# $\lambda$-calculus and algebraic operations

Renato Neves

Universidade do Minho

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

**Architecture and Calculi Course Unit**

# Table of Contents

# Recalling $\lambda$-calculus

Types: $\mathbb{A} \ni 1 \mid \mathbb{A} \times \mathbb{A} \mid \mathbb{A} \to \mathbb{A}$

Programs are built according to the rules

$$\frac{x : \mathbb{A} \in \Gamma}{\Gamma \vdash x : \mathbb{A}} \qquad \frac{}{\Gamma \vdash * : 1} \qquad \frac{\Gamma \vdash V : \mathbb{A} \times \mathbb{B}}{\Gamma \vdash \pi_1 V : \mathbb{A}}$$

$$\frac{\Gamma \vdash V : \mathbb{A} \qquad \Gamma \vdash U : \mathbb{B}}{\Gamma \vdash \langle V, U \rangle : \mathbb{A} \times \mathbb{B}} \qquad \frac{\Gamma, x : \mathbb{A} \vdash V : \mathbb{B}}{\Gamma \vdash \lambda x : \mathbb{A}.\, V : \mathbb{A} \to \mathbb{B}}$$

$$\frac{\Gamma \vdash V : \mathbb{A} \to \mathbb{B} \quad \Gamma \vdash U : \mathbb{A}}{\Gamma \vdash V\, U : \mathbb{B}}$$

$\Gamma$ is a non-repetitive list of typed variables $x_1 : \mathbb{A}_1 \ldots x_n : \mathbb{A}_n$

# Sequential composition

Consider the following "new" deductive rule,

$$\frac{\Gamma \vdash V : \mathbb{A} \qquad x : \mathbb{A} \vdash U : \mathbb{B}}{\Gamma \vdash x \leftarrow V; U : \mathbb{B}}$$

It reads as "bind the computation $V$ to $x$ and then run $U$"

Its interpretation is defined as

$$\frac{[\![\Gamma \vdash V : \mathbb{A}]\!] = f \qquad [\![x : \mathbb{A} \vdash U : \mathbb{B}]\!] = g}{[\![\Gamma \vdash x \leftarrow V; U : \mathbb{B}]\!] = g \cdot f}$$

Can you show that this operator is definable from the previous rules of $\lambda$-calculus?

# Table of Contents

# Signatures

A signature $\Sigma = \{(\sigma_1, n_1), (\sigma_2, n_2), \dots\}$ is a set of operations $\sigma_i$ paired with the number of inputs $n_i$ they are supposed to receive

Signatures will later be integrated in $\lambda$-calculus

They constitute aforementioned the algebraic operations (a.k.a effects)

## Examples

- Exceptions: $\Sigma = \{(\mathrm{e}, 0)\}$
- Read a bit from the environment: $\Sigma = \{(\mathrm{read}, 2)\}$
- Wait calls: $\Sigma = \{(\mathrm{wait}_n, 1) \mid n \in \mathbb{N}\}$
- Non-deterministic choice: $\Sigma = \{(+, 2)\}$

# Table of Contents

# Simply-typed $\lambda$-calculus with algebraic operations

Types: $\mathbb{A} \ni 1 \mid \mathbb{A} \times \mathbb{A} \mid \mathbb{A} \to \mathbb{A}$

We choose a signature $\Sigma$ of algebraic operations and introduce a new deductive rule

$$\frac{(\sigma, n) \in \Sigma \qquad \forall i \leq n.\, \Gamma \vdash M_i : \mathbb{A}}{\Gamma \vdash \sigma(M_1, \ldots, M_n) : \mathbb{A}}$$

# Examples of **effectful** $\lambda$-terms

$x : \mathbb{A} \vdash \mathrm{wait}_1(x) : \mathbb{A}$ (adds a delay of one second to returning $x$)

$\Gamma \vdash \mathrm{e} : \mathbb{A}$ (raises an exception $e$)

$x : \mathbb{A} \times \mathbb{A} \vdash \mathrm{read}(\pi_1\, x, \pi_2\, x) : \mathbb{A}$ (receives a bit: if the bit is 0 it returns $\pi_1\, x$ otherwise it returns $\pi_2\, x$)

