Syntax and Semantics of Abstract Binding Trees

Jon Sterling and Darin Morrison

Abstract binding trees (abts) are a generalization of abstract syntax trees where operators may express variable binding structure as part of their arities. Originally formulated by Peter Aczel [1], unisorted abts have been deployed successfully as the uniform syntactic framework for several implementations of Constructive Type Theory, including Nuprl [3], MetaPRL [13] and JonPRL [19].

In *Practical Foundations for Programming Languages* [11], Robert Harper develops the multi-sorted version of abstract binding trees and proposes an extension to include families of operators indexed by *symbols*, which are, unlike variables, subject to only distinctness-preserving renaming and not substitution; furthermore, symbols do not appear in the syntax of abts, and are only introduced as parameters to operators. This extension to support symbols is essential for a correct treatment of programming languages with open sums (e.g. ML's `exn` type), as well as assignable references.

In parallel, M. Fiore and his collaborators have developed the categorical semantics for several variations of second-order algebraic theories [7, 9, 5, 6]; of these, the simply sorted variants are equivalent to Harper's abstract binding trees augmented with a notion of second-order variable (metavariable).

The contribution of this paper is the development of the syntax and semantics of multi-sorted abts, an extension of second order universal algebra to support symbol-indexed families of operators. Additionally, we have developed the categorical semantics for abts formally in Constructive Type Theory using the Agda proof assistant [18].

1 Preliminaries

Fix a set $\mathcal{S}$ of *sorts*. We will say τ *sort* when $\tau \in \mathcal{S}$. A valence $\{\vec{\sigma}\}[\vec{\tau}].\,\tau$ specifies an expression of sort τ which binds symbols in $\vec{\sigma}$ and variables in $\vec{\tau}$.

$$\frac{\tau \textit{ sort} \quad \sigma_i \textit{ sort} \ \ (i \leqslant m) \quad \tau_i \textit{ sort} \ \ (i \leqslant n)}{\{\sigma_0, \ldots, \sigma_m\}[\tau_0, \ldots, \tau_n].\,\tau \textit{ valence}}$$

An arity $(\vec{\nu})\,\tau$ specifies an operator of sort τ with arguments of valences $\vec{\nu}$. We will call the set of valences $\mathcal{V}$, and the set of arities $\mathcal{A}$.

$$\frac{\tau\ \mathit{sort} \quad \nu_i\ \mathit{valence}\ \ (i \leqslant n)}{(\nu_0, \ldots, \nu_n)\ \tau\ \mathit{arity}}$$

1.1 Symbols, contexts and their sheaves

Let $\mathbb{I}$ be the category of finite cardinals and their injective maps; then the comma construction $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$, with $\mathcal{S}_{\equiv}$ the discrete category on the set $\mathcal{S}$, is the category of contexts of symbols whose objects are finite sets of symbols U and sort-assignments $U \xrightarrow{s} \mathcal{S}$, and whose morphisms are sort-preserving renamings; we will write Υ for a symbol context (U, s).

Now, a covariant presheaf X on $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$ is an intensional set which is subject to renamings $\Upsilon \xhookrightarrow{\varrho} \Upsilon'$; that is, each element $m \in X(\Upsilon)$ can be mapped to a unique element $m\varrho \in X(\Upsilon')$.

Definition 1.1 (Support)**.** For a presheaf $X : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}_{\equiv}}$, when $\Upsilon \xhookrightarrow{\varrho} \Upsilon'$, we say that Υ supports $m \in \Upsilon'$ (written $\Upsilon \blacktriangleright_\varrho m$) when, for all $\Upsilon' \xhookrightarrow{\varrho_1,\varrho_2} \Upsilon''$, if $\varrho_1 \circ \varrho = \varrho_2 \circ \varrho$ then $m\varrho_1 = m\varrho_2$.

Intuitively, when Υ supports $m \in X(\Upsilon')$, we would expect that we can work backward to a unique $m' : \Upsilon$ such that $m'\varrho = m$; whilst this is not in general the case for presheaves X, it is precisely the sheaf condition for covariant presheafs on $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$ under the atomic topology [15, p. 126]. The sheaf condition, then, ensures that the notion of support is well-behaved.

Let $\mathbf{supp}(m)$ be the *least support* of $m \in X(\Upsilon')$:

$$\mathbf{supp}(m) \triangleq \bigcap_{\Upsilon \xhookrightarrow{\varrho} \Upsilon'} \Upsilon \blacktriangleright_\varrho m$$

Intuitively, $\mathbf{supp}(m)$ is the exact symbol context that m depends on.

Remark 1.2. The presentation we have given above is related both to the Schanuel topos and, equivalently, Pitts' category of nominal sets [8]; we differ only in discussing covariant sheaves on the category of symbol contexts $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$, as opposed to covariant sheaves on finite sets of symbols $\mathbb{I}$ (i.e. unisorted contexts).

1.2 Operators and signatures

Fix a sheaf of operators $\mathcal{O} : \mathbf{Sh}((\mathbb{I} \downarrow \mathcal{S}_{\equiv})^{\mathrm{op}} \times \mathcal{A}_{\equiv})$ such that the arrows in $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$ lift to renamings of operators' parameters. Together, $\mathcal{S}$ and $\mathcal{O}$ form a *signature* $\Sigma \triangleq \langle \mathcal{S}, \mathcal{O} \rangle$.

