From Binding Signature to Bidirectional Type Checking

LIANG-TING CHEN and HSIANG-SHANG KO, Academia Sinica, Taiwan

Additional Key Words and Phrases: abstract syntax, binding signature, F-algebra, bidirectional type checking

LIST OF TODOS

Define a binding signature as a sum of products for untyped language with weakening . . 6
Derive a strong functor along U from a binding signature 6
An example of binding signature 6
Show that the strong functor associated with a signature gives rise to an initial algebra with substitution 6
Define morphism between signatures 7
Show that each signature morphism defines a morphism between strong functors 7
Representations for initial HSSs 7
Translation from named variable to locally nameless representations 7
Translation from locally nameless to finitely-scoped representations 7
Scope checking by initiality 7
Organise this section clearly 7
Define bidirectional type inference by initiality 7

1 INTRODUCTION

Context cannot be finite if we consider the transformation from an abstract syntax where the type of identifiers is not finite.

1.1 Related Work

Formal theory of abstract syntax with binding. [Ahrens and Matthes 2018; Ahrens et al. 2018; Altenkirch and Reus 1999; Fiore 2008; Fiore et al. 1999; Matthes and Uustalu 2004; Tanaka and Power 2006]

Formalisation and implementation. [Allais et al. 2018, 2021][Ahrens et al. 2022][Fiore and Szamozvancev 2022]

Bidirectional typing. [Dunfield and Krishnaswami 2022]

1.2 Synopsis and Contributions

2 MATHEMATICAL FOUNDATIONS OF SUBSTITUTION

We assume the reader is familiar with basic category theory such as category, functor, natural transformation, limit, monad, and monoidal category in the classic textbook by Mac Lane [1978].

Authors' address: Liang-Ting Chen, liang.ting.chen.tw@gmail.com; Hsiang-Shang Ko, joshko@iis.sinica.edu.tw, Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Nankang, Taipei, Taiwan, 115201.

2023. 2475-1421/2023/1-ART1 $15.00
https://doi.org/

Terminology. A *monad* in this paper refers to a *functor* $T\colon C \to C$ with two natural transformations $\eta\colon \mathsf{Id} \to T$ and $\mu\colon T \cdot T \to T$ satisfying monad laws. An equivalent notion is *Kleisli triple* which refers to a *function* $T\colon \mathrm{Obj}(C) \to \mathrm{Obj}(C)$, a morphism $\eta_X\colon X \to TX$ for $X \in \mathrm{Obj}(X)$ and $\{f\} : TX \to TY$ for $f\colon X \to Y$, satisfying monad laws. The natural transformations η, μ, and the mapping $(-)^\star$ are called `return`, `join`, and `bind` (with its two arguments flipped), for T being strong, respectively in functional programming.

2.1 Prelude: F-Algebra and Initial Semantics

In this section, we start with a recap of initial semantics using F-algebras, also fixing our notations. Examples are carefully walked through to motivate our definitions and later developments.

Definition 2.1. An algebra (A, α) for an endofunctor $F\colon C \to C$ is an object A of C and a morphism $\alpha\colon FA \to A$ where A is called the *carrier* and α the *structure map* of the algebra, and F the *signature* functor. An F-homomorphism from (A, α) to (B, β) is a morphism $f\colon A \to B$ satisfying $f \circ \alpha = \beta \circ Ff$. The category of F-algebras and F-homomorphisms is denoted by $\mathrm{Alg}(F)$.

One simplistic example for signature functor is $(\mathbf{1} + -)\colon \mathrm{Set} \to \mathrm{Set}$ which maps any set X to the coproduct the disjoint union $\mathbf{1} + X$ of the singleton $\mathbf{1}$ and X and a function $f\colon X \to Y$ to $\mathsf{id}_{\mathbf{1}} + f$ defined as the mediating function $[\mathsf{inl}_{\mathbf{1},X} \circ \mathsf{id}_{\mathbf{1}}, \mathsf{inr}_{\mathbf{1},X} \circ f]\colon \mathbf{1} + X \to \mathbf{1} + Y$ where $\mathsf{inl}_{1,X}$ (and $\mathsf{inr}_{1,X}$) is the left (and right) injection to the coproduct. For brevity, we follow the convention that $\mathsf{id}_{\mathbf{1}} + f$ is denoted $\mathbf{1} + f$. The functoriality of $\mathbf{1} + (-)$ can be verified directly, or derived *compositionally*, following from that (co-)limits of functors from $\mathcal{D}$ to C are calculated pointwise in C provided the pointwise (co-)limits exist. That is, $\mathbf{1} + (-)$ is the coproduct of the constant functor $\underline{\mathbf{1}_{\mathrm{Set}}}$ and the identity functor $\mathsf{Id}_{\mathrm{Set}}$. Since $\mathbf{1}$ is a singleton, α_1 is viewed interchangeably as its value $\alpha_1(\star)$.

