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
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 $\rightarrow$ cost equations $\rightarrow$ 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

\[ M_1, M_2 ::= x \mid \lambda x. M_1 \mid (M_1 \ M_2) \mid 0 \mid \text{succ}(M_1) \mid \text{nil} \mid \text{cons}(M_1, M_2) \mid \text{iter}(M_1, M_2) \mid \text{ifz}(M_1, M_2) \mid \text{fold}(M_1, M_2) \mid \text{ifn}(M_1, M_2) \]
Language

**Integers Constructors**

\[
M_1, M_2 ::= x \mid \lambda x. M_1 \mid \text{succ}(M_1) \mid \text{iter}(M_1, M_2) \mid \text{fold}(M_1, M_2) \mid (M_1, M_2)
\]

**Lists Constructors**

\[
\text{nil} \mid \text{cons}(M_1, M_2) \mid \text{ifz}(M_1, M_2) \mid \text{ifn}(M_1, M_2)
\]
Language

\[ M_1, M_2 ::= \begin{array}{|c|c|c|c|c|}
\hline
x & \lambda x. M_1 & (M_1 \ M_2) \\
\hline
0 & \text{succ}(M_1) & \text{nil} & \text{cons}(M_1, M_2) \\
\hline
\text{iter}(M_1, M_2) & \text{ifz}(M_1, M_2) & \text{fold}(M_1, M_2) & \text{ifn}(M_1, M_2) \\
\hline
\end{array} \]
Language

\[ M_1, M_2 ::= x \mid \lambda x. M_1 \mid (M_1 \ M_2) \mid 0 \mid \text{succ}(M_1) \mid \text{nil} \mid \text{cons}(M_1, M_2) \mid \text{iter}(M_1, M_2) \mid \text{ifz}(M_1, M_2) \mid \text{fold}(M_1, M_2) \mid \text{ifn}(M_1, M_2) \]

\text{iter}(M_1, M_2) 3 \equiv (M_1(M_1(M_1 \ M_2)))
\[
M_1, M_2 ::= x \mid \lambda x. M_1 \mid (M_1 \ M_2) \mid 0 \mid \text{succ}(M_1) \mid \text{nil} \mid \text{cons}(M_1, M_2) \mid \text{iter}(M_1, M_2) \mid \text{ifz}(M_1, M_2) \mid \text{fold}(M_1, M_2) \mid \text{ifn}(M_1, M_2)
\]

\[
\text{iter}(M_1, M_2) 3 \equiv (M_1(M_1(M_1 M_2)))
\]
**Language**

\[ M_1, M_2 ::= x \mid \lambda x. M_1 \mid (M_1 M_2) \mid \\
\mid 0 \mid \text{succ}(M_1) \mid \text{nil} \mid \text{cons}(M_1, M_2) \mid \\
\mid \text{iter}(M_1, M_2) \mid \text{ifz}(M_1, M_2) \mid \\
\mid \text{fold}(M_1, M_2) \mid \text{ifn}(M_1, M_2) \]

*iteration and zero-test for lists*

\[ \text{iter}(M_1, M_2) \ 3 \equiv (M_1(M_1(M_1 \ 2))) \]

\[ \text{fold}(M_1, M_2) [a_1; a_2; a_3] \equiv M_1(a_1(M_1(a_2M_1(a_3M_2)))) \]
Types

Types:

\[
T ::= B \mid N \mid L(T) \quad \text{base types}
\]

\[
A, B ::= T \mid A \rightarrow A
\]

