CerCo project: 2nd year review
Task 5.3: Case studies

Yann Régis-Gianas

University Paris Diderot – PPS – INRIA πr^2

March 2012, the 16th

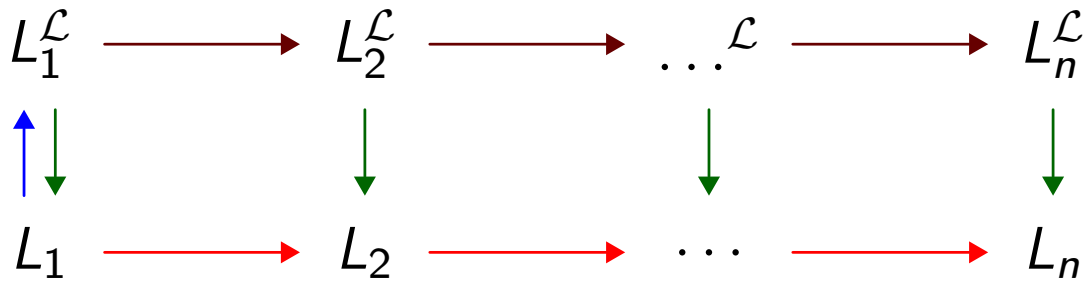
SEVENTH FRAMEWORK PROGRAMME

Context : Functional Programming

"A Lisp programmer
knows the value of everything
but the cost of nothing."

SEVENTH FRAMEWORK PROGRAMME

Does this diagram scale?



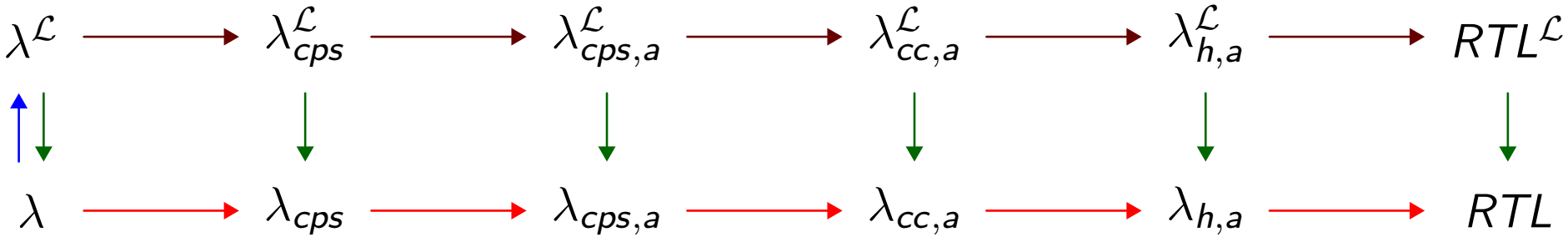
Labelling $\mathcal{L}$ Erasure $\mathcal{E}_i$

Compilation $\mathcal{C}_i$ Labelled compilation $\mathcal{C}_i^{\ell}$

- Last year, we implemented a C compiler following this architecture.
- This year, we applied the labelling method on a **functional language**.

SEVENTH FRAMEWORK PROGRAMME

A compilation chain for the $\lambda-$calculus



Labelling $\mathcal{L}$ Erasure $\mathcal{E}_i$

Compilation $\mathcal{C}_i$ Labelled compilation $\mathcal{C}_i^{\ell}$

λ_{cps}	:	Continuation Passing Style λ-calculus
$\lambda_{cps,a}$	:	... in administrative form
$\lambda_{cc,a}$	:	... with only closed λ-abstractions
$\lambda_{h,a}$	:	... with only toplevel closed λ-abstractions
RTL	:	Register Transfer Level language

SEVENTH FRAMEWORK PROGRAMME

The compilation chain in action

Source code written in λ

fun (f, g) $\rightarrow$ **fun** x $\rightarrow$ f (g (x))

The compilation chain in action

Continuation Passing Style translation in λ_{cps}.

```
halt (fun (f, g, k) →
        k (fun (x, k) →
           g (x, (fun x → f (x, (fun x → k x))))))
```

The compilation chain in action

Administrative form in $\lambda_{cps,a}$.

$$
\begin{array}{l}
\textbf{let } x_{14} = \textbf{fun } (\mathrm{f},\ \mathrm{g},\ k_4) \rightarrow \\
\quad \textbf{let } x_{15} = \textbf{fun } (\mathrm{x},\ k_5) \rightarrow \\
\quad\quad \textbf{let } x_{16} = \textbf{fun } x_{10} \rightarrow \\
\quad\quad\quad \textbf{let } x_{17} = \textbf{fun } x_6 \rightarrow k_5\ (x_6) \textbf{ in } \mathrm{f}\ (x_{10},\ x_{17}) \textbf{ in} \\
\quad\quad \mathrm{g}\ (\mathrm{x},\ x_{16}) \\
\quad \textbf{in} \\
\quad k_4\ (x_{15}) \\
\textbf{in} \\
\mathrm{halt}\ x_{14}
\end{array}
$$