The structural map of a $(\mathbf{1} + -)$-algebra (A, α) is equal to the mediating morphism $[\alpha_1, \alpha_2]$ of $\alpha_1 = \alpha \circ \mathsf{inl}_{\mathbf{1},A}$ and $\alpha_2 = \alpha \circ \mathsf{inr}_{\mathbf{1},A}\colon A \to A$. Hence, a $(\mathbf{1} + -)$-algebra can be regarded equivalently as a triple (A, α_1, α_2). For example, the set $\mathbb{N}$ of natural numbers with the constant 0 and the successor function $s\colon \mathbb{N} \to \mathbb{N}$ forms an F-algebra $(\mathbb{N}, [\underline{0}, s])$. Moreover, this algebra $(\mathbb{N}, [\underline{0}, s])$ is initial: For every $(\mathbf{1} + -)$-algebra (A, α), there is a unique homomorphism $f\colon (\mathbb{N}, [\underline{0}, s]) \to (A, \alpha)$. Equivalently, every triple $(A, \underline{z}, \alpha_2)$ associates with a unique function $f\colon \mathbb{N} \to A$ such that $f(0) = z$ and $f(s(n)) = \alpha_2(f(n))$, modelling the structural recursion over the inductive type $\mathbb{N}$. In terms of functional programming, this is exactly $\mathtt{fold}_{\mathbb{N}}\colon A \to (A \to A) \to \mathbb{N} \to A$ for natural numbers.

The next is one of our leading example: Bird and Paterson [1999]'s formulation of λ-terms as a nested type.

Example 2.2. Define λ-terms up to α-equivalence over an *arbitrary* set X of variables by

$$\frac{x \in X}{\text{`}x \in \Lambda X}\ (\mathrm{var}) \qquad \frac{t \in \Lambda X \qquad u \in \Lambda X}{t \cdot u \in \Lambda X}\ (\mathrm{app}) \qquad \frac{t \in \Lambda(\mathbf{1} + X)}{\lambda t \in \Lambda X}\ (\mathrm{abs})$$

also a function named after each rule—$\mathrm{var}_X(x) = \text{`}x$, $\mathrm{app}_X(t, u) = t \cdot u$, and $\mathrm{abs}_X(t) = \lambda t$. This formulation is a variant of *de Bruijn representation* in the sense each bound variable in $t \in \Lambda X$ is of the form $(\mathsf{inr}^n \circ \mathsf{inl})(\star) \in \mathbf{1} + (\cdots + (\mathbf{1} + X))$ for some natural number n; free variables are $\mathsf{inr}^{n+1}(x)$ for some x in X. For brevity, we denote $\mathsf{inl}_{1,X}(\star)$ and $\mathsf{inr}_{1,X}\colon X \to \mathbf{1} + X$ by z_X and s_X respectively, e.g., $\lambda\,\lambda\text{`}s(z)$ is the first projection (where the subscripts are omitted). This formulation resembles the *locally nameless representation* by Charguéraud [2012] where bound variables are represented by de Bruijn indices but free variables, introduced by a separate rule, are from another set.

Simultaneous renaming $\Lambda\rho\colon \Lambda X \to \Lambda Y$ for a renaming rule $\rho\colon X \to Y$ defined by

$$\Lambda\rho(\text{`}x) = \text{`}\rho(x) \qquad \Lambda\rho(t \cdot u) = \Lambda\rho(t) \cdot \Lambda\rho(u) \qquad \Lambda\rho(\lambda\, t) = \lambda\,(\Lambda(\mathbf{1} + \rho)(t))$$

replaces variables $x \in X$ by $\rho(x)$ but leaves bound variables intact, illustrating three diagrams

