


# 1 $\alpha$ -avoidance

2 Samuel Frontull  

3 University of Innsbruck, Austria

4 Georg Moser  

5 University of Innsbruck, Austria

6 Vincent van Oostrom 

7 Independent researcher, Netherlands

## 8 — Abstract —

---

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

20 **2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Program analysis

21 **Keywords and phrases**  $\lambda$ -calculus, variable capture,  $\alpha$ -conversion, developments, safe  $\lambda$ -calculus,  
22 undecidability

23 **Digital Object Identifier** 10.4230/LIPIcs.FSCD.2023.18

24 **Supplementary Material** *Software (Alpha Avoidance Tool)*: [https://tcs-informatik.uibk.ac.at/  
25 tools/alpha/](https://tcs-informatik.uibk.ac.at/tools/alpha/)

## 26 **1** Introduction

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

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

---

<sup>1</sup> As far as we know this problem is the  $\alpha$ - $\alpha$ -problem, that is, this is the first  $\alpha$ -problem in the literature on the  $\lambda$ -calculus.



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

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

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

## 66 Contributions.

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

$$\begin{array}{l}
\frac{(\lambda x.x x) (\lambda y z.y z)}{\rightarrow_{\beta} \frac{(\lambda y z.y z) (\lambda y z.y z)}{\rightarrow_{\beta} \lambda z.(\lambda y z.y z) z}} \text{A} \\
\rightarrow_{\beta} \lambda z.(\lambda y z.y z) z \\
\rightarrow_{\alpha} \lambda z.(\lambda y z'.y z') z \text{C} \\
\rightarrow_{\beta} \lambda z z'.z z'
\end{array}
\qquad
\begin{array}{l}
\frac{(\lambda f c.f (f c)) (\lambda z x y.z y x)}{\rightarrow_{\beta} \lambda c.(\lambda z x y.z y x) ((\lambda z x y.z y x) c)} \text{A} \\
\rightarrow_{\beta} \lambda c.(\lambda z x y.z y x) (\lambda x y.c y x) \text{B} \\
\rightarrow_{\beta} \lambda c.(\lambda x y.(\lambda x y.c y x) y x) \\
\rightarrow_{\alpha} \lambda c.(\lambda x y.(\lambda x y'.c y' x) y x) \text{C} \\
\rightarrow_{\beta} \lambda c x y.(\lambda y'.c y' y) x \\
\rightarrow_{\beta} \lambda c x y.c x y
\end{array}$$

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

■ **Figure 1**  $\alpha$  is needed in the  $\lambda$ -calculus.

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

96 1. *Developments* [14] are reductions in which no *created* redexes are contracted. Stated  
 97 differently, in a development from  $M$  only *residuals* of redexes in  $M$  (no virtual redexes)  
 98 are contracted. Intuitively,  $\alpha$  can then be avoided since all residuals of a given redex  
 99 are disjoint along a development.

100 Developments [14, 6] feature prominently in the  $\lambda$ -calculus since its inception. They  
 101 form the basis for the proof that  $\beta$ -reduction has the Church–Rosser property, more  
 102 precisely, that parallel  $\beta$ -reduction has the diamond property and satisfies the cube  
 103 law, using that all developments are finite (unlike arbitrary reductions).

104 2. The *linear* (*affine*)  $\lambda$ -calculus [21], forbids *duplication*. That is achieved formally by  
 105 restricting term-formation, requiring the variable  $x$  to occur free exactly (at most) once  
 106 in  $M$  in an abstraction term  $\lambda x.M$ . Intuitively,  $\alpha$  can then be avoided since variables  
 107 persist linearly along reductions.

108 Though the linear  $\lambda$ -calculus [21, 23, 27, 38] had been studied before, it rose to  
 109 prominence after the introduction of linear logic, via the connection between linear  
 110  $\lambda$ -terms and MLL-proofnets, with abstractions and applications corresponding to pars  
 111 and tensors. Linearity affords nice properties, e.g. termination and simple typability.

