On a type-based complexity analysis of subrecursive programs

Patrick Baillot

LIP, CNRS & ENS de Lyon

joint work with Gilles Barthe (IMDEA Madrid) and Ugo Dal Lago (Bologna)

Languages, Compilation and Semantics LIP seminar
Nov 3, 2016.

Introduction

- we consider the problem of complexity analysis for higher-order functional programs
- implicit computational complexity has designed *restricted* languages with guaranteed complexity bounds, in particular polynomial time
- however here we rather aim at:
  - an analysis covering a *larger* class of programs
  - allowing to extract concrete complexity bounds

Complexity analysis

- a popular approach to complexity analysis, consisting 2 steps:

  program + cost model $\longrightarrow$ cost equations $\longrightarrow$ closed-form bounds

- examples:

  imperative : COSTA [Albert et al.07]

  functional : [LeMetayer88,Benzinger04,Danner et al.15...]

Higher-order functional programs

- modular code, but difficult to analyse
- actually here we are mainly interested in the complexity of first-order programs, but which might use higher-order subprograms, e.g. with combinators such as `fold`, `map` ...

Plan

1. Source language

2. Indexed types and the time complexity soundness

3. Type inference

4. Extension and perspective

5. An example

Our approach

- based on types
- uses restriction on duplication

Our approach

- based on types
- uses restriction on duplication

logic can help !

Our approach

- based on types
- uses restriction on duplication

logic can help !

linear logic can help !

Our approach: continued

- an analysis of *sizes* by a type system
- then from size bounds deduce time bounds, since we will consider subrecursive programs (structural recursion)

Language

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
& & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
& & \texttt{iter}(M_1, M_2) \mid \texttt{ifz}(M_1, M_2) \mid \\
& & \texttt{fold}(M_1, M_2) \mid \texttt{ifn}(M_1, M_2)
\end{array}
$$

Language

integers constructors

lists constructors

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
& & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
& & \mathtt{iter}(M_1, M_2) \mid \mathtt{ifz}(M_1, M_2) \mid \\
& & \mathtt{fold}(M_1, M_2) \mid \mathtt{ifn}(M_1, M_2)
\end{array}
$$

Language

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
 & & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
 & & \mathtt{iter}(M_1, M_2) \mid \mathtt{ifz}(M_1, M_2) \mid \\
 & & \mathtt{fold}(M_1, M_2) \mid \mathtt{ifn}(M_1, M_2)
\end{array}
$$

iteration and zero-test for integers

Language

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
 & & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
 & & \mathtt{iter}(M_1, M_2) \mid \mathtt{ifz}(M_1, M_2) \mid \\
 & & \mathtt{fold}(M_1, M_2) \mid \mathtt{ifn}(M_1, M_2)
\end{array}
$$

iteration and zero-test for integers

$$\mathtt{iter}(M_1, M_2)\ \underline{3} \equiv (M_1(M_1(M_1\ M_2)))$$

Language

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
 & & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
 & & \mathtt{iter}(M_1, M_2) \mid \mathtt{ifz}(M_1, M_2) \mid \\
 & & \mathtt{fold}(M_1, M_2) \mid \mathtt{ifn}(M_1, M_2)
\end{array}
$$

iteration and zero-test for lists

$$\mathtt{iter}(M_1, M_2)\ \underline{3} \equiv (M_1(M_1(M_1\ M_2)))$$

Language

$$
\begin{array}{lcl}
M_1, M_2 & ::= & x \mid \lambda x.M_1 \mid (M_1\ M_2) \mid \\
 & & 0 \mid \mathbf{succ}(M_1) \mid \mathbf{nil} \mid \mathbf{cons}(M_1, M_2) \mid \\
 & & \texttt{iter}(M_1, M_2) \mid \texttt{ifz}(M_1, M_2) \mid \\
 & & \texttt{fold}(M_1, M_2) \mid \texttt{ifn}(M_1, M_2)
\end{array}
$$

iteration and zero-test for lists

$$\texttt{iter}(M_1, M_2)\ \underline{3} \equiv (M_1(M_1(M_1\ M_2)))$$

$$\texttt{fold}(M_1, M_2)\ [a_1; a_2; a_3] \equiv M_1(a_1(M_1(a_2 M_1(a_3 M_2))))$$

Types

- Types:

$$
\begin{array}{rcl}
T & ::= & \mathbf{B} \mid \mathbf{N} \mid \mathbf{L}(T) \\
A, B & ::= & T \mid A \multimap A
\end{array}
$$

base types

- Type contexts:

$$
\begin{array}{rcl}
\Gamma & = & x_1 : A_1, \ldots, x_n : A_n \\
\ell\Gamma & = & x_1 : T_1, \ldots, x_n : T_n
\end{array}
$$

base types context

Linearity policy

- duplication of base type values **OK**
- re-use of step function in recursion **OK**
- re-use of functional variable $\lambda f.(f\ (f\ M))$ **NO**

it will be enforced by a (first) type system, ℓT.

