Frex^2: Normalisation by Evaluation for Second-Order Algebras

Greg Brown

February 14, 2023

1 Second-Order Algebra

Universal second-order algebra is a technique to describe syntax and theories for algebras with operations that can bind variables. One notable example of a second-order algebra is the simply-typed lambda calculus (STLC). Second-order algebras depend on a set T of types. For instance, the grammar $T \coloneqq N \mid T \rightarrowtail T$ gives the types for the STLC, where N describes some neutral base type (e.g. natural numbers).

A *signature* is a pair of a set O and function $|\cdot| \in O \to (T^* \times T)^* \times T$. The set O is the set of operators in the signature. The function gives the arity and return type of each operator. Each operator can take a number of arguments, hence the arity is a list of descriptors. Because arguments can bind variables, each descriptor has a list of bound variable types, as well as the argument type. Let the notation $(\gamma_i)\sigma_i \vdash o : \tau$ denote $|o| = (\gamma_i, \sigma_i)_i, \tau$.

The signature for the STLC has the two operators $\mathtt{app}_{\alpha,\beta}$ and $\mathtt{abs}_{\alpha,\beta}$ for each pair of types α and β. Their arities are $\alpha \rightarrowtail \beta, \alpha \vdash \mathtt{app}_{\alpha,\beta} : \beta$ and $(\alpha)\beta \vdash \mathtt{abs}_{\alpha,\beta} : \alpha \rightarrowtail \beta$. Application (`app`) takes two arguments—a function and value—and returns a result. Neither argument to application binds variables. Abstraction (`abs`) takes a single argument, whose type is the output type of the function. This argument has access to a freshly-bound variable of the input type.

The unpairing operation in STLC with products is also a second-order operation. The operation `let (x, y) = e in e'` is represented by an operator $\mathtt{unpair}_{\alpha,\beta,\tau}$, with $\alpha \times \beta, (\alpha, \beta)\tau \vdash \mathtt{unpair}_{\alpha,\beta,\tau} : \tau$. The argument `e` is a pair and has no additional variables in scope. The second argument `e'` has access to two new variables, corresponding to the two components of the pair `x` and `y`[1].

[1]Think of a good non-example.

Define a *sorted family* to be a type-and-context-indexed set. For example, the family of variables $\mathcal{I}\,\tau\,\Gamma$ is the set of positions of type τ in the context Γ. Write $\Gamma \vdash^{\mathcal{X}} \mathtt{t} : \tau$ for $\mathtt{t} \in \mathcal{X}\,\tau\,\Gamma$, for any sorted family $\mathcal{X}$.

Given a signature $\Sigma = (O, |\cdot|)$, you can define an *algebra*. Algebras have four constituent parts:

- a carrier sorted family $\mathcal{A}$
- an operation $[\![\cdot]\!]_o \in (\mathcal{A}\,\sigma_i\,(\gamma_i, \Gamma))_i \to \mathcal{A}\,\tau\,\Gamma$ for each operator $(\gamma_i)\sigma_i \vdash o : \tau$ and context $\Gamma \in T^*$
- a mapping $\mathtt{var} \in \mathcal{I}\,\tau\,\Gamma \to \mathcal{A}\,\tau\,\Gamma$
- a mapping $\mathtt{sub} \in \mathcal{A}\,\tau\,\Gamma \to (\forall\sigma.\mathcal{I}\,\sigma\,\Gamma \to \mathcal{A}\,\sigma\,\Delta) \to \mathcal{A}\,\tau\,\Delta$

subject to the following conditions:

left unit: $\mathtt{sub}(\mathtt{x}_i, \sigma) = \sigma_i$

right unit: $\mathtt{sub}(t, \mathtt{var}) = t$

associativity: $\mathtt{sub}(\mathtt{sub}(t, \sigma), \varsigma) = \mathtt{sub}(t, i \mapsto \mathtt{sub}(\sigma_i, \varsigma))$

naturality: $\mathtt{sub}([\![ts]\!]_o, \sigma) = [\![(\mathtt{sub}(ts_i, \Uparrow \sigma))_i]\!]_o$

where $\Uparrow \sigma$ uses $\mathtt{var}$ and $\mathtt{sub}$ to lift a substitution σ from $\mathcal{I}\,\tau\,\Gamma \to \mathcal{A}\,\tau\,\Delta$ to $\mathcal{I}, \tau\,\Theta, \Gamma \to \mathcal{A}, \tau\,\Theta, \Delta$.

