A reasonable time measure for the weak call-by-value lambda calculus

Marc Roth

October 30, 2015

We provide a formalization of a generalized version of the time measure for the weak call-by-value lambda calculus as introduced by Dal Lago and Martini in 2008. We will use the language L as formalization of the weak call-by-value lamda calculus. The key insight is the fact that every normalizing reduction of a term is unique up to the order of the beta redexes. In this context, we show that there is only one closed beta redex that reduces to itself in one step. Furthermore, we prove that the weak call-by-value lambda calculus fulfills the weak invariance thesis if we use the number of beta steps during a reduction sequence as time measure. This means that reasonable machines such as Turing Machines and L can simulate each other with only a polynomial overhead in time. We therefore proceed in three steps: First, we represent L terms as closures, an idea introduced by Dal Lago and Accattoli in 2014 and prove that a computation in L can be simulated in the closure calculus with only a polynomial overhead in the number of reduction steps. Second, we present an algorithm for simulating a computation in the closure calculus to argue that a computation in L can by simulated by every reasonable machine with only a polynomial overhead in time. And third, we apply a technique for simulating Turing Machines in the weak call-by-value lambda calculus as shown by Dal Lago in 2008 to prove that L can simulate a Turing Machine with only a polynomial overhead in time. Finally, we use this result to provide a basic notation of classical complexity w.r.t. L and argue that all total computable functions can be implemented in L such that the size of every intermediate term during a reduction sequence does not exceed a polynomial in the number of steps that were done before. The formalization — except for the very last sections dealing with Turing Machines — is done in the proof assistant Coq and continues the work of Fabian Kunze and Yannik Forster.

1 Introduction

We continue the investigation of the weak call-by-value lamda calculus as a reasonable machine, started by Dal Lago and Martini in 2008 [3]. In their paper they proved that it is possible to construct a cost function for reduction steps, such that the weak version of the **invariance thesis** as introduced in [2] is fulfilled:

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

The weak version drops the requirement about space. In this work, we use the language L as a formal implementation of the weak call-by-value lamda calculus. In particlular, the formalization in the proof assistant COQ uses L.
At first, we will take up the result of Dal Lago and Martini in [3], but in a more general fashion: In Section 3 we will show that the weak call-by-value lamda calculus is uniform confluent with respect to the multiset of the beta redexes in a reduction sequence. This means that every computation in L starting with s and converging at the normal form t consists of the same beta redexes. More precisely: Two different reduction sequences to t can be obtained from each other only by permuting the beta redexes. This implies both, Proposition 2 and 4 of [3]. At the end of Section 3, we will provide an abstract time measure w.r.t. to a cost function for the beta redexes
While working on this, we had to handle one-step self divergence. Consider the following terms:

$$\omega := \lambda x.xx$$

$$\Omega := \omega\omega$$

It holds that $\Omega \succ \Omega$ and in Section 4 we will prove that Ω is in fact the only closed beta redex that reduces to itself in one step. Note that this property does not hold in the full lambda calculus. The proof uses size induction and requires two rather strange lemmas.
After this we continue to focus on a concrete time measure for L in Section 5. Our main result will be the proof that setting the abstract cost function to be constant 1 is a valid cost model. More precisely, the weak invariance thesis is fulfilled if we use the number of reduction steps as time measure. The reader should be sceptical at this point: There are terms that reduce in n steps to a term of size $\geq 2^n$ (see e.g.[3], Section 1). We solve this problem by avoiding an explicit representation of a term. This idea origins from Accattoli and Dal Lago (see [1]). In this paper the authors used a calculus of explicit substitutions, the linear substitution calculus (LSC), to show that the number of beta steps of a reduction sequence in the general lamda calculus is a valid cost model if left-most outer-most (LO) derivations are prefered. The key insight was that it suffices to compute over terms of the LSC rather than

the lamda calculus itself. This causes the outcome of a computation to be small in the implicit representation w.r.t. the LSC. However the corresponding term of the lamda calculus - the **declosure** - might be large. Note that we cannot apply this result to show that the weak call-by-value lamda calculus has the same property, since a reduction sequence of L does not correspond to a LO derivation of the full lambda calculus. However, the idea can be adapted. We therefore use a simple and elegant substition calculus provided by Fabian Kunze in his Bachelor Thesis (see [5]). This calculus is denoted by LC and works very well when used to prove the invariance thesis for L to be fulfilled. It should be mentioned that Kunze used this closure calculus to speed up the self interpreter for L, an indicator for the utility we need. Our prove proceeds in three steps:

- First we show that a reduction sequence of L can be simulated by LC with only a polynomial overhead. (Section 5.1)
- Second we show how to simulate a reduction sequence in LC with only a polynomial overhead. (Section 5.2)
- And third we use Dal Lagos and Martinis method for simulating Turing Machines in the weak call-by-value lambda calculus (see [3], Theorem 1) to show that a Turing Machine can be simulated by L with only a polynomial overhead.

Note that we refer to time if we speak of a polynomial overhead. This proves the conjecture in [3], Section 1.2 to be true.
After that we will introduce a notion of classical complexity with respect to L in the obvious way (Section 5.4) and shortly investigate **economic** lamda terms. These terms implement a function without exploding in size. More precisely, when applied to the encoding of a string, an economic lambda term *does not need much more space than time*. Note that this notion of space is not meant as a valid cost model. It rather describes the maximum size a term can reach during a reduction sequence. However, our last result in this work will be the proof, that every total function can be implemented by an economic lamda term. The proof idea is to use the simulation technique of Dal Lago and Martini again, but this time we explicitely attach importance to their concrete time measure, which includes the size of a term. Finally, Sections 3 - 5.1 are formalized in the proof assistant COQ and continue the work of Kunze and Forster as mentioned above (see [5] and [4]). Sections 5.2 - 5.4 deal with Turingmachines and are not formalized in COQ.

2 Preliminaries - kept short

As mentioned above, this Research Immersion Lab continues the work of Fabian Kunze and Yannick Forster (see [5] and [4]). Therefore we assume the reader to be familiar with their results, especially regarding Syntax and Semantic of L as well

as LC. Furthermore we assume familarity with basic computability and complexity theory and the proof assistant COQ.