Type contexts:

\[
\Gamma = x_1 : A_1, \ldots, x_n : A_n \quad \text{base types context}
\]

\[
\ell\Gamma = x_1 : T_1, \ldots, x_n : T_n
\]
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, \( \ell \Gamma \).
Source typed language: $\ell T$ (selected rules)

\[
\begin{array}{c}
\Gamma \vdash 0 : N \\
\hline
\Gamma \vdash M : N \\
\hline
\Gamma \vdash \text{succ}(M) : N
\end{array}
\]

\[
\ell \Gamma, \Delta_1 \vdash M_1 : A \rightarrow B \\
\ell \Gamma, \Delta_2 \vdash M_2 : A \\
\ell \Gamma, \Delta_1, \Delta_2 \vdash (M_1 \ M_2) : B
\]

\[
\ell \Gamma \vdash M_1 : A \rightarrow A \\
\ell \Gamma \vdash M_2 : A \\
\ell \Gamma \vdash \text{iter}(M_1, M_2) : N \rightarrow A
\]
Source typed language: $\mathcal{L}T$ (selected rules)

\[
\begin{align*}
\Gamma \vdash 0 : \textbf{N} & \\
\Gamma \vdash M : \textbf{N} & \Rightarrow \Gamma \vdash \text{succ}(M) : \textbf{N}
\end{align*}
\]

\[
\begin{align*}
\ell\Gamma, \Delta_1 \vdash M_1 : A \rightarrow B & \quad \ell\Gamma, \Delta_2 \vdash M_2 : A \\
\ell\Gamma, \Delta_1, \Delta_2 \vdash (M_1, M_2) : B
\end{align*}
\]

\[
\begin{align*}
\ell\Gamma \vdash M_1 : A \rightarrow A & \quad \ell\Gamma \vdash M_2 : A \\
\ell\Gamma \vdash \text{iter}(M_1, M_2) : \textbf{N} \rightarrow A
\end{align*}
\]

base types context
Examples of $\lambda T$ terms

\[
\begin{align*}
\text{add } x 0 &= x \\
\text{add } x \text{ succ}(y) &= \text{succ}(\text{add } x y)
\end{align*}
\]

\[
\text{add} = \lambda x.\text{iter}(\lambda y.\text{succ}(y), x) : N \rightarrow N \rightarrow N
\]

\[
\begin{align*}
\text{mul } x 0 &= 0 \\
\text{mul } x \text{ succ}(y) &= \text{add } x (\text{mul } x y)
\end{align*}
\]

\[
\text{mul} = \lambda x.\text{iter}(\text{add } x, \text{zero}) : N \rightarrow N \rightarrow N
\]
Examples of λT terms

$$\begin{align*}
\text{sum nil} & = 0 \\
\text{sum cons}(x, u) & = \text{add } x \ (\text{sum } u)
\end{align*}$$

$$\begin{align*}
\text{sum} & = \text{fold}(\text{add}, \text{zero}) : \ L(N) \rightarrow N \\
\text{append nil } u_2 & = u_2 \\
\text{append cons}(x, v) u_2 & = \text{cons}(x, (\text{append } v \ u_2))
\end{align*}$$

$$\begin{align*}
\text{append} & = \lambda u_1.\lambda u_2. (\text{fold}(\lambda x.\lambda v.\text{cons}(x, v), u_2) u_1) : \ L(N) \rightarrow L(N) \rightarrow L(N)
\end{align*}$$
Examples of $\lambda$T terms

\[ \text{sum } [3;2;7] = 12 \]

\[ \text{sum nil} = 0 \]
\[ \text{sum cons}(x, u) = \text{add } x \text{ (sum } u) \]

\[ \text{sum} = \text{fold} \text{(add, zero)} : \text{ L(N) } \rightarrow \text{ N} \]

\[ \text{append nil } u_2 = u_2 \]
\[ \text{append cons}(x, v) u_2 = \text{cons}(x, (\text{append } v u u_2)) \]

\[ \text{append } [3;2;7] [1;4] = [3;2;7;1;4] \]

\[ \text{append} = \lambda u_1.\lambda u_2. (\text{fold} \text{(lx.lu.cons}(x, v), u_2) u_1) : \text{ L(N) } \rightarrow \text{ L(N) } \rightarrow \text{ L(N)} \]
Examples of $\mathcal{L}T$ terms