The compilation chain in action

Closure explicitation in $\lambda_{cps,cc}$.

```
let x18 = fun (c19, f, g, k4) →
      let x20 = fun (c21, x, k5) →
      let f = proj 1 (c21) in
      let g = proj 2 (c21) in
      let x23 = fun (c24, x10) →
          let f = proj 1 (c24) in
          let k5 = proj 2 (c24) in
          let x26 = fun _k9 (c27, x6) →
          let k5 = proj 1 (c27) in
          let x29 = proj 1 (k5) in x29 (k5, x6) in
          let x17 = (x26, k5) in
          let x28 = proj 1 (f) in x28 (f, x10, x17) in
          let x16 = (x23, f, k5) in
          let x25 = proj 1 (g) in x25 (g, x, x16) in
          let x15 = (x20, f, g) in
          let x22 = proj 1 (k4) in x22 (k4, x15)
in
let x14 = (x18) in halt x14
```

The compilation chain in action

Hoisting in $\lambda_{h,a}$.

$$
\begin{array}{l}
\textbf{let } x_{26} = \textbf{fun } (c_{27}, x_6) \rightarrow \\
\quad \textbf{let } k_5 = \text{proj } 1\ (c_{27}) \textbf{ in} \\
\quad \textbf{let } x_{29} = \text{proj } 1\ (k_5) \textbf{ in } x_{29}\ (k_5, x_6) \\
\textbf{in} \\
\textbf{let } x_{23} = \textbf{fun } (c_{24}, x_{10}) \rightarrow \\
\quad \textbf{let } \mathrm{f} = \text{proj } 1\ (c_{24}) \textbf{ in} \\
\quad \textbf{let } k_5 = \text{proj } 2\ (c_{24}) \textbf{ in} \\
\quad \textbf{let } x_{17} = (x_{26}, k_5) \textbf{ in} \\
\quad \textbf{let } x_{28} = \text{proj } 1\ (\mathrm{f}) \textbf{ in } x_{28}\ (\mathrm{f}, x_{10}, x_{17}) \\
\textbf{in} \\
\textbf{let } x_{20} = \textbf{fun } (c_{21}, \mathrm{x}, k_5) \rightarrow \\
\quad \textbf{let } \mathrm{f} = \text{proj } 1\ (c_{21}) \textbf{ in} \\
\quad \textbf{let } \mathrm{g} = \text{proj } 2\ (c_{21}) \textbf{ in} \\
\quad \textbf{let } x_{16} = (x_{23}, \mathrm{f}, k_5) \textbf{ in} \\
\quad \textbf{let } x_{25} = \text{proj } 1\ (\mathrm{g}) \textbf{ in } x_{25}\ (\mathrm{g}, \mathrm{x}, x_{16}) \\
\textbf{in} \\
\textbf{let } x_{18} = \textbf{fun } (c_{19}, \mathrm{f}, \mathrm{g}, k_4) \rightarrow \\
\quad \textbf{let } x_{15} = (x_{20}, \mathrm{f}, \mathrm{g}) \textbf{ in} \\
\quad \textbf{let } x_{22} = \text{proj } 1\ (k_4) \textbf{ in } x_{22}\ (k_4, x_{15}) \\
\textbf{in} \\
\textbf{let } x_{14} = (x_{18}) \textbf{ in halt } x_{14}
\end{array}
$$

The compilation chain in action

Register Transfer Level language.

```
routine x26 (c27, x6)
   k5 ← proj 1 c27 ;
   x29 ← proj 1 k5 ;
   call x29 (k5, x6)
routine x23 (c24, x10)
   f ← proj 1 c24 ;
   k5 ← proj 2 c24 ;
   x17 ← mktuple (x26, k5) ;
   x28 ← proj 1 f ;
   call x28 (f, x10, x17)
routine x20 (c21, x, k5)
   f ← proj 1 c21 ;
   g ← proj 2 c21 ;
   x16 ← mktuple (x23, f, k5) ;
   x25 ← proj 1 g ;
   call x25 (g, x, x16)
routine x18 (c19, f, g, k4)
   x15 ← mktuple (x20, f, g) ;
   x22 ← proj 1 k4 ;
   call x22 (k4, x15)
in
   x14 ← mktuple (x18);
   halt x14
```

The labelling approach, again

Which labelling should we choose
to obtain a sound and precise cost annotation?

and

Which instrumentation is suitable
to reason on the cost of a functional language?