3 Extending uniform confluence by beta redexes

We extend the notion of a reduction step by the beta redex that is reduced. In the following, let s and t be terms and b be a beta redex.

$$\frac{}{\lambda s \lambda t \rhd_{\lambda s \lambda t} s^0_{\lambda t}} \qquad \frac{s \rhd_b s'}{st \rhd_b s't} \qquad \frac{t \rhd_b t'}{st \rhd_b st'}$$

Fact 1 For all terms s and t, it holds that $s \succ t \Leftrightarrow \exists b\ s \rhd_b t$

Next, we define an extended reduction sequence in the obvious way. Let s and t be terms, b be a beta redex and B a list of beta redexes. Furthermore let ϵ denote the empty list.

$$\frac{}{s \rhd^*_\epsilon s} \qquad \frac{s \rhd_b t \quad t \rhd^*_B u}{s \rhd^*_{b::B} u}$$

Fact 2 $\rhd^*$ is transitive, i.e. $s \rhd^*_A t$ and $t \rhd^*_B u$ imply that $s \rhd^*_{A +\!\!+ B} u$.

We are going to make use of multiset equality, which is defined as follows:

Definition 3 Let A and B lists of beta redexes. A and B are **multiset equal**, denoted by $A \approx B$, if the following holds for all beta redexes b:

$$\#_b A = \#_b B$$

where $\#_b A$ denotes the number of occurences of b in the list A.

Fact 4 $\approx$ is an equivalence relation.

Fact 5 $\approx$ is a congruence relation w.r.t. cons and append.

Note that $\approx$ is decidable since equality on beta redexes is decidable.
We will now state and prove the uniform confluence theorem extended by the notion of a time measure. We start with the extended diamond property.

Lemma 6 (Extended diamond property) Let s, t and u be terms and a and b beta redexes, such that $s \rhd_a t$ and $s \rhd_b u$. It follows that either $t = u$ or there is a term v, such that $t \rhd_b v$ and $u \rhd_a v$ (see Figure 1)

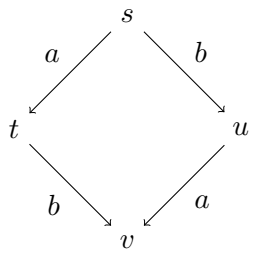

Figure 1: Extended diamond property

Proof By structural induction over s. We only have to consider the case where $s = s_1 s_2$, since otherwise, s cannot reduce. We distinguish three cases:

- $s_1 s_2 \rhd_a t$ via beta reduction. As this was the only possible rule, we conclude that $t = u$.
- $s_1 s_2 \rhd_a s_1 s_2' (= t)$ via $s_2 \rhd_a s_2'$. We observe that $s_1 s_2 \rhd_b u$ cannot be infered via beta reduction, i.e. there are two cases:
  - $s_1 s_2 \rhd_b s_1' s_2 (= u)$ via $s_1 \rhd_b s_1'$. We set $v := s_1' s_2'$ and observe that $s_1 s_2' \rhd_b s_1' s_2'$ and $s_1' s_2 \rhd_a s_1' s_2'$.
  - $s_1 s_2 \rhd_b s_1 s_2'' (= u)$ via $s_2 \rhd_b s_2''$. We use the induction hypothesis on s_2. If $s_2' = s_2''$ then $u = t$. Otherwise we obtain v', such that $s_2' \rhd_b v'$ and $s_2'' \rhd_a v'$. The claim follows by choosing $v = s_1 v'$. (see Figure 2)
- $s_1 s_2 \rhd_a s_1' s_2 (= t)$ via $s_1 \rhd_a s_1'$. This case is symmetrical to the second one. ∎

3.1 Handling self-divergence

At this point, the reader might notice a problem: If we want to show uniform confluence of the extended reduction sequence, we have to deal with the case, that a term s reduces to the same term t with two different beta redexes. It turns out, that the only possibility for the beta redexes to be different is self divergence.

Lemma 7 (Single step property) Let s and t be terms, and a and b beta redexes, such that $s \rhd_a t$ and $s \rhd_b t$. It follows that $a = b$ or $t = s$.

Proof By structural induction over s. We only have to consider the case where $s = s_1 s_2$, since otherwise, s cannot reduce. We distinguish three cases:

- $s_1 s_2 \rhd_a t$ via beta reduction. As this was the only possible rule, we conclude that the beta redex is unique, i.e. $a = b$.
- $s_1 s_2 \rhd_a s_1 s_2' (= t)$ via $s_2 \rhd_a s_2'$. We observe that $s_1 s_2 \rhd_b t$ cannot be infered via beta reduction, i.e. there are two cases:

  - $s_1 s_2 \rhd_b s_1 s_2' (= t)$ via $s_2 \rhd_b s_2'$. The claim follows by applying the induction hypothesis to s_2.
  - $s_1 s_2 \rhd_b s_1' s_2 (= t)$ via $s_1 \rhd_b s_1'$. It follows that $s_1 s_2' = s_1' s_2$, i.e. $s = s_1 s_2 = s_1' s_2' = t$.
- $s_1 s_2 \rhd_a s_1' s_2 (= t)$ via $s_1 \rhd_a s_1'$. This case is symmetrical to the second one. ∎

In Section 4 we will see that Ω is in fact the only **closed** beta redex that reduces to itself in one step.

Next we have to show that internal self-divergence "cannot be undone".

Lemma 8 Let s and u be terms and a and b beta redexes, such that $s \rhd_a s$ and $s \rhd_b u$. It follows that $u \rhd_a u$.

Proof By structural induction over s. We only have to consider the case where $s = s_1 s_2$, since otherwise, s cannot reduce. We distinguish three cases:

- $s_1 s_2 \rhd_b u$ via beta reduction. As this was the only possible rule, we conclude that $s_1 s_2 = u$ and therefore $u \rhd_a u$.
- $s_1 s_2 \rhd_b s_1 s_2' (= u)$ via $s_2 \rhd_b s_2'$. We observe that $s_1 s_2 \rhd_b s_1 s_2$ cannot be infered via beta reduction, i.e. there are two cases:
  - $s_1 s_2 \rhd_a s_1 s_2$ via $s_2 \rhd_a s_2$. We apply the induction hypothesis to s_2 and obtain $s_2' \rhd_a s_2'$, i.e. $u \rhd_a u$.
  - $s_1 s_2 \rhd_a s_1 s_2$ via $s_1 \rhd_a s_1$. It follows that $u \rhd_a u$.
- $s_1 s_2 \rhd_b s_1' s_2 (= u)$ via $s_1 \rhd_b s_1'$. This case is symmetrical to the second one. ∎

Next we lift Lemma 9 to a reduction sequence.

