Coeffects: type systems for resource analysis

Riccardo Bianchini[⋆]

joint work with Francesco Dagnino[⋆], Paola Giannini[†] and Elena Zucca[⋆]

⋆ DIBRIS, Università di Genova
† DiSSTE, Università del Piemonte Orientale

T-Ladies kick-off
6-7 July 2022

Outline

Motivation/aim from the project's description

T3.3: Substructural types for entities

Applications based on complex dynamic systems as IoT applications have to verify **resource sensitive** properties since often entities have limited interaction and synchronization capabilities, computational power and storage space.

Motivation/aim from the project's description

T3.3: Substructural types for entities

Applications based on complex dynamic systems as IoT applications have to verify **resource sensitive** properties since often entities have limited interaction and synchronization capabilities, computational power and storage space.

Type systems should support the analysis of several kinds of **resource sensitive** as information flow, program sensitivity, ...

Motivation/aim from the project's description

T3.3: Substructural types for entities

Applications based on complex dynamic systems as IoT applications have to verify **resource sensitive** properties since often entities have limited interaction and synchronization capabilities, computational power and storage space.

Type systems should support the analysis of several kinds of **resource sensitive** as information flow, program sensitivity, ...

We will investigate the use of various forms of **substructural types** to express such properties, i.e., type abstraction mechanisms able to control the number of uses of a data or structure operation.

Motivation/aim in summary

- modern applications are **resource-aware**

Motivation/aim in summary

- modern applications are **resource-aware**
- important to **keep track of the use** of resources

Motivation/aim in summary

- modern applications are **resource-aware**
- important to **keep track of the use** of resources
- in programs, resources are modeled as **variables**

Motivation/aim in summary

- modern applications are **resource-aware**
- important to **keep track of the use** of resources
- in programs, resources are modeled as **variables**
- **substructural** type systems keep track of the use of variables

Standard type systems

Typing judgment

$$x_1 : T_1, \ldots, x_n : T_n \vdash e : T$$

- e expression
- T type
- $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ type context

Weakening and contraction are allowed

$$(\text{WEAK})\ \frac{\Gamma \vdash e : T_2}{\Gamma, x : T_1 \vdash e : T_2} \qquad (\text{CONTR})\ \frac{\Gamma, x : T_1, x : T_1 \vdash e : T_2}{\Gamma, x : T_1 \vdash e : T_2}$$

Coeffect systems

- a flexible form of substructural type systems
- recently introduced [PetricekOM@ICFP14,BrunelGMZ@ESOP14]
- contexts are enriched with **coeffects** tracking the use of variables

Coeffect systems

Coeffect judgment

$$x_1 :_{c_1} T_1, \ldots, x_n :_{c_n} T_n \vdash e : T$$

- e expression
- T type
- $\Gamma = x_1 :_{c_1} T_1, \ldots, x_n :_{c_n} T_n$ type **and coeffect** context

Coeffect systems

Coeffect judgment

$$x_1 :_{c_1} T_1, \ldots, x_n :_{c_n} T_n \vdash e : T$$

- e expression
- T type
- $\Gamma = x_1 :_{c_1} T_1, \ldots, x_n :_{c_n} T_n$ type **and coeffect** context
- c_i models how variable x_i is used in e

Simply-typed lambda-calculus with pairs and integers

$$
\begin{array}{lcl}
t & ::= & n \mid \langle t_1, t_2 \rangle \mid x \mid \lambda x{:}T.t \mid t_1\, t_2 \\
n & ::= & 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \ldots \\
\\
T & ::= & \mathit{int} \mid T_1 \times T_2 \mid T_1 \to T_2
\end{array}
$$

Rules of the standard type system

$$(\text{T-CONST})\ \frac{}{\Gamma \vdash n : \mathsf{int}} \qquad (\text{T-PAIR})\ \frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \langle t_1, t_2 \rangle : T_1 \times T_2}$$

$$(\text{T-VAR})\ \frac{}{\Gamma, x : T \vdash x : T} \qquad (\text{T-ABS})\ \frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x{:}T_1.t : T_1 \to T_2}$$

$$(\text{T-APP})\ \frac{\Gamma \vdash t_1 : T_2 \to T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\, t_2 : T_1}$$

Types do not track the use of variables

- $\mathit{discard} = \lambda x{:}\mathrm{int}.\langle 1, 1\rangle$

Types do not track the use of variables

- *discard* $= \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- *linear* $= \lambda x{:}\mathsf{int}.\langle x, 1\rangle$

Types do not track the use of variables

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$

Types do not track the use of variables

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$

Types do not track the use of variables

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$

Types do not track the use of variables

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{duplicate} : \mathsf{int} \to \mathsf{int} \times \mathsf{int}$

