An effect system for dynamic instances

Albert ten Napel

1 Basic effect handler effect system

The following system is a simplification of the effect system described by Bauer and Pretnar in [1]. Instances are removed and handlers can only handle a single effect. Handlers also should always handle every operations of an effect. In the effect annotations we will keep track of effect and not operations.

1.1 Syntax

We assume there is set of effect names $E = \{\varepsilon_1, ..., \varepsilon_n\}$. Each effect has a set of operation names $O_\varepsilon = \{op_1, ..., op_n\}$. We every operation name only corresponds to a single effect. Each operation has a parameter type τ^1_{op} and a return type τ^2_{op}. Annotations e are a subset of E.

$\tau ::=$ (value types)

$\quad ()$ (unit type)

$\quad \tau \rightarrow \underline{\tau}$ (type of functions)

$\quad \underline{\tau} \Rightarrow \underline{\tau}$ (type of handlers)

$\underline{\tau} ::=$ (computation types)

$\quad \tau \; ! \; e$ (annotated type)

$\nu ::=$ (values)

$\quad x, y, z, k$ (variables)

$\quad ()$ (unit value)

$\quad \lambda x.c$ (abstraction)

$\quad \mathit{handler} \, \{\mathit{return} \, x \rightarrow c, op_1(x; k) \rightarrow c, ..., op_n(x; k) \rightarrow c\}$ (handler)

$c ::=$ (computations)

$\quad \mathit{return} \, \nu$ (return value as computation)

$\quad \nu \; \nu$ (application)

$\quad x \leftarrow c; c$ (sequencing)

$\quad op(\nu; \lambda x.c)$ (operation call)

$\quad \mathit{with} \, \nu \, \mathit{handle} \, c$ (handler application)

1.2 Subtyping rules

$$\frac{}{() <: ()} \qquad \frac{a' <: a \qquad b <: b'}{a \rightarrow b <: a' \rightarrow b'}$$

$$\frac{a' <: a \qquad b <: b'}{a \Rightarrow b <: a' \Rightarrow b'} \qquad \frac{a <: a' \qquad e \subseteq e'}{a \; ! \; e <: a' \; ! \; e'}$$

The following rule says that we can add effects to both sides of a handler type as long as these effects are the same. This is necessary when applying a handler to a computation that has more effects than the handler handles.

With this subtyping rule we can simply extend the type knowing that the effects will appear on the right-hand side of the handler type and so the extra effects will not become lost.

$$\frac{}{a \,!\, e_1 \Rightarrow b \,!\, e_2 <: a \,!\, (e_1 \cup e) \Rightarrow b \,!\, (e_2 \cup e)}$$

1.3 Typing rules

For the typing rules there are two judgments, $\Gamma \vdash \nu : \tau$ for assigning types to values and $\Gamma \vdash c : \underline{\tau}$ for assigning computation types to computations. Γ stores bindings of variables to types.

$$\frac{\Gamma \vdash \nu : \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash \nu : \tau_2} \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{}{\Gamma \vdash () : ()} \qquad \frac{\Gamma, x : \tau_1 \vdash c : \underline{\tau}_2}{\Gamma \vdash \lambda x.c : \tau_1 \to \underline{\tau}_2}$$

In the following rule
$h = \mathit{handler}\,\{\mathit{return}\; x_r \to c_r, \mathit{op}_1(x_1; k_1) \to c_1, ..., \mathit{op}_n(x_n; k_n) \to c_n\}$.

$$\frac{O_\varepsilon = \{\mathit{op}_1, ..., \mathit{op}_n\} \quad \Gamma, x_r : \tau_1 \vdash c_r : \underline{\tau}_2 \quad \Gamma, x_i : \tau^1_{\mathit{op}_i}, k_i : \tau^2_{\mathit{op}_i} \to \underline{\tau}_2 \vdash c_i : \underline{\tau}_2}{\Gamma \vdash h : \tau_1 \,!\, \{\varepsilon\} \Rightarrow \underline{\tau}_2}$$

$$\frac{\Gamma \vdash c : \underline{\tau}_1 \quad \underline{\tau}_1 <: \underline{\tau}_2}{\Gamma \vdash c : \underline{\tau}_2} \qquad \frac{\Gamma \vdash \nu : \tau}{\Gamma \vdash \mathit{return}\; \nu : \tau \,!\, \varnothing}$$