Lemma 9 (Intern divergence) Let s and u be terms, b a beta redex and B a list of beta redexes, such that $s \rhd_b s$ and $s \rhd_B^* u$. It follows that $u \rhd_b u$.

Proof By a simple induction over B and Lemma 9. ∎

3.2 The uniform confluence theorem

To avoid nested induction, we first state and prove the one-sided version of the uniform confluence theorem.

Lemma 10 Let s, t and u be terms, a a beta redex and B a list of beta redexes. If $s \rhd_a t$ and $s \rhd_B^* u$ then there exists a term v and lists of beta redexes C and D, such that $t \rhd_C^* v$, $u \rhd_D^* v$ and $a :: C \approx B +\!\!+ D$ (see Figure 2).

Proof By structural induction over B.

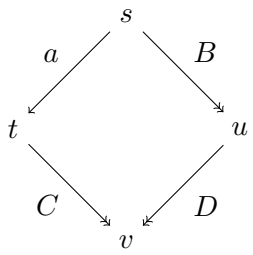

Figure 2: One sided uniform confluence

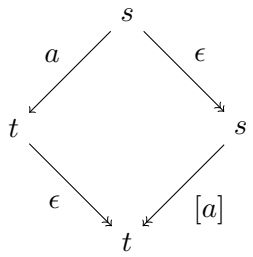

Figure 3: $B = \epsilon$

- $B = \epsilon$. We conclude that $u = s$. The claim follows by choosing $v = t$, $C = \epsilon$ and $D = [a]$ (See Figure 3).
- $B = b :: B'$. It follows that $s \rhd_b u'$ and $u' \rhd^*_{B'} u$ for a term u'. We apply the extended diamond property to s, t and u' and obtain two cases:
  - $t = u'$. Using the single step property, we get yet another two cases:
    - $a = b$. The claim follows by choosing $v = u$, $C = B'$ and $D = \epsilon$ (see Figure 4).
    - $u' = s$. We use intern divergence to obtain that $u \rhd_a u$, i.e. $u \rhd^*_{[a]} u$. The claim follows by choosing $v = u$, $C = b :: B'$ and $D = [a]$ (see Figure 5).
  - $\exists\ v' :\ u' \rhd_a v'$ and $t \rhd_b v'$. Applying the induction hypothesis to u' yields a term v'' and lists of beta redexes C' and D', such that $v' \rhd^*_{C'} v''$, $u \rhd^*_{D'} v''$ and $a :: C' \approx B' +\!\!+ D'$. The claim follows by choosing $v = v''$, $C = b :: C'$ and $D = D'$ since

    $$a :: C \approx a :: b :: C' \approx b :: a :: C' \approx b :: B' +\!\!+ D' \approx B +\!\!+ D.$$

    (See Figure 6)

■

Finally we can prove the uniform confluence theorem.

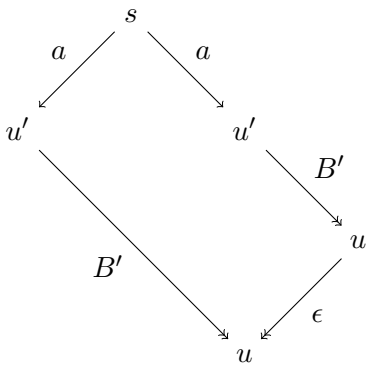

Figure 4: $t = u'$ and $b = a$

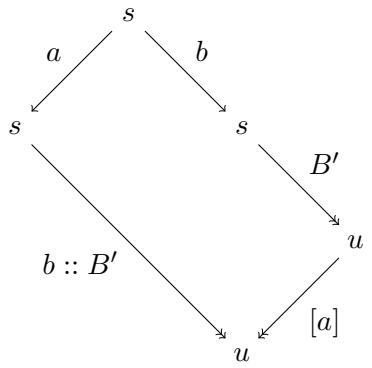

Figure 5: $t = u' = s$

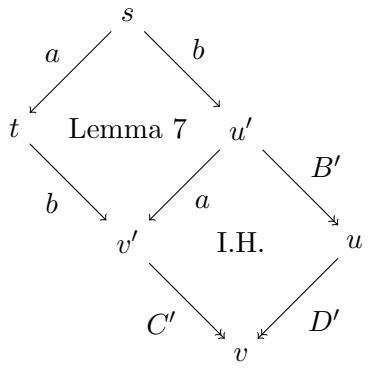

Figure 6: Use of the extended diamond property (Lemma 7) and the induction hypothesis (I.H.)

Theorem 11 (Extended uniform confluence) Let s, t and u be terms and A and B lists of beta redexes. If $s \rhd^*_A t$ and $s \rhd^*_B u$ then there exists a term v and lists of beta redexes C and D, such that $t \rhd^*_C v$, $u \rhd^*_D v$ and $A +\!\!+ C \approx B +\!\!+ D$ (see Figure 7).

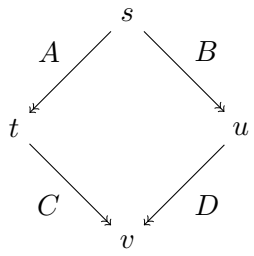

Figure 7: Extended uniform confluence. It holds that $A +\!\!+ C \approx B +\!\!+ D$

Proof By structural induction over A.

- $A = \epsilon$. We conclude that $s = t$. The claim follows by choosing $v = u$, $C = B$ and $D = \epsilon$.
- $A = a :: A'$. The claim follows by applying Lemma 11 and the induction hypothesis. (See Figure 8) ∎

Corollary 12 Every normalizing computation is unique up to the order of the beta reductions.

