firstname.lastname@lip6.fr
Equipe APR, LIP6 (UMR 7606), Université Pierre et Marie Curie Jérémie SALVUCCI & Emmanuel CHAILLOUX

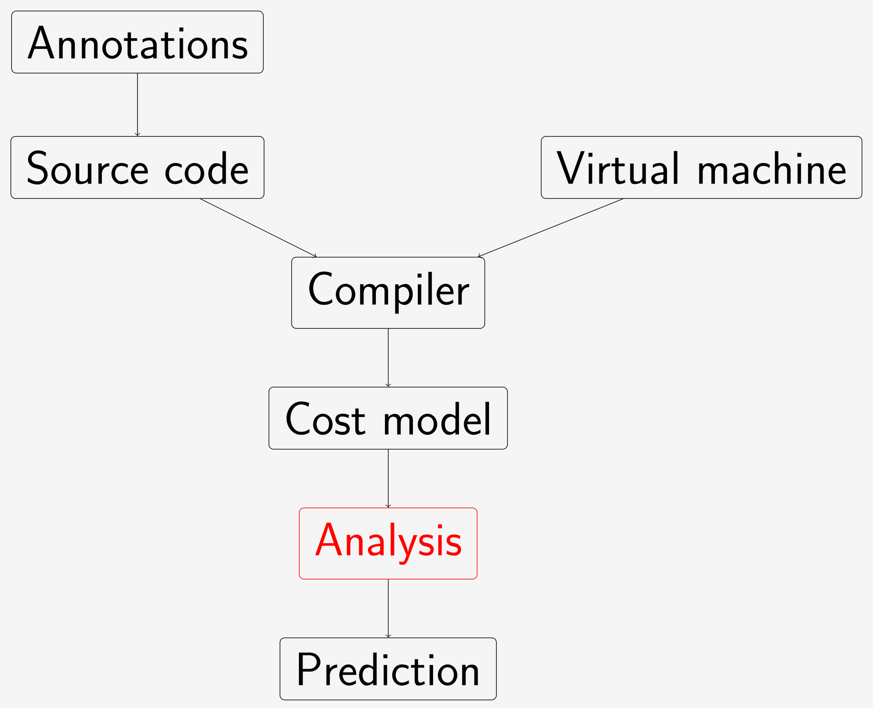
MEMORY CONSUMPTION ANALYSIS

FOR APPLICATIVE LANGUAGES

Objectives

- Guarantee a safe upper bound of used memory.
- Consider how **garbage collection** affects this upper bound.
- Get a more precise upper bound than the existing methods.
- Consider the minimum amount of allocated memory to accomplish this.

Analysis of a Mini-ML



To create a cost model, we use

- Source code annotation.
- Data representation (data from the virtual machine).
- Compiler code transformation.

We target a Mini-ML composed of booleans, integers, data constructors and closures.

The whole process relies on two measures

- An **upper** bound on the amount of heap-allocated memory.
- A **lower** bound on the minimum of recycled memory.

The result of this analysis is the subtraction of these measures.

Targetting a virtual machine

The analysis is performed directly on **bytecode**.

- Explicit stack.
- Garbage Collector root set represented by this stack.
- Tail calls are considered.

We introduce the analysis on program source code (without considering compilation optimizations).

How do we proceed ?

- We use Abstract Interpretation techniques (boolean domain, integer domain (interval)).
- We apply the analysis to monotonic functions.
- We evaluate on both bounds of the interval (depending on function growth).