112 3. The *weak*  $\lambda$ -calculus [39] forbids to contract *open* redexes. Intuitively,  $\alpha$  is then avoided  
 113 since when substituting by closed terms only, there’s no risk of variable capture either.  
 114 Weak reduction [37, 31, 1, 39, 8, 5] forms the basis of functional programming languages  
 115 such as Haskell that evaluate to weak head normal form. Indeed, the fact that  $\alpha$ -  
 116 conversion can then be avoided was stated as an explicit motivation in [31], and makes  
 117 that functional programs can be represented as orthogonal term rewrite systems and  
 118 weak reduction can be optimally implemented via Wadsworth’s graph reduction. (Weak  
 119 reduction is often paraphrased as ‘no reduction under a  $\lambda$ ’, but that restriction is  
 120 undesirable as it breaks the Church–Rosser property.)

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

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

126 that this can be obtained via  $\alpha$ -paths, under a standard embedding of modal  $\mu$ -formulas  
 127 in the  $\lambda$ -calculus, representing unfolding using Turing's fixed-point combinator.

128 Though the literature on the modal  $\mu$ -calculus is rich, only recently issues related  
 129 to  $\alpha$ -conversion seem to have been addressed [26]. The main point of this example  
 130 is to suggest that the technology developed here for avoiding  $\alpha$  in the  $\lambda$ -calculus,  
 131 can be transferred to other calculi with binding, in mathematics, logic, programming  
 132 languages, linguistics, music theory, etc..

133 5. In the *safe  $\lambda$ -calculus* [10, 9, 11] the occurrences of (free) variables are restricted  
 134 according to the order of their type. Intuitively, this restriction on the order of the  
 135 types of the variables can be transferred to the variables, guaranteeing that  $\alpha$  can be  
 136 avoided. (Note that as observed above, typing disciplines, say simple typing, in general  
 137 do *not* suffice to be able to avoid  $\alpha$ .)

138 Analysing the safe  $\lambda$ -calculus as presented in [9, 11] using our tools, we found that the  
 139 claim that  $\alpha$  could be avoided in it, was not entirely correct, provoking the further  
 140 analysis, and a proposal for slight amendments, presented below.

#### 141 Related Work.

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

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

#### 159 Outline.

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

---

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

<sup>3</sup> 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.

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

## 2 Preliminaries

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

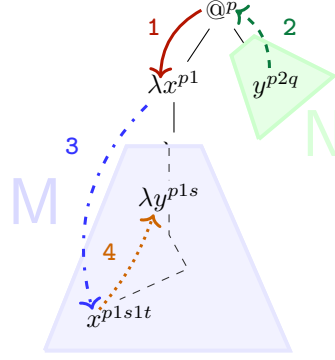
► **Definition 1.** A reduction sequence starting from a  $\lambda$ -term  $M$  is said to be  $\alpha$ -free, if each  $\beta$ -reduction step is  $\alpha$ -free. A  $\lambda$ -term  $M$  has  $\alpha$ -free simulations, if there exists an  $\alpha$ -equivalent  $\lambda$ -term  $N$  such that every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

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

The *position* in a  $\lambda$ -term is a finite sequence of 1s and 2s. The set of positions of a  $\lambda$ -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  $\lambda$ -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  $p \not\preceq q \wedge q \not\preceq p$ . A position  $p$  is set 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  $\blacktriangleright$  to be the relation between positions in the source and in the target of a  $\beta$ -step  $s \rightarrow_\beta t$  contracting a redex at position  $o$  (cf. [36, Section 8.6.1]):



■ **Figure 2** Substitution dynamics leading to variable capture.

199 ■ (context)  $p \blacktriangleright p$  if  $o$  is not prefix of  $p$ ,

200 ■ (body)  $o11p \blacktriangleright op$  if  $p \neq \epsilon$  and  $p \neq q$ ,

201 ■ (arg)  $o2p \blacktriangleright oqp$  for all positions  $q$ , such that  $o11q$  is bound by  $o1$ .

202 A redex in a term  $t$  at position  $q$  is called a *residual* of a redex in some origin  $s$  ( $s \rightarrow_{\beta} \dots \rightarrow_{\beta} t$ ),

203 if  $p \blacktriangleright \dots \blacktriangleright q$  and  $s|_p$  is a redex (cf. [15, Chapter 4, Section 4]).

204 A path  $\sigma = (p_1, p_2, \dots, p_n)$  in a  $\lambda$ -term  $M$  is a sequence of positions in  $\mathcal{Pos}(M)$ . The

205 length  $|\sigma|$  of a path  $\sigma$  is the number of positions minus 1. An *edge* is a path of length 1.

