Formalism for Datatype Extensions in Haskell

Tim Zakian

July 23, 2015

1 Language Definition

- Why does this not create open datatypes? How does it differ and how does it get around the problems found in open datatypes?
- Currently only have core $F_C(X)$ as presented in [1]. Need to extend this with the extendable data type section (i.e., the hard stuff...).

Symbol Classes

$$
\begin{array}{lll}
a, b, c, co & \rightarrow & \langle \text{type variable} \rangle \\
x, f & \rightarrow & \langle \text{term variable} \rangle \\
C & \rightarrow & \langle \text{coercion constant} \rangle \\
T & \rightarrow & \langle \text{value type constructor} \rangle \\
S_n & \rightarrow & \langle n - \text{ary type function} \rangle \\
K & \rightarrow & \langle \text{data constructor} \rangle
\end{array}
$$

Sorts and Kinds

$$
\begin{array}{llll}
\delta & \rightarrow & \mathrm{TY} \mid \mathrm{CO} & \text{Sorts} \\
\kappa, \iota & \rightarrow & \star \mid \kappa_1 \rightarrow \kappa_2 \mid \sigma_1 \sim \sigma_2 & \text{Kinds}
\end{array}
$$

Syntactic sugar

Types $\kappa \Rightarrow \sigma \;\equiv\; \forall\_ : \kappa.\sigma$

Declarations

$$
\begin{array}{lll}
pgm & \rightarrow & \overline{cdecl}; \overline{decl}; e \\
cdecl & \rightarrow & \textbf{data open } T : \overline{\kappa} \rightarrow \star \textbf{ where} \\
 & & \overline{\mathtt{K} : \forall\, \overline{a : \kappa}.\, \forall\, \overline{b : \iota}. \overline{\sigma} \rightarrow T\, \overline{a}} \\
 & \mid & \textbf{data } T : \overline{\kappa} \rightarrow \star \textbf{ where} \\
 & & \overline{\mathtt{K} : \forall\, \overline{a : \kappa}.\, \forall\, \overline{b : \iota}. \overline{\sigma} \rightarrow T\, \overline{a}} \\
 & & \textbf{extends}(\overline{T}) \\
 & \mid & \textbf{data open } T : \overline{\kappa} \rightarrow \star \textbf{ where} \\
 & & \overline{\mathtt{K} : \forall\, \overline{a : \kappa}.\, \forall\, \overline{b : \iota}. \overline{\sigma} \rightarrow T\, \overline{a}} \\
 & & \textbf{extends}(\overline{T}) \\
decl & \rightarrow & \textbf{data } T : \overline{\kappa} \rightarrow \star \textbf{ where} \\
 & & \overline{\mathtt{K} : \forall\, \overline{a : \kappa}.\, \forall\, \overline{b : \iota}. \overline{\sigma} \rightarrow T\, \overline{a}} \\
 & \mid & \textbf{type } S_n : \overline{\kappa}^n \rightarrow \iota \\
 & \mid & \textbf{axiom } C : \sigma_1 \sim \sigma_2
\end{array}
$$

Types and Coercions

$$
\begin{array}{llll}
d & \rightarrow & a \mid T & \text{Atom of sort TY} \\
g & \rightarrow & c \mid C & \text{Atom of sort CO} \\
\varphi, \rho, \sigma, & & & \\
\tau, \nu, \gamma & \rightarrow & a \mid C \mid T \mid \varphi_1\ \varphi_2 \mid S_n\ \overline{\varphi}^n \mid \forall a : \kappa.\varphi & \\
 & \mid & \text{sym } \gamma \mid \gamma_1 \circ \gamma_2 \mid \gamma @ \varphi \mid \text{left } \gamma \mid \text{right } \gamma & \\
 & \mid & \gamma \sim \gamma \mid \text{rightc } \gamma \mid \text{leftc } \gamma \mid \gamma \blacktriangleright \gamma & \\
 & \mid & \zeta_1 \underset{[T_1, T_2]}{\sim} \zeta_2 & \\
\zeta & \rightarrow & (K, \sigma) \mid \sigma &
\end{array}
$$

