THE STRONG INVARIANCE THESIS FOR A $\lambda$-CALCULUS
LOLA workshop 2017

Yannick Forster $^1$  Fabian Kunze $^{1,2}$  Marc Roth $^{1,3}$

$^1$Saarland University
$^2$Max Planck Institute for Informatics
$^3$Cluster of Excellence (MMCI)
TURING MACHINES
TURING MACHINES (PROS)

- easy to imagine
- easy to explain
- de-facto the standard model of computation for computation theory and complexity theory
TURING MACHINES (CONS)

Notoriously hard to reason about (in a formally precise way):

- not compositional
- tedious encodings
- no nice abstractions for verification (e.g. no separation logic)
- Formalisation of Computability Theory is out of reach
- Formalisation of Complexity Theory is even further away
Exemplary related work

- Ugo Dal Lago and Simone Martini
  *The Weak Lambda Calculus as a Reasonable Machine*
  Theoretical Computer Science, 2008

- Beniamino Accattoli and Ugo Dal Lago
  *(Leftmost-Outermost) Beta Reduction is Invariant, Indeed*
  Logical Methods in Computer Science, 2016
EXEMPLARY RELATED WORK

- Ugo Dal Lago and Simone Martini  
  *The Weak Lambda Calculus as a Reasonable Machine*  
  Theoretical Computer Science, 2008

- Beniamino Accattoli and Ugo Dal Lago  
  *(Leftmost-Outermost) Beta Reduction is Invariant, Indeed*  
  Logical Methods in Computer Science, 2016

- Andrea Asperti and Wilmer Ricciotti  
  *A formalization of multi-tape Turing machines*  
  Theoretical Computer Science, 2015
A certain flavour of λ-calculus called L

- compositional
- straightforward encodings of data types
- equational reasoning for verification
- Formalisation for Computability theory

Yannick Forster and Gert Smolka
Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq
ITP 2017

- Reasonable with respect to time [Dal Lago, Martini (2008)]
- Reasonable with respect to space?
(Strong) Invariance Thesis

‘Reasonable’ machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space.

[Slot, van Emde Boas (1998)]
THE INVARIANCE THESIS

(Strong) Invariance Thesis

‘Reasonable’ machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space.

Ensures consistency w.r.t classes closed under poly-time/constant-space reductions.

[Slot, van Emde Boas (1998)]
CONTRIBUTION

▶ Simple time and space measures for L
▶ substitution-based interpreter with constant-factor overhead in space
▶ heap-based interpreter with polynomially bounded overhead in time
▶ hybrid interpreter fulfilling the strong invariance thesis
Introduction

CONTRIBUTION

Theorem (Strong Invariance Thesis for L)

$L$ and Turing Machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space for decision functions with non-sublinear running time.
L: **Weak Call-by-Value λ-Calculus**

\[
s, t ::= x \mid \lambda x.s \mid s \cdot t
\]

- \((\lambda x.s)(\lambda y.t) \succ s[x := \lambda y.t]\)
- \(st \succ s't\)
- \(st \succ st'\)

- uniformly confluent (reductions to normal forms have the same length)
- data represented by abstractions (Scott encoding)
- recursion using fixed-point combinator

[Dal Lago, Martini (2008)]
Time measure

If 

\[ s = s_0 \succ s_1 \succ \cdots \succ s_k \]

then

\[ \text{Time}(s) := k \]

i.e. the number of $\beta$-reduction steps
**Space measure**

\[
\text{Space}(s) := \max_{\{s_i | s \succ *s_i\}} |s_i|
\]

i.e. size of the largest intermediate term of the reduction for

\[
|x| = \text{de Bruijn index of } x
\]

\[
|st| = 1 + |s| + |t|
\]

\[
|\lambda x.s| = 1 + |s|
\]
**Definition of Turing machine**

- a finite type of states $Q$
- a transition function $\delta : Q \times \Sigma^{n+1} \rightarrow Q \times \Sigma^{n+1} \times \{L, N, R\}$
- a start state $s : Q$
- a halting function $Q \rightarrow \mathbb{B}$

Semantics: Loop $\delta$ until a halting state is reached.

[Asperti, Ricciotti (2015)], [Dal Lago, Martini (2008)]
Definition of Turing machine

- a finite type of states $Q$
- a transition function $\delta : Q \times \Sigma^{n+1} \rightarrow Q \times \Sigma^{n+1} \times \{L, N, R\}$
- a start state $s : Q$
- a halting function $Q \rightarrow \mathbb{B}$

Semantics: Loop $\delta$ until a halting state is reached.
Encode $\delta$ and halting function using Scott encodings (linear size, polynomial operations) and loop.