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{var}_X} & \Lambda X \\ {\scriptstyle\rho}\downarrow & & \downarrow{\scriptstyle\Lambda(\rho)} \\ Y & \xrightarrow[\mathrm{var}_Y]{} & \Lambda Y \end{array} \qquad \begin{array}{ccc} \Lambda X \times \Lambda X & \xrightarrow{\mathrm{app}_X} & \Lambda X \\ {\scriptstyle\Lambda\rho\times\Lambda\rho}\downarrow & & \downarrow{\scriptstyle\Lambda\rho} \\ \Lambda Y \times \Lambda Y & \xrightarrow[\mathrm{app}_Y]{} & \Lambda Y \end{array} \qquad \begin{array}{ccc} \Lambda(\mathbf{1}+X) & \xrightarrow{\mathrm{abs}_X} & \Lambda X \\ {\scriptstyle\Lambda(\mathbf{1}+\rho)}\downarrow & & \downarrow{\scriptstyle\Lambda\rho} \\ \Lambda(\mathbf{1}+Y) & \xrightarrow[\mathrm{abs}_Y]{} & \Lambda Y \end{array}$$

where $\Lambda(\mathbf{1}+X)$ is the composed functor $\Lambda \cdot (\mathbf{1}+-)$ applied to X. It is straightforward to check that with renaming $\Lambda\colon \mathsf{Set} \to \mathsf{Set}$ forms a functor, i.e.

$$\Lambda\rho(\mathsf{id}_X) = \mathsf{id}_{\Lambda X} \quad \text{and} \quad \Lambda(\rho_2 \circ \rho_1) = \Lambda\rho_2 \circ \Lambda\rho_1$$

for any $\rho_1\colon X \to Y$ and $\rho_2\colon Y \to Z$. Therefore, each family of functions above is a natural transformation to Λ. Moreover, each construct can be seen as an algebra for a suitable endofunctor: (1) the constant functor $\underline{\mathsf{Id}_{\mathsf{Set}}}$, (2) the product $\mathsf{Id} \times \mathsf{Id}$, and (3) the (context) extension δ defined by $\delta T \coloneqq T \cdot (\mathbf{1}+-)$ and $\delta\tau \coloneqq \tau \cdot (\mathbf{1}+-)$ for $\tau\colon T \to T'$, respectively. In short, the three constructs of λ-terms exhibit an algebra structure on Λ:

$$[\mathsf{var}, \mathsf{app}, \mathsf{abs}]\colon \Lambda + \mathsf{LC}(\Lambda) \to \Lambda$$

for the endofunctor $\underline{\mathsf{Id}} + \mathsf{LC}$ of $\mathsf{Endo}(\mathsf{Set})$ where $\mathsf{LC} \coloneqq \mathsf{Id} \times \mathsf{Id} + \delta$. This algebra is even the *initial* algebra, as we will show later .

Remark. We do not require the context set X being finite or, in particular, a list of variables to make sense of context operations, contrast to the frameworks proposed by Allais et al. [2021]; Fiore et al. [1999]. For example, the *weakening* operation is (still) defined solely via functorality and the injection $s\colon X \to \mathbf{1}+X$

$$\Lambda s\colon \Lambda X \to \Lambda(\mathbf{1}+X) \tag{1}$$

where each variable x in $t \in \Lambda X$ is 'shifted' to $s(x)$, resulting a term $\Lambda s(t)$ in the context $\mathbf{1}+X$.

Formulating λ-terms as an algebra would be useless if this was the only algebra for $(\underline{\mathsf{Id}} + \mathsf{LC})$. Indeed, there are other examples and such an algebra needs not be syntactic at all.

Example 2.3. The set $fv_X(t)$ of free variables in $t \in \Lambda X$ is defined for arbitrary X by

$$fv_X(\text{`}x) = \{x\} \qquad fv_X(t \cdot u) = fv_X(t) \cup fv_X(u) \qquad fv_X(\lambda\, t) = fv_{\mathbf{1}+X}(t) \cap X$$

Put differently, the definition of fv_X can be illustrated by three commutative diagrams

