Polymorphic Abstract Syntax via Grothendieck Construction

Makoto Hamana

Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp

Abstract. Abstract syntax with variable binding is known to be characterised as an initial algebra in a presheaf category. This paper extends it to the case of polymorphic typed abstract syntax with binding. We consider two variations, second-order and higher-order polymorphic syntax. The central idea is to apply Fiore's initial algebra characterisation of typed abstract syntax with binding repeatedly, i.e. first to the type structure and secondly to the term structure of polymorphic system. In this process, we use the Grothendieck construction to combine differently staged categories of polymorphic contexts.

1 Introduction

It is well-known that first-order abstract syntax is modelled as an initial algebra [GTW76] in the framework of ordinary universal algebra. Because this algebraic characterisation cleanly captures various important aspects of syntax, such as structural recursion and induction principles, in terms of algebraic notions, it has been extended to more enriched abstract syntax: abstract syntax with *variable binding* [Hof99, FPT99], *simply-typed* abstract syntax with variable binding [Fio02, MS03, TP08], and *dependently-sorted* abstract syntax [Fio08]. These are uniformly modelled in the framework of categorical universal algebra in presheaf categories.

The solid algebraic basis of enriched abstract syntax has produced fruitful applications. The untyped case [FPT99] was applied to characterisations of second-order abstract syntax with metavariables [Ham04, Fio08], higher-order rewriting [Ham05], explicit substitutions [GUH06], and the Fusion calculus [Mic08]. The simply-typed case [Fio02, MS03] was applied to normalisation by evaluation [Fio02], pre-logical predicates [Kat04], simply-typed higher-order rewriting [Ham07], cyclic sharing tree structures [Ham10], and second-order equational logic [FH10].

However, an important extension of abstract syntax remains untouched, namely *polymorphic typed abstract syntax*.

This paper provides the initial algebra characterisation of polymorphic typed abstract syntax with variable binding in a presheaf category. We consider two variations, second-order and higher-order polymorphic syntax. The central idea is to repeatedly apply Fiore's initial algebra characterisation of typed abstract syntax with binding [Fio02] *twice*, i.e. first to the type structure and secondly to the term structure of polymorphic system. In this process, we use the Grothendieck construction to combine differently staged categories of polymorphic contexts, which is a key to defining the category of discourse in our formulation.

M. Hofmann (Ed.): FOSSACS 2011, LNCS 6604, pp. 381–395, 2011.
© Springer-Verlag Berlin Heidelberg 2011

This characterisation will be a basis of further fruitful research. It is applicable to modern functional programming language such as ML and Haskell. Moreover, it can be a basis of more interesting systems, polymorphic equational logic (along the line of Fiore's programme on synthesis of equational logic [Fio09]), polymorphic higher-order rewriting systems as an extension of untyped [Ham05] and simply-typed [Ham07] higher-order rewriting systems based on algebraic semantics.

Organisation. This paper is organised as follows. We first review the previous algebraic models of abstract syntax with binding in Section 2. We then characterise polymorphic syntax by examining the syntax of system F in Section 3. We further characterise higher-order polymorphic syntax by examining the syntax of system F_ω in Section 4. Finally, in Section 5, we discuss how substitutions on polymorphic syntax can be modelled.

2 Background

2.1 Algebras in $\mathbf{Set}^{\mathbb{F}}$ for Abstract Syntax with Binding

Firstly, we review algebras in a presheaf category $\mathbf{Set}^{\mathbb{F}}$ for modelling untyped abstract syntax with binding by Fiore, Plotkin and Turi [FPT99]. Hofmann [Hof99] also used the same approach to model higher-order abstract syntax. This is the basis of typed abstract syntax in next subsection and polymorphic syntax in §3.

The aim is to model syntax involving variable binding. A typical example is the syntax for untyped λ-terms:

$$\frac{}{x_1,\ldots,x_n \vdash x_i} \qquad \frac{x_1,\ldots,x_n \vdash t \quad x_1,\ldots,x_n \vdash s}{x_1,\ldots,x_n \vdash t@s} \qquad \frac{x_1,\ldots,x_n,x_{n+1} \vdash t}{x_1,\ldots,x_n \vdash \lambda(x_{n+1}.t)}$$

This is seen as abstract syntax generated by three constructors, i.e. the variable former, the application @, and the abstraction λ. The point is that the variable former is a unary and @ is a binary function symbol, but λ is not merely a unary function symbol. It also makes the variable x_{n+1} bound and decreases the context, which is seen as taking the "internal-level abstraction" $(x_{n+1}.t)$ as the argument of the constructor λ.

In order to model this phenomenon of variable binding generally (not only for λ-terms), Fiore et al. took the presheaf category $\mathbf{Set}^{\mathbb{F}}$ to be the universe of discourse, where $\mathbb{F}$ is the category which has finite cardinals $n = \{1,\ldots,n\}$ (n is possibly 0) as objects, and all functions between them as arrows $m \to n$. This is the category of object variables by the method of de Bruijn index/levels (i.e. natural numbers) and their renamings.

Fiore et al. showed that abstract syntax with variable binding is precisely characterised as the initial algebra of suitable endofunctor modelling a signature (e.g. for λ-terms). More precisely, we need the functor $\delta : \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ for context extension $(\delta A)(n) = A(n+1)$ for $A \in \mathbf{Set}^{\mathbb{F}}, n \in \mathbb{F}$, and the presheaf $\mathrm{V} \in \mathbf{Set}^{\mathbb{F}}$ of variables defined by $\mathrm{V}(n) = \mathbb{F}(1,n) \cong \{1,\ldots,n\}$. Using these, for example, we can define the endofunctor Σ_λ on $\mathbf{Set}^{\mathbb{F}}$ for abstract syntax of λ-terms by

$$\Sigma_\lambda(A) = \mathrm{V} + A \times A + \delta A$$

where each summand corresponds to the arity of variable former, @ and λ symbols. Then we use ordinary notion of functor-algebras in $\mathbf{Set}^{\mathbb{F}}$. Generally, an endofunctor Σ on $\mathbf{Set}^{\mathbb{F}}$ defined using $+, \times, \delta$ is called *signature functor*, and a Σ*-algebra* is a pair (A, α) consisting of a presheaf A and a map $\alpha : \Sigma A \to A$, called an *algebra structure*. A *homomorphism* of Σ-algebras is a map $\phi : (A, \alpha) \to (B, \beta)$ such that $\phi \circ \alpha = \beta \circ \Sigma\phi$.

The initial Σ_λ-algebra (Λ, in) exists and can be constructed by the method in [SP82]. It can be expressed as the presheaf $\Lambda(n) = \{t \mid x_1, \ldots, x_n \vdash t\}/=_\alpha$ of all terms, where the algebra structure $\mathsf{in} : \Sigma_\lambda\Lambda \to \Lambda$ consists of constructors of λ-terms, i.e. the variable former, @, and λ.

This process is generic with respect to arbitrary signature functor Σ, hence an initial Σ-algebra in $\mathbf{Set}^{\mathbb{F}}$ models abstract syntax with variable binding.

2.2 Algebras in $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ for Typed Abstract Syntax with Binding

Algebras in $\mathbf{Set}^{\mathbb{F}}$ lack the treatment of type restrictions on syntax. A typical example of typed abstract syntax with binding is the syntax for *simply-typed λ-terms*:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash t : \sigma \Rightarrow \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash t@s : \tau} \qquad \frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda(x\colon \sigma.t) : \sigma \Rightarrow \tau}$$

To model this syntax, we need to model typed contexts $\Gamma = \{x_1 : \tau_1, \cdots, n : \tau_n\}$.

For this, instead of the category $\mathbb{F}$, Fiore [Fio02], and Miculan and Scagnetto [MS03] took the comma category[1] $\mathbb{F}\downarrow U$ for the index category. Now U is the set containing all type names used in syntax. It is similar to a universe used in Martin-Löf type theory (i.e. the set U of all codes of small sets (= types)), hence we call U *type universe* hereafter. In the case of simply typed λ-calculus, we take U to be the set of all simple types generated by base types. The presheaf category $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ is now our working category.