206 The reversal of a path  $\sigma$  is denoted by  $(\sigma)^r$ . Two paths  $\sigma = (p_1, p_2, \dots, p_n)$  and

207  $\psi = (q_1, q_2, \dots, q_n)$  are said to be *composable*, if  $p_n = q_1$ . We write  $\sigma \cdot \psi$  to denote the

208 composition of two (composable) paths  $\sigma, \psi$  resulting in  $(p_1, p_2, \dots, p_n, q_2, \dots, q_n)$ .

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

210 1) @-v, if  $M(p) = @$  and  $M(q) = x$  for some  $x$ .

211 2) @- $\lambda$ , if  $M(p) = @$  and  $M(q) = \lambda x$  for some  $x$ .

212 3) @-@, if  $M(p) = @$  and  $M(q) = @$ .

213 4) v-v, if  $M(p) = x$  and  $M(q) = y$  for some  $x, y$ .

214 5) v- $\lambda$ , if  $M(p) = x$  and  $M(q) = \lambda y$  for some  $x, y$ .

215 To illustrate, let  $M = (\lambda x.x x) (\lambda y z.y z)$ .  $\sigma = (\epsilon, 2, 2112)$  is a @-v-path in  $M$  with  $|\sigma| = 2$ .

216  $\sigma$  and  $(\sigma)^r$  are composable and the path  $(\epsilon, 2, 2112, 2, \epsilon)$  resulting from their composition

217  $\sigma \cdot (\sigma)^r$  is of type @-@.

### 218 3 Developments are $\alpha$ -avoiding

219 Recall that reductions of residuals, also known as *developments*, are finite. This was proved

220 already in 1936 by Church–Rosser for the  $\lambda I$ -calculus [14] and then generalised to the full

221  $\lambda$ -calculus by Schroer [34] and independently by Hindley [20]. It is well known that in *finite*

222 *developments*  $\alpha$ -renaming can be avoided, cf. [24]. Intuitively, this is due to the fact that

223 in developments the residuals of a redex-pattern remain disjoint [22]. Thus, if all binders

224 are initially properly renamed apart,  $\alpha$  can be avoided. To prepare the ground for our main

225 contribution— $\alpha$ -paths—we sketch a purely syntactic proof of this result in this section.

226 We start by giving an intuition for how a capturing-potential can be characterised by

227 paths. A naïve substitution leads to a variable capture whenever we

228 (i) naïvely contract a redex  $(\lambda x.M) N$  where

229 (ii) some variable  $y$  occurring free in  $N$

230 (iii) is moved into  $M$ , where some  $x$  free in  $M$

231 (iv) is in the scope of a  $\lambda y$ .

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

235 Let  $M$  be a  $\lambda$ -term. We conceive the AST of  $M$  as a graph and define four additional  
 236 types of edges for  $M$ :

- 237 1. ( $r$ -edge  $\longrightarrow$ ) A redex-edge  $(p, p1)$  connects an  $\textcircled{\lambda}$ -node at position  $p$  to its left son at  
 238 position  $p1$ , if  $M(p1) = \lambda x$  for some  $x$ .
- 239 2. ( $a$ -edge  $\dashrightarrow$ ) An application-edge  $(p2q, p)$  connects a variable  $x$  at position  $p2q$  to an  
 240  $\textcircled{\lambda}$ -node at position  $p$ , if  $x$  is free in  $M|_{p2}$ .
- 241 3. ( $b$ -edge  $\dashrightarrow$ ) A binding-edge  $(p, p1q)$  connects a  $\lambda x$  at position  $p$  to a variable  $y$  at  
 242 position  $p1q$ , if  $x = y$  and  $y$  is free in  $M|_{p1}$ .
- 243 4. ( $c$ -edge  $\dashrightarrow$ ) A capturing-edge  $(p1q, p)$  connects a variable  $y$  at position  $p1q$  to a  $\lambda x$   
 244 at position  $p$  to, if  $x \neq y$  and  $y$  is free in  $M|_{p1}$ .