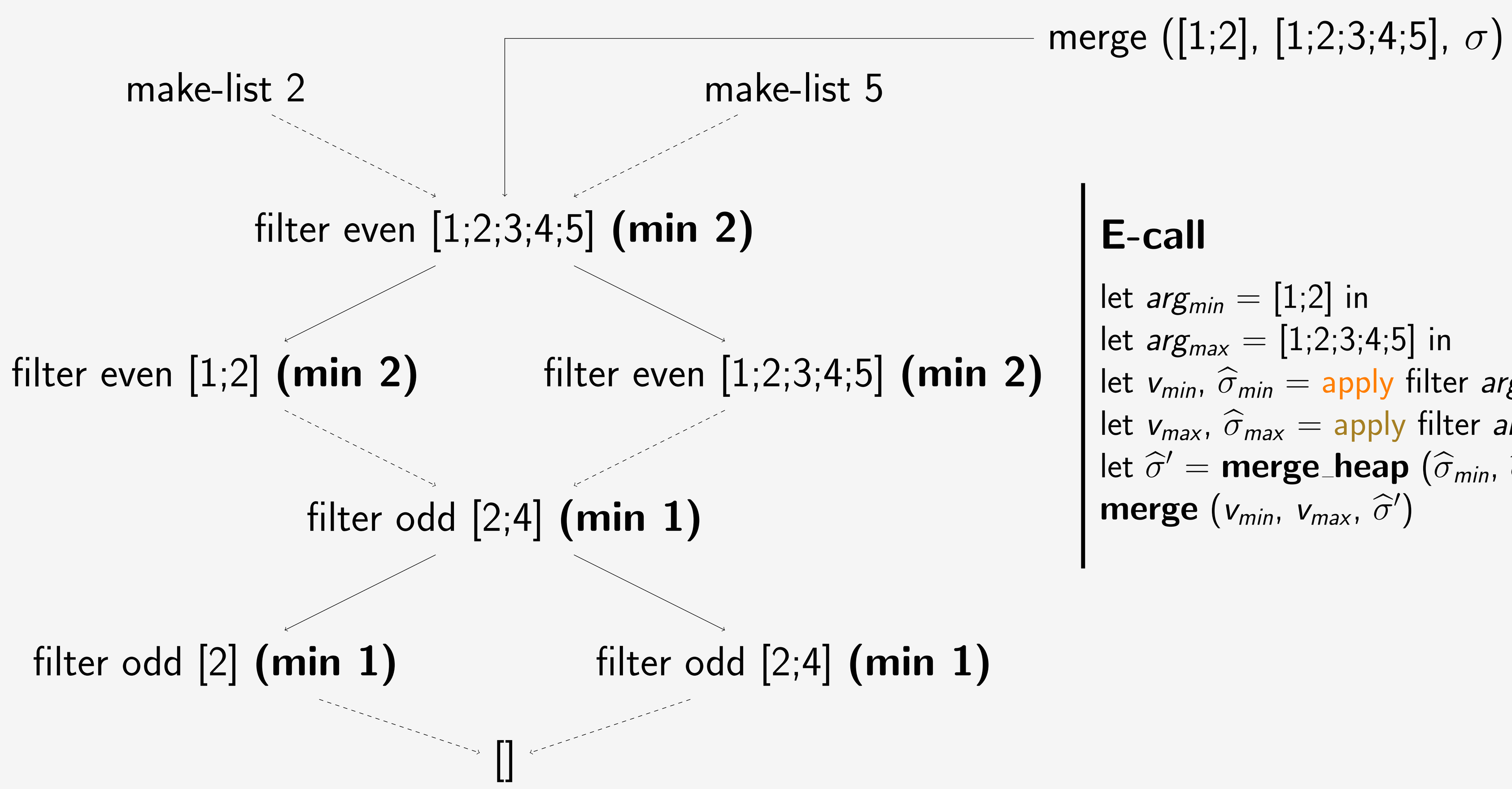
Analysis run on an example

Analysis of the following piece of code

```
let l = if Cond then make-list 2 else make-list 5 (* rule E-if *) in
  filter odd (filter even l) (* rule E-call *)
```

We suppose several things

- make-list n returns a list from 1 to n
- **min n** represents the relevant information to build back the minimal allocated structure.



