‘Monads and Effects’

@myuon

May 25, 2019

About me

Twitter: @myuon_myon

Figure 1: Kawaii

Table of Contents

Table of Contents

What's the Effect?

$$\Gamma \vdash M : A \,!\, e$$

- Γ: Context
- M: Term
- A: Type
- e: effect

Idea: "The type system estimates the effect of computation M to be (at most) e"

What's the Effect? (Observation)

What effects are potentially caused during the execution?

$\vdash \langle M_1, M_2 \rangle : A \times B$

$\vdash \textbf{let } x = M \textbf{ in } N : B$

$\vdash \lambda x.\, M : A \to B$

$\vdash (\lambda x.\, M) N : B$

What's the Effect? (Answer-1)

Let $(\cdot)$ be an "accumulating" operator (or an effect product).

$\vdash \langle M_1^{e_1}, M_2^{e_2} \rangle : A \times B \mathrel{!} e_1 \cdot e_2$

$\vdash \textbf{let } x = M^{e_1} \textbf{ in } N^{e_2} : B \mathrel{!} e_1 \cdot e_2$

Observation: effects can be stacked

We will discuss the others later.

Memory-state Effect

Effect $J \subseteq \texttt{Addr}$ means the computation contains I "memory" effect (at most).

- $\vdash \texttt{read}_i() : \texttt{nat} \,!\, \{i\}$
- $n : \texttt{nat} \vdash \texttt{write}_i(n) : 1 \,!\, \{i\}$

Example

$\vdash \textbf{let}\ x = \texttt{read}_i()\ \textbf{in}\ \texttt{write}_j(x)\ :\ 1 \,!\, \{i, j\}$

Existing Effect System

- Koka
- Eff
- Frank
- Multicore OCaml
- and more. . .

What are the effects of those languages?

Table of Contents

Koka

Deveoped by Daan Leijen in Microsoft Research since 2012.

> *Koka is a function-oriented programming language that seperates pure values from side-effecting computations, where the effect of every function is automatically inferred.*

Koka Base Effects

Computation in Real-world are decomposed into effects:

- `exn`: exception
- `div`: divergence (infinite loop)
- `ndet`: non-determinism (random value)
- `alloc(h)`, `read(h)`, `write(h)`: memory operation (h for some heap)

In Koka, effect order does not matter.

Koka Effect Aliases

- $\texttt{total} \equiv \langle\rangle$
- $\texttt{pure} \equiv \langle\texttt{exn}, \texttt{div}\rangle$
- $\texttt{st(h)} \equiv \langle\texttt{alloc}(h), \texttt{read}(h), \texttt{write}(h)\rangle$
- $\texttt{io} \equiv \langle\texttt{st}(io), \texttt{pure}, \texttt{ndet}\rangle$

Example

- $\texttt{print} : \texttt{string} \xrightarrow{\texttt{io}} 1$
- $\texttt{error} : \forall\alpha.\, \texttt{string} \xrightarrow{\texttt{exn}} \alpha$
- $(:=) : \forall\alpha.\, (\texttt{ref}(h, a), a) \xrightarrow{\texttt{write}(h)} 1$

Koka Syntax* (kind)

kinds:

- $*$: type of values
- e: effect rows
- k: effect constraints
- h: heap
- $\_ \to \_$: constructor

Koka Syntax* (type)

type:

- α: type variable
- $()$: unit type
- $\_ \xrightarrow{e} \_$: function type with effects
- $\mathtt{ref}(h, *)$: reference