Source typed language : ℓT (selected rules)

$$\frac{}{\Gamma \vdash 0 : \mathbf{N}} \qquad \frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \mathtt{succ}(M) : \mathbf{N}}$$

$$\frac{\ell\Gamma, \Delta_1 \vdash M_1 : A \multimap B \qquad \ell\Gamma, \Delta_2 \vdash M_2 : A}{\ell\Gamma, \Delta_1, \Delta_2 \vdash (M_1\ M_2) : B}$$

$$\frac{\ell\Gamma \vdash M_1 : A \multimap A \qquad \ell\Gamma \vdash M_2 : A}{\ell\Gamma \vdash \mathtt{iter}(M_1, M_2) : \mathbf{N} \multimap A}$$

Source typed language : ℓT (selected rules)

$$\frac{}{\Gamma \vdash 0 : \mathbf{N}} \qquad \frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \mathtt{succ}(M) : \mathbf{N}}$$

$$\frac{\ell\Gamma, \Delta_1 \vdash M_1 : A \multimap B \qquad \ell\Gamma, \Delta_2 \vdash M_2 : A}{\ell\Gamma, \Delta_1, \Delta_2 \vdash (M_1\ M_2) : B}$$

$$\frac{\ell\Gamma \vdash M_1 : A \multimap A \qquad \ell\Gamma \vdash M_2 : A}{\ell\Gamma \vdash \mathtt{iter}(M_1, M_2) : \mathbf{N} \multimap A}$$

base types context

Examples of ℓT terms

$$\begin{aligned} \mathbf{add}\ x\ 0 &= x \\ \mathbf{add}\ x\ \mathbf{succ}(y) &= \mathbf{succ}(\mathbf{add}\ x\ y) \end{aligned}$$

$$\mathtt{add} = \lambda x.\mathtt{iter}(\lambda y.\mathbf{succ}(y), x) : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$$

$$\begin{aligned} \mathtt{mul}\ x\ 0 &= 0 \\ \mathtt{mul}\ x\ \mathbf{succ}(y) &= \mathtt{add}\ x\ (\mathtt{mul}\ x\ y) \end{aligned}$$

$$\mathtt{mul} = \lambda x.\mathtt{iter}(\mathtt{add}\ x, \mathbf{zero}) : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$$

Examples of ℓT terms

$$\begin{aligned} \textbf{sum nil} &= 0 \\ \textbf{sum cons}(x, u) &= \textbf{add } x \ (\textbf{sum } u) \end{aligned}$$

$$\texttt{sum} = \texttt{fold}(\textbf{add}, \textbf{zero}) : \quad \mathbf{L}(\mathbf{N}) \multimap \mathbf{N}$$

$$\begin{aligned} \textbf{append nil } u_2 &= u_2 \\ \textbf{append cons}(x, v)\ u_2 &= \textbf{cons}(x, (\textbf{append } v\ u_2)) \end{aligned}$$

$$\texttt{append} = \lambda u_1.\lambda u_2.(\texttt{fold}(\lambda x.\lambda v.\textbf{cons}(x, v), u_2)\ u_1) : \quad \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N})$$

Examples of ℓT terms

sum [3;2;7]=12

$$\begin{array}{lcl} \textbf{sum nil} & = & 0 \\ \textbf{sum cons}(x, u) & = & \textbf{add } x \ (\textbf{sum } u) \end{array}$$

$$\texttt{sum} \quad = \quad \texttt{fold}(\textbf{add}, \textbf{zero}) : \qquad \mathbf{L}(\mathbf{N}) \multimap \mathbf{N}$$

$$\begin{array}{lcl} \textbf{append nil } u_2 & = & u_2 \\ \textbf{append cons}(x, v)\ u_2 & = & \textbf{cons}(x, (\textbf{append } v\ u_2)) \end{array}$$

append [3;2;7] [1,4]=[3;2;7;1;4]

$$\textbf{append} \quad = \quad \lambda u_1.\lambda u_2.(\texttt{fold}(\lambda x.\lambda v.\textbf{cons}(x, v), u_2)\ u_1) : \qquad \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N})$$

Examples of ℓT terms

$$\begin{aligned} \texttt{map\_add}\ x\ \mathbf{nil} &= \mathbf{nil} \\ \texttt{map\_add}\ x\ \mathbf{cons}(y, v) &= \mathbf{cons}((\texttt{add}\ x\ y), (\texttt{map\_add}\ x\ v)) \end{aligned}$$

$$\texttt{map\_add} = \lambda x.\texttt{fold}(\lambda y.\lambda v.\mathbf{cons}(\texttt{add}\ x\ y, v), \mathbf{nil}) : \quad \mathbf{N} \multimap \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N})$$

$$\begin{aligned} \texttt{enumerate}\ 0 &= \mathbf{cons}(\mathbf{nil}, \mathbf{nil}) \\ \texttt{enumerate}\ \mathbf{succ}(x) &= \texttt{append}(\texttt{map\_cons0}(\texttt{enumerate}\ x), \texttt{map\_cons1}(\texttt{enumerate}\ x)) \end{aligned}$$

$$\texttt{enumerate} = \lambda x.\texttt{iter}(\lambda u.(\texttt{append}\ (\texttt{map\_cons0}u)\ (\texttt{map\_cons1}u)), \mathbf{cons}(\mathbf{nil}, \mathbf{nil})) : \quad \mathbf{N} \multimap \mathbf{L}(\mathbf{L}(\mathbf{N}))$$

