A Machine-Independent, Log-Sensitive Space-Cost Measure for the Weak λ-Calculus

Thibaut Balabonski
Laboratoire Méthodes Formelles, Université Paris-Saclay

LICS'26

Research question inspired by a teaching project

Teaching project

A modern course on computability/complexity,
building on student's intuition on programming

Requirements

- Computation model: $\left\{ \begin{array}{l} \text{a simple functional language,} \\ \text{running with unbounded memory} \end{array} \right.$
- Rigorous definitions and proofs

Research question

How to characterize complexity in an implementation-independant way?

Computability is universal

Question: What can be computed by an algorithm?

Many models, giving the same answer

- Turing machines
- λ-calculus
- RAM machines
- ...
- Your favorite programming language*

Takeaway: computability is a very robust notion

What about complexity?

What can be $\begin{cases} \text{computed/checked within polynomial time ? (P/NP)} \\ \text{computed/checked within polynomial space ? (PSPACE)} \\ \text{computed/checked within logarithmic space ? (L/NL)} \end{cases}$

"Category 1" machines (Sloat & van Emde Boas) :

reasonable machines can simulate each other
within a polynomially bounded overhead in time
and a constant factor overhead in space

Takeaway: complexity is robust, with the right definitions

Reasonable cost models

Reasonable cost models: a matter of definition

	Time	Space	
Turing	nb. transitions	tape length	OK
RAM (unit.)	nb. transitions	nb. cells	KO
RAM (log.)	nb. transitions	log. values	OK
...	...	...	...
...	...	...	...
...	...	...	...

Reasonable cost models: a matter of definition

	Time	Space	
Turing	nb. transitions	tape length	OK
RAM (unit.)	nb. transitions	nb. cells	KO
RAM (log.)	nb. transitions	log. values	OK
...	...	...	...
Weak λ-calculus	nb. β-reductions?	size of terms?	??
My language	nb. steps?	size of terms?	??

β-reduction $(\lambda x.t)\ u \ \longrightarrow\ t^{\{x \leftarrow u\}}$: complex operation

- explore subterm t
- duplicate subterm u
- possible exponential growth

λ-calculus: known complexity results

Time

1995–2008: nb. β-steps in weak call-by-value

2014: nb. β-steps in normal order (strong)

Space

1996: “max ink” cost: maximal size of an intermediate term

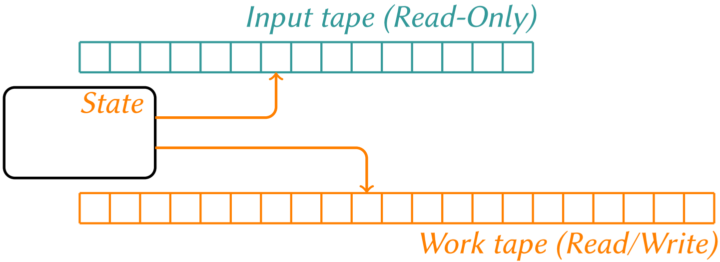
Limitation: λ-calculus mixes program, input data, and work space

The “ink” model cannot express sublinear space

Program, input, workspace (Turing)

Transition table

$q_0, 0, 0$	$\longrightarrow$	$\longrightarrow, 1, \downarrow, q_0$
$q_0, 0, 1$	$\longrightarrow$	$\downarrow, 0, \longleftarrow, q_2$
$q_0, 1, 0$	$\longrightarrow$	$\downarrow, 1, \downarrow, q_1$
$q_0, 1, 1$	$\longrightarrow$	$\longleftarrow, 1, \longrightarrow, q_1$
$q_1, 0, 0$	$\longrightarrow$	$\longrightarrow, 0, \downarrow, q_0$
$q_1, 0, 1$	$\longrightarrow$	$\longrightarrow, 0, \downarrow, q_3$
	$\dots$	



Program, input, workspace (λ-calculus)

$$
\begin{aligned}
& (\lambda p.p(\lambda xy.x)(\lambda xy.y)(p(\lambda xy.y)))\ (\lambda x.x(\lambda yz.y)(\lambda yz.z)) \\
\rightarrow\ & (\lambda x.x(\lambda yz.y)(\lambda yz.z))(\lambda xy.x)(\lambda xy.y)((\lambda x.x(\lambda yz.y)(\lambda yz.z))(\lambda xy.y)) \\
\rightarrow\ & (\lambda xy.x)(\lambda yz.y)(\lambda yz.z)(\lambda xy.y)((\lambda x.x(\lambda yz.y)(\lambda yz.z))(\lambda xy.y))
\end{aligned}
$$

