xi)}$$
$$[\![M_i(t_1, \ldots, t_m)]\!]_\Gamma(\boldsymbol{\sigma}) = \sigma_i[\mathsf{var}_1^{(\Gamma,\Xi)}, \ldots, \mathsf{var}_n^{(\Gamma,\Xi)}, [\![t_1]\!]_\Gamma(\boldsymbol{\sigma}), \ldots, [\![t_m]\!]_\Gamma(\boldsymbol{\sigma})]$$
$$[\![\mathsf{o}((\vec{x}_1.\, t_1), \ldots, (\vec{x}_m.\, t_m))]\!]_\Gamma(\boldsymbol{\sigma}) = [\![\mathsf{o}]\!]_{\Gamma,\Xi}([\![t_1]\!]_\Gamma(\boldsymbol{\sigma}), \ldots, [\![t_m]\!]_\Gamma(\boldsymbol{\sigma}))$$

▶ **Definition 20.** *An algebra $(\mathbf{X}, [\![-]\!])$ for a second-order presentation $\boldsymbol{\Sigma} = (\Sigma, E)$ is a Σ-algebra such that, for all equations $(t, u) \in E(\Psi; A)$ and contexts Γ, we have $[\![t]\!]_\Gamma = [\![u]\!]_\Gamma$. We let $\boldsymbol{\Sigma}$-**Alg** be the category of $\boldsymbol{\Sigma}$-algebras and all Σ-algebra homomorphisms between them.*

▶ **Example 21.** An algebra for the presentation $\boldsymbol{\Sigma}^{\Lambda_{\beta\eta}}$ of the STLC with $\beta\eta$-equality consists of a Ty-sorted clone $\mathbf{X}$ and functions

$$[\![\mathsf{app}]\!]_\Gamma : X(\Gamma; A \Rightarrow B) \times X(\Gamma; A) \to X(\Gamma; B) \qquad [\![\mathsf{abs}]\!]_\Gamma : X(\Gamma, A; B) \to X(\Gamma; A \Rightarrow B)$$

that commute with substitution and satisfy

$$[\![\mathsf{app}]\!]_\Gamma([\![\mathsf{abs}]\!]_\Gamma(t),\, t') = t[\mathsf{var}^{(\Gamma)}, t'] \quad \text{for } t \in X(\Gamma, A; B), t' \in X(\Gamma; A) \qquad (\beta)$$
$$[\![\mathsf{abs}]\!]_\Gamma([\![\mathsf{app}]\!]_{\Gamma,A}(t[\triangleright\mathsf{wk}_A^{(\Gamma)}], \mathsf{var}_{n+1}^{(\Gamma,A)})) = t \quad \text{for } t \in X(\Gamma; A \Rightarrow B) \qquad (\eta)$$

For each set Z we have a set-theoretic interpretation of the STLC, which forms a $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra $(\mathcal{M}_Z, \mathcal{M}_Z[\![-]\!])$ as follows. Define interpretations $\mathcal{M}_Z[\![A]\!] \in \mathbf{Set}$ of each sort $A \in \mathsf{Ty}$ recursively by setting $\mathcal{M}_Z[\![\mathsf{b}]\!] = Z$ and $\mathcal{M}_Z[\![A \Rightarrow B]\!] = \mathbf{Set}(\mathcal{M}_Z[\![A]\!], \mathcal{M}_Z[\![B]\!])$ (where $\mathbf{Set}(Y, Y')$ is the set of functions $Y \to Y'$). We then have a Ty-sorted clone $\mathcal{M}_Z$, where the sets of terms are given by $\mathcal{M}_Z(A_1, \dots, A_n; B) = \mathbf{Set}(\prod_i \mathcal{M}_Z[\![A_i]\!], \mathcal{M}_Z[\![B]\!])$, the variables by projections $\mathsf{var}_i^{(\Gamma)} = \pi_i$, and substitution by $f[\boldsymbol{\sigma}] = (\xi \mapsto f(\sigma_1(\xi), \dots, \sigma_n(\xi)))$. This forms a $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra, with interpretations of the operators given by function application and currying. More generally, the interpretation of the STLC in any cartesian-closed category $\mathbf{C}$ with a specified object $Z \in \mathbf{C}$ forms a $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra taking $\mathcal{M}_Z(A_1, \dots, A_n; B) = \mathbf{C}(\prod_i \mathcal{M}_Z[\![A_i]\!], \mathcal{M}_Z[\![B]\!])$ to be the sets of terms, where $\mathcal{M}_Z[\![\mathsf{b}]\!] = Z$ and $\mathcal{M}_Z[\![A \Rightarrow B]\!] = \mathcal{M}_Z[\![B]\!]^{\mathcal{M}_Z[\![A]\!]}$.

The cartesian structure of $\mathbf{Clone}(S)$ lifts to $\mathbf{\Sigma}\text{-}\mathbf{Alg}$ for every presentation $\mathbf{\Sigma}$: the clone $\mathbf{1}$ uniquely forms a $\mathbf{\Sigma}$-algebra, and the product $(\mathbf{X}_1, [\![-]\!]_1) \times (\mathbf{X}_2, [\![-]\!]_2)$ is the clone $\mathbf{X}_1 \times \mathbf{X}_2$ equipped with interpretations $[\![\mathsf{o}]\!]_\Gamma((\sigma_{11}, \sigma_{21}), \dots, (\sigma_{1n}, \sigma_{2n})) = ([\![\mathsf{o}]\!]_{1,\Gamma}(\boldsymbol{\sigma}_1), [\![\mathsf{o}]\!]_{2,\Gamma}(\boldsymbol{\sigma}_2))$.

4 Free algebras

Second-order S-sorted presentations $\mathbf{\Sigma}$ can be viewed as descriptions of simple type theories for which S is the set of types. In particular, the operators specify the term formers of the type theory (such as λ-abstraction, or application). From this perspective, the syntax of the type theory described by $\mathbf{\Sigma}$ is the *initial* $\mathbf{\Sigma}$-algebra: there is a unique Σ-algebra homomorphism from the algebra formed by the syntax to any other algebra, given by induction on terms. More generally, given the syntax of an existing theory in the form of a clone $\mathbf{X}$, the *free* $\mathbf{\Sigma}$-algebra on $\mathbf{X}$ is given by augmenting $\mathbf{X}$ by the operators and equations of $\mathbf{\Sigma}$; or, from another perspective, augmenting the type theory described by $\mathbf{\Sigma}$ with the operations specified by $\mathbf{X}$. For example, the free $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra on $\mathbf{GS}_V$ (Example 13) may be seen as the STLC extended by additional term formers (get and $\mathsf{put}_{v_1}, \dots, \mathsf{put}_{v_k}$) representing the side-effects of global state.