[Asperti, Ricciotti (2015)], [Dal Lago, Martini (2008)]
Definition of Turing machine

- a finite type of states $Q$
- a transition function $\delta : Q \times \Sigma^{n+1} \rightarrow Q \times \Sigma^{n+1} \times \{L, N, R\}$
- a start state $s : Q$
- a halting function $Q \rightarrow \mathbb{B}$

Semantics: Loop $\delta$ until a halting state is reached.
Encode $\delta$ and halting function using Scott encodings (linear size, polynomial operations) and loop.

In Coq:
Generation and verification of L-code from functional specification is automatic with our framework.
Time-complexity of the extract is semi-automatic.
Space-complexity has to be done by hand.

[Asperti, Ricciotti (2015)], [Dal Lago, Martini (2008)]
Theorem (Invariance thesis part I)

$L$ can simulate Turing machines with a polynomially bounded overhead in time and a constant-factor overhead in space.
**Example: Evaluating by Substitution**

Let $I := \lambda x. x$:

$$(\lambda y. x \ x) \ I \ ((\lambda y. x \ x) \ II)$$
**EXAMPLE: EVALUATING BY SUBSTITUTION**

Let $I := \lambda x. x$:

$$(\lambda xy. x x) \ I \ ((\lambda xy. x x) II) \triangleright (\lambda y. I) \ I \ ((\lambda xy. x x) II)$$
Example: Evaluating by substitution

Let $I := \lambda x. x$:

$$(\lambda y. x \ x) \ I \ ((\lambda y. x \ x) \ II) \triangleright (\lambda y. I \ I) \ ((\lambda y. x \ x) \ II)$$

$$\triangleright (\lambda y. I \ I) \ ((\lambda y. II) \ I)$$
Example: Evaluating by substitution

Let $I := \lambda x. x$:

$$(\lambda x. x) \ I \ ((\lambda x. x) \ II) \succ (\lambda y. I \ I) \ ((\lambda x. x) \ II)$$

$$\succ (\lambda y. I \ I) \ ((\lambda y. II) \ I)$$

$$\succ (\lambda y. I \ I) \ (I \ I)$$
Example: Evaluating by Substitution

Let $I := \lambda x. x$:

$$(\lambda xy. x x) I ((\lambda xy. x x) I) \gg (\lambda y. I I) ((\lambda xy. x x) I)$$

$$\gg (\lambda y. I I) ((\lambda y. I I) I)$$

$$\gg (\lambda y. I I) (I I)$$

$$\gg (\lambda y. I I) I$$
Example: Evaluating by substitution

Let \( I := \lambda x.x : \)

\[
(\lambda xy.x \ x) \ I \ ((\lambda xy.x \ x) \ II) \succ (\lambda y. I \ I) \ ((\lambda xy.x \ x) \ II)
\]

\[
\succ (\lambda y. I \ I) \ ((\lambda y. II) I)
\]

\[
\succ (\lambda y. I \ I) \ (I \ I)
\]

\[
\succ (\lambda y. I \ I) I
\]

\[
\succ II
\]
Example: Evaluating by substitution

Let $I := \lambda x. x$:

$$(\lambda xy. x\ x)\ I\ ((\lambda xy. x\ x)\ II) \triangleright (\lambda y. I\ I)\ ((\lambda xy. x\ x)\ II)$$

$$\triangleright (\lambda y. I\ I)\ ((\lambda y. II)\ I)$$

$$\triangleright (\lambda y. I\ I)\ (I\ I)$$

$$\triangleright (\lambda y. I\ I)$$

$$\triangleright I\ I$$

$$\triangleright I$$
Encoding Terms

- terms: prefix notation with tokens @, λ, ▷ and |.
- Positions: strings with tokens @L, @R, λ

Example

\[(\lambda xy.x y) (\lambda x.x) \approx (\lambda\lambda10) (\lambda0)\] is encoded by string @λλ@▷ |▷ λ▷.
In this term, ‘1’ occurs at position @Lλλ@L
**Substitution-based interpreter**