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{var}_X} & \Lambda X \\ {\scriptstyle\mathsf{id}}\downarrow & & \downarrow{\scriptstyle fv_X} \\ X & \xrightarrow[\{-\}_X]{} & \mathcal{P}X \end{array} \qquad \begin{array}{ccc} \Lambda X \times \Lambda X & \xrightarrow{\mathrm{app}_X} & \Lambda X \\ {\scriptstyle fv_X\times fv_X}\downarrow & & \downarrow{\scriptstyle fv_X} \\ \mathcal{P}X \times \mathcal{P}X & \xrightarrow[-\cup_X -]{} & \mathcal{P}X \end{array} \qquad \begin{array}{ccc} \Lambda(\mathbf{1}+X) & \xrightarrow{\mathrm{abs}_X} & \Lambda X \\ {\scriptstyle fv_{\mathbf{1}+X}}\downarrow & & \downarrow{\scriptstyle fv_X} \\ \mathcal{P}(\mathbf{1}+X) & \xrightarrow[-\cap X]{} & \mathcal{P}X \end{array} \tag{2}$$

where $\mathcal{P}$ is the powerset functor and metaphorically $(-) \cap X$ strengthens indices in $S \in \mathcal{P}(\mathbf{1}+X)$ by removing z_X and decrementing $s_X(x)$ to x. We can readily check that functions of the lower legs in (2) are natural in X, forming a $(\underline{\mathsf{Id}} + \mathsf{LC})$-algebra on the functor $\mathcal{P}$. Renaming variables does not make a free variable bound or vice versa, so $\mathcal{P}\rho \circ fv_X = fv_Y \circ \Lambda\rho$ holds for any ρ, showing that $fv_X\colon \Lambda X \to \mathcal{P}X$ is natural in X. Thus, by (2), the natural transformation fv is a homomorphism. Indeed, this is the unique LC-homomorphism from $(\Lambda, [\mathsf{var}, \mathsf{app}, \mathsf{abs}])$ as shown later.

We may wonder if *simultaneous substitution* $\{\sigma\}\colon \Lambda X \to \Lambda Y$ for a substitution rule $\sigma\colon X \to \Lambda Y$ is a homomorphism. However, $\{\sigma\}(x) = \sigma(x)$ is not necessarily a variable, so it is not homomorphic.

2.2 Algebra with Substitution

In this section, we extend algebras by substitution that is compatible with its algebra structure. Simultaneous substitution $\{\sigma\} : \Lambda X \to \Lambda Y$ for a substitution rule $\sigma\colon X \to \Lambda Y$ is defined by

$$\{\sigma\}\,`x = \sigma(x) \qquad \{\sigma\}\,(t \cdot u) = \{\sigma\}\,t \cdot \{\sigma\}\,u \qquad \{\sigma\}\,\lambda\,t = \lambda\left\{[\mathsf{var}_{1+Y} \circ \underline{z_Y}, \Lambda s \circ \sigma]\right\} t$$

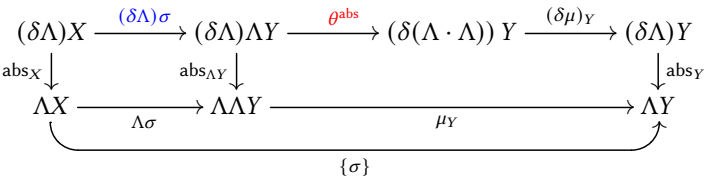
where the substitution rule $[\mathsf{var}_{1+Y} \circ \underline{z_Y}, \Lambda s \circ \sigma]$ in the third rule amounts to two functions

$$\mathsf{var}_{1+Y} \circ \underline{z_Y}\colon \mathbf{1} \to \Lambda(\mathbf{1}+Y) \quad \text{and} \quad \Lambda s \circ \sigma\colon X \to \Lambda(\mathbf{1}+Y)$$

specifying that the variable z bound to λ remains the same and every other variable $x \in X$ is replaced by $\sigma(x)$ but weakened using (1). It is is well-known [Altenkirch and Reus 1999] that $(\Lambda, \mathsf{var}, \{-\})$ forms a Kleisli triple, so by the equivalence between Kleisli triple and monad, substitution $\{\sigma\}$ boils down to—$\mu_Y \circ \Lambda\sigma$ with $\mu_Y = \{\mathsf{id}_{TY}\}$—simultaneous renaming for the rule σ followed by flattening a λ-term over λ-terms. It turns out that $\left\{[\mathsf{var}_{1+Y} \circ \underline{z_Y}, \Lambda s \circ \sigma]\right\}$ decomposes into