Koka Effect Rows*

"Row-polymorphic" effect: $\langle \mathit{exn} \mid \mu \rangle$

Example)

$$\texttt{catch} : \forall \alpha \mu.\, (1 \xrightarrow{\langle \texttt{exn} | \mu \rangle} \alpha, \texttt{exception} \xrightarrow{\mu} \alpha) \xrightarrow{\mu} \alpha$$

$$\text{(EQ-REFL)}\ \varepsilon \equiv \varepsilon \qquad \text{(EQ-TRANS)}\ \frac{\varepsilon_1 \equiv \varepsilon_2 \quad \varepsilon_2 \equiv \varepsilon_3}{\varepsilon_1 \equiv \varepsilon_3}$$

$$\text{(EQ-HEAD)}\ \frac{\varepsilon_1 \equiv \varepsilon_2}{\langle l \,|\, \varepsilon_1 \rangle \equiv \langle l \,|\, \varepsilon_2 \rangle} \qquad \text{(EQ-SWAP)}\ \frac{l_1 \not\equiv l_2}{\langle l_1 \,|\, \langle l_2 \,|\, \varepsilon \rangle \rangle \equiv \langle l_2 \,|\, \langle l_1 \,|\, \varepsilon \rangle \rangle}$$

Figure 2: effect equivalence

Koka ST Types

$$(\text{Alloc})\ \Gamma \vdash \texttt{ref} : \tau \xrightarrow{\langle \texttt{st}(h) | \varepsilon \rangle} \texttt{ref}(h, \tau) \,!\, \varepsilon'$$

$$(\text{Read})\ \Gamma \vdash (!) : \texttt{ref}(h, \tau) \xrightarrow{\langle \texttt{st}(h) | \varepsilon \rangle} \tau \,!\, \varepsilon'$$

$$(\text{Write})\ \Gamma \vdash (:=) : (\texttt{ref}(h, \tau), \tau) \xrightarrow{\langle \texttt{st}(h) | \varepsilon \rangle} 1 \,!\, \varepsilon'$$

$$(\text{Run})\ \Gamma \vdash e : \tau \,!\, \langle \texttt{st}(h) \mid \varepsilon \rangle \Longrightarrow \Gamma \vdash \texttt{run}(e) : \tau \,!\, \varepsilon$$

Koka ST Types (Fib Example)

```
function fib(n: int) {
  val x = ref(0); val y = ref(1);
  repeat(n) {
    val y0 = !y;
    y := !x + !y;
    x := y0;
  }
  !x
}
```

$$\vdash \mathtt{fib} : \mathtt{int} \xrightarrow{\mathrm{st}(h)} \mathtt{int} \,!\, \emptyset$$

Koka Effects** (Effect Duplication)

Allows effect duplication: $\langle \texttt{exn}, \texttt{exn} \rangle \not\equiv \langle \texttt{exn} \rangle$

Thanks to effect equivalence and the effect duplication, Koka can solve the following equation.

$$\langle \texttt{exn} \mid \mu \rangle \equiv \langle \texttt{exn} \rangle \implies \mu \equiv \langle \rangle$$

Table of Contents

What's the Effect? (Answer-2)

$$\frac{\Gamma, x : A \vdash M : B \mathbin{!} e}{\Gamma \vdash \lambda x.\, M : A \xrightarrow{e} B \mathbin{!} \emptyset}$$

Figure 3: abstraction

$$\frac{\Gamma \vdash M_1 : A \xrightarrow{e} B \mathbin{!} e_1 \qquad \Gamma \vdash M_2 : A \mathbin{!} e_2}{\Gamma \vdash M_1\, M_2 : B \mathbin{!} e_1 \cdot e_2 \cdot e}$$

Figure 4: application

Laws in Effect System (Monad)

Effect can be viewed as "Parametric" computation.
$\Longrightarrow$ So does the laws?

