

THE STRONG INVARIANCE THESIS FOR A λ -CALCULUS LOLA WORKSHOP 2017

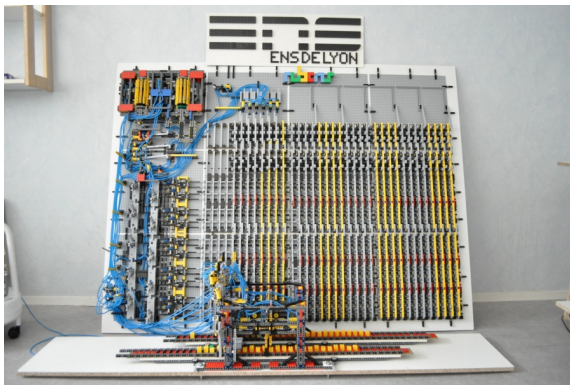
Yannick Forster¹ Fabian Kunze^{1,2} Marc Roth^{1,3}

¹SAARLAND UNIVERSITY

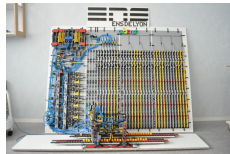
²MAX PLANCK INSTITUTE FOR INFORMATICS

³CLUSTER OF EXCELLENCE (MMCI)

TURING MACHINES

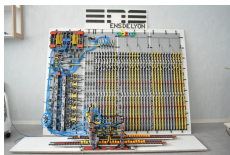


TURING MACHINES (PROS)



- ▶ 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

ANOTHER MODEL OF COMPUTATION

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?

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.

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.

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

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

$$s, t ::= x \mid \lambda x.s \mid st$$

$$\frac{}{(\lambda x.s)(\lambda y.t) \succ s[x := \lambda y.t]}$$

$$\frac{s \succ s'}{st \succ s't}$$

$$\frac{t \succ t'}{st \succ st'}$$

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

TIME MEASURE

If

$$s = s_0 \succ s_1 \succ \cdots \succ s_k$$

then

$$\text{Time}(s) := k$$

i.e. the number of β -reduction steps

SPACE MEASURE

$$\text{Space}(s) := \max_{\{s_i \mid s \multimap^* s_i\}} |s_i|$$

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 δ until a halting state is reached.

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 δ until a halting state is reached.

Encode δ and halting function using Scott encodings (linear size, polynomial operations) and loop.

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 δ until a halting state is reached.

Encode δ 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 $\mathbf{I} := \lambda x.x$:

$(\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\ \mathbf{II})$

EXAMPLE: EVALUATING BY SUBSTITUTION

Let $\mathbf{I} := \lambda x.x$:

$$(\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\mathbf{II}) \succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda xy.x\ x)\mathbf{II})$$

EXAMPLE: EVALUATING BY SUBSTITUTION

Let $\mathbf{I} := \lambda x.x$:

$$\begin{aligned} (\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\mathbf{II}) &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda xy.x\ x)\mathbf{II}) \\ &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda y.\mathbf{II})\mathbf{I}) \end{aligned}$$

EXAMPLE: EVALUATING BY SUBSTITUTION

Let $\mathbf{I} := \lambda x.x$:

$$\begin{aligned}
 (\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\mathbf{II}) &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda xy.x\ x)\mathbf{II}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda y.\mathbf{II})\mathbf{I}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ (\mathbf{I}\ \mathbf{I})
 \end{aligned}$$

EXAMPLE: EVALUATING BY SUBSTITUTION

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

$$\begin{aligned}
 (\lambda xy.x\ x)\ I\ ((\lambda xy.x\ x)\ II) &\succ (\lambda y.I\ I)\ ((\lambda xy.x\ x)\ II) \\
 &\succ (\lambda y.I\ I)\ ((\lambda y.II)\ I) \\
 &\succ (\lambda y.I\ I)\ (I\ I) \\
 &\succ (\lambda y.I\ I)\ I
 \end{aligned}$$

