α-Avoidance

Samuel Frontull
Universität Innsbruck, Austria

Georg Moser
Universität Innsbruck, Austria

Vincent van Oostrom
Barendrecht, The Netherlands

Abstract

When substitutions and bindings interact, there is a risk of undesired side effects if the substitution is applied naïvely. The λ-calculus captures this phenomenon concretely, as β-reduction may require the renaming of bound variables to avoid variable capture. In this paper we introduce α-paths as an estimation for α-avoidance, roughly expressing that α-conversions are not required to prevent variable capture. These paths provide a novel method to analyse and predict the potential need for α in different calculi. In particular, we show how α-path characterises α-avoidance for several sub-calculi of the λ-calculus like (i) developments, (ii) affine/linear λ-calculi, (iii) the weak λ-calculus, (iv) μ-unfolding and (iv) finally the safe λ-calculus. Furthermore, we study the unavoidability of α-conversions in untyped and simply-typed λ-calculi and prove undecidability of the need of α-conversions for (leftmost–outermost reductions) in the untyped λ-calculus. To ease the work with α-paths, we have implemented the method and the tool is publicly available.

2012 ACM Subject Classification Theory of computation → Program analysis

Keywords and phrases λ-calculus, variable capture, α-conversion, developments, safe λ-calculus, undecidability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.22

Supplementary Material *Software (Alpha Avoidance Tool)*: `https://tcs-informatik.uibk.ac.at/tools/alpha/` archived at `swh:1:dir:e2731de499d52916453f9ac5d90cf9db879e26c5`

1 Introduction

Substitution is a fundamental concept in computer science. It is, for example, a core operation for computation in the λ-calculus, applied by compilers to optimise programs and, in general, key for reasoning with logical expressions. As is well-known, undesired side effects may arise when substitution and bindings interact, if the substitution is naïvely applied. Here, we study substitution and in particular the need for α-conversion in the context of the λ-calculus. We emphasise, however, that the same idea could be applied in different fields.

In the λ-calculus, the contraction of a redex by means of naïve substitution may cause *variable capture* where a variable originally occurring free ends up being bound in the resulting term due to a name collision. Variable captures may lead to inconsistent results and invalidate the confluence property. Such fallacies have occurred already quite early in the literature, for example in work from the 1940s by Newman [29]. As discovered by Schroer, and as presented by Rosser in his review [33], Newman's proof that the projection axioms were satisfied for the λI-calculus was erroneous. The purported proof contained an α-problem; cf. [30], [29, Remark 6.14(ii)] and [13, Sect. 5.2].[1]

[1] As far as we know this problem is the α-α-problem, that is, this is the first α-problem in the literature on the λ-calculus.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 22; pp. 22:1–22:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Since the specific variable names are actually irrelevant (cf. [12]), the result of an evaluation should also not be influenced by the specific names. An option is to work with some kind of unique representatives of α-equivalence classes of λ-term, e.g. with De Bruijn's λ-terms with nameless dummies [12] (see below for more). Though that certainly is a possibility, here we stick to Church's original proposal and work with explicit α-conversion steps, enabling to state the main questions addressed in this paper: can α-conversion steps be avoided for a λ-term M, by suitably α-converting it up front, say to a term M' such that no α-conversion step needs to be invoked along any reduction from M'. Such a characterisation should allow for a more efficient reduction, based on naïve substitutions, that applies α-conversion (if at all) only on the initial term. In the sequel, by "avoiding α" we mean that we can α-convert a λ-term M to some λ-term M' so that subsequent α-conversions are not needed in any computation from M'.

Before proceeding let us relate the question addressed here to the so-called Variable Convention [6] stating that variables may be assumed to be suitably renamed apart in *a given context*. On the one hand, this convention has been widely adopted in the literature. On the other hand, examples as in Figure 1, where renaming apart in the initial term does not suffice, abound. From that perspective our investigation addresses the question in *what contexts* exactly does the Variable Convention work?

In the examples presented in Figure 1, α cannot be avoided, no matter how the variables are (re)named initially. Without the explicit α-conversion steps and substituting naïvely, would lead to variable capture and give rise to the λ-terms $\lambda zz.z\,z$ and $\lambda cxy.c\,x\,x$ respectively, which do not have the intended semantics. (Hence omitting the α-conversion steps would break the Church–Rosser property.) Note that though the example on the left in Figure 1 cannot be (simply) typed, the example on the right can, showing that type regimes in general do not guarantee that α can be avoided.

Contributions

- As already indicated in Figure 1, α-conversion may be unavoidable in the (untyped) λ-calculus. This motivates the question about the algorithmic feasibility of the problem. We establish that (for leftmost–outermost reductions) the problem is undecidable.
- We present a *sound* characterisation for α-avoidance via α*-paths*. *A*-paths give a novel perspective on α; they can be utilised as a tool to predict for a given λ-term M the *potential need* for α-conversion, i.e. the need for α-conversion in *any* step $N \to_\beta L$ after *any* β-reduction $M \to_\beta^* N$. To that end, α-paths combine two known ideas.
  Foremost, α-paths build on the notion of *legal* path, cf. [3], characterising the so-called *virtual* redexes of a term M, where a virtual redex of M is a redex that can arise in *any* term N along *any* reduction $M \to_\beta^* N$. Legal paths arose from Girard's geometry of interaction; see [2] for various characterisations of them attesting to their canonicity. The intuition for them employed here, is that a redex R in N is represented in the *graph* of N by a single-edge-*path* from an application node to an abstraction node, and that *pulling that path back* along the reduction $M \to_\beta^* N$ gives rise to a path in M, the legal path representing the redex R in N as a virtual redex in M.
  The intuition then is that α-conversion is *potentially needed* in M when there is a *virtual* redex in M, that is, a redex in N, whose contraction *needs* α-conversion. Since also the need for α-conversion can be characterised by means of paths, namely by so-called *binding-capturing chains* [17, 7], we arrive at our notion of α-path, combining legal paths with binding-capturing chains.
- To ease work with α-paths, we have implemented the method; the tool is publicly available.

$$\begin{array}{ll}
 & \underline{(\lambda x.x\,x)\,(\lambda yz.y\,z)}_{\mathsf{A}} \\
\to_\beta & \underline{(\lambda yz.yz)\,(\lambda yz.y\,z)}_{\mathsf{B}} \\
\to_\beta & \lambda z.(\lambda yz.y\,z)\,z \\
\to_\alpha & \lambda z.\underline{(\lambda yz'.y\,z')\,z}_{\mathsf{C}} \\
\to_\beta & \lambda zz'.z\,z'
\end{array}$$

$$\begin{array}{ll}
 & \underline{(\lambda fc.f\,(f\,c))\,(\lambda zxy.z\,y\,x)}_{\mathsf{A}} \\
\to_\beta & \lambda c.(\lambda zxy.z\,y\,x)\,\underline{((\lambda zxy.z\,y\,x)\,c)} \\
\to_\beta & \lambda c.\underline{(\lambda zxy.z\,y\,x)\,(\lambda xy.c\,y\,x)}_{\mathsf{B}} \\
\to_\beta & \lambda c.(\lambda xy.(\lambda xy.c\,y\,x)\,y\,x) \\
\to_\alpha & \lambda c.(\lambda xy.\underline{(\lambda xy'.c\,y'\,x)\,y}_{\mathsf{C}}\,x) \\
\to_\beta & \lambda cxy.\underline{(\lambda y'.c\,y'\,y)\,x} \\
\to_\beta & \lambda cxy.c\,x\,y
\end{array}$$

A...duplication B...redex creation C...open redex contraction

Figure 1 α is needed in the λ-calculus.

- We exemplify the versatility of α-paths through various important sub-calculi of the λ-calculus, listed below. The first three calculi arise from a careful analysis of the canonical example illustrating why α-conversion is unavoidable in the λ-calculus, the λ-term $(\lambda x.x\,x)\,(\lambda yz.y\,z)$. As illustrated in Figure 1, its reduction to normal form comprises first a *duplicating* step A (the subterm $\lambda yz.y\,z$ is duplicated), then a *creating* step B (creating the redex $(\lambda yz.y\,z)\,z$), and finally a *non-closed* step C (contracting an *open* redex $(\lambda yz.y\,z)\,z$, containing the free variable z bound outside). Somewhat surprisingly, forbidding *one* of these three types of steps suffices for α-avoidance.

1. *Developments* [14] are reductions in which no *created* redexes are contracted. Stated differently, in a development from M only *residuals* of redexes in M (no virtual redexes) are contracted. Intuitively, α can then be avoided since all residuals of a given redex are disjoint along a development.
   Developments [14, 6] feature prominently in the λ-calculus since its inception. They form the basis for the proof that β-reduction has the Church–Rosser property, more precisely, that parallel β-reduction has the diamond property and satisfies the cube law, using that all developments are finite (unlike arbitrary reductions).
2. The *linear* (*affine*) λ-calculus [21] forbids *duplication*. That is achieved formally by restricting term-formation, requiring the variable x to occur free exactly (at most) once in M in an abstraction term $\lambda x.M$. Intuitively, α can then be avoided since variables persist linearly along reductions.
   Though the linear λ-calculus [21, 23, 27, 38] had been studied before, it rose to prominence after the introduction of linear logic, via the connection between linear λ-terms and MLL-proofnets, with abstractions and applications corresponding to pars and tensors. Linearity affords nice properties, e.g. termination and simple typability.
3. The *weak* λ-calculus [39] forbids to contract *open* redexes. Intuitively, α is then avoided since when substituting by closed terms only, there's no risk of variable capture either.
   Weak reduction [37, 31, 1, 39, 8, 5] forms the basis of functional programming languages such as Haskell that evaluate to weak head normal form. Indeed, the fact that α-conversion can then be avoided was stated as an explicit motivation in [31], and makes that functional programs can be represented as orthogonal term rewrite systems and weak reduction can be optimally implemented via Wadsworth's graph reduction. (Weak reduction is often paraphrased as "no reduction under a λ", but that restriction is undesirable as it breaks the Church–Rosser property.)

