Coeffects: The essence of context dependence

Tomas Petricek[1], supervisor: Alan Mycroft[1]

1 University of Cambridge, 15 JJ Thomson Avenue, CB3 0FD, UK
tomas.petricek@cl.cam.ac.uk, alan.mycroft@cl.cam.ac.uk

1. Motivation

Modern applications run in diverse environments, such as mobile phones or the cloud. Different environments provide capabilities to perform operations, retrieve data or to use environment's resources. Some also provide meta-data about quality, provenance and security. Understanding how applications *depend* on contextual information is often more important than understanding how they *affect* the environment.

Tracking interactions with the environment can be done in a unified way using effect systems, but no such unified mechanisms exist for tracking how programs depend on the context. As a result, tracking of context-dependence is often done in an ad-hoc fashion.

```
let validate(input) =
    (input ≠ null) && (input.ForAll(isLetter))
let displayProduct(name) =
    if validate(name) then
        let product = lookup(name, access ProductsDb)
        generateProductPage(product)
    else generateErrorPage()
```

Listing 1. Generating web page in an online store application

In Listing 1 we would like to check the following properties at compile time:

- **Cross-platform and versioning.** The `ForAll` function is only available when running on .NET or JVM, but cannot be translated to SQL and executed as database code.
- **Tracking resource usage.** The construct `access product` connects to a database and so `displayProducts` can only execute on a server in the cloud.
- **Provenance and security.** For auditing purposes, we want to track provenance to know that the result of `displayProduct` relies only on the given input and database data.

2. Background

Effect systems introduced by Gifford and Lucasen (1986) track actions performed by computations, such as memory operations or communication. The judgments are of a form $\Gamma \vdash e : \tau, \sigma$, associating effects σ with the result. Moggi (1991) models the semantics of effectful computations as *monadic* computations $A \to M$ and Wadler with Thiemann (2003) relate the two concepts.

Conversely, typing judgements of context-dependent properties (or *coeffects*) have a form $\Gamma, \sigma \vdash e : \tau$, associating context information σ with free variables Γ. Syntactically, coeffects are similar to effects (i.e. Murphy, Cray and Harper (2008) use effect syntax to track coeffect properties), but there is a number of differences. Coeffects support both call-by-name and call-by-value, while effects require explicit evaluation order and they can be also used to track context information about individual variables.

Semantically, effects and coeffects differ significantly. Uustalu and Vene (2008) show that context-dependent programs can be modelled as *comonadic* computations $CA \to B$. Our work extends their semantics in a number of ways discussed later.

3. Approach

We develop *coeffect calculus* to capture context-dependence properties. Although a number of existing systems fit with the model, no unified systems have been presented so far:

- Our *flat calculus* is syntactically similar to effect systems. It tracks single information about the entire context. In distributed language the information would be the set of required resources.
- Our *structural calculus* generalizes the flat calculus and captures more fine-grained structure. It tracks information about individual variables in the variable context. When tracking provenance, the information describe data sources that can influence the value of the variable.

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

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

Figure 1a. Application and abstraction of the *flat calculus*

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

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

Figure 1b. Application and contraction of the *structural calculus*

Figure 1 shows interesting typing rules. The *flat calculus* (Figure 1a) uses tags of a semi-lattice $(S, \vee)$. Variable contexts and domain of functions are annotated with a tag (written $C^r\Gamma$ and $C^r\tau_1 \to \tau_2$, respectively) to denote the context requirements. Application is typeable in a context that satisfies requirements of the two expressions and the requirements of the function. Lambda abstraction splits the requirements of the body between the declaring context and the function (i.e. resources can be provided by both declaration and use site). This is crucial for supporting both call-by-name and call-by-value.

To track more fine-grained calculus, the *structural calculus* (Figure 1b) mirrors the structure of the variable context Γ in the tag using $\otimes$. Information associated with individual variables can be merged using the $\oplus$ operation. The contraction rule combines information about two individual variables r and s into information $r \oplus s$ associated with a single variable. The application rule combines information about the first part of the context (r corresponding to Γ_1) with an information $t \oplus s$, which specifies that all variables from Γ_2 (tagged with s) may affect the input of the function e_1 (tagged with t).

4. Contributions

There is an increasing need for capturing how computations depend on the context in which they execute. Examples from the literature include tracking of security information, provenance, resources or executing nodes in distributed programs, but all of the above are single-purpose mechanisms.

- We present a unified calculus for tracking context-dependence, providing a counterpart to the well-known effect systems. The calculus can be used *syntactically*, as a basis for type system, and *semantically*, to give comonadic semantics of a language.
- We identify the information structures that can be tracked using coeffects. We use the *flat calculus* to track dynamically scoped parameters, resources and execution environments. We use the *structural calculus* to track security and provenance information.
- We extend the comonadic semantic of Uustalu and Vene (2008) to use tagged comonads and to accommodate sub-coeffecting and fine-grained tracking of information in our structural calculus.

References

D. K. Gifford and J. M. Lucassen (1986). Integrating Functional and Imperative Programming. In Proceedings of Conference on LISP and Functional Programming, 1986. ISBN 0897912004.

E. Moggi (1991). Notions of Computation and Monads. Information Computation, Vol 93: 55–92. July 1991. ISSN 0890-5401

P.Wadler and P. Thiemann (2003). The Marriage of Effects and Monads. ACM Transactions on Computational Logic, Vol 4:1–32, January 2003.

T. Murphy, VII., K. Crary, and R. Harper (2008). Type-safe distributed programming with ML5. In Proceedings of Conference on Trustworthy Global Computing, 108–123, 2008.

T. Uustalu and V. Vene (2008). Comonadic Notions of Computation. In Electronic Notes in Theoretical Computer Science. Vol 203:263–284, June 2008. ISSN 1571-0661.