By the Grothendieck construction $\oint \mathcal{O} \cong \mathbb{1} \downarrow \mathcal{O}$ we have a category of elements of $\mathcal{O}$ where objects are triples $\langle \langle \Upsilon, a \rangle, \vartheta \rangle$, interpreted as $\vartheta \in \mathcal{O} \langle \Upsilon, a \rangle$, and morphisms are

paired diagrams:

$$\langle\, \Upsilon, a \,\rangle \xrightarrow[\langle\, \rho, f \,\rangle]{} \langle\, \Upsilon', a' \,\rangle \qquad \begin{array}{ccc} & \mathbb{1} & \\ {\scriptstyle \vartheta}\swarrow & & \searrow{\scriptstyle \vartheta'} \\ \mathcal{O}\langle\, \Upsilon, a \,\rangle & \xrightarrow[\mathcal{O}\langle\, \rho, f \,\rangle]{} & \mathcal{O}\langle\, \Upsilon', a' \,\rangle \end{array}$$

We will write $\Upsilon \Vdash \vartheta : a$ in case $\vartheta \in \mathcal{O}\langle\, \Upsilon, a \,\rangle$ and note that this judgment enjoys the structural properties of weakening and exchange via functoriality of $\mathcal{O}$. Operators $\vartheta_0, \ldots, \vartheta_n$ are defined by specifying the fibers of $\pi_{\mathcal{O}} : \oint \mathcal{O} \longrightarrow \mathbb{I} \downarrow \mathcal{S}_{\equiv} \times \mathcal{A}_{\equiv}$ in which they reside, such that the glueing condition is satisfied:

$$\begin{aligned} {\pi_{\mathcal{O}}}^{-1}\langle\, \Upsilon, a \,\rangle \;&:\; \mathbb{I} \downarrow \mathcal{S}_{\equiv} \times \mathcal{A}_{\equiv} \longrightarrow \mathbf{Set} \\ {\pi_{\mathcal{O}}}^{-1}\langle\, \Upsilon, a \,\rangle \;&= \{\, \vartheta \in \langle\, \Upsilon, a \,\rangle \mid \pi_{\mathcal{O}}\langle\, \langle\, \Upsilon, a \,\rangle, \vartheta \,\rangle = \langle\, \Upsilon, a \,\rangle \,\} \end{aligned}$$

Examples For instance, consider the λ-calculus with a single sort, `exp`; we give its signature Σ_λ by asserting the following about its operators:

$$\begin{aligned} \Upsilon &\Vdash \mathtt{lam} : (\mathtt{[exp].\,exp})\,\mathtt{exp} \\ \Upsilon &\Vdash \mathtt{ap} : (\mathtt{.\,exp}, \mathtt{.\,exp})\,\mathtt{exp} \end{aligned}$$

These rules correspond to objects of $\oint \mathcal{O}_\lambda$ and we collect their fibers into sets of operators:

$$\begin{aligned} {\pi_{\mathcal{O}_\lambda}}^{-1}\langle\, \Upsilon, (\mathtt{[exp].\,exp})\,\mathtt{exp} \,\rangle \quad &= \quad \{\mathtt{lam}\} \\ {\pi_{\mathcal{O}_\lambda}}^{-1}\langle\, \Upsilon, (\mathtt{.\,exp}, \mathtt{.\,exp})\,\mathtt{exp} \,\rangle \quad &= \quad \{\mathtt{ap}\} \\ {\pi_{\Sigma_\lambda}}^{-1} \triangleq \bigcup_{\langle\, \Upsilon, a \,\rangle} {\pi_{\mathcal{O}_\lambda}}^{-1}\langle\, \Upsilon, a \,\rangle \quad &= \quad \{\mathtt{lam},\ \mathtt{ap}\} \end{aligned}$$

So far, we have made no use of symbols and parameters; however, consider the extension of the calculus with assignables (references):

$$\begin{aligned} \Upsilon &\Vdash \mathtt{decl} : (\mathtt{.\,exp}, \{\mathtt{exp}\}.\,\mathtt{exp})\,\mathtt{exp} \\ \Upsilon, u : \mathtt{exp} &\Vdash \mathtt{get}[u] : ()\,\mathtt{exp} \\ \Upsilon, u : \mathtt{exp} &\Vdash \mathtt{set}[u] : (\mathtt{.\,exp})\,\mathtt{exp} \end{aligned}$$

Declaring a new assignable consists in providing an initial value, and an expression binding a symbol (which shall represent the assignable in scope). Note that the functoriality of $\mathcal{O}$ guarantees for any renaming $\varrho : \Upsilon \hookrightarrow \Upsilon'$ and $f : a \longrightarrow a'$, a lifted map $\mathcal{O}\langle \varrho, f \rangle : \mathcal{O}\langle\, \Upsilon, a \,\rangle \hookrightarrow \mathcal{O}\langle\, \Upsilon', a' \,\rangle$ such that $\Upsilon' \Vdash \mathcal{O}\langle \varrho, f \rangle(\vartheta) : a'$ when $\Upsilon \Vdash \vartheta : a$. In particular, the renaming $\Upsilon, u \longmapsto \Upsilon, v$ shall take $\mathtt{get}[u]$ to $\mathtt{get}[v]$. Because the category $\mathcal{A}_{\equiv}$ is discrete and has only identity arrows, through an abuse of notation we will often write $\vartheta\varrho$ to mean $\mathcal{O}\langle \varrho, 1 \rangle(\vartheta)$.