To compute $s \Rightarrow s'$, use tapes pre, funct, arg, post, position:

\[
\cdots \at \lambda \cdots \lambda \cdots \cdots
\]

pre funct arg post

1. Find the first $\beta$-redex,
   - copy to pre until $@\lambda$ is read
   - copy next complete term to funct (and remember its position on the position tape)
   - if the next token is $\lambda$, copy the next term to arg and remaining tokens to post
   - otherwise, move funct onto pre and start from beginning

2. copy funct to pre, replacing variable with arg

3. copy post to pre

[Dal Lago, Martini (2008)]
Complexity analysis

Per step for $s \succ s'$:

$\mathcal{O}(|s|^2)$ time

$\mathcal{O}(|s| + |s'|)$ space
**Complexity analysis**

Per step for $s \succ s'$:

- $O(|s|^2)$ time
- $O(|s| + |s'|)$ space

In total for $s = s_0 \succ s_1 \succ \cdots \succ s_k$:

- $O(\sum_i |s_i|^2)$ time
- $O(max_i |s_i|) = O(\text{Space}(s))$ space
Theorem (Invariance thesis part II for space)

* Turing machines can simulate L with a constant-factor overhead in space. *
EXPLOSIVE TERMS

$\bar{2} := \lambda xy. x (x y)$ can double the size of a term in one step:

$$\bar{2} \, t \succ \lambda y. t \, (t \, y)$$

So, with $I := \lambda x. x$:

$$\underbrace{\bar{2} \, (\bar{2} \, \cdots \, (\bar{2} \, \, I) \, \cdots)}_{k \text{ times}}$$

normalizes in $k$ L-steps, but needs $\Omega(2^k)$ interpretation time
Example: Evaluating with a heap

\[ \bar{2} := \lambda xy. x (x y) \]

\[ \bar{2}(\bar{2}I)I \]
**Example: Evaluating with a heap**

\[ \bar{2} := \lambda xy. x (xy) \]

\[ \bar{2}(\bar{2}I)I \Rightarrow \bar{2}(\lambda y. h_1(h_1y))I \quad h_1 := I \]
**Example: Evaluating with a Heap**

\[ \overline{2} := \lambda xy. x (xy) \]

\[ \overline{2}(\overline{2}I) \Rightarrow \overline{2}(\lambda y. h_1(h_1y))I \quad h_1 := I \]
\[ \Rightarrow (\lambda y. h_2(h_2y))I \quad h_1 := I, h_2 := (\lambda y. h_1(h_1y)) \]
EXAMPLE: EVALUATING WITH A HEAP

\[ 2 := \lambda xy. x (xy) \]

\[ 2(2I)I > 2(\lambda y. h_1(h_1y))I \]
\[ > (\lambda y. h_2(h_2y))I \]
\[ > h_2(h_2h_3) \]

\[ h_1 := I \]
\[ h_1 := I, h_2 := (\lambda y. h_1(h_1y)) \]
\[ h_1 := I, h_2 := \lambda y. h_1(h_1y), h_3 := I \]
Example: Evaluating with a heap

\[2 := \lambda xy. x (xy)\]

\[\begin{array}{ll}
\bar{2} := & \lambda xy. x (xy) \\
\bar{2}(\bar{2}I) \succ & \bar{2}(\lambda y. h_1(h_1y))I \\
\succ & (\lambda y. h_2(h_2y))I \\
\succ & h_2(h_2h_3) \\
\succ^2 & h_2((\lambda y. h_1(h_1y))I)
\end{array}\]

\[h_1 := I\]

\[h_1 := I, h_2 := (\lambda y. h_1(h_1y))\]

\[h_1 := I, h_2 := \lambda y. h_1(h_1y), h_3 := I\]
**Example: Evaluating with a heap**

\[ \bar{2} := \lambda xy. x(y) \]

\[
\bar{2}(\bar{2}I) \succ \bar{2}(\lambda y. h_1(h_1y))I \\
\succ (\lambda y. h_2(h_2y))I \\
\succ h_2(h_2h_3) \\
\succ^2 h_2((\lambda y. h_1(h_1y))I) \\
\succ h_2(h_1(h_1h_4)) \\
\ldots, h_4 := I
\]
**Example: Evaluating with a heap**

\[ \bar{2} := \lambda xy. x (xy) \]

\[
\begin{align*}
\bar{2}(\bar{2}I) & \Rightarrow \bar{2}(\lambda y. h_1(h_1y))I \\
& \Rightarrow (\lambda y. h_2(h_2y))I \\
& \Rightarrow h_2(h_2h_3) \\
& \Rightarrow^2 h_2((\lambda y. h_1(h_1y))I) \\
& \Rightarrow h_2(h_1(h_1h_4)) \\
& \Rightarrow^2 h_2(h_1(\text{II}))
\end{align*}
\]

