Introduction to the (Weak) Invariance Thesis in λ-Calculus

Haileselassie Gaspar

Supervisors: Femke van Raamsdonk, Jörg Endrullis

there should be a title page (from canvas)

August 18, 2025

Abstract

The idea of computational Invariance is one that is essential to complexity theory, ever since Slot and Van Emde Boas proposed it in 1984 add reference . While this thesis relates to many areas of computability theory, in this paper we will focus on the invariance of λ-calculus. Accatoli and Dal Lago proposed, using a calculus with explicit substitutions called the **Linear Substitution Calculus**, a strategy that relates polynomially to Turing Machines when only time complexity is concerned. In this paper we will show, using this strategy, a more developed proof on why the computational steps of λ are not invariant with respect to the calculus itself.

Sounds a bit like "this thesis" is different from "this paper". I would suggest: "in this paper we..." -> "we ..."

"using this strategy" sounds weird here, like a proof strategy. Maybe drop it and say instead the "steps of this calculus".

more developed with respect to what?

steps of lambda? you mean general lambda calculus, or this particular calculus? or special lambda steps in this calculus?

1 Introduction

During the start of the 20th century, the idea of computability started to be a main mathematical problem. David Hilbert posed the question in his set of problems to solve during the 20th century: What does it mean for a function to be *computable*? The first step in answering this question would be to define the concept of *computability*. Intuitively, computability is a property of problems or functions that can be solved by some mechanical process. Both Alonzo Church and Alan Turing proved almost simultaneously that there exists a model defining the group of computable functions, the former through general recursive functions and the latter with the conceptual "Turing" machines. The analysis of computability and complexity of functions and algorithms is done through a computational model, or *machine model*.

2 x start

the following question

sudden jump in thoughts / break of flow

1.1 Machine models

While there are many models of computation, for the context of this paper it is only necessary to introduce two.

maybe name them

Turing Machines In 1936, Alan Turing proposed a model now referred to as a *Turing Machine* in order to analyze the Halting Problem [7]. This machine has an unlimited memory in the form of a tape, and a set of symbols already present on the tape. This is the main computational model most used when talking about complexity of algorithms [6].

While the details of its logical implementation are not important in this paper, we will note that the reason they are very useful when analyzing the complexity of algorithms is due to the fact that they posess a very clear and stable *cost model*. That is, for every step in an algorithm implemented in a Turing Machine, there is a clear cost both when talking in terms of space and time.

how much does it cost? maybe say more clearly that you mean the time or space that a step needs.

space

This idea of*cost* *models* comes from the Halting Problem posed by David Hilbert. While the initial problem relates to Peano arithmetical Axioms, it can be seen in the context of Turing Machines as:

jump in thoughts here. you were just talking about complexity, the halting problem is already earlier.

comma

> *"Is there a Turing Machine $\mathcal{M}$ so that given as an input another Turing Machine $\mathcal{M}'$, $\mathcal{M}$ can decide if $\mathcal{M}'$ halts?"*

decidability of the halting problem (not Turing machines)

While Turing proved that this is impossible in [7], a natural extension of this is to not focus on the decidability of a Turing Machine, but on the number of steps that a Turing Machine that we know is decidable takes to reach a halting state.

what is a decidable Turing machine? you mean that always halts?

Lambda calculus The lambda calculus was conceived as a foundational system for mathematics and logic in the 1930s by Alonzo Church. Although this initial idea was proved inconsistent by Kleene and Rosser in [4]. This led to Church publishing in 1936 a simplified version of this system with a focus on computability, now called the *untyped lambda calculus*. A more formal introduction to this system will be provided in Section 2.1. λ-calculus can also be seen as a Higher-Order Term Rewriting System, another concept that will not be introduced in this paper.

the idea was inconsistent or the calculus?

yet another thing you will not tell us about :) I would drop "another ...".

1.2 Invariance

Due to the design of Turing Machines, they serve as a more accurate and measurable way of understanding the space and time that computations need in order to produce an output. While they compute the same functions as λ-calculus does, the use of a single unitary cost model for a computational step in time, with the number of head movements, and in space, with the amount of cells used, provide a clearer view of the complexity of an algorithm, which is why they are the most used method of measuring complexity nowadays. First, it is important to note the difference between equivalence and invariance, two terms very often used in the literature of computation. Equivalence refers to the ability of any model to simulate a Turing machine, while invariance, in the words of Van Embde Boas, stipulates the following:

"more" compared to what?

