The Implicit Calculus

Wontae Choi[1], Wonchan Lee[1], Bruno C. d. S. Oliveira[1], Tom Schrijvers[2] and Kwangkeun Yi[1]

[1] Seoul National University [2] Universiteit Gent

1. Implicit Programming

- Programming paradigm in which some function arguments are inferred by their types
- Proven to be practical by many programming languages, such as Haskell, Scala and C++

C++

```
bool eq(int& x, int& y)
{
    return x == y;
}

template<typename a, typename b>
bool eq(pair<a, b>& x, pair<a, b>& y)
{
    return eq(x.first, y.first) &&
           eq(x.second, y.second);
}

int main()
{
    pair<pair<int, int>, int> a, b;
    a = ((1, 1), 1);
    b = ((2, 2), 2);
    eq(a, b);
}
```

Scala

```
trait Eq[a] { def eq(x:a, y:a):Boolean }

implicit def EqPair[a, b]
(implicit eqA : Eq[a], eqB : Eq[b]) =
  new Eq[(a, b)] {
      def eq (x:(a, b), y:(a, b)) =
        eqA.eq(a._1, b._1) &&
        eqB.eq(a._2, b._2)
    }

implicit def EqPair[Int] =
  new Eq[Int] {
    def eq (x:Int, y:Int) = eqInt x y
  }

eq ((1, 1), 1) ((2, 2), 2)
```

Haskell

```
class Eq a where eq :: a -> a -> Bool

instance Eq Int where eq = eqInt

instance (Eq a, Eq b) => Eq (a, b) where
    eq a b = eq (fst a) (fst b) &&
             eq (snd a) (snd b)

main = eq ((1, 1), 1) ((2, 2), 2)
```

2. Problems of state-of-the-art Implicit Programming

- Current practices are either well-formalized but restricted (e.g. Haskell) or not well-formalized (e.g. Scala and C++)
- Coherence of programs, especially with overlapping instances, is not well-studied:

module A

```
let f : ∀b.b -> b =
    implicit { λx.x     : ∀a.a -> a,
               λx.x + 1 : int -> int } in
    ?(b -> b)
```

module B

```
let a = A.f 1
let b = A.f 'x'
```

  - the program is coherent if `B.a = 2` and `B.b = 'x'`
  - no system in the literature can make this program coherent
- No existing system supports all useful features in one system

	Haskell	Scala	C++
local instances	X	O	X
overlapping instances	▲	X	▲
higher-order rules	X	X	X

3. Solution: Implicit Calculus λ_ρ

- Core calculus for implicit programming
- Encompasses all the current practices
- Type-safe program = coherent program

$$\textbf{let } f = \Lambda X.(\textbf{rule}(\forall b.\{\forall a.a \to a, \textbf{int} \to \textbf{int}\} \Rightarrow b \to b)(?b \to b))[X] \\ \textbf{with } \{\forall c.c \to c : \lambda x.x, \textbf{int} \to \textbf{int} : \lambda x.x + 1\}$$

$$\textbf{let } a = f\ \textbf{int}\ 1$$
$$\textbf{let } b = f\ \textbf{char}\ 'x'$$

  - this program is coherent according to our operational semantics

4. Formalism

Syntax

$$e ::= ?\rho \mid \textbf{rule}\ \rho\ e \mid e[\bar{\tau}] \mid e\ \textbf{with}\ \overline{e:\rho}$$
$$\rho ::= \forall\overline{\alpha}.\pi \Rightarrow \rho \qquad \pi ::= \bar{\rho}$$

Operational Semantics

$$\boxed{\mathcal{E} \vdash e \Downarrow v}$$

(OpQuery)
$$\frac{\mathcal{E} \vdash_r \rho \Downarrow v}{\mathcal{E} \vdash ?\rho \Downarrow v}$$

(OpRule)
$$\frac{}{\mathcal{E} \vdash \textbf{rule}\ \rho\ e \Downarrow \langle \rho, e, \mathcal{E}, \varnothing \rangle}$$

(OpInst)
$$\frac{\mathcal{E} \vdash e \Downarrow \langle \rho', e', \mathcal{E}', \eta' \rangle \quad \rho' = \forall\overline{\alpha}.\pi \Rightarrow \tau \quad \theta = [\overline{\alpha} \mapsto \overline{\tau}]}{\mathcal{E} \vdash e[\bar{\tau}] \Downarrow \langle \theta\pi \Rightarrow \theta\tau, \theta e', \mathcal{E}', \theta\eta' \rangle}$$

(OpRApp)
$$\frac{\mathcal{E} \vdash e \Downarrow \langle \rho', e', \mathcal{E}', \eta' \rangle \quad \mathcal{E} \vdash e_i \Downarrow v_i \quad (\forall e_i : \rho_i \in \eta') \quad \mathcal{E}'; \overline{\rho : v} \cup \eta' \vdash e' \Downarrow v}{\mathcal{E} \vdash e\ \textbf{with}\ \overline{e:\rho} \Downarrow v}$$

$$\boxed{\mathcal{E} \vdash_r \rho \Downarrow v}$$

(DynRes)
$$\frac{\mathcal{E}(\tau) = \langle \forall\overline{\beta}.\pi' \Rightarrow \tau', e', \mathcal{E}', \eta' \rangle \quad \rho = \forall\overline{\alpha}.\pi \Rightarrow \tau \quad \theta = \mathcal{U}_{\bar{\beta}}(\tau', \tau) \quad \mathcal{E} \vdash_r \rho_i \Downarrow v_i \quad (\forall \rho_i \in \theta\pi' - \pi)}{\mathcal{E} \vdash_r \rho \Downarrow \langle \rho, \theta e', \mathcal{E}', \overline{\rho : v} \cup \theta\eta' \rangle}$$

Type System

$$\boxed{\Gamma \vdash e : \tau}$$

(TyQuery)
$$\frac{\Gamma \vdash_r \rho}{\Gamma \vdash ?\rho : \rho}$$

(TyRule)
$$\frac{\rho = \forall\overline{\alpha}.\pi \Rightarrow \tau \quad \Gamma; \pi \vdash e : \tau}{\Gamma \vdash \textbf{rule}\ \rho\ e : \rho}$$

(TyInst)
$$\frac{\Gamma \vdash e : \forall\overline{\alpha}.\pi \Rightarrow \tau \quad \theta = [\overline{\alpha} \mapsto \overline{\tau}]}{\Gamma \vdash e[\bar{\tau}] : \theta\pi \Rightarrow \theta\tau}$$

(TyRApp)
$$\frac{\Gamma \vdash e : \bar{\rho} \Rightarrow \tau \quad \Gamma \vdash e_i : \rho_i \quad (\forall e_i : \rho_i \in \overline{e:\rho})}{\Gamma \vdash (e\ \textbf{with}\ \overline{e:\rho}) : \tau}$$

$$\boxed{\Gamma \vdash_r \rho}$$

(StaRes)
$$\frac{\Gamma(\tau) = \forall\overline{\beta}.\pi' \Rightarrow \tau' \quad \theta = \mathcal{U}_{\bar{\beta}}(\tau', \tau) \quad \Gamma \vdash_r \rho_i \quad (\forall \rho_i \in \pi'' = \theta\pi' - \pi)}{\Gamma \vdash_r \forall\overline{\alpha}.\pi \Rightarrow \tau}$$