Terms

$$
\begin{array}{llll}
u & \rightarrow & x \mid K & \text{Variables and data consructors} \\
e & \rightarrow & u & \text{Term atoms} \\
 & \mid & \Lambda a : \kappa.e \mid e\varphi & \text{Type abstractions/application} \\
 & \mid & \lambda x : \sigma.e \mid e_1\ e_2 & \text{Term abstraction/application} \\
 & \mid & \textbf{let } x : \sigma = e_1 \textbf{ in } e_2 & \\
 & \mid & \textbf{case } e_1 \textbf{ of } \overline{p \rightarrow e_2} & \\
 & \mid & e \blacktriangleright \gamma \mid e \underset{[T_1, T_2]}{\triangleright} \gamma & \text{Cast} \\
 & & & \\
p & \rightarrow & K\, \overline{b : \kappa}\ \overline{x : \sigma} & \text{Pattern}
\end{array}
$$

Environments

$\Gamma \;\rightarrow\; \epsilon \mid \Gamma, u : \sigma \mid \Gamma, d : \kappa \mid \Gamma, g : \kappa \mid \Gamma, S_n : \kappa \mid \Gamma, (T : \overline{\zeta})$

A top-level environment binds only type constructors, T, S_n, data constructors K, and coercion constants C.

Figure 1: The core language for extensible data types – $F_C(X)$

2 Typing Rules

2.1 Extended typing rules

TODO:

- **Explain what cast extrusion is and why it's bad**
- Need to formalize "Get all the data constructors for this type constructor" in this system since we will need it for ExtData in our environment extension rules.
- We need the extra equivalences since we need to prevent casts between differenet constructors that have the same type.

We now extend the core typing rules for System F_C with a way to cast between extended and core data constructors, while at the same time preventing "cast extrusion" on the constructors and corresponding types.

Definition We say that $\sigma_1 \underset{[T_1,T_2]}{\equiv} \sigma_2$ if:

$$\sigma_1 = \forall \overline{a:\kappa}\forall\overline{b:\iota}.\overline{\sigma} \to T_1\ \overline{a}$$

and

$$\sigma_2 = \forall \overline{a:\kappa}\forall\overline{b:\iota}.\overline{\sigma} \to T_2\ \overline{a}$$

i.e., that they are they same type modulo substitution of the type constructor.

Definition: We define $\zeta[\![T/T']\!]$ to be the substitution of the type constructor application the underlying type σ.

Definition: We say that T extends T' and that $T > T'$ if T is a type constructor derived using a **extends** form.

Theorem 1 (Transitivity of $(>)$). *Proof.* WRITE ME □

Theorem 2 ($[\![T/T']\!]$ preserves well-typedness). *Proof.* WRITE ME □

Theorem 3 (Extension casts preserve well-typedness). *Proof.* WRITE ME □

Theorem 4 (Domain-restricted reachability of types). *Proof.* WRITE ME □

We then extend the typing judgements in Figure 2 with the two rules in Figure 4, and omit the other rules (transitivity etc.) since these can be easily figured out.

$$\text{ECoreCast}\ \frac{\Gamma \vdash_{\mathrm{e}} e : \zeta' \qquad \Gamma \vdash_{\mathrm{CO}} \gamma : \zeta' \underset{[T_1,T_2]}{\sim} \zeta}{\Gamma \vdash_{\mathrm{e}} e \underset{[T_1,T_2]}{\triangleright} \gamma : \zeta} \qquad \text{CoCoreVar}\ \frac{\begin{array}{cc} \zeta_1 = (\mathtt{K}, \sigma_1) \in \Gamma & \zeta_2 = (\mathtt{K}, \sigma_2) \in \Gamma \\ \exists T_1, T_2. \left(\sigma_1 \underset{[T_1,T_2]}{\equiv} \sigma_2\right) & (\gamma : \zeta_1 \underset{[T_1,T_2]}{\sim} \zeta_2) \in \Gamma \end{array}}{\Gamma \vdash_{\mathrm{CO}} \gamma : \zeta_1 \underset{[T_1,T_2]}{\sim} \zeta_2}$$

$$\boxed{\Gamma \vdash cdecl : \Gamma'}$$

$$\text{CoreData}\ \frac{\overline{\Gamma \vdash_{\mathrm{TY}} \sigma : \star} \qquad \Gamma \vdash_{\mathrm{k}} \kappa : TY}{\Gamma \vdash \textbf{data open}\ T\ :\ \overline{\kappa}\ \textbf{where}\ \overline{K:\sigma} : (T:\kappa, \overline{K:\sigma}, (T; \overline{(K,\sigma)}))}$$