3.3 An abstract time measure

At this point we have proven that the running time of a term only depends on the cost of the beta reductions. To stay abstract as long as possible, we introduce a cost function $\mathbf{c}$ that maps every beta redex to a positive natural number. Having all the work done, the definition of the running time (w.r.t. $\mathbf{c}$) is almost straight forward. [1]

Definition 13 Let s be a term, t be a normal form and B a list of beta redexes, such that $s \rhd^*_B t$. We define the running time of s with respect to $\mathbf{c}$ as follows:

$$\mathbb{T}_{\mathbf{c}}(s) := |s| + \textstyle\sum_{b \in B} \mathbf{c}(b)$$

Note that $\mathbb{T}_{\mathbf{c}}(s)$ is well-defined, since t and B are unique (the latter up to multiset equality). We will make the time measure concrete after investigating one-step divergence in the next section.

[1] In fact we have to add the size of the input term to obtain a concrete time measure that fulfills the invariant thesis ([3], Section 1)

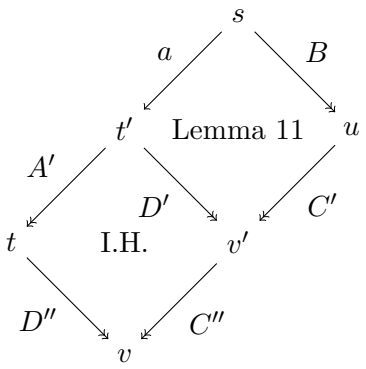

Figure 8: Use of Lemma 11 and the induction hypothesis (I.H.)

4 Uniqueness of self-divergence

In this section, we prove that Ω is the only closed beta redex that reduces to itself in one step. More formally, we will show the following key lemma:

Lemma 14 (Uniqueness Lemma) Let $b := \lambda s \lambda t$ be a beta redex such that λt is closed. If $b \rhd_b b$ then $s = 0\ 0$ and $t = 0\ 0$.

The proof requires some effort. We start with some simple facts and lemmas.

Fact 15 If λt is closed, then t does not contain a variable greater than 0.

Lemma 16 Let s and t be terms and $n > 0$. If $0 = s^n_{\lambda t}$ then $s = 0$.

Proof By a simple case analysis over s. ∎

Lemma 17 Let n be a natural number and v, s and t be terms, such that $|t| > |s|$. If $s = v^n_t$ then $v = s$.

Proof Assuming $v \neq s$, we conclude that n is a variable of v, i.e. n is substituted by t. It follows that $|t| > |s| \geq |t|$ which is a contradiction. ∎

Note that this proof is constructive, since equality on terms is decidable.

Lemma 18 Let t and v be terms and n be a natural number. If $\lambda t = v^n_{\lambda t}$ then either $v = n$ or $v = \lambda t$.

Proof By case analysis over the structure of v:

- If v is a variable n', then either $n = n'$ and therefore $v = n$ or $n \neq n'$ and therefore $\lambda t = v^n_{\lambda t} = n'^n_{\lambda t} = n' = v$.

- If v is an application, then the premise cannot be fulfilled.
- If $v = \lambda u$ for some term u. It follows that $t = u^{n+1}_{\lambda t}$. Since $|\lambda t| > |t|$ we can apply Lemma 18 and obtain that $t = u$ and therefore $\lambda t = \lambda u = v$. ∎

In the following we need another tool for our proofs.

4.1 Obtaining structural insights via size induction

As the title suggests, structural induction as done all along does not suffice to prove the next two lemmas. Instead we will use size induction w.r.t. the length of a term.

Lemma 19 Let n be a natural number and u and t be terms such that λt is closed. If $\lambda(u\ 0) = u^n_{\lambda t}$ then $u = n$ and $t = n\ 0$.

Proof By size induction over the length of u. We distinguish three cases:

- If u is a variable, then u must be n. Furthermore $\lambda(u\ 0) = u^n_{\lambda t} = n^n_{\lambda t} = \lambda t$ and therefore $t = (u\ 0)$.
- If u is an application, then the premise cannot be fulfilled.
- If $u = \lambda v$ for some term v, we have $\lambda((\lambda v)\ 0) = \lambda(u\ 0) = u^n_{\lambda t} = (\lambda v)^n_{\lambda t} = \lambda v^{n+1}_{\lambda t}$ and therefore $(\lambda v)\ 0 = v^{n+1}_{\lambda t}$. It follows that $v = a\ b$ for some terms a, b, i.e. $(\lambda(a\ b))\ 0 = a^{n+1}_{\lambda t}\ b^{n+1}_{\lambda t}$. As a consequence $0 = b^{n+1}_{\lambda t}$ and by applying Lemma 17, we obtain $b = 0$. After substituting b, we have $(\lambda(a\ 0)) = a^{n+1}_{\lambda t}$ and can apply the induction hypothesis on a (Note that $|a| < |u|$ by construction). It follows that $a = n + 1$ and - even more important - that $t = (n + 1)\ 0$. The latter contradicts the assumption that λt is closed (Fact 16). Therefore, u cannot be a lambda term. ∎

Lemma 20 Let n, n' be natural numbers and u, t terms, such that $n' > 0$ and λt is closed. It holds that $\lambda(u\ (\lambda t)) \neq u^n_{\lambda t}$ (I) and $\lambda(u\ n') \neq u^n_{\lambda t}$ (II).

Proof By size induction over the length of u. We distinguish three cases:

- $u = m$ for a variable m.
  (I) If $m = n$ we have $\lambda(u\ (\lambda t)) = \lambda(n\ (\lambda t)) \neq \lambda t = u^n_{\lambda t}$. Otherwise we have $\lambda(u\ (\lambda t)) = \lambda(m\ (\lambda t)) \neq m = m^n_{\lambda t} = u^n_{\lambda t}$.
  (II) If $m = n$ we have $\lambda(u\ n') = \lambda(m\ n')$ and $u^n_{\lambda t} = \lambda t$. It holds, that $\lambda(m\ n') \neq \lambda t$ since λt is closed and $n' > 0$ (Fact 16).
  If $m \neq n$ we have $\lambda(u\ n') = \lambda(m\ n') \neq m = m^n_{\lambda t} = u^n_{\lambda t}$.
- If u is an abstraction we are done.
- $u = \lambda v$ for some term v.