# Examples of **effectful** $\lambda$-terms

$x : \mathbb{A} \vdash \mathrm{wait}_1(x) : \mathbb{A}$ (adds a delay of one second to returning $x$)

$\Gamma \vdash \mathrm{e} : \mathbb{A}$ (raises an exception $e$)

$x : \mathbb{A} \times \mathbb{A} \vdash \mathrm{read}(\pi_1\, x, \pi_2\, x) : \mathbb{A}$ (receives a bit: if the bit is 0 it returns $\pi_1\, x$ otherwise it returns $\pi_2\, x$)

## Exercise

Define a $\lambda$-term $x : \mathbb{A} \vdash ? : \mathbb{A}$ that requests a bit from the user and depending on the value read it returns $x$ with either one or two seconds of delay.

# Table of Contents

# Semantics of $\lambda$-calculus with algebraic operations

How to provide a suitable semantics to this family of programming languages?

The short answer: via monads

The long answer: see the next slides ...

# The core idea

Recall that programs $\Gamma \vdash V : \mathbb{A}$ are interpreted as functions

$$[\![\Gamma \vdash V : \mathbb{A}]\!] : [\![\Gamma]\!] \longrightarrow [\![\mathbb{A}]\!]$$

Recall as well that there exists only one function of type

$$[\![\Gamma]\!] \longrightarrow [\![1]\!]$$

Problem: it is then necessarily the case that

$$[\![\Gamma \vdash x : 1]\!] = [\![\Gamma \vdash \mathrm{wait}_1(x) : 1]\!]$$

despite these programs having different execution times

# The core idea II

Previously, we interpreted a program $\Gamma \vdash V : \mathbb{A}$ as a function

$$[\![\Gamma \vdash V : \mathbb{A}]\!] : [\![\Gamma]\!] \longrightarrow [\![\mathbb{A}]\!]$$

which returns values in $[\![\mathbb{A}]\!]$. But now values come with effects ...

So instead of having $[\![\mathbb{A}]\!]$ as the set of outputs, we have a set of effects $T[\![\mathbb{A}]\!]$ over $[\![\mathbb{A}]\!]$ as outputs

$$[\![\Gamma \vdash M : \mathbb{A}]\!] : [\![\Gamma]\!] \longrightarrow T[\![\mathbb{A}]\!]$$

$T$ is a set-constructor: *i.e.* given a set of outputs $X$ it returns a set of effects $TX$ over $X$

# The core idea III

For wait calls, the corresponding set-constructor $T$ is defined as

$$X \mapsto \mathbb{N} \times X$$

*i.e.* values in $X$ paired with an execution time

For exceptions, the corresponding set-constructor $T$ is defined as

$$X \mapsto X + \{e\}$$

*i.e.* values in $X$ plus an element $e$ representing the exception

# The problem

This idea of a set-constructor $T$ seems good, but it breaks sequential composition

$$\llbracket \Gamma \vdash M : \mathbb{A} \rrbracket \ : \llbracket \Gamma \rrbracket \to T\llbracket \mathbb{A} \rrbracket$$
$$\llbracket x : \mathbb{A} \vdash N : \mathbb{B} \rrbracket \ : \llbracket \mathbb{A} \rrbracket \to T\llbracket \mathbb{B} \rrbracket$$

We need a way to convert a function $h : X \to TY$ into a function of the type

$$h^{\star} : TX \to TY$$

# The problem II

There are set-constructors $T$ for which this is possible

In the case of wait-calls

$$\frac{f : X \to TY = \mathbb{N} \times Y}{f^\star(n, x) = (n + m, y) \text{ where } f(x) = (m, y)}$$

