Università di Genova
DIPARTIMENTO INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Computer Science Workshop
PhD program in Computer Science and Systems Engineering

A Java-like calculus with user-defined coeffects

Riccardo Bianchini, Francesco Dagnino, Paola Giannini, Elena Zucca

Abstract

Modern applications are thought to be resource-aware, so it is very useful to focus on the concept of resource and to keep track of the use of them. *Coeffect systems* provide a static control capable to guarantee interesting properties on the usage of tracked objects. Our goal is to develop a Java-like calculus where declared variables can be annotated by *coeffects* specifying constraints on their use, such as linearity or security levels. Such annotations are written in the language itself, as expressions of type `Coeffect`, a predefined class which can be extended by user-defined subclasses, modeling coeffects desired for a specific application. We will also show a simple example of coeffect system checking the linear use of a variable.

Coeffects

Coeffect systems are, in a sense, the dual of effect systems. The latter track *how the program modifies the environment*, coeffect systems *what the program requires from the context* of a computation. There are two kinds of coeffect systems:

- *Structural* coeffects annotate each variable in the context independently
- *Flat* coeffects annotates the whole context

Here we consider structural coeffects.

Structure of coeffects

Coeffects are assumed to form a *semiring*, that is, a tuple $(\mathcal{C}, +, 0, \times, 1)$ such that

- $(\mathcal{C}, +, 0)$ is a commutative monoid
- $(\mathcal{C}, \times, 1)$ is a monoid
- Given c in $\mathcal{C}$
  - $0 \times c = c \times 0 = 0$
- Given c_1, c_2, c_3 in $\mathcal{C}$
  - $c_1 \times (c_2 + c_3) = (c_1 \times c_2) + (c_1 \times c_3)$
  - $(c_1 + c_2) \times c_3 = (c_1 \times c_3) + (c_2 \times c_3)$

Coeffect annotations

Our Java-like calculus supports *user-defined* structural coeffects. That is, coeffect annotations are values of (subclasses of) a predefined class `Coeffect`, analogously to Java exceptions which are expressions of (subclasses of) `Exception`.

Coeffects can be seen as costraints on the use of declared variables, so with user-defined coeffects we can impose user-defined constraints. Metavariable $\hat{v}$ is used for used-defined coeffects, that is, values expected to be of a subclass of `Coeffect`.

$$\{\, T[\hat{v}]\ x = e;\ e' \}$$

This syntax imposes the constraint represented by $\hat{v}$ on the use of x in e'.

Subclasses of `Coeffect` should implement methods:

- `sup`, corresponding to a $\vee$ operator that induces an order relation among coeffects
- `sum` and `mult`, corresponding respectively to the $+$ and $\times$ operators of the semiring
- `zero` and `one`, returning a new `Coeffect` object corresponding respectively to 0 and 1

Properties of user-defined coeffects

To guarantee preservation of coeffects during execution, operators determined by user-defined methods have to respect some equations, notably:

Given coeffects $\hat{v}_1, \hat{v}_2, \hat{v}_3$:

1. $\hat{v}_1 + \hat{v}_2 = \hat{v}_2 + \hat{v}_1$
2. $\hat{v}_1 + (\hat{v}_2 + \hat{v}_3) = (\hat{v}_1 + \hat{v}_2) + \hat{v}_3$
3. $\hat{v}_1 + 0 = \hat{v}_1$
4. $\hat{v}_1 \times 1 = 1 \times \hat{v}_1 = \hat{v}_1$
5. $\hat{v}_1 \times 0 = 0 \times \hat{v}_1 = 0$
6. $\hat{v}_1 \times (\hat{v}_2 \times \hat{v}_3) = (\hat{v}_1 \times \hat{v}_2) \times \hat{v}_3$
7. $\hat{v}_1 \times (\hat{v}_2 + \hat{v}_3) = \hat{v}_1 \times \hat{v}_2 + \hat{v}_1 \times \hat{v}_3$
8. $(\hat{v}_1 + \hat{v}_2) \times \hat{v}_3 = \hat{v}_1 \times \hat{v}_3 + \hat{v}_2 \times \hat{v}_3$
9. $\hat{v}_1 \vee \hat{v}_1 = \hat{v}_1$
10. $\hat{v}_1 \vee (\hat{v}_2 \vee \hat{v}_3) = (\hat{v}_1 \vee \hat{v}_2) \vee \hat{v}_3$
11. $\hat{v}_1 \vee \hat{v}_2 = \hat{v}_2 \vee \hat{v}_1$
12. $\hat{v}_1 \vee (\hat{v}_1 + \hat{v}_2) = (\hat{v}_1 + \hat{v}_2)$
13. $\hat{v}_1 + (\hat{v}_2 \vee \hat{v}_3) = (\hat{v}_1 + \hat{v}_2) \vee (\hat{v}_1 + \hat{v}_3)$
14. $\hat{v}_1 \times (\hat{v}_2 \vee \hat{v}_3) = (\hat{v}_1 \times \hat{v}_2) \vee (\hat{v}_1 \times \hat{v}_3)$
15. $(\hat{v}_2 \vee \hat{v}_3) \times \hat{v}_1 = (\hat{v}_2 \times \hat{v}_1) \vee (\hat{v}_3 \times \hat{v}_1)$

An example: 0, 1, ω coeffects

In this simple example, coeffect `new Zero()` is meant to be assigned to unused variables, `new One()` to variables used linearly (exactly once), `new Omega()` to unrestricted variables. Class `Linearity` is used to define methods `zero` and `one` only once.

```
class Linearity extends Coeffect{
  Coeffect zero(){ new Zero()}
  Coeffect one(){new One()}
}

class Zero extends Linearity{
  Coeffect sup(Coeffect c) {
    case c of
      (Linearity x) x
      (Coeffect x) new Coeffect()
  }

  Coeffect sum(Coeffect c) {
    case c of
      (Linearity x) x
      (Coeffect x) new Coeffect()
  }
  Coeffect mult(Coeffect c) {
    case c of
      (Linearity x) new Zero()
      (Coeffect x) new Coeffect()
  }
}
```

```
class One extends Linearity{
  Coeffect sup(Coeffect c) {
    case c of
      (Zero x) new One()
      (One x) new One()
      (Omega x) new Omega()
      (Coeffect x)
        new Coeffect()}
  }
  Coeffect sum(Coeffect c) {
    case c of
      (Zero x) new One()
      (One x) new Omega()
      (Omega x) new Omega()
      (Coeffect x)
        new Coeffect()
  }
  Coeffect mult(Coeffect c) {
    case c of
      (Linearity x) x
      (Coeffect x)
        new Coeffect()
  }
}
```

```
class Omega extends Linearity{
  Coeffect sup(Coeffect c) {
    case c of
      (Linearity x) new Omega()
      (Coeffect x) new Coeffect()
  }
  Coeffect sum(Coeffect c) {
    case c of
      (Linearity x) new Omega()
      (Coeffect x) new Coeffect()
  }
  Coeffect mult(Coeffect c) {
    case c of
      (Zero x) new Zero()
      (One x) new Omega()
      (Omega x) new Omega()
      (Coeffect x)
        new Coeffect()
  }
}
```

Soundness result

Execution preserves types and coeffects assuming that user-defined coeffects guarantee conditions 1-15

Future goals

- Adding *graded modal types*
- Allowing a "global" annotation in a method's signature
- Allowing variables in coeffect annotations

CONTACTS

Riccardo Bianchini
riccardo.bianchini@edu.unige.it

Paola Giannini
paola.giannini@uniupo.it

Francesco Dagnino
francesco.dagnino@dibris.unige.it

Elena Zucca
elena.zucca@unige.it

CSW