The intention of the use of $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ is that the inner index $\mathbb{F}\downarrow U$ models contexts, and the outer index U models the target types of judgments. The category $\mathbb{F}\downarrow U$ has objects $\Gamma : n \to U$, which are seen as contexts $\{1 : \tau_1, \cdots, n : \tau_n\}$, and arrows $\rho : \Gamma \to \Gamma'$, which are functions $\rho : n \to n'$ such that $\Gamma = \Gamma' \circ \rho$, i.e. renaming between Γ and Γ'. The coproduct is Γ, Γ' defined by $[\Gamma, \Gamma'] : n + n' \to U$.

All constructions used in the case of $\mathbf{Set}^{\mathbb{F}}$ are smoothly extended to the case of $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$, which makes modelling typed syntax possible. The context extension $\delta_\tau : \mathbf{Set}^{\mathbb{F}\downarrow U} \to \mathbf{Set}^{\mathbb{F}\downarrow U}$ by a variable of type τ is defined by $(\delta_\tau A)(\Gamma) = A(\Gamma, \tau)$. The presheaf $\mathrm{V} \in (\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ of variables is defined by the Yoneda embedding $\mathrm{V}_\tau = \mathbb{F}\downarrow U(\langle\tau\rangle, -)$, where $\langle\tau\rangle : 1 \to U$ maps $1 \mapsto \tau \in U$. Hence $\mathrm{V}_\tau(\Gamma) \cong \{x \mid x : \tau \in \Gamma\}$, i.e. the set of variables of a certain type τ taken from a context Γ.

For example, the signature functor $\Sigma_{\vec{\lambda}} : (\mathbf{Set}^{\mathbb{F}\downarrow U})^U \to (\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ for abstract syntax of simply-typed λ-terms can be defined by

$$(\Sigma_{\vec{\lambda}} A)_\tau = \mathrm{V}_\tau + \coprod_{\tau' \in U} (A_{\tau' \Rightarrow \tau} \times A_{\tau'}) + \coprod_{\tau_1, \tau_2 \in U} (\tau \equiv \tau_1 \Rightarrow \tau_2) \times (\delta_{\tau_1} A_{\tau_2})$$

[1] In the rigorous notation of comma category, $(\mathbb{F}\downarrow U)$ should be written as $(J_{\mathbb{F}} \downarrow U)$, where $J_{\mathbb{F}} : \mathbb{F} \to \mathbf{Set}$ is the inclusion functor.

Here the binary operator '≡' on types gives a set defined by $(\tau \equiv \tau') \triangleq 1$ (the one point set) if $\tau = \tau'$, $(\tau \equiv \tau') \triangleq 0$ (the empty set) if $\tau \neq \tau'$. This style using '≡' to define a signature functor can be found in [MA09]. Throughout this paper, we use this '≡' operator. The initial $\Sigma_{\vec{\lambda}}$-algebra $(\Lambda^{\rightarrow}, \mathsf{in})$ in $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ exists and can be constructed [SP82] as the presheaf of all simply-typed λ-terms

$$(\Lambda^{\rightarrow})_\tau(\Gamma) = \{t \mid \Gamma \vdash t : \tau\}/ =_\alpha$$

with algebra structure giving constructors of simply-typed λ terms.

This process is again generic with respect to arbitrary signature functor Σ, hence an initial Σ-algebra in $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ models arbitrary typed abstract syntax with variable binding.

Remark. These works do not intend to directly give semantics of λ-calculi. The initial algebras Λ and $\Lambda^{\rightarrow}$ are not models of untyped and typed λ-calculi respectively, because they do not validate the β-axioms. They are the initial models of *abstract syntax* of λ-calculi. Similarly, we do not intend to give models of polymorphic λ-calculi, system F and F_ω in this paper. We give algebraic models of *abstract syntax* of types and terms.

Convention on α-equivalence. In this paper, hereafter we use the method of *de Bruijn levels* [dB72] for representing bound and free variables in a term (and a type, a judgment, etc.). However, keeping de Bruijn level notation strictly (as in [Ham04, Ham05, Ham07]) is sometimes clumsy and hides the essence. Hence, in this paper, at the level of text, we use the usual named notation for terms to avoid clutter. We assume that these actually denote (or are automatically normalised to) de Bruijn level normal forms. For example, when we write $\lambda x.\lambda y.yx$, it actually means $\lambda 1.\lambda 2.21$. Another example is $\alpha_1, \ldots, \alpha_n \vdash \forall \alpha_{n+1}.\tau$ to mean $1, \ldots, n \vdash \forall (n+1.\tau)$. Hence we will drop the explicit quotienting "$/ =_\alpha$" by the α-equivalence in defining a term set hereafter. De Bruijn level notation is different from more well-known de Bruijn *index* notation, and levels are the reverse numbering of variables. See [FPT99] for illustrations of de Bruijn level notation.

3 Second-Order Polymorphic Abstract Syntax

We extend the treatment reviewed in the previous section to the case of second-order polymorphic abstract syntax with variable binding. The leading example of such a syntax is the abstract syntax for Girard and Reynolds' system F. Hence we review its definition.

3.1 System F

Types

$$\tau ::= \alpha \mid b \mid \tau_1 \Rightarrow \tau_2 \mid \forall \alpha.\tau$$

where α ranges over type variables, and b ranges over base types.

Well-formed types

$$\frac{1 \le i \le n}{\alpha_1, \ldots, \alpha_n \vdash \alpha_i} \qquad \frac{}{\alpha_1, \ldots, \alpha_n \vdash b}$$

$$\frac{\alpha_1, \ldots, \alpha_n \vdash \sigma \quad \alpha_1, \ldots, \alpha_n \vdash \tau}{\alpha_1, \ldots, \alpha_n \vdash \sigma \Rightarrow \tau} \qquad \frac{\alpha_1, \ldots, \alpha_n, \alpha_{n+1} \vdash \tau}{\alpha_1, \ldots, \alpha_n \vdash \forall \alpha_{n+1}.\tau}$$

Well-typed terms

$$\frac{x : \tau \in \Gamma}{\Xi \mid \Gamma \vdash x : \tau} \qquad \frac{\Xi \mid \Gamma, x : \sigma \vdash t : \tau}{\Xi \mid \Gamma \vdash \lambda x : \sigma. t : \sigma \Rightarrow \tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \sigma \Rightarrow \tau \quad \Xi \mid \Gamma \vdash s : \sigma}{\Xi \mid \Gamma \vdash t\, s : \tau}$$

$$\frac{\Xi, \alpha \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \Lambda \alpha. t : \forall \alpha. \tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \forall \alpha. \tau \quad \Xi \vdash \sigma}{\Xi \mid \Gamma \vdash t\, \sigma : \tau[\alpha := \sigma]}$$

Notes

- $\Xi \mid \Gamma \vdash t : \tau$ is well-formed if $\Xi \vdash \tau_i$ for each $x_i : \tau_i \in \Gamma$, and $\Xi \vdash \tau$.
- $\Xi = \alpha_1, \ldots, \alpha_n$ is a *type context*, i.e., a sequence of type variables.
- $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$ is a *term context*.

We use the formulation that a type context and a term context are explicitly separated in a judgment as $\Xi \mid \Gamma$, where any type variable α appearing in Γ is taken from Ξ. The substitution operation $[- := -]$ on terms and types is the standard capture-avoiding substitution.

3.2 Modelling Syntax of F

First we concentrate on modelling syntax for system F. We generalise it to arbitrary polymorphic abstract syntax later.

The basic idea we take is to use algebras in $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$ as in §2.2. Now the type universe U is not merely a *set* of all types, since types involve type variables and quantification. This means that U must be given by abstract syntax *with variable binding*. This point of view was also taken in [AHS96].

We use a two-stage approach to model system F terms. Firstly, we construct the universe $\mathbb{T}$ of all system F types as a *presheaf* $\mathbb{T} \in \mathbf{Set}^{\mathbb{F}}$ by an initial algebra in $\mathbf{Set}^{\mathbb{F}}$. Then we move to another presheaf category $\mathbf{Set}^{\int \mathsf{G}}$ (explained later) defined using $\mathbb{T}$, and construct an initial algebra for all well-typed terms in it. We proceed by the following three steps.