\[
\text{map\_add } x \text{ nil } = \text{ nil} \\
\text{map\_add } x \text{ cons}(y, v) = \text{ cons}((\text{add } x \ y), (\text{map\_add } x \ v))
\]

\[
\text{map\_add} = \lambda x. \text{fold}(\lambda y. \lambda v. \text{cons}(\text{add } x \ y, v), \text{nil}) : \quad \mathbb{N} \rightarrow \text{L}(\mathbb{N}) \rightarrow \text{L}(\mathbb{N})
\]

\[
\text{enumerate } 0 = \text{cons}(\text{nil}, \text{nil}) \\
\text{enumerate succ}(x) = \text{append}(\text{map\_cons0}(\text{enumerate } x), \text{map\_cons1}(\text{enumerate } x))
\]

\[
\text{enumerate} = \lambda x. \text{iter}(\lambda u. (\text{append} (\text{map\_cons0} u) (\text{map\_cons1} u)), \text{cons}(\text{nil}, \text{nil}) : \quad \mathbb{N} \rightarrow \text{L}(\text{L}(\mathbb{N}))
\]
Examples of \( \lambda \text{T} \) terms

\[
\text{map}_\text{add} \ x \ \text{nil} \quad = \quad \text{nil}
\]
\[
\text{map}_\text{add} \ x \ \text{cons}(y, v) \quad = \quad \text{cons}((\text{add} \ x \ y), (\text{map}_\text{add} \ x \ v))
\]

\[
\text{map}_\text{add} \ 2 \ [3;2;7]= [5;4;9]
\]

\[
\text{map}_\text{add} \quad = \quad \lambda x.\text{fold}(\lambda y.\lambda v.\text{cons}(\text{add} \ x \ y, v), \text{nil}) : \quad \mathbb{N} \rightarrow \mathbb{L}(\mathbb{N}) \rightarrow \mathbb{L}(\mathbb{N})
\]

\[
\text{enumerate} \ 0 \quad = \quad \text{cons}(\text{nil}, \text{nil})
\]
\[
\text{enumerate} \ \text{succ}(x) \quad = \quad \text{append}(\text{map}_\text{cons0}(\text{enumerate} \ x), \text{map}_\text{cons1}(\text{enumerate} \ x))
\]

\[
\text{enumerate} \ \text{nil} = [\text{nil}]
\]
\[
\text{enumerate} \ 3 = [[0;0;0]; [0;0;1]; [0;1;0]; [0;1;1]; [1;0;0]; [1;0;1]; [1;1;0]; [1;1;1]]
\]
\[
\text{enumerate} \quad = \quad \lambda x.\text{iter}(\lambda u. (\text{append} \ (\text{map}_\text{cons0} u) \ (\text{map}_\text{cons1} u)), \text{cons}(\text{nil}, \text{nil})) : \quad \mathbb{N} \rightarrow \mathbb{L}(\mathbb{L}(\mathbb{N}))
\]
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

$N^3, \quad N^{2a+b}, \quad L^{a+1}(N^{2b})$, 
Sizes: functions

\[ L^a (N^b) \rightarrow N^a \]

\[(L^a (N^b) \rightarrow L^a (N^b)) \rightarrow (L^a (N^b) \rightarrow N^b)\]
Sizes: functions

more generally

\[ \mathbb{N} g(a) \rightarrow \mathbb{N} f(a) \]

\[ \varepsilon \text{ first-order equational program, e.g.} \]

\[
\begin{align*}
  f(0) & = 1 \\
  f(x+1) &= g(x) + f(x)
\end{align*}
\]
Indexes: definitions

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

\[ I := \{ a \mid f(I_1, ..., I_k) \} \quad \text{for } f \text{ in a set } \mathcal{IF} \]

and a system of equations \( \varepsilon \) defining \( [f] : \mathbb{N}^k \rightarrow \mathbb{N} \)