2 Contexts

In general, we will have three kinds of context: symbolic (parameter) contexts, variable contexts, and metavaraible contexts The symbol contexts have already been defined via the comma construction $\mathbb{I} \downarrow \mathcal{S}_{\equiv}$, but they also admit a syntactic characterization,

$$\frac{}{\cdot \; sctx} \qquad \frac{\Upsilon \; sctx \quad \tau \; sort \quad u \notin |\Upsilon|}{\Upsilon, u : \tau \; sctx}$$

Because, modulo notation, we have $\Upsilon \in \mathbb{I} \downarrow \mathcal{S}_{\equiv}$ just when $\Upsilon \; sctx$, we will use the syntactic view when it is convenient.

Contexts of variables are similar to contexts of symbols, except that they admit *any* renamings, not just the injective ones. As such, when $\mathbb{F}$ is the category of finite cardinals and all functions between them, the comma construction $\mathbb{F} \downarrow \mathcal{S}_{\equiv}$ is the category of variable contexts. As above, we can give them an equivalent syntactic treatment:

$$\frac{}{\cdot \; vctx} \qquad \frac{\Gamma \; vctx \quad \tau \; sort \quad x \notin |\Gamma|}{\Gamma, x : \tau \; vctx}$$

A metavariable context consists of bindings of *valences* to metavariables; let $\mathcal{V} \triangleq \{\nu \mid \nu \; valence\}$ be the set of valences. Then, the category of metavariable contexts is the comma construction $\mathbb{F} \downarrow \mathcal{V}_{\equiv}$, which likewise admits an equivalent inductive definition:

$$\frac{}{\cdot \; mctx} \qquad \frac{\Theta \; mctx \quad \nu \; valence \quad m \notin |\Theta|}{\Theta, m : \nu \; mctx}$$

3 Abstract Binding Trees

Let the judgment $\Theta \triangleright \Upsilon \parallel \Gamma \vdash M : \tau$ presuppose $\Theta \; mctx$, $\Upsilon \; sctx$, $\Gamma \; vctx$ and $\tau \; sort$, meaning that M is an abstract binding tree of sort s, with metavariables in Θ, parameters in Υ, and variables in Γ. Let the judgment $\Theta \triangleright \Upsilon \parallel \Gamma \vdash E : \nu$ presuppose $\nu \; valence$. Then, the syntax of abstract binding trees is inductively defined in four rules:

$$\frac{\Gamma \ni x : \tau}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash x : \tau}\ \mathit{var}$$

$$\frac{\begin{array}{l}\Theta \ni m : \{\sigma_0, \ldots, \sigma_m\}[\tau_0, \ldots, \tau_n].\, \tau \\ \Upsilon \ni u_i : \sigma_i \quad (i \leqslant m) \\ \Theta \triangleright \Upsilon \parallel \Gamma \vdash M_i : \tau_i \quad (i \leqslant n)\end{array}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash m\{u_0, \ldots, u_m\}(M_0, \ldots, M_n) : \tau}\ \mathit{mvar}$$

$$\frac{\begin{array}{l}\Upsilon \Vdash \vartheta : (\nu_1, \ldots, \nu_n)\, \tau \\ \Theta \triangleright \Upsilon \parallel \Gamma \vdash E_i : \nu_i \quad (i \leqslant n)\end{array}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \vartheta(E_0, \ldots, E_n) : \tau}\ \mathit{app}$$

$$\frac{\Theta \triangleright \Upsilon, \vec{u} : \vec{\sigma} \parallel \Gamma, \vec{x} : \vec{\tau} \vdash M : \tau}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, M : \{\vec{\sigma}\}[\vec{\tau}].\, \tau}\ \mathit{abs}$$

Abts are identified up to α-equivalence.

3.1 Calculating free variables

We can easily calculate the variables free in an expression by recursion on its structure:

$$\frac{}{\mathbf{FV}\,(x) \rightsquigarrow \{x\}}\ \mathit{var}$$

$$\frac{\mathbf{FV}\,(M_i) \rightsquigarrow \vec{x}_i \quad (i \leqslant n)}{\mathbf{FV}\,(m\{\vec{u}\}(M_0, \ldots, M_n)) \rightsquigarrow \bigcup_{i \leqslant n} \vec{x}_i}\ \mathit{mvar}$$

$$\frac{\mathbf{FV}\,(E_i) \rightsquigarrow \vec{x}_i \quad (i \leqslant n)}{\mathbf{FV}\,(\vartheta(E_0, \ldots, E_n)) \rightsquigarrow \bigcup_{i \leqslant n} \vec{x}_i}\ \mathit{app}$$

$$\frac{\mathbf{FV}\,(M) \rightsquigarrow \vec{x}}{\mathbf{FV}\,(\mathbb{\lambda}\{\vec{u}\}[\vec{y}].\, M) \rightsquigarrow \vec{x} \setminus \vec{y}}\ \mathit{abs}$$

Because this is a total relation, henceforth we will write $\mathbf{FV}\,(M)$ for $\vec{x}$ when $\mathbf{FV}\,(M) \rightsquigarrow x$.