Examples of ℓT terms

$$\begin{aligned}
\texttt{map_add}\ x\ \mathbf{nil} &= \mathbf{nil} \\
\texttt{map_add}\ x\ \mathbf{cons}(y, v) &= \mathbf{cons}((\mathrm{add}\ x\ y), (\texttt{map_add}\ x\ v))
\end{aligned}$$

map_add 2 [3;2;7]=[5;4;9]

$$\texttt{map\_add} = \lambda x.\texttt{fold}(\lambda y.\lambda v.\mathbf{cons}(\mathrm{add}\ x\ y, v), \mathbf{nil}) : \quad \mathbf{N} \multimap \mathbf{L}(\mathbf{N}) \multimap \mathbf{L}(\mathbf{N})$$

$$\begin{aligned}
\texttt{enumerate}\ 0 &= \mathbf{cons}(\mathbf{nil}, \mathbf{nil}) \\
\texttt{enumerate}\ \mathbf{succ}(x) &= \mathrm{append}(\texttt{map_cons0}(\texttt{enumerate}\ x), \texttt{map_cons1}(\texttt{enumerate}\ x))
\end{aligned}$$

enumerate nil =[nil]

enumerate 3 =[[0;0;0]; [0;0;1]; [0;1;0]; [0;1;1]; [1;0;0]; [1;0;1]; [1;1;0]; [1;1;1]]

$$\texttt{enumerate} = \lambda x.\texttt{iter}(\lambda u.(\mathrm{append}\ (\texttt{map\_cons0}u)\ (\texttt{map\_cons1}u)), \mathbf{cons}(\mathbf{nil}, \mathbf{nil})) : \quad \mathbf{N} \multimap \mathbf{L}(\mathbf{L}(\mathbf{N}))$$

Sizes

- Now, we want to track more information about the computation:
  - sizes of the values computed (size of integer, length of list)
  - time complexity of functions
- for that we will consider types enriched with size annotations (*indexes*)

Sizes

$$N^{3}\,, \qquad N^{2a+b}\,, \qquad L^{a+1}\,(N^{2b})\,,$$

Sizes: functions

$$L^a\ (N^b) \text{ -> } N^a$$

$$(L^a\ (N^b) \text{ -> } L^a\ (N^b)) \text{ -> } (L^a\ (N^b) \text{-> } N^b)$$

Sizes: functions

more generally

$$N^{g(a)} \to N^{f(a)}$$

ε first-order equational program, e.g.

$$\begin{cases} f(0) = 1 \\ f(x+1)=g(x)+f(x) \end{cases}$$

Indexes: definitions

- An «open» language of indexes: first-order terms

$$I := a \mid f(I_1, \ldots, I_k) \qquad \text{for } f \text{ in a set } \mathcal{IF}$$

and a system of equations ε defining $[\![\mathtt{f}]\!] : \mathbb{N}^k \to \mathbb{N}$

$\mathcal{IF}$ will contain such functions as 0, +, s ...

- example: ε containing

$$\begin{cases} e(0) = 1 \\ e(s(a)) = e(a) + e(a) \end{cases}$$

defines exponentiation.

dℓT types and judgements

- Indexed types:

$$U ::= \mathbf{B} \mid \mathbf{N}^I \mid \mathbf{L}^I(U)$$
$$D ::= U \mid D \multimap D$$

- Judgements:

$$\Gamma \vdash^{\mathcal{E}} M : D$$

dℓT types and judgements

- Indexed types:

$$U ::= \mathbf{B} \mid \mathbf{N}^I \mid \mathbf{L}^I(U)$$
$$D ::= U \mid D \multimap D$$

- Judgements:

$$\Gamma \vdash^{\mathcal{E}} M : D$$

first-order equational program

The need for subtyping

- example:

$$\vdash^{\mathcal{E}} \mathbf{N}^2 \sqsubseteq \mathbf{N}^5$$

- subtyping rules

$$\frac{\models^{\mathcal{E}} I \leq J}{\vdash^{\mathcal{E}} \mathbf{N}^I \sqsubseteq \mathbf{N}^J} \qquad \frac{\vdash^{\mathcal{E}} D' \sqsubseteq D \qquad \vdash^{\mathcal{E}} E \sqsubseteq E'}{\vdash^{\mathcal{E}} D \multimap E \sqsubseteq D' \multimap E'}$$

dℓT typing rules (selection)

$$\frac{\vdash^{\mathcal{E}} D \sqsubseteq E}{\Gamma, x : D \vdash^{\mathcal{E}} x : E}$$

$$\frac{\models^{\mathcal{E}} I + 1 \leq J \qquad \Gamma \vdash^{\mathcal{E}} M : \mathbf{N}^{I}}{\Gamma \vdash^{\mathcal{E}} \mathtt{succ}(M) : \mathbf{N}^{J}}$$

$$\frac{\ell\Gamma, \Delta_1 \vdash^{\mathcal{E}} M_1 : D \multimap E \qquad \ell\Gamma, \Delta_2 \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma, \Delta_1, \Delta_2 \vdash^{\mathcal{E}} (M_1 M_2) : E}$$

dℓT typing rules (selection, continued)