$$\text{ExtData}\ \frac{\begin{array}{cc} \overline{\Gamma \vdash_{\mathrm{TY}} \sigma : \star} & \Gamma \vdash_{\mathrm{k}} \kappa : TY \\ \multicolumn{2}{c}{\overline{(K',\sigma')} \triangleq \overline{\forall (K',\sigma'') \in lookup(T,\Gamma).(K',\sigma'')[\![T'/T]\!]}} \end{array}}{\Gamma \vdash \textbf{data}\ T\ :\ \overline{\kappa}\ \textbf{where}\ \overline{K:\sigma}\ \textbf{extends}(\overline{T}) : (T':\kappa, \overline{K:\sigma}, \overline{K':\sigma'}, \overline{(K',\sigma') \underset{[T,T']}{\sim} (K',\sigma'')})}$$

Figure 4: Extension of the typing rules to allow extension of data types

$$\boxed{\Gamma \vdash_{\text{TY}} \sigma : \kappa}$$

$$\text{TyVar}\ \frac{(d : \kappa) \in \Gamma \qquad \Gamma \vdash_{\text{k}} \kappa : TY}{\Gamma \vdash_{\text{TY}} d : \kappa} \qquad \text{TyApp}\ \frac{\Gamma \vdash_{\text{TY}} \sigma_1 : \kappa_1 \to \kappa_2 \qquad \Gamma \vdash_{\text{TY}} \sigma_2 : \kappa_1}{\Gamma \vdash_{\text{TY}} \sigma_1\ \sigma_2 : \kappa_2}$$

$$\text{TySCon}\ \frac{(S_n : \overline{\kappa}^n \to \iota) \in \Gamma \qquad \Gamma \vdash_{\text{TY}} \overline{\sigma : \kappa}^n}{\Gamma \vdash_{\text{TY}} S_n \overline{\sigma}^n : \iota} \qquad \text{TyAll}\ \frac{\Gamma, a : \kappa \vdash_{\text{TY}} \sigma : \star \qquad \Gamma \vdash_{\text{k}} \kappa : \delta \qquad a \notin fv(\Gamma)}{\Gamma \vdash_{\text{TY}} \forall a : \kappa.\sigma : \star}$$

$$\boxed{\Gamma \vdash_{\text{CO}} \gamma : \sigma \sim \tau}$$

$$\text{CoRefl}\ \frac{(a : \kappa) \in \Gamma \qquad \Gamma \vdash_{\text{k}} \kappa : TY}{\Gamma \vdash_{\text{CO}} a : a \sim a} \qquad \text{CoVar}\ \frac{(g : \sigma \sim \tau) \in \Gamma}{\Gamma \vdash_{\text{CO}} g : \sigma \sim \tau} \qquad \text{CoAllT}\ \frac{\begin{array}{c}\Gamma, a : \kappa \vdash_{\text{CO}} \gamma : \sigma \sim \tau \\ \Gamma \vdash_{\text{k}} \kappa : TY \qquad a \notin fv(\Gamma)\end{array}}{\Gamma \vdash_{\text{CO}} \forall a : \kappa.\gamma : \forall a : \kappa.\sigma \sim \forall a : \kappa.\tau}$$

$$\text{CoInstT}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CO}} \gamma : \forall a.\kappa.\sigma \sim \forall b : \kappa.\tau \\ \Gamma \vdash_{\text{TY}} v : \kappa\end{array}}{\Gamma \vdash_{\text{CO}} \gamma @ v : [v/a]\sigma \sim [v/b]\tau} \qquad \text{SComp}\ \frac{\Gamma \vdash_{\text{CO}} \overline{\gamma : \sigma \sim \tau}^n \qquad \Gamma \vdash_{\text{TY}} S_n \overline{\sigma}^n : \kappa}{\Gamma \vdash_{\text{CO}} S_n \overline{\gamma}^n : S_n \overline{\sigma}^n \sim S_n \overline{\tau}^n} \qquad \text{Sym}\ \frac{\Gamma \vdash_{\text{CO}} \gamma : \sigma \sim \tau}{\Gamma \vdash_{\text{CO}} \gamma : \tau \sim \sigma}$$