These three examples are mainly methodological, since the fact *that* α can be avoided for them is well-known. We also present two important but less well-known examples.

4. The *modal μ-calculus* [25] has unfolding rules for least (μ) and greatest (ν) fixed-points in its formulas. Intuitively, α can then be avoided for the same reason it can for developments, namely that unfolding does not *create* new redexes [17, 7]. Here we show

that this can be obtained via α-paths, under a standard embedding of modal μ-formulas in the λ-calculus, representing unfolding using Turing's fixed-point combinator.
Though the literature on the modal μ-calculus is rich, only recently issues related to α-conversion seem to have been addressed [26]. The main point of this example is to suggest that the technology developed here for avoiding α in the λ-calculus, can be transferred to other calculi with binding, in mathematics, logic, programming languages, linguistics, music theory, etc..

5. In the *safe λ-calculus* [10, 9, 11] the occurrences of (free) variables are restricted according to the order of their type. Intuitively, this restriction on the order of the types of the variables can be transferred to the variables, guaranteeing that α can be avoided. (Note that as observed above, typing disciplines, say simple typing, in general do *not* suffice to be able to avoid α.)
Analysing the safe λ-calculus as presented in [9, 11] using our tools, we found that the claim that α could be avoided in it, was not entirely correct, provoking the further analysis, and a proposal for slight amendations, presented below.

Related Work

In the classical literature on the λ-calculus the focus was not on α-conversion. However, when the λ-calculus started being used as a tool, α-conversion had to be addressed. We briefly discuss two important strands of research. One approach is to abstract α away and to exclusively work with (representatives of) α-equivalence classes of λ-terms.[2] De Bruijn's lambda notation with nameless dummies [12] is widely adopted in implementations. This typically side-steps the issue but does not resolve it: the cost of α is now inextricably hidden in the cost of β, and α-conversion disappears in the notation with nameless dummies only to resurface in the form of reindexing. Moreover, any such representation runs the risk of creating a gap between the theory in the literature and the representation.[3] Another approach is to bring α-conversion about in another way. The *nominal* approach [19] is a prominent exponent of this, recasting the notion of a variable being bound via the dual notion of a variable being free for, allowing to recast α-conversion via the classical notion of *permutation*. We stress that α-conversion resurfaces in this setting, but unlike the modulo-approach now in an explicit form as in our case, making it interesting to study our question for it (and then compare both). We leave that to further research.

Finally, we mention that several other decision problems about α have been considered by Statman, which were reported in [35]. This work is based on [18].

Outline

This paper is structured as follows. In the next section, we recall fundamental concepts and notions. In Section 3, we motivate the definition of α-paths and provide a syntactic proof that developments can avoid α by using of a restricted form of α-paths. The latter are generalised in Section 4, where we establish the main contribution of this work, a sound characterisation of α-avoidance via α-paths. Section 5 applies this characterisation to affine, weak and the safe λ-calculus. Finally, we conclude in Section 6.

[2] Higher-Order Abstract Syntax goes one (big) step further by working with simply typed $\alpha\beta\eta$-equivalence classes of terms.

[3] The same holds for programming; everyone will have encountered inscrutable error-messages on De Bruijn-indices representing variables.

Table 1 Capture-avoiding and capture-permitting substitution.

M	$[\![x := N]\!]$ (capture-avoiding)	$[x := N]$ (capture-permitting)
x	N	N
y	y	y
$e_1\,e_2$	$e_1[\![x := N]\!]\,e_2[\![x := N]\!]$	$e_1[x := N]\,e_2[x := N]$
$\lambda x.e$	$\lambda x.e$	$\lambda x.e$
$\lambda y.e$	$\lambda y.e[\![x := N]\!]$ if $y \notin \mathcal{FV}(N)$	$\lambda y.e[x := N]$
	$\lambda z.e[\![y := z]\!][\![x := N]\!]$ else with z fresh for e and N.	

2 Preliminaries

We assume acquaintance with the standard definitions of the λ-calculus, cf. [6], but recall relevant concepts and notations. We use $=$ to denote syntactic equality of λ-terms, and $\equiv_\alpha$ for equality modulo α. We write $\mathcal{FV}(M)$ for the set of free variables in a λ-term M and $\mathcal{BV}(M)$ for the set of bound variables. We distinguish between a capture-*avoiding* and a capture-*permitting* substitution, cf. Table 1. The capture-avoiding substitution, denoted as $M[\![x := N]\!]$, deals with a potential variable capture, whereas the capture-permitting substitution, denoted as $M[x := N]$, naïvely substitutes. If $M[\![x := N]\!] \equiv_\alpha M[x := N]$ then we say that the substitution of N for x in M is *α-free*. The single-step β-reduction contracting a redex $(\lambda x.M)\,N$ in some arbitrary context, is said to be *α-free*, if the applied substitution is α-free.

▶ **Definition 1.** *A reduction sequence starting from a λ-term M is said to be* α-free, *if each β-reduction step is α-free. A λ-term M has α-free simulations, if there exists an α-equivalent λ-term N such that every reduction sequence starting from N is α-free. In such case we say that N* avoids α*. We say that we can avoid α in a calculus, if every term in this calculus has* α-free simulations*.*

The reduction sequence illustrated in Figure 1 is not α-free. The λ-term $(\lambda x.x\,x)\,(\lambda yz.y\,z)$ does not have α-free simulations, which shows that α cannot be avoided in the pure λ-calculus. The λ-term $(\lambda fx.f\,(f\,x))\,(\lambda fx.f\,(f\,(f\,x)))$, denoting the exponentiation 3^2 via Church numerals, has α-free simulations as the α-equivalent λ-term $(\lambda fy.f\,(f\,y))\,(\lambda fx.f\,(f\,(f\,x)))$ avoids α. (This can also be checked with our tool, see Listing 1 in Section 4 below).

The *position* in a λ-term is a finite sequence of 1s and 2s. The set of positions of a λ-term M is denoted as $\mathcal{Pos}(M)$. We write $M|_p$ for the subterm at position p in M and $M(p)$ for the *symbol* at position p (the head-symbol of $M|_p$), where $M(p) \in \{x, @, \lambda x\}$ for some x. In the following we may write s^p when we depict a specific symbol s of a λ-term M at position p, $s = M(p)$, whenever both the position and the symbol are of interest.

$$M|_p := \begin{cases} M & \text{if } p = \epsilon \\ N|_{p'} & \text{if } M = \lambda x.N \text{ and } p = p'1 \\ N_i|_{p'} & \text{if } M = N_1\,N_2 \text{ and } p = p'i \end{cases}$$

A position p is a *prefix* of a position q, if $q = pq'$ for some position q'. We use the notation $p \preceq q$ to denote that p is a prefix of q and $p \prec q$ to denote that p is a *strict* prefix of q (q' is non-empty). Two positions p, q are said to be *parallel*, denoted by $p \parallel q$, if $p \npreceq q \wedge q \npreceq p$. A position p is said to be *left* of a position q, written as $p \parallel_l q$, if $p = s \cdot 1 \cdot p'$ and $q = s \cdot 2 \cdot q'$. We define the trace relation ▶ to be the relation between positions in the source and in the target of a β-step $s \to_\beta t$ contracting a redex at position o (cf. [36, Section 8.6.1]):

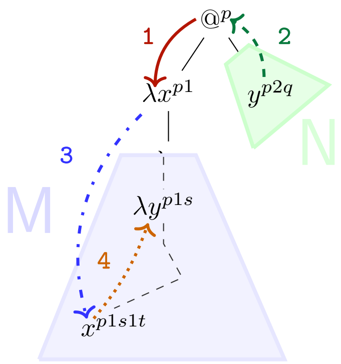


Figure 2 Substitution dynamics leading to variable capture.

- (context) $p \blacktriangleright p$ if o is not prefix of p,
- (body) $o1 1p \blacktriangleright op$ if $p \neq \epsilon$ and $p \neq q$,
- (arg) $o2p \blacktriangleright oqp$ for all positions q, such that $o11q$ is bound by $o1$.

A redex in a term t at position q is called a *residual* of a redex in some origin s ($s \to_\beta \ldots \to_\beta t$), if $p \blacktriangleright \ldots \blacktriangleright q$ and $s|_p$ is a redex (cf. [15, Chapter 4, Section 4]).

A path $\sigma = (p_1, p_2, \ldots, p_n)$ in a λ-term M is a sequence of positions in $\mathcal{P}os(M)$. The length $|\sigma|$ of a path σ is the number of positions minus 1. An *edge* is a path of length 1.

The reversal of a path σ is denoted by $(\sigma)^r$. Two paths $\sigma = (p_1, p_2, \ldots, p_n)$ and $\psi = (q_1, q_2, \ldots, q_n)$ are said to be *composable*, if $p_n = q_1$. We write $\sigma \cdot \psi$ to denote the composition of two (composable) paths σ, ψ resulting in $(p_1, p_2, \ldots, p_n, q_2, \ldots, q_n)$.

A path in M starting at position p and ending at position q is of type:

1) `@-v`, if $M(p) = @$ and $M(q) = x$ for some x.
2) `@-`λ, if $M(p) = @$ and $M(q) = \lambda x$ for some x.
3) `@-@`, if $M(p) = @$ and $M(q) = @$.
4) `v-v`, if $M(p) = x$ and $M(q) = y$ for some x, y.
5) `v-`λ, if $M(p) = x$ and $M(q) = \lambda y$ for some x, y.

To illustrate, let $M = (\lambda x.x\, x)\, (\lambda yz.y\, z)$. $\sigma = (\epsilon, 2, 2112)$ is a `@-v`-path in M with $|\sigma| = 2$. σ and $(\sigma)^r$ are composable and the path $(\epsilon, 2, 2112, 2, \epsilon)$ resulting from their composition $\sigma \cdot (\sigma)^r$ is of type `@-@`.