What about iteration ?

$$\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma \vdash^{\mathcal{E}} \texttt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D}$$

dℓT typing rules (selection, continued)

What about iteration ?

$$\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D}$$

not sufficient

dℓT typing rules (selection, continued)

What about iteration ?

$$\cancel{\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D}}$$

not sufficient

2nd try :

$$\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D\{a/a+1\} \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D\{a/1\}}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D\{a/I\}}$$

dℓT typing rules (selection, continued)

What about iteration ?

$$\cancel{\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma \vdash^{\mathcal{E}} \texttt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D}}$$

not sufficient

2nd try :

$$\cancel{\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D\{a/a+1\} \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D\{a/1\}}{\ell\Gamma \vdash^{\mathcal{E}} \texttt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D\{a/I\}}}$$

dℓT typing rules (selection, continued)

What about iteration ?

$$\cancel{\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D}}$$

not sufficient

2nd try :

$$\cancel{\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D\{a/a+1\} \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D\{a/1\}}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap D\{a/I\}}}$$

3rd try :

$$\frac{\ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D\{a/a+1\} \qquad \ell\Gamma \vdash^{\mathcal{E}} M_2 : D\{a/1\} \qquad \vdash^{\mathcal{E}} D \sqsubseteq E \qquad \vdash^{\mathcal{E}} E \sqsubseteq E\{a/a+1\}}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap E\{a/I\}}$$

OK

Examples of ℓT terms

$$\begin{aligned} \mathtt{add}\ x\ 0 &= x \\ \mathtt{add}\ x\ \mathbf{succ}(y) &= \mathbf{succ}(\mathtt{add}\ x\ y) \end{aligned}$$

$$\mathtt{add} \quad : \quad \mathbf{N}^{a} \multimap \mathbf{N}^{b} \multimap \mathbf{N}^{a+b}$$

$$\begin{aligned} \mathtt{mul}\ x\ 0 &= 0 \\ \mathtt{mul}\ x\ \mathbf{succ}(y) &= \mathtt{add}\ x\ (\mathtt{mul}\ x\ y) \end{aligned}$$

$$\mathtt{mul} \quad : \quad \mathbf{N}^{a} \multimap \mathbf{N}^{b} \multimap \mathbf{N}^{a \cdot b}$$

$$\varepsilon \begin{cases} a \cdot 0 = 0 \\ a \cdot (sb) = a \cdot b + a \end{cases}$$

Examples of ℓT terms

$$\begin{array}{lcl} \mathbf{sum}\ \mathbf{nil} & = & 0 \\ \mathbf{sum}\ \mathbf{cons}(x, u) & = & \mathbf{add}\ x\ (\mathbf{sum}\ u) \end{array}$$

$$\mathtt{sum} \quad : \quad \mathbf{L}^{a}(\mathbf{N}^{b}) \multimap \mathbf{N}^{a \cdot b}$$

$$\begin{array}{lcl} \mathbf{append}\ \mathbf{nil}\ u_2 & = & u_2 \\ \mathbf{append}\ \mathbf{cons}(x, v)\ u_2 & = & \mathbf{cons}(x, (\mathbf{append}\ v\ u_2)) \end{array}$$

$$\mathtt{append} \quad : \quad \mathbf{L}^{a}(\mathbf{N}^{c}) \multimap \mathbf{L}^{b}(\mathbf{N}^{d}) \multimap \mathbf{L}^{a+b}(\mathbf{N}^{\max(c,d)})$$

$$\varepsilon \left\{ \begin{array}{l} \max(a,0)=a \\ \max(0,b)=b \\ \max(sa,sb)=s(\max(a,b)) \end{array} \right.$$

Examples of ℓT terms

$$\begin{array}{lcl} \texttt{map\_add}\ x\ \mathbf{nil} & = & \mathbf{nil} \\ \texttt{map\_add}\ x\ \mathbf{cons}(y, v) & = & \mathbf{cons}((\texttt{add}\ x\ y), (\texttt{map\_add}\ x\ v)) \end{array}$$

$$\texttt{map\_add} \quad : \quad \mathbf{N}^{a} \multimap \mathbf{L}^{b}(\mathbf{N}^{c}) \multimap \mathbf{L}^{b}(\mathbf{N}^{a+c})$$

$$\begin{array}{lcl} \texttt{enumerate}\ 0 & = & \mathbf{cons}(\mathbf{nil}, \mathbf{nil}) \\ \texttt{enumerate}\ \mathbf{succ}(x) & = & \texttt{append}(\texttt{map\_cons0}(\texttt{enumerate}\ x), \texttt{map\_cons1}(\texttt{enumerate}\ x)) \end{array}$$

$$\texttt{enumerate} \quad : \quad \mathbf{N}^{a} \multimap \mathbf{L}^{e(a)}(\mathbf{L}^{a}(\mathbf{N}^{2}))$$