3.2 Calculating free symbols

Whereas the calculation of free variables pivoted on the *var* rule, the calculation of free symbols will pivot on the *app* rule, because the only place a symbol can be introduced is as a parameter to an operator.

$$\frac{}{\mathbf{FS}\,(x) \rightsquigarrow \{\,\}}\ \mathit{var}$$

$$\frac{\mathbf{FS}\,(M_i) \rightsquigarrow \vec{u}_i \quad (i \leqslant n)}{\mathbf{FS}\,(m\{\vec{u}\}(M_0, \ldots, M_n)) \rightsquigarrow \bigcup_{i \leqslant n} \vec{u}_i}\ \mathit{mvar}$$

$$\frac{\mathbf{FS}\,(E_i) \rightsquigarrow \vec{u}_i \quad (i \leqslant n)}{\mathbf{FS}\,(\vartheta(E_0, \ldots, E_n)) \rightsquigarrow |\mathbf{supp}(\vartheta)| \cup \bigcup_{i \leqslant n} \vec{u}_i}\ \mathit{app}$$

$$\frac{\mathbf{FS}\,(M) \rightsquigarrow \vec{u}}{\mathbf{FS}\,(\mathbb{\lambda}\{\vec{v}\}[\vec{x}].\, M) \rightsquigarrow \vec{u} \setminus \vec{v}}\ \mathit{abs}$$

Because this is a total relation, henceforth we will write $\mathbf{FS}\,(M)$ for $\vec{u}$ when $\mathbf{FS}\,(M) \rightsquigarrow u$.

3.3 Renaming of symbols

The only place that symbols appear in our calculus is as parameters to operators (unlike variables, symbols are not terms). Therefore, the functorial action of the operator sheaf can be lifted into terms by recursion on their structure, via a pair of judgments $M\{\varrho\} \rightsquigarrow N$ and $E\{\varrho\} \rightsquigarrow F$, presupposing $\varrho : \Upsilon \hookrightarrow \Upsilon'$, and $\Theta \triangleright \Upsilon \parallel \Gamma \vdash M : \tau$ and $\Theta \triangleright \Upsilon \parallel \Gamma \vdash E : \nu$ respectively:

$$\frac{}{x\{\varrho\} \rightsquigarrow x}$$

$$\frac{\varrho(\vec{u}) \equiv \vec{v} \quad M_i\{\varrho\} \rightsquigarrow N_i \quad (i \leqslant n)}{m\{\vec{u}\}(M_0, \ldots, M_n)\{\varrho\} \rightsquigarrow m\{\vec{v}\}(N_0, \ldots, N_n)}$$

$$\frac{E_i\{\varrho\} \rightsquigarrow F_i \quad (i \leqslant n)}{\vartheta(E_0, \ldots, E_n)\{\varrho\} \rightsquigarrow \vartheta\varrho(F_0, \ldots, F_n)}$$

$$\frac{M\{\varrho \setminus \vec{u}\} \rightsquigarrow N}{\mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, M\{\varrho\} \rightsquigarrow \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, N}$$

Above, the notation $\varrho \setminus \vec{u}$ means the omission of the variables $\vec{u}$ from the renaming ϱ. Because terms are identified up to α-equivalence, the renaming judgment is functional in its input, and so we are justified in writing $M\{\varrho\}$ for N when $M\{\varrho\} \rightsquigarrow N$.

3.4 Substitution of variables

Variable substitution in abts is defined inductively by a pair of judgments, $[N \,/\, x]\, M \rightsquigarrow M'$ and $[N \,/\, x]\, E \rightsquigarrow F$:

$$\frac{x = y}{[N \,/\, x]\, y \rightsquigarrow N} \qquad \frac{x \# y}{[N \,/\, x]\, y \rightsquigarrow y}$$

$$\frac{[N \,/\, x]\, M_i \rightsquigarrow M_i' \quad (i \leqslant n)}{[N \,/\, x]\, m\{\vec{u}\}(M_0, \ldots, M_n) \rightsquigarrow m\{\vec{u}\}(M_0', \ldots, M_n')}$$

$$\frac{[N \,/\, x]\, E_i \rightsquigarrow F_i \quad (i \leqslant n)}{[N \,/\, x]\, \vartheta(E_0, \ldots, E_n) \rightsquigarrow \vartheta(F_0, \ldots, F_n)}$$

$$\frac{x \notin \vec{y} \quad \vec{u} \# \mathbf{FS}(N) \quad \vec{y} \# \mathbf{FV}(N) \quad [N \,/\, x]\, M \rightsquigarrow M'}{[N \,/\, x]\, \mathbb{\lambda}\{\vec{u}\}[\vec{y}].\, M \rightsquigarrow \mathbb{\lambda}\{\vec{u}\}[\vec{y}].\, M'} \qquad \frac{x \in \vec{y} \quad \vec{u} \# \mathbf{FS}(N) \quad \vec{y} \# \mathbf{FV}(N)}{[N \,/\, x]\, \mathbb{\lambda}\{\vec{u}\}[\vec{y}].\, M \rightsquigarrow \mathbb{\lambda}\{\vec{u}\}[\vec{y}].\, M}$$

Going forward, we will write $[N \,/\, x]\, M$ for M' when $[N \,/\, x]\, M \rightsquigarrow M'$, and $\left[\vec{N} \,/\, \vec{x}\right] M$ for the simultaneous substitution of $\vec{N}$ for $\vec{x}$ in M.