"same functions" should be said earlier, more prominently

very long sentence, maybe split?

Turing machines are the most used method for measuring complexity? For complexity of algorithms, we don't use Turing machines (e.g. RAM access is constant time).

jump in thoughts, how are equivalence and invariance related to what you said before?

> *"Reasonable" machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space.*

-

This is an extension to the idea of equivalence between computational models. It proposes that not only is there a method in which machine models can simulate each other, but that there exists such a method so that the overhead is polynomial in time and constant in space. In other words, it would only take a polynomial amount of extra time and a constant amount of extra space to evaluate a term in λ-calculus than in a Turing machine and viceversa. – talk about cost models a bit and explain it in terms of that – This paper will use the *weak* invariance thesis, meaning the space requirements, a long-standing problem with lambda calculus, will be dropped. In [2], it is proven that there is an implementation of Turing Machines in the deterministic λ-calculus, with only a linear overhead in time when only head reduction is considered. The differences will be explained in Section 2.4 This paper will introduce the reader to many terms and concepts necessary to understand complexity theory, but it should be viewed as nothing more than a small introduction to the subject with an interesting example, the invariance of the complete untyped λ-calculus with regards to Turing machines.

In order to acquire a more in depth understanding of the topic, refer to the references, that although incomplete, may provide a more formal introduction into the subject.

evaluated a term ... than in a Turing machine? What terms? Lambda terms in Turing machines?

differences between what?

, -> : Otherwise it reads like an enumeration

in-depth?

In order to do what?

incomlete is not good, just mention somehwere that this is a large body of literature and you can only consider a small number of papers

2 Theoretical Background

2.1 Introduction to λ-calculus

TRS is usually singular "term rewriting system"

why say the first sentence, you don't introduce TRSs anyway? you can of course say that lambda predates TRSs

I would not just say lambda, but lambda-calcul us

While an introduction to TRS is a proper base to start understanding the concepts of λ, the idea of it predates that of TRS.

The idea of λ-calculus, or its current use, is based on the idea of computability of functions. When talking about functions there are two main ways to view them. The *extensional* view, which observes only the mapping from input to output, and the *intensional* view, which treats functions not as just a mapping, but a rule. This means that if two functions are given by the same formula, they are *intensionally equal*. This allows mathematicians and computer scientists to talk about the behaviour of a function outside of just what it produces [5].

The λ-calculus uses the intensional view of functions to treat them as expressions, and analyze their behaviour and, more importantly, their computability. It is important to notice that since Turing machines and the λ-calculus treat functions *intensionally*, most of the proof is centered around the process of computation. In order to talk about the invariance of λ-calculus it is first necesary to define some notation that will be used in this paper.

which proof?

Terms Assuming a countably infinite set $\mathcal{VAR} = \{x, y, z, \cdots\}$ of variables et $M, N, P....$ denote arbitrary λ-terms. The set of λ-terms Λ is inductively defined as:

typo

$$\text{Variables: } x \in \Lambda$$
$$\text{Abstraction: } M \in \Lambda \implies (\lambda x.M) \in \Lambda$$
$$\text{Application: } M, N \in \Lambda \implies (MN) \in \Lambda$$

$$C ::= \langle\cdot\rangle \mid \lambda x.C \mid Ct \mid tC$$

Substitution This topic will be expanded on Section 2.1.2, but a basic understanding is provided here. The result of substituting N for the free ocurrences of x in M (notation $M\{x := N\}$) is inductively defined as:

$$\begin{aligned}
&x\{x := N\} \equiv N \\
&y\{x := N\} \equiv y \text{ iff } x \neq y \\
&(\lambda y.M_1)\{x := N\} \equiv \lambda y.(M_1\{x := N\}) \\
&(M_1 M_2)\{x := N\} \equiv (M_1\{x := N\})(M_2\{x := N\})
\end{aligned}$$

A free variable is any variable not bound by an abstraction. For example:

$$(\lambda x.xyz)$$

In this term, y and z are not bound, therefore free, while x is bound by the abstraction λx. When replacing terms it is important to take care not to accidentally bind a free variable by mistake. That is why when replacing terms, if any free variables would become bound by the replacement, they must be renamed. This is called α-conversion. Note that we use braces to denote *implicit* substitutions due to the fact that when introducing *explicit* substitutions we will use the classical brackets, but when reading literature of the classical lambda-calculus, square brackets will mean implicit substitution. For further reading on the syntax and axioms of the lambda calculus, refer to [3].