Looking for a sound and precise labelling...

Register Transfer Level language.

```
routine x26 (c27, x6)
  ℓ1: k5 ← proj 1 c27 ;
      x29 ← proj 1 k5 ;
      call x29 (k5, x6)
routine x23 (c24, x10)
  ℓ2: f ← proj 1 c24 ;
      k5 ← proj 2 c24 ;
      x17 ← mktuple (x26, k5) ;
      x28 ← proj 1 f ;
      call x28 (f, x10, x17)
routine x20 (c21, x, k5)
  ℓ3: f ← proj 1 c21 ;
      g ← proj 2 c21 ;
      x16 ← mktuple (x23, f, k5) ;
      x25 ← proj 1 g ;
      call x25 (g, x, x16)
routine x18 (c19, f, g, k4)
  ℓ4: x15 ← mktuple (x20, f, g) ;
      x22 ← proj 1 k4 ;
      call x22 (k4, x15)
in
ℓ5: x14 ← mktuple (x18);
      halt x14
```

Looking for a sound and precise labelling...

Hoisting in $\lambda_{h,a}$.

```
let x26 = fun (c27, x6) →
      ℓ1> let k5 = proj 1 (c27) in
      let x29 = proj 1 (k5) in x29 (k5, x6)
in
let x23 = fun (c24, x10) →
      ℓ2> let f = proj 1 (c24) in
      let k5 = proj 2 (c24) in
      let x17 = (x26, k5) in
      let x28 = proj 1 (f) in x28 (f, x10, x17)
in
let x20 = fun (c21, x, k5) →
      ℓ3> let f = proj 1 (c21) in
      let g = proj 2 (c21) in
      let x16 = (x23, f, k5) in
      let x25 = proj 1 (g) in x25 (g, x, x16)
in
let x18 = fun (c19, f, g, k4) →
      ℓ4> let x15 = (x20, f, g) in
      let x22 = proj 1 (k4) in x22 (k4, x15)
in
ℓ5> let x14 = (x18) in halt x14
```

SEVENTH FRAMEWORK PROGRAMME

Looking for a sound and precise labelling...

Closure explicitation in $\lambda_{cps,cc}$.

```
let x18 = fun (c19, f, g, k4) →
      ℓ1> let x20 = fun (c21, x, k5) →
      ℓ2> let f = proj 1 (c21) in
      let g = proj 2 (c21) in
      let x23 = fun (c24, x10) →
          ℓ3> let f = proj 1 (c24) in
          let k5 = proj 2 (c24) in
          let x26 = fun (c27, x6) →
          ℓ4> let k5 = proj 1 (c27) in
          let x29 = proj 1 (k5) in x29 (k5, x6) in
          let x17 = (x26, k5) in
          let x28 = proj 1 (f) in x28 (f, x10, x17) in
          let x16 = (x23, f, k5) in
          let x25 = proj 1 (g) in x25 (g, x, x16) in
          let x15 = (x20, f, g) in
          let x22 = proj 1 (k4) in x22 (k4, x15)
in
ℓ5> let x14 = (x18) in halt x14
```

SEVENTH FRAMEWORK PROGRAMME

Looking for a sound and precise labelling...

Administrative form in $\lambda_{cps,a}$.

$$
\begin{array}{l}
\textbf{let } x_{14} = \textbf{fun } (\mathrm{f},\ \mathrm{g},\ k_4) \rightarrow \\
\quad \ell_1 > \textbf{ let } x_{15} = \textbf{fun } (\mathrm{x},\ k_5) \rightarrow \\
\qquad \ell_2 > \textbf{ let } x_{16} = \textbf{fun } x_{10} \rightarrow \\
\qquad\quad \ell_3 > \textbf{ let } x_{17} = \textbf{fun } x_6 \rightarrow \\
\qquad\quad \ell_4 > k_5\ (x_6) \textbf{ in } \mathrm{f}\ (x_{10},\ x_{17}) \textbf{ in} \\
\qquad \mathrm{g}\ (\mathrm{x},\ x_{16}) \\
\quad \textbf{in} \\
\quad k_4\ (x_{15}) \\
\textbf{in} \\
\ell_5 > \mathrm{halt}\ x_{14}
\end{array}
$$

Looking for a sound and precise labelling...

Continuation Passing Style translation in λ_{cps}.