3.5 Substitution of metavariables

Metavariables are substituted by bound terms; since a metavariable may only appear in an application expression $m\{\cdots\}(\cdots)$, we will instantiate the bound term at the supplied parameters and arguments. Substitution for metavariables is defined inductively by the judgments $[E \,/\, m]\, M \rightsquigarrow N$ and $[E \,/\, m]\, F \rightsquigarrow F'$:

$$\frac{}{[E \,/\, m]\, x \rightsquigarrow x}$$

$$\frac{m \# n \quad [E \,/\, n]\, M_i \rightsquigarrow N_i \quad (i \leqslant n)}{[E \,/\, m]\, n\{\vec{u}\}(M_0, \ldots, M_n) \rightsquigarrow n\{\vec{u}\}(N_0, \ldots, N_n)}$$

$$\frac{m = n \quad \left[\vec{M} \,/\, \vec{x}\right] N \rightsquigarrow N' \quad N'\{\vec{u} \longmapsto \vec{v}\} \rightsquigarrow N''}{[\mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, N \,/\, m]\, n\{\vec{v}\}(\vec{M}) \rightsquigarrow N''}$$

$$\frac{[E \,/\, m]\, F_i \rightsquigarrow F_i' \quad (i \leqslant n)}{[E \,/\, m]\, \vartheta(F_0, \ldots, F_n) \rightsquigarrow \vartheta(F_0', \ldots, F_n')}$$

$$\frac{\vec{u} \# \mathbf{FS}(E) \quad \vec{x} \# \mathbf{FV}(E) \quad [E \,/\, m]\, M \rightsquigarrow N}{[E \,/\, m]\, \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, M \rightsquigarrow \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, N}$$

As usual, we will write $[E \,/\, m]\, M$ for N when $[E \,/\, m]\, M \rightsquigarrow N$.

4 Model Theory

Let $\mathbf{H} \triangleq (\mathbb{I} \downarrow \mathcal{S}_{\equiv} \times \mathbb{F} \downarrow \mathcal{S}_{\equiv})^{\mathrm{op}}$; then we fix the functor category $\widehat{\mathbf{H}}^{\mathcal{S}}$ as our semantic universe.

Let $V_\tau(\Upsilon \parallel \Gamma) \triangleq \{x \in |\Gamma| \mid \Gamma(x) = \tau\}$ be called the presheaf of variables; additionally, we have a presheaf of symbols $S_\tau(\Upsilon \parallel \Gamma) \triangleq \{u \in |\Upsilon| \mid \Upsilon(x) = \tau\}$. Lastly, we have the presheaf of operators with arity a, $\mathcal{O}_a(\Upsilon \parallel \Gamma) \triangleq \mathcal{O}\langle \Upsilon, a \rangle$

4.1 Substitution monoidal structures

For an object $P : \widehat{\mathbf{H}}^{\mathcal{S}}$, we will use the notation $P^{[\Gamma]}$ to mean $\prod_{x \in |\Gamma|} P_{\Gamma(x)}$; likewise, $\mathbf{S}^{\{\Upsilon\}}$ shall mean $\prod_{u \in |\Upsilon|} S_{\Upsilon(u)}$. For a presheaf $A : \widehat{\mathbf{H}}$ and a sort-indexed family of presheaves $P : \widehat{\mathbf{H}}^{\mathcal{S}}$, we have an operation $A \bullet P$, defined as a coend in the following way:

$$(A \bullet P)(\Upsilon \parallel \Gamma) \triangleq \int^{(\Upsilon' \parallel \Delta) \in \mathbf{H}} A(\Upsilon' \parallel \Delta) \times \mathbf{S}^{\{\Upsilon'\}}(\Upsilon \parallel \Gamma) \times P^{[\Delta]}(\Upsilon \parallel \Gamma)$$

Using this, we can define a tensor $P \odot Q$ for $P, Q : \widehat{\mathbf{H}}^{\mathcal{S}}$ as follows:

$$(P \odot Q)_\tau \triangleq P_\tau \bullet Q \qquad (\tau \in \mathcal{S})$$

Then, V is the unit to this tensor. We will say that an object $P : \widehat{\mathbf{H}}^{\mathcal{S}}$ is a Σ-monoid in case it is equipped with the following natural transformations where ν embeds variables into P and ς equips P with an operation for simultaneous substitutions of variables. Furthermore, ν and ς induce maps ν_Γ and $\varsigma^\tau_{\Upsilon \parallel \Gamma}$:

$$V \xrightarrow{\ \nu\ } P \xleftarrow{\ \varsigma\ } P \odot P$$

$$V^{[\Gamma]} \xrightarrow{\ \nu_\Gamma\ } P^{[\Gamma]} \qquad P_\tau^{\mathbf{y}(\Upsilon \parallel \Gamma)} \times \mathbf{S}^{\{\Upsilon\}} \times P^{[\Gamma]} \xrightarrow{\ \varsigma^\tau_{\Upsilon \parallel \Gamma}\ } P_\tau$$

4.2 The signature endofunctor and its initial algebras

For each signature $\Sigma \equiv \langle \mathcal{S}, \mathcal{O} \rangle$, we have an endofunctor $\mathcal{F}_\Sigma : \widehat{\mathbf{H}}^{\mathcal{S}} \longrightarrow \widehat{\mathbf{H}}^{\mathcal{S}}$, which is defined as follows:

$$\mathcal{F}_\Sigma(X)_\tau \triangleq \coprod_{\vartheta \in \mathcal{O}_{(\vec{\nu})\tau}} \prod_{\{\vec{\sigma}\}[\vec{\tau}].\, \tau_i \in \vec{\nu}} X_{\tau_i}^{\mathbf{y}(\vec{\sigma} \parallel \vec{\tau})}$$

Then, a Σ-model is a Σ-monoid P which is equipped with an initial algebra $\alpha : \mathcal{F}_\Sigma(P) \longrightarrow P$, which shall interpret applications of each operator.