$$\begin{aligned}
&\mu_{1+Y} \circ \Lambda[\mathsf{var} \circ \underline{z}, \Lambda s \circ \sigma] \\
&= \mu_{1+Y} \circ \Lambda[\mathsf{var} \circ \underline{z}, \Lambda s] \circ \Lambda(\mathsf{id}_1 + \sigma) && \{\text{ by functorality and the 'fusion law' for coproduct }\} \\
&= (\delta\mu)_Y \circ \theta^{\mathsf{abs}} \circ (\delta\Lambda)\sigma. && \{\text{ by } \theta^{\mathsf{abs}} \coloneqq \Lambda[\mathsf{var}_{1+Y} \circ \underline{z_Y}, \Lambda s] \text{ and by the definition of } \delta\ \}
\end{aligned}$$

(with some subscripts omitted) so that the following diagram illustrates the third rule



By examining the diagram carefully, we find that the only datum required to have substitution compatible with abs is θ^{abs}, motivating Definitions 2.4 and 2.5. In order to define θ^{abs} independent of Λ, we work with pointed endofunctors (Z, e) representing structures with a (semantic) variable rule. A *pointed functor* consists of an endofunctor Z of C and a natural transformation $e\colon \mathsf{Id} \to Z$ and morphisms $f\colon (Z, e) \to (Z', e')$ between them are natural transformations $f\colon Z \to Z'$ satisfying $e' = f \circ e$. We denote the category of pointed functors of C by $\mathsf{Pt}(C)$ and its forgetful functor $(Z, e) \mapsto Z$ by $U\colon \mathsf{Pt}(C) \to \mathsf{Endo}(C)$.

Definition 2.4. A (right) *strength* θ for $H\colon \mathsf{Endo}(C) \to \mathsf{Endo}(C)$ relative to $U\colon \mathsf{Pt}(C) \to \mathsf{Endo}(C)$ is a family θ of natural transformations

$$\theta_{T,(Z,e)}\colon HT \cdot Z \to H(T \cdot Z)$$

natural in $T\colon C \to C$ and the pointed functor (Z, e) such that

(1) $\theta_{T,(\mathsf{Id},\mathsf{id})} = \mathsf{id}_{HT}$ for each T and
(2) $\theta_{T,(Z' \cdot Z, e' \cdot e)} = \theta_{T \cdot Z',(Z,e)} \circ \theta_{T,(Z',e')} \cdot Z$ for any T and pointed functors (Z, e) and (Z', e').

For brevity, we simply call H a *functor with strength*, if the associated strength is clear from context.

There are three naturality conditions, i.e. for T, (Z, e) and objects X in C, imposed on the strength but we shall not worry about checking them. In practice, we will work with a presentation for (H, θ), called binding signatures, for a class of functors with strength so that the condition will be checked once for all.

Definition 2.5. An *H-algebra with substitution* (T, η, α, μ) consists of a monad (T, η, μ) and an H-algebra (T, α) satisfying

$$\begin{array}{ccccc} HT \cdot T & \xrightarrow{\theta_{T,(T,\eta)}} & H(T \cdot T) & \xrightarrow{H\mu} & HT \\ {\scriptstyle \alpha \cdot T}\downarrow & & & & \downarrow{\scriptstyle \alpha} \\ T \cdot T & & \xrightarrow[\mu]{} & & T \end{array} \tag{3}$$

We may write $(T, \eta, \alpha, -^\star)$ for an H-algebra with substitution and call $-^\star$ *(semantic) substitution* where $(T, \eta, -^\star)$ is the Kleisli triple in bijection with (T, η, μ).

In other words, an (H, θ)-algebra with substitution consists of a monad structure and an H-algebra on the same functor T such that the algebra map commutes with the 'semantic' substitution operation $(-)^\star$ in the way determined by the strength θ.

Remark. This notion is a special case of Σ*-monoids* [Fiore 2008] for an endofunctor Σ with a (I/C)-strength θ on a monoidal category $(C, I, \otimes)$ defined as a Σ-algebra $(A, \alpha \colon \Sigma A \to A)$ for A in C equipped with a monoid structure (A, η, μ) satisfying (3) while $\Sigma = H$ and C is the monoidal category of endofunctors with the functor composition $\cdot$ as the monoidal tensor $\otimes$.

Two previous examples are indeed algebras with substitution.