EXAMPLE: EVALUATING BY SUBSTITUTION

Let $\mathbf{I} := \lambda x.x$:

$$\begin{aligned}
 (\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\mathbf{II}) &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda xy.x\ x)\mathbf{II}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda y.\mathbf{II})\mathbf{I}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ (\mathbf{I}\ \mathbf{I}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ \mathbf{I} \\
 &\succ \mathbf{I}\ \mathbf{I}
 \end{aligned}$$

EXAMPLE: EVALUATING BY SUBSTITUTION

Let $\mathbf{I} := \lambda x.x$:

$$\begin{aligned}
 (\lambda xy.x\ x)\ \mathbf{I}\ ((\lambda xy.x\ x)\mathbf{II}) &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda xy.x\ x)\mathbf{II}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ ((\lambda y.\mathbf{II})\mathbf{I}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ (\mathbf{I}\ \mathbf{I}) \\
 &\succ (\lambda y.\mathbf{I}\ \mathbf{I})\ \mathbf{I} \\
 &\succ \mathbf{I}\ \mathbf{I} \\
 &\succ \mathbf{I}
 \end{aligned}$$

ENCODING TERMS

- terms: prefix notation with tokens @, λ , $\triangleright$ and |.
- Positions: strings with tokens @_L, @_R, λ

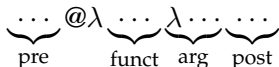
Example

$(\lambda xy.x y) (\lambda x.x) \approx (\lambda \lambda 10) (\lambda 0)$ is encoded by string @ $\lambda \lambda @ \triangleright | \triangleright \lambda \triangleright$.

In this term, '1' occurs at position @_L $\lambda \lambda @_L$

SUBSTITUTION-BASED INTERPRETER

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



- Find the first β -redex,
 - ▶ copy to pre until @λ is read
 - ▶ copy next *complete* term to funct (and remember its position on the position tape)
 - ▶ if the next token is λ, copy the next term to arg and remaining tokens to post
 - ▶ otherwise, move funct onto pre and start from beginning
- copy funct to pre, replacing variable with arg
- copy post to pre

COMPLEXITY ANALYSIS

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

Per step for $s \succ s'$:

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

COMPLEXITY ANALYSIS

Per step for $s \succ s'$:

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

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

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

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

$\mathcal{O}(\max_i |s_i|) = \mathcal{O}(\mathbf{Space}(s))$ 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 $\mathbf{I} := \lambda x.x$:

$$\underbrace{\bar{2} (\bar{2} (\dots (\bar{2} \mathbf{I}) \dots))}_{k \text{ times}}$$

normalizes in k L-steps, but needs $\Omega(2^k)$ interpretation time

EXAMPLE: EVALUATING WITH A HEAP

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

$$\bar{2}(\bar{2}\mathbf{I})\mathbf{I}$$

EXAMPLE: EVALUATING WITH A HEAP

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

$$\bar{2}(\bar{2}\mathbf{I})\mathbf{I} \succ \bar{2}(\lambda y.h_1(h_1y))\mathbf{I} \quad h_1 := \mathbf{I}$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$h_1 := \mathbf{I}$$

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

$$\succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I})$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$h_1 := \mathbf{I}$$

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

$$\succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I})$$

$$\succ h_2(h_1(h_1h_4))$$

$$\dots, h_4 := \mathbf{I}$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$h_1 := \mathbf{I}$$

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

$$\succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I})$$

$$\succ h_2(h_1(h_1h_4))$$

$$\dots, h_4 := \mathbf{I}$$

$$\succ^2 h_2(h_1(\mathbf{II}))$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$h_1 := \mathbf{I}$$