$$\varepsilon \begin{cases} e(0) = 1 \\ e(s(a)) = e(a) + e(a) \end{cases}$$

Weight of a derivation

- we associate to each derivation π an index term $\mathbb{W}(\pi)$, its *weight*, defined by induction on π. For example:

$$\pi \triangleright \frac{\vdash^{\mathcal{E}} D \sqsubseteq E}{\Gamma, x : D \vdash^{\mathcal{E}} x : E} \qquad \mathbb{W}(\pi) = 1$$

$$\pi \triangleright \frac{\rho \triangleright \ell\Gamma, \Delta_1 \vdash^{\mathcal{E}} M_1 : D \multimap E \qquad \sigma \triangleright \ell\Gamma, \Delta_2 \vdash^{\mathcal{E}} M_2 : D}{\ell\Gamma, \Delta_1, \Delta_2 \vdash^{\mathcal{E}} (M_1 M_2) : E} \qquad \mathbb{W}(\pi) = \mathbb{W}(\rho) + \mathbb{W}(\sigma) + 1$$

$$\pi \triangleright \frac{\begin{array}{c}\sigma \triangleright \ell\Gamma \vdash^{\mathcal{E}} M_1 : D \multimap D\{a/a+1\} \qquad \rho \triangleright \ell\Gamma \vdash^{\mathcal{E}} M_2 : D\{a/1\} \\ \vdash^{\mathcal{E}} D \sqsubseteq E \qquad \vdash^{\mathcal{E}} E \sqsubseteq E\{a/a+1\}\end{array}}{\ell\Gamma \vdash^{\mathcal{E}} \mathtt{iter}(M_1, M_2) : \mathbf{N}^I \multimap E\{a/I\}} \qquad \mathbb{W}(\pi) = \mathbb{W}(\rho) + \sum_{1 \leq a < I} \mathbb{W}(\sigma) + I$$

Time complexity soundness

Theorem (Complexity soundness):
If π *is derivation of* $\Gamma \vdash^{\mathcal{E}} M : D$ *then the execution of* M *on the abstract machine is done in time bounded by* $\mathbb{w}(\pi)$.

Examples

$$\begin{aligned} \mathbf{add}\ x\ 0 &= x \\ \mathbf{add}\ x\ \mathbf{succ}(y) &= \mathbf{succ}(\mathbf{add}\ x\ y) \end{aligned}$$

$$\mathtt{add} \quad : \quad \mathbf{N}^{a} \multimap \mathbf{N}^{b} \multimap \mathbf{N}^{a+b}$$

we get

$$\mathbb{W}(\mathrm{add}) = 3b+2$$

Examples

$$\begin{aligned} \texttt{sum}\ \mathbf{nil} &= 0 \\ \texttt{sum}\ \mathbf{cons}(x, u) &= \texttt{add}\ x\ (\texttt{sum}\ u) \end{aligned}$$

$$\texttt{sum} \quad : \quad \mathbf{L}^{a}(\mathbf{N}^{b}) \multimap \mathbf{N}^{a \cdot b}$$

$$\mathbb{W}(\text{sum}) \leq 2a^2 \cdot b + a + 1$$

The type inference problem

- So, given a $(d\ell T)$ type derivation for a term M^A we can deduce from it:
  - a bound on the size of resulting values
  - a bound on its time complexity
- But... given a term M^A how can we obtain a type derivation for it??

  this is the *type inference problem*

A type inference algorithm

$$TI(M^A)=(\Pi,\varepsilon)$$

- defined by induction on the type derivation for M^A
- standard for rules (var), (succ), (--o i), (--o e) ...
- ... but what about iteration ? find a suitable D is not straightforward

A type inference algorithm

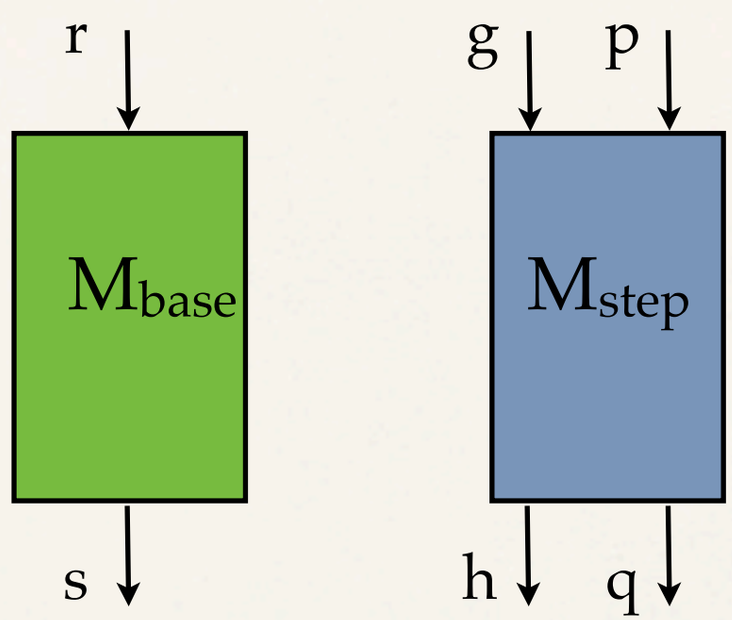
$$TI(M^A)=(\Pi,\varepsilon)$$

- defined by induction on the type derivation for M^A
- standard for rules (var), (succ), (--o i), (--o e) ...
- ... but what about iteration ? find a suitable D is not straightforward
- a solution: linear logic ‘s geometry of interaction