$$\text{Trans}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CO}} \gamma_1 : \sigma_1 \sim \sigma_2 \\ \Gamma \vdash_{\text{CO}} \gamma_2 : \sigma_2 \sim \sigma_3\end{array}}{\Gamma \vdash_{\text{CO}} \gamma_1 \circ \gamma_2 : \sigma_1 \sim \sigma_3} \qquad \text{Comp}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CO}} \gamma_1 : \sigma_1 \sim \tau_1 \qquad \Gamma \vdash_{\text{CO}} \gamma_2 : \sigma_2 \sim \tau_2 \\ \Gamma \vdash_{\text{TY}} \sigma_1\ \sigma_2 : \kappa\end{array}}{\Gamma \vdash_{\text{CO}} \gamma_1\ \gamma_2 : \sigma_1\ \sigma_2 \sim \tau_1\ \tau_2} \qquad \text{Left}\ \frac{\Gamma \vdash_{\text{CO}} \gamma : \sigma_1\ \sigma_2 \sim \tau_1\ \tau_2}{\Gamma \vdash_{\text{CO}} \text{left}\ \gamma : \sigma_1 \sim \tau_1}$$

$$\text{Right}\ \frac{\Gamma \vdash_{\text{CO}} \gamma : \sigma_1\ \sigma_2 \sim \tau_1\ \tau_2}{\Gamma \vdash_{\text{CO}} \text{right}\ \gamma : \sigma_1 \sim \tau_1} \qquad \text{CompC}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CO}} \gamma : \kappa_1 \sim \kappa_2 \qquad \Gamma \vdash_{\text{CO}} \gamma' : \sigma_1 \sim \sigma_2 \\ \Gamma \vdash_{\text{k}} \kappa_1 : CO\end{array}}{\Gamma \vdash_{\text{CO}} \gamma \Rightarrow \gamma' : (\kappa_1 \Rightarrow \sigma_1) \sim (\kappa_2 \Rightarrow \sigma_2)}$$

$$\text{LeftC}\ \frac{\Gamma \vdash_{\text{CO}} \gamma : \kappa_1 \Rightarrow \sigma_1 \sim \kappa_2 \Rightarrow \sigma_2}{\Gamma \vdash_{\text{CO}} \text{leftc}\ \gamma : \kappa_1 \sim \kappa_2} \qquad \text{RightC}\ \frac{\Gamma \vdash_{\text{CO}} \gamma : \kappa_1 \Rightarrow \sigma_1 \sim \kappa_2 \Rightarrow \sigma_2}{\Gamma \vdash_{\text{CO}} \text{rightc}\ \gamma : \sigma_1 \sim \sigma_2}$$

$$(\sim)\ \frac{\Gamma \vdash_{\text{CO}} \gamma_1 : \sigma_1 \sim \tau_1 \qquad \Gamma \vdash_{\text{CO}} \gamma_2 : \sigma_2 \sim \tau_2}{\Gamma \vdash_{\text{CO}} \gamma_1 \sim \gamma_2 : (\sigma_1 \sim \sigma_2) \sim (\tau_1 \sim \tau_2)} \qquad \text{CastC}\ \frac{\Gamma \vdash_{\text{CO}} \gamma_1 : \kappa \qquad \Gamma \vdash_{\text{CO}} \gamma_2 \kappa \sim \kappa'}{\Gamma \vdash_{\text{CO}} \gamma_1 \blacktriangleright \gamma_2}$$

$$\boxed{\Gamma \vdash_{\text{e}} e : \zeta}$$

$$\text{Var}\ \frac{(u : \sigma) \in \Gamma}{\Gamma \vdash_{\text{e}} u : \sigma} \qquad \text{Case}\ \frac{\Gamma \vdash_{\text{e}} e : \sigma \qquad \overline{\Gamma \vdash_{\text{p}} p \to e : \sigma \to \tau}}{\Gamma \vdash_{\text{e}} \textbf{case}\ e\ \textbf{of}\ \overline{p \to e} : \tau} \qquad \text{Let}\ \frac{\Gamma \vdash_{\text{e}} e_1 : \sigma_1 \qquad \Gamma, x : \sigma_1 \vdash_{\text{e}} e_2 : \sigma_2}{\Gamma \vdash_{\text{e}} \textbf{let}\ x : \sigma_1 = e_1\ \textbf{in}\ e_2 : \sigma_2}$$