2.1.1 Conversion

alpha-conv.

This section will be introduced non-mathematically, since an intuitive understanding of α conversion suffices to understand the main points of the proof. However, to understand the more finer points, the reader should again refer to [3]. When referring to conversion in the λ-calculus, it is usually in the context of renaming bound variables. As these variables are already "locked" in regards to an abstraction, any renaming of these with the abstraction to another variable name is an idempotent operation, and can be done at any time. The need for this comes from the unintentional possible binding of a variable during a substitution step. A short example:

$$\begin{aligned}
&\text{Let } F = \lambda xy.yx, \text{ then } \forall M, N : \\
&FMN = NM
\end{aligned} \tag{1}$$

What means "this" here? (1) does not lead to this. By this you mean doing substitution wrongly with captuing free variables?

This would follow from the inductive definition of substitution, however, when taking $M = y$ and $N = x$, this leads to the expression $xy \equiv xx$. This comes due to the fact that the substitution of N in M should not capture any free variables in N.

2.1.2 Reduction

A reduction in λ-calculus can be defined as a conversion between terms that contracts the term tree. In this sense, it can be seen as a simplification of the multiple

What do you mean by "contracts" here? Making smaller? Or contracing a redex? I am confused

Just some line above, you said that you will use "conversion" to refer to alpha-conversion. Now you use it differently as a

also because of "simplification" in the next sentence.

rewrite sequence with backwards steps.

abstractions and applications in a term to a more simple, albeit long, form. Once a term is at a point were it has no possible reduction possible, it is in what is called "normal form". It is simple to see the relation with the concept of reduction in TRS, and most of the terminology introduced in that section can be used in regards to λ.

2x possible

lambda-calculus

Reduction Let $\mathbf{R}$ be a notion of reduction on Λ. Then $\mathbf{R}$ induces the binary relations:

What is the symbol Λ ? What is a notion of redcuciton? Introduce it as a binary relation?

$$\begin{aligned} &\to_R \textit{ one step } R\textit{-reduction} \\ &\to_R^* R\textit{-reduction} \\ &=_R \ R\textit{-equality or } R\textit{-convertibility} \end{aligned} \tag{2}$$

You did not speak about beta-reduction before. In general: make sure your story flows. Everything should be introduced in a good order.

Where $\to_R^*$ represents 0, 1 or more reduction steps. On this simple idea of reduction we can define the classical notion of reduction in the lambda calculus, β-reduction. When talking about measuring time complexity in the *lambda*-calculus, this is a good place to start, as it is the main computational device used. It is based on the substitution rule introduced earlier:

What means "on"? On top of it?

$$\beta : (\lambda x.M)N \to M[x := N] \tag{3}$$

An introduction to some terms associated with reduction; An *R-redex* is a term or subterm that is of the form $(\lambda x.M)N$. Now when talking about introducing a "computational cost",β-reduction is the reasonable choice, as it seems to provide a relation to transitions in a Turing machine. However, the problem in this case is with the arbitrary duplication of terms that can occur during a reduction. But before diving into that, a brief explanation of reduction strategies.

Reduction Strategies When talking about reduction strategies, we refer to a map $F : \Lambda \to \Lambda$ such that for all terms $M \in \Lambda : M \xrightarrow{*} F(M)$. That is, there exists one and only one term to which M reduces in this map. Strategies are defined as terminating if for every M with a normal form, for some $n \in \mathbb{N}$,$F^n(M)$ is in normal form, where $M \xrightarrow{n} F^n(M)$.

