Axiom Groups in FC

Richard Eisenberg

August 20, 2012

1 Introduction

Currently, the consistency of an FC context Γ requires that no two axioms (created from Haskell type families) overlap. We wish to lift that restriction by defining a new form of axiom that encapsulates an ordered list of potentially overlapping axioms. Because an old-style single axiom can be seen as a degenerate case of an axiom group, this replaces the old notion of an axiom.

2 Motivation

The primary motivation for overlapping axioms is to be able to define Boolean equality at the type level without needing a quadratic number of instance declarations. We would like the following to work:

```
type family (a :: k) :==: (b :: k) :: Bool
type instance where
  a :==: a = True
  a :==: b = False
```

The concrete syntax is something Dimitris, Simon, and I cooked up and is subject to change.

3 Old axioms

To make a meaningful comparison, it is helpful to review the current state (as proposed in the *Down with kinds* paper) of axioms.

The *old* form of an axiom binding is

$$C{:}\forall\, \Delta.\, (F\, \overline{\tau} \sim \sigma)$$

The variables in the telescope Δ are the free type, kind, and coercion variables taken from the $\overline{\tau}$. Different axioms arising from the same type family might have Δs of different sizes. If we imagine the code above to be rewritten with separate instances, the first instance would have a Δ of $\kappa{:}\star, \alpha{:}\kappa$ and the second would have a Δ of $\kappa{:}\star, \alpha{:}\kappa, \beta{:}\kappa$. The different arities are acceptable because the type inference engine figures out which axiom to apply and creates the FC term with the appropriate axiom and the appropriate number of applications. Upon application to an appropriate Θ such that $\Gamma \vdash_{\mathsf{tc}} \Delta \leftrightsquigarrow \Theta$, the axiom becomes a coercion according to the following rule:

$$\frac{C{:}\forall\, \Delta.\, (F\, \overline{\tau} \sim \sigma) \in \Gamma \qquad \Gamma \vdash_{\mathsf{tc}} \Delta \leftrightsquigarrow \Theta}{\Gamma \vdash_{\mathsf{co}} C\, \Theta : \Theta_1(F\, \overline{\tau}) \sim \Theta_2(\sigma)} \quad \text{CT\_VarAx}$$

The type of the coercion $C\, \Theta$ is the declared type of C with the appropriate substitutions taken out of Θ.

(Θ is a mapping from type variables to triples (τ_1, τ_2, γ) where $\Gamma \vdash_{\mathsf{co}} \gamma : \tau_1 \sim \tau_2$ (modulo internal dependencies). $\Theta_1(\cdot)$ substitutes the corresponding τ_1 and $\Theta_2(\cdot)$ substitutes the corresponding τ_2.)

4 New axiom groups

The *new* form is

$$C{:}F\ \overline{\{\Delta\}\,\overline{\tau}^{n} \rightsquigarrow \sigma}^{m}$$

This type contains a list of m equations. The name of the type family, F, must be the same for each equation and thus is included outside the list. Each equation lists its free type, kind, and coercion variables in Δ. Then comes the list of type patterns $\overline{\tau}$, the left-hand side of one of the equations, and σ, the right-hand side of that equation. We require each equation to have the same number, n, of type patterns. (The value of n does *not* depend on which equation we are looking at.)

This axiom C will be applied to a list of n coercions. One might think that this should be applied to n types, or perhaps n types-and-coercions, but each parameter is a coercion, due to the implementation of coercion optimization within GHC. The idea is that the left-hand sides of the coercions' types will match up with the type patterns $\overline{\tau}$. In fact, even with the new proposed syntax in *Down with kinds*, one can think of the application to the telescoped coercion Θ to be an application to a list of coercions stored in the Θ.

The intuition here is that we wish to search through the list of equations, in order, looking for an equation that matches. When we find one, we can type $C\ \overline{\gamma}^{n}$ according to the match. Here is the well-formedness and coercion typing rules:

$$\frac{\begin{array}{l} C \,\#\, \Gamma \\ \textit{for each } i \leq m \\ \qquad \Gamma, \Delta_i \vdash_{\mathsf{ty}} F\,\overline{\tau_i}^{n} \sim \sigma_i : \star \end{array}}{\vdash_{\mathsf{wf}} \Gamma, C{:}F\ \overline{\{\Delta\}\,\overline{\tau}^{n} \rightsquigarrow \sigma}^{m}} \quad \textsc{GWF\_AxiomGroup}$$

$$\frac{\begin{array}{l} C{:}F\ \overline{\{\Delta\}\,\overline{\tau}^{n} \rightsquigarrow \sigma}^{m} \in \Gamma \\ \overline{\Gamma \vdash_{\mathsf{co}} \gamma : \tau_1 \sim \tau_2}^{n} \\ \Gamma \vdash_{\mathsf{ty}} F\,\overline{\tau_1}^{n} : \kappa \\ (i, \Omega) =_{\Gamma} \mathit{select}_1\,\overline{\gamma}^{n}\ \overline{\{\Delta\}\,\overline{\tau}^{n}}^{m} \end{array}}{\Gamma \vdash_{\mathsf{co}} C\ \overline{\gamma}^{n} : \Omega_L(F\,\overline{\tau_i}) \sim \Omega_R(\sigma_i)} \quad \textsc{CT\_AxiomGroup}$$