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

- example: \( \varepsilon \) containing

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

defines exponentiation.
ΔL types and judgements

* Indexed types:

\[ U ::= \mathbf{B} \mid N^I \mid L^I(U) \]
\[ D ::= U \mid D \rightarrow D \]

* Judgements:

\[ \Gamma \vdash^\mathcal{E} M : D \]
\[ \Gamma \vdash \mathcal{E} \mathcal{M} : D \]

**Indexed types:**

\[
\begin{align*}
U &::= B \mid N^I \mid L^I(U) \\
D &::= U \mid D \rightarrow D
\end{align*}
\]

**Judgements:**

- First-order equational program
The need for subtyping

example:

\[ \vdash^{\mathcal{E}} \mathbb{N}^2 \subseteq \mathbb{N}^5 \]

subtyping rules

\[ \vdash^{\mathcal{E}} I \leq J \quad \frac{\vdash^{\mathcal{E}} D' \subseteq D \quad \vdash^{\mathcal{E}} E \subseteq E'}{\vdash^{\mathcal{E}} D \rightarrow E \subseteq D' \rightarrow E'} \quad \frac{\vdash^{\mathcal{E}} N^I \subseteq N^J}{\vdash^{\mathcal{E}} N^J \subseteq N^J} \]
\[
\frac{\vdash \mathcal{E} \ D \supseteq \ E}{\Gamma, \ x : \ D \vdash \mathcal{E} \ x : \ E}
\]

\[
\frac{\vdash \mathcal{E} \ I + 1 \leq \ J \quad \Gamma \vdash \mathcal{E} \ M : \mathbb{N}^I}{\Gamma \vdash \mathcal{E} \ \text{succ}(M) : \mathbb{N}^J}
\]

\[
\frac{\ell\Gamma, \Delta_1 \vdash \mathcal{E} \ M_1 : \ D \rightarrow E \quad \ell\Gamma, \Delta_2 \vdash \mathcal{E} \ M_2 : \ D}{\ell\Gamma, \Delta_1, \Delta_2 \vdash \mathcal{E} \ (M_1 M_2) : E}
\]
d\ell T typing rules (selection, continued)

What about iteration?

\[
\ell \Gamma \vdash^{\mathcal{E}} M_1 : D \rightarrow D \quad \ell \Gamma \vdash^{\mathcal{E}} M_2 : D \\
\ell \Gamma \vdash^{\mathcal{E}} \text{iter}(M_1, M_2) : \mathbb{N}^I \rightarrow D
\]
What about iteration?

\[ \ell \Gamma \vdash^\varepsilon M_1 : D \rightarrow D \quad \ell \Gamma \vdash^\varepsilon M_2 : D \]

\[ \ell \Gamma \vdash^\varepsilon \text{iter}(M_1, M_2) : N^I \rightarrow D \]

not sufficient
d\ell T typing rules (selection, continued)

What about iteration?

\[
\begin{align*}
\ell \Gamma \vdash^\varepsilon M_1 : D \rightarrow D & \quad \ell \Gamma \vdash^\varepsilon M_2 : D \\
\ell \Gamma \vdash^\varepsilon & \text{iter}(M_1, M_2) : \mathbb{N}^I \rightarrow D \\
\end{align*}
\]

2nd try:

\[
\begin{align*}
\ell \Gamma \vdash^\varepsilon M_1 : D \rightarrow D\{a/a + 1\} & \quad \ell \Gamma \vdash^\varepsilon M_2 : D\{a/1\} \\
\ell \Gamma \vdash^\varepsilon & \text{iter}(M_1, M_2) : \mathbb{N}^I \rightarrow D\{a/I\} \\
\end{align*}
\]

not sufficient
d\ell T typing rules (selection, continued)

What about iteration?