4.3 Interpretation of terms

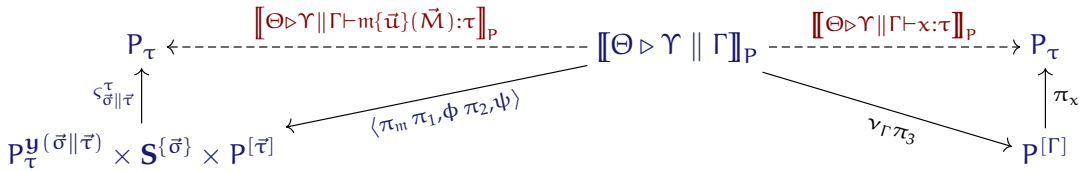
The metavariable, symbol and variable contexts are interpreted in a model P as an environment presheaf in the following way:

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_{\mathsf{P}} \triangleq \left(\prod_{(m:\{\vec{\sigma}\}[\vec{\tau}].\,\tau)\in\Theta} \mathsf{P}_\tau{}^{\mathsf{y}(\vec{\sigma}\parallel\vec{\tau})} \right) \times \mathbf{S}^{\{\Upsilon\}} \times \mathbf{V}^{[\Gamma]}$$

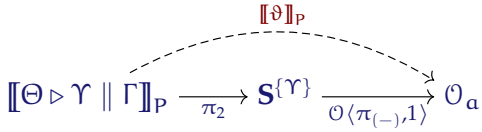
Then, the interpretation of a term in a model P is a map from its environment to P:

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma \vdash M : \tau]\!]_{\mathsf{P}} : [\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_{\mathsf{P}} \longrightarrow \mathsf{P}_\tau$$

Variables are interpreted by the map $[\![\Theta \triangleright \Upsilon \parallel \Gamma \vdash x : \tau]\!]_{\mathsf{P}}$ which projects them from the environment and embeds them into the model. Metavariables are resolved by the map $\left[\!\left[\Theta \triangleright \Upsilon \parallel \Gamma \vdash m\{\vec{u}\}(\vec{M}) : \tau\right]\!\right]_{\mathsf{P}}$ (where $\Theta \ni m : \{\vec{\sigma}\}[\vec{\tau}].\,\tau$) which projects their interpretation from the environment and instantiates it via substitution:



Interpretation of operator applications is the most complicated. Recall that, unlike in standard treatments of universal algebra, our operators are indexed by symbol collections; therefore, operators must pass through suitable renamings in order to be used in the interpretation. Let us begin by constructing for each operator $\Upsilon \Vdash \vartheta : a$ the morphism $[\![\vartheta]\!]_{\mathsf{P}}$ which shall rename the parameters of the operator using the environment:



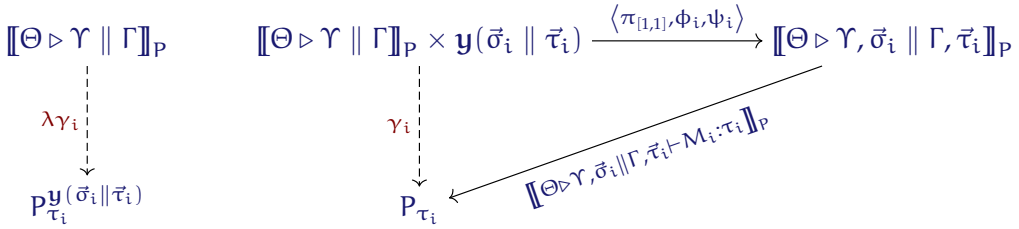
We will proceed using the initial $\mathcal{F}_\Sigma$-algebra α, as follows, by postcomposing it with a morphism β from the environment into the signature endofunctor, which interprets the syntax of operator applications:

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_{\mathsf{P}} \xrightarrow{\beta} \mathcal{F}_\Sigma(\mathsf{P})_\tau \xrightarrow{\alpha_\tau} \mathsf{P}_\tau$$

The construction of β proceeds by renaming the parameters of the operator ϑ and constructing the (bound) exponentiated arguments γ of the operator.

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_P \xrightarrow{\beta \triangleq \langle [\![\vartheta]\!]_P, \lambda\gamma \rangle} \coprod_{\vartheta \in \mathcal{O}_{(\vec{\nu})\tau}} \prod_{\{\vec{\sigma}_i\}[\vec{\tau}_i].\, \tau_i \in \vec{\nu}} P^{\mathbf{y}(\vec{\sigma}_i \parallel \vec{\tau}_i)}_{\tau_i}$$

Arguments $\lambda\gamma_i$ are the exponential transposes (curried form) of the composites γ_i:



where ϕ_i, ψ_i are defined as follows:

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_P \times \mathbf{y}(\vec{\sigma}_i \parallel \vec{\tau}_i) \xrightarrow{\phi_i \triangleq \left\langle \langle \pi_u \circ \pi_1 \rangle_{u \in |\Upsilon|}, \langle \pi_u \circ \pi_{[2,1]} \rangle_{u \in |\vec{\sigma}_i|} \right\rangle} \mathbf{S}^{\{\Upsilon, \vec{\sigma}_i\}}$$

$$[\![\Theta \triangleright \Upsilon \parallel \Gamma]\!]_P \times \mathbf{y}(\vec{\sigma}_i \parallel \vec{\tau}_i) \xrightarrow{\psi_i \triangleq \left\langle \langle \pi_x \circ \pi_1 \rangle_{x \in |\Gamma|}, \langle \pi_x \circ \pi_{[2,1]} \rangle_{x \in |\vec{\tau}_i|} \right\rangle} \mathbf{V}^{[\Gamma, \vec{\tau}_i]}$$

This concludes the interpretation of well-sorted terms into any Σ-model.