(I) Polymorphic types. We follow the method reviewed in §2.1. Let $\mathrm{B} \in \mathbf{Set}^{\mathbb{F}}$ be the constant functor to the set of all base types, and $\mathbb{V}$ the presheaf of type variables defined by $\mathbb{V}(n) = \mathbb{F}(1, n) \cong \{1, \ldots, n\}$. We define the signature functor $\mathrm{F}^{\mathrm{ty}} : \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ for system F types by

$$\mathrm{F}^{\mathrm{ty}}(A) = \mathbb{V} + \mathrm{B} + A \times A + \delta A.$$

Each summand corresponds to the arity of type variable, base type, arrow type, and universal type. An initial F^{ty}-algebra exists and can be constructed. We define $\mathbb{T} \in \mathbf{Set}^{\mathbb{F}}$ by the initial F^{ty}-algebra $(\mathbb{T}, \mathsf{in})$ described as the presheaf of all well-formed types:

$$\mathbb{T}(n) = \{\tau \mid \alpha_1, \ldots, \alpha_n \vdash \tau\}.$$

The arrow part $\mathbb{T}(\rho)$ is a renaming action on types using ρ defined by structural recursion [FPT99], i.e., $\mathbb{T}(\rho)(\tau)$ renames each type variable in a type τ by ρ. The algebra structure $\mathsf{in} : \mathrm{F}^{\mathrm{ty}}(\mathbb{T}) \to \mathbb{T}$ consists of constructors of system F types

$$\mathsf{tvar} : \mathbb{V} \to \mathbb{T} \qquad \mathsf{base} : \mathrm{B} \to \mathbb{T} \qquad \mathsf{arrow} : \mathbb{T} \times \mathbb{T} \to \mathbb{T} \qquad \mathsf{forall} : \delta\mathbb{T} \to \mathbb{T}.$$

(II) Contexts. In order to model terms, next we need to choose a presheaf category on some index category using $\mathbb{T}$. We basically follow the style to use $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U \simeq \mathbf{Set}^{(\mathbb{F}\downarrow U)\times U}$ for modelling terms as in §2.2. So we need to choose a suitable type universe U. Now let's try to choose the disjoint union: $U = \coprod_{n\in\mathbb{N}} \mathbb{T}(n)$. But this is imprecise, because in this attempt $\mathbf{Set}^{(\mathbb{F}\downarrow\coprod_{n\in\mathbb{N}}\mathbb{T}(n))\times\coprod_{n\in\mathbb{N}}\mathbb{T}(n)}$, the index n in the left sum on the index category does not synchronize with the index n in the right sum (since each n is locally bound by each sum). These must be equal because

$$n \mid \Gamma \vdash t : \tau \text{ is well-formed} \quad \Leftrightarrow \quad \text{for every } \tau_i \text{ in } \Gamma,\, n \vdash \tau_i \text{ and } n \vdash \tau.$$

Another attempt to use $\mathbf{Set}^{\coprod_{n\in\mathbb{N}}(\mathbb{F}\downarrow\mathbb{T}(n)\times\mathbb{T}(n))}$ is again insufficient because this does not model renaming between two terms in different type contexts n and n'.

The right way to combine all of these $\mathbb{T}(n)$ for the index category is the *Grothendieck construction*. Before going to it, we need to state the following.

Definition 1. (**Categories of context-with-types**) Let $n \in \mathbb{N}$. We use a comma category $\mathbb{F}\downarrow(\mathbb{T}(n))$, where $\mathbb{T}(n)$ is a set. We also regard $\mathbb{T}(n)$ as a discrete category. Then we take the product: *Category* $\mathbb{F}\downarrow(\mathbb{T}(n)) \times \mathbb{T}(n)$

- objects (Γ, τ) where $\Gamma \in \mathbb{F}\downarrow(\mathbb{T}(n)),\ \tau \in \mathbb{T}(n)$
- arrows $\pi : (\Gamma, \tau) \to (\Delta, \tau)$ given by a renaming $\pi : \Gamma \to \Delta$ in $\mathbb{F}\downarrow(\mathbb{T}(n))$.

We use the Grothendieck construction to glue all categories of context-with-types together. We recall the construction [Gro70].

Definition 2. (**Grothendieck**) Given a functor $\mathcal{F} : C^{\mathrm{op}} \to \mathbf{Cat}$, the *Grothendieck construction* of $\mathcal{F}$ is a category $\int \mathcal{F}$ with objects (I, A) where $I \in C$ and $A \in \mathcal{F}(I)$, and arrows $(u, \gamma) : (I, A) \to (J, B)$ where $u : J \to I$ in C^{op} and $\gamma : \mathcal{F}(u)(A) \to B$ in $\mathcal{F}(J)$.

We now define a functor $\mathsf{G} : \mathbb{F}^{\mathrm{op}} \to \mathbf{Cat}$ by

$$\begin{aligned} \mathsf{G}(x) &= \mathbb{F}\downarrow(\mathbb{T}(x)) \times \mathbb{T}(x) \\ \mathsf{G}(f) &= \mathbb{F}\downarrow(\mathbb{T}(f)) \times \mathbb{T}(f) \qquad \text{for } f : x \to y \text{ in } \mathbb{F} \end{aligned}$$

The Grothendieck construction $\displaystyle\int \mathsf{G}$ has

- objects $(n \mid \Gamma \vdash \tau)$, where $n \in \mathbb{F}$, $\Gamma \in \mathbb{F}\downarrow(\mathbb{T}(n))$, $\tau \in \mathbb{T}(n)$,
- arrows $(\rho, \pi) : (m \mid \Gamma \vdash \tau) \to (n \mid \Delta \vdash \sigma)$,
  where $\rho : m \to n$ in $\mathbb{F}$ such that $\mathbb{T}(\rho)(\tau) = \sigma$, and
  $\pi : (\mathbb{F}\downarrow\mathbb{T}\rho)(\Gamma) \to \Delta$ in $\mathbb{F}\downarrow(\mathbb{T}(n))$.

We now explain why objects and arrows are described as above and their syntactic meaning. If we follow the above definition strictly, an object of $\int \mathsf{G}$ should be $(n, (\Gamma, \tau))$, where $n \in \mathbb{F}$ and $(\Gamma, \tau) \in \mathbb{F}\!\downarrow\!(\mathbb{T}(n)) \times \mathbb{T}(n)$. We merely use another notation $(n \mid \Gamma \vdash \tau)$ for this triple.

Meaning of arrows. For arrows, the above description is obtained by expanding the definition. An arrow $(\rho, \pi) : (m \mid \Gamma \vdash \tau) \to (n \mid \Delta \vdash \sigma)$ consists of a renaming $\rho : m \to n$ between type contexts, and a renaming $\pi : (\mathbb{F}\!\downarrow\!\mathbb{T}\rho)\,(\Gamma) \to \Delta$ between term contexts.

Moreover, an arrow (ρ, π) in $\int \mathsf{G}$ can be understood as renaming type and term variables between two judgments having different contexts and result types. To discuss it, we first define the indexed set T of all well-typed terms by

$$\mathrm{T}(\Xi \mid \Gamma \vdash \tau) = \{t \mid (\Xi \mid \Gamma \vdash t : \tau) \text{ is derivable}\}.$$

Let $\rho^\sharp t$ denote renaming each type variable in a term t by ρ, and $\pi^\sharp t$ denotes renaming each term variable in a term t by π. Then, we have an admissible rule for well-typedness of renamed term:

$$\frac{m \mid \Gamma \vdash t : \tau}{n \mid \Delta \vdash \pi^\sharp \rho^\sharp(t) : \mathbb{T}(\rho)(\tau)} \text{ by applying } (\rho, \pi) \text{ in } \int \mathsf{G}$$

This process defines the arrow part of T being a presheaf in $\mathbf{Set}^{\int \mathsf{G}}$

$$\mathrm{T}(\rho, \pi) : \mathrm{T}(m \mid \Gamma \vdash \tau) \to \mathrm{T}(n \mid \Delta \vdash \sigma); \quad t \mapsto \pi^\sharp \rho^\sharp(t)$$

where $\mathbb{T}(\rho)(\tau) = \sigma$. When ρ is the identity $\mathrm{id}_n : n \to n$, then $\mathrm{T}(\mathrm{id}_n, \pi) : \mathrm{T}(n \mid \Gamma \vdash \tau) \to \mathrm{T}(n \mid \Gamma' \vdash \tau)$ is the usual renaming on Γ.