Program, input, workspace (λ-calculus)

$$
\begin{aligned}
& (\lambda p.((p(\lambda xy.x))(\lambda xy.y))(p(\lambda xy.y)))\ (\lambda x.(x(\lambda yz.y))(\lambda yz.z)) \\
\to\ & (((\lambda x.(x(\lambda yz.y))(\lambda yz.z))(\lambda xy.x))(\lambda xy.y))((\lambda x.(x(\lambda yz.y))(\lambda yz.z))(\lambda xy.y)) \\
\to\ & ((((\lambda xy.x)(\lambda yz.y))(\lambda yz.z))(\lambda xy.y))((\lambda x.(x(\lambda yz.y))(\lambda yz.z))(\lambda xy.y))
\end{aligned}
$$

Space complexity in the λ-calculus

λ-calculus: isolating workspace?

Abstract machine

B. Accattoli, U. Dal Lago, G. Vanoni, 2022

$$\langle\ \text{term} \mid \text{environment} \mid \text{stack}\ \rangle$$

Computation rules

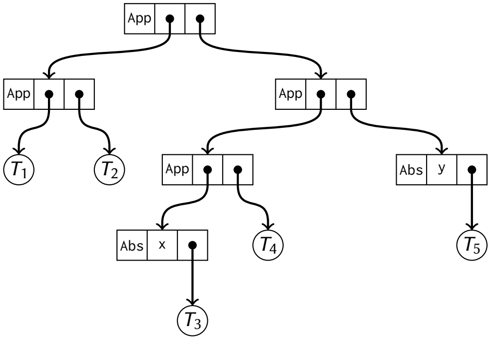
$$
\begin{array}{lcll}
\langle\ t\ x \mid e \mid \pi\ \rangle & \to & \langle\ t \mid e|_t \mid e(x); \pi\ \rangle & \\
\langle\ t\ u \mid e \mid \pi\ \rangle & \to & \langle\ t \mid e|_t \mid (u, e|_u); \pi\ \rangle & u \notin \text{Vars} \\
\langle\ \lambda x.t \mid e \mid c; \pi\ \rangle & \to & \langle\ t \mid e \mid \pi\ \rangle & x \notin \text{fv}(t) \\
\langle\ \lambda x.t \mid e \mid c; \pi\ \rangle & \to & \langle\ t \mid [x{\leftarrow}c]; e \mid \pi\ \rangle & x \in \text{fv}(t) \\
\langle\ x \mid e \mid \pi\ \rangle & \to & \langle\ u \mid e' \mid \pi\ \rangle & e(x) = (u, e')
\end{array}
$$

Result: first cost model for the λ-calculus able to capture the L class

Limitation: low-level, bound to an implementation

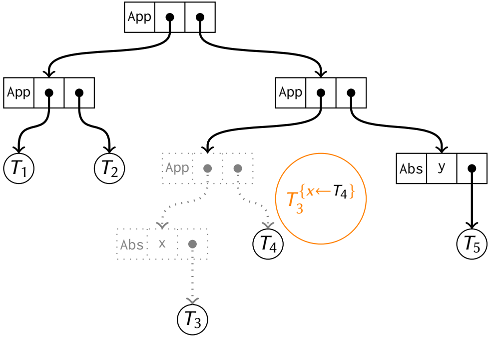
Detour in the world of functional programming

```
type t = Var of string | App of t * t | Abs of string * t
```



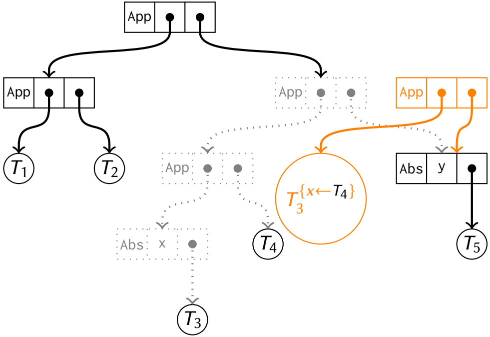
Detour in the world of functional programming

```
type t = Var of string | App of t * t | Abs of string * t
```



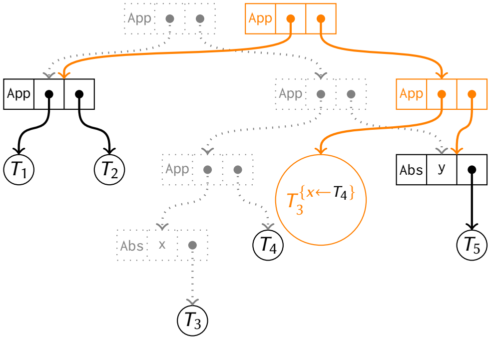
Detour in the world of functional programming

```
type t = Var of string | App of t * t | Abs of string * t
```