Adding coeffects

- **0** assigned to unused variables

Adding coeffects

- **0** assigned to unused variables
- **1** assigned to variables used linearly (exactly once)

Adding coeffects

- **0** assigned to unused variables
- **1** assigned to variables used linearly (exactly once)
- ω assigned to variables used more than once

Lambda calculus with coeffects: enriching function types

$$
\begin{array}{lcl}
t & ::= & n \mid \langle t_1, t_2 \rangle \mid x \mid \lambda x{:}T.t \mid t_1\ t_2 \\
n & ::= & 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \ldots \\
 & & \\
T & ::= & \mathsf{int} \mid T_1 \times T_2 \mid T_1 \xrightarrow{c} T_2
\end{array}
$$

Lambda calculus with coeffects: typing rules

$$(\text{T-PAIR})\ \frac{\Gamma_1 \vdash t_1 : T_1 \qquad \Gamma_2 \vdash t_2 : T_2}{\Gamma_1 \oplus \Gamma_2 \vdash \langle t_1, t_2 \rangle : T_1 \times T_2}$$

$\Gamma_1 \oplus \Gamma_2$ context obtained by pointwise sum of coeffects

$\oplus$	**0**	**1**	ω
0	**0**	**1**	ω
1	**1**	ω	ω
ω	ω	ω	ω

Lambda calculus with coeffects: typing rules

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

$c \otimes \Gamma$ context obtained by pointwise product of coeffects

in $\mathbf{0} \otimes \Gamma$ all variables have coeffect $\mathbf{0}$

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

Lambda calculus with coeffects: typing rules

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

annotation c = coeffect of x in the body t

Lambda calculus with coeffects: typing rules

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

- sum of the coeffects of t_1 and
- the coeffects of the argument t_2 multiplied by the coeffects of the function

Typing judgments with coeffects

- *discard* $= \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- *linear* $= \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- *duplicate* $= \lambda x{:}\mathsf{int}.\langle x, x\rangle$

Typing judgments with coeffects

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$

Typing judgments with coeffects

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \xrightarrow{\mathbf{1}} \mathsf{int} \times \mathsf{int}$

Typing judgments with coeffects

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$

- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \xrightarrow{\mathbf{1}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{duplicate} : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}$

Type derivation (1): function

$$\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}$$

Type derivation (1): function

$$\text{(T-ABS)}\ \frac{x :_{\omega} \mathsf{int} \vdash \langle x, x \rangle : \mathsf{int} \times \mathsf{int}}{\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x \rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}}$$

Type derivation (1): function

$$\text{(T-ABS)}\ \dfrac{\text{(T-PAIR)}\ \dfrac{x :_{1} \mathsf{int} \vdash x : \mathsf{int} \qquad x :_{1} \mathsf{int} \vdash x : \mathsf{int}}{x :_{\omega} \mathsf{int} \vdash \langle x, x \rangle : \mathsf{int} \times \mathsf{int}}}{\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x \rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}}$$

Type derivation (1): function

$$\dfrac{(\text{T-PAIR})\ \dfrac{(\text{T-VAR})\ \dfrac{}{x :_{\mathbf{1}} \mathsf{int} \vdash x : \mathsf{int}} \qquad (\text{T-VAR})\ \dfrac{}{x :_{\mathbf{1}} \mathsf{int} \vdash x : \mathsf{int}}}{x :_{\omega} \mathsf{int} \vdash \langle x, x \rangle : \mathsf{int} \times \mathsf{int}}}{\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x \rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}}\ (\text{T-ABS})$$

in the consequence of rule (T-PAIR) x has coeffect ω since $\mathbf{1} \oplus \mathbf{1} = \omega$

Type derivation (2): application

$$y :_{\omega} \mathsf{int} \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle\ y : \mathsf{int} \times \mathsf{int}$$

Type derivation (2): application

$$(\text{T-APP})\ \frac{\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int} \qquad y :_{\mathbf{1}} \mathsf{int} \vdash y : \mathsf{int}}{y :_{\omega} \mathsf{int} \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle\ y : \mathsf{int} \times \mathsf{int}}$$

Type derivation (2): application

$$
(\text{T-APP})\ \dfrac{\dfrac{\cdots}{\emptyset \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle : \mathsf{int} \xrightarrow{\omega} \mathsf{int} \times \mathsf{int}} \qquad (\text{T-VAR})\ \dfrac{}{y :_{\mathbf{1}} \mathsf{int} \vdash y : \mathsf{int}}}{y :_{\omega} \mathsf{int} \vdash \lambda x{:}\mathsf{int}.\langle x, x\rangle\ y : \mathsf{int} \times \mathsf{int}}
$$

in the consequence of rule (T-APP) y has coeffect ω since $\omega \otimes \mathbf{1} = \omega$