In the case of exceptions

$$\frac{f : X \to TY = Y + \{e\}}{f^\star(x) = f(y) \qquad f^\star(e) = e}$$

# Testing the idea with a simple example

$$\llbracket x : 1 \vdash y \leftarrow \mathrm{wait}_1(x); \mathrm{wait}_2(y) : 1 \rrbracket$$

$$= \llbracket y : 1 \vdash \mathrm{wait}_2(y) : 1 \rrbracket^* \cdot \llbracket x : 1 \vdash \mathrm{wait}_1(x) : 1 \rrbracket$$

$$= (v \mapsto (2, v))^* \cdot (v \mapsto (1, v))$$

$$= v \mapsto (3, v)$$

# Another problem

The idea of interpreting $\lambda$-terms $\Gamma \vdash M : \mathbb{A}$ as functions

$$[\![\Gamma \vdash M : \mathbb{A}]\!] : [\![\Gamma]\!] \longrightarrow T[\![\mathbb{A}]\!]$$

looks good but it presupposes that all terms invoke effects

There are terms that do not do this, *e.g.*

$$[\![x : \mathbb{A} \vdash x : \mathbb{A}]\!] : [\![\mathbb{A}]\!] \longrightarrow [\![\mathbb{A}]\!]$$

## Solution

$T[\![\mathbb{A}]\!]$ should also include values free of effects, specifically there should exist a function

$$\eta_{[\![\mathbb{A}]\!]} : [\![\mathbb{A}]\!] \longrightarrow T[\![\mathbb{A}]\!]$$

that maps a value to the corresponding effect-free representation in $T[\![\mathbb{A}]\!]$

# Another problem II

Again there are set-constructors $T$ for which this is possible:

In the case of wait-calls

$$\frac{TX = \mathbb{N} \times X}{\eta_X(x) = (0, x)}$$

(*i.e.* no wait call was invoked)

In the case of exceptions

$$\frac{TX = X + \{e\}}{\eta_X(x) = x}$$

(*i.e.* the exception $e$ was never raised)

# Monads unlocked

The analysis we did in the previous slides naturally leads to the notion of a monad

## Definition

A monad $(T, \eta, (-)^\star)$ is as triple such that $T$ is a set-constructor, $\eta$ is a function $\eta_X : X \to TX$ for each set $X$, and $(-)^\star$ is an operation

$$\frac{f : X \to TY}{f^\star : TX \to TY}$$

such that the following laws are respected: $\eta^\star = \mathrm{id}$, $f^\star \cdot \eta = f$, $(f^\star \cdot g)^\star = f^\star \cdot g^\star$

The laws above are required to forbid “weird” computational behaviour

# Exercise

Show that the set-constructor

$$X \mapsto \mathbb{N} \times X$$

can be equipped with a monadic structure

Show that the set-constructor

$$X \mapsto X + 1$$

can be equipped with a monadic structure

# Table of Contents

# To keep in mind

Let us use what we learned thus far to extend $\lambda$-calculus with algebraic operations and provide it with a proper semantics

To this effect, recall that,

- we fix a signature $\Sigma$ of algebraic operations
- we have monads $(T, \eta, (-)^\star)$ at our disposal
- Programs $\Gamma \vdash V : \mathbb{A}$ can be seen either as functions of type $[\![\Gamma]\!] \to [\![\mathbb{A}]\!]$ or of type $[\![\Gamma]\!] \to T[\![\mathbb{A}]\!]$

# Semantics for effectful simply-typed $\lambda$-calculus

Types $\mathbb{A}$ are interpreted as sets $[\![\mathbb{A}]\!]$

$$[\![1]\!] = \{\star\} \qquad [\![\mathbb{A} \times \mathbb{B}]\!] = [\![\mathbb{A}]\!] \times [\![\mathbb{B}]\!] \qquad [\![\mathbb{A} \to \mathbb{B}]\!] = (T[\![\mathbb{B}]\!])^{[\![\mathbb{A}]\!]}$$

A typing context $\Gamma$ is interpreted as

$$[\![\Gamma]\!] = [\![x_1 : \mathbb{A}_1 \times \cdots \times x_n : \mathbb{A}_n]\!] = [\![\mathbb{A}_1]\!] \times \cdots \times [\![\mathbb{A}_n]\!]$$