▶ **Definition 22.** *Suppose $\mathbf{\Sigma} = (\Sigma, E)$ is an S-sorted second-order presentation and $\mathbf{X}$ is an S-sorted clone. A $\mathbf{\Sigma}$-algebra $F_{\mathbf{\Sigma}}\mathbf{X}$ equipped with a clone homomorphism $\eta_{\mathbf{X}} : \mathbf{X} \to F_{\mathbf{\Sigma}}\mathbf{X}$ is the* free $\mathbf{\Sigma}$-algebra *on $\mathbf{X}$ if, for any other $\mathbf{\Sigma}$-algebra $(\mathbf{Y}, [\![-]\!])$ and clone homomorphism $f : \mathbf{X} \to \mathbf{Y}$, there is a unique Σ-algebra homomorphism $f^\dagger : F_{\mathbf{\Sigma}}\mathbf{X} \to (\mathbf{Y}, [\![-]\!])$ such that $f^\dagger \circ \eta_{\mathbf{X}} = f$. The* initial $\mathbf{\Sigma}$-algebra *is the free $\mathbf{\Sigma}$-algebra on Var_S.*

▶ **Example 23.** Recall the presentation $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$ of the STLC with $\beta\eta$-equality from Example 17. The initial $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra is the clone $\mathbf{\Lambda}_{\beta\eta}$ of STLC terms up to $\approx_{\beta\eta}$ (Example 5), with the operators app and abs interpreted as

$$((f, a) \mapsto \mathsf{app}\ f\ a) : \Lambda_{\beta\eta}(\Gamma; A \Rightarrow B) \times \Lambda_{\beta\eta}(\Gamma; A) \to \Lambda_{\beta\eta}(\Gamma; B)$$
$$(t \mapsto \lambda x : A.\, t) : \Lambda_{\beta\eta}(\Gamma, A; B) \to \Lambda_{\beta\eta}(\Gamma; A \Rightarrow B)$$

The free $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra on the clone $\mathbf{GS}_V$ of global V-valued state (Example 13) can be described as follows for $V = \{v_1, \dots, v_k\}$. The underlying Ty-sorted clone is defined in the same way as $\mathbf{\Lambda}_{\beta\eta}$, but with the following additional term formers and equations (omitting the typing constraints on equations).

$$\frac{\Gamma \vdash t_1 : \mathsf{b} \quad \cdots \quad \Gamma \vdash t_k : \mathsf{b}}{\Gamma \vdash \mathsf{get}(t_1, \dots, t_k) : \mathsf{b}} \qquad \frac{\Gamma \vdash t : \mathsf{b}}{\Gamma \vdash \mathsf{put}_{v_i}(t) : \mathsf{b}}\ (i \le k)$$

$$\mathsf{get}(\mathsf{put}_{v_1}(t), \dots, \mathsf{put}_{v_k}(t)) \approx_{\beta\eta} t$$
$$\mathsf{put}_{v_i}(\mathsf{get}(t_1, \dots, t_k)) \approx_{\beta\eta} \mathsf{put}_{v_i}(t_i) \qquad (i \le k)$$
$$\mathsf{put}_{v_i}(\mathsf{put}_{v_j}(t)) \approx_{\beta\eta} \mathsf{put}_{v_j}(t) \qquad (i, k \le k)$$

Terms

$$\frac{}{\Gamma, x : A, \Delta \vdash_{\mathbf{X}} x : A} \qquad \frac{\Gamma \vdash_{\mathbf{X}} t_1 : A_1 \quad \cdots \quad \Gamma \vdash_{\mathbf{X}} t_n : A_n}{\Gamma \vdash_{\mathbf{X}} f(t_1, \ldots, t_n) : B} \; (f \in X(A_1, \ldots, A_n; B))$$

$$\frac{\Gamma, \vec{x}_1 : \Delta_1 \vdash_{\mathbf{X}} t_1 : A_1 \quad \cdots \quad \Gamma, \vec{x}_n : \Delta_n \vdash_{\mathbf{X}} t_n : A_n}{\Gamma \vdash_{\mathbf{X}} \mathsf{o}((\vec{x}_1 . t_1), \ldots, (\vec{x}_n . t_n)) : B} \; (\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B))$$

Equations (reflexivity, symmetry, transitivity omitted)

$$\frac{\Gamma \vdash_{\mathbf{X}} t_1 \approx u_1 : A_1 \quad \cdots \quad \Gamma \vdash_{\mathbf{X}} t_n \approx u_n : A_n}{\Gamma \vdash_{\mathbf{X}} f(t_1, \ldots, t_n) \approx f(u_1, \ldots, u_n) : B} \; (f \in X(A_1, \ldots, A_n; B))$$

$$\frac{\Gamma, \vec{x}_1 : \Delta_1 \vdash_{\mathbf{X}} t_1 \approx u_1 : A_1 \quad \cdots \quad \Gamma, \vec{x}_n : \Delta_n \vdash_{\mathbf{X}} t_n \approx u_n : A_n}{\Gamma \vdash_{\mathbf{X}} \mathsf{o}((\vec{x}_1 . t_1), \ldots, (\vec{x}_n . t_n)) \approx \mathsf{o}((\vec{x}_1 . u_1), \ldots, (\vec{x}_n . u_n)) : B} \; (\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots; B))$$

$$\frac{\Gamma, \vec{x}_1 : \Delta_1 \vdash_{\mathbf{X}} t_1 \approx u_1 : A_1 \quad \cdots \quad \Gamma, \vec{x}_n : \Delta_n \vdash_{\mathbf{X}} t_n \approx u_n : A_n}{\Gamma \vdash_{\mathbf{X}} t'\{M_i \mapsto (\vec{x}_i . t_i)\}_i \approx u'\{M_i \mapsto (\vec{x}_i . u_i)\}_i : B} \; ((t', u') \in E((\Delta_1; A_1), \ldots; B))$$

$$\frac{\Gamma \vdash_{\mathbf{X}} t_1 : A_1 \quad \cdots \quad \Gamma \vdash_{\mathbf{X}} t_n : A_n}{\Gamma \vdash_{\mathbf{X}} t_i \approx \mathsf{var}_i^{(A_1, \ldots, A_n)}(t_1, \ldots, t_n) : B} \; (i \leq n)$$

$$\frac{\Gamma \vdash_{\mathbf{X}} t_1 : A_1 \quad \cdots \quad \Gamma \vdash_{\mathbf{X}} t_n : A_n}{\Gamma \vdash_{\mathbf{X}} f(\sigma_1(t_1, \ldots, t_n), \ldots, \sigma_k(t_1, \ldots, t_n)) \approx (f[\boldsymbol{\sigma}])(t_1, \ldots, t_n) : B} \; (f \in X(A'_1, \ldots, A'_k; B), \boldsymbol{\sigma} \in X(A_1, \ldots, A_n; A'_1, \ldots, A'_k))$$

Figure 1 Construction of the free (Σ, E)-algebra on a clone $\mathbf{X} = (X, \mathsf{var}, \mathsf{subst})$.

This forms a $\boldsymbol{\Sigma}^{\Lambda_{\beta\eta}}$-algebra in the same way as $\boldsymbol{\Lambda}_{\beta\eta}$ above. The morphism $\eta_{\mathbf{GS}_V}$ is given by $\eta_{\mathbf{GS}_V}(\mathsf{get}(t_1, \ldots, t_k)) = \mathsf{get}(\eta_{\mathbf{GS}_V}(t_1), \ldots, \eta_{\mathbf{GS}_V}(t_k))$ and $\eta_{\mathbf{GS}_V}(\mathsf{put}_{v_i}(t)) = \mathsf{put}_{v_i}(\eta_{\mathbf{GS}_V}(t))$.