Other examples: counting occurrences

- tracking the exact number of occurrences of a variable in a term

Other examples: counting occurrences

- tracking the exact number of occurrences of a variable in a term
- coeffects = natural numbers

Other examples: counting occurrences

- tracking the exact number of occurrences of a variable in a term
- coeffects = natural numbers
- sum and product = sum and product in $\mathbb{N}$

Other examples: counting occurrences

- tracking the exact number of occurrences of a variable in a term
- coeffects = natural numbers
- sum and product = sum and product in $\mathbb{N}$
- rules remain the same, only coeffects and their operations change

Typing judgments counting occurrences

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$

Typing judgments counting occurrences

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$

Typing judgments counting occurrences

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$
- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \xrightarrow{\mathbf{1}} \mathsf{int} \times \mathsf{int}$

Typing judgments counting occurrences

- $\mathit{discard} = \lambda x{:}\mathsf{int}.\langle 1, 1\rangle$
- $\mathit{linear} = \lambda x{:}\mathsf{int}.\langle x, 1\rangle$
- $\mathit{duplicate} = \lambda x{:}\mathsf{int}.\langle x, x\rangle$

- $\emptyset \vdash \mathit{discard} : \mathsf{int} \xrightarrow{\mathbf{0}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{linear} : \mathsf{int} \xrightarrow{\mathbf{1}} \mathsf{int} \times \mathsf{int}$
- $\emptyset \vdash \mathit{duplicate} : \mathsf{int} \xrightarrow{2} \mathsf{int} \times \mathsf{int}$

Other examples

- **confidentiality**: check that a variable declared as private will not become public during the execution [GaboardiKOBU@ICFP16]

Other examples

- **confidentiality**: check that a variable declared as private will not become public during the execution [GaboardiKOBU@ICFP16]
- **noise sensitivity**: outputs of function with type $A \xrightarrow{r} B$ evaluated on input x and input $x + d$ differ at most $r * d$ [PierceR@ICFP10,AbelB@ICFP20]

Other examples

- **confidentiality**: check that a variable declared as private will not become public during the execution [GaboardiKOBU@ICFP16]
- **noise sensitivity**: outputs of function with type $A \xrightarrow{r} B$ evaluated on input x and input $x + d$ differ at most $r * d$ [PierceR@ICFP10,AbelB@ICFP20]
- **Granule**: fully-fledged Haskell-like language in which various kinds of coeffects (counting occurences, confidentiality, ...) are supported [OrchardLE@ICFP19]

Structure

- examples show the same pattern

Structure

- examples show the same pattern
- we can keep the rules and change only the coeffects

Structure

- examples show the same pattern
- we can keep the rules and change only the coeffects
- general structure of coeffects = **semiring** = $(\mathcal{C}, \oplus, \mathbf{0}, \otimes, \mathbf{1})$ 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 $\mathcal{C}$
    - $\mathbf{0} \otimes c = c \otimes \mathbf{0} = \mathbf{0}$

Current work: coeffects for Java-like languages

Investigate the use of coeffects to express different properties of interest in Java-like languages

1. **Sharing coeffects** for an imperative Java-like calculus
   [BianchiniDGZ@submitted]
2. Java-like calculus with **user-defined** coeffects
   [BianchiniDGZ@ICTCS22]

Sharing coeffects

- coeffects modeling **sharing** possibly introduced by an imperative program
- key issue for correctness in presence of mutable state, even more with concurrency

Sharing coeffects

- coeffects modeling **sharing** possibly introduced by an imperative program
- key issue for correctness in presence of mutable state, even more with concurrency
- huge literature on sharing and mutation control, **never modeled by coeffects**
- example of property of interest:
  the result of an expression will be the **unique entry point** for a portion of store
  hence, can be safely handled by a thread

Sharing coeffects

- assume a countable set of **links** ℓ with a distinguished element **res**

Sharing coeffects

- assume a countable set of **links** ℓ with a distinguished element **res**
- coeffects X, Y, $\ldots =$ sets of links
- in a judgment $\Gamma, x :_X T_1, y :_Y T_2 \vdash e : T_3$

Sharing coeffects

- assume a countable set of **links** ℓ with a distinguished element **res**
- coeffects $X, Y, \ldots =$ sets of links
- in a judgment $\Gamma, x :_X T_1, y :_Y T_2 \vdash e : T_3$
- $X \cap Y \neq \emptyset$ means: sharing could be introduced between x and y