Example 2.6 (Example 2.2, continued). Based on the discussion at the beginning of this section, define a strength θ^{LC} for LC as the coproduct $\theta^\times + \theta^\delta$ of another two strengthens for $\mathsf{Id} \times \mathsf{Id}$ and δ respectively given by

$$\theta^\times_{T,(Z,e)} \colon (T \times T) \cdot Z \to TZ \times TZ \qquad \theta^\delta_{T,(Z,e)} \colon \delta T \cdot Z \to \delta(T \cdot Z)$$
$$\left(\theta^\times_{T,(Z,e)}\right)_Y = \mathsf{id}_{TZY} \times \mathsf{id}_{TZY} \qquad \left(\theta^\delta_{T,(Z,e)}\right)_Y = T[e_{1+Y} \circ \underline{z_Y}, Zs_Y].$$

By instantiating θ^δ with Λ and (Λ, var), we derive θ^{abs} exactly as desired, so by construction $(\Lambda, \mathsf{var}, [\mathsf{app}, \mathsf{abs}], \{-\})$ is an LC-algebra with substitution.

Example 2.7 (Example 2.3, continued). By instantiating θ with $\mathcal{P}$ and $(\mathcal{P}, \subset\ -)$, the strength becomes a family of morphisms $\mathcal{P}[\{-\} \circ \underline{z_X}, \mathcal{P}s] \colon \mathcal{P}(\mathbf{1} + \mathcal{P}X) \to \mathcal{P}^2(\mathbf{1} + X)$ natural in X where

$$[\{-\} \circ \underline{z_X}, \mathcal{P}s] \colon \mathbf{1} + \mathcal{P}X \to \mathcal{P}(\mathbf{1} + X)$$

maps z_X to the singleton $\{z\}$ and every subset $U \subseteq X$ to $s[U] \subseteq \mathbf{1} + X$, i.e. every element $x \in U$ is weakened. Then, it is routine to check the condition (3), so we omit it. It follows that $(\mathcal{P}, \{-\}, [-\cup_X -, -\cap X]_X, \bigcup)$ is indeed an LC-algebra with substitution.

2.3 Morphism of Algebra with Substitution

We observe that the homomorphism fv satisfies a nice property called *semantic substitution lemma*:

Lemma 2.8. *Substitution for any rule $\sigma \colon X \to \Lambda Y$ commutes with fv in the sense that the diagram*

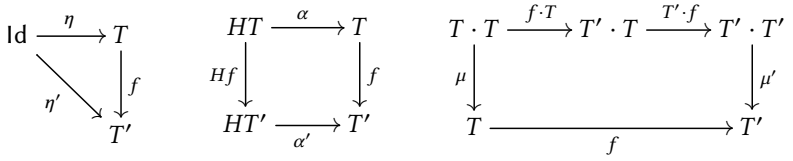
$$\begin{array}{ccc} \Lambda X & \xrightarrow{\{\sigma\}} & \Lambda Y \\ {\scriptstyle fv_X}\downarrow & & \downarrow{\scriptstyle fv_Y} \\ \mathcal{P}X & \xrightarrow[(fv_Y \circ \sigma)^\star]{} & \mathcal{P}Y \end{array} \quad \textit{or, equivalently} \quad \begin{array}{ccccc} \Lambda X & \xrightarrow{\Lambda\sigma} & \Lambda\Lambda Y & \xrightarrow{\mu_Y} & \Lambda Y \\ {\scriptstyle fv_X}\downarrow & & \downarrow{\scriptstyle fv_{\Lambda Y}} & & \downarrow{\scriptstyle fv_Y} \\ \mathcal{P}X & \xrightarrow[\mathcal{P}\sigma]{} & \mathcal{P}\Lambda Y & \xrightarrow[\mathcal{P}(fv_Y)]{} & \mathcal{P}\mathcal{P}Y \xrightarrow[\cup]{} \mathcal{P}Y \end{array} \tag{4}$$

commutes where $(-)^\star$ is the semantic substitution for the powerset monad $(\mathcal{P}, \{-\}, \bigcup)$.

From a computational viewpoint, calculating the set $fv_Y(\{\sigma\}\, t)$ of free variables can be performed priori to substitution without building the intermediate term $\{\sigma\}\, t$. Moreover, we only need to traverse the term t once while collecting free variables x_i's and compute the free variables of $\sigma(x_i)$ once even if x_i may occur more than once in t, saving us not only space but also time.