$$\frac{\Gamma \vdash \nu_1 : \tau_1 \to \underline{\tau}_2 \quad \Gamma \vdash \nu_2 : \tau_1}{\Gamma \vdash \nu_1\, \nu_2 : \underline{\tau}_2} \qquad \frac{\Gamma \vdash c_1 : \tau_1 \,!\, e \quad \Gamma, x : \tau_1 \vdash c_2 : \tau_2 \,!\, e}{\Gamma \vdash x \leftarrow c_1; c_2 : \tau_2 \,!\, e} \qquad \frac{\Gamma \vdash \nu : \underline{\tau}_1 \Rightarrow \underline{\tau}_2 \quad \Gamma \vdash c : \underline{\tau}_1}{\Gamma \vdash \mathit{with}\; \nu\; \mathit{handle}\; c : \underline{\tau}_2}$$

$$\frac{\mathit{op} \in O_\varepsilon \quad \Gamma \vdash \nu : \tau^1_{\mathit{op}} \quad \Gamma, x : \tau^2_{\mathit{op}} \vdash c : \tau \,!\, e \quad \varepsilon \in e}{\Gamma \vdash \mathit{op}(\nu; \lambda x.c) : \tau \,!\, e}$$

2 Dynamic instances

2.1 Syntax

We extend the system from the previous section with dynamic instances. We add instances variables to the types. Annotation r in this system are sets of instance variables. We add existential quantifiers to the computation types. Handlers now take an instance as argument. Operations are called on instances. We add a new computation to create fresh instances.

$$
\begin{array}{lr}
\tau ::= ... & \text{(extended value types)} \\
\qquad i, j, k & \text{(instance variables)} \\
\underline{\tau} ::= & \text{(computation types)} \\
\qquad \tau \,!\, r & \text{(annotated type)} \\
\qquad \exists (i : \varepsilon).\underline{\tau} & \text{(existential)} \\
\nu ::= & \text{(updated values)} \\
\qquad \mathit{handler}(x)\ \{\mathit{return}\ x \to c, \mathit{op}_1(x; k) \to c, ..., \mathit{op}_n(x; k) \to c\} & \text{(handler)} \\
c ::= ... & \text{(updated/extended computations)} \\
\qquad x\#\mathit{op}(\nu; \lambda y.c) & \text{(operation call)} \\
\qquad \mathit{new}\ \varepsilon & \text{(instance creation)}
\end{array}
$$

2.2 Subtyping

$$
\frac{}{i <: i} \qquad\qquad \frac{a <: b}{\exists (i : \varepsilon).a <: \exists (i : \varepsilon).b}
$$

3 Typing rules

We update the judgments to include an environment for the instance variables: $\Delta;\Gamma \vdash \nu : \tau$ and $\Delta;\Gamma \vdash \nu : \tau$. Δ contains bindings of instance variables to effects.

In the following rule $h = \mathit{handler}(x)\ \{\mathit{return}\ x_r \rightarrow c_r, op_1(x_1;k_1) \rightarrow c_1, ..., op_n(x_n;k_n) \rightarrow c_n\}$.

$$\frac{\begin{array}{l} O_\varepsilon = \{op_1, ..., op_n\} \\ \Delta;\Gamma \vdash x : i \\ \Delta \vdash i : \varepsilon \\ \Delta;\Gamma, x_r : \tau_1 \vdash c_r : \underline{\tau}_2 \\ \Delta;\Gamma, x_i : \tau^1_{op_i}, k_i : \tau^2_{op_i} \rightarrow \underline{\tau}_2 \vdash c_i : \underline{\tau}_2 \end{array}}{\Delta;\Gamma \vdash h : \tau_1 \mathbin{!} \{i\} \Rightarrow \underline{\tau}_2}$$

$$\frac{\begin{array}{l} op \in O_\varepsilon \\ \Delta;\Gamma \vdash x : i \\ \Delta \vdash i : \varepsilon \\ \Delta;\Gamma \vdash \nu : \tau^1_{op} \\ \Delta;\Gamma, y : \tau^2_{op} \vdash c : \tau \mathbin{!} r \\ i \in r \end{array}}{\Delta;\Gamma \vdash x\#op(\nu; \lambda y.c) : \tau \mathbin{!} r} \qquad \frac{i \text{ is a fresh instance variable}}{\Delta;\Gamma \vdash \mathit{new}\,\varepsilon : i \mathbin{!} \varnothing}$$

$$\frac{\Delta, i : \varepsilon;\Gamma \vdash c : \underline{\tau}}{\Delta;\Gamma \vdash c : \exists(i : \varepsilon).\underline{\tau}} \qquad \frac{\Delta;\Gamma \vdash c : \exists(i : \varepsilon).\underline{\tau}}{\Delta;\Gamma \vdash c : \underline{\tau}}$$