3 Developments Are α-Avoiding

Recall that reductions of residuals, also known as *developments*, are finite. This was proved already in 1936 by Church–Rosser for the λI-calculus [14] and then generalised to the full λ-calculus by Schroer [34] and independently by Hindley [20]. It is well known that in *finite developments* α-renaming can be avoided, cf. [24]. Intuitively, this is due to the fact that in developments the residuals of a redex-pattern remain disjoint [22]. Thus, if all binders are initially properly renamed apart, α can be avoided. To prepare the ground for our main contribution – α-paths – we sketch a purely syntactic proof of this result in this section.

We start by giving an intuition for how a capturing-potential can be characterised by paths. A naïve substitution leads to a variable capture whenever we

(i) naïvely contract a redex $(\lambda x.M)\, N$ where
(ii) some variable y occurring free in N
(iii) is moved into M, where some x is free in M
(iv) is in the scope of a λy.

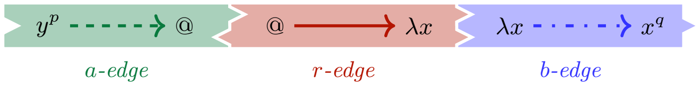
Each of these conditions can be represented via edges in the abstract syntax tree (AST), as formalised below and illustrated in Figure 2 for the redex $(\lambda x.M)\,N$. More precisely, we have an a-edge $(p2q, p)$, an r-edge $(p, p1)$, a b-edge $(p1, p1s1t)$ and a c-edge $(p1s1t, p1s)$.

Let M be a λ-term. We conceive the AST of M as a graph and define four additional types of *edges* for M:

1. (r-edge $\longrightarrow$) A *r*edex-edge $(p, p1)$ connects an `@`-node at position p to its left son at position $p1$, if $M(p1) = \lambda x$ for some x.
2. (a-edge $\dashrightarrow$) An *a*pplication-edge $(p2q, p)$ connects a variable x at position $p2q$ to an `@`-node at position p, if x is free in $M|_{p2}$.
3. (b-edge $\dashrightarrow$) A *b*inding-edge $(p, p1q)$ connects a λx at position p to a variable y at position $p1q$, if $x = y$ and y is free in $M|_{p1}$.
4. (c-edge $\cdots\rightarrow$) A *c*apturing-edge $(p1q, p)$ connects a variable y at position $p1q$ to a λx at position p to, if $x \neq y$ and y is free in $M|_{p1}$.

We add the a-, r-, b- and c-edges as actual edges to the graph of M in the standard way. We call such a graph the *α-graph* of a λ-term M, denoted as $G_\alpha(M)$. From the definition of an r-edge, we immediately obtain that for any r-edge in $G_\alpha(M)$ with the source at position p, $M|_p$ is a redex.

▶ **Definition 2.** *Let M be a λ-term, a an* a*-edge, r an* r*-edge and b a* b*-edge in $G_\alpha(M)$ with a, r and r, b composable. We call the* v*–*v*-path $\sigma_{arb} = a \cdot r \cdot b$ an* arb*-path of M.*

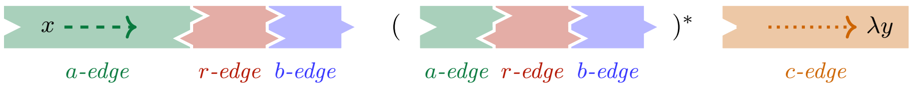


Let p be the position of the starting `v`-node y and q the position of the ending `v`-node of an *arb*-path φ. Then we have $q \parallel_l p$.

The example term from Figure 3 illustrates an example where an outermost reduction strategy needs α in the second reduction step. To characterise the need for α after the contraction of one or multiple redexes, *arbic*-paths are introduced next.

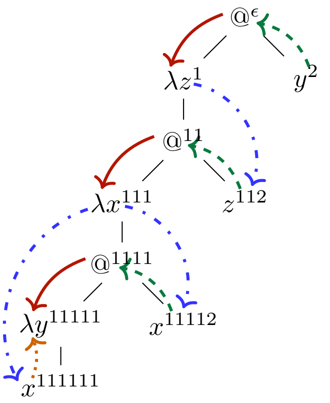
▶ **Definition 3.** *The set of* arbic*-paths of a λ-term M is inductively defined as follows.*

- *(base case) Let σ_{arb} be an* arb*-path of M and c a* c*-edge in $G_\alpha(M)$ with σ_{arb}, c composable. Then the* v*–λ–path $\sigma_{arb} \cdot c$ is an* arbic*-path of M.*
- *(*arb*-composition) Let σ_{arb} be an* arb*-path and ψ an* arbic*-path of M with σ_{arb}, ψ composable. Then the* v*–λ–path $\sigma_{arb} \cdot \psi$ is an* arbic*-path of M.*



From Definition 3 we see that *arbic*-paths are non-empty sequences of *arb*-paths followed by a c-edge $(\sigma_{arb}^{+} \cdot c)$. As already remarked, an *arb*-path connects the occurrence of a variable to the occurrence of another variable at its left. By consequence arbic-paths are acyclic and the set of *arbic*-paths of a λ-term M is finite. The paths $\sigma_0 = 112 \to 11 \to 111 \to 111111 \to 11111$ and $\sigma_1 = 2 \to \epsilon \to 1 \to \sigma_0$ are *arbic*-paths for the λ-term illustrated in Figure 3. Specialising *arbic*-paths such that the names of the initial variable and of the final abstraction are equal, we obtain a characterisation of the need for α in some reduction sequence.

▶ **Definition 4** (*arbic α-path*)**.** *Let M be a λ-term and ψ an* arbic*-path of M. If ψ starts with a variable x and ends with a λ-node λy where $x = y$, then ψ is called an arbic α-path.*



leftmost–outermost reduction

$\underline{(\lambda z.(\lambda x.(\lambda y.x)\,x)\,z)\,y}$
$\to_\beta$ $\underline{(\lambda x.(\lambda y.x)\,x)\,y}$
$\to_\beta$ $\underline{(\lambda y'.y)\,x}$
$\to_\beta$ y

leftmost–innermost reduction

$(\lambda z.(\lambda x.\underline{(\lambda y.x)\,x})\,z)\,y$
$\to_\beta$ $(\lambda z.\underline{(\lambda x.x)\,z})\,y$
$\to_\beta$ $\underline{(\lambda z.z)\,y}$
$\to_\beta$ y

Figure 3 Leftmost–outermost needs α.

The path σ_1 as defined above is an arbic α-path for the λ-term illustrated in Figure 3. Now, essentially by construction, we can see that if there is no arbic α-path in $G_\alpha((\lambda x.M)\,N)$ starting at a free variable in N and ending in M, then $M[\![x := N]\!] \equiv_\alpha M[x := N]$. We emphasise, that we only claim α-equivalence and not syntactic equivalence $(=)$ of $M[\![x := N]\!]$ and $M[x := N]$. To clarify, let $(\lambda x.M)\,N$ be a redex with $M = \lambda y.y$ and $N = y$. Then $M[\![x := N]\!] = \lambda z.z$ and $M[x := N] = \lambda y.y$. We have $M[\![x := N]\!] \equiv_\alpha M[x := N]$, but $M[\![x := N]\!] \neq M[x := N]$. Hence, α-equivalence is the strongest property that we can conclude.[4]

▶ **Lemma 5.** *Let $s \to_\beta t$. If $G_\alpha(s)$ contains no arbic α-path, then $G_\alpha(t)$, where the set of* r*-edges is restricted to those denoting residuals, also does not.*

Proof. We write $\langle G_\alpha(t)\rangle$ for the sub-graph of $G_\alpha(t)$, where the set of r-edges is restricted to those denoting residuals of s. Since there are no arbic α-paths in $G_\alpha(s)$, the β-step can be performed by means of capture-permitting substitution (no variable capture). We have $s = C[(\lambda x.M)\,N]$ and $t \equiv_\alpha C[M[x := N]]$ for some context C, body M and argument N, with $(\lambda x.M)\,N$ being the contracted redex at position o. We prove the lemma by relating the edges in $\langle G_\alpha(t)\rangle$ to edges and paths in $G_\alpha(s)$ and making a distinction according to the components as they appear in the source and the target. As done in [17], we use primed variables (p', q') to range over positions in the target term t, indicating the positions they trace back to in the source term s, by unpriming (p, q).

Consider an a- or a c-edge from p' to q' in $\langle G_\alpha(t)\rangle$ where p' denotes the position of a variable y and q' the position of an application (in the case of an a-edge) or an abstraction (in the case of a c-edge). We have $q' \prec p'$ and the variable y at $t(p')$ occurs free in $t|_{q'}$. We distinguish the following cases:

- p', q' in the same component: we have the same edge from p to q in $G_\alpha(s)$.
- q' in the context and p' in the body: then $x \neq y$ (otherwise the y would have been replaced by N) and we have the same edge in $G_\alpha(s)$ with 11 inserted at o.
- q' in the context and p' in the argument: there is no variable capture so $s(p)$ must occur free in $s|_q$. Therefore, we have the same edge from p to q in $G_\alpha(s)$.
- q' in the body and p' in the argument: the origin of the a-edge/c-edge is an arb-path from p to qq_1, for some q_1, followed by an a-edge/c-edge from qq_1 to q in $G_\alpha(s)$.

[4] We stick to the standard definition of substitution $M[\![x := N]\!]$, which renames even if the variable x to be replaced does not occur in the body M [6]. We note that, if we were to adapt the substitution so that it is not applied when the argument is erased ($x \notin \mathcal{FV}(M)$), then we could claim syntactic equivalence.