(I) Assume $\lambda(u\ (\lambda t)) = u^n_{\lambda t} \Rightarrow \lambda((\lambda v)\ (\lambda t)) = \lambda(v^{n+1}_{\lambda t})$, i.e. $(\lambda v)\ (\lambda t) = v^{n+1}_{\lambda t}$. It follows that $v = ab$ for some terms a and b. We obtain $(\lambda(a\ b))\ (\lambda t) = a^{n+1}_{\lambda t}\ b^{n+1}_{\lambda t}$ and therefore $\lambda(a\ b) = a^{n+1}_{\lambda t}$ and $\lambda t = b^{n+1}_{\lambda t}$. Applying Lemma 19 yields two cases for b. The contradiction follows in both cases from substituting b and applying the induction hypothesis to a.

(II) Assume $\lambda(u\ n') = u^n_{\lambda t} \Rightarrow \lambda((\lambda v)\ n') = \lambda(v^{n+1}_{\lambda t})$, i.e. $(\lambda v)\ n' = v^{n+1}_{\lambda t}$. It follows that $v = ab$ for some terms a and b. We obtain $(\lambda(a\ b))\ n' = a^{n+1}_{\lambda t}\ b^{n+1}_{\lambda t}$ and therefore $\lambda(a\ b) = a^{n+1}_{\lambda t}$ and $n' = b^{n+1}_{\lambda t}$. Applying Lemma 18 yields $b = n'$ (Note that $|\lambda t| \geq 2 > 1 = |n'|$). It follows that $\lambda(a\ n') = a^{n+1}_{\lambda t}$ which contradticts the induction hypothesis for a. ∎

4.2 Proof of the Uniqueness Lemma

Finally we are able to prove the main result of this section.

Proof If $\lambda s \lambda t$ reduces to itself in one step, it holds that $\lambda s \lambda t = s^0_{\lambda t}$. It follows that $s = u\ v$ for some terms u and v, i.e. $\lambda(u\ v)\lambda t = u^0_{\lambda t}\ v^0_{\lambda t}$ and therefore $\lambda(u\ v) = u^0_{\lambda t}$ and $\lambda t = v^0_{\lambda t}$. Since λt is closed, we can apply Lemma 19 to the latter and obtain two cases:

- $v = 0$. This yields $\lambda(u\ 0) = u^0_{\lambda t}$. It follows that $u = 0$ and $t = 0\ 0$ using Lemma 20. We conclude that $\lambda s \lambda t = \lambda(u\ v)\lambda t = \lambda(0\ 0)\lambda(0\ 0) = \Omega$.
- $v = \lambda t$. This yields $\lambda(u\ \lambda t) = u^0_{\lambda t}$. Since λt is closed, we can apply Lemma 21 to prove this to be impossible. ∎

The Uniqueness Lemma can be lifted to arbitrary closed terms which leads to the following theorem:

Theorem 21 If a closed term s reduces to itself in one step, then the beta redex is equal to Ω

Proof By a simple induction over s using the Uniqueness Lemma ∎

5 A concrete but reasonable time measure

In [3], Dal Lago and Martini provided a time measure for the weak call-by-value lambda calculus such that the weak invariance thesis is fulfilled. Using our framework, Dal Lagos time measure can be defined by setting

$$c(\lambda s \lambda t) := \max\{1, |s^0_{\lambda t}| - |\lambda s \lambda t|\}.$$

Furthermore they conjectured that it would suffice to set c to be constant 1 if one avoids explicit representation of lambda-terms. In 2014 Accatolli and Dal Lago

finally showed a similar statement for the full lambda calculus [1]. As conjectured, they used a non-explicit representation of lamda-terms, namely the linear substition calculus (LSC). Their main result was the proof that using a certain evaluation strategy of a term in the full lambda calculus combined with term representation in LSC, the number of beta steps is an invariant cost model. In this work, we will show that these results can be combined, using the uniform confluence property of L. More precisely, we will show that setting $c := 1$ fulfills the weak invariance thesis, i.e. we prove the original conjecture to be true. Note that we cannot adapt the proof of Dal Lago and Accatolli, since the evaluation strategy used in [1] does not correspond to a reduction in L.

5.1 Simulating L with LC

In this section, we will prove that L can be simulated by LC with only a polynomial overhead of steps. At first, we extend the notion of a reduction step of LC by **true** and **false** steps as follows:

- $p \rhd_{\mathsf{LC}}^{\top} q$ iff $p \succ_{\mathsf{LC}} q$ via a beta step
- $p \rhd_{\mathsf{LC}}^{\bot} q$ iff $p \succ_{\mathsf{LC}} q$ via another step

Next, we lift this notation to a reduction sequence:

$$\frac{}{p \rhd_{\mathsf{LC}}^{(0,0)} p} \qquad \frac{p \rhd_{\mathsf{LC}}^{\top} q \quad q \rhd_{\mathsf{LC}}^{(m,n)} r}{p \rhd_{\mathsf{LC}}^{(m,n+1)} r} \qquad \frac{p \rhd_{\mathsf{LC}}^{\bot} q \quad q \rhd_{\mathsf{LC}}^{(m,n)} r}{p \rhd_{\mathsf{LC}}^{(m+1,n)} r}$$

Lemma 22 True steps of LC correspond one-to-one with steps in L, i.e. for all closed L-terms s and t

$$s \succ^n t \;\Leftrightarrow\; \exists\, m, t_c : s\epsilon \rhd_{\mathsf{LC}}^{(m,n)} t_c \wedge \lfloor t_c \rfloor = t$$

This lemma seems obvious by definition. However, the formal proof consists of many technicalities and is therefore omitted here. It can be found in the COQ formalization.
We will use the following notations:

Definition 23 Let p be a valid LC-term, then

- $|p|_S$ denotes the size of p,
- $|p|_T$ denotes the size of the biggest L-term in p,
- and $|p|_P$ denotes the potential of p. The potential of a valid, LC-term is the number of false steps that can be applied consecutively before applying another rule.