Sharing coeffects

- assume a countable set of **links** ℓ with a distinguished element **res**
- coeffects X, Y, $\ldots =$ sets of links
- in a judgment $\Gamma, x :_X T_1, y :_Y T_2 \vdash e : T_3$
- $X \cap Y \neq \emptyset$ means: sharing could be introduced between x and y
- **res** $\in X$ means: sharing could be introduced between x and the final result

An example

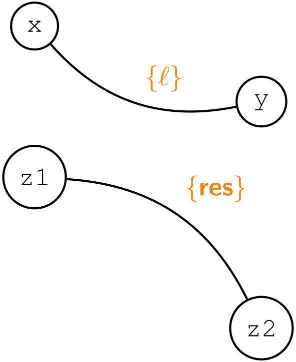
```
class B {int f;}
class C {B f1; B f2;}

x.f1=y; new C(z1,z2)
```

$\texttt{x} :_{\{\ell\}} \texttt{C}, \texttt{y} :_{\{\ell\}} \texttt{B}, \texttt{z1} :_{\{\textbf{res}\}} \texttt{B}, \texttt{z2} :_{\{\textbf{res}\}} \texttt{B} \vdash \texttt{x.f1=y;new C(z1,z2)} : \texttt{C}$

An example

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



User-defined coeffects

- Java-like calculus where declared variables can be annotated by coeffects
- Coeffect annotations are written in the language itself
- They are expressions of (subclasses of) a predefined class `Coeffect`
- Analogous to Java exceptions which are expressions of (subclasses of) `Exception`

The `Coeffect` class

general structure of coeffects = **semiring** = $(\mathcal{C}, \oplus, \mathbf{0}, \otimes, \mathbf{1})$

```
class Coeffect {
  Coeffect sum(Coeffect c) { new Coeffect() }
  Coeffect mult(Coeffect c) { new Coeffect() }
  Coeffect zero() { new Coeffect() }
  Coeffect one() { new Coeffect() }
}
```

Example of user-defined coeffects: 0, 1, ω

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

0 coeffects

```
class Zero extends Linearity{
   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()
  }
}
```

1 coeffects

```
class One extends Linearity{
  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()
  }
}
```

ω coeffects

```
class Omega extends Linearity{
  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()
  }
}
```

Example

```
class Pair {
  A fst; A snd;
}

class A {
  Pair discard [new Zero()] () {
    return new Pair{new A(),new A()}
  }

  Pair linear [new One()] () {
    return new Pair{this, new A()}
  }

  Pair duplicate [new Omega()] () {
    return new Pair(this,this)
  }
}
```

Future work

- From coeffects to **graded modal types**
  - with coeffect annotations it is possible to specify how **a variable** should be used, but not to do the same for **the result of an expression**
  - **graded modal types**, which are, roughly, types annotated with coeffects (grades), would allow to overcome this limitation

Future work

- From coeffects to **graded modal types**
  - with coeffect annotations it is possible to specify how **a variable** should be used, but not to do the same for **the result of an expression**
  - **graded modal types**, which are, roughly, types annotated with coeffects (grades), would allow to overcome this limitation
- **Integration** of different coeffect systems
  - different coeffect systems coexist in Granule and our Java-like calculus
  - we plan to provide a general foundation

Hints for collaborations

- Other tasks/applications where coeffects could be fruitfully employed
- Implementation:
  - rules in coeffect systems directly lead to an algorithm (coeffects are **computed** bottom up)
  - user-defined coeffects in Java could be implemented as an **extension to be translated in plain Java**

Thank You

Bibliography

Tomas Petricek, Dominic Orchard, and Alan Mycroft, *Coeffects: A calculus of context-dependent computation*, ICFP 2014

Aloïs Brunel, Marco Gaboardi, Damiano Mazza and Steve Zdancewic, *A Core Quantitative Coeffect Calculus*, ICFP 2014

Dominic Orchard, Vilem-Benjamin Liepelt and Harley Eades III, *Quantitative Program Reasoning with Graded Modal Types*, ICFP 2019

Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart and Tarmo Uustalu, *Quantitative Program Reasoning with Graded Modal Types*, ICFP 2016

Pritam Choudhury, Harley Eades III, and Richard A. Eisenberg and Stephanie Weirich, *A graded dependent type system with a usage-aware semantics*, POPL 2021

Jason Reed and Benjamin Pierce, *Distance Makes the Types Grow Stronger*, ICFP 2010

Andreas Abel and Jean-Philippe Bernardy, *A Unified View of Modalities in Type Systems*, ICFP 2020

Dominic Orchard, Vilem-Benjamin Liepelt, Harley Eades III, *Quantitative Program Reasoning with Graded Modal Types*, ICFP 2019