Detour in the world of functional programming

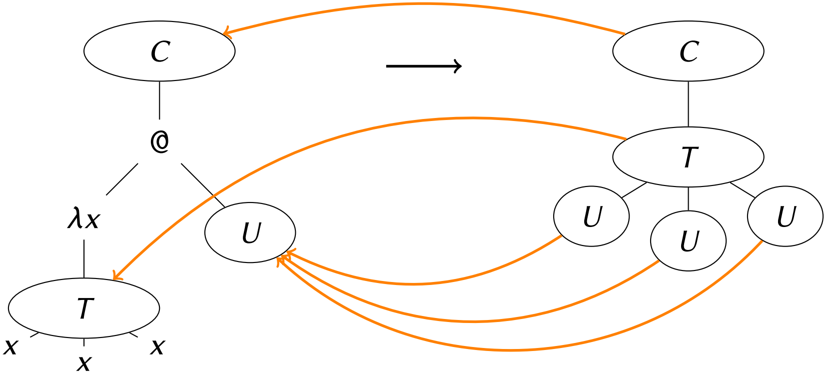
```
type t = Var of string | App of t * t | Abs of string * t
```




```
let rec step = function
  | App(Abs(x1, t1), v2) when is_val v2 -> subst t1 x1 v2
  | App(t1         , v2) when is_val v2 -> App(step t1,      v2)
  | App(t1         , t2) (* default *)  -> App(     t1, step t2)
```

Descendance relation

Descendance after a reduction step $t \longrightarrow t'$:
each position of t' comes from a position of t



Full descendant: equal subterm, with pointwise descending positions

History-aware cost model

Space cost after a reduction sequence $t_0 \to t_1 \to \ldots \to t_n$:

- 1 pointer for each full descendant of a subterm of t_0
- ink cost otherwise

Cost of a pointer: $\log(|t_0|)$

Result:

Reasonable space cost model for any logarithmic or higher complexity

A decorated λ-calculus

Tracking subterms

We inject marks in the λ-terms

- which do not interfere with reduction
- which can be observed in the reduced terms

Specifics:

- mark any (closed) initial subterm
- no individual identification
- marks disappear with any modification

Marked λ-calculus

Syntax

$$\begin{array}{rrcl} \textit{Terms} & t & ::= & \theta \mid \{\theta\} \\ \textit{Preterms} & \theta & ::= & x \mid t\ t \mid \lambda x.t \end{array}$$

Interpretation

$\{\theta\}$ is a full descendant
of an initial (closed) subterm

Reduction rules (weak)

$$\frac{}{(\lambda x.t)\ s \rightarrow t^{\{x \leftarrow s\}}} \qquad \frac{}{\{\lambda x.t\}\ s \rightarrow t^{\{x \leftarrow s\}}}$$

$$\frac{t \rightarrow t'}{t\ u \rightarrow t'\ u} \qquad \frac{u \rightarrow u'}{t\ u \rightarrow t\ u'} \qquad \frac{\theta \rightarrow t'}{\{\theta\} \rightarrow t'}$$

A model of space cost

Parameters *(shadows of an implicit initial term)*

- c_p: cost of a pointer
- c_v: cost of a variable

Space cost

$$
\begin{array}{rcl}
\|\{\theta\}\|_{c_v,c_p} & = & c_p \\
\|x\|_{c_v,c_p} & = & c_v \\
\|t\ u\|_{c_v,c_p} & = & 1 + \|t\|_{c_v,c_p} + \|u\|_{c_v,c_p} \\
\|\lambda x.t\|_{c_v,c_p} & = & 1 + \|t\|_{c_v,c_p}
\end{array}
$$

Result

Equivalent to history-aware model, with $c_p = c_v = \log|t_0|$

Implementation of the marked λ-calculus

Programs and configurations

Compilation

$$
\begin{aligned}
\llparenthesis x \rrparenthesis_\rho &= \mathsf{VAR}\ k && \textit{de Bruijn: } \rho[k] = x \\
\llparenthesis t\ u \rrparenthesis_\rho &= \mathsf{APP}\ ;\ \llparenthesis t \rrparenthesis_\rho\ ;\ \llparenthesis u \rrparenthesis_\rho && \textit{prefix application} \\
\llparenthesis \lambda x.t \rrparenthesis_\rho &= \mathsf{LAM}\ ;\ \llparenthesis t \rrparenthesis_{x;\rho}\ ;\ \mathsf{MAL} && \textit{explicit range}
\end{aligned}
$$