\[ h_1 := I \]
\[ h_1 := I, h_2 := (\lambda y. h_1(h_1y)) \]
\[ h_1 := I, h_2 := \lambda y.h_1(h_1y), h_3 := I \]
\[ \ldots, h_4 := I \]
Example: Evaluating with a heap

\[ \bar{2} := \lambda xy. x (xy) \]

\[ \bar{2}(\bar{2}I) \Rightarrow \bar{2}(\lambda y. h_1(h_1y))I \]

\[ \Rightarrow (\lambda y. h_2(h_2y))I \quad h_1 := I \]

\[ \Rightarrow h_2(h_2h_3) \quad h_1 := I, h_2 := (\lambda y. h_1(h_1y)) \]

\[ \Rightarrow^2 h_2((\lambda y. h_1(h_1y))I) \quad h_1 := I, h_2 := \lambda y. h_1(h_1y), h_3 := I \]

\[ \Rightarrow h_2(h_1(h_1h_4)) \quad \ldots, h_4 := I \]

\[ \Rightarrow^2 h_2(h_1(I)) \]

\[ \Rightarrow^5 h_2I \quad \ldots, h_5 := I, h_6 := I \]
Example: Evaluating with a heap

\[ \overline{2} := \lambda xy. x (xy) \]

\[ \overline{2}(\overline{2} I) \triangleright \overline{2}(\lambda y. h_1(h_1y)) I \]

\[ \triangleright (\lambda y. h_2(h_2y)) I \]

\[ \triangleright h_2(h_2 h_3) \]

\[ \triangleright h_2((\lambda y. h_1(h_1y)) I) \]

\[ \triangleright h_2(h_1(h_1 h_4)) \]

\[ \triangleright^2 h_2(h_1(I I)) \]

\[ \triangleright^5 h_2 I \]

\[ \triangleright h_1(h_1 h_7) \]
**Example: Evaluating with a heap**

$$\bar{2} := \lambda xy. x (xy)$$

$$\bar{2}(\bar{2}I) \Rightarrow \bar{2}(\lambda y. h_1(h_1y))I$$

$$\Rightarrow (\lambda y. h_2(h_2y))I$$

$$\Rightarrow h_2(h_2h_3)$$

$$\Rightarrow ^2 h_2((\lambda y. h_1(h_1y))I)$$

$$\Rightarrow h_2(h_1(h_1h_4))$$

$$\Rightarrow ^2 h_2(h_1(I))$$

$$\Rightarrow ^5 h_2I$$

$$\Rightarrow h_1(h_1h_7)$$

$$\Rightarrow ^* I$$
HEAP-BASED INTERPRETER

\[ s = s_0 \succ \cdots \succ s_i \succ \cdots \succ s_k \]

Tapes: main (contains \( s_i \)), heap, hc (heap counter)
Invariant: size of heap is always polynomial in \( k \) and \( |s| \).
HEAP-BASED INTERPRETER

\[ s = s_0 \succ \cdots \succ s_i \succ \cdots \succ s_k \]

Tapes: main (contains \( s_i \)), heap, hc (heap counter)
Invariant: size of heap is always polynomial in \( k \) and \(|s|\).

▶ For beta-reduction of \((\lambda x.t_1)t_2\): Copy \( t_2 \) to heap, replace \( x \) in \( t_1 \) with address
**Heap-based interpreter**

\[ s = s_0 \succ \cdots \succ s_i \succ \cdots \succ s_k \]

**Tapes:** main (contains \( s_i \)), heap, hc (heap counter)

**Invariant:** size of heap is always polynomial in \( k \) and \( |s| \).

- For beta-reduction of \((\lambda x.t_1)t_2\): Copy \( t_2 \) to heap, replace \( x \)
in \( t_1 \) with address (linear in the heap, \( O(|t_1|) \) many copies of an address linear in the heap)
HEAP-BASED INTERPRETER

\[ s = s_0 \succ \cdots s_i \succ \cdots \succ s_k \]

Tapes: main (contains \( s_i \)), heap, hc (heap counter)
Invariant: size of heap is always polynomial in \( k \) and \(|s|\).