5 Case Study: Wellformed Sequents

The representation of telescopes and sequents in a logical framework is notoriously difficult; whilst it is possible to use higher-order abstract syntax or abts to encode the binding-structure of telescopes and sequents, the encoding is sufficiently laborious and obscure that it is not used in practice.

Crary has demonstrated a first-order encoding of contexts in the logical framework in bijection with actual LF-contexts [4], which has been successfully used in large-scale mechanization efforts, including that of Standard ML [14] and the Edinburgh Logical Framework itself [16].

talk about context encodings in Abella

We will approach the problem of encoding telescopes and sequents from the *refinements* perspective, where a conservative approximation of the grammar is first given using the abt logical framework, and then the correctness of a code is expressed separately in a judgment that refines the existing specification.

Because we have not committed to using the built-in binding machinery to express the well-scopedness of telescopes and sequents, we are free to use *symbols* in order to

model the variables in the context. This is in fact quite sensible if we are actually trying to faithfully represent the syntax of telescopes and sequents, rather than replace them with their counterparts on the meta-level.

This insight leads the way to a simple abt signature for the theory of telescopes and sequents.

$$\begin{array}{l}
\texttt{tele}\ \textit{sort} \\
\texttt{exp}\ \textit{sort} \\
\texttt{type}\ \textit{sort} \\
\texttt{jdg}\ \textit{sort}
\end{array}$$

$$\Upsilon, u : \texttt{exp} \Vdash \texttt{var}[u] : ()\ \texttt{exp}$$

$$\begin{array}{l}
\Upsilon \Vdash \texttt{nil} : ()\ \texttt{tele} \\
\Upsilon, u : \texttt{exp} \Vdash \texttt{snoc}[u] : (.\ \texttt{tele}, .\ \texttt{type})\ \texttt{tele}
\end{array}$$

$$\Upsilon \Vdash \texttt{sequent} : (.\ \texttt{tele}, .\ \texttt{type})\ \texttt{jdg}$$

Suppose we have encoded a fragment of type theory as well:

$$\begin{array}{l}
\Upsilon \Vdash \top : ()\ \texttt{type} \\
\Upsilon \Vdash \bot : ()\ \texttt{type} \\
\Upsilon \Vdash \texttt{bool} : ()\ \texttt{type} \\
\Upsilon \Vdash \texttt{isTrue} : (.\ \texttt{exp})\ \texttt{type} \\
\Upsilon \Vdash \texttt{pi} : (.\ \texttt{type}, [\texttt{exp}].\ \texttt{type})\ \texttt{type} \\
\Upsilon \Vdash \texttt{sg} : (.\ \texttt{type}, [\texttt{exp}].\ \texttt{type})\ \texttt{type}
\end{array}$$

Terms written using the abstract syntax will be difficult to read, so let us define some notation:

$$\begin{array}{rcl}
\diamond & \triangleq & \texttt{nil} \\
H, u : P & \triangleq & \texttt{snoc}[u](H, P) \\
H \gg A & \triangleq & \texttt{sequent}(H, A) \\
`u & \triangleq & \texttt{var}[u]
\end{array}$$

Now, we have the following well-formed sequent:

$$\cdot \triangleright u : \texttt{exp}, v : \texttt{exp} \parallel \cdot \vdash \diamond, u : \texttt{bool}, v : \texttt{isTrue}(`u) \gg \texttt{isTrue}(`u) : \texttt{jdg}$$

The above sequent has free symbols, but we can close over them by adding a form of parametric higher-order judgment to our object language, indexed by a collection of sorts $\vec{\sigma}$:

$$\Upsilon \Vdash \nabla[\vec{\sigma}] : (\{\vec{\sigma}\}.\ \texttt{jdg})\ \texttt{jdg}$$

Then, we may write a closed sequent judgment as follows:

$$\cdot \triangleright \cdot \parallel \cdot \vdash \nabla[\texttt{exp}, \texttt{exp}](\mathbb{\lambda}\{u, v\}[].\ \diamond, u : \texttt{bool}, v : \texttt{isTrue}(`u) \gg \texttt{isTrue}(`u)) : \texttt{jdg}$$

5.1 Refinements for wellformedness

Having specified an approximation of the grammar of telescopes and sequents in the abt logical framework, we can proceed to define proper wellformedness via *inductive refinement* [11]. The basic idea is to introduce a new form of (meta)-judgment $\Theta \triangleright \Upsilon \parallel \Gamma \vdash M \in_{\mathbf{wf}} \tau$ which expresses the extrinsic wellformedness properties we wish to verify, presupposing $\Theta \triangleright \Upsilon \parallel \Gamma \vdash M : \tau$. Additionally, we introduce an analogous judgment on bound terms, $\Theta \triangleright \Upsilon \parallel \Gamma \vdash E \in_{\mathbf{wf}} \nu$ presupposing $\Theta \triangleright \Upsilon \parallel \Gamma \vdash E : \nu$, defined uniformly as follows:

$$\frac{\Theta \triangleright \Upsilon, \vec{u} : \vec{\sigma} \parallel \Gamma, \vec{x} : \vec{\tau} \vdash M \in_{\mathbf{wf}} \tau}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, M \in_{\mathbf{wf}} \{\vec{\sigma}\}[\vec{\tau}].\, \tau}$$