$$\text{Cast}\ \frac{\Gamma \vdash_{\text{e}} e : \sigma \qquad \Gamma \vdash_{\text{CO}} \gamma : \sigma \sim \tau}{\Gamma \vdash_{\text{e}} e \blacktriangleright \gamma : \tau} \qquad \text{Abs}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{TY}} \sigma_x : \star \\ \Gamma, x : \sigma_x \vdash_{\text{e}} e : \sigma\end{array}}{\Gamma \vdash_{\text{e}} \lambda x : \sigma_x.e : \sigma_x \to \sigma} \qquad \text{App}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{e}} e_1 : \sigma_2 \to \sigma_1 \\ \Gamma \vdash_{\text{e}} e2 : \sigma_2\end{array}}{\Gamma \vdash_{\text{e}} e_1\ e_2 : \sigma_1}$$

$$\text{AbsT}\ \frac{\Gamma a : \kappa \vdash_{\text{e}} e : \sigma \qquad \Gamma \vdash_{\text{k}} \kappa : \delta \qquad a \notin fv(\Gamma)}{\Gamma \vdash_{\text{e}} \Lambda a : \kappa.e : \forall a : \kappa.\sigma} \qquad \text{AppT}\ \frac{\Gamma \vdash_{\text{e}} e : \forall a : \kappa.\sigma \qquad \Gamma \vdash_{\text{k}} \kappa : \delta \qquad \Gamma \vdash_{\delta} \varphi : \kappa}{\Gamma \vdash_{\text{e}} e\ \varphi : \sigma[\varphi/a]}$$

$$\boxed{\Gamma \vdash_{\text{p}} p \to e : \sigma \to \tau}$$

$$\text{Alt}\ \frac{K : \forall \overline{a : \kappa}.\forall \overline{b : \iota}.\overline{\sigma} \to T\ \overline{a} \in \Gamma \qquad \theta = [\overline{v/a}] \qquad \Gamma, \overline{b : \theta(\iota)}, \overline{x : \theta(\sigma)} \vdash_{\text{e}} e : \tau}{\Gamma \vdash_{\text{p}} K\ \overline{b : \theta(\iota)}\ \overline{x : \theta(\sigma)} \to e : T\ \overline{v} \to \tau}$$

Figure 2: Typing rules for the core language of $F_C(X)$

$$\boxed{\Gamma \vdash decl : \Gamma'}$$

$$\text{Data}\ \frac{\overline{\Gamma \vdash_{\text{TY}} \sigma : \star} \qquad \Gamma \vdash_{\text{k}} \kappa : TY}{\Gamma \vdash \textbf{data}\ T : \kappa\ \textbf{where}\ \overline{K : \sigma} : (T : \kappa, \overline{K : \sigma})}$$

$$\text{Type}\ \frac{\Gamma \vdash_{\text{k}} \kappa : TY}{\Gamma \vdash (\textbf{type}\ S : \kappa) : (S : \kappa)} \qquad \text{Coerce}\ \frac{\Gamma \vdash_{\text{k}} \kappa : CO}{\Gamma \vdash (\textbf{axiom}\ C : \kappa) : (C : \kappa)}$$

$$\boxed{\Gamma \vdash pgm : \sigma}$$

$$\text{Pgm}\ \frac{\begin{array}{cc}\overline{\Gamma \vdash cdecl : \Gamma_c} & \Gamma = \Gamma_0, \overline{\Gamma_c} \\ \overline{\Gamma \vdash decl : \Gamma_d} & \Gamma = \Gamma, \overline{\Gamma_d} \\ \multicolumn{2}{c}{\Gamma \vdash_{\text{e}} e : \sigma}\end{array}}{\Gamma_0 \vdash \overline{cdecl}; \overline{decl}; e : \sigma}$$

Figure 3: Environment extension rules for $F_C(X)$

References

[1] Martin Sulzmann, Manuel MT Chakravarty, Simon Peyton Jones, and Kevin Donnelly. System f with type equality coercions. In *Proceedings of the 2007 ACM SIGPLAN international workshop on Types in languages design and implementation*, pages 53–66. ACM, 2007.