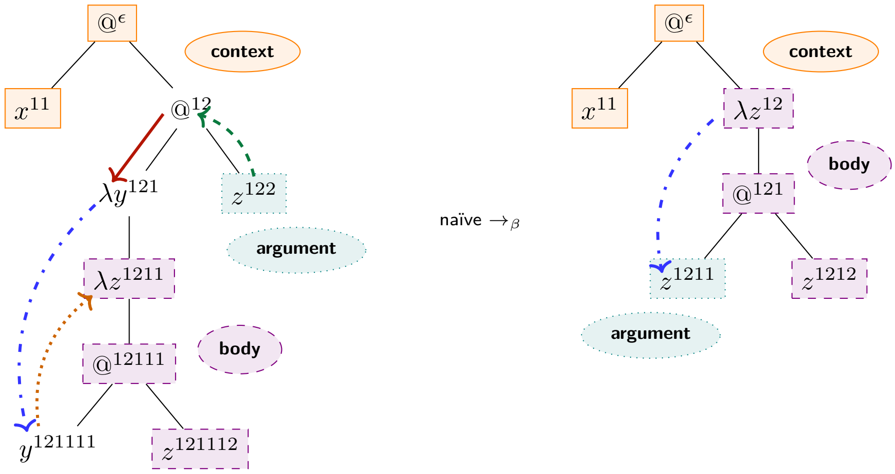


Figure 4 A b-edge that traces back to an arbic α-path.

Given a b-edge from q' to p' in $\langle G_\alpha(t)\rangle$. p' denotes the position of the bound variable y, q' the position of the binder λy. We have $q' \prec p'$ and distinguish following cases:

- p', q' in the same component: then we have a b-edge from q to p in $G_\alpha(s)$.
- q' in the context and p' in the body: we have $x \neq y$ and a b-edge in $G_\alpha(s)$ with 11 inserted at o.
- q' in the context and p' in the argument: there is no variable capture so $s(p)$ must occur free in $s|_q$. Therefore, we have a b-edge from q to p in $G_\alpha(s)$.
- q' in the body and p' in the argument: such a b-edge would map back to an arbic α-path from p to q in $G_\alpha(s)$, which is excluded by the assumption (Figure 4 illustrates an example).

For the r-edges $(p', p'1)$ in $\langle G_\alpha(t)\rangle$ we make the following case distinction:

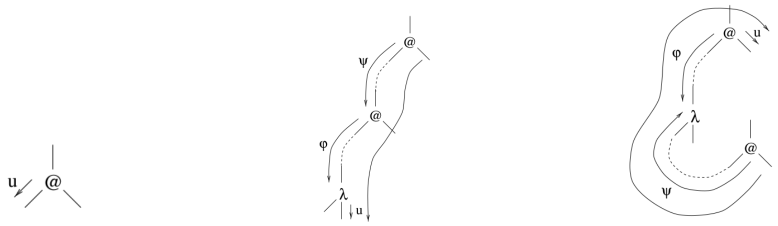
- p' and q' are in the same component: then we have an r-edge from p to q in $G_\alpha(s)$.
- in all other cases: such an r-edge would denote a created redex in t. We have no such r-edge in $\langle G_\alpha(t)\rangle$.

We have seen that a r-edges and b-edges in $\langle G_\alpha(t)\rangle$ map back to an edge of the same type in $G_\alpha(s)$. a-edges and c-edges map back to a path of shape $\sigma^*_{arb} \cdot e$, where e denotes an edge of the same type and σ_{arb} an arb-path in $G_\alpha(s)$. An arbic α-path in $G_\alpha(t)$ has the following shape $(a'_1, r'_1, b'_1, \ldots, a'_n, r'_n, b'_n, c')$, where x_i denotes an x-edge (p_i, q_i). If we replace the edges in this path by the edges and paths they map back to, we get a path of the shape $(\sigma^*_{arb_1} \cdot a_1, r_1, b_1, \ldots, \sigma^*_{arb_n} \cdot a_n, r_n, b_n, \sigma^*_{arb} \cdot c)$, which would be an arbic α-path in $G_\alpha(s)$. ◀

Based on the lemma, we obtain the characterisation of α-freeness via arbic α-paths. Let M be a λ-term. If M contains no arbic α-path, then every development from M is α-free. Arbic α-paths can also witness to the capture-potential for the term shown in Figure 3, where α is needed in the second reduction step. Note that with these arbic α-paths we do not yet characterise variable captures that result from the contraction of *created* redexes. This we will take up in Section 4 below, where we make use of *legal* paths, cf. [3].

In sum, α-paths allow us to reprove the well-known result that in finite developments α-conversions potentially only need to be performed on the initial term (and are thus cheap).

▶ **Theorem 6.** *In finite developments α can be avoided.*



(a) Base case. **(b)** @-composition. **(c)** λ-composition.

Figure 5 Well-balanced paths [3].

Proof. Let M be a λ-term. By the above, if M contains no arbic α-path, then every development from M is α-free. Thus, it remains to observe that for every λ-term M there exists a λ-term N where $M \equiv_\alpha N$, such that N does not contain any arbic α-paths. The latter follows as all binders in M can trivially be renamed apart. ◀

This result is not new, as noted above, but illustrates how α-paths give a new perspective on this problem and therefore offer a different way to reason about α.

4 α-Paths – A Sound Characterisation For α

In this section, we generalise arbic α-paths so that the thus obtained α-paths reflect the conditions that necessitate the application of α. For that we also have to characterise the need for α that may arise for created redexes. A (sub)term, which is not a redex yet, but might become one along reduction, is called a *virtual* redex, which in turn is characterised by *legal paths*, cf. [3].

Legal Paths

In the following, to keep this paper self-contained, we briefly recall the formal definition of legal paths as established in [3]. For motivation and underlying intuitions, we kindly refer the reader to [3] and to [4], where the legal paths have been introduced. *Legal* paths start at an @-node and connect via a path the @-node with all the subterms with which it can interact in some reduction sequence. Legal paths ending at a λ-node therefore characterise a virtual redex. Legal paths are defined via the *well-balanced* paths.

The set of *well-balanced* paths (abbreviated as *wbp*) of a term M is inductively defined on $G_\alpha(M)$ as described in the following and illustrated in Figure 5.

- (base case) The path $(p, p1)$ with $M(p) = @$ is a wbp.
- (*@-composition*) let ψ, φ be two composable wbps of type `@-@` and `@-`λ, respectively. Then $\psi \cdot \varphi \cdot u$ is a wbp, where $u = (p, p1)$ with p the position of the final abstraction of φ.
- (λ-composition) Let $\varphi = (p, \dots, p_n)$ a wbp of type `@-`λ and $\psi = \sigma_a \cdot (\sigma_b)^r$ with σ_a a wbp of type `@-v` ending at position q and $\sigma_b = (p_n, q)$ a b-edge in $G_\alpha(M)$. Then $\psi \cdot (\varphi)^r \cdot u$, where $u = (p, p2)$, is a wbp.

Legal paths impose a legality constraint on the well-balanced paths, restricting the call and return paths of *cycles*. Next, we recall the definition of a cycle. Let φ be a wbp. A subpath ψ of φ is an *elementary* @-cycle of ψ (over an @-node) when (i) it starts and ends with the argument edge of the @-node and (ii) is internal to the argument N of the application corresponding to the @-node (i.e., does not traverse any variable that occurs free in N). The set of *@-cycles* of φ (over an @-node) and of the *v-cycles* of φ (over the occurrence `v` of a variable) is defined inductively, as follows: (i) every elementary @-cycle of ζ is an @-cycle; (ii) *$\mathtt{v}$-cycle*: every cyclic subpath of ζ of the form $(v)^r \cdot (\phi)^r \cdot \psi \cdot \phi \cdot v$, where $\phi = (p_2, \ldots, q_n)$ is a wbp, ψ is an @-cycle and $v = (p_1, p_2)$ a b-edge, is an $\mathtt{v}$-cycle; (iii) *@-cycle*: every subpath ψ of ζ that starts and ends with the argument edge of a given @-node, and that is composed of subpaths internal to the argument N of @- and $\mathtt{v}$-cycles over free variables of N is an @-cycle (over the @-node). As stated by the following proposition @-cycles are always surrounded by two wbps of type $\mathtt{@}$-λ, cf. [3].

▶ **Proposition 7** ([3, Corollary 6.2.26]). *Let ψ be an @-cycle of ϕ over an @-node. The wbp ϕ can be uniquely decomposed as: $\phi = \zeta_1\ \lambda\ (\zeta_2)^r\ @\ \psi\ @\ \zeta_3\ \lambda\ \zeta_4$,[5] where ζ_2 (call-path) and ζ_3 (return-path) are wbps of type $\mathtt{@}$-λ.*

Considering the statement of the proposition, the last label of ζ_1 and the first label of ζ_4 are called *discriminants*. Finally, the *legality* constraint ensures that the *call-* and the *return*-path of such cycles coincide.

▶ **Definition 8** ([3, Definition 6.2.27]). *A wbp is a* legal *path if the call and return paths of any @-cycle are one the reverse of the other and their discriminants are equal.*

▶ **Proposition 9** ([3, Section 6.2.5]). *For all (virtual) redexes of a λ-term M there is a legal path of type $\mathtt{@}$-λ in M.*

It follows that for any (created) redex along a reduction sequence starting from a λ-term M, we have a legal path in M characterising the redex. This path also encodes the reduction sequence that leads to its creation, if it is not already a redex in M.

Characterisation of α-avoidance via α-paths

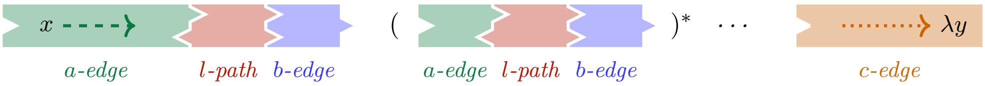
In Section 3, we have seen how arbic α-paths characterise the need for α for developments with no redex creation. The α-paths presented in this section are an extension of them and allow to characterise the need for α in λ-calculi with redex creation. α-paths are defined on the so-called albic-paths that rely on *legal* paths.[6] First, we define *alb*-paths.

▶ **Definition 10.** *Let M be λ-term, a an* a-edge*, l a legal path and b a* b-edge *in $G_\alpha(M)$ with a, l and l, b composable and b, a not composable. We call the $\mathtt{v}$–$\mathtt{v}$-path $\sigma_{alb} = a \cdot l \cdot b$ an* alb-path *of M.*

Second, essentially iterating *alb*-paths, we obtain the definition of *albic*-paths. Note that each *arbic*-path is also an *albic*-path, as each r-edge constitutes a legal path.