E-call

let arg_{min} = [1;2] in
let arg_{max} = [1;2;3;4;5] in
let $v_{min}, \widehat{\sigma}_{min}$ = apply filter arg_{min} $\widehat{\sigma}$ in
let $v_{max}, \widehat{\sigma}_{max}$ = apply filter arg_{max} $\widehat{\sigma}$ in
let $\widehat{\sigma}'$ = **merge_heap** $(\widehat{\sigma}_{min}, \widehat{\sigma}_{max})$ in
merge $(v_{min}, v_{max}, \widehat{\sigma}')$

Rules

merge_call (l, v, $\widehat{\sigma}$) :=
  let clss = get (l,$\widehat{\sigma}$) in
  fold (fun acc_v cls →
    let v, $\widehat{\sigma}'$ = fold (fun acc_v arg →
      let arg_{min} = min arg in
      let arg_{max} = max arg in
      let $v_{min}, \widehat{\sigma}_{min}$ = apply cls arg_{min} $\widehat{\sigma}$ in
      let $v_{max}, \widehat{\sigma}_{max}$ = apply cls arg_{max} $\widehat{\sigma}$ in
      let $\widehat{\sigma}'$ = **merge_heap** $(\widehat{\sigma}_{min}, \widehat{\sigma}_{max})$ in
      let v, $\widehat{\sigma}'$ = **merge** $(v_{min}, v_{max}, \widehat{\sigma}')$ in
      merge (acc_v, v, $\widehat{\sigma}'$))
    ϵ
    get (v, $\widehat{\sigma}$)
  in **merge** (acc_v, v, $\widehat{\sigma}'$))
  ϵ
  clss

$$\frac{l \notin dom(\widehat{\sigma}) \qquad \widehat{\sigma}' = \widehat{\sigma}[l \rightarrow (fun\ x \Rightarrow e)[\widehat{\rho}]]}{\widehat{\rho}, \widehat{\sigma} \vdash fun\ x \rightarrow e \rightsquigarrow l} \quad \textbf{(E-fun)}$$

$$\frac{\widehat{\rho}, \widehat{\sigma} \vdash e_0 \rightsquigarrow l, \widehat{\sigma}' \qquad \widehat{\rho}, \widehat{\sigma}' \vdash e_1 \rightsquigarrow v_s, \widehat{\sigma}'' \quad v, \widehat{\sigma}'' = merge\_call(l, v_s, \widehat{\sigma}''')}{\widehat{\rho}, \widehat{\sigma} \vdash e_0\ e_1 \rightsquigarrow v, \widehat{\sigma}'''} \quad \textbf{(E-call)}$$

$$\frac{\widehat{\rho}, \widehat{\sigma} \vdash e_{cd} \rightsquigarrow \top_b, \widehat{\sigma}' \quad \widehat{\rho}, \widehat{\sigma}' \vdash e_{cs} \rightsquigarrow v_{cs}, \widehat{\sigma}_{cs} \qquad \widehat{\rho}, \widehat{\sigma}' \vdash e_{alt} \rightsquigarrow v_{alt}, \widehat{\sigma}_{alt}}{\widehat{\rho}, \widehat{\sigma} \vdash if\ e_{cd}\ then\ e_{cs}\ else\ e_{alt} \rightsquigarrow merge(v_{cs}, v_{alt}, merge\_heap(\widehat{\sigma}_{cs}, \widehat{\sigma}_{alt}))} \quad \textbf{(E-if)}$$

Concrete domains	Abstract domains
$z \in Num \equiv \mathbb{Z}$	$\widehat{z} \in \widehat{Num} \equiv \{\bot_n\} \cup \{[a, b] \mid a \in \bar{\mathbb{Z}},\ b \in \bar{\mathbb{Z}}\}$
$b \in Bool \equiv \{False, True\}$	$\widehat{b} \in \widehat{Bool} \equiv \{\bot_b, False, True, \top_b\}$
$l \in Loc \equiv \mathbb{N}$	$\widehat{l} \in \widehat{Loc} \equiv \mathbb{N} \cup \{\top_l\}$
$v_s \in V_S \equiv Num + Bool + Loc$	$\widehat{v_s} \in \widehat{V_S} \equiv \widehat{Num} + \widehat{Bool} + \widehat{Loc}$
$\rho : Name \rightarrow V_S$	$\widehat{\rho} : Name \rightarrow \widehat{V_S}$
$dc \in DC \equiv K\ \vec{v_s}\ with\ v_s \in V_S$	$\widehat{dc} \in \widehat{DC} \equiv \mathcal{P}(DC)$
$cls \in Cls \equiv Code \times \vec{v_s}\ with\ v_s \in V_S$	$\widehat{cls} \in \widehat{Cls} \equiv \mathcal{P}(Cls)$
$v_h \in V_H \equiv DC + Cls$	$\widehat{v_h} \in \widehat{V_H} \equiv \widehat{DC} + \widehat{Cls}$
$\sigma : Loc \rightarrow V_H$	$\widehat{\sigma} : \widehat{Loc} \rightarrow \widehat{V_H}$