The *select* function (defined below) does the work of choosing which equation matches, returning the number of the equation i and a substitution Ω. This Ω is a mapping from type and coercion variables to coercions (which will be built from the $\overline{\gamma}^{n}$ passed to C). The substitution $\Omega_L(\cdot)$ substitutes variables with the left-hand side of the type of the mapped coercion; $\Omega_R(\cdot)$ substitutes with the right-hand side. (If Ω contains $\alpha \mapsto \gamma$ and $\Gamma \vdash_{\mathsf{co}} \gamma : \tau_1 \sim \tau_2$ (modulo internal dependencies), $\Omega_L(\alpha) = \tau_1$ and $\Omega_R(\alpha) = \tau_2$.)

The 1 subscript to the *select* operation says to start the search at the first equation.

Here is the definition of *select*:

$\boxed{(i, \Omega) =_{\Gamma} \mathit{select}_j\,\overline{\gamma}^{n}\ \overline{\{\Delta\}\,\overline{\tau}^{n}}^{m}}$ The select algorithm

$$\frac{\begin{array}{l} j \leq m \\ i = j \\ \Omega =_{\Gamma} \mathit{match}\,\overline{\gamma}^{n}\ \{\Delta_i\}\,\overline{\tau_i}^{n} \\ \forall \Omega'.\, \forall (r < i).\, \mathit{fail} =_{\Gamma} \mathit{match}\,\overline{\Omega'(\gamma)}^{n}\ \{\Delta_r\}\,\overline{\tau_r}^{n} \end{array}}{(i, \Omega) =_{\Gamma} \mathit{select}_j\,\overline{\gamma}^{n}\ \overline{\{\Delta\}\,\overline{\tau}^{n}}^{m}} \quad \textsc{Select\_Match}$$

$$\frac{\begin{array}{l} j' = j + 1 \\ i = j \\ \mathit{fail} =_{\Gamma} \mathit{match}\,\overline{\gamma}^{n}\ \{\Delta_i\}\,\overline{\tau_i}^{n} \\ (r, \Omega) =_{\Gamma} \mathit{select}_{j'}\,\overline{\gamma}^{n}\ \overline{\{\Delta\}\,\overline{\tau}^{n}}^{m} \end{array}}{(r, \Omega) =_{\Gamma} \mathit{select}_j\,\overline{\gamma}^{n}\ \overline{\{\Delta\}\,\overline{\tau}^{n}}^{m}} \quad \textsc{Select\_NoMatch}$$

Unsurprisingly, *select* iterates through the list of equations and returns the result from the one that matches, delegating the match operation to *match*. (The various indexing operations are just to get things to work with ott. A more formal presentation would fix that up.)

The only interesting piece to this judgment is the last premise to the first rule. It asserts that, for all possible substitutions Ω', all previous equations will not match. At first, this seems redundant, because we start matching at equation #1 and stop at the first match. However, it is important to note that the domain of Ω' is *not* the domain of Δ. This premise is stating that for any possible assignment of type variables in $\overline{\gamma}^n$ (*not* the type variables in $\overline{\tau}$), the match will fail. Another way of saying this is that the left-hand types in the types of $\overline{\gamma}^n$ cannot unify with the $\overline{\tau}$. Say a coercion is $\langle\alpha\rangle$ for some type variable $\alpha \in \mathit{dom}\,\Gamma$. According to *match* (defined below), $\langle\alpha\rangle$ will not match with **Int**. However, maybe α will be substituted with **Int** at some point. Because of this possibility, we must make sure that before committing to a match, no possible match could have worked no matter what happens to the variables in the $\overline{\gamma}^n$. In this case, the overall coercion will be ill-typed, because there is no unambiguous match.