Typing iteration

M_{base}:$(\mathbf{N}^{r(a,b)}$-o $\mathbf{N}^{s(p,q)})$

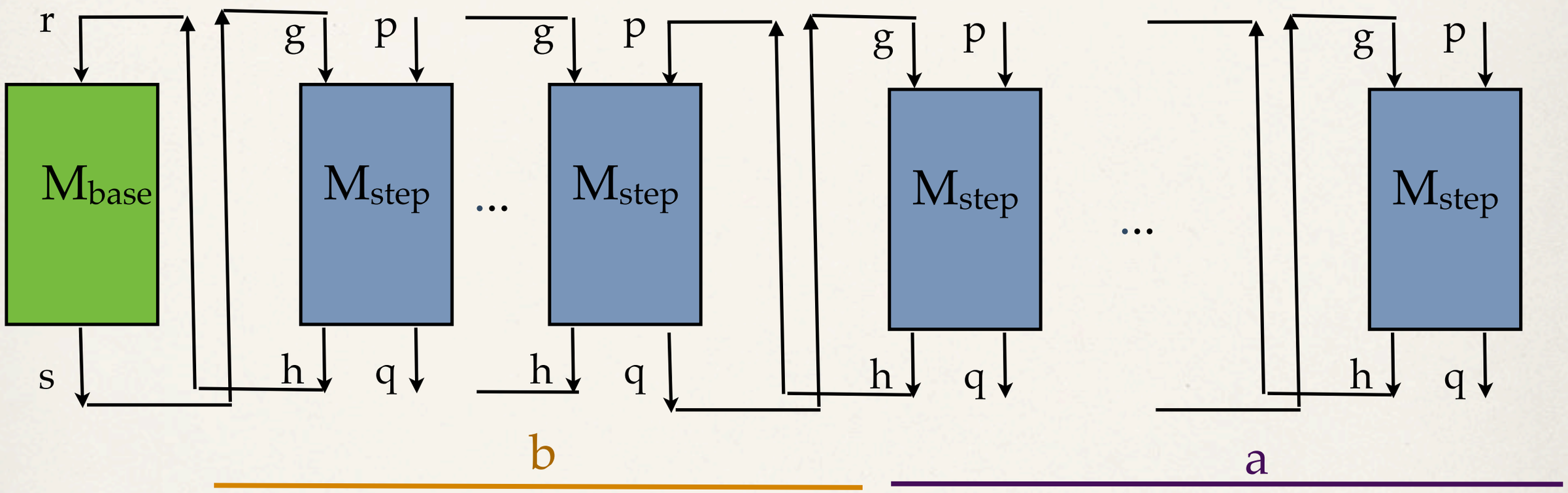
M_{step}:$(\mathbf{N}^{h(a,b)}$-o $\mathbf{N}^{g(p,q)})$ -> $(\mathbf{N}^{p(a,b)}$-o $\mathbf{N}^{q(a,b)})$



Typing iteration

M_{base}:$(\mathbf{N}^{r(a,b)}$-o $\mathbf{N}^{s(p,q)})$

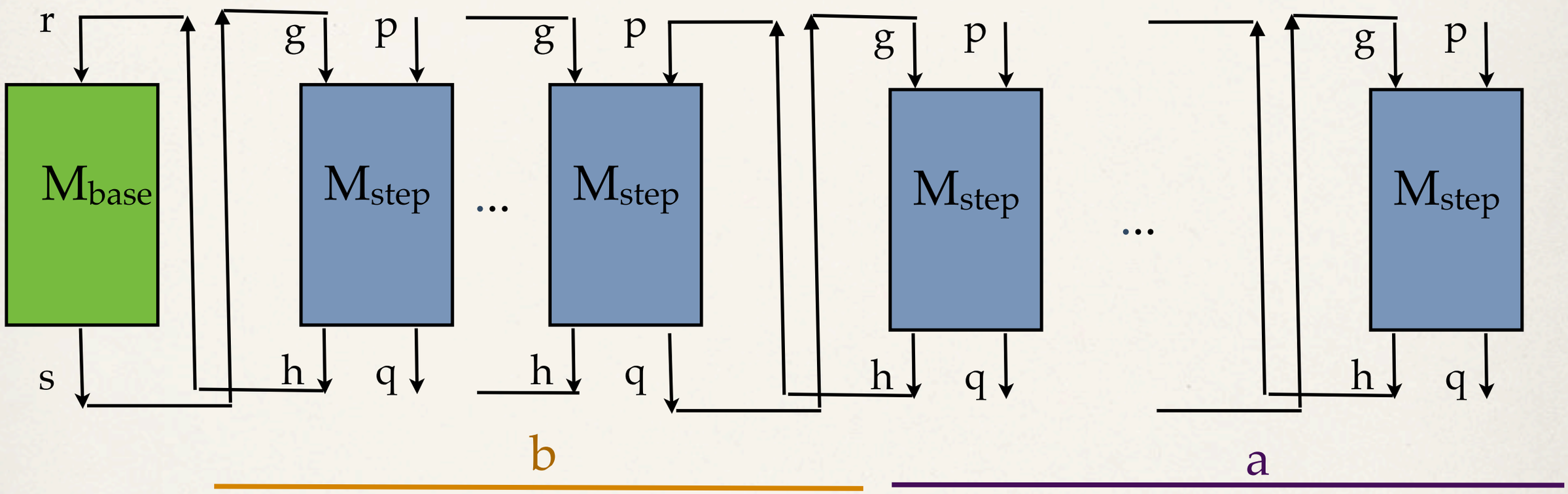
M_{step}:$(\mathbf{N}^{h(a,b)}$-o $\mathbf{N}^{g(p,q)})$ -> $(\mathbf{N}^{p(a,b)}$-o $\mathbf{N}^{q(a,b)})$