If $\boldsymbol{\Sigma}'$ is a first-order presentation, the free $\boldsymbol{\Sigma}$-algebra on $\mathbf{Term}^{\boldsymbol{\Sigma}'}$ is closed under the operators of $\boldsymbol{\Sigma}'$: each $\mathsf{o} \in \boldsymbol{\Sigma}'(A_1, \ldots, A_n; B)$ induces a term $\eta(\mathsf{o}(x_1, \ldots, x_n)) \in F_{\boldsymbol{\Sigma}}\mathbf{X}(A_1, \ldots, A_n; B)$ and hence functions $(\boldsymbol{\sigma} \mapsto \eta(\mathsf{o}(\vec{x}))[\boldsymbol{\sigma}]) : F_{\boldsymbol{\Sigma}}\mathbf{X}(\Gamma; A_1, \ldots, A_n) \to F_{\boldsymbol{\Sigma}}\mathbf{X}(\Gamma; B)$.

We show that free algebras for any signature, and on any clone, exist, by constructing them explicitly. Existence of these free algebras facilitates the developments in the next sections. However, note that we do not rely on the explicit description: after this section, we reason about free algebras solely using the universal property in Definition 22. This is important, as we wish to reason about type theories independently of their syntax, which leads to greatly simplified proofs. (It is also possible to prove the existence of free algebras entirely abstractly using a monadicity theorem and Remark 7, avoiding concrete syntax.)

In universal algebra, free algebras of first-order presentations are constructed in two steps: by first closing a sort-indexed set X of constants under the operators of the presentation; and then quotienting the terms by the equations of the presentation. Figure 1 gives the analogous construction in the second-order setting. First, we construct terms $\Gamma \vdash_{\mathbf{X}} t : B$ from variables, the terms of the clone $f \in X(A_1, \ldots, A_n; B)$ (viewed as function symbols), and the operators of the presentation $\boldsymbol{\Sigma}$. Second, we quotient by the equivalence relation $\approx$ generated by congruence, the equations of $\boldsymbol{\Sigma}$ (using metasubstitution), and rules imposing compatibility with the clone structure of $\mathbf{X}$. The clone $F_{\boldsymbol{\Sigma}}\mathbf{X}$ has terms $F_{\boldsymbol{\Sigma}}\mathbf{X}(\Gamma; B) = \{\Gamma \vdash_{\mathbf{X}} t : B\}/\approx$, with variables and substitution defined in the evident way; the homomorphism $\eta_{\mathbf{X}} : \mathbf{X} \to F_{\boldsymbol{\Sigma}}\mathbf{X}$

sends $t \in X(\Gamma; B)$ to $x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathbf{X}} t(x_1, \ldots, x_n) : B$, where $\Gamma = [A_1, \ldots, A_n]$.

▶ **Proposition 24.** *For every S-sorted second-order presentation $\mathbf{\Sigma}$ and S-sorted clone $\mathbf{X}$, the free $\mathbf{\Sigma}$-algebra $F_{\mathbf{\Sigma}}\mathbf{X}$ exists.*

The forgetful functor $\mathbf{\Sigma}\text{-}\mathbf{Alg} \to \mathbf{Clone}(S)$ therefore has a left adjoint (in fact, it is monadic).

5 Induction over second-order syntax

We now describe how the formalism of abstract clones may be used to prove properties of simple type theories. To begin, we consider predicates over abstract clones, which are predicates over the terms of the type theory induced by the clone, closed under the structural operations of variable projection and substitution. Below, we extend each family of subsets $P(\Gamma; A) \subseteq Y(\Gamma; A)$ to contexts by defining $P(\Gamma; A_1, \ldots, A_n)$ to be the set of all substitutions $\boldsymbol{\sigma} \in Y(\Gamma; A_1, \ldots, A_n)$ such that $\sigma_i \in P(\Gamma; A_i)$ for all $i \leq n$.

▶ **Definition 25.** *A* predicate *P over an S-sorted clone $\mathbf{X}$ consists of a subset $P(\Gamma; A) \subseteq X(\Gamma; A)$ for each $(\Gamma; A) \in S^* \times S$ such that, for all contexts $\Gamma = [A_1, \ldots, A_n]$ and $i \leq n$, we have $\mathsf{var}_i^{(\Gamma)} \in P(\Gamma; A_i)$, and, for all $t \in P(\Delta; B)$ and $\boldsymbol{\sigma} \in P(\Gamma; \Delta)$, we have $t[\boldsymbol{\sigma}] \in P(\Gamma; B)$.*

Closure under variables and under substitution imply that P forms a clone $\mathbf{P}$ whose inclusion $\mathbf{P} \hookrightarrow \mathbf{X}$ into $\mathbf{X}$ is a clone homomorphism. Predicates over S-sorted clones are equivalently the subobjects in $\mathbf{Clone}(S)$, and are hence closed under arbitrary conjunction, existential quantification, and quotients of equivalence relations. (This follows from Remark 7, since varieties are exact categories [7, Theorem 5.11], and all exact categories enjoy these properties.) They are also closed under context extension: if P is a predicate over $\mathbf{X}$ and Ξ is a context, then $\Uparrow^{\Xi} P$ is a predicate over $\Uparrow^{\Xi}\mathbf{X}$.

We present a meta-theorem for establishing properties of simple type theories.

▶ **Theorem 26** (Induction principle for second-order syntax)**.** *Suppose that $(\mathbf{Y}, [\![-]\!])$ is an algebra for an S-sorted second-order presentation $\mathbf{\Sigma}$, that $f : \mathbf{X} \to \mathbf{Y}$ is a clone homomorphism from an S-sorted clone $\mathbf{X}$, and that P is a predicate over $\mathbf{Y}$. If*

- *for all operators $\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$, contexts $\Gamma \in S^*$, and tuples of terms $(t_i \in P(\Gamma, \Delta_i; A_i))_i$ we have $[\![\mathsf{o}]\!]_\Gamma(t_1, \ldots, t_n) \in P(\Gamma; B)$;*
- *for all terms $t \in X(\Gamma; A)$ we have $f_{\Gamma;A}(t) \in P(\Gamma; A)$,*

then, for all free terms $t \in (F_{\mathbf{\Sigma}}\mathbf{X})(\Gamma; A)$, we have $f^{\dagger}_{\Gamma;A}(t) \in P(\Gamma; A)$.

Proof. The predicate P is closed under operators, so the interpretations of operators in $\mathbf{Y}$ make $\mathbf{P}$ into a $\mathbf{\Sigma}$-algebra. The image of f is contained in P, so f forms a clone homomorphism $\mathbf{X} \to \mathbf{P}$. By the universal property of the free algebra $F_{\mathbf{\Sigma}}\mathbf{X}$, we therefore have an algebra homomorphism $F_{\mathbf{\Sigma}}\mathbf{X} \to \mathbf{P}$. This necessarily sends $t \in (F_{\mathbf{\Sigma}}\mathbf{X})(\Gamma; A)$ to $f^{\dagger}_{\Gamma;A}(t) \in P(\Gamma; A)$. ◀

We give two corollaries of this induction principle. The first is for proving properties of closed terms, which take the form of families of subsets $P(A) \subseteq Y(\diamond; A)$. Given such a family P, let $P(A_1, \ldots, A_n)$ be the set of all $\boldsymbol{\sigma} \in Y(\diamond; A_1, \ldots, A_n)$ such that $\sigma_i \in P(A_i)$ for all $i \leq n$, and define a predicate $P^\sharp$ over $\mathbf{Y}$ by $P^\sharp(\Gamma; A) = \{t \in Y(\Gamma; A) \mid \forall \boldsymbol{\sigma} \in P(\Gamma).\, t[\boldsymbol{\sigma}] \in P(A)\}$. Applying the induction principle above to $P^\sharp$ gives us the following.