Here is the match relation:

$\boxed{\Omega =_\Gamma \mathit{match}\,\overline{\gamma}^n\,\{\Delta\}\,\overline{\tau}^n}$ Coercion list matching

$$\frac{\overline{\Gamma \vdash_{\mathsf{co}} \gamma : \sigma_1 \sim \sigma_2}^n \quad \overline{\Omega_L(\tau) = \sigma_1}^n \quad \mathit{dom}\,\Omega = \mathit{dom}\,\Delta}{\Omega =_\Gamma \mathit{match}\,\overline{\gamma}^n\,\{\Delta\}\,\overline{\tau}^n}\ \textsc{MatchI\_Match}$$

For a Ω such that the domain of Ω matches the free variables in the $\overline{\tau}$, check if the substitution returns the desires types, and succeed if so. Unfortunately, this judgment gives us no way of finding Ω. So, we use the algorithmic version presented below, with all its auxiliary judgments:

$\boxed{\Omega =_\Gamma \mathit{match}\,\overline{\gamma}^n\,\{\Delta\}\,\overline{\tau}^n}$ Coercion list matching (algorithmic)

$$\frac{n = 0}{\epsilon =_\Gamma \mathit{match}\,\overline{\gamma}^n\,\{\Delta\}\,\overline{\tau}^n}\ \textsc{Match\_Nil}$$

$$\frac{\begin{array}{l}\overline{\gamma}^n = \gamma, \overline{\gamma'}^m \\ \overline{\tau}^n = \tau, \overline{\tau'}^m \\ n = m + 1 \\ \Delta' \textit{ is the smallest prefix of } \Delta \textit{ s.t. } \Gamma, \Delta' \vdash_{\mathsf{ty}} \tau : \kappa \\ \Omega_1 =_\Gamma \mathit{match}\,\overline{\gamma'}^m\,\{\Delta \backslash \Delta'\}\,\overline{\Omega_{2L}(\tau')}^m \\ \Gamma \vdash \gamma \overset{\Delta'}{\leftrightsquigarrow} \tau : \Omega_2 \\ \Omega = \Omega_1, \Omega_2\end{array}}{\Omega =_\Gamma \mathit{match}\,\overline{\gamma}^n\,\{\Delta\}\,\overline{\tau}^n}\ \textsc{Match\_Cons}$$

The "smallest prefix" condition is there to select out the prefix of Δ that contains all of the free variables in τ.

$\boxed{\Gamma \vdash \gamma \overset{\Delta}{\leftrightsquigarrow} \tau : \Omega}$ Coercion / type pattern match

$$\frac{\Gamma \vdash_{\mathsf{co}} \gamma : \tau \sim \sigma}{\Gamma \vdash \gamma \overset{\epsilon}{\leftrightsquigarrow} \tau : \epsilon}\ \textsc{Match1\_Exact}$$

$$\frac{\Gamma \vdash_{\mathsf{co}} \gamma : \phi}{\Gamma \vdash \gamma \overset{\beta:\kappa}{\leftrightsquigarrow} \beta : \beta \mapsto \gamma}\ \textsc{Match1\_Var}$$

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{co}} \gamma : \sigma\,\overline{\sigma_1}^n \sim \sigma\,\overline{\sigma_2}^n \\ \Delta_0 \textit{ is the smallest prefix of } \Delta \textit{ s.t. } \Gamma, \Delta_0 \vdash_{\mathsf{ty}} \tau : \kappa \\ \Gamma \vdash \langle \sigma \rangle \overset{\Delta_0}{\leftrightsquigarrow} \tau : \Omega_0 \\ \textit{for each } \tau'_i \in \overline{\tau'}^n \\ \quad i = r + 1 \\ \quad \Delta_i \textit{ is the smallest prefix of } \Delta \backslash \Delta_r \textit{ s.t. } \Gamma, \Delta_i \vdash_{\mathsf{ty}} \Omega_{rL}(\tau'_i) : \kappa_i \\ \quad \Gamma \vdash \mathbf{nth}^i\ \gamma \overset{\Delta_i}{\leftrightsquigarrow} \Omega_{rL}(\tau'_i) : \Omega'_i \\ \quad \Omega_i = \Omega_r, \Omega'_i \\ \Omega = \Omega_n \\ \tau \textit{ is not a type application}\end{array}}{\Gamma \vdash \gamma \overset{\Delta}{\leftrightsquigarrow} \tau\,\overline{\tau'}^n : \Omega}\ \textsc{Match1\_App}$$

In the last rule, we go left-to-right, carefully building up a substitution Ω as we go. Once again, some care must be taken to get the Δ right so that the following lemma holds:

Lemma 1 (Substitution domain)**.** *If* $\Gamma \vdash \gamma \overset{\Delta}{\leftrightsquigarrow} \tau : \Omega$ *then the domain of* Ω *is the domain of* Δ*.*

Acknowledgments

Thanks to José Pedro Magalhães for starting this design process. Thanks also to Dimitrios Vytiniotis and Simon Peyton Jones for helping me start this project.