Hence, the Grothendieck construction provides the category of context-with-types and renamings in a mathematically uniform way.

(III) Terms. Now our working category is $\mathbf{Set}^{\int \mathsf{G}}$. As we have seen, system F terms form a presheaf T in $\mathbf{Set}^{\int \mathsf{G}}$. The presheaf $\mathrm{V} \in \mathbf{Set}^{\int \mathsf{G}}$ of variables is defined by

$$\begin{aligned} \mathrm{V}(n \mid \Gamma \vdash \tau) &= (\mathbb{F}\!\downarrow\!\mathbb{T}(n))(\langle \tau \rangle, \Gamma) \;\cong\; \{x \mid x : \tau \in \Gamma\} \\ \mathrm{V}(\rho, \pi) &= \pi \circ -. \end{aligned}$$

We define the signature functor $\mathrm{F} : \mathbf{Set}^{\int \mathsf{G}} \to \mathbf{Set}^{\int \mathsf{G}}$ for system F terms by

$$\begin{aligned} \mathrm{F}(A)(n \mid \Gamma \vdash \tau) = \;& \mathrm{V}(n \mid \Gamma \vdash \tau) \\ &+ \coprod_{\tau_1, \tau_2 \in \mathbb{T}(n)} (\tau \equiv \tau_1 \Rightarrow \tau_2) \times A(n \mid \Gamma, \tau_1 \vdash \tau_2) \\ &+ \coprod_{\sigma \in \mathbb{T}(n)} A(n \mid \Gamma \vdash \sigma \Rightarrow \tau) \times A(n \mid \Gamma \vdash \sigma) \\ &+ \coprod_{\tau' \in \mathbb{T}(n+1)} (\tau \equiv \forall(\alpha.\tau')) \times A(n+1 \mid \mathsf{wk}(\Gamma) \vdash \tau') \\ &+ \coprod_{\substack{\sigma \in \mathbb{T}(n) \\ \tau' \in \mathbb{T}(n+1)}} (\tau \equiv \tau'[\alpha := \sigma]) \times A(n \mid \Gamma \vdash \forall(\alpha.\tau')). \end{aligned}$$

Each summand corresponds to the arity of variable, abstraction, application, type abstraction, and type application. The weakening $\mathsf{wk} : \mathbb{F}\downarrow\mathbb{T}(n) \to \mathbb{F}\downarrow\mathbb{T}(n+1)$ maps a context under n to the same context but under a weakened $n+1$. The use of "≡" operator is inessential to define the signature functor. It is merely a shortcut way of describing the definition by case analyse, see §3.3 for the general case.

Theorem 3. T *forms an initial* F-*algebra.*

Proof. An initial F-algebra is constructed by the colimit of the ω-chain $0 \to \mathrm{F}0 \to \mathrm{F}^2 0 \to \cdots$ [SP82]. These construction steps correspond to derivations of terms by term forming rules, hence their union T is the colimit. The algebra structure $\mathsf{in} : \mathrm{FT} \to \mathrm{T}$ of the initial algebra is obtained by one-step inference of the term forming rules, i.e., given by the following operations

$$
\begin{array}{llll}
\mathsf{var}_\tau & : \mathrm{V}(n \mid \Gamma \vdash \tau) & \to \mathrm{T}(n \mid \Gamma \vdash \tau) & ;\ x \mapsto x \\
\mathsf{abs}_{\sigma,\tau} & : \mathrm{T}(n \mid \Gamma, \sigma \vdash \tau) & \to \mathrm{T}(n \mid \Gamma \vdash \sigma \Rightarrow \tau) & ;\ t \mapsto \lambda x.t \\
\mathsf{app}_{\sigma,\tau} & : \mathrm{T}(n \mid \Gamma \vdash \sigma \Rightarrow \tau) & & \\
 & \quad \times \mathrm{T}(n \mid \Gamma \vdash \sigma)) & \to \mathrm{T}(n \mid \Gamma \vdash \tau) & ;\ s,t \mapsto s\,t \\
\mathsf{tabs}_{\tau'} & : \mathrm{T}(n+1 \mid \mathsf{wk}(\Gamma) \vdash \tau') & \to \mathrm{T}(n \mid \Gamma \vdash \forall(\alpha.\tau')) & ;\ t \mapsto \Lambda\alpha.t \\
\mathsf{tapp}_{\sigma,\tau'} & : \mathrm{T}(n \mid \Gamma \vdash \forall(\alpha.\tau')) & \to \mathrm{T}(n \mid \Gamma \vdash \tau'[\alpha := \sigma]) & ;\ t \mapsto t\,\sigma
\end{array}
$$

where $n \in \mathbb{N}$, $\sigma, \tau \in \mathbb{T}(n)$, $\tau' \in \mathbb{T}(n+1)$, and α is actually a de Bruijn level $n+1$. □

3.3 General Signature

Not only for system F, we seek a general framework for polymorphic abstract syntax. Generalising the case of system F, we arrive at the following definition.

Definition 4. A *polymorphic signature* $\Sigma = (\Sigma^{\mathsf{ty}}, \Sigma^{\mathsf{tm}})$ consists of the following data.

- Σ^{ty} for types is a *binding signature* [Acz78, FPT99], i.e. a set of type formers with an arity function $a : \Sigma^{\mathsf{ty}} \to \mathbb{N}^*$ (NB. '*' denotes the Kleene closure). A type former of arity $\langle n_1, \ldots, n_l \rangle$, denoted by $o : \langle n_1, \ldots, n_l \rangle$, has l arguments and binds n_i variables in the i-th argument ($1 \le i \le l$).
- Let $\mathbb{T} \in \mathbf{Set}^{\mathbb{F}}$ be the free Σ^{ty}-algebra over $\mathbb{V}$, represented by term syntax in de Bruijn levels [FPT99, Ham04].
- Σ^{tm} for terms is a set of function symbols with arities. This is denoted by

$$f : \langle k_1 \rangle (\vec{\sigma}_1)\tau_1, \ldots, \langle k_l \rangle (\vec{\sigma}_l)\tau_l \to \tau$$

  where $n \in \mathbb{N}, k_i \in \mathbb{N}, \vec{\sigma}_i \in \mathbb{T}(n+k_i)^*, \tau_i \in \mathbb{T}(n+k_i), \tau \in \mathbb{T}(n)$, has l arguments, and binds k_i type variables and $|\vec{\sigma}_i|$ variables in the i-th argument ($1 \le i \le l$). In case of $k_i = 0$ or $|\vec{\sigma}_i| = 0$, each part is omitted. Here, $|-|$ denotes the length of a sequence.

Example 5. The polymorphic signature $\Sigma_{\mathrm{F}} = (\Sigma_{\mathrm{F}}^{\mathsf{ty}}, \Sigma_{\mathrm{F}}^{\mathsf{tm}})$ for system F can be given as follows: $\Sigma_{\mathrm{F}}^{\mathsf{ty}} = \{b : \langle 0 \rangle, \Rightarrow : \langle 0, 0 \rangle, \forall : \langle 1 \rangle\}$, and $\Sigma_{\mathrm{F}}^{\mathsf{tm}}$ is

$$
\begin{array}{llll}
\mathsf{abs}_{\sigma,\tau} : (\sigma)\tau \to \sigma \Rightarrow \tau & & \mathsf{app}_{\sigma,\tau} & : \sigma \Rightarrow \tau, \sigma \to \tau \\
\mathsf{tabs}_{\tau'} : \langle 1 \rangle \tau' \to \forall(\alpha.\tau') & & \mathsf{tapp}_{\sigma,\tau'} & : \forall(\alpha.\tau') \to \tau'[\alpha := \sigma]
\end{array}
$$

generated by all $n \in \mathbb{N}$, $\sigma, \tau \in \mathbb{T}(n)$, $\tau' \in \mathbb{T}(n+1)$, and α should be regarded as a de Bruijn level $n+1$.