Our goal is to prove that the number of false steps of a reduction sequence in LC is polynomial bounded by the number of true steps. More precisely,

Theorem 24 Let p and q be valid LC-terms and m and n natural numbers, such that $p \rhd_{\mathsf{LC}}^{(m,n)} q$. Then there is a polynomial q, s.t.

$$m \leq q(|p|_P, |p|_T, n)$$

Proof First, we make two observations:

- If $|p|_T$ is bounded by b then so are all LC-terms that are reached from p via a reduction sequence.
- $|p|_P \leq |p|_S$.

Unsurprisingly, $|\cdot|_P$ is used as a potential function: The only steps that can add potential are true steps. As all intermediate LC terms of the reduction sequence contain only L-terms of size $\leq |p|_T$, the overall available potential is bounded by $|p|_P$ (the potential at the beginning) $+q'(|p|_T \cdot n)$ where q' is a polynomial. Finally we observe, that every false step costs at least 1 potential. This very nice property holds only because we are dealing with call-by-value. More precisely, the var-rule substitutes variables only with LC-terms of the form $s[\sigma]$ where s is a lamda, i.e. $|s[\sigma]|_P = 0$. This concludes the proof. Therefore m is indeed bounded by a polynomial in n, $|p|_P$ and $|p|_T$. ∎

In the formalization we achieved the following bound:

$$m \leq |p|_P \cdot (n+1) + |p|_T \cdot (n+1)^2$$

Theorem 25 Let s and t be closed L-terms such that $s \succ^n t$ and t is a normalform, i.e. $\mathbb{T}_1(s) = |s| + n$. Then there is a polynomial q and a valid LC-term t_c such that

$$s\epsilon \succ_{\mathsf{LC}}^{q(\mathbb{T}_1(s))} t_c \wedge \lfloor t_c \rfloor = t$$

Proof It holds that both $|s\epsilon|_P$ and $|s\epsilon|_T$ are smaller or equal than $|s|$ by definition. Therefore the claim follows by applying Lemma 22 and Theorem 24. ∎

5.2 Simulating LC with (something like) a Turing Machine

The purpose of LC was to show that the simulation of an L-term can be done by a Turing Machine (or something equivalent w.r.t. to the invariance thesis) with only a polynomial overhead in time. Therefore this Section will require less effort. In fact, the main work was done in Section 5.1.
Recalling the inference rules of LC, we notice that only one rule can make some trouble:

$$\frac{}{st[\sigma] \succ_{\mathsf{LC}} s[\sigma]t[\sigma]}$$

The problem is, that the size might explode due to copying of lists. We will solve this problem by working with pointers, i.e. it would be better to think of RAM machines instead of Turing Machines. More precisely, we will introduce so called **substitution trees**.

Theorem 26 A computation in LC, starting with $s\epsilon$ for a closed L-term s, can be simulated by a RAM Machine with only a polynomial overhead in time.

Proof First we observe that every explicitely represented term occuring in the reduction sequence is a subterm of s. The reason for this is the fact, that new terms can only be produced via substitutions, but we only deal with explicit substitutions. Therefore every LC-term of the form $t[A]$ can be represented as a pointer to the subterm t of s and to the list A. Furthermore the lists are stored in a tree like structure, which will be the **substitution tree**. Finally a valid LC-term p will be stored as a tree for the structure of p and a substitution tree. Every node of the tree of p either has a pointer to the substitution tree and therefore the form $t[A]$ or no pointer and therefore the form qr. In the following, we will show how to simulate each rule in this representation. Note that, due to uniform confluence, the algorithm may use the first applicable rule it finds. As the overall size will not exceed a polynomial in the number of steps that were done before, searching for an applicable rule in each step only leads to a polynomial overhead and corresponds to the usage of the rules *APPL* and *APPR*. First we will illustrate how to extract a list from a substitution tree. In the following solid arcs will correspond to pointers w.r.t. to the structure of the trees and dashed arcs will represent pointers from terms to lists.

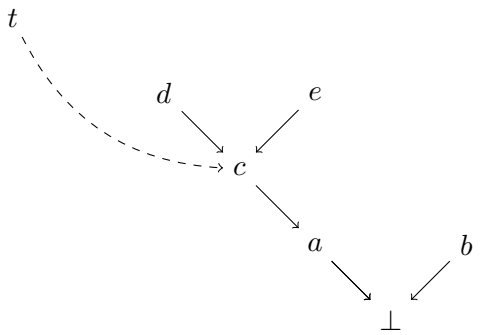

Figure 9: Example for a substitution tree: Let t be an L-term. This corresponds to $t[c;a]$, i.e. the list is obtained by going down from the pointer to the root. Note that each LC-term $a, \cdots, e$ has the form $\lambda v[A]$ itself.

Now we are able to show how to simulate the three remaining rules (see Figures 10,11 and 12). Note that every rule increases the overall size by at most a constant

number of pointers. Now consider a normalizing computation $q \succ_{\mathsf{LC}}{}^{n} r$, starting with a valid LC-term q. If we simulate this computation as described above, there is a polynomial p, such that the overall size of every intermediate term of the derivation is bounded by a $p(|q|_S + n)$. Therefore every step takes time at most $\mathcal{O}(p)$. We conclude that the overall running time of the described algorithm is bounded by a polynomial. ∎