The associativity, left and right unit conditions assert that $\mathcal{A}$ is a *substitution monoid*. This means that substitution interacts with itself and variables in an intuitive way. The naturality condition asserts that substitution passes through operators and is capture-avoiding. Substitution lifting maps freshly-bound variables to themselves.

For the STLC, an algebra $\mathcal{A}$ has a carrier $\mathcal{A}$; a pair of operations $[\![\cdot]\!]_{\mathtt{app}} \in \mathcal{A}\,(\alpha \rightarrowtail \beta)\,\Gamma \to \mathcal{A}\,\alpha\,\Gamma \to \mathcal{A}\,\beta\,\Gamma$, and $[\![\cdot]\!]_{\mathtt{abs}} \in \mathcal{A}\,\beta\,(\alpha + \Gamma) \to \mathcal{A}\,(\alpha \rightarrowtail \beta)\,\Gamma$; and the variable and substitution maps.

One example algebra for any signature is the unit algebra $\mathbf{1}$, where $\mathbf{1}\,\tau\,\Gamma = \{*\}$. Each operation $[\![\cdot]\!]_o$, and the maps $\mathtt{var}$ and $\mathtt{sub}$, return the unique value of the appropriate type. This trivially satisfies the conditions.

Another example is the algebra of sets and functions for the STLC. First, take any set V to represent values of type N. A type T can then be interpreted as a set of values $\mathcal{V}[T]$:

$$\begin{aligned} \mathcal{V}[N] &\mapsto V \\ \mathcal{V}[\alpha \rightarrowtail \beta] &\mapsto \mathcal{V}[\alpha] \to \mathcal{V}[\beta] \end{aligned}$$

We can take the carrier sorted family $\mathcal{A}\,\tau\,\Gamma = (\forall\sigma.\mathcal{I}\,\sigma\,\Gamma \to \mathcal{V}[\sigma]) \to \mathcal{V}[\tau]$. From here, defining the operations and mappings is straight forward:

$$\begin{aligned}
[\![f, g]\!]_{\mathtt{app}}(\gamma) &= f(\gamma, (g(\gamma))) \\
[\![f]\!]_{\mathtt{abs}}(\gamma) &= x \mapsto f(\mathtt{here} \mapsto x; \mathtt{there}\, v \mapsto \gamma\, v) \\
\mathtt{var}(i, \gamma) &= \gamma_i \\
\mathtt{sub}(f, \sigma, \gamma) &= f\,(i \mapsto \sigma_i(\gamma))
\end{aligned}$$

Verifying that this definition satisfies the coherence conditions is left as an exercise[2].

2 Metavariables

Here is a typical way of writing the β-reduction relation for STLC:

```
(λ x. t) $ u ⤳ t[x ↦ u]
```

The variable `t` is implicitly parameterised by the bound variable `x`, meaning `x` $:\alpha + \Gamma \vdash$ `t` $:\beta$. In contrast, `u` is not parameterised by any terms. We can make the parameters explicit:

```
(λ x. t<x>) $ u<> ⤳ t<u<>>
```

`t` and `u` are *metavariables*—free variables with parameters. We can instantiate metavariables with concrete terms to create new ones. This property is formalised in the definition of a *syntactic algebra*.

A syntactic algebra for a signature Σ over a sorted family of metavariables $\mathfrak{X}$ is:

- an algebra $\mathcal{A}$ over the signature
- a map $\mathtt{mvar} \in \mathfrak{X}\,\tau\,\Pi \to (\forall\sigma.\mathcal{I}\,\sigma\,\Pi \to \mathcal{A}\,\sigma\,\Gamma) \to \mathcal{A}\,\tau\,\Gamma$
- and a map $\mathtt{msub} \in \mathcal{A}\,\tau\,\Gamma \to (\forall\sigma, \Theta.\mathfrak{X}\,\sigma\,\Theta \to \mathcal{A}\,\sigma\,(\Theta, \Delta)) \to \mathcal{A}\,\tau\,(\Gamma, \Delta)$

which satisfy these conditions[3]:

1. $\mathtt{msub}(\mathtt{var}(i), \zeta) = \mathtt{var}(\mathtt{inl}(i))$
2. $\mathtt{msub}(\mathtt{sub}(t, \sigma), \zeta) = \mathtt{sub}(\mathtt{msub}(t, \zeta), (\mathtt{inl} \circ \sigma) + \mathtt{var})$
3. $\mathtt{msub}([\![ts]\!]_o, \zeta) = [\![\mathtt{msub}(ts_i, \zeta)]\!]_o$
4. $\mathtt{msub}(\mathtt{mvar}(\mathfrak{m}, \sigma), \varsigma) = \mathtt{sub}(\zeta_{\mathfrak{m}}, (i \mapsto \mathtt{msub}(\sigma_i, \zeta)) + \mathtt{var})$
5. $\mathtt{sub}(\mathtt{mvar}(\mathfrak{m}, \sigma), \varsigma) = \mathtt{mvar}(\mathfrak{m}, i \mapsto \mathtt{sub}(\sigma_i, \varsigma))$

[2] This is me being lazy.

[3] Explain why these conditions exist, and why they are so ugly.

6. $\texttt{sub}(\texttt{msub}(t, \zeta), \texttt{var} + (\texttt{inr} \circ \sigma)) = \texttt{msub}(t, \mathfrak{m} \mapsto \texttt{sub}(\zeta_{\mathfrak{m}}, \texttt{var} + (\texttt{inr} \circ \sigma)))$

The map `mvar` takes a metavariable $\Pi \vdash^{\mathfrak{X}} \mathfrak{m} : \tau$ and converts it into a term in context Γ. This is by giving a value for each of $\mathfrak{m}$'s parameters, achieved via the substitution argument.

The syntactic substitution map `msub` is more complex. First note that it is linear: the output context is an extension of the input. This means that metasubstitution can use bound variables, as long as they were bound in a higher scope. Also observe that the substitution map (the second parameter) is independant of the input context Γ. Because metavariables are always associated with a substitution, it is sufficient to map only the metavariables in their original contexts. The context can be "corrected" by a substitution.

Note that the term algebra $\mathbb{T}[\mathfrak{X}]$ for a signature is the free syntactic algebra over $\mathfrak{X}$, with for each map $f \in \mathfrak{X}\, \tau\, \Gamma \to \mathcal{A}\, \tau\, \Gamma$ a unique homomorphism $\texttt{bind}(f) \in \mathbb{T}[\mathfrak{X}] \to \mathcal{A}$.

Return to the earlier example of β reduction. We have $\alpha \vdash \texttt{t} : \beta$ and $\vdash \texttt{u} : \alpha$ as our metavariables $\mathfrak{X}$. We then have two terms, both of type β in an empty context: `(λ x. t<x>) $ u<>` and `t<u<>>`.

Working in a new context $\Gamma = \texttt{f} : \alpha \rightarrowtail \beta + \texttt{x} : \alpha$, we can construct the syntactic substitution map $\zeta \in \mathfrak{X}\, \sigma\, \Theta \to \mathcal{A}\, \sigma\, (\Theta, \Gamma)$:

$$\texttt{y} : \alpha + \Gamma \vdash \zeta(\texttt{t}) = \texttt{f \$ y} : \beta$$
$$\Gamma \vdash \zeta(\texttt{u}) = x : \alpha$$

where `y` in the first equation is the parameter of metavariable `t`.

An example application is `(t<u<>>)<ζ>`, which is equivalent to `f $ x`.

3 Free Extensions

Given an algebra $\mathcal{A}$ and metavariables $\mathfrak{X}$, a *free extension* of $\mathcal{A}$ by $\mathfrak{X}$ is a syntactic algebra $\mathcal{F}$ over $\mathfrak{X}$ with the following additional structure:

- a homomorphism $\texttt{sta} \in \mathcal{A} \to \mathcal{F}$
- for every pair of homomorphism $f \in \mathcal{A} \to \mathcal{B}$ and map $g \in \mathcal{X}\, \tau\, \Gamma \to \mathcal{B}\, \tau\, \Gamma$, a homomorphism $\texttt{interp}(f, g) \in \mathcal{F} \to \mathcal{B}$

such that things play nicely[4].

The free extension is an extension in the sense that the algebra $\mathcal{A}$ is given an interpretation of metavariables. It is free in the sense that adding the

[4] I am still resisting giving conditions.

metavariables minimally restricts the homomorphisms to or from the algebra.

Define the map $\texttt{dyn} \in \mathfrak{X}\,\tau\,\Gamma \to \mathcal{F}\,\tau\,\Gamma$ by $\texttt{dyn}(\mathfrak{m}) = \texttt{msub}(\mathfrak{m}, \texttt{var})$. The names `sta` and `dyn` come from the perspective of staged compilation: values made with `sta` are known statically and can be acted on during compilation. Values defined via `dyn` are only known at runtime and have no inherent algebraic structure.