Example 6. If we want to add the let-binding construct, Σ^{tm} has the function symbol $\mathsf{let} : \sigma,\ (\sigma)\tau \to \tau$, where $\sigma, \tau \in \mathbb{T}(n)$.

To a polymorphic signature Σ, we associate the *signature functor* $\Sigma : \mathbf{Set}^{\int \mathsf{G}} \to \mathbf{Set}^{\int \mathsf{G}}$ given by

$$\Sigma A(n \mid \Gamma \vdash \tau) = \coprod_{f:\langle k_1\rangle(\vec{\sigma}_1)\tau_1,\ldots,\langle k_l\rangle(\vec{\sigma}_l)\tau_l \to \tau \in \Sigma^{\mathsf{tm}}} \ \prod_{1\le i\le l} A(n+k_i \mid \Gamma, \vec{\sigma}_i \vdash \tau_i).$$

3.4 General Syntax Rules

Using a polymorphic signature Σ, we can construct polymorphic terms syntactically and generally. The following construction rules are extracted from the semantic structure we have obtained so far.

Well-formed types

$$\frac{1 \le i \le n}{\alpha_1, \ldots, \alpha_n \vdash \alpha_i} \qquad \frac{\Xi, \vec{\alpha}_1 \vdash \tau_1 \ \ldots \ \Xi, \vec{\alpha}_l \vdash \tau_l}{\Xi \vdash o(\vec{\alpha}_1.\tau_1, \ldots, \vec{\alpha}_l.\tau_l\tau)}$$

where $o : \langle n_1, \ldots, n_l\rangle \in \Sigma^{\mathsf{ty}}$, $|\vec{\alpha}_i| = n_i$.

Well-typed terms

$$\frac{x : \tau \in \Gamma}{\Xi \mid \Gamma \vdash x : \tau} \qquad \frac{\Xi, \vec{\alpha}_1 \mid \Gamma, \vec{x}_1 : \vec{\sigma}_1 \vdash t_1 : \tau_1 \quad \cdots \quad \Xi, \vec{\alpha}_l \mid \Gamma, \vec{x}_l : \vec{\sigma}_l \vdash t_l : \tau_l}{\Xi \mid \Gamma \vdash f(\vec{x}_1.t_1, \ldots, \vec{x}_l.t_l) : \tau}$$

where $f : \langle k_1\rangle(\vec{\sigma}_1)\tau_1, \ldots, \langle k_l\rangle(\vec{\sigma}_l)\tau_l \to \tau \in \Sigma^{\mathsf{tm}}$, $|\vec{\alpha}_i| = k_i$.

We define $\mathrm{TV}(\Xi \mid \Gamma \vdash \tau) \triangleq \{t \mid (\Xi \mid \Gamma \vdash t : \tau) \text{ is derivable}\}$, which we call *polymorphic abstract syntax*.

Theorem 7. *Given a polymorphic signature* Σ*,* TV *forms a free* Σ*-algebra over* V.

Proof. Similarly to Thm. 3. Notice that a free Σ-algebra over V is an initial V + Σ-algebra. □

4 Higher-Order Polymorphic Abstract Syntax

We extend the previous algebraic characterisation to the case of higher-order polymorphic abstract syntax with variable binding. The leading example of such a syntax is the abstract syntax for Girard's system F_ω. Hence we review its definition.

4.1 System F_ω

Kinds and types $\kappa ::= * \mid \kappa_1 \Rightarrow \kappa_2$

$\tau ::= \alpha \mid b \mid \tau_1 \Rightarrow \tau_2 \mid \forall\alpha : \kappa.\tau \mid \lambda\alpha : \kappa.\tau \mid \tau_1\tau_2$

Well-kinded types

$$\frac{\alpha : \kappa \in \Xi}{\Xi \vdash \alpha : \kappa} \qquad \frac{}{\Xi \vdash b : \kappa} \qquad \frac{\Xi, \alpha : \kappa' \vdash \tau : \kappa}{\Xi \vdash \lambda\alpha : \kappa'.\tau : \kappa' \Rightarrow \kappa}$$

$$\frac{\Xi \vdash \sigma : * \quad \Xi \vdash \tau : *}{\Xi \vdash \sigma \Rightarrow \tau : *} \qquad \frac{\Xi, \alpha : \kappa \vdash \tau : *}{\Xi \vdash \forall\alpha : \kappa.\tau : *} \qquad \frac{\Xi \vdash \sigma : \kappa' \Rightarrow \kappa \quad \Xi \vdash \tau : \kappa'}{\Xi \vdash \sigma\tau : \kappa}$$

Well-typed terms

$$\frac{x : \tau \in \Gamma}{\Xi \mid \Gamma \vdash x : \tau} \qquad \frac{\Xi \mid \Gamma, x : \sigma \vdash t : \tau}{\Xi \mid \Gamma \vdash \lambda x : \tau.t : \sigma \Rightarrow \tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \sigma \Rightarrow \tau \quad \Xi \mid \Gamma \vdash s : \sigma}{\Xi \mid \Gamma \vdash t\,s : \tau}$$

$$\frac{\Xi, \alpha : \kappa \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \Lambda\alpha : \kappa.t \ : \ \forall\alpha : \kappa.\tau} \qquad \frac{\Xi \mid \Gamma \vdash t \ : \ \forall\alpha : \kappa.\tau \quad \Xi \vdash \sigma : \kappa}{\Xi \mid \Gamma \vdash t\sigma \ : \ \tau[\alpha := \sigma]}$$

Notes

- $\Xi \mid \Gamma \vdash t : \tau$ is well-formed if $\Xi \vdash \tau_i : \kappa_i$ for each $x_i : \tau_i \in \Gamma$ and $\Xi \vdash \tau : *$.
- $\Xi = \alpha_1 : \kappa_1, \ldots, \alpha_n : \kappa_n$ is a sequence of (type variable,kind)-pairs.

4.2 Modelling Syntax of F_ω

First we concentrate on modelling abstract syntax for system F_ω terms. We generalise it to arbitrary higher-order polymorphic abstract syntax later.

The basic idea we take is again to follow the style using algebras in $(\mathbf{Set}^{\mathbb{F}\downarrow U})^U$. In the case of system F, U was given by *untyped* abstract syntax with variable binding since types contain universal quantification. In system F_ω, a type has a kind. This means that U must be given by *typed* (= used for kinding) abstract syntax with variable binding.

We use again a two-stage approach to model system F_ω terms, but the working categories are different from the previous case. Let $\mathcal{K}$ be the set of all kinds (and considered as a discrete category). Firstly, we construct the universe $\mathbb{T}$ of all system F_ω types as a presheaf $\mathbb{T} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$ by an initial algebra in the category $(\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$. Then we move to another presheaf category $\mathbf{Set}^{\int \mathsf{H}}$ (explained later) defined using $\mathbb{T}$, and construct an initial algebra for all well-typed F_ω terms in it.

(I) Kinded types. Let $\mathrm{B}_\kappa \in \mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}}$ be the constant functor to the set all base types of kind κ, $\mathbb{V} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$ the presheaf of kinded type variables defined by $\mathbb{V}_\kappa = \mathbb{F}\downarrow\mathcal{K}(\kappa, -)$. We define the signature functor $\mathrm{F}^{\mathrm{ty}}_\omega : (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}} \to (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$ for system F_ω types by

$$\begin{aligned}\mathrm{F}^{\mathrm{ty}}_\omega(A)_\kappa &= \mathbb{V}_\kappa + \mathrm{B}_\kappa + (\kappa \equiv *) \times (A_* \times A_* + \delta_\kappa A_*)\\ &\quad + \coprod_{\kappa_1,\kappa_2 \in \mathcal{K}} ((\kappa \equiv \kappa_1 \Rightarrow \kappa_2) \times \delta_{\kappa_1} A_{\kappa_2}) + \coprod_{\kappa' \in \mathcal{K}} (A_{\kappa' \Rightarrow \kappa} \times A_{\kappa'}).\end{aligned}$$

Each summand corresponds to the arity of type variable, base type, (arrow type and universal type), type-level λ, and type-level application.