Typing iteration

M_{base}:$(\mathbf{N}^{r(a,b)}$-o $\mathbf{N}^{s(p,q)})$

M_{step}:$(\mathbf{N}^{h(a,b)}$-o $\mathbf{N}^{g(p,q)})$ -> $(\mathbf{N}^{p(a,b)}$-o $\mathbf{N}^{q(a,b)})$



ε:

$$p(a+1,b)=h(a,b+1)$$
$$g(a,b+1)=q(a+1,b)$$
$$r(a+1)=h(a,0)$$

...

Type inference properties

- The algorithm is sound:

> Thm: *if* CTI(M)=(Π,ε) *then* ε *is completely specified and* Π *is a correct* dℓT *type derivation for* M.

- The equational program terminates... for free!

> Thm: *if* CTI(M)=(Π,ε) *then* ε *is terminating*.

- Type inference always succeeds, that is to say CTI(.) is total.

What about imperative features?

- The whole approach scales to a language with references

  assign r:=M *dereference* (!r)

- ... provided one takes care of linearity conditions:

  higher-type references are read-once

- and the dℓT typing judgements need to keep track of the size-change on memory:

  type-and-effects style discipline

- finally, the theorems go through!

Summary

[B Barthe DalLago 15]

CTI algorithm for ℓT

automatic inference

closed form analysis

program $\longrightarrow$ program with indexed types $\longrightarrow$ termination of ε $\dashrightarrow$ explicit bounds

$\vdash M : A$ $\qquad$ $\vdash^{\mathcal{E}} M : D$

Towards explicit bounds

π is derivation of $\Gamma \vdash^{\mathcal{E}} M : D$

$\mathbb{W}(\pi)$ is a time bound

however instead of an equational program we would prefer explicit closed forms, e.g.

$$\mathbb{W}(\pi) \leq 2a^3 + a^2 + 1$$

Towards explicit bounds: work in progress

M. Lesourd's Master thesis (2016):

- implementation of type inference CTI(.) algorithm, in CAML
- study of the shape of index equational programs
- attempts for searching for closed-forms bounds

Towards explicit bounds: M. Lesourd's Master thesis

- grammar of index equational programs:

  $t ::= 0 \mid a \mid a+1$

  $r ::= 0 \mid a \mid f(\underline{a})+1 \mid f(\underline{t}) \mid \max(f(\underline{a}), g(\underline{a})) \mid \max_{v \leq f(\underline{a})} (g(\underline{a}))$

  where $f(\underline{a})$ denotes $f(a_1,\ldots,a_n)$

- equational programs ε generated by type inference can contain:

  $f(\underline{a},0,\underline{b}) = g(t_1)$

  $f(\underline{a},c+1,\underline{b}) = g(t_2)$

  $f(\underline{a}) = r$

Towards explicit bounds: M. Lesourd's Master thesis

- grammar of index equational programs:

  $t ::= 0 \mid a \mid a+1$

  $r ::= 0 \mid a \mid f(\underline{a})+1 \mid f(\underline{t}) \mid \max(f(\underline{a}), g(\underline{a})) \mid \max_{v \leq f(\underline{a})} (g(\underline{a}))$

  where $f(\underline{a})$ denotes $f(a_1,\ldots,a_n)$

- equational programs ε generated by type inference can contain:

  $f(\underline{a},0,\underline{b}) = g(t_1)$

  $f(\underline{a},c+1,\underline{b}) = g(t_2)$

  $f(\underline{a}) = r$

can we get a closed form (upper bound) with a solver ?

Towards explicit bounds: M. Lesourd's Master thesis

- attempts for searching for closed-forms bounds:
  - off-the-shelf PUBS solver (COSTA) for cost relations but insufficient
  - look for specific solving heuristics
  - other solvers?

An example: a cryptographic reduction

- Example of hardcore predicate.
- If $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function, then $g_f : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined by $g_f(x,y)=(f(x),y)$ is also one-way.
- A **hardcore predicate** p for a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ is a function which is efficiently computable from $\{0,1\}^n$ to $\{0,1\}$ such that it is difficult to guess p(x), when one only knows f(x).