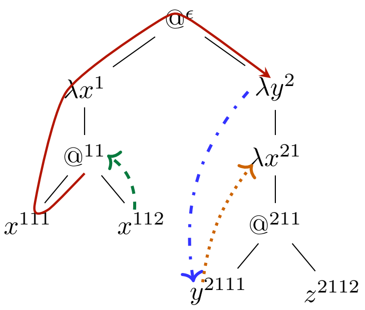
▶ **Definition 11.** *The set of* albic-paths *of M is inductively defined:*

- *(*base case*) let σ_{alb} be an* alb-*path and c a c-edge with σ_{alb}, c composable; Then the $\mathtt{v}$-λ–path $\sigma_{alb} \cdot c$ is an albic*-path.
- *(*alb-composition*) let σ_{alb} be an* alb-*path and ψ an albic*-path *with σ_{alb}, ψ composable. Then the $\mathtt{v}$-λ–path $\sigma_{alb} \cdot \psi$ is an albic*-path.



[5] We use the λ- and @-symbol to point out the start- and end-nodes of the different wbps.

[6] We call them *albic*, or $(alb)^i c$, because they consist of i (with $i \geq 1$) sequences of *alb*-paths and a final c-edge.



$$\begin{array}{ll} & \underline{(\lambda x.x\,x)\,(\lambda yx.y\,z)} \\ \to_\beta & \underline{(\lambda yx.y\,z)\,(\lambda yx.y\,z)} \\ \to_\beta & \lambda x.\underline{(\lambda yx.y\,z)\,z} \\ \to_\beta & \lambda x.(\lambda x.z\,z) \end{array}$$

$(\lambda x.x\,x)\,(\lambda yx.y\,z)$ is α-free

Figure 6 α-paths overapproximate the need for α.

Finally, based on Definitions 10 and 11 we can define *α-paths*.

▶ **Definition 12** (α-path)**.** *Let ψ be an albic-path of λ-term M. If ψ starts at a variable x and ends at a λ-node λy, where $x = y$, then the $\mathtt{v}$-λ-path ψ is called an* α-path*.*

Inductively, we can conclude that the absence of α-paths implies α-avoidance.

▶ **Lemma 13** (α-free)**.** *Suppose that there is no α-path in $G_\alpha((\lambda x.M)\,N)$ starting at a free variable in N and ending in M. Then $M[\![x := N]\!] \equiv_\alpha M[x := N]$.*

Proof. If there is no α-path, then by Definition 12 there is no albic α-path hence also no arbic α-path, as observed above. From this we conclude $M[\![x := N]\!] \equiv_\alpha M[x := N]$ by using the observation below Definition 4. ◀

Further, α-path freeness is preserved by β-reduction.

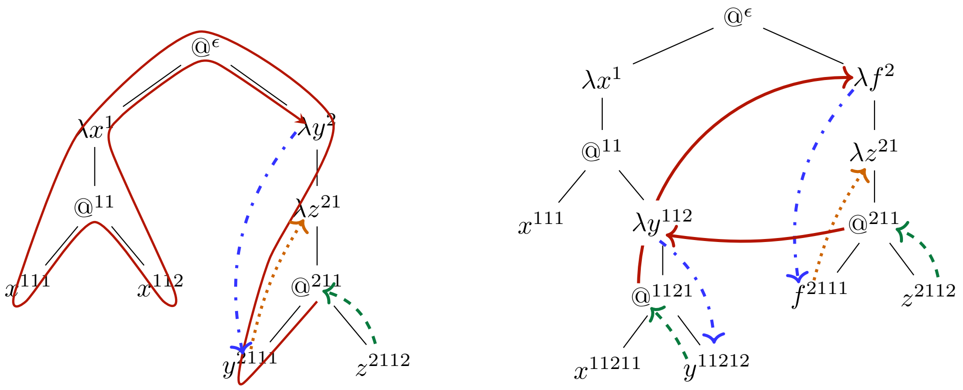
▶ **Lemma 14** (β-invariance)**.** *$\to_\beta$ preserves α-path-freeness.*

Proof. The proof proceeds the same way as the proof of Lemma 5. We restrict ourselves to the most interesting parts here. Again, we use primed variables (p', q') to range over positions in the target term t, indicating the positions they trace back to in the source term s, by unpriming (p, q). Let $s \to_\beta t$. r-edges and b-edges in $\langle G_\alpha(t)\rangle$ map back to an edge of the same type in $G_\alpha(s)$. a-edges and c-edges map back to a path of shape $\sigma_{alb}^* \cdot e$, where e denotes an edge of the same type and σ_{alb} an alb-path in $G_\alpha(s)$. An α-path in t has the following shape $(a_1', l_1', b_1', \ldots, a_n', l_n', b_n', c')$, where x_i denotes an x-edge/legal path from p_i to q_i. If we replace a-edges and c-edges by the path the map back to we get $(\sigma_{alb_1}^* \cdot a_1, l_1, b_1, \ldots, \sigma_{alb_n}^* \cdot a_n, l_n, b_n, \sigma_{alb}^* \cdot c)$, where $\sigma_{alb_i}^* \cdot x_i$ in s connects the same positions as the corresponding x-edge in t. It follows that if we have an α-path in t, then we have an α-path in s. ◀

▶ **Theorem 15.** *Let M be a λ-term. If M contains no α-path, then M avoids α.*

Proof. Assume M contains no α-path. Due to Lemma 14, α-path freeness is preserved by β-reduction. Then it follows by Lemma 13 that capture-permitting substitutions can be employed in place of capture-avoiding ones. Thus M avoids α. ◀

Not every α-path is problematic in the sense that it characterises a variable capture. An α-path may predict name collisions that will never occur if the starting variable gets substituted before the characterised redex will be contracted. This is the case for the term depicted in Figure 6. The α-path $112 \to 11 \to 111 \to 1 \to \epsilon \to 2 \to 2111 \to 21$ is harmless, as



(a) $G_\alpha((\lambda x.x\,x)\,(\lambda yz.y\,z))$.

(b) $G_\alpha((\lambda x.x\,(\lambda y.x\,y))\,(\lambda fz.f\,z))$.

Figure 7 Unremovable α-paths.

the variable x^{112} gets substituted by the argument $\lambda yx.y\;z$ before the redex characterised by the legal path $11 \to 111 \to 1 \to \epsilon \to 2$ gets contracted. Thus, α-paths overapproximate the need of α. This overapproximation is sufficiently accurate to still allow interesting statements about different calculi, since α-avoidance is mainly about *unremovable* α-paths.

An α-path is called *unremovable*, if it starts at a variable occurrence at position $p1q$ and ends at its binder at position p ($p \prec p1q$). In Theorem 6 we employed that we can get rid of arbic α-paths by naming all binders appropriately. This is possible because the starting and the ending position of these paths are always parallel. For unremovable α-paths this is not always the case, as illustrated by the λ-terms in Figures 7a and 7b. Note that Figure 7b illustrates that an unremovable α-path does not necessarily have to contain legal paths from a position p to a position q with $q \prec p$.

▶ **Lemma 16.** *For every λ-term M containing no unremovable α-paths, there exists a λ-term N where $M \equiv_\alpha N$, such that N does not contain any α-paths.*

Undecidability

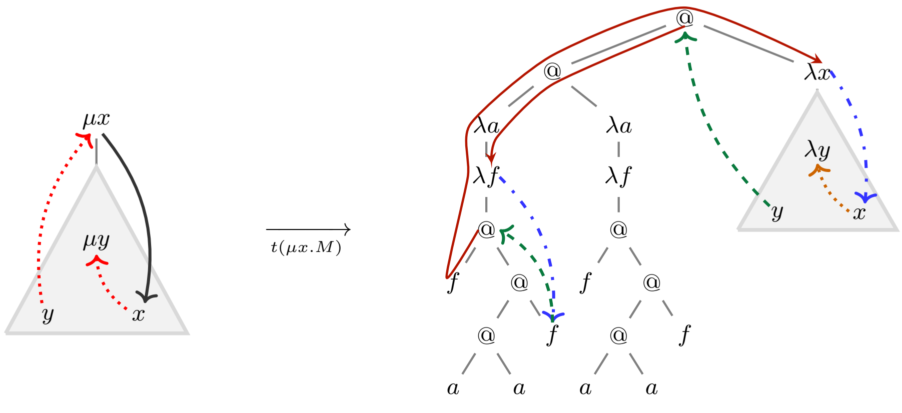
Arbitrary λ-terms may have an unbounded set of legal paths, all of them characterising a different virtual redex. For such terms, making a prediction about the need for α via α-paths is not feasible. This problem is even undecidable for leftmost–outermost reductions, as established by our next result.

▶ **Theorem 17.** *α-avoidance is undecidable for the leftmost–outermost reduction strategy.*

Proof. In proof, we employ a reduction from Post's correspondence problem (PCP short), whose undecidability is well-known [32]. Recall that PCP asks whether for an arbitrary finite set of string pairs $\langle s_1, s_1'\rangle, \langle s_2, s_2'\rangle, \dots, \langle s_n, s_n'\rangle$ over the alphabet $\{\mathsf{a},\mathsf{b}\}$, there exists indices $i_j \in \{1, 2, \dots, n\}$ such that

$$s_{i_1} s_{i_2} \dots s_{i_k} = s'_{i_1} s'_{i_2} \dots s'_{i_k} \qquad k \geqslant 1\ .$$

It is not difficult to define λ-terms for (i) strings aa, bb, namely $AA := \lambda abx.a\,(a\,x)$ and $BB := \lambda abx.b\,(b\,x)$, respectively; (ii) *conditionals* (denoted as ITE); (iii) *pairs* ($PAIRS$) and (iv) in particular PCP (PCP), such that the λ-term PCP takes an (encoding) of list of pairs



(a) A self-capturing chain in $\mu.M$.