One universal instance of the free extension is given by $\mathbb{T}[\mathcal{A} \cup \mathfrak{X}]$, quotient the conditions listed earlier. Define homomorphisms $\texttt{eval} \in \mathbb{T}[\mathcal{A} \cup \mathfrak{X}] \to \mathcal{F}$ and $\texttt{reify} \in \mathcal{F} \to \mathbb{T}[\mathcal{A} \cup \mathfrak{X}]$ given by `bind` and `interp` respectively. `eval` ideally performs static computations, normalising the term in some way, and `reify` exports the normal form as a term.

For brevity, let $\overline{\texttt{t}}$ represent $\texttt{sta}(\texttt{t})$ and understand $\underline{\mathfrak{m}}$ to be $\texttt{dyn}(\mathfrak{m})$.

4 βη-Normal Form as a Free Extension

In the STLC, βη-normal form is a way of writing terms such that test for βη equivalence is a simple identity check. I will adapt this normal form to attempt to produce a free extension of a STLC model $\mathcal{A}$ by metavariables $\mathfrak{X}$.

.

A term is *dynamic* if there is a subterm that uses a dynamic metavariable. Otherwise it is static. For example, $\underline{\mathfrak{m}}\langle\zeta\rangle$ and $\overline{\texttt{t}}\ \$\ (\lambda\texttt{x}.\ \underline{\mathfrak{n}}\langle\texttt{x}\rangle)$ are both dynamic, whilst $\texttt{x}_i\ \overline{\texttt{t}}$ and $\lambda\texttt{x}.\ \overline{\mathfrak{u}}\langle\texttt{x}\rangle$ are static.

Here is where I define the normal form[5].

5 The Equality Problem

Unfortunately, the normal form just presented is not a model of STLC. In fact, it is not even an algebra. The problem comes from substitution, and can be demonstrated by the following two equal terms:

$$\begin{aligned}
\overline{\texttt{x}_i\ \$\ \texttt{t}}\{\underline{\zeta}\} &= (\overline{\texttt{x}_i}\ \$\ \overline{\texttt{t}})\{\underline{\zeta}\} \\
&= (\texttt{x}_i\ \$\ \overline{\texttt{t}})\{\underline{\zeta}\} \\
&= \texttt{x}_i\{\underline{\zeta}\}\ \$\ \overline{\texttt{t}}\{\underline{\zeta}\} \\
&= \underline{\zeta_i}\ \$\ \overline{\texttt{t}}\langle\underline{\zeta}\rangle
\end{aligned}$$

[5] I should do this.

Assuming the terms have type N, then they are in normal form. The top two lines normalise to the first term, and the bottom two lines normalise to the final term. However, because the subterm $\overline{\texttt{x}_i\ \texttt{\$}\ \texttt{t}}$ is opaque for an arbitrary algebra $\mathcal{A}$, these two normal forms for the same term are necessarily distinct.

For this normal form to be an algebra, it must be taken as a quotient with respect to some relation[6].

6 Removing the Quotient

This is full of loose ideas, without any proof.

The quotient over the normal form can be eliminated in some cases. Starting simple, the quotient is eliminable when the algebra $\mathcal{A}$ to be extend by $\mathfrak{X}$ is given by $\mathcal{A} = \mathbb{T}[\mathfrak{Y}]$ for some fixed family of metavariables $\mathfrak{Y}$. The presented normal form is also the $\beta\eta$-normal form of $\mathbb{T}[\mathfrak{X} \cup \mathfrak{Y}]$, which is an algebra without needing a quotient.

Another example are algebras $\mathcal{A}$ that are "normal forms", with a $\mathfrak{Y}$-syntactic-algebra homomorphism $\mathcal{A} \to \mathbb{T}[\mathfrak{Y}]$ for some fixed family $\mathfrak{Y}$. To evaluate terms $\mathbb{T}[\mathcal{A} \cup \mathfrak{X}]$, first forget $\mathcal{A}$ to obtain $\mathbb{T}[\mathbb{T}[\mathcal{A}] \cup \mathfrak{X}]$. Next normalise this as above, and recompute $\mathcal{A}$ using `interp` and `bind`.

[6] I need to work out what exactly.