Monad laws:

- (assoc) $(\textbf{let } x = M \textbf{ in } (\textbf{let } y = N \textbf{ in } L)) \equiv (\textbf{let } y = (\textbf{let } x = M \textbf{ in } N) \textbf{ in } L$
- (left id) $(\textbf{let } x = x \textbf{ in } M) \equiv M$
- (right id) $(\textbf{let } x = M \textbf{ in } x) \equiv M$

Laws in Effect System**

For ($\textbf{let}\ x = x\ \textbf{in}\ M^e) \equiv M^e$, we should assume:

- Variable x has no effect, namely $\emptyset$
- Right-hand side, the effect is e
- Left-hand side, the effect should be $\emptyset \cdot e$

We should have left identity law of effect: $\emptyset \cdot e = e$

Inevitable Over-estimation*

Effect system estimates its effect of the computation.

$\Longrightarrow$ Compiler can always estimate the correct effect?

$\Longrightarrow$ The answer is “No”

$$\vdash (\textbf{if}\ x = 0\ \textbf{then}\ \texttt{raise}\ \text{"error"}\ \textbf{else}\ x) : \texttt{nat} \mathbin{!} \{\texttt{exn}\}$$

$$\frac{\Gamma \vdash M : A \mathbin{!} e \qquad e \leq e'}{\Gamma \vdash M : A \mathbin{!} e'}$$

Figure 5: (cast)

Structure in Effects

Let $\mathbb{E}$ be an effect set.

- Have an unit $\emptyset \in \mathbb{E}$
- Have a binary operator $(\cdot) : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$
- $\langle \mathbb{E}, (\cdot), \emptyset \rangle$ is a monoid
- Have a preorder $(\leq) \subseteq \mathbb{E} \times \mathbb{E}$
- The (monoid/preorder) operators are compatible

Memory-state Monad

Monad $T(-)$ means the computation contains some "memory" effect. (like `IO` in Haskell)

We have memory operations:

- $\vdash \mathtt{read}_i() : T\mathtt{nat}$
- $n : \mathtt{nat} \vdash \mathtt{write}_i(n) : T1$

Memory-state Effects Structure

Let `Addr` be a set of addresses. In this example, an effect set is $\mathcal{P}(\texttt{Addr})$.

- $\emptyset$: The emptyset
- $(\cdot) = (\cup)$: The union of sets
- $(\leq) = (\subseteq)$: Set inclusion

“Parametric” Memory-state (Semantics)

Let $\mathtt{Mem}(J)$ be the memory state which the address set J might be modified from the initial state.

$$T_I(A) = \forall J \in \mathcal{P}(\mathtt{Addr}).\, \mathtt{Mem}(J) \to (\mathtt{Mem}(I \cup J) \times A)$$

Interpret $\Gamma \vdash M \ : \ A \,!\, e$ as $\overline{M} : \overline{\Gamma} \to T_e(\overline{A})$

- $\mathtt{read}_i : 1 \to T_{\{i\}}\mathtt{nat}$
- $\mathtt{write}_i : \mathtt{nat} \to T_{\{i\}}1$

Further topics

- Semantics
- Graded Monad (for the formal semantics)
- Recursion
- Algebraic Effects and Effect Handlers
- Effect Inference
- Polymorphic Effect

Recursion**

$$\vdash \mathtt{map} : (A \xrightarrow{e} B) \to (\mathtt{List}(A) \xrightarrow{???} \mathtt{List}(B))$$

Recursive Effects**

$$\vdash \texttt{map} : (A \xrightarrow{e} B) \to (\texttt{List}(A) \xrightarrow{???} \texttt{List}(B))$$

Simple idea

Introduce (possibly many-times) multiplied effect, $e^{\dagger}$.
Then, how do we give the semantics of the fixpoint?

$$\texttt{mfix} : (A \xrightarrow{e} A) \xrightarrow{e^{\dagger}} A$$

References

- Daan Leijen. 2016. “Koka: Programming with Row-polymorphic Effect Types”
- Shin-ya Katsumata. 2014. “Parametric effect monads and semantics of effect systems”
- Andrej Bauer and Matija Pretnar. 2012. “Programming with Algebraic Effects and Handlers”
- Sam Lindley, Conor McBride, Craig McLaughlin. 2016. “Do Be Do Be Do”