2nd try:

\[
\begin{align*}
\ell \Gamma & \vdash^\varepsilon M_1 : D \to D \\
\ell \Gamma & \vdash^\varepsilon M_2 : D \\
\ell \Gamma & \vdash^\varepsilon \text{iter}(M_1, M_2) : \mathbb{N}^I \to D \\
\ell \Gamma & \vdash^\varepsilon M_1 : D \to D\{a/a + 1\} \\
\ell \Gamma & \vdash^\varepsilon M_2 : D\{a/1\} \\
\ell \Gamma & \vdash^\varepsilon \text{iter}(M_1, M_2) : \mathbb{N}^I \to D\{a/I\}
\end{align*}
\]

not sufficient
What about iteration?

\[
\ell \Gamma \vdash^\varepsilon M_1 : D \to D \quad \ell \Gamma \vdash^\varepsilon M_2 : D \\
\ell \Gamma \vdash^\varepsilon \text{iter}(M_1, M_2) : \mathbb{N}^I \to D
\]

2nd try:

\[
\ell \Gamma \vdash^\varepsilon M_1 : D \to D[a/a + 1] \quad \ell \Gamma \vdash^\varepsilon M_2 : D[a/1] \\
\ell \Gamma \vdash^\varepsilon \text{iter}(M_1, M_2) : \mathbb{N}^I \to D[a/I]
\]

not sufficient

3rd try:

\[
\ell \Gamma \vdash^\varepsilon M_1 : D \to D[a/a + 1] \quad \ell \Gamma \vdash^\varepsilon M_2 : D[a/1] \\
\vdash^\varepsilon D \sqsubseteq E \quad \vdash^\varepsilon E \sqsubseteq E[a/a + 1] \\
\ell \Gamma \vdash^\varepsilon \text{iter}(M_1, M_2) : \mathbb{N}^I \to E[a/I]
\]

OK
Examples of $\mathcal{L}T$ terms

\[\begin{align*}
\text{add} \ : \ N^a \rightarrow N^b \rightarrow N^{a+b} \\
\text{mul} \ : \ N^a \rightarrow N^b \rightarrow N^{a\cdot b}
\end{align*}\]

\[\begin{align*}
\text{add} \ x \ 0 & = x \\
\text{add} \ x \ \text{succ}(y) & = \text{succ}(\text{add} \ x \ y)
\end{align*}\]

\[\begin{align*}
\text{mul} \ x \ 0 & = 0 \\
\text{mul} \ x \ \text{succ}(y) & = \text{add} \ x \ (\text{mul} \ x \ y)
\end{align*}\]

\[\begin{align*}
a \cdot 0 & = 0 \\
a \cdot (sb) & = a \cdot b + a
\end{align*}\]
Examples of $\lambda T$ terms

\[
\begin{align*}
\text{sum} & \quad : \quad L^a(N^b) \rightarrow N^{a\cdot b} \\
\text{sum nil} & \quad = \quad 0 \\
\text{sum cons}(x, u) & \quad = \quad \text{add } x \text{ (sum } u) \\
\text{append nil } u_2 & \quad = \quad u_2 \\
\text{append cons}(x, v) u_2 & \quad = \quad \text{cons}(x, (\text{append } v \ u_2)) \\
\text{append nil } u_2 & \quad = \quad u_2 \\
\text{append cons}(x, v) u_2 & \quad = \quad \text{cons}(x, (\text{append } v \ u_2)) \\
\end{align*}
\]

\[
\begin{align*}
\text{append} & \quad : \quad L^a(N^c) \rightarrow L^b(N^d) \rightarrow L^{a+b\cdot \max(c,d)}(N) \\
\varepsilon & \quad \left\{ \begin{array}{l}
\text{max}(a,0)=a \\
\text{max}(0,b)=b \\
\text{max}(sa,sb)=s(\text{max}(a,b))
\end{array} \right.
\end{align*}
\]
Examples of \( \lambda \)T terms

\[
\begin{align*}
\text{map}\_\text{add} \ x \ \text{nil} &= \text{nil} \\
\text{map}\_\text{add} \ x \ \text{cons}(y, v) &= \text{cons}((\text{add} \ x \ y), (\text{map}\_\text{add} \ x \ v))
\end{align*}
\]