This property holds even for every $(\underline{\mathsf{Id}} + \mathsf{LC})$-morphism from the $(\Lambda, [\mathsf{var}, \mathsf{app}, \mathsf{abs}])$ if the target is an LC-algebra with substitution. Accordingly we revise the notion of morphisms for algebras with substitution. Note that for (4) to commute, it suffice to consider the right rectangle of the diagram on the right. Therefore the condition boils down to the following definition.

Definition 2.9. An *H-homomorphism* f between algebras with substitution from (T, η, α, μ) to $(T', \eta', \alpha', \mu')$ is a natural transformation $f : T \to T'$ such that f is an H-algebra homomorphism from (T, α) to (T', α') and f is a monad morphism from (T, η, μ) to (T', η', μ'), i.e. satisfying



Let $\mathsf{Alg}^\star(H)$ denote the category formed by H-algebras with substitution and their homomorphisms.

That is, an H-homomorphism between algebras with substitution commutes not only with H-algebra structure but also with the monad structure. In Section 3.1 we will show that the functor with strength associated with a binding signature, including LC, has an initial algebra with substitution and $(\mathsf{LC}, \mathsf{var}, [\mathsf{app}, \mathsf{abs}], \{-\})$ is indeed the initial algebra. Lemma 2.8 follows from initiality immediately once substitution is proved compatible with the LC-algebra map.

2.4 Initial H-algebra with substitution

3 FROM SIGNATURES TO REPRESENTATIONS AND SCOPE CHECKING

3.1 Binding Signature

LT: Define a binding signature as a sum of products for untyped language with weakening

LT: Derive a strong functor along U from a binding signature

Example 3.1.

LT: An example of binding signature

Theorem 3.2.

LT: Show that the strong functor associated with a signature gives rise to an initial algebra with substitution

3.2 Morphism between Signatures

For two (mere) endofunctors F and F' on the same category C, a natural transformation τ may be rightfully called a morphism between signature functors, as it gives rise to a functor turning every F'-algebra (A, α) to an F-algebra $(A, \alpha \circ \tau_A)$. If we consider endofunctors on *different* categories, say C and C', connected by a functor $E : C' \to C$ then a suitable notion of morphism from $F : C \to C$ to $F' : C' \to C'$, defined as a natural transformation τ from $F \cdot E$ to $E \cdot F'$, also gives rise to a functor from $\mathsf{Alg}(F')$ to $\mathsf{Alg}(F)$.

Definition 3.3 (Strong J-relative functor). Let (J, e^J, m^J) be a monoidal functor from $\mathsf{Endo}(C')$ to $\mathsf{Endo}(C)$ where $e^J : \mathsf{Id}_C \to J\mathsf{Id}_{C'}$ and $m_{F,G} : JF \cdot JG \to J(F \cdot G)$.

Definition 3.4.

LT: Define morphism between signatures

Example 3.5 (Type erasure).

LT: Show that each signature morphism defines a morphism between strong functors

3.3 Representing Substitution Systems

LT: Representations for initial HSSs: 1. Named variable representation; 2. Locally nameless representation; 3. Finitely-scoped representation

LT: Translation from named variable to locally nameless representations

LT: Translation from locally nameless to finitely-scoped representations

3.4 Scope Checking by Initiality

LT: Scope checking by initiality

4 FROM BIDIRECTIONAL TYPING SIGNATURES TO TYPE INFERENCE

LT: Organise this section clearly

$H\colon \mathsf{Endo}(\mathsf{Set}^{2\times\mathcal{S}}) \to \mathsf{Endo}(\mathsf{Set}^{2\times\mathcal{S}})$ where $\mathbf{2}$ is the free category over the graph $0_2, 1_2$ with edges $0_2 \to 1_2$ and $1_2 \to 0_2$. There is a natural *embedding* $J\colon \mathsf{Set}^{\mathcal{S}} \to \mathsf{Set}^{2\times\mathcal{S}}$ defined by $(J\Gamma)(1_2, s) = \Gamma(s)$ and $(J\Gamma)(0_2, s) = \mathbf{0}$.

4.1 Signature for Bidirectional Typing

Mode-Correctness and Annotatability

4.2 Signature Functor for Bidirectional Typing

Example 4.1 (Direction erasure).