An initial F_ω^{ty}-algebra exists as in §2.2. We define $\mathbb{T} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$ by an initial F_ω^{ty}-algebra $(\mathbb{T}, \mathsf{in})$ described as the presheaf of all well-kinded types[2]

$$\mathbb{T}_\kappa(\alpha_1 : \kappa_1, \ldots, \alpha_n : \kappa_n) = \{\tau \mid \alpha_1 : \kappa_1, \ldots, \alpha_n : \kappa_n \vdash \tau : \kappa\},$$

with algebra structure consisting of constructors

$$\begin{array}{lll} \mathsf{tvar}_\kappa : \mathbb{V}_\kappa \to \mathbb{T}_\kappa & \mathsf{base}_\kappa : \mathrm{B}_\kappa \to \mathbb{T}_\kappa & \mathsf{tyabs}_{\kappa_1,\kappa_2} : \delta_{\kappa_1}\mathbb{T}_{\kappa_2} \to \mathbb{T}_{\kappa_1 \Rightarrow \kappa_2} \\ \mathsf{arrow} : \mathbb{T}_* \times \mathbb{T}_* \to \mathbb{T}_* & \mathsf{forall}_\kappa : \delta_\kappa \mathbb{T}_* \to \mathbb{T}_* & \mathsf{tyapp}_{\kappa',\kappa} : \mathbb{T}_{\kappa' \Rightarrow \kappa} \times \mathbb{T}_{\kappa'} \to \mathbb{T}_\kappa. \end{array}$$

These six arrows of $\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}}$ correspond to the six rules of *Well-kinded types* of F_ω.

(II) Contexts. Let $\Xi \in \mathbb{F}\downarrow\mathcal{K}$. We now take $\mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi)$ to be the category of contexts. As we have seen, $\mathbb{T}_\kappa(\Xi)$ is the set of all types of kind κ under $\Xi = \alpha_1 : \kappa_1, \ldots, \alpha_l : \kappa_l$. An object $\Gamma \in \mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi)$ is a map such that

$$\mathbb{F} \ni n \ni \text{a variable } x_i \stackrel{\Gamma}{\longmapsto} \tau_i \in \mathbb{T}_\kappa(\Xi) \qquad \text{i.e. } \Xi \vdash \tau_i : \kappa.$$

Hence, Γ expresses a context $x_1 : \tau_1, \ldots, x_n : \tau_n$, and all types τ_i are of kind κ. The set $\mathbb{T}_\kappa(\Xi)$ is also regarded as a discrete category.

(III) Terms. Again, we use the Grothendieck construction to glue all categories of context-with-types together. We define a functor $\mathsf{H} : (\mathbb{F}\downarrow\mathcal{K} \times \mathcal{K})^{\mathsf{op}} \to \mathbf{Cat}$ by

$$\begin{aligned} \mathsf{H}(\Xi, \kappa) &= \mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi) \times \mathbb{T}_\kappa(\Xi) \\ \mathsf{H}(\rho, \mathrm{id}_\kappa) &= \mathbb{F}\downarrow(\mathbb{T}_\kappa\rho) \times \mathbb{T}_\kappa(\rho) \quad \text{for } \rho : \Xi \to \Xi' \text{ in } \mathbb{F}\downarrow\mathcal{K}. \end{aligned}$$

The Grothendieck construction $\int \mathsf{H}$ has

- objects $(\Xi \mid \Gamma \vdash \tau : \kappa)$, where $\Xi \in \mathbb{F}\downarrow\mathcal{K}$, $\kappa \in \mathcal{K}$, $\Gamma \in \mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi)$, $\tau \in \mathbb{T}_\kappa(\Xi)$,
- arrows $(\rho, \pi) : (\Theta \mid \Gamma \vdash \tau : \kappa) \to (\Xi \mid \Delta \vdash \sigma : \kappa)$,
  where $\rho : \Theta \to \Xi$ in $\mathbb{F}\downarrow\mathcal{K}$ such that $\mathbb{T}_\kappa(\rho)(\tau) = \sigma$, and
  $\pi : (\mathbb{F}\downarrow\mathbb{T}_\kappa\rho)(\Gamma) \to \Delta$ in $\mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi)$.

Now our working category is $\mathbf{Set}^{\int \mathsf{H}}$. We define $\mathrm{T}_\omega \in \mathbf{Set}^{\int \mathsf{H}}$ of all well-typed terms by

$$\begin{aligned} \mathrm{T}_\omega(\Xi \mid \Gamma \vdash \tau : *) &= \{t \mid (\Xi \mid \Gamma \vdash t : \tau) \text{ is derivable}\} \\ \mathrm{T}_\omega(\Xi \mid \Gamma \vdash \tau : \kappa) &= \varnothing \qquad \text{if } \kappa \neq * \end{aligned}$$

The second clause is due to that there are no terms of higher-kinded types in F_ω. The arrow part is defined similarly to the case of system F.

The presheaf $\mathrm{V} \in \mathbf{Set}^{\int \mathsf{H}}$ of term variables is defined by

$$\begin{aligned} \mathrm{V}(\Xi \mid \Gamma \vdash \tau : \kappa) &= (\mathbb{F}\downarrow(\mathbb{T}_\kappa\Xi))(\langle\tau\rangle, \Gamma) \cong \{x \mid x : \tau \in \Gamma\} \\ \mathrm{V}(\rho, \pi) &= \pi \circ -. \end{aligned}$$

[2] Usually, system F_ω types are identified modulo type-level β-conversion. Since we only focus on abstract syntax in this paper, we do not treat this process here. This will be uniformly treated within general polymorphic equational logic (cf. the discussion in §5).

We define the signature functor $\mathrm{F}_\omega : \mathbf{Set}^{\int \mathsf{H}} \to \mathbf{Set}^{\int \mathsf{H}}$ for system F_ω terms by

$$\begin{aligned}\mathrm{F}_\omega(A)(\Xi \mid \Gamma \vdash \tau : *) = \;& \mathrm{V}(\Xi \mid \Gamma \vdash \tau : *)\\ &+ \coprod_{\tau_1,\tau_2\in\mathbb{T}_*(\Xi)} (\tau \equiv \tau_1 \Rightarrow \tau_2) \times A(\Xi \mid \Gamma, \tau_1 \vdash \tau_2 : *)\\ &+ \coprod_{\sigma\in\mathbb{T}_*(\Xi)} (A(\Xi \mid \Gamma \vdash \sigma \Rightarrow \tau : *) \times A(\Xi \mid \Gamma \vdash \sigma : *))\\ &+ \coprod_{\tau'\in\mathbb{T}_*(\Xi,\,\alpha:\kappa)} (\tau \equiv \forall(\alpha : \kappa.\tau')) \times A(\Xi,\, \alpha : \kappa \mid \mathsf{wk}_{\alpha:\kappa}(\Gamma) \vdash \tau' : *)\\ &+ \coprod_{\substack{\sigma\in\mathbb{T}_*(\Xi)\\ \tau'\in\mathbb{T}_*(\Xi,\,\alpha:\kappa)}} (\tau \equiv \tau'[\alpha := \sigma]) \times A(\Xi \mid \Gamma \vdash \forall(\alpha : \kappa.\tau') : *).\end{aligned}$$

The weakening $\mathsf{wk}_{\alpha:\kappa} : \mathbb{F}\downarrow\mathbb{T}(\Xi) \to \mathbb{F}\downarrow\mathbb{T}(\Xi,\, \alpha : \kappa)$ maps a term context under a type context Ξ to the same term context but under a weakened one $(\Xi,\, \alpha : \kappa)$.