(b) An α-path in $t(\mu.M)$.

Figure 8 A self-capturing chain in μ is an α-path in Λ_μ.

as input and recursively combines them, until a solution is produced (if it exists at all). As the leftmost–outermost strategy is normalising for the λ-calculus, this solution can be found by this strategy. Now, consider the following program

$$(ITE\,(PCP\,PAIRS)\,AA\,BB)\,(\lambda xyz.(x\,z)\,y)\ ,$$

ITE, PCP, $PAIRS$, AA and BB are defined as above. As $ITE\,(PCP\,PAIRS)\,AA\,BB$ is typable, α can be avoided in its reduction, cf. Section 5.3 or [31, Section 11.3.2]. If the problem has a solution, it will reduce to the λ-term $AA\,(\lambda xyz.(x\,z)\,y)$, where α is unavoidable. Otherwise, it will reduce to the λ-term $BB\,(\lambda xyz.(x\,z)\,y)$, from which we get with one β-step to $\lambda bx.b\,(b\,x)$. Moreover, as mentioned the reduction sequence to these terms is α-free. Thus, if we further reduce these terms to normal form, then we need α iff the PCP problem has a solution. Thus, we conclude the theorem. ◀

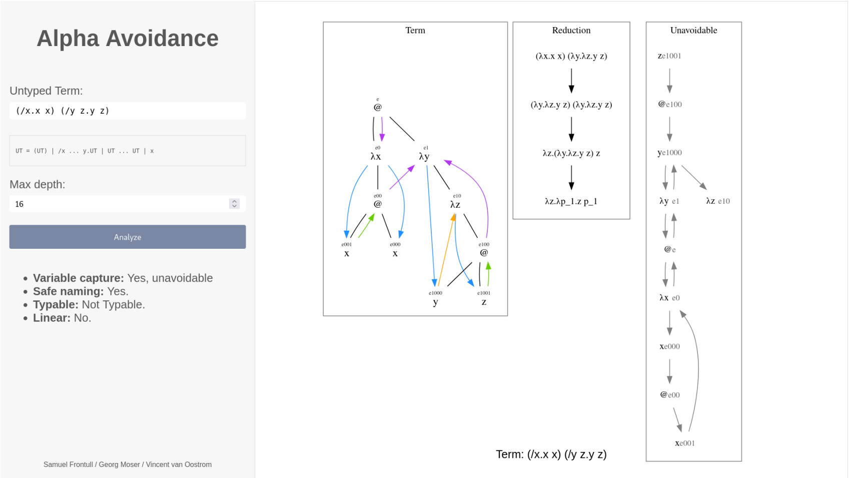
As already mentioned, α-paths characterise α-avoidance for seemingly unrelated calculi like (i) developments, (ii) affine λ-calculus, (iii) weak λ-calculus and (iv) safe λ-calculus. In Section 3 we have already seen this for developments and in the next section we illustrate this characterisation of the affine and the weak λ-calculus as well as the safe λ-calculus [11, 9].

In the sequel, we clarify the ancestry of α-paths wrt. the concept of *chains* in the μ-calculus, cf. [17]. Further, we briefly detail our tool Alpha that can be used to compute and illustrate α-paths.

Interpretation of μ in the λ-calculus

We show that α-paths are a strict generalisation of the chains considered for the μ-calculus in [17]. We do this by considering the sub-calculus Λ_μ of the λ-calculus obtained by the t-image of μ-terms defined as $t(x) = x$, $t(M\,N) = t(M)\,t(N)$, $t(\mu x.M) := A\,A\,(\lambda x.t(M))$ for $A = \lambda af.f\,(a\,a\,f)$. As suggested in [17], this translation allows simulating μ-terms in the λ-calculus, provided that we adopt the leftmost–outermost reduction strategy.

$$\begin{aligned} t(\mu x.M) := A\,A\,(\lambda x.t(M)) &\to_\beta (\lambda f.f\,(A\,A\,f))\,(\lambda x.t(M)) \to_\beta (\lambda x.t(M))\,(A\,A\,(\lambda x.t(M))) \\ &\to_\beta t(M)[\![x := A\,A\,(\lambda x.t(M))]\!] = t(M)[\![x := t(\mu x.M)]\!] \\ &= t(M[\![x := \mu x.M]\!])\ . \end{aligned}$$



Alpha Avoidance

Untyped Term: (/x.x x) (/y z.y z)

UT = (UT) | /x ... y.UT | UT ... UT | x

Max depth: 16

- **Variable capture:** Yes, unavoidable
- **Safe naming:** Yes.
- **Typable:** Not Typable.
- **Linear:** No.

Samuel Frontull / Georg Moser / Vincent van Oostrom

Term: (/x.x x) (/y z.y z)

Figure 9 α-avoidance tool web-interface.

In Λ_μ we can use α-paths to characterise α. We sketch the argument that an α-path in $t(M)$ for some μ-term M correspond a self-capturing chain in M. Note that, as reducing Turing's fixed point combinator $A\,A$ itself does not cause any capturing problems, we do not introduce "new α-problems". Thus, we only need to characterise the paths that correspond to reductions at the root of a reduct of $A\,A\,(\lambda x.M)$ to characterise the need for α in Λ_μ. For that, we observe that a pair of connected binding and capturing links in μ correspond to an *alb*-path in Λ_μ and a self-capturing chain to an α-path. Figure 8 illustrates this correspondence.

Implementation

Based on the notion of α-paths, we have implemented a tool, dubbed Alpha, to (partially) decide whether or not α-conversion can be avoided. The tool is publicly available and can either be accessed via the command-line or its web interface. The web interface also visualises the computed α-paths.

Depending on whether α-paths can be found or not (up to a variable depth), the tool gives one of the following results:

1) `alpha free`, if no α-paths were found and the calculation is terminated;
2) `alpha can be avoided`, if α-paths were found (but no unremovable α-paths); in this case, the tools prints an α-equivalent term for which the calculation is α-free;
3) `alpha is unavoidable`, if unremovable α-paths have been found;

or returns `maybe`, if no α-paths have been found, but the computation has not been terminated (the maximum depth has been reached). Recall that the problem is undecidable, cf. Section 4.[7] Listing 1 shows an exemplary output of the command line tool.

Listing 1 Church encoding of 3^2.

```
$ dune exec bin/main.exe "(/f␣x.f␣(f␣x))␣(/f␣x.f␣(f␣(f␣x)))"
alpha can be avoided:
(/f x.f (f x)) (/f p_12.f (f (f p_12)))
```

The web interface displays the α-graph and the computed α-paths. Figure 9 shows a screenshot of the tool illustrating this for $(\lambda x.x\,x)\,(\lambda yz.y\,z)$.

[7] The command line tool and the link to the web interface can be found at `https://tcs-informatik.uibk.ac.at/tools/alpha/`.

5 α-Avoidance In Affine, Safe And Weak λ-Calculi

In this section, we show how α-paths can be applied to analyse the need for α in restricted λ-calculi.

5.1 The affine λ-calculus

The affine λ-calculus [21, 23, 27, 38], forbids duplication by restricting term-formation, requiring the variable x to occur free at most once in M in an abstraction term $\lambda x.M$. This calculus is strongly normalising; we recall the central definition.

▶ **Definition 18.** *The set Λ_{AFF} of* affine λ*-terms is a subset of Λ and inductively defined as follows:*

- *(var)* $x \in \Lambda_{AFF}$*, for all variables* x*;*
- *(app)* $M, N \in \Lambda_{AFF} \implies M\,N \in \Lambda_{AFF}$*, if* $\mathcal{FV}(M) \cap \mathcal{FV}(N) = \emptyset$*;*
- *(abs)* $M \in \Lambda_{AFF} \implies \lambda x.M \in \Lambda_{AFF}$*.*

Since the size of terms steadily decreases with each reduction step and variables persist linearly along reductions, it follows that this calculus is strongly normalising. This allows a precise analysis for the need of α.

▶ **Lemma 19.** *Let $M \in \Lambda_{AFF}$, $M \to_\beta N$ and $q \prec p$ for some positions p, q in M. If $p \blacktriangleright p'$ and $q \blacktriangleright q'$, then $q' \prec p'$.*

Proof. Since we have no duplication, each symbol has at most one copy in N. We distinguish the following cases where we have $p \prec q$, with $p \blacktriangleright p'$ and $q \blacktriangleright q'$:

1. p, q both in the context: Then as $p' = p$ and $q' = q$ so by assumption we have $p' \prec q'$.
2. $p = o11s_1$ and $q = o11s_2$ both in the body: Then from $p \prec q$ we know that $s_1 \prec s_2$ and we have $os_1 = p' \prec q' = os_2$.
3. $p = o2s_1$ and $q = o2s_2$ both in the argument: Then from $p \prec q$ we know that $s_1 \prec s_2$ and we have $ots_1 = p' \prec q' = ots_2$.
4. p is in the context and $q = o11s$ in the body. Then $p' = p$ and $q' = os$ and since $p \prec q$ we also have $p' \prec q'$.
5. p is in the context and $q = o2s$ in the argument. Then $p' = p$ and $q' = oqs$. Since we know that $p \prec o$ (because it is in the context), we also have $p' \prec q'$.

The other cases can be omitted because they violate the assumption that $p \prec q$. ◀

Since each β-step preserves the property proven in Lemma 19, we cannot have a reduct of M where for the copy of p (the position of a variable), p', and the copy q (the position of an abstraction), q', we have $p' \parallel q'$, if for the origins we have $q \prec p$. This would temporarily be needed to form a redex whose contraction causes a variable capture. Moreover, as argued above we could map back such a setting to an (unremovable) α-path in M. We conclude that no such path can exist in M.

▶ **Lemma 20.** *Let M be an arbitrary term in Λ_{AFF}. There are no unremovable α-paths in M.*

In sum, we obtain the following, well-known result.