4 Example

Given a boolean type *Bool* and an effect *Flip* with one operation $\mathit{flip} : () \rightarrow \mathit{Bool}$. The following short example results in the derivations below.

```
\() ->
  x <- new Flip;
  x#flip ()
```

$$\frac{\cdot; u : () \vdash x \leftarrow \mathit{new}\ \mathit{Flip}; x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \exists(i : \mathit{Flip}).\mathit{Bool} \mathbin{!} \{i\}}{\cdot;\cdot \vdash \lambda u.x \leftarrow \mathit{new}\ \mathit{Flip}; x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : () \rightarrow \exists(i : \mathit{Flip}).\mathit{Bool} \mathbin{!} \{i\}}$$

$$\frac{i : \mathit{Flip}; u : () \vdash x \leftarrow \mathit{new}\ \mathit{Flip}; x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \mathit{Bool} \mathbin{!} \{i\}}{\cdot; u : () \vdash x \leftarrow \mathit{new}\ \mathit{Flip}; x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \exists(i : \mathit{Flip}).\mathit{Bool} \mathbin{!} \{i\}}$$

$$\frac{\begin{array}{l} i : \mathit{Flip}; u : () \vdash \mathit{new}\ \mathit{Flip} : i \mathbin{!} \{i\} \\ i : \mathit{Flip}; u : (), x : i \vdash x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \mathit{Bool} \mathbin{!} \{i\} \end{array}}{i : \mathit{Flip}; u : () \vdash x \leftarrow \mathit{new}\ \mathit{Flip}; x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \mathit{Bool} \mathbin{!} \{i\}}$$

$$\frac{\begin{array}{l} i : \mathit{Flip}; u : () \vdash \mathit{new}\ \mathit{Flip} : i \mathbin{!} \varnothing \\ i \mathbin{!} \varnothing <: i \mathbin{!} \{i\} \end{array}}{i : \mathit{Flip}; u : () \vdash \mathit{new}\ \mathit{Flip} : i \mathbin{!} \{i\}}$$

$$\frac{\begin{array}{l} i : \mathit{Flip}; u : (), x : i \vdash x : i \\ i : \mathit{Flip} \vdash i : \mathit{Flip} \\ i : \mathit{Flip}; u : (), x : i \vdash () : () \\ i : \mathit{Flip}; u : (), x : i, y : \mathit{Bool} \vdash \mathit{return}\ y : \mathit{Bool} \mathbin{!} \{i\} \end{array}}{i : \mathit{Flip}; u : (), x : i \vdash x\#\mathit{flip}((); \lambda y.\mathit{return}\ y) : \mathit{Bool} \mathbin{!} \{i\}}$$

$$\frac{\begin{array}{l} i : \mathit{Flip}; u : (), x : i, y : \mathit{Bool} \vdash \mathit{return}\ y : \mathit{Bool} \mathbin{!} \varnothing \\ \mathit{Bool} \mathbin{!} \varnothing <: \mathit{Bool} \mathbin{!} \{i\} \end{array}}{i : \mathit{Flip}; u : (), x : i, y : \mathit{Bool} \vdash \mathit{return}\ y : \mathit{Bool} \mathbin{!} \{i\}}$$

Similarly the following program:

```
\() ->
  inst <- new Exception;
  (
    \x -> if x == 0 then inst#throw () else x,
    \f -> handler(inst) { throw () k -> f () }
  )
:
() -> exists (i : Exception).
  (
    Int -> Int!{i},
    (() -> t!e) -> (Int!{i} => t!e)
  )
```

5 Alternative rules for existentials

Another idea is to have *new* ε return an existential type and to unpack existentials only when sequencing. If the instance is mentioned in the return type of the sequencing we have to repack the existential.

$$\frac{}{\Delta; \Gamma \vdash \mathit{new}\, \varepsilon : \exists (i : \varepsilon). i \mathrel{!} \varnothing}$$

$$\frac{\begin{array}{l} \Delta; \Gamma \vdash c_1 : \exists (i : \varepsilon). \tau_1 \mathrel{!} e \\ \Delta, i : \varepsilon; \Gamma, x : \tau_1 \vdash c_2 : \tau_2 \mathrel{!} e \\ i \notin \tau_2 \end{array}}{\Delta; \Gamma \vdash x \leftarrow c_1; c_2 : \tau_2 \mathrel{!} e} \qquad \frac{\begin{array}{l} \Delta; \Gamma \vdash c_1 : \exists (i : \varepsilon). \tau_1 \mathrel{!} e \\ \Delta, i : \varepsilon; \Gamma, x : \tau_1 \vdash c_2 : \tau_2 \mathrel{!} e \\ i \in \tau_2 \end{array}}{\Delta; \Gamma \vdash x \leftarrow c_1; c_2 : \exists (i : \varepsilon). \tau_2 \mathrel{!} e}$$

References

[1] Bauer, Andrej, and Matija Pretnar. "An effect system for algebraic effects and handlers." International Conference on Algebra and Coalgebra in Computer Science. Springer, Berlin, Heidelberg, 2013.