▶ **Corollary 27.** *Suppose that $\mathbf{\Sigma}$ is an S-sorted second-order presentation, that $(\mathbf{Y}, [\![-]\!])$ is a $\mathbf{\Sigma}$-algebra, and that $(P(A) \subseteq Y(\diamond; A))_{A \in S}$ is a family of subsets. For every S-sorted clone $\mathbf{X}$ and clone homomorphism $f : \mathbf{X} \to \mathbf{Y}$, if*

- *for every operator* $\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$ *and tuple* $(t_i \in P^\sharp(\Delta_i; A_i))_{i \le n}$ *of terms, we have* $[\![\mathsf{o}]\!]_\diamond(t_1, \ldots, t_n) \in P(B)$*;*
- *for every term* $t \in X(\Gamma; B)$*, we have* $f_{\Gamma;B}(t) \in P^\sharp(\Gamma; B)$*,*

then, for every type $A \in S$ *and free term* $t \in (F_{\mathbf{\Sigma}}\mathbf{X})(\diamond; A)$*, we have* $f^\dagger_{\diamond;A}(t) \in P(A)$*.*

Proof. $P^\sharp(\diamond; A) = P(A)$, so it suffices to apply Theorem 26 to the predicate $P^\sharp$. We therefore check the two assumptions of that theorem. Closure of $P^\sharp$ under f is immediate; and $P^\sharp$ is closed under operators because, if $(t_i \in P^\sharp(\Gamma, \Delta_i; A_i))_i$ and $\boldsymbol{\sigma} \in P(\Gamma)$, then $t_i[\mathsf{lift}_{\Delta_i}\boldsymbol{\sigma}] \in P^\sharp(\Delta_i; A_i)$ for all $i \le n$, so that $[\![\mathsf{o}]\!]_\Gamma(t_1, \ldots, t_n)[\boldsymbol{\sigma}] = [\![\mathsf{o}]\!]_\diamond(t_1[\mathsf{lift}_{\Delta_1}\boldsymbol{\sigma}], \ldots, t_n[\mathsf{lift}_{\Delta_n}\boldsymbol{\sigma}]) \in P(B)$. ◀

Families of subsets $P(A) \subseteq X(\diamond; A)$ are closed under arbitrary conjunction and disjunction, complements, and universal and existential quantification. They form a *tripos* [25, 34], and hence a model of higher-order logic over $\mathbf{Clone}(S)$; the tripos-theoretic methods of Hofmann [22] carry over in this way to the setting of abstract clones.

The second corollary is for families of subsets $P(\Gamma; A) \subseteq Y(\Gamma; A)$ that are not known to be closed under substitution. (In some cases proving closure under substitution requires an induction over terms, but induction over terms is what this section is meant to enable.) Analogously to the construction $P^\sharp$ for predicates over closed terms, we define a predicate $P^\flat$ over $\mathbf{Y}$ by $P^\flat(\Gamma; A) = \{t \in Y(\Gamma; A) \mid \forall \Delta, \boldsymbol{\sigma} \in P(\Delta; \Gamma).\, t[\boldsymbol{\sigma}] \in P(\Delta; A)\}$.

▶ **Corollary 28.** *Suppose that* $\mathbf{\Sigma}$ *is an* S*-sorted second-order presentation, that* $(\mathbf{Y}, [\![-]\!])$ *is a* $\mathbf{\Sigma}$*-algebra, and that* $(P(\Gamma; A) \subseteq Y(\Gamma; A))_{(\Gamma;A) \in S^* \times S}$ *is a family of subsets. For every* S*-sorted clone* $\mathbf{X}$ *and homomorphism* $f : \mathbf{X} \to \mathbf{Y}$*, if*

- *for every context* Γ *we have* $\mathsf{var}^{(\Gamma)} \in P(\Gamma; \Gamma)$*;*
- *for every context* Γ*, operator* $\mathsf{o} \in \Sigma((\Delta_1; A_1), \ldots, (\Delta_n; A_n); B)$*, and tuple of terms* $(t_i \in P^\flat(\Gamma, \Delta_i; A_i))_i$ *we have* $[\![\mathsf{o}]\!]_\Gamma(t_1, \ldots, t_n) \in P^\flat(\Gamma; B)$*;*
- *for every term* $t \in X(\Gamma; B)$ *we have* $f_{\Gamma;B}(t) \in P^\flat(\Gamma; B)$*,*

then, for every free term $t \in (F_{\mathbf{\Sigma}}\mathbf{X})(\Gamma; A)$*, we have* $f^\dagger_{\Gamma;A}(t) \in P(\Gamma; A)$*.*

Proof. We can apply Theorem 26 to $P^\flat$ because it is closed under operators and under f. Hence $f^\dagger_{\Gamma;A}(t) \in P^\flat(\Gamma; A)$ for each $t \in (F_{\mathbf{\Sigma}}\mathbf{X})(\Gamma; A)$, and so $\mathsf{var}^{(\Gamma)} \in P(\Gamma; \Gamma)$ implies that $f^\dagger_{\Gamma;A}(t) = (f^\dagger_{\Gamma;A}(t))[\mathsf{var}^{(\Gamma)}] \in P(\Gamma; A)$. ◀

The above corollaries are designed to enable logical relations arguments, in which the fundamental lemma is proven using an induction hypothesis that quantifies over substitutions. In particular, in Corollary 28 we require $P^\flat$ to be closed under the operators, rather than P. There is a third corollary that instead requires closure of P under operators (this would essentially be the principle of induction on $\Gamma \vdash_{\mathbf{X}} t : A$), but this is less useful for our purposes.

6 Logical relations

We provide two extended examples of proofs using the induction principles of the previous section, both involving the presentation $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$ of the STLC with $\beta\eta$-equality. The first is a proof of the adequacy of the set-theoretic model of the STLC, which uses induction on closed terms; the second is a proof that every STLC term is $\beta\eta$-equal to one in normal form, using induction on open terms. Both examples are logical relations proofs, the former using ordinary logical relations and the latter using Kripke relations [27]. Though both properties are known to hold, these proofs in particular illustrate that our induction principles are powerful enough to justify logical relations arguments. We include a proof of normalization for the STLC with global state in Appendix A, as a further motivating example.

6.1 Closed terms and adequacy

We say that a model $\mathcal{M}$ of the STLC is *adequate* when, for all closed terms t and u of the base type b, if $\mathcal{M}[\![t]\!] = \mathcal{M}[\![u]\!]$, then t and u are equal up to $\beta\eta$-equality. (In adequate models, equality of denotations implies observational equivalence for terms of arbitrary types.)