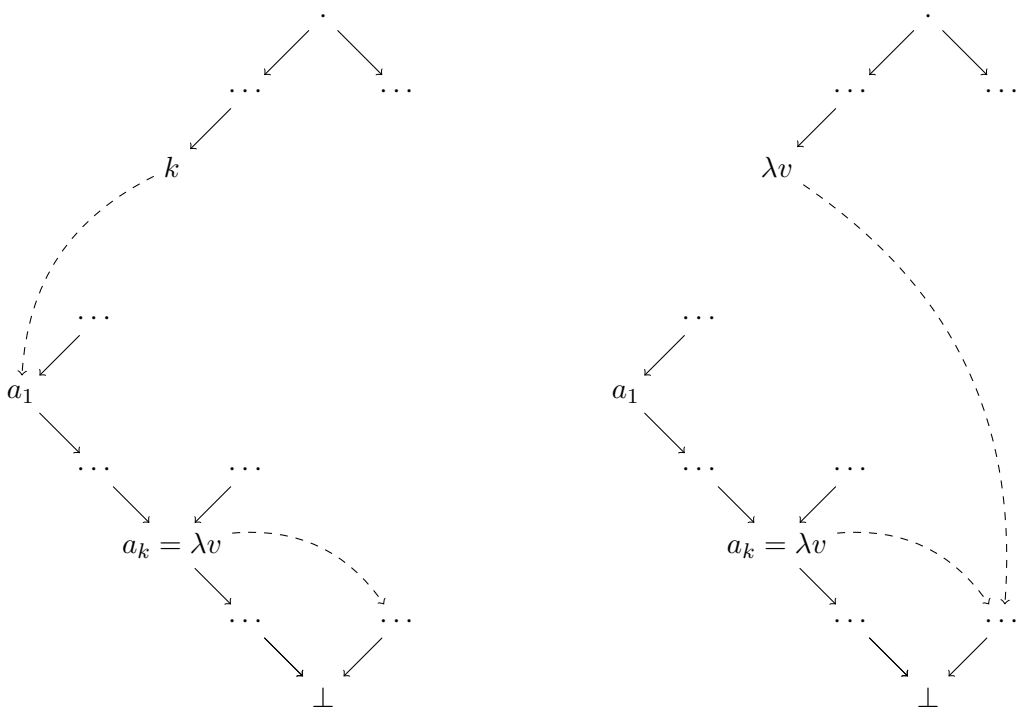

Figure 10: Application of the *VAR*-rule: $k[a_1, \cdots] \succ_{\mathsf{LC}} a_k$

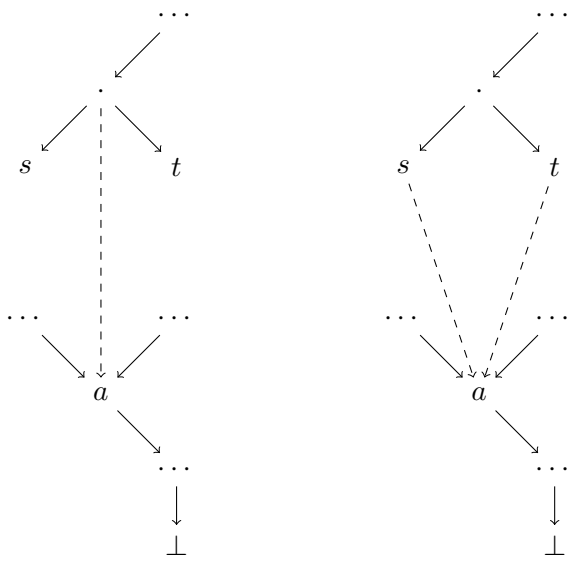

Figure 11: Application of the *APP*-rule: $st[a, \cdots] \succ_{\mathsf{LC}} s[a, \cdots]t[a, \cdots]$

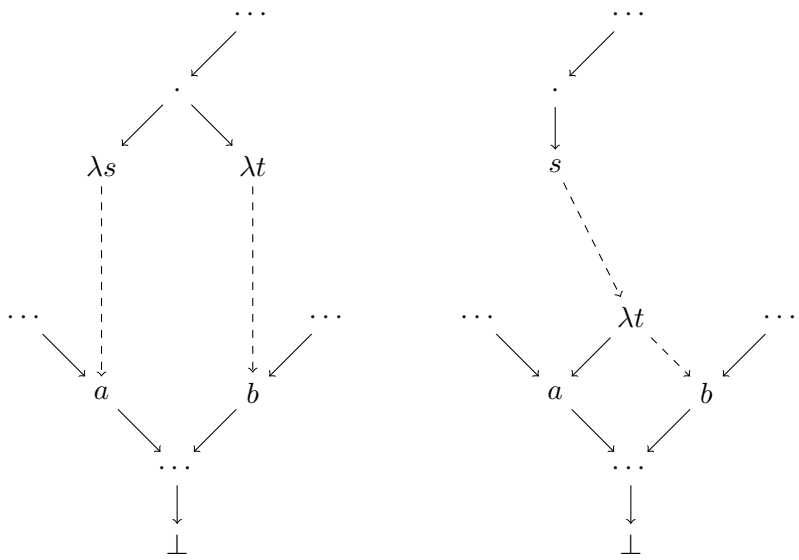

Figure 12: Application of the *BETA*-rule: $\lambda s[a, \cdots]\lambda t[b, \cdots] \succ_{\mathsf{LC}} s[\lambda t[b, \cdots], a, \cdots]$

5.3 Main result

We are finally able to show that the number of beta steps is a reasonable time measure for L:

Theorem 27 (Polynomial simulation) Setting $c := 1$, Turing Machines and L can simulate each other with polynomial overhead. More precisely:

- A Turing Machine can compute the implicit representation of the normal form of a term s w.r.t. $\lfloor \cdot \rfloor$ in time $p(\mathbb{T}_c(s))$ for al polynomial p.
- A lamda term can simulate a Turing Machine M in polynomial time w.r.t. $\mathbb{T}_c$. Hereby we rely on the encoding used by Dal Lago and Martini in [3].

Proof The first direction was shown in Section 5.1 and 5.2. For the second direction we can apply the proof of Dal Lago and Martini (See [3], Theorem 1), since their time measure is more expensive than counting the beta steps. ∎

5.4 Classical Complexity Theory

We will now go one step further by investigating lamda terms that decide languages in the complexity class P. These procedures will - after applied to the encoding of a string - converge after polynomial many steps in either the encoding of $\top$ or $\bot$. Therefore a Turing Machine can even produce the **explicit** representation of the normal forms. At first we introduce a notion of classical complexity w.r.t. L:

Definition 28 Let Σ be a fixed alphabet of size c_Σ and let $\top, \bot \in \Sigma$. Furthermore, let $\lceil \cdot \rceil$ be the encoding function of a string as used by Dal Lago and Martini in [3].