\[
\text{map}\_\text{add} : \ N^a \rightarrow L \ (N^b) \rightarrow L \ (N^{a+c})
\]

\[
\begin{align*}
\text{enumerate} \ 0 &= \text{cons}(\text{nil}, \text{nil}) \\
\text{enumerate succ}(x) &= \text{append}((\text{map}\_\text{cons0}(\text{enumerate} \ x), \text{map}\_\text{cons1}(\text{enumerate} \ x)))
\end{align*}
\]

\[
\text{enumerate} : \ N^a \rightarrow L \ (L \ (N^2))
\]

\[
\varepsilon \left\{ \begin{array}{l}
e(0) = 1 \\
e(s(a)) = e(a) + e(a)
\end{array} \right. \]
Weight of a derivation

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

\[
\pi \triangleright \quad \frac{\vdash^E D \sqsubseteq E}{\Gamma, x : D \vdash^E x : E} \quad W(\pi) = 1
\]

\[
\pi \triangleright \quad \frac{\rho \triangleright \ell \Gamma, \Delta_1 \vdash^E M_1 : D \rightarrow E \quad \sigma \triangleright \ell \Gamma, \Delta_2 \vdash^E M_2 : D}{\ell \Gamma, \Delta_1, \Delta_2 \vdash^E (M_1 M_2) : E} \quad W(\pi) = W(\rho) + W(\sigma) + 1
\]

\[
\pi \triangleright \quad \frac{\sigma \triangleright \ell \Gamma \vdash^E M_1 : D \rightarrow D\{a/a + 1\} \quad \rho \triangleright \ell \Gamma \vdash^E M_2 : D\{a/1\}}{\vdash^E D \sqsubseteq E \quad \vdash^E E \sqsubseteq E\{a/a + 1\}} \quad W(\pi) = W(\rho) + \sum_{1 \leq a < I} W(\sigma) + I
\]

\[
\ell \Gamma \vdash^E \text{iter}(M_1, M_2) : N^I \rightarrow E\{a/I\}
\]
Time complexity soundness

**Theorem (Complexity soundness):**

If $\pi$ is derivation of $\Gamma \vdash_{\mathcal{E}} M : D$ then the execution of $M$ on the abstract machine is done in time bounded by $\mathcal{W}(\pi)$. 
Examples

\[
\begin{align*}
\text{add } x \ 0 &= x \\
\text{add } x \ \text{succ}(y) &= \text{succ}(\text{add } x \ y)
\end{align*}
\]

\[
\text{add : } \mathbb{N}^a \to \mathbb{N}^b \to \mathbb{N}^{a+b}
\]

we get

\[w(\text{add}) = 3b + 2\]
Examples

\[
\begin{align*}
\text{sum } \text{nil} & = 0 \\
\text{sum } \text{cons}(x, u) & = \text{add } x \ (\text{sum } u)
\end{align*}
\]

\[
\text{sum} : L^a(N^b) \rightarrow N^{a \cdot b}
\]

\[\mathfrak{w}(\text{sum}) \leq 2a^2 \cdot b + a + 1\]
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

\[ \text{TI}(M^A) = (\Pi, \varepsilon) \]

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

\[ \text{TI}(M^A) = (\Pi, \varepsilon) \]

- 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

\[ M_{\text{base}}: (N^{r(a,b)} - o N^{s(p,q)}) \]

\[ M_{\text{step}}: (N^{h(a,b)} - o N^{g(p,q)}) \rightarrow (N^{p(a,b)} - o N^{q(a,b)}) \]
Typing iteration

\[ M_{\text{base}}: (N^{r(a,b)} - o N^{s(p,q)}) \]
\[ M_{\text{step}}: (N^{g(a,b)} - o N^{h(p,q)}) -> (N^{p(a,b)} - o N^{q(a,b)}) \]
Typing iteration

\[ M_{\text{base}} : (N^{r(a,b)} \circ N^{s(p,q)}) \]
\[ M_{\text{step}} : (N^{h(a,b)} \circ N^{g(p,q)}) \rightarrow (N^{p(a,b)} \circ N^{q(a,b)}) \]

\[ p(a+1,b) = h(a,b+1) \]
\[ g(a,b+1) = q(a+1,b) \]
\[ r(a+1) = h(a,0) \]

\[ \varepsilon : \]

