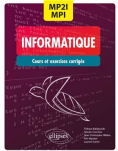
Lambda-calculus and space complexity

Thibaut Balabonski
LMF @ Université Paris-Saclay

GT Scalp 2025

Research question inspired by a teaching project



CPGE MPI, chapter on decidability and complexity classes

- universal machine, halting problem, P, NP...
- no Turing machines
- model of computation: computer with unbounded memory running a C or OCaml program

Project: *modernized presentation of computability theory, building on student's intuition on programming*

Models of computation and complexity

Computability is universal

What can be computed by an algorithm?

Many models, giving the same answer

- Turing machines
- λ-calculus
- RAM machines
- ...

Takeaway: computability is a very robust notion

What about complexity?

What can be
- computed/checked within polynomial time ? (P/NP)
- computed/checked within polynomial space ? (PSPACE)
- computed/checked within logarithmic space ? (L/NL)

"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, under some conditions

Reasonable cost models

	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

	Time	Space	
Turing	nb. transitions	tape length	OK
RAM (unit.)	nb. transitions	nb. cells	KO
RAM (log.)	nb. transitions	log. values	OK
...	...	...	...
λ-calculus	nb. β-reductions?	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
- several possible strategies

λ-calculus: complexity models

Time

1996: "total ink" cost: time required for writing down the reduction

2008: nb. β + size diffs in (weak) call by value

2014: nb. β in normal order (strong)

Space

1996: "max ink" cost: maximal size of an intermediate term

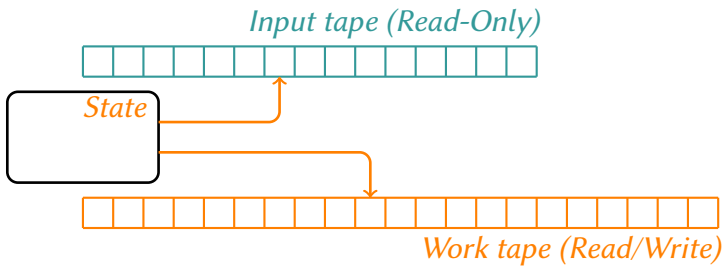
Limitation: λ-calculus mixes program, input data, and work space

The "ink" model cannot express sublinear space

Program, input, workspace (Turing)

Transition table

$q_0, 0, 0$	$\rightarrow$	$\rightarrow, 1, \downarrow, q_0$
$q_0, 0, 1$	$\rightarrow$	$\downarrow, 0, \leftarrow, q_2$
$q_0, 1, 0$	$\rightarrow$	$\downarrow, 1, \downarrow, q_1$
$q_0, 1, 1$	$\rightarrow$	$\leftarrow, 1, \rightarrow, q_1$
$q_1, 0, 0$	$\rightarrow$	$\rightarrow, 0, \downarrow, q_0$
$q_1, 0, 1$	$\rightarrow$	$\rightarrow, 0, \downarrow, q_3$
	...	



Program, input, workspace (λ-calculus)

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

Program, input, workspace (λ-calculus)

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

Program, input, workspace (λ-calculus)

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

Space complexity in the λ-calculus

λ-calculus: isolating workspace?

Abstract machine

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

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

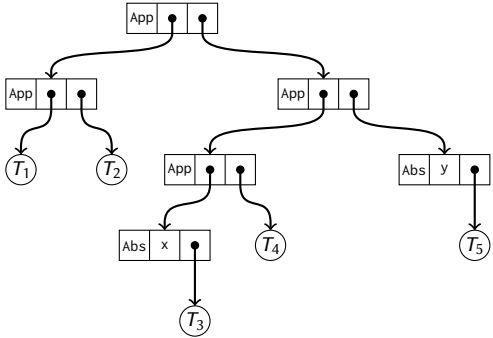
Computation rules

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

Result: first cost model for the λ-calculus able to capture the L class
Limitation: low-level, bound to an implementation

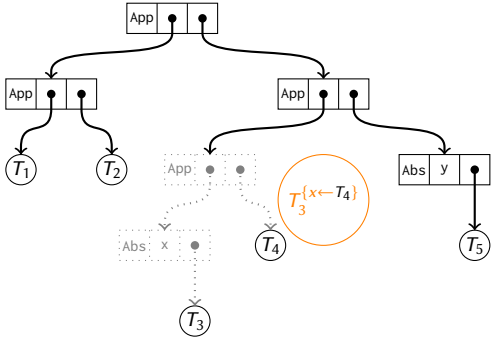
Detour in the world of functional programming

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



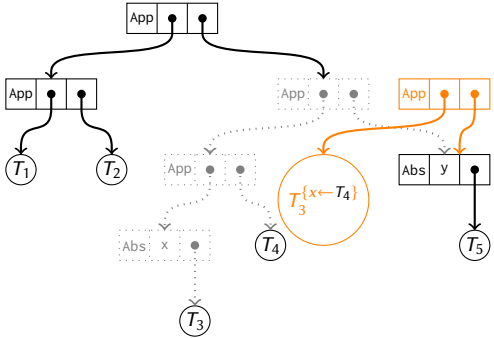
Detour in the world of functional programming

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



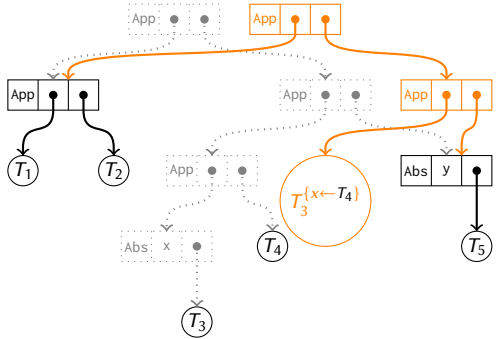
Detour in the world of functional programming