- For beta-reduction of \((\lambda x. t_1)t_2\): Copy \( t_2 \) to heap, replace \( x \)
in \( t_1 \) with address (linear in the heap, \( \mathcal{O}(|t_1|) \) many copies
of an address linear in the heap)

- For variable-unfolding of \( x \): Find the element associated to \( x \) in the heap
**Heap-based interpreter**

\[ s = s_0 \succ \cdots s_i \succ \cdots \succ s_k \]

Tapes: main (contains \( s_i \)), heap, hc (heap counter)

Invariant: size of heap is always polynomial in \( k \) and \( |s| \).

- For beta-reduction of \((\lambda x.t_1)t_2\): Copy \( t_2 \) to heap, replace \( x \) in \( t_1 \) with address (linear in the heap, \( O(|t_1|) \) many copies of an address linear in the heap)
- For variable-unfolding of \( x \): Find the element associated to \( x \) in the heap (linear in the heap)
**Heap-based interpreter**

\[ s = s_0 \succ \cdots \succ s_i \succ \cdots \succ s_k \]

Tapes: main (contains \(s_i\)), heap, hc (heap counter)

Invariant: size of heap is always polynomial in \(k\) and \(|s|\).

- For beta-reduction of \((\lambda x.t_1)t_2\): Copy \(t_2\) to heap, replace \(x\) in \(t_1\) with address (linear in the heap, \(O(|t_1|)\) many copies of an address linear in the heap)
- For variable-unfolding of \(x\): Find the element associated to \(x\) in the heap (linear in the heap)

A bit more complicated for de-Bruijn, but doable.
Theorem

There is a constant $c$ such that any reduction $s = s_0 \succ \cdots \succ s_k$ in $L$ can be simulated by the heap-based Turing machine in time and space $O(|s| \cdot k^c)$.
Theorem (Invariance thesis part II for time)

*Turing machines can simulate L with a polynomially bounded overhead in time.*
Sub-linear-logarithmicly Small Terms

Let $N := (\lambda xy.x x) I$, then

$$N (\cdots (N I) \cdots) \sim^k (\lambda y.II) (\cdots ((\lambda y.II) I) \cdots) \sim 2^k I$$

Needs $3k$ entries (with addresses of size $O(k)$) on heap, but definition permits only $O(k)$ space
**Complexity Overview**

\[ s = s_0 \succ s_1 \succ \cdots \succ s_k \]

for \( s_k \) with constant size:

<table>
<thead>
<tr>
<th></th>
<th>substitution-based</th>
<th>heap-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>( \mathcal{O}(\sum_i</td>
<td>s_i</td>
</tr>
<tr>
<td>space</td>
<td>( \mathcal{O}(\text{Space}(s)) )</td>
<td>( \mathcal{O}(</td>
</tr>
</tbody>
</table>
Problem analysis

\[ s = s_0 \triangleright \cdots \triangleright s_k \]

Heap-based interpreter needs \( \mathcal{O}(|s| \cdot k^c) \) space on sublinear-logarithmically reducing terms (in \( k \) steps).
PROBLEM ANALYSIS

\[ s = s_0 \succ \cdots \succ s_k \]

Heap-based interpreter needs \( O(|s| \cdot k^c) \) space on sublinear-logarithmically reducing terms (in \( k \) steps).

Substitution-based interpreter needs more than polynomial time on explosive terms where \( |s_i| \) is asymptotically non-polynomial.
PROBLEM ANALYSIS

\[ s = s_0 \succ \cdots \succ s_k \]

Heap-based interpreter needs \( O(|s| \cdot k^c) \) space on sublinear-logarithmically reducing terms (in \( k \) steps).

Substitution-based interpreter needs more than polynomial time on explosive terms where \( |s_i| \) is asymptotically non-polynomial.

But: Heap-based interpreter works on explosive terms!
Hybrid interpreter

Input: A term $s$. Set $k = 0$.
Execute the substitution-based interpreter on $s$ for $k$ steps:

- If a normal form is reached, output it.
- If the space consumption is larger than $|s| \cdot k^c$, abort and use the heap-based interpreter for $k$ steps.
- If no normal form is reached, delete everything except $s$, set $k := k + 1$ and repeat.
TIME ANALYSIS

Running time for fixed $s$ and $k$:

In total:

$O( )$
**TIME ANALYSIS**

Running time for fixed $s$ and $k$:

heap-based interpreter: $|s| \cdot k^c$

In total:

\[ O(\quad) \]
**TIME ANALYSIS**

Running time for fixed $s$ and $k$:

heap-based interpreter: $|s| \cdot k^c$

In total:

$$O\left(\sum_{k=0}^{\infty} |s| \cdot k^c\right)$$
Time analysis

Running time for fixed $s$ and $k$:

unfolding the normal form: $\text{poly}(|s|, \text{Time}(s))$

In total:

$$O\left(\sum_{k=0}^{\text{Time}(s)} |s| \cdot k^c \right)$$

heap-based
TIME ANALYSIS

Running time for fixed $s$ and $k$:

unfolding the normal form: $\text{poly}(|s|, \text{Time}(s))$

In total:

$O(\text{poly}(|t|, \text{Time}(s)) + \sum_{k=0}^{\text{Time}(s)} |s| \cdot k^c)$

unfolding the normal form $t$

heap-based
**Time analysis**

Running time for fixed $s$ and $k$:

- substitution-based interpreter: $O\left(\sum_{i=1}^{k} |s_i|^2\right)$

In total:

$$O\left(\underbrace{\text{poly}(|t|, \text{Time}(s))}_{\text{unfolding the normal form } t} + \sum_{k=0}^{\text{heap-based}} |s| \cdot k^c\right)$$
Running time for fixed $s$ and $k$:

substitution-based interpreter: $O\left(\sum_{i=1}^{k} |s_i|^2\right)$

In total:

$O\left(\text{poly}(|t|, \text{Time}(s)) \right) + \sum_{k=0}^{\infty} |s| \cdot k^c + \sum_{i=1}^{k} |s_i|^2$
TIME ANALYSIS

Running time for fixed $s$ and $k$:

because of the space bound: $|s_i| \leq |s| \cdot k^c$

In total:

$O\left(\text{poly}(|t|, \text{Time}(s)) + \sum_{k=0}^{k} |s| \cdot k^c + \sum_{i=1}^{k} |s_i|^2\right)$
**Time Analysis**

Running time for fixed $s$ and $k$:

because of the space bound: $|s_i| \leq |s| \cdot k^c$

thus $\sum_{i=1}^{k} |s_i|^2 \leq k \cdot |s|^2 \cdot k^{2c}$

In total:

$$O\left(\text{poly}(|t|, \text{Time}(s))\right) + \sum_{k=0}^{k} |s| \cdot k^c + \sum_{i=1}^{k} |s_i|^2$$

- unfolding the normal form $t$
- heap-based
- substitution-based
TIME ANALYSIS

Running time for fixed $s$ and $k$:

because of the space bound: $|s_i| \leq |s| \cdot k^c$

thus $\sum_{i=1}^{k} |s_i|^2 \leq k \cdot |s|^2 \cdot k^{2c}$

In total:

$O(\underbrace{\text{poly}(|t|, \text{Time}(s))}_{\text{unfolding the normal form } t} + \sum_{k=0}^{\infty} |s| \cdot k^{c} \underbrace{\text{heap-based}}_{\text{heap-based}} + \underbrace{k \cdot |s| \cdot k^{2c}}_{\text{substitution-based}})$
**Time analysis**

Simplified:

\[
O\left(\text{poly}\left(|t|, \text{Time}(s)\right) + \sum_{k=0}^{\text{Time}(s)} |s| \cdot k^c + k \cdot |s| \cdot k^{2c}\right)
\]

\[\subseteq O\left(\text{poly}\left(|s|, \text{Time}(s)\right) + \text{Time}(s)^{2c+2} \cdot |s|^2\right)\]
Space analysis

Space consumption for fixed $s$ and $k$:

In total:

$$O\left(\max_{k \leq \text{Space}(s)}\right)$$
Space analysis

Space consumption for fixed \( s \) and \( k \):

substitution-based interpreter: \( \max_{i \in \{0, \ldots, k\}} |s_i| \)

In total:

\( O(\max_{k \leq \text{Space}(s)}) \)
Space analysis

Space consumption for fixed $s$ and $k$:

substitution-based interpreter: $\max_{i \in \{0, \ldots, k\}} |s_i| = \mathcal{O}(\text{Space}_k(s))$

In total:

$$\mathcal{O}(\max_{k \leq \text{Space}(s)})$$
**SPACE ANALYSIS**

Space consumption for fixed $s$ and $k$:

substitution-based interpreter: $\max_{i \in \{0, \ldots, k\}} |s_i| = \mathcal{O}(\text{Space}_k(s))$