Theorem 1 (Goldreich-Levin) *If f is one-way, then the function p defined by $\mathsf{p}(x,y) = \bigoplus_{i=1}^{n} x_i \cdot y_i$ is a hardcore predicate for the function g_f.*

Example: hardcore predicate

$\mathcal{A}_1$ adversary for

$\mathsf{p}(x, y) = \bigoplus_{i=1}^{n} x_i \cdot y_i$

is a hardcore predicate for g_f

$\mathcal{A}_2$ adversary for

g_f is a one-way function

If $\mathcal{A}_1$ is PPT, is $\mathcal{A}_2$ also PPT ?

.

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.
2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.
3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$
   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).
5. Output $x = x_1 \cdots x_n$.

Example: reduction, written in ℓT

$\lambda y.$	*argument of the inverter*
$\texttt{let}\ \ell = \lceil \log(n+1) \rceil$	*defines* ℓ
$\texttt{let}\ \mathbf{P} = \mathsf{pow}_0\ \ell\ \texttt{in}$	*enumerates all non-empty subsets of* $\{1, \ldots, \ell\}$
$\texttt{let}\ R = \mathsf{map}\ (\mathsf{flip})\ *^{\ell}\ \texttt{in}$	*samples uniformly at random* $(r_1, \ldots, r_\ell)$
$\texttt{let}\ Z = \mathsf{map}\ (\lambda\_.\ \mathsf{map}\ (\mathsf{flip})\ *^{n})\ *^{\ell}\ \texttt{in}$	*samples uniformly at random* $(z_1, \ldots, z_\ell)$
$\texttt{let}\ g_r = \lambda X.\mathsf{fold}\ (0, \oplus)\ (\mathsf{map}_2(\otimes)\ X\ R)\ \texttt{in}$	
$\texttt{let}\ \bigotimes = \lambda x.\mathsf{map}\ (\otimes\ x)\ \texttt{in}$	
$\texttt{let}\ \bigoplus = \mathsf{map}_2\ (\oplus)\ \texttt{in}$	
$\texttt{let}\ g_z = \lambda X.\mathsf{fold}\ (0^n,\ \bigoplus)\ (\mathsf{map}_2\ (\bigotimes)\ X\ Z)\ \texttt{in}$	
$\texttt{let}\ R^P = \mathsf{map}\ (g_r)\ \mathbf{P}\ \texttt{in}$	*computes the list* $(r^X)_{X \in \mathbf{P}}$
$\texttt{let}\ Z^P = \mathsf{map}\ (g_z)\ \mathbf{P}\ \texttt{in}$	*computes the list* $(z^X)_{X \in \mathbf{P}}$
$\texttt{let}\ G = \lambda i.\ \mathsf{majority}\ (\mathsf{map}_2\ (\lambda r\ z.\ \oplus\ r\ (\mathcal{A}\ (\mathsf{app}\ y\ (\mathsf{zerobut}\ i\ z))))\ R^P\ Z^P)\ \texttt{in}$	
$\mathsf{map}\ (G)(1, \ldots, n)$	

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.
2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.
3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$
   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).
5. Output $x = x_1 \cdots x_n$.

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

$L^a(B)$

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.
2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.
3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$
   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).
5. Output $x = x_1 \cdots x_n$.

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

N^a

$L^a(B)$

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.
2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.
3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$
   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).
5. Output $x = x_1 \cdots x_n$.

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

N^a

$L^a(B)$

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.

$N^{\log(r(a))}$

2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.
3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$
   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).
5. Output $x = x_1 \cdots x_n$.

Example: reduction for hardcore predicate [KatzLindell]

The inversion algorithm $\mathcal{A}'$. We now provide a full description of an algorithm $\mathcal{A}'$ that receives input y and tries to compute an inverse of y. The algorithm proceeds as follows:

N^a

$L^a(B)$

1. Set $n := |y|$ and $\ell := \lceil \log(2n/\varepsilon(n)^2 + 1) \rceil$.

$N^{\log(r(a))}$

2. Choose $s^1, \ldots, s^\ell \leftarrow \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^\ell \leftarrow \{0,1\}$ uniformly at random.

$L^{\log(r(a))}(B)$

3. For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set $r^I := \oplus_{i \in I} s^i$ and compute $\rho^I := \oplus_{i \in I} \sigma^i$.

4. For $i = 1, \ldots, n$:

   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set

   $$x_i^I := \rho^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$

   (b) Set $x_i := \mathsf{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).

5. Output $x = x_1 \cdots x_n$.

Example: reduction for hardcore predicate [KatzLindell]

- in this example we obtain for the type derivation of the inverter $\mathcal{A}'$ the weight

$$\mathbb{W}(\pi)=O(n^2 f_{\mathcal{A}}(1+2n)),$$

where $f_{\mathcal{A}}$ is a function bounding the complexity of adversary $\mathcal{A}$.

- the bound given by the soundness thm for the complexity of $\mathcal{A}'$ corresponds with the one obtained by a complexity analysis *by hand*.

Conclusion and perspectives

- a language and a type system to analyse the complexity of higher-order functional programs with structural recursion
- perspective: methods for finding closed-form bounds?
- possible extensions: non-linearity; full recursion (while loops)...
- application to cryptographic reductions and integration into verification tool, such as Easycrypt