We first show that we can perform logical relations arguments for the STLC using our induction principle: specifically Corollary 27. Fix a $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra $(\mathbf{Y}, [\![-]\!])$, homomorphism $f : \mathbf{X} \to \mathbf{Y}$ from some clone $\mathbf{X}$, and a subset $P(\mathsf{b}) \subseteq Y(\diamond; \mathsf{b})$ of closed terms of base type. We extend P to a family of subsets $P(A) \subseteq Y(\diamond; A)$ in the standard way for logical relations:

$$P(A \Rightarrow B) = \{t \in Y(\diamond; A \Rightarrow B) \mid \forall a \in P(A).\, [\![\mathsf{app}]\!]_\diamond(t, a) \in P(B)\}$$

Applying Corollary 27 to P gives us the following:

▶ **Lemma 29.** *If, for every context Γ and term $t \in X(\Gamma; B)$, we have $f_{\Gamma;B}(t) \in P^\sharp(\Gamma; B)$, then, for every free term $t \in (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{X})(\diamond; A)$, we have $f^\dagger_{\diamond;A}(t) \in P(A)$.*

Proof. The only non-trivial assumption of Corollary 27 is closure under operators. Closure under app is immediate from the definition of the logical relation. Closure under abs holds because, if $t \in P^\sharp(A; B)$, then, for all $a \in P(A)$, we have $[\![\mathsf{app}]\!]_\diamond([\![\mathsf{abs}]\!]_\diamond(t), a) = t[a] \in P(B)$ using the β law, so that $[\![\mathsf{abs}]\!]_\diamond(t) \in P(A \Rightarrow B)$. ◀

Note that if terms are generated only by λ-abstraction and application then there are no closed terms of base type. For a more interesting example, we therefore consider the STLC with booleans (where the base type b is the type of booleans). Consider the Ty-sorted first-order presentation $\mathbf{\Sigma}_{\mathbf{Bool}}$ with two $(\diamond; \mathsf{b})$-ary operators true, false, and, for each $A \in \mathsf{Ty}$, a $(\mathsf{b}, A, A; A)$-ary operator ite ("if-then-else"), along with two equations:

$$y : A, z : A \vdash \mathsf{ite}(\mathsf{true}(), y, z) \approx y : A \qquad y : A, z : A \vdash \mathsf{ite}(\mathsf{false}(), y, z) \approx z : A$$

Let $\mathbf{Bool}$ be the Ty-sorted clone that is presented by $\mathbf{\Sigma}_{\mathbf{Bool}}$.

Consider the free $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra $F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool}$, and the $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra $\mathcal{M}_{\mathbb{B}}$ (as defined in Example 21) with $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$. The former should be thought of as containing the terms of the STLC with booleans (we make this precise below); the latter is the usual model in $\mathbf{Set}$. Both have clone homomorphisms from $\mathbf{Bool}$: the free algebra has $\eta_{\mathbf{Bool}} : \mathbf{Bool} \to F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool}$; the model $\mathcal{M}_{\mathbb{B}}$ has the unique $g : \mathbf{Bool} \to \mathcal{M}_{\mathbb{B}}$ such that

$$g_{\Gamma;\mathsf{b}}(\mathsf{true}()) = \zeta \mapsto \mathrm{tt} \qquad g_{\Gamma;\mathsf{b}}(\mathsf{false}()) = \zeta \mapsto \mathrm{ff}$$

$$g_{\Gamma;A}(\mathsf{ite}(t_1, t_2, t_3)) = \zeta \mapsto \begin{cases} g_{\Gamma;A}(t_2)(\zeta) & \text{if } g_{\Gamma;\mathsf{b}}(t_1)(\zeta) = \mathrm{tt} \\ g_{\Gamma;A}(t_3)(\zeta) & \text{if } g_{\Gamma;\mathsf{b}}(t_1)(\zeta) = \mathrm{ff} \end{cases}$$

The algebra homomorphism $g^\dagger : F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool} \to \mathcal{M}_{\mathbb{B}}$ gives the interpretation of STLC terms in the model. Define a subset $P(\mathsf{b}) \subseteq (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool} \times \mathcal{M}_{\mathbb{B}})(\diamond; \mathsf{b}) = (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool})(\diamond; \mathsf{b}) \times \mathbb{B}$ by

$$P(\mathsf{b}) = \{(\eta_{\mathbf{Bool}}(\mathsf{true}()), \mathrm{tt}), (\eta_{\mathbf{Bool}}(\mathsf{false}()), \mathrm{ff})\}$$

This extends to a logical relation P by the definition on function types above and, by a simple proof, satisfies the precondition of Lemma 29, where the clone homomorphism f is $\langle \eta_{\mathbf{Bool}}, g\rangle : \mathbf{Bool} \to F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool} \times \mathcal{M}_{\mathbb{B}}$. Hence, for all $t \in (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool})(\diamond; A)$, we have $(t, g^\dagger_{\diamond;A}(t)) = \langle \eta_{\mathbf{Bool}}, g\rangle^\dagger_{\diamond;A}(t) \in P(A)$. When $A = \mathsf{b}$ this immediately implies, for all $t, t' \in (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool})(\diamond; \mathsf{b})$, that if $g^\dagger_{\diamond;\mathsf{b}}(t) = g^\dagger_{\diamond;\mathsf{b}}(t')$ then $t = t'$.

This last property is seen to be adequacy of the set-theoretic model $\mathcal{M}_{\mathbb{B}}$ as follows. Let $\mathbf{\Lambda}_{\beta\eta\mathbf{Bool}}$ be the Ty-sorted clone that is defined in the same way as $\mathbf{\Lambda}_{\beta\eta}$ (Example 5) but with additional term formers and equations (omitting the typing constraints on equations):

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{b}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{b}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{b} \quad \Gamma \vdash t_2 : A \quad \Gamma \vdash t_3 : A}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : A} \qquad \begin{array}{l}\mathsf{if\ true\ then}\ t_2\ \mathsf{else}\ t_3 \approx_{\beta\eta} t_2\\ \mathsf{if\ false\ then}\ t_2\ \mathsf{else}\ t_3 \approx_{\beta\eta} t_3\end{array}$$

$\mathbf{\Lambda}_{\beta\eta\mathbf{Bool}}$ forms an $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra, and there is a clone homomorphism $\eta : \mathbf{Bool} \to \mathbf{\Lambda}_{\beta\eta}$ making it into the free $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra on $\mathbf{Bool}$. Hence we can apply the method above with $F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{Bool} = \mathbf{\Lambda}_{\beta\eta\mathbf{Bool}}$. The algebra homomorphism $g^\dagger : \mathbf{\Lambda}_{\beta\eta\mathbf{Bool}} \to \mathcal{M}_{\mathbb{B}}$ sends each term $\Gamma \vdash t : A$ to its interpretation as a function $\prod_i \mathcal{M}_{\mathbb{B}}[\![\Gamma_i]\!] \to \mathcal{M}_{\mathbb{B}}[\![A]\!]$. Adequacy is therefore exactly the property that $g^\dagger_{\diamond;\mathsf{b}}(t) = g^\dagger_{\diamond;\mathsf{b}}(t')$ implies $t = t'$.

