System FC with Explicit Kind Equality

Stephanie Weirich  Justin Hsu  Richard A. Eisenberg

International Conference on Functional Programming
Thursday, September 26, 2013
Boston, MA, USA
Dependent types + Haskell
Disclaimer

No dependent types in Haskell, yet.

No dependent types in FC.

Yes: Support for dependently-typed programming using singletons in FC.
What we can do now

Generalized Algebraic Data Types (GADTs):

data Typ = TInt | TArrow Typ Typ

data Var :: [Typ] → Typ → ★ where
  VZero :: Var (a ': ctx) a
  VSucc :: Var ctx a → Var (b ': ctx) a

strengthen :: Var (b ': ctx) a
            → Maybe (Var ctx a)
strengthen VZero = Nothing
strengthen (VSucc v) = Just v
data Typ = TInt | TArrow Typ Typ

data Var :: [Typ] → Typ → ★ where
  VZero :: Var (a ‘: ‘ ctx) a
  VSucc :: Var ctx a → Var (b ‘: ‘ ctx) a

ghci> :kind Var TInt TInt

The first argument of Var should have kind [Typ] but TInt has kind Typ
Programming in types

Type-level functions:

```haskell
type family Interpret (t :: Typ) :: ★
type instance Interpret TInt = Int
type instance Interpret (TArrow a b) = (Interpret a) → (Interpret b)
```

Kind polymorphism:

```
(′:′) :: ∀ k. k → [k] → [k]
```
WellScoped

data OutOfScope :: [Typ] -> Nat -> ★ where
   Oops :: OutOfScope '[] n
   Succ :: OutOfScope ctx n
         -> OutOfScope (a ': ctx) (1 + n)

data WellScoped :: [Typ] -> Nat -> ★ where
   Yes :: ∀ (x :: Var ctx a).
       WellScoped ctx (EraseVar x)
   No :: OutOfScope ctx n -> WellScoped ctx n

Var of kind [Typ] -> Typ -> ★ is not promotable
Types vs. Kinds

**Types**

Typ

(\texttt{::}) : \forall a. a \rightarrow [a] \rightarrow [a]

Var

EraseVar

**Kinds**

Typ

(\texttt{`}::`) : \forall k. k \rightarrow [k] \rightarrow [k]

Need universal promotion of types to kinds
We need universal promotion to be able to express dependently-typed programs in Haskell.
How to proceed?

GHC compiles Haskell to System FC, a strongly-typed intermediate language

⇒

System FC must support universal promotion
System FC

- System FC must have decidable, fast type-checking
  - “System FC” = “System F with coercions”
  - ... but only type coercions

- Type coercions are used to...
  - ... implement GADTs
  - ... implement type families
GADTs to Coercions

```
data Typ = TInt | TArrow Typ Typ

data Var :: [Typ] → Typ → ★ where
  VZero :: Var (a ': ctx) a
  VSucc :: Var ctx a → Var (b ': ctx) a
```

```
Typ :: ★
TInt :: Typ
TArrow :: Typ → Typ → Typ
Var :: [Typ] → Typ → ★

VZero :: ∀ (ctx :: [Typ]) (a :: Typ). ∀ (ctx0 :: [Typ]).
        (ctx ~ (a ': ctx0)) → Var ctx a

VSucc :: ∀ (ctx :: [Typ]) (a :: Typ).
        ∀ (ctx0 :: [Typ]) (b0 :: Typ).
        (ctx ~ (b0 ': ctx0)) → Var ctx0 a → Var ctx a
```
GADT Pattern-match

```
strengthen :: Var (b `':` ctx) a -> Maybe (Var ctx a)
strengthen VZero = Nothing
strengthen (VSucc v) = Just v
```

```
VSuccess :: \((\text{ctx} :: [\text{Typ}]) \to (\text{a} :: \text{Typ})\).
  \forall (\text{ctx0} :: [\text{Typ}]) (\text{b0} :: \text{Typ})
  (\text{ctx} \sim (\text{b0 `':` ctx0})) \to (\text{Var ctx0 a}) \to (\text{Var ctx a})
```

In pattern match:

```
\ldots
co :: (b `':` ctx) \sim (\text{b0 `':` ctx0})
v :: \text{Var ctx0 a}
--------------------
\text{Var ctx a}
```

Answer:
Cast by a coercion built from co
If we want type-level GADTs, we need kind-level coercions.
Adding kind coercions is hard.
Merging types and kinds

\[
\tau ::= \alpha \\
| H \quad \text{constant} \\
| \tau_1 \tau_2 \quad \text{application} \\
| \forall (\alpha:\kappa).\tau \quad \text{polymorphism}
\]

\[
\kappa ::= \chi \\
| D \quad \star \quad \text{constants} \\
| \kappa_1 \kappa_2 \quad \text{application} \\
| \forall \chi.\kappa \quad \text{polymorphism}
\]

\[
\tau, \kappa ::= \\
| \alpha \quad \text{variable} \\
| H \quad \star \quad \text{constants} \\
| \tau_1 \tau_2 \quad \text{application} \\
| \forall (\alpha:\kappa).\tau \quad \text{polymorphism} \\
| \ldots \quad \ldots
\]
• What is ★’s type?
  ▸ Common answer: infinite hierarchy of universes (★₀, ★₁, ★₂, ...)
  ▸ Our answer: ★ : ★
• Isn’t that dangerous?
  ▸ Haskell is not a logic: all types are inhabited already
  ▸ Type safety requires consistency of coercions
  ▸ Proof of coercion consistency in paper
Heterogeneous Equality

• Consider:

\[
\begin{align*}
\text{id} & :: \forall (a :: \star). a \rightarrow a \\
\kappa & :: \star \\
\gamma_1 & :: \text{id} \sim \text{id} \\
\gamma_2 & :: \kappa \sim \star \\
\gamma_3 & :: \text{id} \kappa \sim \text{id} \star \\
id \kappa & :: \kappa \rightarrow \kappa \\
\text{id} \star & :: \star \rightarrow \star
\end{align*}
\]

• Thus, \( \gamma_3 \) is a heterogeneous coercion.

• Design option: do we allow these?

• Design decision: yes -- “John Major” equality
Our contributions

- Full details of enhanced System FC, supporting
  - universal promotion of datatypes
  - kind-level functions
  - kind-indexed GADTs (see paper)
- Operational semantics and “push rules”
  - $\Rightarrow$ lifting lemma, for the Preservation Theorem
- The consistency lemma: why $\text{Int} \not\rightarrow \text{Bool}$
  - $\Rightarrow$ necessary for the Progress Theorem
- Prototype implementation (Core language only)
Future work