Computer Laboratory University of Cambridge

Coeffects: Programming languages for rich environments

Tomas Petricek with **Dominic Orchard** and **Alan Mycroft**
University of Cambridge, {name.surname}@cl.cam.ac.uk

Motivation: Modern software challenges

- Applications today run in diverse environments, such as mobile phones or the cloud. Different environments provide different capabilities, data with meta-data or other resources.
- Applications access information and resources of the environment. Such context-dependent interactions are often more important than how the application affects or changes the environment.
- Tracking and verifying how computations affect the environment can be done in a unified way using monadic effect systems, but no such mechanism exists for tracking and verifying how computations access and rely on the context.

Example: Mobile online store application

```
let validate(input) =
  (input ≠ null) && (input.ForAll(isLetter))

let displayProduct (name) =
  if validate(name) then
    let product = lookup(name, access products)
    generateProductPage(product)
  else generateEmptyPage()
```

During compilation, we want to infer what environment capabilities the application requires and check that it will use them correctly:

- **Cross-platform and versioning.** The `ForAll` library function used in `validate` is only available when compiling program to .NET or JVM, but cannot be translated to SQL and executed in database.
- **Tracking resource usage.** The construct `access product` obtains connection to a database of products, thus `displayProducts` can be only executed on the server node, running in the cloud.
- **Provenance and security.** For security and auditing purposes, we want to know how data flow through the program. For example, the result of `displayProduct` relies only the argument and the database.

Effect systems

$$\Gamma \vdash e : \tau \,\&\, \sigma$$

- Track or infer information about what the computation *does* to the environment
- Information σ, such as set of performed memory operations, attached to the result
- Propagate information forward to the overall result
- Modeled as morphisms $\alpha \to \mathcal{C}\beta$ where $\mathcal{C}$ is a monad

Coeffect systems

$$\Gamma\ @\ \sigma \vdash e : \tau$$

- Track or infer information about what the computation *requires* from the environment
- Information σ, such as set of accessed resources, attached to the variable context
- Propagate information backward to the initial input
- Modeled as morphisms $\mathcal{D}\alpha \to \beta$ where $\mathcal{D}$ is a comonad

Flat coeffects for distributed programming

Use monoid of resource names with union to track required resources.

```
let getCategory(name) =
  let id = lookupProduct(name, access products)
  lookupProductCategory(id, access categories)
```

Required resources `{products,categories}` are split between the scope where a function is declared and the scope where it is called (using Δ).

Structural coeffects for provenance tracking

For every variable, the context records whether its value is allowed to come from an untrusted source, such as user input.

```
let storeUser name password salt =
  writeUser(user, hashKey(password, salt, serverKey))
```

The last two parameters of `hashKey` must be secure thus the last argument of `storeUser` and global variable `serverKey` cannot depend on user input.

Coeffect systems

Core calculus for tracking context-dependence that can be used as basis for type systems and semantics of context-dependent computations.

Flat coeffect types

Uses annotations that form a symmetric, idempotent monoid (R, ⊕, 0) with operation Δ that represents splitting of requirements in a lambda abstraction.

$$\frac{C^r \Gamma \vdash e_1 : C^t \tau_1 \to \tau_2 \qquad C^s \Gamma \vdash e_2 : \tau_1}{C^{r \oplus s \oplus t} \Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{C^t\ (\Gamma, x : \tau_1) \vdash e : \tau_2 \qquad \Delta t = (r, s)}{C^s \Gamma \vdash \lambda x.\, e\ : C^r \tau_1 \to \tau_2}$$

Structural coeffect types

Generalization that captures fine-grained structure with information corresponding to variables. Uses ring-like structure (R, ⊗, ⊕, 0). Structural rules specify how variable manipulation affects the context structure.

$$\frac{C^r \Gamma_1 \vdash e_1 : C^t \tau_1 \to \tau_2 \qquad C^s \Gamma_2 \vdash e_2 : \tau_1}{C^{r \times (t \otimes s)} (\Gamma_1, \Gamma_2) \vdash e_1\ e_2 : \tau_2}$$

$$\frac{C^{r \otimes s} (x : \tau, y : \tau) \vdash e : \tau_1}{C^{r \oplus s} (z : \tau) \vdash e[z/x][z/y] : \tau_1}$$

UNIVERSITY OF CAMBRIDGE