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

$$\succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I})$$

$$\succ h_2(h_1(h_1h_4))$$

$$\dots, h_4 := \mathbf{I}$$

$$\succ^2 h_2(h_1(\mathbf{II}))$$

$$\succ^5 h_2\mathbf{I}$$

$$\dots, h_5 := \mathbf{I}, h_6 := \mathbf{I}$$

EXAMPLE: EVALUATING WITH A HEAP

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

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

$$h_1 := \mathbf{I}$$

$$\succ (\lambda y.h_2(h_2y))\mathbf{I}$$

$$h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y))$$

$$\succ h_2(h_2h_3)$$

$$h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I}$$

$$\succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I})$$

$$\succ h_2(h_1(h_1h_4))$$

$$\dots, h_4 := \mathbf{I}$$

$$\succ^2 h_2(h_1(\mathbf{II}))$$

$$\succ^5 h_2\mathbf{I}$$

$$\dots, h_5 := \mathbf{I}, h_6 := \mathbf{I}$$

$$\succ h_1(h_1h_7)$$

EXAMPLE: EVALUATING WITH A HEAP

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

$$\begin{array}{ll}
 \bar{2}(\bar{2}\mathbf{I})\mathbf{I} \succ \bar{2}(\lambda y.h_1(h_1y))\mathbf{I} & h_1 := \mathbf{I} \\
 \succ (\lambda y.h_2(h_2y))\mathbf{I} & h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1y)) \\
 \succ h_2(h_2h_3) & h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1y), h_3 := \mathbf{I} \\
 \succ^2 h_2((\lambda y.h_1(h_1y))\mathbf{I}) & \\
 \succ h_2(h_1(h_1h_4)) & \dots, h_4 := \mathbf{I} \\
 \succ^2 h_2(h_1(\mathbf{II})) & \\
 \succ^5 h_2\mathbf{I} & \dots, h_5 := \mathbf{I}, h_6 := \mathbf{I} \\
 \succ h_1(h_1h_7) & \\
 \succ^* \mathbf{I} &
 \end{array}$$

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 $|s|$.

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

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)

HEAP-BASED INTERPRETER

$$s = s_0 \succ \dots s_i \succ \dots \succ s_k$$

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

$$s = s_0 \succ \dots s_i \succ \dots \succ s_k$$

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 (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 (linear in the heap)

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

COMPLEXITY ANALYSIS

Theorem

There is a constant c such that any reduction $s = s_0 \succ \dots \succ s_k$ in L can be simulated by the heap-based Turing machine in time and space $\mathcal{O}(|s| \cdot k^c)$.

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 $N := (\lambda xy.x x) \mathbf{I}$, then

$$\begin{aligned}
 & \underbrace{N (\dots (N \mathbf{I}) \dots)}_{k \text{ times}} \\
 & \succ^k \underbrace{(\lambda y.\mathbf{I} \mathbf{I}) (\dots ((\lambda y.\mathbf{I} \mathbf{I}) \mathbf{I}) \dots)}_{k \text{ times}} \\
 & \succ^{2k} \mathbf{I}
 \end{aligned}$$

Needs $3k$ entries (with addresses of size $\mathcal{O}(k)$) on heap, but definition permits only $\mathcal{O}(k)$ space

COMPLEXITY OVERVIEW

$$s = s_0 \succ s_1 \succ \cdots \succ s_k$$

for s_k with constant size:

	substitution-based	heap-based
time	$\mathcal{O}(\sum_i s_i ^2)$	$\mathcal{O}(\text{poly}(\text{Time}(s)))$
space	$\mathcal{O}(\text{Space}(s))$	$\mathcal{O}(s \cdot k^c)$

PROBLEM ANALYSIS

$$s = s_0 \succ \cdots \succ s_k$$

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 $\mathcal{O}(|s| \cdot k^c)$ 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 $\mathcal{O}(|s| \cdot k^c)$ 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:

$\mathcal{O}(\quad)$

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

$\mathcal{O}(\quad)$

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

$$\mathcal{O}\left(\sum_{k=0}^{\text{Time}(s)} \underbrace{|s| \cdot k^c}_{\text{heap-based}}\right)$$

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

$$\mathcal{O}\left(\sum_{k=0}^{\text{Time}(s)} \underbrace{|s| \cdot k^c}_{\text{heap-based}}\right)$$