Theorem 8. T_ω *forms an initial* F_ω*-algebra.*

4.3 General Signature

Generalising the case of system F_ω, we arrive at the following definition.

Definition 9. A *higher-order polymorphic signature* $\Sigma = (\mathcal{K}, \Sigma^{\mathsf{ty}}, \Sigma^{\mathsf{tm}})$ consists of the following data.

- $\mathcal{K}$ is the set of all kinds.
- Σ^{ty} for types is a *second-order signature* [FH10], i.e. a set of type formers with an arity function $a : \Sigma^{\mathsf{ty}} \to (\mathcal{K}^* \times \mathcal{K})^* \times \mathcal{K}$. A type former with arity, denoted by

  $$o : (\vec{\sigma}_1)\tau_1, \ldots, (\vec{\sigma}_l)\tau_l \to \tau$$

  has l arguments, and binds $|\vec{\sigma}_i|$ variables of types $\vec{\sigma}_i$ in the i-th argument.
- Let $\mathbb{T} \in (\mathbf{Set}^{\mathbb{F}\downarrow\mathcal{K}})^{\mathcal{K}}$ be the free Σ^{ty}-algebra over $\mathbb{V}$, represented by term syntax. This is the presheaf of all types.
- Σ^{tm} for terms is a set of function symbols with arities. This is denoted by

  $$f : \langle\Theta_1\rangle(\vec{\sigma}_1)\tau_1 : \kappa_1, \ldots, \langle\Theta_l\rangle(\vec{\sigma}_l)\tau_l : \kappa_l \;\to\; \tau : \kappa$$

  where $\Xi, \Theta_i \in \mathbb{F}\downarrow\mathcal{K}$, $\vec{\sigma}_i \in \mathbb{T}_{\kappa_i}(\Xi, \Theta_i)^*$, $\tau_i \in \mathbb{T}_{\kappa_i}(\Xi, \Theta_i)$, $\tau \in \mathbb{T}_\kappa(\Xi)$, has l arguments, and binds $|\Theta_i|$ type variables (of kind κ_i) and $|\vec{\sigma}_i|$ variables (of types $\vec{\sigma}_i$ that have the same kind κ_i) in the i-th argument $(1 \le i \le l)$.

Example 10. The higher-order polymorphic signature $\Sigma_{\mathrm{F}_\omega} = (\mathcal{K}, \Sigma^{\mathsf{ty}}_{\mathrm{F}_\omega}, \Sigma^{\mathsf{tm}}_{\mathrm{F}_\omega})$ for system F_ω is as follows: $\mathcal{K} = \{*\} \cup \{\kappa_1 \Rightarrow \kappa_1 \mid \kappa_1, \kappa_2 \in \mathcal{K}\}$, $\Sigma^{\mathsf{ty}}_{\mathrm{F}_\omega}$ is

$$b : * \quad \Rightarrow : *, * \to * \quad \forall_\kappa : (\kappa)* \to * \quad \lambda_{\kappa,\kappa'} : (\kappa')\kappa \to \kappa' \Rightarrow \kappa \quad @_{\kappa,\kappa'} : \kappa' \Rightarrow \kappa,\, \kappa' \to \kappa$$

generated by all $\kappa, \kappa' \in \mathcal{K}$, and $\Sigma^{\mathsf{tm}}_{\mathrm{F}_\omega}$ is

$$\begin{array}{ll}\mathsf{abs}_{\sigma,\tau} : (\sigma)\tau : * \to \sigma \Rightarrow \tau : * & \mathsf{app}_{\sigma,\tau} : \sigma \Rightarrow \tau : *,\; \sigma : * \to \tau : *\\ \mathsf{tabs}_{\tau',\kappa} : \langle\kappa\rangle\tau' : * \to \forall_\kappa(\alpha.\tau') : * & \mathsf{tapp}_{\sigma,\tau'} : \forall_\kappa(\alpha.\tau') : * \quad \to \tau'[\alpha := \sigma] : *\end{array}$$

generated by all $\kappa \in \mathcal{K}$, $\Xi \in \mathbb{F}\downarrow\mathcal{K}$, $\kappa' \in \mathcal{K}$, $\sigma, \tau \in \mathbb{T}_{\kappa'}(\Xi)$, $\tau' \in \mathbb{T}_{\kappa'}(\Xi, \alpha)$.

Example 11. The higher-order polymorphic signature for system F is given by $\Sigma_{\mathrm{F}} = (\{*\}, \Sigma^{\mathsf{ty}}_{\mathrm{F}_\omega} - \{\lambda_{*,*}, @_{*,*}\}), \Sigma^{\mathsf{tm}}_{\mathrm{F}_\omega})$.

To a higher-order polymorphic signature Σ, we associate the *signature functor* $\Sigma : \mathbf{Set}^{\int \mathsf{H}} \to \mathbf{Set}^{\int \mathsf{H}}$ given by

$$\Sigma A(\Xi \mid \Gamma \vdash \tau : \kappa) = \coprod_{f:\langle\Theta_1\rangle(\vec{\sigma}_1)\tau_1:\kappa_1,\ldots\to\tau:\kappa\in\Sigma^{\mathsf{tm}}} \prod_{1\le i\le l} A(\Xi, \Theta_i \mid \Gamma, \vec{\sigma}_i \vdash \tau_i : \kappa_i).$$

4.4 General Syntax Rules

If $\Xi \vdash \tau_i : \kappa$ for all $x_i : \tau_i \in \Gamma$, and $\Xi \vdash \tau : \kappa$, then a term judgment $\Xi \mid \Gamma \vdash t : \tau : \kappa$ is well-formed.

Well-kinded types

$$\frac{1 \le i \le n}{\alpha_1 : \kappa_1, \ldots, \alpha_n : \kappa_n \vdash \alpha_i : \kappa_i} \qquad \frac{\Xi, \vec{\alpha}_1 : \vec{\kappa}'_1 \vdash \tau_1 : \kappa_1 \ \cdots \ \Xi, \vec{\alpha}_l : \vec{\kappa}'_l \vdash \tau_l : \kappa_l}{\Xi \vdash o(\vec{\alpha}_1.\tau_1, \ldots, \vec{\alpha}_l.\tau_l\tau) : \kappa}$$

where $o : (\vec{\kappa}'_1)\kappa_1, \ldots, (\vec{\kappa}'_l)\kappa_l \to \kappa \in \Sigma^{\mathsf{ty}}$.

Well-typed terms

$$\frac{x : \tau \in \Gamma}{\Xi \mid \Gamma \vdash x : \tau : \kappa}$$

$$\frac{\Xi, \Theta_1 \mid \Gamma, \vec{x}_1 : \vec{\tau}_1 \vdash t_1 : \tau_1 : \kappa_1 \quad \cdots \quad \Xi, \Theta_l \mid \Gamma, \vec{x}_l : \vec{\tau}_l \vdash t_l : \tau_l : \kappa_l}{\Xi \mid \Gamma \vdash f(\vec{x}_1.t_1, \ldots, \vec{x}_l.t_l) : \tau : \kappa}$$

where $f : \langle\Theta_1\rangle(\vec{\sigma}_1)\tau_1 : \kappa_1, \ldots, \langle\Theta_l\rangle(\vec{\sigma}_l)\tau_l : \kappa_l \to \tau : \kappa \in \Sigma^{\mathsf{tm}}$.

We define $\mathrm{TV}(\Xi \mid \Gamma \vdash \tau : \kappa) \triangleq \{t \mid (\Xi \mid \Gamma \vdash t : \tau : \kappa)$ is derivable$\}$, which we call *higher-order polymorphic abstract syntax*.

Theorem 12. *Given a higher-order polymorphic signature* Σ*,* TV *forms a free* Σ*-algebra over* V.

5 On Substitutions and Future Work

In this paper, we have focused on abstract syntax. In this final section, we briefly consider the equational axioms of system F and F_ω, and how we can express them in our framework. These remarks pertain to future work on seeking a general equational logic on polymorphic terms.