Configuration

$$\langle \ldots ; \mathsf{LAM} ; \mathsf{APP} ; \mathsf{VAR}\ 0 ; \mathsf{LAM} ; \mathsf{APP} ; \ldots \mid \mathsf{APP} ; \mathsf{PTR}\ 27 ; \mathsf{LAM} ; \mathsf{APP} ; \mathsf{PTR}\ 24 ; \mathsf{VAR}\ 0 ; \mathsf{MAL} \rangle$$

input tape: *pure program, RO*
init: $\llparenthesis t_0 \rrparenthesis_\varepsilon$

work tape: *program with pointers, RW*
init: PTR 0

Execution

Computation rules

$$\textit{β-reduction} \quad \langle\, I \mid C_1^* \,;\, \mathsf{APP} \,;\, \mathsf{LAM} \,;\, U \,;\, \mathsf{MAL} \,;\, S \,;\, C_2^* \,\rangle \;\longrightarrow\; \langle\, I \mid C_1^* \,;\, U^{\{0 \leftarrow S\}} \,;\, C_2^* \,\rangle$$

$$\textit{Dereferencing} \quad \langle\, I \mid C_1^* \,;\, \mathsf{PTR}\ a \,;\, C_2^* \,\rangle \;\longrightarrow\; \langle\, I \mid C_1^* \,;\, I[\![a..b]\!] \,;\, C_2^* \,\rangle$$

Auxiliary operations

- ***Parsing*** for identifying U, S, and b
- ***Substitution*** of programs $U^{\{0 \leftarrow S\}}$
- ***Copy*** $I[\![a..b]\!]$ with pointer introduction for (closed) subterms

each with a bounded space cost

Implementation theorem

If $t \longrightarrow^* t'$ *with space cost* $c \geqslant \log|t|$, *then*

$$\langle\ (\!|t|\!)_\rho \mid \mathsf{PTR}\ 0\ \rangle \longrightarrow^* \langle\ (\!|t|\!)_\rho \mid T'\ \rangle$$

with $T' \gg_I t'$ *and with space cost* $c' \leqslant 6c + 2$

Takeaway

Weak λ-calculus reduction can be implemented by a Turing machine with a constant factor space overhead

Simulating Turing machines

Configurations

B. Accattoli, U. Dal Lago, G. Vanoni, 2023

work tape

left part, head, right part

$$\langle I, n \mid W_l, b, W_r \mid q \rangle$$

contents, position   state

input tape

Representation in the λ-calculus: tuple

$$\langle\!\langle C \rangle\!\rangle \quad = \quad \lambda x.x\ I\ n\ W_l\ b\ W_r\ q$$

Execution

$$\text{Compute} \quad = \quad \text{Fix}\,(\lambda f.\lambda c.c\,(\lambda i n w_l b w_r q.\cdots)\,f) \qquad \textit{(iterate until halt)}$$

Transitions

input tape

$$\underset{\text{state}}{\delta(q}, \overset{\text{read}}{a}, \underset{\text{read}}{b}) \quad \longrightarrow \quad \langle\ \overset{\text{move}}{m_i} \mid \underset{\text{write}}{b'}, \underset{\text{move}}{m_w} \mid \overset{\text{new state}}{q'}\ \rangle$$

work tape

Representation of a character: selector

$$[q_i]_\Sigma \quad = \quad \lambda x_1 \ldots x_{|\Sigma|}.x_i$$

Execution: 3 cascading selections

$$\begin{aligned}
&q\ \{\mathrm{Q}_{q_1}\}\ \{\mathrm{Q}_{q_2}\}\ \cdots\ \{\mathrm{Q}_{q_m}\} \\
\mathrm{Q}_q &\quad = \quad \text{Get } n\ i\ \{\mathrm{A}_{q,0}\}\ \{\mathrm{A}_{q,1}\} \cdots \qquad \textit{(unless halting state)} \\
\mathrm{A}_{q,a} &\quad = \quad b\ \{\mathrm{B}_{q,a,0}\}\ \{\mathrm{B}_{q,a,1}\} \cdots \\
\mathrm{B}_{q,a,b} &\quad = \quad \textit{transition } \delta(q, a, b)
\end{aligned}$$

