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

Computer Science Workshop
PhD program in Computer Science and Systems Engineering

Sharing coeffects for Java-like languages

Riccardo Bianchini

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 coeffect system to track the sharing among objects in memory and to express interesting properties of references, such as the *capsule* property.

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

Each variable in the context is annotated independently. For instance, it is possible to express that a variable is used a certain number of times by a structural coeffect marking each variable in the context with the corresponding number.

Flat coeffects

The whole context is annotated. For instance, a flat coeffect can be used in Haskell to keep track of implicit parameters that are required by an expression (and their types).

Structure of coeffects

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

- $(\mathcal{C}, \oplus, \mathbf{0})$ is a commutative monoid.
- $(\mathcal{C}, \otimes, \mathbf{1})$ is a monoid.
- Given c_1, c_2, c_3 in $\mathcal{C}$
  - $c_1 \otimes (c_2 \oplus c_3) = (c_1 \otimes c_2) \oplus (c_1 \otimes c_3)$.
  - $(c_1 \oplus c_2) \otimes c_3 = (c_1 \otimes c_3) \oplus (c_2 \otimes c_3)$.
- Given c in $\mathcal{C}$
  - $\mathbf{0} \otimes c = c \otimes \mathbf{0} = \mathbf{0}$.

A simple example of coeffect system

This coeffect system for the simply-typed lambda calculus checks how many times variables are used.

$$\begin{array}{lcl} t & ::= & x \mid \lambda x{:}T.t \mid t_1\, t_2 \mid n \\ n & ::= & 1 \mid 2 \mid \ldots \\ T & ::= & T_1 \xrightarrow{c} T_2 \mid \mathsf{int} \end{array}$$

Functional types are enriched with an annotation c specifying the coeffect required for the parameter in the body.

Semiring of coeffects

The semiring is $(\{\mathbf{0}, \mathbf{1}, \omega\}, \oplus, \mathbf{0}, \otimes, \mathbf{1})$. Coeffect $\mathbf{0}$ is assigned to unused variables, $\mathbf{1}$ to variables used linearly (exactly once), ω to unrestricted variables.

$\oplus$	$\mathbf{0}$	$\mathbf{1}$	ω
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	ω
$\mathbf{1}$	$\mathbf{1}$	ω	ω
ω	ω	ω	ω

$\otimes$	$\mathbf{0}$	$\mathbf{1}$	ω
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	ω
ω	$\mathbf{0}$	ω	ω

Typing rules

(T-VAR) $$\frac{}{\mathbf{0} \otimes \Gamma, x :_{\mathbf{1}} T \vdash x : T}$$

(T-ABS) $$\frac{\Gamma, x :_c T_1 \vdash t : T_2}{\Gamma \vdash \lambda x{:}T_1.t : T_1 \xrightarrow{c} T_2}$$

(T-APP) $$\frac{\Gamma_1 \vdash t_1 : T_2 \xrightarrow{c} T_1 \qquad \Gamma_2 \vdash t_2 : T_2}{\Gamma_1 \oplus (c \otimes \Gamma_2) \vdash t_1\, t_2 : T_1}$$

For the term $\lambda x{:}\mathsf{int}.\lambda f{:}\mathsf{int} \xrightarrow{\mathbf{1}} T.f\,3$ the following judgments holds:

$$\emptyset \vdash \lambda x{:}\mathsf{int}.\lambda f{:}\mathsf{int} \xrightarrow{\mathbf{1}} T.f\,3 : \mathsf{int} \xrightarrow{\mathbf{0}} (\mathsf{int} \xrightarrow{\mathbf{1}} T) \xrightarrow{\mathbf{1}} T$$

$$x :_{\mathbf{0}} \mathsf{int} \vdash \lambda f{:}\mathsf{int} \xrightarrow{\mathbf{1}} T.f\,3 : (\mathsf{int} \xrightarrow{\mathbf{1}} T) \xrightarrow{\mathbf{1}} T$$

$$x :_{\mathbf{0}} \mathsf{int}, f :_{\mathbf{1}} \mathsf{int} \xrightarrow{\mathbf{1}} T \vdash f\,3 : T$$

Sharing

In the imperative programming paradigm, *sharing* is the situation when a portion of the store can be accessed through more than one reference, say x and y, so that a change to x affects y as well.

Interesting properties of a reference

- *Capsule*: the subgraph reachable from x cannot be reached through other references.
- *Lent*: the subgraph reachable from x can be manipulated, but not shared, by a client.
- *Read-only*: the object graph of x cannot be modified through x.
- *Immutable*: the object graph of x will not be modified through any reference.

A coeffect system for sharing

We assume a countable set Lnk of *links*, ranged over by ℓ, with a distinguished element res, and an operation $\triangleleft$ defined by

$$\ell \triangleleft \ell' = \begin{cases} \ell & \text{if } \ell' = \mathsf{res} \\ \ell' & \text{otherwise} \end{cases}$$

Structure

The *sharing coeffects semiring* is the tuple $(\mathcal{P}_f(\mathsf{Lnk}), \oplus, \mathbf{0}, \triangleleft, \mathbf{1})$, where $\mathcal{P}_f(\mathsf{Lnk})$ is the finite powerset of Lnk, $\triangleleft$ is the lifting of $\triangleleft$ to sets, that is, $X \triangleleft Y = \{\ell_1 \triangleleft \ell_2 \mid \ell_1 \in X, \ell_2 \in Y\}$, and $\oplus$ is the set union. The fact that the link res (a link with the result) is in X models possible sharing between x and the final result of the expression.

An example

```
class B {int f;}
class C {B f1; B f2;}
```

in `x.f1=y; new C(z1,z2)` the evaluation of the expression introduces sharing between `x` and `y`, and between `z1`, `z2`, and the final result.
The following typing judgment is derivable:

$$\mathtt{x} :_{\{\ell\}} \mathtt{C}, \mathtt{y} :_{\{\ell\}} \mathtt{B}, \mathtt{z1} :_{\{\mathsf{res}\}} \mathtt{B}, \mathtt{z2} :_{\{\mathsf{res}\}} \mathtt{B} \vdash \mathtt{x.f{=}y;\ new\ C(z1,z2)} : \mathtt{C}$$

Expected result

Execution preserves sharing

Future goals

- Providing full proofs
- Checking the expressivity on significant examples from the literature
- Designing a convenient programmer's interface
- Analyzing the impact on other OO features

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