The reduction strategy that [1] uses is named Leftmost Outermost. This is a normalizing strategy where if M is not in normal form, the leftmost-outermost redex is reduced until it is. Formally, we can define this in the syntax of contexts inductively, but first we introduce the notion of ordering on terms. As ppreviously stated, TRS have a notion of ordering that is based on strings, but for a better understanding of it, it could also be seen as a tree. The ordering introduced by Accatoli and Dal Lago in their paper is based on the idea of contexts, generalizing the notion and allowing for relative positioning of subterms without needing to understand or define the whole term. It is defined as follows [1]:

1. Outside-in order:
   (a) Root: $\langle\cdot\rangle \prec_O C$ for every context $C \neq \langle\cdot\rangle$.
   (b) Contextual closure: If $C \prec_O D$ then $E\langle C\rangle \prec_O E\langle D\rangle$ for any context E.

2. Left-to-right order:

   (a) Application: If $C \prec_p t$ and $D \prec_p u$ then $Cu \prec_L tD$.

   (b) Contextual Closure: If $C \prec_L D$ then $E\langle C\rangle \prec_L E\langle D\rangle$ for any context E.

Both of these orderings are partial orders in relation to terms. A total order is achieved by joining the two:

$$\text{If } C \prec_O D \text{ and } C \prec_L D \text{ then } C \prec_{LO} D.$$

2.2 Size-exploding family

As mentioned in the introduction, there is a family of terms in λ that, in a linear number of steps, reduces to a term of exponential size. While the invariance result of Accatoli and Dal Lago does not concern itself with space invariance, it is important to note that due to the design of Turing Machines, they have a conceptually different relation between space and time complexity as λ does. While the former has the space complexity be bound by the time complexity, that is, the number of used cells on the tape must be equal or less to the amount of times the head moves, the latter has the opposite. In λ-calculus, time complexity, or the amount of β-reduction steps, is bound by the size of the inital and final size. That means that to provide an accurate invariance result, we not only need to fix the inconsistencies with the number of reduction steps in λ, but we must also take care to show the normal form of a term in a way that is at least polynomially related to the size of the initial term, and the number of steps taken, closing the gap between the two ideas.

Consider the example provided by Accatoli and Dal Lago in [1]. It is introduced here to make the following sections easier to follow. We will also define the ordering of the subterms, and show that LO is a total order on it.

$$\begin{gathered}\text{Let } u = yxx, \text{ and consider the sequence of terms } t_n \in \Lambda_{Exp} \\ \text{for } n \in \mathbb{N} \text{ be defined inductively as} \\ t_0 = u \ , \ t_{n+1} = (\lambda x.t_n)u \text{ for every } n \in \mathbb{N}.\end{gathered}$$

Lemma 2.1. *Given a tern of the form $(\lambda x.t_n)a$ for some $a \in \Lambda$:*

$$(\lambda x.t_n)a \xrightarrow[LO\beta]{1} (\lambda x.t_{n-1})yaa \tag{4}$$

Proof.

$$(\lambda x.t_n)a = (\lambda x.(\lambda x.t_{n-1})u)a \xrightarrow[LO\beta]{1} (\lambda x.t_{n-1})yxx\{x \leftarrow a\} = (\lambda x.t_{n-1})yaa \tag{5}$$

□

Lemma 2.2. *Given a term of the form $t_{n+1} = (\lambda x.t_n)u$:*

1. *Its β-normal form will be $r_n = yr_nr_n$*

2. *It will reach β-normal form in exactly $n+1$ steps.*

Proof. 1. By induction:

$$t_1 = (\lambda x.t_0)u \xrightarrow[LO\beta]{1} yuu = yr_0r_0 = r_1$$
$$t_{n+1} = (\lambda x.t_n)u = (\lambda x.(\lambda x.t_{n-1})u)u \xrightarrow[LO\beta]{1} (\lambda x.t_{n_1})(yuu) = \tag{6}$$
$$(\lambda x.t_{n-1})r1 \xrightarrow[LO\beta]{*} r_{n+1} = y(r_nr_n)$$

By Lemma 2.1, any term of the form $(\lambda x.t_n)a$ reduces in one step to $(\lambda x.t_{n-1})yaa$,and so any term $(\lambda x.t_n)u$ will evaluate to $(\lambda x.t_{n-1})r_1$, therefore, it is clear that the normal form of t_{n+1} is $r_{n+1} = y(r_nr_n)$

2. By contradiction: Assume that the number of steps from t_n to r_n,k for $\rho : t_n \xrightarrow[LO\beta]{k} r_n$, is $n-1$. Then, after $n-1$ LO β steps, we reach r_n, but by Lemma 2.2.1, and Lemma 2.1 we can see that the form of t_n after $n-1$ steps will be $(\lambda x.u)r_{n-1} \neq r_n$ Now assume that $k = n+1$. Then after n steps there should be another LO β step, from r_n, but by definition of the normal form, this is impossible.