System F has the axioms:

$$\begin{array}{lllll} (\beta) & \Xi \mid \Gamma \vdash (\lambda x{:}\,\sigma.\,t)\,s & = & t[x := s] & : \tau \\ (\text{type app.}) & \Xi \mid \Gamma \vdash (\Lambda\alpha.t)\,\sigma & = & t[\alpha := \sigma] & : \tau[\alpha := \sigma] \end{array}$$

The terms of the left-hand sides of equations are just elements of the presheaf T of terms. In the right-hand sides and in types, various substitutions are used. We can model these as follows.

Substitution on types: $\tau[\alpha := \sigma]$**.** The category $\mathbf{Set}^{\mathbb{F}}$ has so-called a substitution monoidal structure [FPT99] $(\mathbf{Set}^{\mathbb{F}}, \bullet, \mathbb{V})$, where the monoidal product is given by a co-end $(A \bullet B)(n) = \int^{m \in \mathbb{F}} A(m) \times B(n)^m$. The presheaf $\mathbb{T}$ of system F types is a monoid in $(\mathbf{Set}^{\mathbb{F}}, \bullet, \mathrm{V})$, and its multiplication $\mu^{\mathbb{T}} : \mathbb{T} \bullet \mathbb{T} \to \mathbb{T}$ models the substitution operation on types.

Substitution on terms: $t[x := s]$**.** Since both terms t and s are under the same type context Ξ, it suffices to consider the substitution monoidal structure in $(\mathbf{Set}^{\mathbb{F}\Downarrow(\mathbb{T}(n))})^{\mathbb{T}(n)}$ for each $n = |\Xi| \in \mathbb{N}$. This case is covered by the substitution structure explored in [MS03, FH10], i.e. $(A \bullet B)_\tau(\Gamma) = \int^{\Delta \in \mathbb{F}\Downarrow(\mathbb{T}(n))} A_\tau(\Delta) \times \prod_{1 \leq i \leq |\Delta|} B_{\Delta(i)}(\Gamma)$. The presheaf $\mathrm{T}(n \mid - \vdash -) \in (\mathbf{Set}^{\mathbb{F}\Downarrow(\mathbb{T}(n))})^{\mathbb{T}(n)}$ of system F terms in a fixed type context n is a monoid in it, and its multiplication μ^{T} models the substitution operation on terms.

Substitution of a type for a type variable in a term: $t[\alpha := \sigma]$**.** It can be directly modelled by a map $\mathsf{tsub}_{\sigma,n} : \mathrm{T}(n+1 \mid \Gamma \vdash \tau) \to \mathrm{T}(n \mid \Gamma' \vdash \tau[n+1 := \sigma])$ defined by structural recursion on term that replaces each $n+1$ in a term with $\sigma \in \mathrm{T}(n)$ using $\mu^{\mathbb{T}}$, where $n+1$ is the de Bruijn level of the type variable α. The context Γ' is the one obtained by replacing all $n+1$ with σ in Γ.

Substitution on kinded types: $\tau[\alpha := \sigma]$**.** System F_ω has additionally the axiom

$$(\text{type}\,\beta) \qquad \Xi \mid \Gamma \vdash (\lambda\alpha{:}\,\kappa.\,\tau)\,\sigma \quad = \quad \tau[\alpha := \sigma] : \kappa'$$

The substitution in the right-hand side of the equation is modelled using the substitution monoidal structure in $(\mathbf{Set}^{\mathbb{F}\Downarrow\mathcal{K}})^{\mathcal{K}}$ again following [MS03, FH10]. The presheaf $\mathbb{T}$ of F_ω types is a monoid in $((\mathbf{Set}^{\mathbb{F}\Downarrow\mathcal{K}})^{\mathcal{K}}, \bullet, \mathbb{V})$, and its multiplication $\mu^{\mathbb{T}} : \mathbb{T} \bullet \mathbb{T} \to \mathbb{T}$ models the substitution operation on kinded types.

On the use of metavariables. When formalising an axiom, e.g. (β), there are actually two different views:

(1) (β) expresses infinitary many axioms generated by all concrete terms s, t.
(2) (β) should be regarded as a single axiom, where each letter "s" and "t" is the symbol of a *metavariable* denoting a concrete term.

Throughout this paper, we have taken the view (1). The view (2) was explored in [Ham04, Ham05, Fio08, FH10]. The presentation of axioms using metavariables is certainly more economical than (1), but technically more involved. This paper focuses on polymorphism in abstract syntax, so, for clarity, we did not go into the issue on metavariables. This should be explored in a future work.

Acknowledgments. I am grateful to Marcelo Fiore for important comments, especially related to the presheaf of variables. I also thank to the anonymous referees for useful comments that have improved the presentation of this paper. This work is supported by the JSPS Grant-in-Aid for Scientific Research (22700004).

References

[Acz78] Aczel, P.: A general Church-Rosser theorem. Technical report, University of Manchester (1978)

[AHS96] Altenkirch, T., Hofmann, M., Streicher, T.: Reduction-free normalisation for a polymorphic system. In: Proc. of LICS 1996, pp. 98–106 (1996)

[dB72] de Bruijn, N.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indagationes Mathematicae 34, 381–391 (1972)

[FH10] Fiore, M., Hur, C.-K.: Second-order equational logic. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 320–335. Springer, Heidelberg (2010)

[Fio02] Fiore, M.: Semantic analysis of normalisation by evaluation for typed lambda calculus. In: Proc. of PPDP 2002, pp. 26–37. ACM Press, New York (2002)

[Fio08] Fiore, M.: Second-order and dependently-sorted abstract syntax. In: Proc. of LICS 2008, pp. 57–68 (2008)

[Fio09] Fiore, M.: Algebraic meta-theories and synthesis of equational logics (2009), Research Programme

[FPT99] Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: Proc. of LICS 1999, pp. 193–202 (1999)

[Gro70] Grothendieck, A.: Catégories fibrées et descente (exposé VI). In: Grothendieck, A. (ed.) Revêtement Etales et Groupe Fondamental (SGA1). Lecture Notes in Mathematics, vol. 224, pp. 145–194. Springer, Heidelberg (1970)

[GTW76] Goguen, J., Thatcher, J., Wagner, E.: An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T. J. Watson Research Center (1976)

[GUH06] Ghani, N., Uustalu, T., Hamana, M.: Explicit substitutions and higher-order syntax. Higher-Order and Symbolic Computation 19(2/3), 263–282 (2006)

[Ham04] Hamana, M.: Free Σ-monoids: A higher-order syntax with metavariables. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 348–363. Springer, Heidelberg (2004)

[Ham05] Hamana, M.: Universal algebra for termination of higher-order rewriting. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 135–149. Springer, Heidelberg (2005)

[Ham07] Hamana, M.: Higher-order semantic labelling for inductive datatype systems. In: Proc. of PPDP 2007, pp. 97–108. ACM Press, New York (2007)

[Ham10] Hamana, M.: Initial algebra semantics for cyclic sharing tree structures. Logical Methods in Computer Science 6(3) (2010)

[Hof99] Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Proc. of LICS 1999, pp. 204–213 (1999)

[Kat04] Katsumata, S.: A generalisation of pre-logical predicates to simply typed formal systems. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 831–845. Springer, Heidelberg (2004)

[MA09] Morris, P., Altenkirch, T.: Indexed containers. In: LICS 2009, pp. 277–285 (2009)

[Mic08] Miculan, M.: A categorical model of the Fusion calculus. In: Proc. of MFPS XXIV. ENTCS, vol. 218, pp. 275–293. Elsevier, Amsterdam (2008)

[MS03] Miculan, M., Scagnetto, I.: A framework for typed HOAS and semantics. In: Proc. of PPDP 2003, pp. 184–194. ACM Press, New York (2003)

[SP82] Smyth, M.B., Plotkin, G.D.: The category-theoretic solution of recursive domain equations. SIAM J. Comput 11(4), 763–783 (1982)

[TP08] Tanaka, M., Power, J.: Category theoretic semantics for typed binding signatures with recursion. Fundam. Inform. 84(2), 221–240 (2008)