4.3 Type Inference by Initiality

LT: Define bidirectional type inference by initiality

5 MORE EXAMPLES

6 DISCUSSION AND FUTURE WORK

Relative monad. For the sake of clarity, we only consider ordinary monads as the carrier of a heterogeneous substitution system, but however it is nature to use a category of contexts with a different chosen context extension than the one stated in this paper. The view motivates the use of relative monad proposed by Altenkirch et al. [2015].

Comparing frameworks for scoped- and typed-safe syntaxes.

ACKNOWLEDGMENTS

The work is supported by the Ministry of Science and Technology of Taiwan under grant MOST 109-2222-E-001-002-MY3.

REFERENCES

Benedikt Ahrens and Ralph Matthes. 2018. Heterogeneous Substitution Systems Revisited. In *21st International Conference on Types for Proofs and Programs (TYPES 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 69)*, Tarmo Uustalu (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2:1—-2:23. https://doi.org/10.4230/LIPIcs.TYPES.2015.2 arXiv:1601.04299 1

Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. 2018. From Signatures to Monads in UniMath. *Journal of Automated Reasoning* (jul 2018), 1–34. https://doi.org/10.1007/s10817-018-9474-4 1

Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. 2022. Implementing a category-theoretic framework for typed abstract syntax. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, New York, NY, USA, 307–323. https://doi.org/10.1145/3497775.3503678 arXiv:2112.06984 1

Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2018. A type and scope safe universe of syntaxes with binding: their semantics and proofs. *Proceedings of the ACM on Programming Languages* 2, ICFP (jul 2018), 1–30. https://doi.org/10.1145/3236785 1

Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2021. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *Journal of Functional Programming* 31, 1996 (oct 2021), e22. https://doi.org/10.1017/S0956796820000076 arXiv:2001.11001 1, 3

Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. 2015. Monads need not be endofunctors. *Logical Methods in Computer Science* 11, 1 (mar 2015), 1–40. https://doi.org/10.2168/LMCS-11(1:3)2015 7

Thorsten Altenkirch and Bernhard Reus. 1999. Monadic Presentations of Lambda Terms Using Generalized Inductive Types. In *Computer Science Logic*, Jörg Flum and Mario Rodriguez-Artalejo (Eds.). Lecture Notes in Computer Science, Vol. 1683. Springer Berlin Heidelberg, Berlin, Heidelberg, 453–468. https://doi.org/10.1007/3-540-48168-0_32 1, 4

Richard Bird and Ross Paterson. 1999. de Bruijn notation as a nested datatype. *Journal of Functional Programming* 9, 1 (jan 1999), 77–91. https://doi.org/10.1017/S0956796899003366 2

Arthur Charguéraud. 2012. The locally nameless representation. *Journal of Automated Reasoning* 49, 3 (2012), 363–408. https://doi.org/10.1007/s10817-011-9225-2 2

Jana Dunfield and Neel Krishnaswami. 2022. Bidirectional Typing. *Comput. Surveys* 54, 5 (jun 2022), 1–38. https://doi.org/10.1145/3450952 arXiv:1908.05839 1

Marcelo Fiore. 2008. Second-Order and Dependently-Sorted Abstract Syntax. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*. IEEE, 57–68. https://doi.org/10.1109/LICS.2008.38 1, 5

Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal metatheory of second-order abstract syntax. *Proceedings of the ACM on Programming Languages* 6, POPL (jan 2022), 1–29. https://doi.org/10.1145/3498715 1

Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science*. IEEE Comput. Soc, Trento, 193–202. https://doi.org/10.1109/LICS.1999.782615 1, 3

Saunders Mac Lane. 1978. *Categories for the Working Mathematician*. Graduate Texts in Mathematics, Vol. 5. Springer New York, NY. https://doi.org/10.1007/978-1-4757-4721-8 1

Ralph Matthes and Tarmo Uustalu. 2004. Substitution in non-wellfounded syntax with variable binding. *Theoretical Computer Science* 327, 1-2 (oct 2004), 155–174. https://doi.org/10.1016/j.tcs.2004.07.025 1

Miki Tanaka and A. John Power. 2006. Pseudo-distributive laws and axiomatics for variable binding. *Higher-Order and Symbolic Computation* 19, 2-3 (sep 2006), 305–337. https://doi.org/10.1007/s10990-006-8750-x 1