In total:

$\mathcal{O}(\max_{k \leq \text{Space}(s)} \text{Space}_k(s))$
Space analysis

Space consumption for fixed $s$ and $k$:

heap-based interpreter: $O(|s| \cdot k^c)$

In total:

$O\left(\max_{k \leq \text{Space}(s)} \text{Space}_k(s)\right)$
Space analysis

Space consumption for fixed $s$ and $k$:

heap-based interpreter: $O(|s| \cdot k^c)$

In total:

$$O(\max_{k \leq \text{Space}(s)} \text{Space}_k(s) + |s| \cdot k^c)$$
Space analysis

Space consumption for fixed $s$ and $k$:

because of the space bound: $\text{Space}_k(s) \geq |s| \cdot k^c$

In total:

$O(\max_{k \leq \text{Space}(s)} \text{Space}_k(s) + |s| \cdot k^c)$
Space analysis

Space consumption for fixed $s$ and $k$:

because of the space bound: $\text{Space}_k(s) \geq |s| \cdot k^c$

In total:

$O\left( \max_{k \leq \text{Space}(s)} \text{Space}_k(s) + \text{Space}_k(s) \right)$
Space analysis

Simplified:

$$\mathcal{O}(\max_{k \leq \text{Space}(s)} \text{Space}_k(s) + \text{Space}_k(s))$$

$$\subseteq \mathcal{O}(\text{Space}(s))$$
Theorem (Strong Invariance Thesis for L)

$L$ and Turing Machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space for decision functions with non-sublinear running time.
## Work in Progress: Formalisation

<table>
<thead>
<tr>
<th></th>
<th>spec</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional correctness of L-interpreters</td>
<td>1192</td>
<td>1390</td>
</tr>
<tr>
<td>L-extraction framework</td>
<td>1316</td>
<td>610</td>
</tr>
<tr>
<td>TM-Interpreter (no verified complexity analysis)</td>
<td>388</td>
<td>335</td>
</tr>
</tbody>
</table>
**Work in progress: Formalisation**

<table>
<thead>
<tr>
<th></th>
<th>spec</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional correctness of L-interpreters</td>
<td>1192</td>
<td>1390</td>
</tr>
<tr>
<td>L-extraction framework</td>
<td>1316</td>
<td>610</td>
</tr>
<tr>
<td>TM-interpreter (no verified complexity analysis)</td>
<td>388</td>
<td>335</td>
</tr>
</tbody>
</table>

Missing:

- TM implementation and verification of L-interpreters
- Time and space analysis of L-interpreters
- Time and space analysis of TM-interpreter
The weak call-by-value $\lambda$-calculus L is as reasonable for complexity theory as Turing machines.
Summary

The weak call-by-value \( \lambda \)-calculus \( L \) is as reasonable for complexity theory as Turing machines.

Future work:

- Formalise the complexity analysis
- Complexity theory \textit{using} \( L \): \( NP \), many-one-reductions, hierarchy theorems, …
Summary

The weak call-by-value λ-calculus L is as reasonable for complexity theory as Turing machines.

Future work:

- Formalise the complexity analysis
- Complexity theory using L: NP, many-one-reductions, hierarchy theorems, …

Thanks!
THE HEAP-BASED INTERPRETER

Use environments on a heap to delay substitutions:

▶ call (thunk) \( c = s\langle E \rangle \): pair of encoded L-term \( s \) and heap-address \( E \)
▶ heap \( H \): list of entries (⊥ or \( c \# E' \)), addressed by position.
▶ call stack \( CS \): list of tuples (\( @_L, c \)) or (\( @_R, c \)) (for \( @_R, c \) fully reduced)
▶ interpreter state: current call \( CC \), \( CS \) and \( H \).
▶ initial state: \( CC = s\langle 0 \rangle \), \( CS = [] \) and \( H = [\bot] \)

Example

The result of \((\lambda x.x) ((\lambda xy.xy) (\lambda x.x)) \triangleright (\lambda x.x) (\lambda x.x y)_{\lambda x.x}^{y} \) is represented by

\[
\text{CC} = (\lambda @ \triangleright |\triangleright\rangle \langle 1 \rangle \\
\text{CS} = [(@_R, (\lambda \triangleright \langle 0 \rangle ) )] \\
\text{H} = [\bot, (\lambda \triangleright \langle 0 \rangle ) |\#0]}
\]
THE HEAP-BASED INTERPRETER (2)