Domain abbreviations

- $Loc, \widehat{Loc}$: Addresses
- $DC, \widehat{DC}$: Data Constructors
- $Cls, \widehat{Cls}$: Closures
- $V_S, \widehat{V_S}$: Immediate values
- $V_H, \widehat{V_H}$: Allocated values
- $\rho, \widehat{\rho}$: Environment
- $\sigma, \widehat{\sigma}$: Heap

Restrictions

- Analysed programs should be terminating.
- Monotonic functions.
- No side-effects.
- Only linear data structures are allowed.

Current issues

- Notion of **shapes** for nonlinear data structures.
- Adapt the same method to nonlinear data structures is difficult.
- Termination of the analysis.
- Liveness analysis required.
- References are part of a future work.

Related Works

The embounded project - Pedro Vasconcelos' work on sized types (target : Mini-ML)

- $Nil :: \forall i < \forall a.List^i a, i = 0 >$
- $Cons :: \forall ij < \forall a.(a, List^i a) \rightarrow List^j a, j = i + 1, i >= 0 >$
- $append :: (List^i a, List^j a) \rightarrow List^k a, k = i + j$

$$\frac{\dfrac{\Gamma_1 \vdash l_2 : List^j a \quad \Gamma_3 \vdash append(xs, l_2) : List^{k'} a, k' = i' + j' \qquad \Gamma_4 \vdash Cons(x, r) : List^k a, k = 1 + k'}{\Gamma_2 \vdash let\ r = append(xs, l_2)\ in\ Cons(x, r) : List^m a}}{\Gamma_0 \vdash fun\ x\ with\ Nil \rightarrow l_2 | Cons\ (x, xs) \rightarrow let\ r = append(xs, l_2)\ in\ Cons(x, r) : List^n a}$$

RAML - Martin Hofmann's work on automatic amortized analysis (target : Mini-ML)

Hypothesis : $append :: (List(A, b_1), List(A, b_2), c) \rightarrow (List(A, b_3), d)$

$$\frac{\dfrac{\Gamma_1, n_1 \vdash l_2 : List(A, a_2), m_1 \quad \Gamma_2, n_2 \vdash append\ xs\ l_2 : List(A, a_4), m_2 \qquad \Gamma_3, n_3 \vdash Cons(x, r), n_3 : List(A, a_5), m_3}{\Gamma_4, n_4 \vdash let\ r = append\ xs\ l_2\ in\ Cons(x, r) : List(A, a_6), m_4}}{\Gamma_0, n_0 \vdash fun\ l_1\ with\ Nil \rightarrow l_2 | Cons(x, xs) \rightarrow let\ r = append\ xs\ l_2\ in\ Cons(x, r) : List(A, a_7), m_0}$$

The analysis consists in finding the coefficient of the potential function which is a linear combination of the functions inputs.

References

- P. Hudak.
  A Semantic Model of Reference Counting and its Abstraction.
  In *Abstract Interpretation of Declarative Languages*, pages 45–62. Ellis Horwood, 1987.
- Steffen Jost.
  Automated Amortised Analysis.
  PhD thesis, Faculty of Mathematics, Computer Science and Statistics, LMU Munich, Germany, September 2010.
- Pedro Vasconcelos.
  Space Cost Analysis Using Sized Types.
  PhD thesis, School of Computer Science, University of St Andrews, Scotland, 2008.