Diagram:
Type inference properties

- The algorithm is sound:

Thm: if $\text{CTI}(M) = (\Pi, \varepsilon)$ then $\varepsilon$ is completely specified and $\Pi$ is a correct $\mathcal{d\ell T}$ type derivation for $M$.

- The equational program terminates... for free!

Thm: if $\text{CTI}(M) = (\Pi, \varepsilon)$ then $\varepsilon$ is terminating.

- Type inference always succeeds, that is to say $\text{CTI}(.)$ is total.
What about imperative features?

- The whole approach scales to a language with references

\[
\text{assign} \quad r := M \quad \text{dereference} \quad (!r)
\]

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

  higher-type references are read-once

- and the d\textit{dT} typing judgements need to keep track of the size-change on memory:

  type-and-effects style discipline

- finally, the theorems go through!
Summary

CTI algorithm for $\mathcal{ET}$

program $\vdash M : A$

program with indexed types $\vdash^\mathcal{E} M : D$

termination of $\epsilon$ $\Rightarrow$ explicit bounds

[B Barthe DalLago 15]
Towards explicit bounds

\[ \pi \text{ is derivation of } \quad \Gamma \vdash \varepsilon \quad M : D \]

\[ \mathcal{W}(\pi) \text{ is a time bound} \]

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

\[ \mathcal{W}(\pi) \leq 2a^3 + a^2 + 1 \]
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:
  
  \[
  t ::= 0 \mid a \mid a+1
  \]
  
  \[
  r ::= 0 \mid a \mid f(a)+1 \mid f(t) \mid \max(f(a), g(a)) \mid \max_{v \leq f(a)} (g(a))
  \]
  
  where \( f(a) \) denotes \( f(a_1, \ldots, a_n) \)

* equational programs \( \mathcal{E} \) generated by type inference can contain:
  
  \[
  f(a,0,b) = g(t_1)
  \]
  
  \[
  f(a,c+1,b) = g(t_2)
  \]
  
  \[
  f(a) = r
  \]
Towards explicit bounds: M. Lesourd’s Master thesis

* grammar of index equational programs:

\[ t ::= 0 \mid a \mid a+1 \]
\[ r ::= 0 \mid a \mid f(a)+1 \mid f(t) \mid \max(f(a), g(a)) \mid \max v \leq f(a) (g(a)) \]

where \( f(a) \) denotes \( f(a_1,\ldots,a_n) \)

* equational programs \( c \) generated by type inference can contain:

\[ f(a,0,b) = g(t_1) \]
\[ f(a,c+1,b) = g(t_2) \]
\[ f(a) = r \]

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 \( f : \{0,1\}^n \rightarrow \{0,1\}^n \) is a one-way function, then \( g_f : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n} \) defined by \( g_f (x,y)=(f(x),y) \) is also one-way.

- A **hardcore predicate** \( p \) for a one-way function \( f : \{0,1\}^n \rightarrow \{0,1\}^n \) 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 \( p(x,y) = \bigoplus_{i=1}^{n} x_i \cdot y_i \) is a hardcore predicate for the function \( g_f \).
Example: hardcore predicate

$A_1$ adversary for

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

is a hardcore predicate for $g_f$

$A_2$ adversary for

$g_f$ is a one-way function

If $A_1$ is PPT, is $A_2$ also PPT?
The inversion algorithm $A'$. We now provide a full description of an algorithm $A'$ 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 $r^I := \bigoplus_{i \in I} s^i$ and compute $\rho^I := \bigoplus_{i \in I} \sigma^i$.