- Let $f : \Sigma^* \to \Sigma^*$ be a (possibly partial) function. A closed term λs **computes** f, iff
$$\forall t, u \in \Sigma^* : (\lambda s \lceil t \rceil) \succ^* \lceil u \rceil \Leftrightarrow f(t) = u.$$
We say that λs is an **implementation** of f, denoted by $\lambda s \downarrow f$.
- The **running time** $\mathbb{T}_{\lambda s} : \mathbb{N} \to \mathbb{N}$ of an implementation λs of a total function is given by
$$\mathbb{T}_{\lambda s}(n) := \max\{\mathbb{T}_1(\lambda s \lceil t \rceil) \mid t \in \Sigma^n\}$$
If $\mathbb{T}_{\lambda s}$ is bounded by a polynomial, λs is called **polynomial time bounded**.
- A subset $L \subseteq \Sigma^*$ is called a **language** w.r.t. Σ.
- Given a language L, the characteristic function $\chi_L : \Sigma^* \to \Sigma^*$ is defined as
$$\chi_L(t) := \begin{cases} \top & t \in L \\ \bot & t \notin L \end{cases}$$
- A closed term λs **decides** a language L iff it computes χ_L. We write $\lambda s \downarrow L$.

- The class P_L is defined as the set of all languages that can be decided by a polynomial time-bounded lamda term.

From the polynomial simulation theorem it follows that $\mathsf{P}_\mathsf{L} = \mathsf{P}$. In the next section, we will show that lamda terms that are implementations of total functions have a surprising property.

5.4.1 Term size optimization

As stated in the motivation the general problem of simulating a reduction in L is that the size of a term might blow up exponentially in the number of the reduction steps. We will now show that every λs, such that $\lambda s \downarrow L$ for $L \in \mathsf{P}$ can be transformed into a function that only produces polynomial size bounded terms. In the following, let Σ be a fixed but finite alphabet.

Definition 29 We write $\lambda s \simeq_\Sigma \lambda s'$ if λs and $\lambda s'$ are implementations of the same function $f : \Sigma^* \to \Sigma^*$

Definition 30 Let λs be an implementation of a function $f : \Sigma^* \to \Sigma^*$. Furthermore let $\ell : \mathbb{N} \to \mathbb{N}$. λs is called ℓ **term size-bounded**, denoted by $s \lesssim \ell$, if the following holds for every $t \in \Sigma^*$:

$$\forall u : s\lceil t \rceil \succ^* u \to |u| \leq \ell(|s\lceil t \rceil|)$$

Note that this **is not meant to be** an approach for a valid space measure on L. The definition simply catches how large a term can grow w.r.t. the input size. As stated above, it could be possible, that there are polynomial time bounded lamda terms which are not polynomial term-size bounded. This fact allone shows that a term size bound is not a valid space measure, since one would expect that a reasonable machine does not need (much) more space than time. For example, it is a well-known fact, that a Turing Machine does not need more space than time for every computation, since writing on tapes also costs time.
However, we can show that, among other classes, P can be expressed by *economic* implementations:

Definition 31 Let λs be an implementation of a total function $f : \Sigma^* \to \Sigma^*$. λs is called **economic** if there is a polynomial p such that $s \lesssim p \circ \mathbb{T}_{\lambda s}$.

Informally, an implementation is economic if the size of the terms during any reduction secquence does not explode w.r.t. to the running time and the input size.

Theorem 32 (Constructive optimization) Let λs be an implementation of a total function $f : \Sigma^* \to \Sigma^*$. Then one can construct an implementation $\lambda s'$ such that $\lambda s \simeq \lambda s'$ and $\lambda s'$ is economic. Furthermore the running time of $\lambda s'$ is bounded by a polynomial in the running time of λs.

Proof Given λs, there is a polynomial p such that one can construct a Turing Machine M that computes f with running time $p \circ \mathbb{T}_{\lambda s}$, using the results of the previous sections. Next we construct $\lambda s'$ as in Theorem 1 in [3]. Recall Dal Lagos and Martinis cost-function for beta steps:

$$c(\lambda s \lambda t) := \max\{1, |s^0_{\lambda t}| - |\lambda s \lambda t|\}$$

This means, that a super polynomial blow up of the term size during the reduction would require super polynomial time w.r.t. to c. However, Dal Lago and Martini proved that the required time is bounded by a polynomial. Therefore we can use uniform confluence w.r.t. to c to conclude that no reduction requires super polynomial space. Note that uniform confluence w.r.t. to c is proven in [3]. Furthermore it follows by Corollary 12. Last but not least c is more expensive than the function that is constant 1. Therefore the running time of $\lambda s'$ is bounded by a polynomial in the running time of λs. ∎

6 Conclusion and open questions

At this point, we have obtained strong evidence that the weak call-by-value lambda calculus is a valid machine model, at least regarding the weak invariance thesis. Furthermore the time measure we established, namely the number of the reduction steps is somewhat more natural than the measure in [3]. Comparing this to the result for the full lambda calculus (see [1]) one might wonder if it is necessary to consider the weak call-by-value lamda calculus at all. However, there are at least two advantages in doing so: First, the weak call-by-value lambda calculus is uniform confluent, i.e. one does not have to consider a derivation strategy to argue about time and second, it is more realistic, since it describes functional programming languages. Although we provided evidence for using L as model to formalize complexity theory, there is still much work to do:

- As we have seen, a normalizing reduction $s \succ^* t$ in L can efficiently be simulated by LC, but only with a non-explicit representation of t. However, if we consider decision procedures and reductions with polynomial sized output, the corresponding normalforms can be explicitely represented. This means, that we can forget about LC, if we deal with robust time complexity classes that contain P. Therefore, the next task would be starting to try to formalize other robust complexity classes like NP, PH and/or EXP as well as polynomial time reductions and perhaps the first hardness results.
- Furthermore we conjecture that it should be possible to use the size of the substitution trees (see Section 5.2) to obtain a uniform space measure and to eventually prove that the weak call-by-value lambda calculus even fulfills the strong invariance thesis.

References

[1] Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 8. ACM, 2014.

[2] Peter Van Emde Boas. Machine models and simulations. *Handbook of Theoretical Computer Science, volume A*, pages 1–66, 2014.

[3] Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theoretical Computer Science*, 398(1):32–50, 2008.

[4] Yannick Forster. A formal and constructive theory of computation. 2014.

[5] Fabian Kunze. To be renamed. 2015.