6.2 Open terms and normalization

As a second example, we show that every term of the STLC is equal (up to $\beta\eta$-equality) to one in *η-long β-normal form* (we define these normal forms below). The proof mostly follows Fiore [10], except that we reason abstractly using the universal property of free algebras via our induction principle. It makes use of *Kripke logical relations* (with varying arity), which were introduced by Jung and Tiuryn [27] to study λ-definability.

We first show that our induction principle enables arguments using Kripke logical relations over the STLC. Fix a $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra $(\mathbf{Y}, [\![-]\!])$, homomorphism $f : \mathbf{X} \to \mathbf{Y}$ from a clone $\mathbf{X}$, and a subset $P(\Gamma;\mathsf{b}) \subseteq Y(\Gamma;\mathsf{b})$ for each Γ. We extend P from the base type b to all types by

$$P(\Gamma;A{\Rightarrow}B) = \{t{\in}Y(\Gamma;A{\Rightarrow}B) \mid \forall\Delta,\boldsymbol{\rho}{\in}\mathsf{Var}_S(\Delta;\Gamma), a{\in}P(\Delta;A).\ [\![\mathsf{app}]\!]_\Delta(t[\triangleright\boldsymbol{\rho}], a) \in P(\Delta;B)\}$$

This is the standard definition of a Kripke logical relation on function types (other than using all renamings $\boldsymbol{\rho}$ rather than just weakenings, which is inessential). We therefore have a family of subsets $P(\Gamma;A) \subseteq Y(\Gamma;A)$, to which we apply Corollary 28 and obtain the following.

▶ **Lemma 30.** *If the family of subsets P satisfies*

- *for every context Γ we have $\mathsf{var}^{(\Gamma)} \in P(\Gamma;\Gamma)$;*
- *for every variable renaming $\boldsymbol{\rho} \in \mathsf{Var}_S(\Delta;\Gamma)$ and term $t \in P(\Gamma;\mathsf{b})$ we have $t[\triangleright\boldsymbol{\rho}] \in P(\Delta;\mathsf{b})$;*
- *for every term $t \in X(\Gamma;B)$ and substitution $\boldsymbol{\sigma} \in P(\Delta;\Gamma)$ we have $(f_{\Gamma;B}t)[\boldsymbol{\sigma}] \in P(\Delta;B)$,*

then, for every free term $t \in (F_{\mathbf{\Sigma}^{\Lambda_{\beta\eta}}}\mathbf{X})(\Gamma;A)$, we have $f^\dagger_{\Gamma;A}(t) \in P(\Gamma;A)$.

Proof. The only non-trivial assumption of Corollary 28 is closure under operators. For closure under app, if $t \in P^\flat(\Gamma;A \Rightarrow B)$ and $u \in P^\flat(\Gamma;A)$, then, for all $\boldsymbol{\sigma} \in P(\Delta;\Gamma)$, we have $([\![\mathsf{app}]\!]_\Gamma(t,u))[\boldsymbol{\sigma}] = [\![\mathsf{app}]\!]_\Delta(t[\boldsymbol{\sigma}], u[\boldsymbol{\sigma}])$, because interpretations of operators commute with substitution; this is an element of $P(\Delta;B)$ using $t[\boldsymbol{\sigma}] \in P(\Delta;A \Rightarrow B)$ on the identity variable-renaming. For closure under abs, suppose that $t \in P^\flat(\Gamma,A;B)$. The assumption of the present lemma that P is closed under variable renamings at the base type b extends to all types A by an easy induction on A. For every $\boldsymbol{\sigma} \in P(\Delta;\Gamma)$, $\boldsymbol{\rho} \in \mathsf{Var}_S(\Xi;\Delta)$, and $a \in P(\Xi;A)$, we then have that $t[(\boldsymbol{\sigma} \circ \triangleright\boldsymbol{\rho}), a] \in P(\Xi;B)$. Preservation of substitution by $[\![\mathsf{abs}]\!]$, and the β law, together imply that $[\![\mathsf{app}]\!]_\Xi(([\![\mathsf{abs}]\!]_\Gamma(t))[\boldsymbol{\sigma}][\triangleright\boldsymbol{\rho}]) = t[(\boldsymbol{\sigma} \circ \triangleright\boldsymbol{\rho}), a] \in P(\Xi;B)$. Hence $[\![\mathsf{abs}]\!]_\Gamma(t) \in P^\flat(\Gamma;A \Rightarrow B)$ as required. ◀

We use this to show normalization as follows. *Normal forms* $\Gamma \vdash_n t : A$ are defined mutually inductively with the *neutral forms* $\Gamma \vdash_m t : A$ by the following rules:

$$\frac{}{\Gamma, x : A, \Delta \vdash_m x : A} \qquad \frac{\Gamma \vdash_m f : A \Rightarrow B \quad \Gamma \vdash_n a : A}{\Gamma \vdash_m \mathsf{app}\, f\, a : B} \qquad \frac{\Gamma \vdash_m t : \mathsf{b}}{\Gamma \vdash_n t : \mathsf{b}} \qquad \frac{\Gamma, x : A \vdash_n t : B}{\Gamma \vdash_n \lambda x : A.\, t : A \Rightarrow B}$$

Consider the initial $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$-algebra, which is the clone $\mathbf{\Lambda}_{\beta\eta}$ of STLC terms up to $\approx_{\beta\eta}$. We write $\mathsf{Nf}(\Gamma; A)$ for the subset of STLC terms that are equivalent to a term in normal form under $\approx_{\beta\eta}$; and likewise write $\mathsf{Ne}(\Gamma; A)$ for neutral forms. We consider both as subsets of $\Lambda_{\beta\eta}(\Gamma; A)$; both are closed under variable renaming. The family of subsets we consider, $P(\Gamma; A) \subseteq \Lambda_{\beta\eta}(\Gamma; A)$, is defined on the base type as $P(\Gamma; \mathsf{b}) = \mathsf{Nf}(\Gamma; \mathsf{b})$, and on other types by the logical relations definition above. By a simple induction on the sort A, one can show that $\mathsf{Ne}(\Gamma; A) \subseteq P(\Gamma; A) \subseteq \mathsf{Nf}(\Gamma; A)$ (e.g. as in [10]). Since variables are neutral, this tells us in particular that $\mathsf{var}^{(\Gamma)} \in P(\Gamma; \Gamma)$ for all Γ. It then follows from Lemma 30 that $t = (\triangleright)^\dagger(t) \in P(\Gamma; A) \subseteq \mathsf{Nf}(\Gamma; A)$ for all $t \in \Lambda_{\beta\eta}(\Gamma; A)$, and so that every term of the STLC is $\beta\eta$-equivalent to one in normal form.

7 Comparison to other approaches

While we promote abstract clones as an elementary approach to simple type theories (qua multisorted second-order abstract syntax), there are several equivalent concepts that have been used to similar effect. We give a brief overview of the existing literature on the subject and a comparison with our work; we give references where possible, but unfortunately some of the relationships here exist only in the mathematical folklore.