□

Theorem 2.3. *Every λ-term $t \in \Lambda_{Exp}$ reduces in n LO β-reduction steps to a term r_n, with $|r_n| \in O(2^n)$.*

Proof. It follows from Lemma 2.1 and 2.2. □

As an example for the rest of the paper, we will use the term t_2 from this family. In order to define the ordering of this term, it must be first converted to a context based syntax:

$$C_0\langle\lambda x.C_1\langle\lambda x.C_3\langle C_4\langle yxx\rangle C_5\langle yxx\rangle\rangle C_2\langle yxx\rangle\rangle$$

An ordering on this term in LO would then be: – fix this –

$$C_0 \prec_{LO} C_1 \prec_{LO} C_2 \prec_{LO} C_3 \prec_{LO} C_4 \prec_{LO} C_5 \tag{7}$$

by means of the contextual closure of LO. For further reading on the syntax and axioms of the lambda calculus, refer to [3].

2.3 LSC

In order to properly illustrate the properties of the LSC, we will use as an example the **size exploding** family that was introduced in 2.1. Any member of this family could be a good example, but for brevity, this paper will use the previously introduced $t_3 \equiv \lambda x.((\lambda x.(yxx))(yxx)(yxx))(yxx)$. As explained before, this term, under regular β-reduction, will evaluate to an exponential length in a number of steps linear to its size, the main obstacle in the invariance of λ. The linear substitution calculus is

based in **explicit substitutions**, as oposed to the regular implicit substitutions that *beta*-reduction produces. The grammar is defined as follows:

$$t, u, r, p = x \mid \lambda x.t \mid tu \mid t[x \leftarrow u] \tag{8}$$

where $t[x \leftarrow u]$ is an explicit substitution. We need to introduce the idea of a new kind of contexts in order to properly define the reduction strategies for the proof of invariance. Besides the addition of explicit substitution to regular contexts, we introduce **shallow** and **substitution** contexts (S and L respectively):

$$\begin{aligned} S &= \langle\cdot\rangle \mid \lambda x.S \mid St \mid tS \mid S[x \leftarrow t] \\ L &= \langle\cdot\rangle \mid L[x \leftarrow t] \end{aligned} \tag{9}$$

We introduce here the operational semantics of the LSC.There are two elementary reductions, coupled with α-conversion, required to provide the explicit subsitutions up to an equivalnece with the untyped λ-calculus.

$$\begin{aligned} L\langle\lambda x.t\rangle u &\rightarrow_{dB} L\langle t[x \leftarrow u]\rangle \\ S\langle x\rangle[x \leftarrow u] &\rightarrow_{ls} S\langle u\rangle[x \leftarrow u] \end{aligned} \tag{10}$$

Based on this, [1] introduces a new reduction strategy, called **Leftmost-Outermost Useful** or **LOU** which we will use to show that the size explosion family does indeed not grow exponentially in the LSC under this strategy. It is important to note that while the Size-exploding family does not grow exponentially under the Linear Head Reduction of the LSC [2], it would never reach normal form if we included an application before the initial term, and so, an example of the use of LOU on it will show that even when completely reducting the term, the lenght increase will be polynomially related to the size of the initial term.

2.4 Leftmost-Outermost Useful

The definition of *usefulness* in [1] of a substitution step is defined in base to wether a redex is *useful* in its unfolding of the surrounding (shallow) context.

Useful Step A reduction step is useful if it is either a dB-step or a ls-step $S\langle x\rangle \rightarrow_{ls} S\langle r\rangle$ so that the unfolding $r \downarrow_S$:

1. Contains a β-redex,
2. Or is an abstraction and S is an applicative context, that is, a context of the form $A ::= S\langle Lt\rangle$.

We now extend our definition of the complete LO order on redexes to the LSC by introducing a new rule:

Substitution: If $C \prec_p t$ and $D \prec_p$ then $C[x \leftarrow u] \prec_L t[x \leftarrow D]$.