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

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

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

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

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

$$\mathcal{O}\left(\underbrace{\text{poly}(|t|, \text{Time}(s))}_{\text{unfolding the normal form } t} + \sum_{k=0}^{\text{Time}(s)} \underbrace{|s| \cdot k^c}_{\text{heap-based}} + \underbrace{\sum_{i=1}^k |s_i|^2}_{\text{substitution-based}} \right)$$

TIME ANALYSIS

Running time for fixed s and k :

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

In total:

$$\mathcal{O}\left(\underbrace{\text{poly}(|t|, \text{Time}(s))}_{\text{unfolding the normal form } t} + \sum_{k=0}^{\text{Time}(s)} \underbrace{|s| \cdot k^c}_{\text{heap-based}} + \underbrace{\sum_{i=1}^k |s_i|^2}_{\text{substitution-based}} \right)$$

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:

$$\mathcal{O}\left(\underbrace{\text{poly}(|t|, \text{Time}(s))}_{\text{unfolding the normal form } t} + \sum_{k=0}^{\text{Time}(s)} \underbrace{|s| \cdot k^c}_{\text{heap-based}} + \underbrace{\sum_{i=1}^k |s_i|^2}_{\text{substitution-based}} \right)$$

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:

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

TIME ANALYSIS

Simplified:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

In total:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

substitution-based interpreter: $\max_{i \in \{0, \dots, k\}} |s_i|$

In total:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

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

In total:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

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

In total:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

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

In total:

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

SPACE ANALYSIS

Space consumption for fixed s and k :

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

In total:

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

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:

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

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:

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

SPACE ANALYSIS

Simplified:

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

$$\subseteq \mathcal{O}(\text{Space}(s))$$

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

	spec	proof
Functional correctness of L-interpreters	1192	1390
L-extraction framework	1316	610
TM-interpreter (no verified complexity analysis)	388	335

WORK IN PROGRESS: FORMALISATION

	spec	proof
Functional correctness of L-interpreters	1192	1390
L-extraction framework	1316	610
TM-interpreter (no verified complexity analysis)	388	335

Missing:

- ▶ TM implementation and verification of L-interpreters
- ▶ Time and space analysis of L-interpreters
- ▶ Time and space analysis of TM-interpreter

SUMMARY

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

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, ...

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 ($\perp$ 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 $H = [\perp]$

Example

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

$$CC = (\lambda @ \triangleright | \triangleright) \langle 1 \rangle$$

$$CS = [(@_R, (\lambda \triangleright) \langle 0 \rangle)]$$

$$H = [\perp, (\lambda \triangleright) \langle 0 \rangle \# 0]$$

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 $s_0 \succ s_1 \succ \dots \succ s_k$:

- ▶ 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 $\mathcal{O}(|s_i|)$

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

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 $s_0 \succ s_1 \succ \dots \succ s_k$:

- ▶ 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 $\mathcal{O}(|s_i|)$

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

- ▶ time per interpreter step: $\mathcal{O}(|s_i| \cdot \#H + CC + CS)$
- ▶ amortized, $\text{poly}(|s_0|)$ interpreter-steps per β -reduction.

$\Rightarrow$ time consumption: $\mathcal{O}(\text{poly}(k, |s_0|))$

LC: L WITH CLOSURES

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

$$\frac{}{x[\sigma] \succ_{\mathbf{LC}} x\text{-th}_x \sigma} \text{VAR}$$

$$\frac{}{\lambda s[\sigma] \cdot \lambda t[\tau] \succ_{\mathbf{LC}} s[\lambda t[\tau] :: \sigma]} \beta$$

$$\frac{}{st[\sigma] \succ_{\mathbf{LC}} s[\sigma] \cdot t[\sigma]} \text{APP}$$

$$\frac{p \succ_{\mathbf{LC}} p'}{p \cdot q \succ_{\mathbf{LC}} p' \cdot q} \text{APPL}$$

$$\frac{q \succ_{\mathbf{LC}} q'}{p \cdot q \succ_{\mathbf{LC}} p \cdot q'} \text{APPR}$$