Likewise, wellformedness for variables and metavariables is defined uniformly:

$$\frac{}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash x \in_{\mathbf{wf}} \tau} \qquad \frac{\begin{array}{c}\Theta \ni m : \{\vec{\sigma}\}[\tau_0, \ldots, \tau_n].\, \tau \\ \Theta \triangleright \Upsilon \parallel \Gamma \vdash M_i \in_{\mathbf{wf}} \tau_i \quad (i \leqslant n)\end{array}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash m\{\vec{u}\}(M_0, \ldots, M_n) \in_{\mathbf{wf}} \tau}$$

Remark 5.1. Note that the refinement for variables x is not trivial, since it is only defined in case the presupposition $\Theta \triangleright \Upsilon \parallel \Gamma \vdash x : \tau$ is satisfied.

The remainder of the definition of refinement proceeds by induction on sorts and operators. For the sake of this example, we will just stipulate that anything of sort `exp` or `type` is grammatical if its subterms are grammatical:

$$\frac{\begin{array}{c}\Upsilon \Vdash \vartheta : (\nu_0, \ldots, \nu_n)\, \tau \\ \Theta \triangleright \Upsilon \parallel \Gamma \vdash E_i \in_{\mathbf{wf}} \nu_i \quad (i \leqslant n)\end{array}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \vartheta(E_0, \ldots, E_n) \in_{\mathbf{wf}} \tau} \text{ for } \tau \in \{\mathtt{exp}, \mathtt{type}\}$$

The refinements for parametric judgment and sequents simply delegate to their subterms as well:

$$\frac{\Theta \triangleright \Upsilon, \vec{u} : \vec{\sigma} \parallel \Gamma \vdash J \in_{\mathbf{wf}} \mathtt{jdg}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \nabla[\vec{\sigma}](\mathbb{\lambda}\{\vec{u}\}[].\, J) \in_{\mathbf{wf}} \mathtt{jdg}}$$

$$\frac{\Theta \triangleright \Upsilon \parallel \Gamma \vdash H \in_{\mathbf{wf}} \mathtt{tele} \quad \Theta \triangleright \Upsilon \parallel \Gamma \vdash A \in_{\mathbf{wf}} \mathtt{type}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash H \gg A \in_{\mathbf{wf}} \mathtt{jdg}}$$

The refinement for telescopes proceeds by induction:

$$\frac{}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash \diamond \in_{\mathbf{wf}} \mathtt{tele}} \qquad \frac{\begin{array}{c}\Theta \triangleright \Upsilon \setminus \{u\} \parallel \Gamma \vdash H \in_{\mathbf{wf}} \mathtt{tele} \\ \Theta \triangleright \Upsilon \setminus \{u\} \parallel \Gamma \vdash A \in_{\mathbf{wf}} \mathtt{type}\end{array}}{\Theta \triangleright \Upsilon \parallel \Gamma \vdash H, u : A \in_{\mathbf{wf}} \mathtt{tele}}$$

References

[1] P. Aczel. A general Church-Rosser theorem. Technical report, University of Manchester, 1978.

[2] T. Altenkirch, J. Chapman, and T. Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1:3):1–40, 2015.

[3] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[4] K. Crary. Explicit contexts in LF (extended abstract). *Electronic Notes in Theoretical Computer Science*, 228:53 – 68, 2009. Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008).

[5] M. Fiore and C.-K. Hur. Second-order equational logic (extended abstract). In A. Dawar and H. Veith, editors, *Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer Berlin Heidelberg, 2010.

[6] M. Fiore and O. Mamoud. Second-order algebraic theories – (extended abstract). In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 368–380, 2010.

[7] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of the 14th Symposium on Logic in Computer Science*, pages 193–202, 1999.

[8] M. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, Apr. 2006.

[9] M. P. Fiore. Mathematical models of computational and combinatorial structures. In *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 25–46, 2005.

[10] M. Hamana. Free £-monoids: A higher-order syntax with metavariables. In W.-N. Chin, editor, *Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 348–363. Springer Berlin Heidelberg, 2004.

[11] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2016.

[12] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.

[13] J. Hickey, A. Nogin, R. L. Constable, B. E. Aydemir, E. Barzilay, Y. Bryukhov, R. Eaton, A. Granicz, A. Kopylov, C. Kreitz, V. N. Krupski, L. Lorigo, S. Schmitt, C. Witty, and X. Yu. MetaPRL – a modular logical environment. In D. Basin and B. Wolff, editors, *Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 287–303. Springer Berlin Heidelberg, 2003.

[14] D. K. Lee, K. Crary, and R. Harper. Towards a mechanized metatheory of Standard ML. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 173–184, New York, NY, USA, 2007. ACM.

[15] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic : a first introduction to topos theory*. Universitext. Springer, New York, 1992.

[16] C. Martens and K. Crary. LF in LF: Mechanizing the metatheories of LF in Twelf. In *Proceedings of the Seventh International Workshop on Logical Frameworks and Meta-languages, Theory and Practice*, LFMTP '12, pages 23–32, New York, NY, USA, 2012. ACM.

[17] P. Martin-Löf and G. Sambin. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, Napoli, 1984.

[18] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[19] J. Sterling, D. Gratzer, and V. Rahli. JonPRL. `http://www.jonprl.org/`, 2015.