For each operation $(\sigma, n) \in \Sigma$ and set $X$ we postulate the existence of a map

$$[\![\sigma]\!]_X \colon (TX)^n \longrightarrow TX$$

# Semantics for effectful simply-typed $\lambda$-calculus II

$$\frac{x_i : \mathbb{A} \in \Gamma}{[\![\Gamma \vdash x_i]\!] = \pi_i} \qquad \frac{}{[\![\Gamma \vdash *]\!] = !} \qquad \frac{[\![\Gamma \vdash V : \mathbb{A}]\!] = f \qquad [\![\Gamma \vdash U : \mathbb{B}]\!] = g}{[\![\Gamma \vdash \langle V, U\rangle : \mathbb{A} \times \mathbb{B}]\!] = \langle f, g\rangle}$$

$$\frac{[\![\Gamma, x : \mathbb{A} \vdash_{\mathsf{c}} M : \mathbb{B}]\!] = f}{[\![\Gamma \vdash \lambda x : \mathbb{A}.\, M : \mathbb{A} \to \mathbb{B}]\!] = \lambda f} \qquad \frac{[\![\Gamma \vdash V : \mathbb{A} \times \mathbb{B}]\!] = f}{[\![\Gamma \vdash \pi_1 V : \mathbb{A}]\!] = \pi_1 \cdot f}$$

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

$$\frac{[\![\Gamma \vdash V : \mathbb{A}]\!] = f}{[\![\Gamma \vdash_{\mathsf{c}} \mathtt{return}\ V : \mathbb{A}]\!] = \eta \cdot f} \qquad \frac{[\![\Gamma \vdash_{\mathsf{c}} M : \mathbb{A}]\!] = f \qquad [\![x : \mathbb{A} \vdash_{\mathsf{c}} N : \mathbb{B}]\!] = g}{[\![\Gamma \vdash_{\mathsf{c}} x \leftarrow M\, ;\, N : \mathbb{B}]\!] = g^\star \cdot f}$$

$$\frac{[\![\Gamma \vdash V : \mathbb{A} \to \mathbb{B}]\!] = f \quad [\![\Gamma \vdash U : \mathbb{A}]\!] = g}{[\![\Gamma \vdash_{\mathsf{c}} V\, U : \mathbb{B}]\!] = \mathrm{app} \cdot \langle f, g\rangle}$$

$$\frac{(\sigma, n) \in \Sigma \qquad \forall i \leq n.\ [\![\Gamma \vdash_{\mathsf{c}} M_i : \mathbb{A}]\!] = f_i}{[\![\Gamma \vdash_{\mathsf{c}} \sigma(M_1, \ldots M_n)]\!] = [\![\sigma]\!]_{[\![\mathbb{A}]\!]} \cdot \langle f_1, \ldots, f_n\rangle}$$

# Exercise

Use the interpretation rules to prove that the equations below hold

$[\![\Gamma \vdash x \leftarrow \texttt{return} \; * \; ; (\texttt{return}\, x) : 1]\!] = [\![\Gamma \vdash \texttt{return}\, * : 1]\!]$

(hint: one of the monad laws)

$[\![\Gamma \vdash x \leftarrow \mathrm{wait}_1(\texttt{return}\, *) \, ; (\texttt{return}\, x) : 1]\!] = [\![\Gamma \vdash x \leftarrow \texttt{return} \; * \; ; \mathrm{wait}_1(\texttt{return}\, x) : 1]\!]$

(hint: two of the monad laws)

$[\![\Gamma \vdash x \leftarrow \mathrm{wait}_1(\texttt{return}\, *) \, ; \mathrm{wait}_1(\texttt{return}\, x) : 1]\!] = [\![\Gamma \vdash x \leftarrow \mathrm{wait}_2(\texttt{return}\, *) \, ; (\texttt{return}\, x) : 1]\!]$

# Exercises

Build a $\lambda$-term that receives a value, waits one second, and returns the same value. Run this in `Haskell` using `DurationMonad.hs`. What is the value obtained when you feed this function with "Hi"? Justify.

Can you build a $\lambda$-term that receives a function $f : \mathbb{A} \to \mathbb{A}$, receives a value $x : \mathbb{A}$, and applies $f$ to $x$ twice? In classical $\lambda$-calculus such would be defined as

$$\lambda f : \mathbb{A} \to \mathbb{A}.\ \lambda x : \mathbb{A}.\ f(f\ x)$$