▶ **Theorem 21.** *In the affine λ-calculus α can be avoided.*

Proof. Due to Lemma 20 it only remains to prove that for every affine λ-term M there exists an affine λ-term N such that $M \equiv_\alpha N$ and N avoids α. This, however, follows from Lemma 19 in conjunction with Lemma 16. ◀

$$(var)\ \frac{}{x : A \vdash_s x : A} \qquad (const)\ \frac{}{\vdash_s f : A}\ f : A \in \Xi \qquad (wk)\ \frac{\Gamma \vdash_s M : A}{\Delta \vdash_s M : A}\ \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_s M : A}{\Gamma \vdash_{asa} M : A}$$

$$(app_{asa})\ \frac{\Gamma \vdash_{asa} M : A \to B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_{asa} M\ N : B} \qquad (app)\ \frac{\Gamma \vdash_{asa} M : A \to B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_s M\ N : B}\ \boldsymbol{ord\, B \leq ord\, \Gamma}$$

$$(abs)\ \frac{\Gamma, x_1 : A_1, \dots, x_n : A_n \vdash_{asa} M : B}{\Gamma \vdash_s \lambda x_1^{A_1} \dots x_n^{A_n}.M : (A_1, \dots, A_n, B)}\ \boldsymbol{ord\, (A_1, \dots, A_n, B) \leq ord\, \Gamma}$$

Figure 10 The safe λ-calculus [9].

5.2 The safe λ-calculus

In the safe λ-calculus, a variable capture can never occur by definition, thus α is not needed. This calculus was first introduced in [28] and then further developed and formalised in [11]. The fundamental concept allowing α-free computations is known as the *safety* restriction. In the standard form this syntactic restriction restricts the free occurrences of variables according to their type-theoretic order. It was initially introduced for higher-order grammars, cf. [16]. The safe λ-calculus is the result of the transposal of the safety condition for higher-order grammars to the simply-typed calculus à la Church.

In this section, we show that α cannot be avoided in the safe λ-calculus as presented in [11] and [9] by giving a counterexample and clarify why we need to stick to a more restricted version of the safe λ-calculus (as presented in [10]) if we aim for α-free reductions. More precisely, we show how α-paths can be used to reason that α is not needed in the safe λ-calculus and that the absence of α-paths implies the safe variable typing convention.

Simple types over the atomic type o are defined as usual, cf. [6], $A_1 \to \dots \to A_n \to o$ is abbreviated as $(A_1, \dots, A_n, o)$ and (o) as o. The order of a type is given by (i) $ord\, o :=\ 0$ and (ii) $ord\,(A \to B) :=\ max(1 + ord\, A, ord\, B)$. The order of a typed term or symbol is defined to be the order of its type. The lowest order in a set of type assignments Γ is denoted by $ord\ \Gamma$ (0 if Γ empty). A set of type assignments Γ is *order-consistent* if all the types assigned to a given variable are of the same order.

▶ **Example 22.** Let $\Gamma = \{x : o, y : (o, o)\}$, then Γ is order-consistent and $ord\ \Gamma = 0$. Conversely, the set $\{x : ((o, o), o), x : (o, o)\}$ is not order-consistent and $ord\, \Gamma = 1$.

▶ **Definition 23.** *A term M of type A is said to be* safe*, if $\mathcal{FV}(M) \vdash_s M : A$ is a valid statement in the inference system of the safe λ-calculus depicted in Figure 10.*

We can abstract multiple variables at once, $\lambda x_1 \dots x_n.M$, provided that they are pairwise distinct (*abs*-rule). In particular, $\lambda.x$ and $\lambda x^o.\lambda x^o.x$ are valid λ-terms of the safe λ-calculus, $\lambda x^o x^o.x$ is not. The conditions on the types in the *app*- and *abs*-rule ensure that the variables occurring free in some term M have order at least the order M (*safety* condition). The subscript *asa* stands for *almost safe* (application). Almost safe applications can be turned into a safe term via further applications or further abstractions. For example, $(\lambda x^o y^o.x)\, z$ (with z of type o) is an almost safe application but not safe. However, in $(\lambda x^o y^o.x)\, z\, f$ (with f, z of type o) it is a subterm of a safe application.

In the safe λ-calculus, consecutive redexes are contracted simultaneously, as the standard β-reduction does not preserve safety [9, Section 3.1.2]. This requires a notion of simultaneous substitution. The definitions of simultaneous capture-permitting and simultaneous capture-avoiding substitution are given in Table 2.

Table 2 Simultaneous capture-avoiding and simultaneous capture-permitting substitution.

M	$M[\![\overline{N}/\overline{x}]\!]$ (sim. capture-avoiding)	$M[\overline{N}/\overline{x}]$ (sim. capture-permitting)
x_i	N_i	N_i
y	y	y
$e_1\, e_2$	$e_1[\![\overline{N}/\overline{x}]\!]\, e_2[\![\overline{N}/\overline{x}]\!]$	$e_1[\overline{N}/\overline{x}]\, e_2[\overline{N}/\overline{x}]$
$(\lambda\overline{y}.e)[\![\overline{N}/\overline{x}]\!]$	$\lambda\overline{y}.e[\![\overline{N'}/\overline{x'}]\!]$ where $\overline{x'} = \overline{x} - \overline{y}$ if $\overline{y} \cap FV(t) = \emptyset$ for all $t \in \overline{N'}$, else $\lambda\overline{z}.e[\![\overline{z}/\overline{y}]\!][\![\overline{N}/\overline{x}]\!]$ where z_i fresh for $e, \overline{N}$	$\lambda\overline{y}.e[\overline{N'}/\overline{x'}]$ where $\overline{x'} = \overline{x} - \overline{y}$

▶ **Definition 24** (*safe redex* [9, Definition 3.21])**.** *An untyped safe redex is an untyped almost safe application (a succession of several standard redexes) of the form $(\lambda x_1 \dots x_n.M)\, N_1 \dots N_l$ for some $l, n \geq 1$ where $\lambda x_1 \dots x_n.M$ is safe and each N_i, for $1 \leq i \leq n$, is safe.*

▶ **Definition 25** (*safe redex contraction*)**.** *The relation β_s is defined on the set of safe redexes as follows:*

$$\beta_s = \{(\lambda x_1 \dots x_n.M)\, N_1\ \dots\ N_l \mapsto (\lambda x_{l+1} \dots x_n.M)[N_1 \dots N_l/x_1 \dots x_l] \mid n > l\}$$
$$\cup\, \{(\lambda x_1 \dots x_n.M)\, N_1\ \dots\ N_l \mapsto M[N_1 \dots N_n/x_1 \dots x_n]\, N_{n+1}\ \dots\ N_l \mid n \leq l\}$$

where $\lambda.M = M$ and $M[\overline{N}/\overline{x}]$ denotes the simultaneous capture-permitting substitution.

Note that simultaneous capture-permitting substitution cannot be applied serially because it may require α. The statement $M[x_1 := N_1][x_2 := N_2] = M[x := y, y := z]$ is not true in general, as x_2 may be free in N_1, e.g. $x[x := y][y := z] = z$ and $x[x := y, y := z] = y$.

▶ **Definition 26.** *The* safe *β-reduction, written as $\rightarrow_{\beta_s}$, is the compatible closure of the relation β_s with respect to the formation rules of the safe λ-calculus.*

In addition to the inference rules, the safe variable typing convention is adopted, which restricts the naming of variables according to their type.

▶ **Definition 27** (*safe variable typing convention* [9])**.** *A type-annotated term M is order-consistent just if the set of type-assignments induced by the type annotations in M is. In any definition, theorem or proof involving countably many terms, it is assumed that the set of terms involved is order-consistent.*

According the safe variable typing convention, variables of distinct order must have distinct names. This is crucial for α-avoidance in the safe λ-calculus.[8] However, if we consider the term $M = \lambda y^o.(\lambda x^o y^o.x)\, y$ we see that although this term is safe ($\vdash_s \lambda y^o.(\lambda x^o y^o.x)\, y : (o, o)$) and $(\lambda x^o y^o.x)\, y$ is a safe redex, it cannot be contracted by means of capture-permitting substitution, because this would lead to a variable capture. This invalidates a central property of this calculus, according to which a variable capture can never happen, and leads to the fact that we may compute different normal forms for α-equivalent terms.[9]

[8] $\{y : (((o,o),o),o), z : ((o,o),o)\} \vdash_s \underline{(\lambda x^{((o,o),o)} y^{(o,o)} z^o.x)\ (\lambda q^{(o,o)}.y\ z)} : ((o,o),o,((o,o),o))$ is a counter-example to [9, Lemma 3.17].

[9] Compare to the errata published at `https://github.com/blumu/dphil.thesis/blob/erratum/Current/thesis-erratum/dphilerratum.pdf`.

$$(var)\ \frac{}{\{x : A\} \vdash_{s\alpha} x : A} \qquad (const)\ \frac{}{\vdash_{s\alpha} f : A}\ f : A \in \Xi \qquad (wk)\ \frac{\Gamma' \vdash_{s\alpha} M : A}{\Gamma \vdash_{s\alpha} M : A}\ \Gamma' \subset \Gamma$$

$$(app)\ \frac{\Gamma \vdash_{s\alpha} M : (A_1, \dots, A_n, B) \quad \Gamma_{\geq m} \vdash_{s\alpha} N_1 : A_1 \quad \dots \quad \Gamma_{\geq m} \vdash_{s\alpha} N_j : B_j}{\Gamma \vdash_{s\alpha} M\, N_1 \, \dots \, N_j : B} \quad m = ord\, B$$

$$(abs)\ \frac{\Gamma_{\geq m} \cup \{x_1 : A_1, \dots, x_n : A_n\} \vdash_{s\alpha} M : B}{\Gamma \vdash_{s\alpha} \lambda x_1 \dots x_n . M : (A_1, \dots, A_n, B)} \quad \boldsymbol{m = ord\,(A_1, \dots, A_n, B)}$$

Figure 11 An α-avoiding safe λ-calculus.