Presheaves and substitution monoids

The study of second-order abstract syntax was initiated by Fiore et al. [16, 10], who represent term structure using presheaf categories. In their setting, one considers functors $T : \mathbb{L}(S)^{\mathrm{op}} \to \mathbf{Set}^S$, where $\mathbb{L}(S)$ is the category in which objects are contexts Γ, and morphisms $\boldsymbol{\rho} : \Delta \to \Gamma$ are variable renamings $\boldsymbol{\rho} \in \mathsf{Var}_S(\Gamma; \Delta)$ (recall Section 2.1). The S-indexed sets $T(\Gamma)$ consist of the sorted terms in context Γ; while the functions $T(\boldsymbol{\rho})$ rename the variables inside the terms to change their context. Substitution is accounted for by considering the monoidal structure $(\bullet, V)$ on $[\mathbb{L}(S)^{\mathrm{op}}, \mathbf{Set}^S]$, in which $T \bullet T'$ represents (for each context Γ) the simultaneous substitution of each variable in T with a term from T', and V represents the variables in each context. Monoids with respect to this structure are equipped with variables and substitution operations; they are equivalently abstract clones [16, Proposition 3.4]. Fiore and Hur [13] define $\mathbf{\Sigma}$-algebras as monoids in $[\mathbb{L}(S)^{\mathrm{op}}, \mathbf{Set}^S]$ equipped with interpretations of the operators of a presentation $\mathbf{\Sigma}$ satisfying its equations; they are equivalent to our $\mathbf{\Sigma}$-algebras. Our setting is therefore equivalent to that of Fiore et al. The advantage of our approach is that abstract clones require less categorical machinery; for those comfortable with category theory, this will be less of a concern.

There are some technical differences with previous work. Fiore and Hur [13] show the existence of the free $\mathbf{\Sigma}$-algebras on each presheaf T; in light of our free algebra result, the construction of the free algebra on T can be factored into two steps: constructing the free clone $\mathbf{X}$ on T by freely adding variables and substitution, and then taking the free $\mathbf{\Sigma}$-algebra on the clone $\mathbf{X}$. In our examples above, we begin with a clone that admits substitution, and hence do not freely add substitution. In a separate treatment, Hofmann [22] gives an induction principle for the λ-calculus using presheaves, but only considers predicates over closed terms; we obtain induction for closed terms as a corollary of induction over open terms.

Cartesian multicategories

Each abstract clone $\mathbf{X}$ has an identity operation for every sort B, given by the unique variable projection $\mathsf{var}_1^{([B])} \in X(B; B)$, along with admissible operations of exchange, weakening,

and contraction. In this way, the sets of terms $X(\Gamma; A)$ form the structure of a *cartesian multicategory* with object set S (intuitively a category whose morphisms may have multiple inputs, subject to the structural properties of first-order equational logic). Conversely, every cartesian multicategory gives rise to an abstract clone. Thus, one could carry out the development of this paper in the context of cartesian multicategories (cf. [5, Section 9]). Clones are our preferred choice, because the definition of clone (in which projections are the primary operation) provides a more minimal axiomatisation than that of cartesian multicategory (in which the structural operations are primary). Note that one-object cartesian multicategories are usually called *cartesian operads*, which correspond to monosorted abstract clones.

Algebraic theories

The traditional approach to describing first-order algebraic structure in categorical logic is through *algebraic theories* [29]. An algebraic theory is represented by a category with cartesian products, which permit the multimorphisms of a cartesian multicategory to be represented by morphisms from a product: for a context $[A_1, \ldots, A_n]$, the terms $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ are represented by a hom-set $X(A_1 \times \cdots \times A_n, B)$. The relationship between cartesian multicategories and algebraic theories is the notion of *representability* for cartesian multicategories [33]. Second-order structure in the context of algebraic theories is captured by *second-order algebraic theories* [14, 32, 6], which generalize the first-order setting by introducing exponential objects that represent function types. Every second-order presentation $\mathbf{\Sigma}$ induces a second-order algebraic theory, the algebras for which are given by taking coslices over $\mathbf{\Sigma}$ [6].

Monads and relative monads

There is a classical correspondence in category theory between algebraic theories and certain monads on the category of sets [31], which in turn are equivalent to J-relative monads, for J the inclusion of finite sets into sets [4]. This has led to a line of investigation in which monads are used directly for second-order abstract syntax [20, 21, 2, 3, 19]. There are strong connections between this approach and that of presheaves and substitution monoids: for a detailed comparison, see the thesis of Zsidó [38]. In particular, the distinction between abstract clones and J-relative monads is slight, and the results of our development could equivalently be rephrased as statements about relative monads (cf. [6]).

8 Conclusion

We have shown that the abstract syntax of simple type theories has an elementary treatment using abstract clones. The framework we describe allows the specification of the terms and equations of type theories via second-order presentations [13, 14]. Free algebras then give the syntax along with an accompanying induction principle, which we show enables abstract proofs of non-trivial properties such as adequacy. We emphasize that abstract clones axiomatize the syntax only of *simple* type theories: clones cannot express linear types, dependent types, or type theories in which variables stand only for certain classes of term (e.g. polarized type theories [37], and the call-by-value λ-calculus). In some cases, analogous structures are already known (for instance, symmetric multicategories for linear type theories [35, 23]); for others, such as dependent type theories, this remains an open problem.

References

1 Peter Aczel. A general Church–Rosser theorem. Unpublished manuscript, 1978.

2 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26(1):3–37, 2016. doi:10.1017/S0960129514000103.

3 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131, pages 1–16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.6.

4 Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *Foundations of Software Science and Computational Structures*, pages 297–311. Springer, 2010. doi:10.1007/978-3-642-12032-9_21.

5 Nathanael Arkor and Marcelo Fiore. Algebraic models of simple type theories: a polynomial approach. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, page 88–101. Association for Computing Machinery, 2020. doi:10.1145/3373718.3394771.

6 Nathanael Arkor and Dylan McDermott. Higher-order algebraic theories. *Preprint*, 2020. URL: https://www.cl.cam.ac.uk/~na412/Higher-order%20algebraic%20theories.pdf.

7 Michael Barr. Exact categories. In *Exact categories and categories of sheaves*, pages 1–120. Springer, 1971.

8 Garrett Birkhoff. On the structure of abstract algebras. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 31, pages 433–454. Cambridge University Press, 1935. doi:10.1017/S0305004100013463.

9 Garrett Birkhoff and John D Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8(1):115–133, 1970. doi:10.1016/S0021-9800(70)80014-X.

10 Marcelo Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 26–37. Association for Computing Machinery, 2002. doi:10.1145/571157.571161.

11 Marcelo Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.

12 Marcelo Fiore and Makoto Hamana. Multiversal polymorphic algebraic theories: syntax, semantics, translations, and equational logic. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 520–529. IEEE, 2013. doi:10.1109/LICS.2013.59.

13 Marcelo Fiore and Chung-Kil Hur. Second-order equational logic. In *Computer Science Logic*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.

14 Marcelo Fiore and Ola Mahmoud. Second-order algebraic theories. In *Mathematical Foundations of Computer Science*, pages 368–380. Springer, 2010. doi:10.1007/978-3-642-15155-2_33.

15 Marcelo Fiore and Ola Mahmoud. Functorial semantics of second-order algebraic theories, 2014. arXiv:1401.4697.

16 Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 193–202. IEEE, 1999. doi:10.1109/LICS.1999.782615.

17 Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

18 Makoto Hamana. Free σ-monoids: A higher-order syntax with metavariables. In *Asian Symposium on Programming Languages and Systems*, pages 348–363. Springer, 2004. doi:10.1007/978-3-540-30477-7_23.