Therefore, we can define the **Leftmost Outermost Useful redex**:

A redex R of a term t is considered the leftmost outermost useful redex of t if $R \prec_{LO} Q$ for every other useful redex Q of t. We write $t \rightarrow_{LOU} u$ for the step reducing the leftmost outermost useful redex of t.

3 Generalizing Invariance

As stated before the measure employed to analyze the time invariance of lambda calculus is the number of transitions in a turing machine. By means of the Linear Substitution Calculus, it is possible to represent even size-exploding terms in Turing machines in polynomial time. The LOU strategy is one strategy that has the properties required for invariance, but that does not mean that it is the only one. Accatoli and Dal Lago in [1] showed that any strategy having the following properties could be an adequate candidate for time invariance.

3.1 High level implementation systems

The purpose of the high level implementation system definition is to provide a rewriting system invariant to lambda calculus. This step is a bridge of sorts in between lambda calculus and turing machines. For this, we need to define this class of rewriting systems, and which properties should they satisfy in order to be invariant to lambda calculus. We want spcifically termintation and polynomial overhead.

$$\rightsquigarrow \text{ terminates iff} \rightsquigarrow_X \text{terminates}$$

Furthermore,

$$t \rightsquigarrow_X^k u \text{ iff } t \rightsquigarrow^h u \downarrow \text{ with } O(h) \in O(k^n) \text{ for some } n \in \mathbb{N} \text{ and } n \leq 0$$

We will demonstrate that the LOU reduction strategy follows this generalization with the previous example of the size-exploding family.

4 Size exploding family

Using the term t_2 as a first example, the first step is to split the term into (shallow) contexts, and order them according to the LO total order. In this term there is only one outermost context and it is a dB redex, which is inherently useful. Therefore, the first step will be:

$$(\lambda x.(\lambda x.yxx)(yxx))(yxx) \rightarrow_{dB} (\lambda x.yxx)(yxx)[x \leftarrow yx'x']$$

Note the α-conversion to avoid unwanted variable capture. At this point, the polynomial algorithm introduced in [1] to find useful ls steps is necessary, although in this paper we will compute them by hand. There are two possible reductions here, for there are two free x variables. – I need to develop this more – We can see that a derivation that under LO β-reduction produced a term of size 2^n produces a term linear in both the size of the initial term and the number of steps taken. It is important to note that due to this example only requiring head substitution, the Linear Head Reduction introduced in [2] suffices, but the LOU strategy is necessary for reaching normal form in every case.

5 Conclusion

While it only concerns itself with time invariance, the LOU strategy is a significant step in proving that λ-calculus and Turing Machines are not only equivalent, but invariant with respect to one another, which could lead to computational models that combine the two, providing the benefits of both. There is much research to be done, and this paper only scratches the surface of the dive that computational theory is, but we expanded on [1] to provide a more exact definition of the problem of size explosion, and to provide a gentler introduction into the topic of invariance in the λ-calculus.

References

[1] Beniamino Accattoli and Ugo Dal Lago. "Beta Reduction is Invariant, Indeed (Long Version)". In: *CoRR* abs/1405.3311 (2014). arXiv: 1405.3311. URL: http://arxiv.org/abs/1405.3311.

[2] Beniamino Accattoli and Ugo Dal Lago. "On the Invariance of the Unitary Cost Model for Head Reduction (Long Version)". In: *CoRR* abs/1202.1641 (2012). arXiv: 1202.1641. URL: http://arxiv.org/abs/1202.1641.

[3] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in logic and the foundations of mathematics. North-Holland, 1984. ISBN: 9780444875082. URL: https://books.google.es/books?id=KbZFAAAAYAAJ.

[4] S. C. Kleene and J. B. Rosser. "The Inconsistency of Certain Formal Logics". In: *Annals of Mathematics* 36 (1935). URL: http://www.jstor.org/stable/1968646.

[5] Peter Selinger. *Lecture Notes on the Lambda Calculus*. 2001. DOI: https://doi.org/10.48550/arXiv.0804.3434.

[6] Michael Sipser. *Introduction to the theory of Computation*. 2013.

[7] Alan Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: (1936). DOI: https://doi.org/10.1112/plms/s2-42.1.230.