A more restrictive set of rules is needed to resolve this issue. These rules are depicted in Figure 11.[10] In this system we dropped the "almost safety" and allow to type only applications that provide enough arguments to abstractions. More precisely, if an argument of order k is provided, the arguments of all abstracted variables of order k and higher must be provided. In this way, we avoid free variables ending up in the scope of abstractions of the same order during reduction. This avoids potential variable capture, since it can be assumed that free variables are always of a higher order than the abstractions they enter the scope of. Therefore, according to the safe variable typing convention, they are named differently.[11]

▶ **Example 28.** The simply-typed term $(\lambda f^{(o,o,o)} y^o . f\, y)\, (\lambda x^o y^o . x)$ is derivable in the safe λ-calculus from Figure 10, but not in the system from Figure 11 because of the unsafe application $f\, y$. Indeed, this term reduces in one step to $\lambda y^o . (\lambda x^o y^o . x)\, y$ where α is required to further reduce it.

In the following Lemma 29 we show that the safe λ-calculus of Figure 11 avoids α by reasoning with α-paths. This can be done by interpreting safe λ-terms as ordinary terms.

▶ **Theorem 29.** *In the safe λ-calculus no variable capture can occur, provided that the safe variable typing convention is adopted.*

Proof. Suppose we have an α-path in a safe λ-term M with $\Gamma \vdash_{s\alpha} M : A$. Then this path would start at a variable y occurring free in the argument N of some application, which is connected via a legal path to an abstraction λx binding a variable x in the scope of a λy, as illustrated below. In such case, by definition of safe terms, we know that $\lambda x . M$ and N are both safe. Moreover, we know that $ord\, y \geq N$ and $ord\, N = ord\, x$. We can therefore have the following two cases: (i) $ord\, y > ord\, x$ or (ii) $ord\, y = ord\, x$. In any case, as the subterm $\lambda y . M'$ would be unsafe in isolation, we conclude that the λy and the λx must be jointly abstracted. By definition of safe β-reduction, we know that compound abstractions of same order are contracted simultaneously. Therefore, we cannot have a variable capture. ◀

5.3 The weak λ-calculus

The *weak* λ-calculus [39] forbids to contract *open* redexes, i.e. redexes that involve free variables that are bound outside. Thus, if the name of the free variables and the bound variables are chosen to be distinct, a variable capture can by definition never occur. We recall the notion of *weak β-reduction*.

[10] Simultaneous substitutions coincide with the singleton substitutions from Table 1 in the case $|\overline{x}| = 1$.

[11] We note that these rules correspond to the rules of the safe λ-calculus published in [10] and to the typing rules for *long-safe* terms (without constants) listed in [9, Table 3.2].

▶ **Definition 30** (*weak λ-reduction [39, Definition 3.1]*)**.** *A particular occurrence of a redex R in a λ-term M will be called weak in M iff no variable-occurrence free in R is bound in M. A weak β-contraction in M is the contraction of a β-redex-occurrence that is weak in M.*

The characterisation of the virtual redexes by legal paths is not suitable for the weak λ-calculus, since they include redexes that are not reduced at all. However, we can infer from the structure of the unremovable α-paths that α can also be avoided in this calculus. To this end, we rely on the fact that bound variables are never released, i.e. they do not change or loose their binder.

▶ **Lemma 31.** *For every λ-term M there exists a λ-term N such that $M \equiv_\alpha N$ and any $\rightarrow_{\beta w}$-reduction from N is α-free.*

Proof. We prove it by showing that the name-collision characterised by an unremovable α-paths will not arise. Suppose we have an unremovable α-path in a λ-term M. Such path has the shape $\sigma^+_{alb} \cdot c$. Assume, that at some point along the reduction sequence of M we reach a λ-term N, containing a redex R whose contraction leads to the predicted name-collision. Let q be the position of the variable y occurring free in the argument of R in N. Since the position q originates from position p in M ($p \blacktriangleright p' \blacktriangleright \ldots \blacktriangleright q$) and the variable occurrence at position p in M was bound, we know that also the variable y at position q in N is bound (as bound variables are never released). So R would be an open redex and thus not contracted. Any other α-path can be removed by naming each binder distinctly and distinct from the free variables, as proven in Lemma 16. ◀

In sum, α-avoidance is immediate from the definitions.

▶ **Theorem 32.** *In the weak λ-calculus α can be avoided.*

6 Conclusion

We have presented a sound characterisation of α-avoidance, via α-paths, generalising self-capturing chains [17], studied in the context of the μ-calculus; α-paths exploit the predictive power of legal paths, characterising virtual redexes of a λ-term M, that is, all redexes occurring in some reduction sequence starting from M. By reasoning on the structure of the initial term, we estimated whether α is needed, when contracting these virtual redexes. Further, we have shown undecidability of α avoidance for (leftmost-outermost reductions in) the untyped λ-calculus. Moreover, α-paths were instantiated to different restrictive λ-calculi, where they can be used to show that α can be avoided, namely developments, the affine λ-calculus, the weak λ-calculus and the safe λ-calculus. In short, forbidding redex creation, duplication, or the contraction of redexes involving variables bound outside is enough to allow α-avoidance. For all calculi where we can avoid α, we can infer potential α-conversions needed to allow α-free computations from the α-paths. This allows to move a dynamic problem to a static one.

We have shown that α-avoidance is undecidable for the leftmost–outermost strategy in the untyped λ-calculus. These leaves the question open, whether undecidability holds in general. We further note that α-paths only overapproximate the need for α. It remains an open question whether we could tighten the definition of α-paths such that the established (sound) characterisation becomes precise, that is, complete. These questions are left to future work.

References

1 Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267, 1993. doi:10.1006/inco.1993.1044.

2 Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. Paths in the lambda-calculus. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 426–436. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316048.

3 Andrea Asperti and Stefano Guerrini. *The optimal implementation of functional programming languages*, volume 45 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1998.

4 Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995. doi:10.1016/0304-3975(94)00279-7.

5 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 263–274. ACM, 2013. doi:10.1145/2500365.2500606.

6 Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2nd revised edition, 1984. doi:10.2307/2274112.

7 Henk P. Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

8 Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the Weak Lambda-Calculus. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005. doi:10.1007/11601548_7.

9 William Blum. *The Safe Lambda Calculus*. PhD thesis, Oxford University, UK, 2009.

10 William Blum and C.-H. Luke Ong. The Safe Lambda Calculus. In *TLCA*, pages 39–53, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

11 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, Volume 5, Issue 1, February 2009. doi:10.48550/arXiv.0901.2399.

12 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 75(5):381–392, 1972. Proc. 19th International Conference on Automated Deduction. doi:10.1016/1385-7258(72)90034-0.

13 Felice Cardone and J. Roger Hindley. Lambda-Calculus and Combinators in the 20th Century. In *Logic from Russell to Church*, 2009. doi:10.1016/S1874-5857(09)70018-4.

14 Alonzo Church and John Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936. doi:10.1090/S0002-9947-1936-1501858-0.

15 Haskell B. Curry. *Combinatory Logic*. Amsterdam: North-Holland Pub. Co., 1958.

16 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

17 Jörg Endrullis, Clemens Grabmayer, Jan Willem Klop, and Vincent van Oostrom. On equal μ-terms. *Theoretical Computer Science*, 412(28):3175–3202, 2011. doi:10.1016/j.tcs.2011.04.011.

18 Samuel Frontull. Alpha Avoidance. Master's thesis, University of Innsbruck, 2021.

19 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Proc. 14th LICS*, pages 214–224, 1999. doi:10.1109/LICS.1999.782617.

20 J. Roger Hindley. Reductions of Residuals are Finite. *Transactions of the American Mathematical Society*, 240:345–361, 1978. doi:10.2307/1998825.

21 J. Roger Hindley. BCK-Combinators and Linear lambda-Terms have Types. *Theoretical Computer Science*, 64(1):97–105, 1989. doi:10.1016/0304-3975(89)90100-X.

22 Martin Hyland. A simple proof of the Church–Rosser theorem. Oxford University, UK, 1973.

23 Bart Jacobs. Semantics of lambda-I and of other substructure lambda calculi. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 195–208, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. doi:10.1007/BFb0037107.

24 Jan W. Klop. *Combinatory reduction systems*. PhD thesis, Rijksuniversiteit Utrecht, 1980.

25 Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

26 Clemens Kupke, Johannes Marti, and Yde Venema. Size measures and alphabetic equivalence in the μ-calculus. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 18:1–18:13. ACM, 2022. doi:10.1145/3531130.3533339.

27 Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming*, 14(6):623–633, 2004. doi:10.1017/S0956796804005131.

28 Jolie G. de Miranda. *Structures generated by higher-order grammars and the safety constraint*. PhD thesis, University of Oxford, UK, 2006.

29 Maxwell H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of mathematics*, 43(2):223–243, 1942. doi:10.2307/1968867.

30 Vincent van Oostrom and Roel de Vrijer. Four equivalent equivalences of reductions. *Electronic Notes in Theoretical Computer Science*, 70(6):21–61, 2002. WRS 2002, 2nd International Workshop on Reduction Strategies in Rewriting and Programming - Final Proceedings (FLoC Satellite Event). doi:10.1016/S1571-0661(04)80599-1.

31 Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, January 1987.

32 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946. doi:10.2307/2267252.

33 John Barkley Rosser. Review: H. B. Curry, A New Proof of the Church–Rosser Theorem. *Journal of Symbolic Logic*, 21(4):377–378, 1956.

34 David E. Schroer. *The Church–Rosser Theorem*. PhD thesis, Cornell University, 1965.

35 Rick Statman. On the complexity of alpha conversion. *The Journal of Symbolic Logic*, 72(4):1197–1203, 2007. doi:10.2178/jsl/1203350781.

36 Terese. *Term rewriting systems*. Cambridge University Press, 2003.

37 Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.

38 Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. *Journal of Functional Programming*, 26:e21, 2016. doi:10.1017/S095679681600023X.

39 Naim Çağman and J. Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1-2):239–247, 1998. doi:10.1016/S0304-3975(97)00250-8.