4. For $i = 1, \ldots, n$:
   
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   
   $$x_i^I := \rho^I \oplus A(y, r^I \oplus e^i).$$

   (b) Set $x_i := \text{majority}_I\{x_i^I\}$ (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]
Example: reduction, written in \( \ell \)T

\[
\lambda y. \\
\text{let } \ell = \lfloor \log(n + 1) \rfloor \\
\text{let } P = \text{pow}_0 \ell \text{ in} \\
\text{let } R = \text{map } (\text{flip}) \ast^\ell \text{ in} \\
\text{let } Z = \text{map } (\lambda x. \text{map } (\text{flip}) \ast^n) \ast^\ell \text{ in} \\
\text{let } g_r = \lambda X. \text{fold } (0, \oplus) \text{ (map}_2 (\otimes) X R) \text{ in} \\
\text{let } \otimes = \lambda x. \text{map } (\otimes x) \text{ in} \\
\text{let } \oplus = \text{map}_2 (\oplus) \text{ in} \\
\text{let } g_z = \lambda X. \text{fold } (0^n, \oplus) \text{ (map}_2 (\otimes) X Z) \text{ in} \\
\text{let } R^P = \text{map } (g_r) P \text{ in} \\
\text{let } Z^P = \text{map } (g_z) P \text{ in} \\
\text{let } G = \lambda i. \text{majority } (\text{map}_2 (\lambda r z. \oplus r (A (\text{app } y (\text{zerobut } i z)))) R^P Z^P) \text{ in} \\
\text{map } (G')(1, \ldots, n)
\]

argument of the inverter
defines \( \ell \)
enumerates all non-empty subsets of \( \{1, \ldots, \ell\} \)
samples uniformly at random \( (r_1, \ldots, r_\ell) \)
samples uniformly at random \( (z_1, \ldots, z_\ell) \)
computes the list \( (r^X)_{X \in P} \)
computes the list \( (z^X)_{X \in P} \)
The inversion algorithm $A'$. We now provide a full description of an algorithm $A'$ 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
   $$x^I_i := \rho^I \oplus A(y, r^I \oplus e^i).$$
   
   (b) Set $x_i := \text{majority}_I\{x^I_i\}$ (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 $A'$. We now provide a full description of an algorithm $A'$ 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 $r^I := \bigoplus_{i \in I} s^i$ and compute $\rho^I := \bigoplus_{i \in I} \sigma^i$.

4. For $i = 1, \ldots, n$:
   
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   
   $$x_i^I := \rho^I \oplus A(y, r^I \oplus e^i).$$
   
   (b) Set $x_i := \text{majority}_I\{x_i^I\}$ (i.e., take the bit that appeared a majority of the times in the previous step).

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

\[ L^a \text{(B)} \]
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 $r^I := \bigoplus_{i \in I} s^i$ and compute $\rho^I := \bigoplus_{i \in I} \sigma^i$.

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 $x_i := \text{majority}_I\{x^I_i\}$ (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 $A'$. We now provide a full description of an algorithm $A'$ 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 $r^I := \bigoplus_{i \in I} s^i$ and compute $\rho^I := \bigoplus_{i \in I} \sigma^i$.

4. For $i = 1, \ldots, n$:
   
   (a) For every non-empty subset $I \subseteq \{1, \ldots, \ell\}$, set
   
   $$x_i^I := \rho^I \oplus A(y, r^I \oplus e^i).$$
   
   (b) Set $x_i := \text{majority}_1\{x_i^I\}$ (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]
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 $r^I := \bigoplus_{i \in I} s^i$ and compute $\rho^I := \bigoplus_{i \in I} \sigma^i$.

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 $x_i := \text{majority}_I\{x^I_i\}$ (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 $A'$ the weight

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

where $f_A$ is a function bounding the complexity of adversary $A$.

* the bound given by the soundness thm for the complexity of $A'$ 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.