THE STRONG INVARIANCE THESIS FOR A λ-CALCULUS

LOLA WORKSHOP 2017

Yannick Forster [1] Fabian Kunze [1,2] Marc Roth [1,3]

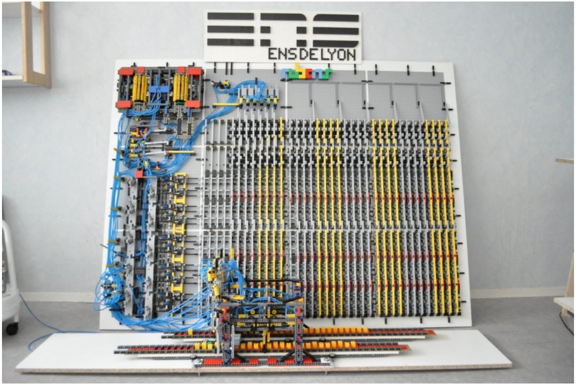
[1]SAARLAND UNIVERSITY

[2]MAX PLANCK INSTITUTE FOR INFORMATICS

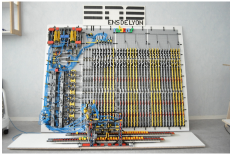
[3]CLUSTER OF EXCELLENCE (MMCI)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

TURING MACHINES

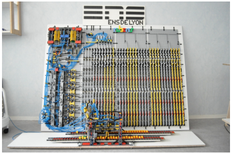


TURING MACHINES (PROS)



- 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.

[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)]

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

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

$$\frac{}{(\lambda x.s)(\lambda y.t) \succ s[x := \lambda y.t]} \qquad \frac{s \succ s'}{st \succ s't} \qquad \frac{t \succ t'}{st \succ st'}$$

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

[Dal Lago, Martini (2008)]

TIME MEASURE

If

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

then

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

i.e. the number of β-reduction steps

SAARLAND UNIVERSITY
COMPUTER SCIENCE

SPACE MEASURE

$$\mathsf{Space}(s) := \max_{\{s_i | s \succ^* s_i\}} |s_i|$$

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

$$\begin{aligned} |x| &= \textit{de Bruijn index of } x \\ |st| &= 1 + |s| + |t| \\ |\lambda x.s| &= 1 + |s| \end{aligned}$$

Definition of Turing machine

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

Semantics: Loop δ until a halting state is reached.

[Asperti, Ricciotti (2015)], [Dal Lago, Martini (2008)]

SAARLAND UNIVERSITY
COMPUTER SCIENCE

DEFINITION OF TURING MACHINE

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

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

[Asperti, Ricciotti (2015)], [Dal Lago, Martini (2008)]

Definition of Turing machine

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

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)]

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

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

Example: Evaluating by substitution

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

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

Example: Evaluating by substitution

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

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

Example: Evaluating by substitution

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

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

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 $@_L\lambda\lambda@_L$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

SUBSTITUTION-BASED INTERPRETER

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

$$\underbrace{\cdots}_{\text{pre}} @\lambda \underbrace{\cdots}_{\text{funct}} \underbrace{\lambda \cdots}_{\text{arg}} \underbrace{\cdots}_{\text{post}}$$

1. Find the first β-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 λ, 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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Complexity analysis

Per step for $s \succ s'$:

$\mathcal{O}(|s|^2)$ time $\quad$ $\mathcal{O}(|s| + |s'|)$ space

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

$\mathcal{O}(\sum_i |s_i|^2)$ time $\quad$ $\mathcal{O}(max_i\, |s_i|) = \mathcal{O}(\mathsf{Space}(s))$ space

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Theorem (Invariance thesis part II for space)

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

Explosive Terms

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

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

So, with $\mathbf{I} := \lambda x.x$:

$$\underbrace{\overline{2}\,(\overline{2}\,(\cdots(\overline{2}}_{k\text{ times}}\ \mathbf{I})\ldots)$$

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

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

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

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\overline{2}(\overline{2}\mathbf{I})\mathbf{I} \succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} \qquad h_1 := \mathbf{I}$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{aligned} \overline{2}(\overline{2}\mathbf{I})\mathbf{I} &\succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} && h_1 := \mathbf{I} \\ &\succ (\lambda y.h_2(h_2 y))\mathbf{I} && h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \end{aligned}$$

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{aligned}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} &\succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} && h_1 := \mathbf{I} \\
&\succ (\lambda y.h_2(h_2 y))\mathbf{I} && h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
&\succ h_2(h_2 h_3) && h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I}
\end{aligned}$$