Scott representation of sequences

Algebraic data type = selector + tuple

$$
\begin{aligned}
[\![\varepsilon]\!]_\Sigma &= \lambda x_1 \ldots x_{|\Sigma|} x_\varepsilon . x_\varepsilon && \textit{simple selector} \\
[\![a_i; s]\!]_\Sigma &= \lambda x_1 \ldots x_{|\Sigma|} x_\varepsilon . x_i \; s && \textit{selector with contents} \\
[\![n]\!] &= [\![0; 1; 1; \ldots ; 0; 1]\!]_{\mathbb{B}} && \textit{binary representation}
\end{aligned}
$$

Pattern matching = selection of a function

$$
\begin{aligned}
\mathrm{Succ} = \; & \mathrm{Fix} \; (\lambda succ . \lambda n . \\
& \quad n \\
& \quad (\lambda s. \; \mathrm{Cons}_1 \; s) \\
& \quad (\lambda s. \; \mathrm{Cons}_0 \; (succ \; s)) \\
& \quad \mathrm{Cons}_1 \; \mathrm{Nil} \\
&)
\end{aligned}
$$

```
let rec succ n =
  match n with
  | 0 :: s -> 1 :: s
  | 1 :: s -> 0 :: succ s
  | []     -> [1]
```

Scott representation of sequences

Algebraic data type = selector + tuple

$$\begin{aligned}
[\![\varepsilon]\!]_\Sigma &= \lambda x_1 \ldots x_{|\Sigma|} x_\varepsilon . x_\varepsilon && \textit{simple selector} \\
[\![a_i; s]\!]_\Sigma &= \lambda x_1 \ldots x_{|\Sigma|} x_\varepsilon . x_i \; s && \textit{selector with contents}
\end{aligned}$$

$$[\![n]\!] = [\![0; 1; 1; \ldots; 0; 1]\!]_{\mathbb{B}} \qquad \textit{binary representation}$$

Pattern matching = selection of a function

$$\begin{aligned}
\mathrm{Succ} = \{\mathrm{Fix}\}\ \{\lambda succ.\lambda n.\ & \\
& n \\
& \{\lambda s.\{\mathrm{Cons}_1\}\ s\} \\
& (\lambda s.\{\mathrm{Cons}_0\}\ (succ\ s)) \\
& \{\{\mathrm{Cons}_1\}\ \{\mathrm{Nil}\}\} \\
\}&
\end{aligned}$$

```
let rec succ n =
  match n with
  | 0 :: s -> 1 :: s
  | 1 :: s -> 0 :: succ s
  | []     -> [1]
```

Low-space arithmetic

Tail recursive functions

```
let rec rev_app s1 s2 = match s1 with
  | []        -> s2
  | x1 :: s1' -> rev_app s1' (x1 :: s2)

let rec succ s n = match n with
  | []     -> rev_app s [1]
  | 0 :: m -> rev_app s (1 :: m)
  | 1 :: m -> succ (0 :: s) m

let rec pred s n = match n with
  | []      -> assert false
  | 0 :: m  -> pred (1 :: s) m
  | 1 :: [] -> rev_app s []
  | 1 :: m  -> rev_app s (0 :: m)
```

(and apply translation to λ-terms)

Simulation theorem

If $\langle I, 0 \mid \varepsilon, \diamond, \varepsilon \mid q_0 \rangle \longmapsto^* \langle I, n \mid W_l, b, W_r \mid q_f \rangle$ *with* $q_f \in \{q_a, q_r\}$ *and cost* c, *then*

$$\text{Compute } (\lambda x.x\, [\![I]\!]_{\mathbb{I}}\, [\![0]\!]\, [\![\varepsilon]\!]_{\mathbb{W}}\, [\diamond]_{\mathbb{W}}\, [\![\varepsilon]\!]_{\mathbb{W}}\, [q_0]_{\mathbb{Q}}) \rightarrow_{\beta}^{*} R$$

with $R \in \{\text{Accept}, \text{Reject}\}$ *and space cost* $O(c + \log |I|)$

Takeaway

Computation of a Turing machine can be simulated by the weak λ-calculus with a constant factor space overhead

Conclusion

Lambda-calculus and space complexity

Contribution

A reasonable space cost model for the weak λ-calculus,
equivalent to Turing cost for any logarithmic or higher space complexity

Consequence

Faithful characterization of the complexity classes L, NL, PSPACE, …

Main idea

Measure term size, while also taking origin into account

Lambda-calculus and space complexity

Contribution

A reasonable space cost model for the weak λ-calculus,
equivalent to Turing cost for any logarithmic or higher space complexity

Consequence

Faithful characterization of the complexity classes L, NL, PSPACE, ...

Main idea

Measure term size, while also taking origin into account

Next questions

- Reasonable model for space *and time*? *Presented at ICFP next month*
- *Strong* reduction? *Open problem*