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



Detour in the world of functional programming

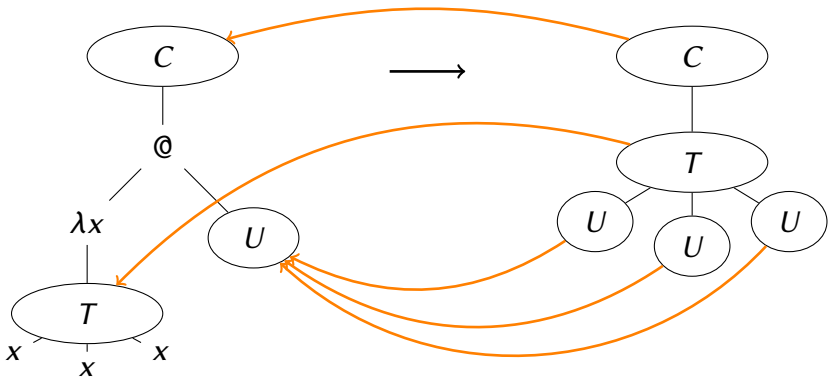
```
type t = Var of string | App of t * t | Abs of string * t
```




```
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)

$$\frac{}{(\lambda x.t)\ s \to\!\triangleright\ t^{\{x \leftarrow s\}}} \qquad \frac{}{\{\lambda x.t\}\ s \to\!\triangleright\ t^{\{x \leftarrow s\}}}$$

$$\frac{t \to\!\triangleright\ t'}{t\ u \to\!\triangleright\ t'\ u} \qquad \frac{u \to\!\triangleright\ u'}{t\ u \to\!\triangleright\ t\ u'} \qquad \frac{\theta \to\!\triangleright\ t'}{\{\theta\} \to\!\triangleright\ t'}$$

A model of space cost

Parameters

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

shadows of an implicit initial term

Space cost

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

Implementation of the marked λ-calculus

Programs and configurations

Compilation

$$
\begin{array}{rcl}
(\!|x|\!)_\rho & = & \mathsf{VAR}\ k \\
(\!|t\ u|\!)_\rho & = & \mathsf{APP}\ ;\ (\!|t|\!)_\rho\ ;\ (\!|u|\!)_\rho \\
(\!|\lambda x.t|\!)_\rho & = & \mathsf{LAM}\ ;\ (\!|t|\!)_{x;\rho}\ ;\ \mathsf{MAL}
\end{array}
$$

de Bruijn: $\rho[k] = x$
prefix application
explicit range

Configuration

input tape: pure program, RO
init: $(\!|t_0|\!)_\varepsilon$

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

Execution

Computation rules

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

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

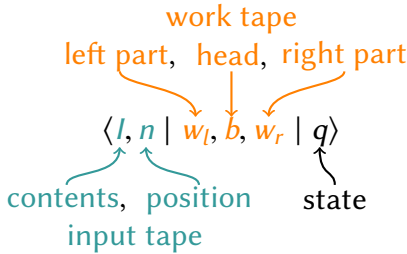
Auxiliary operations

- ***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

Simulating Turing machines

Configurations



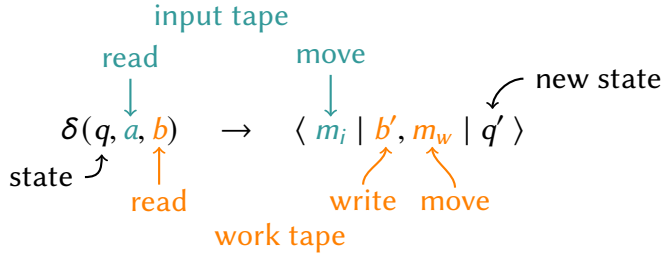
Representation in the λ-calculus: tuple

$$\langle\!\langle C \rangle\!\rangle \quad = \quad \lambda x.x\ I\ n\ w_l\ b\ w_r\ q$$

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



Representation of a character: selector

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

Execution: 3 cascading selections

$$q\ \{\mathrm{Q}_{q_1}\}\ \{\mathrm{Q}_{q_2}\}\ \cdots\ \{\mathrm{Q}_{q_m}\}$$

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

Scott representation of sequences

Algebraic data type = selector + tuple

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

Pattern matching = selection of a function

$$
\begin{array}{l}
\text{Succ} = \; \text{Fix} \;\; (\lambda succ . \lambda n . \\
\qquad n \\
\qquad (\lambda s . \; \text{Cons}_1 \;\; s) \\
\qquad (\lambda s . \; \text{Cons}_0 \;\; (succ \; s)) \\
\qquad \quad \text{Cons}_1 \quad \text{Nil} \\
\qquad)
\end{array}
$$

```
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

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

Pattern matching = selection of a function

$$
\begin{array}{l}
\text{Succ} = \{\text{Fix}\}\ \{\lambda succ . \lambda n . \\
\quad n \\
\quad \{\lambda s . \{\text{Cons}_1\}\ s\} \\
\quad (\lambda s . \{\text{Cons}_0\}\ (succ\ s)) \\
\quad \{\{\text{Cons}_1\}\ \{\text{Nil}\}\} \\
\quad \}
\end{array}
$$

```
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)

Conclusion

Lambda-calculus and space complexity

⟨ *hal-05322360* ⟩

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

Open questions

- *Strong* reduction?
- Reasonable model for space *and time*?