19 André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. Modules over monads and operational semantics. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167, pages 12–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.12.

20 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In *Logic, Language, Information, and Computation*, pages 218–237. Springer, 2007. doi:10.1007/978-3-540-73445-1_16.

21 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, 2010. doi:10.1016/j.ic.2009.07.003.

22 Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 204–213. IEEE, 1999. doi:10.1109/LICS.1999.782616.

23 Mathieu Huot. Operads with algebraic structure. *MPRI Internship Report*, 2016. URL: http://users.ox.ac.uk/~scro3639/M1_Report.pdf.

24 J.M.E. Hyland. Classical lambda calculus in modern dress. *Mathematical Structures in Computer Science*, 27(5):762–781, 2017. doi:10.1017/S0960129515000377.

25 J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 88, pages 205–232. Cambridge University Press, 1980. doi:10.1017/S0305004100057534.

26 Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, 1999.

27 Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In *Typed Lambda Calculi and Applications*, pages 245–257. Springer, 1993. doi:10.1007/BFb0037110.

28 J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1988.

29 F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(5):869–872, 1963. doi:10.1073/pnas.50.5.869.

30 Daniel J. Lehmann and Michael B. Smyth. Algebraic specification of data types: A synthetic approach. *Mathematical systems theory*, 14(1):97–139, 1981. doi:10.1007/BF01752392.

31 F.E.J. Linton. An outline of functorial semantics. In *Seminar on Triples and Categorical Homology Theory*, pages 7–52. Springer, 1969. doi:10.1007/BFb0083080.

32 Ola Mahmoud. Second-order algebraic theories. Technical Report UCAM-CL-TR-807, University of Cambridge, Computer Laboratory, October 2011. doi:10.48456/tr-807.

33 Claudio Pisani. Sequential multicategories. *Theory and Applications of Categories*, 29(19):496–541, 2014. URL: http://www.tac.mta.ca/tac/volumes/29/19/29-19abs.html.

34 Andrew M. Pitts. Tripos theory in retrospect. *Mathematical structures in computer science*, 12(3):265–279, 2002. doi:10.1017/S096012950200364X.

35 Miki Tanaka. Abstract syntax and variable binding for linear binders. In *Mathematical Foundations of Computer Science*, pages 670–679. Springer, 2000. doi:10.1007/3-540-44612-5_62.

36 Walter Taylor. Abstract clone theory. In *Algebras and orders*, volume 389 of *NATO ASI Series C*, pages 507–530. Springer, 1993. doi:10.1007/978-94-017-0697-1_11.

37 Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-matching*. PhD thesis, Carnegie Mellon University, 2009.

38 Julianna Zsidó. *Typed abstract syntax*. PhD thesis, Université Nice Sophia Antipolis, 2010.

A Normalization with global state

As a further example of the application of abstract clones to problems motivated by simple type theories, we prove a normalization result for the STLC with V-valued global state: concretely, this calculus is given by the free algebra of the second-order presentation $\mathbf{\Sigma}^{\Lambda_{\beta\eta}}$ of the STLC with $\beta\eta$-equality on the clone $\mathbf{GS}_V$ of V-valued global state, whose syntax is

described in Example 23. The proof is similar to normalization of the STLC without global state (Section 6.2); in particular, we reuse Lemma 30.

Recall that for $V = \{v_1, \ldots, v_k\}$, the free algebra consists of the syntax of the STLC extended by the additional term formers get and put_{v_i}. Normal and neutral forms are defined as in Section 6.2, except with

$$\text{the rule } \frac{\Gamma \vdash_m t : \mathsf{b}}{\Gamma \vdash_n t : \mathsf{b}} \text{ replaced by } \frac{\Gamma \vdash_m t_1 : \mathsf{b} \quad \cdots \quad \Gamma \vdash_m t_k : \mathsf{b}}{\Gamma \vdash_n \mathsf{get}(\mathsf{put}_{w_1}(t_1), \ldots, \mathsf{put}_{w_k}(t_k)) : \mathsf{b}} \ (w_1, \ldots, w_k \in V).$$

Again we write $\mathsf{Nf}(\Gamma; A)$ (respectively $\mathsf{Ne}(\Gamma; A)$) for the subsets of terms equal to a normal (respectively neutral) form, and define the logical relation $P(\Gamma; A)$ on the base type as $P(\Gamma; \mathsf{b}) = \mathsf{Nf}(\Gamma; \mathsf{b})$, and on other types by the logical relations definition in Section 6.2. Again we have $\mathsf{Ne}(\Gamma; A) \subseteq P(\Gamma; A) \subseteq \mathsf{Nf}(\Gamma; A)$ by induction on A; the only difference with the previous proof is that on base types one has $\mathsf{Ne}(\Gamma; \mathsf{b}) \subseteq \mathsf{Nf}(\Gamma; \mathsf{b})$, because for $t \in \mathsf{Ne}(\Gamma; \mathsf{b})$ we have $t \approx_{\beta\eta} \mathsf{get}(\mathsf{put}_{v_1}(t), \ldots, \mathsf{put}_{v_k}(t)) \in \mathsf{Nf}(\Gamma; \mathsf{b})$. To prove that every term is equal to one in normal form up to $\approx_{\beta\eta}$, it suffices to apply Lemma 30 with f the clone homomorphism $\eta_{\mathbf{GS}_V} : \mathbf{GS}_V \to F_{\mathbf{\Sigma}^{\Lambda}{}_{\beta\eta}} \mathbf{GS}_V$. The first two assumptions of the lemma have the same proof as before. For the third, since $\mathbf{GS}_V$ is presented by $\mathbf{\Sigma}_V^{\mathbf{GS}}$ and clone homomorphisms preserve variables and substitution, it suffices to show that

- for each $t_1, \ldots, t_k \in P(\Gamma; \mathsf{b})$, we have $\mathsf{get}(t_1, \ldots, t_k) \in P(\Gamma; \mathsf{b})$;
- for each $t \in P(\Gamma; \mathsf{b})$ and $i \leq k$, we have $\mathsf{put}_{v_i}(t) \in P(\Gamma; \mathsf{b})$.

The first statement holds because if $t_i = \mathsf{get}(\mathsf{put}_{w_{i1}}(t'_{i1}), \ldots, \mathsf{put}_{w_{ik}}(t'_{ik}))$ then

$$\mathsf{get}(t_1, \ldots, t_k) \approx_{\beta\eta} \mathsf{get}(\mathsf{put}_{w_{11}}(t'_{11}), \ldots, \mathsf{put}_{w_{kk}}(t'_{kk})) \in \mathsf{Nf}(\Gamma; \mathsf{b}) = P(\Gamma; \mathsf{b})$$

The second statement holds because if $t = \mathsf{get}(\mathsf{put}_{w_1}(t'_1), \ldots, \mathsf{put}_{w_k}(t'_k))$ then

$$\mathsf{put}_{v_i}(t) \ \approx_{\beta\eta} \ \mathsf{put}_{w_i}(t'_i) \ \approx_{\beta\eta} \ \mathsf{get}(\mathsf{put}_{w_i}(t'_i), \ldots, \mathsf{put}_{w_i}(t'_i)) \in \mathsf{Nf}(\Gamma; \mathsf{b}) = P(\Gamma; \mathsf{b})$$