Each step of the interpreter depends on the current call $CC = s\langle E\rangle$:

▶ if $s = s_L s_R$: push $(@_L, s_R[E])$ on CS and set CC to $s_R\langle E\rangle$
▶ if $s = x$: get new CC by lookup of $x$ in E
▶ if $s = \lambda s'$:
  ▶ if CS is empty: the term is fully evaluated
  ▶ if CS = $(@_L, c_R) :: CS'$: set CC := $c_R$ and put $(@_R, CC)$ on stack instead.
  ▶ if CS = $(@_R, \lambda t\langle E'\rangle) :: CS'$: store $s_R\langle E\rangle \# E'$ on heap as $\hat{E}$ and set CC := $t\langle \hat{E}\rangle$
The Heap-based Interpreter (2)

Each step of the interpreter depends on the current call $CC = s\langle E \rangle$:

- if $s = s_L s_R$: push $(@_L, s_R[E])$ on $CS$ and set $CC$ to $s_R\langle E \rangle$
- if $s = x$: get new $CC$ by lookup of $x$ in $E$
- if $s = \lambda s'$:
  - if $CS$ is empty: the term is fully evaluated
  - if $CS = (@_L, c_R) :: CS'$: set $CC := c_R$ and put $(@_R, CC)$ on stack instead.
  - if $CS = (@_R, \lambda t\langle E' \rangle) :: CS'$: store $s_R\langle E \rangle\#E'$ on heap as $\hat{E}$ and set $CC := t\langle \hat{E} \rangle$

Observations for evaluation $s_0 \succ s_1 \succ \cdots \succ s_k$:

- all calls contain subterms of $s$
- Heap contains $\#H = k + 1$ elements, each of size $\leq |s| + 2 \cdot \log(\#H)$
- $CS$ & $CC$ representing $s_i$ have size $O(|s_i|)$

$\Rightarrow$ space consumption: $O((\max_i |s_i|) + k \cdot (|s| + \log(k)))$
THE HEAP-BASED INTERPRETER (2)

Each step of the interpreter depends on the current call $CC = s\langle E\rangle$:

- if $s = s_L s_R$: push $(@_L, s_R[E])$ on CS and set CC to $s_R\langle E\rangle$
- if $s = x$: get new CC by lookup of $x$ in $E$
- if $s = \lambda s'$:
  - if CS is empty: the term is fully evaluated
  - if $CS = (@_L, c_R) :: CS'$: set $CC := c_R$ and put $(@_R, CC)$ on stack instead.
  - if $CS = (@_R, \lambda t\langle E'\rangle) :: CS'$: store $s_R\langle E\rangle\#E'$ on heap as $\hat{E}$ and set $CC := t\langle \hat{E}\rangle$

Observations for evaluation $s_0 \succ s_1 \succ \cdots \succ s_k$:

- all calls contain subterms of $s$
- Heap contains $\#H = k + 1$ elements, each of size $\leq |s| + 2 \cdot \log(\#H)$
- CS & CC representing $s_i$ have size $O(|s_i|)$

$\Rightarrow$ space consumption: $O((\max_i |s_i|) + k \cdot (|s| + \log(k)))$

- time per interpreter step: $O(|s| \cdot \#H + CC + CS)$
- amortized, $poly(|s_0|)$ interpreter-steps per $\beta$-reduction.

$\Rightarrow$ time consumption: $O(poly(k, |s_0|))$
LC: L WITH CLOSURES

\[
p, q, r ::= s[\sigma] \mid p \cdot q \quad (s \in L, \sigma \in \text{list } LC)
\]

\[
\begin{align*}
x[\sigma] \triangleright_{LC} x-th_x \sigma & \quad \text{VAR} \\
\lambda s[\sigma] \cdot \lambda t[\tau] \triangleright_{LC} s[\lambda t[\tau]::\sigma] & \quad \beta \\
st[\sigma] \triangleright_{LC} s[\sigma] \cdot t[\sigma] & \quad \text{APP} \\
p \triangleright_{LC} p' & \quad \text{APPL} \\
p \cdot q \triangleright_{LC} p' \cdot q & \quad \text{APP} \\
q \triangleright_{LC} q' & \quad \text{APP} \\
p \cdot q \triangleright_{LC} p \cdot q' & \quad \text{APP} \\
p \cdot q \triangleright_{LC} p \cdot q' & \quad \text{APP}
\end{align*}
\]