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{aligned}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} &\succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} && h_1 := \mathbf{I} \\
&\succ (\lambda y.h_2(h_2 y))\mathbf{I} && h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
&\succ h_2(h_2 h_3) && h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I} \\
&\succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I})
\end{aligned}$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{aligned}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} &\succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} && h_1 := \mathbf{I}\\
&\succ (\lambda y.h_2(h_2 y))\mathbf{I} && h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y))\\
&\succ h_2(h_2 h_3) && h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I}\\
&\succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I})\\
&\succ h_2(h_1(h_1 h_4)) && \ldots, h_4 := \mathbf{I}
\end{aligned}$$

EXAMPLE: EVALUATING WITH A HEAP

$\overline{2} := \lambda xy.x\,(x\,y)$

$$
\begin{array}{lll}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} & \succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} & h_1 := \mathbf{I} \\
& \succ (\lambda y.h_2(h_2 y))\mathbf{I} & h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
& \succ h_2(h_2 h_3) & h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I} \\
& \succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I}) & \\
& \succ h_2(h_1(h_1 h_4)) & \ldots, h_4 := \mathbf{I} \\
& \succ^2 h_2(h_1(\mathbf{I}\mathbf{I})) &
\end{array}
$$

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$
\begin{array}{lll}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} & \succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} & h_1 := \mathbf{I} \\
 & \succ (\lambda y.h_2(h_2 y))\mathbf{I} & h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
 & \succ h_2(h_2 h_3) & h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I} \\
 & \succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I}) & \\
 & \succ h_2(h_1(h_1 h_4)) & \ldots, h_4 := \mathbf{I} \\
 & \succ^2 h_2(h_1(\mathbf{I}\mathbf{I})) & \\
 & \succ^5 h_2\mathbf{I} & \ldots, h_5 := \mathbf{I}, h_6 := \mathbf{I}
\end{array}
$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

EXAMPLE: EVALUATING WITH A HEAP

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{array}{lll}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} & \succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} & h_1 := \mathbf{I} \\
 & \succ (\lambda y.h_2(h_2 y))\mathbf{I} & h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
 & \succ h_2(h_2 h_3) & h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I} \\
 & \succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I}) & \\
 & \succ h_2(h_1(h_1 h_4)) & \ldots, h_4 := \mathbf{I} \\
 & \succ^2 h_2(h_1(\mathbf{II})) & \\
 & \succ^5 h_2\mathbf{I} & \ldots, h_5 := \mathbf{I}, h_6 := \mathbf{I} \\
 & \succ h_1(h_1 h_7) &
\end{array}$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Example: Evaluating with a heap

$$\overline{2} := \lambda xy.x\,(x\,y)$$

$$\begin{aligned}
\overline{2}(\overline{2}\mathbf{I})\mathbf{I} &\succ \overline{2}(\lambda y.h_1(h_1 y))\mathbf{I} && h_1 := \mathbf{I} \\
&\succ (\lambda y.h_2(h_2 y))\mathbf{I} && h_1 := \mathbf{I}, h_2 := (\lambda y.h_1(h_1 y)) \\
&\succ h_2(h_2 h_3) && h_1 := \mathbf{I}, h_2 := \lambda y.h_1(h_1 y), h_3 := \mathbf{I} \\
&\succ^2 h_2((\lambda y.h_1(h_1 y))\mathbf{I}) && \\
&\succ h_2(h_1(h_1 h_4)) && \ldots, h_4 := \mathbf{I} \\
&\succ^2 h_2(h_1(\mathbf{II})) && \\
&\succ^5 h_2\mathbf{I} && \ldots, h_5 := \mathbf{I}, h_6 := \mathbf{I} \\
&\succ h_1(h_1 h_7) && \\
&\succ^* \mathbf{I} &&
\end{aligned}$$

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)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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 $s = s_0 \succ \cdots \succ s_k$ *in L can be simulated by the heap-based Turing machine in time and space* $\mathcal{O}(|s| \cdot k^c)$.

SAARLAND UNIVERSITY

COMPUTER SCIENCE

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

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

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

SAARLAND UNIVERSITY COMPUTER SCIENCE

Complexity Overview

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

for s_k with constant size:

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Problem analysis

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

Heap-based interpreter needs $\mathcal{O}(|s| \cdot k^c)$ space on sublinar-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 sublinar-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 sublinar-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:

$$\mathcal{O}(\qquad)$$

Time analysis

Running time for fixed s and k:

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

In total:

$\mathcal{O}($ $)$

SAARLAND UNIVERSITY

COMPUTER SCIENCE

TIME ANALYSIS

Running time for fixed s and k:

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

In total:

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

SAARLAND UNIVERSITY

COMPUTER SCIENCE

TIME ANALYSIS

Running time for fixed s and k:

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

In total:

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

Time analysis

Running time for fixed s and k:

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

In total:

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

TIME ANALYSIS

Running time for fixed s and k:

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

In total:

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

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Time analysis

Running time for fixed s and k:

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

In total:

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

Time analysis

Running time for fixed s and k:

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

In total:

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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:

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

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:

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Time analysis

Simplified:

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

$$\subseteq \mathcal{O}(\mathrm{poly}(|s|\,, \mathsf{Time}(s)) + \mathsf{Time}(s)^{2c+2} \cdot |s|^2)$$

Space analysis

Space consumption for fixed s and k:

In total:

$$\mathcal{O}(\max_{k \leq \mathsf{Space}(s)} \quad)$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

SPACE ANALYSIS

Space consumption for fixed s and k:

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

In total:

$$\mathcal{O}(\max_{k \le \mathsf{Space}(s)} \qquad\qquad\qquad\qquad)$$

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Space analysis

Space consumption for fixed s and k:

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

In total:

$$\mathcal{O}(\max_{k\leq\mathsf{Space}(s)} \qquad\qquad)$$

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Space analysis

Space consumption for fixed s and k:

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

In total:

$$\mathcal{O}(\max_{k \leq \mathsf{Space}(s)} \mathsf{Space}_k(s) \qquad\qquad)$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

SPACE ANALYSIS

Space consumption for fixed s and k:

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

In total:

$$\mathcal{O}(\max_{k \leq \mathsf{Space}(s)} \mathsf{Space}_k(s) \qquad\qquad)$$

Space analysis

Space consumption for fixed s and k:

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

In total:

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

Space analysis

Space consumption for fixed s and k:

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

In total:

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

Space analysis

Space consumption for fixed s and k:

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

In total:

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

Space analysis

Simplified:

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

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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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!

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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 $(@_{\mathrm{L}}, c)$ or $(@_{\mathrm{R}}, c)$ (for $@_{\mathrm{R}}$, 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 $(\lambda x.x)\,((\lambda xy.x\,y)\,(\lambda x.x)) \succ (\lambda x.x)\,\underline{(\lambda x.x\,y)^{y}_{\lambda x.x}}$ is represented by

$$\begin{aligned} CC &= (\lambda @ \triangleright\, |\triangleright)\langle 1\rangle \\ CS &= [(@_{\mathrm{R}}, (\lambda\triangleright)\langle 0\rangle)] \\ H &= [\perp, (\lambda\triangleright)\langle 0\rangle)\#0] \end{aligned}$$

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 $(@_{\mathrm{L}}, s_R[E])$ 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 $CS = (@_{\mathrm{L}}, c_R) :: CS'$: set $CC := c_R$ and put $(@_{\mathrm{R}}, \mathrm{CC})$ on stack instead.
  - if $CS = (@_{\mathrm{R}}, \lambda t\langle E'\rangle) :: CS'$: 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 $(@_\mathrm{L}, s_R[E])$ 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 $CS = (@_\mathrm{L}, c_R) :: CS'$: set $CC := c_R$ and put $(@_\mathrm{R}, \mathrm{CC})$ on stack instead.
  - if $CS = (@_\mathrm{R}, \lambda t\langle E'\rangle) :: CS'$: 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 $\#\mathrm{H} = k + 1$ elements, each of size $\leq |s| + 2 \cdot \log(\#\mathrm{H})$
- CS & CC representing s_i have size $\mathcal{O}(|s_i|)$

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

SAARLAND UNIVERSITY

COMPUTER SCIENCE

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 $(@_\mathrm{L}, s_R[E])$ 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 $CS = (@_\mathrm{L}, c_R) :: CS'$: set $CC := c_R$ and put $(@_\mathrm{R}, \mathrm{CC})$ on stack instead.
  - if $CS = (@_\mathrm{R}, \lambda t\langle E'\rangle) :: CS'$: 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 $\#\mathrm{H} = k+1$ elements, each of size $\leq |s| + 2 \cdot \log(\#\mathrm{H})$
- 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: $\mathcal{O}(|s_i| \cdot \#\mathrm{H} + \mathrm{CC} + \mathrm{CS})$
- amortized, $\mathrm{poly}(|s_0|)$ interpreter-steps per β-reduction.

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

LC: L WITH CLOSURES

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

$$\frac{}{x[\sigma] \succ_{\mathrm{LC}} x\text{-th}_x\ \sigma}\ \textsc{Var} \qquad \frac{}{\lambda s[\sigma]{\cdot}\lambda t[\tau] \succ_{\mathrm{LC}} s[\lambda t[\tau]{::}\sigma]}\ \beta$$

$$\frac{}{st[\sigma] \succ_{\mathrm{LC}} s[\sigma]{\cdot}t[\sigma]}\ \textsc{App} \qquad \frac{p \succ_{\mathrm{LC}} p'}{p{\cdot}q \succ_{\mathrm{LC}} p'{\cdot}q}\ \textsc{AppL} \qquad \frac{q \succ_{\mathrm{LC}} q'}{p{\cdot}q \succ_{\mathrm{LC}} p{\cdot}q'}\ \textsc{AppR}$$

SAARLAND UNIVERSITY

COMPUTER SCIENCE