$\ell_5>$ halt (**fun** (f, g, k) $\rightarrow$
$\ell_1>$ k (**fun** (x, k) $\rightarrow$
$\ell_2>$ g (x, (**fun** x $\rightarrow$
$\ell_3>$ f (x, (**fun** x $\rightarrow$
$\ell_4>$ k x))))))

SEVENTH FRAMEWORK PROGRAMME

Looking for a sound and precise labelling...

Source code written in λ

fun (f, g) $\rightarrow$ **fun** x $\rightarrow$ f (g (x))

Looking for a sound and precise labelling...

Source code written in λ

Where should we put the labels
corresponding to the abstractions
that were introduced by CPS translation?

CPS translation of applications

"$M : K$" means CPS translation of M given continuation K.

$$@(M_0, \ldots, M_n) : K \quad = \quad M_0 : \lambda x_0. \ldots (M_n : \lambda x_n.@(x_0, \ldots, x_n, K))$$

Looking for a sound and precise labelling...

Source code written in λ

$\ell_1>$ (**fun** (f, g) $\rightarrow$
  $\ell_5>$ (**fun** x $\rightarrow$
    $\ell_4>$ (f
      (g (x) $>$ ℓ_2)))
) $>$ ℓ_3)

Pre and Post labelling

Labelled λ-calculus

We define the labelled version of λ by extending it with two new constructions:

- A pre label "$\ell > M$" emits ℓ before reducing M.
- A post label "$M > \ell$" expects M to be a value in order to emit ℓ.

Sound and precise labelling (informally)

It associates a distinct label with every abstraction and with every application which is not 'immediately surrounded' by an abstraction.

SEVENTH FRAMEWORK PROGRAMME

How to instrument a functional program?

$$
\begin{array}{rcl}
[\![x]\!] & = & (\mathbf{1}, x) \\
[\![\lambda x^{+}.M]\!] & = & (\mathbf{1}, \lambda x^{+}.[\![M]\!]) \\
[\![@(M_0, \ldots, M_n)]\!] & = & \text{let } (m_0, x_0) = [\![M_0]\!] \cdots (m_n, x_n) = [\![M_n]\!], \\
 & & (m_{n+1}, x_{n+1}) = @(x_0, \ldots, x_n) \text{ in} \\
 & & (m_{n+1} \cdot m_n \cdots m_0, x_{n+1})
\end{array}
$$

- Given a **cost monoid** $\mathcal{M}$ with an identity $\mathbf{1}$, we associate with each label ℓ an element m_ℓ of the cost monoid.
- In order to increment a global cost variable with these costs, we use a standard **monadic interpretation** in such a way that every expression computes not only its value but also its cost.

$\Rightarrow$ We are back to standard λ-calculus.

SEVENTH FRAMEWORK PROGRAMME

How to reason on the instrumented program?

- Many tools can be used to reason on functional programs.
- We experimented Higher-Order Hoare Logic.

Proving properties on execution costs using HOHL

1. **Annotate** the functional program with logic assertions.
2. **Compute** a set of proof obligations implying the validity of these assertions.
3. **Prove** these proof obligations.

SEVENTH FRAMEWORK PROGRAMME

How to reason on the instrumented program, in practice?

The monadic interpretation of the functional program
is **not**
human-friendly.

- Logic assertions are written **directly on source code** as if the program *was* in monadic form. An implicit variable `cost` is automatically added to the logical environment.
- The monadic transformation is applied just before the VCG.

The cost of a higher-order function

```
01 type list = Nil | Cons (nat, list)
02 type bool = BTrue | BFalse

03 logic {
04   Definition bound (p : nat ⟶ (nat × bool)) (k : nat) : Prop :=
05       ∀ x m: nat, ∀ r: bool, post p x (m, r) ⇒ m ≤ k.
06   Definition k0 := costof(ℓm) + costof(ℓnil).
07   Definition k1 := costof(ℓm) + costof(ℓp) + costof(ℓc) + costof(ℓf) + costof(ℓr).
08 }

09 let rec pexists (p : nat → bool, l: list) { ∀ x, pre p x } : bool {
10   ((result = BTrue) ⇔ (∃ x c: nat, mem x l ∧ post p x (c, BTrue))) ∧
11   (∀ k: nat, bound p k ∧ (result = BFalse) ⇒ cost ≤ k0 + (k + k1) × length (l))
12 } = ℓm> match l with
13     | Nil → ℓnil> BFalse
14     | Cons (x, xs) → ℓc> match p (x) > ℓp with
15                         | BTrue → BTrue
16                         | BFalse → ℓf> (pexists (p, xs) > ℓr)
```

53 proof obligations. **46** automatically proved. **7** manually proved.

SEVENTH FRAMEWORK PROGRAMME

Conclusion

- The labelling approach can be applied to functional programs.
- What about the cost of automatic memory management?