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

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



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

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

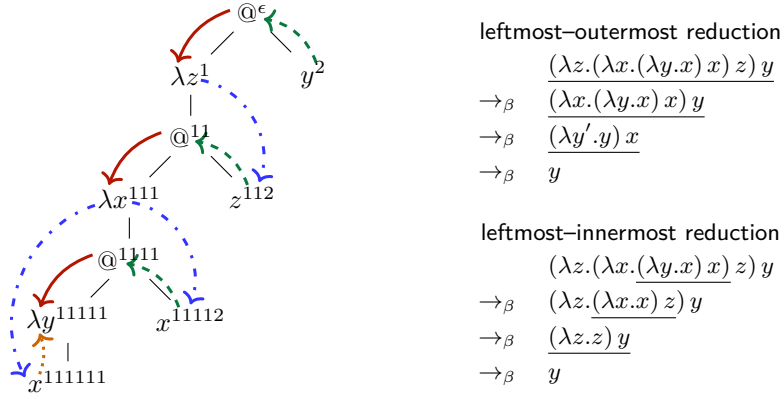
257 ► **Definition 3.** The set of *arbic*-paths of a  $\lambda$ -term  $M$  is inductively defined as follows.

- 258 ■ (base case) Let  $\sigma_{arb}$  be an arb-path of  $M$  and  $c$  a  $c$ -edge in  $G_\alpha(M)$  with  $\sigma_{arb}, c$  composable.  
 259 Then the  $v$ - $\lambda$ -path  $\sigma_{arb} \cdot c$  is an *arbic*-path of  $M$ .
- 260 ■ (arb-composition) Let  $\sigma_{arb}$  be an arb-path and  $\psi$  an *arbic*-path of  $M$  with  $\sigma_{arb}, \psi$  com-  
 261 posable. Then the  $v$ - $\lambda$ -path  $\sigma_{arb} \cdot \psi$  is an *arbic*-path of  $M$ .



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

270 ► **Definition 4 (arbic  $\alpha$ -path).** Let  $M$  be a  $\lambda$ -term and  $\psi$  an *arbic*-path of  $M$ . If  $\psi$  starts  
 271 with a variable  $x$  and ends with a  $\lambda$ -node  $\lambda y$  where  $x = y$ , then  $\psi$  is called an *arbic*  $\alpha$ -path.



■ **Figure 3** Leftmost-outermost needs  $\alpha$ .

272 The path  $\sigma_1$  as defined above is an arbic  $\alpha$ -path for the  $\lambda$ -term illustrated in Figure 3.  
 273 Now, essentially by construction, we can see that if there is no arbic  $\alpha$ -path in  $G_\alpha((\lambda x.M) N)$   
 274 starting at a free variable in  $N$  and ending in  $M$ , then  $M[x := N] \equiv_\alpha M[x := N]$ . We  
 275 emphasise, that we only claim  $\alpha$ -equivalence and not syntactic equivalence ( $=$ ) of  $M[x := N]$   
 276 and  $M[x := N]$ . To clarify, let  $(\lambda x.M) N$  be a redex with  $M = \lambda y.y$  and  $N = y$ . Then  
 277  $M[x := N] = \lambda z.z$  and  $M[x := N] = \lambda y.y$ . We have  $M[x := N] \equiv_\alpha M[x := N]$ , but  
 278  $M[x := N] \neq M[x := N]$ . Hence,  $\alpha$ -equivalence is the strongest property that we can  
 279 conclude.<sup>4</sup>

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

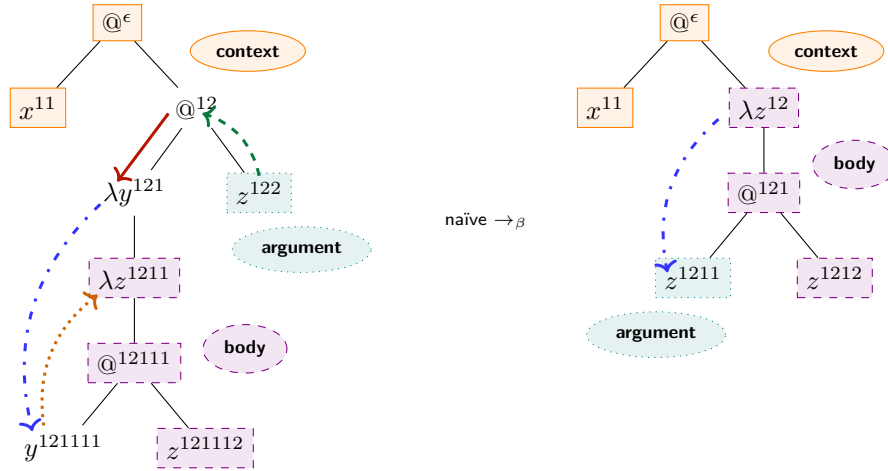
282 **Proof.** We write  $\langle G_\alpha(t) \rangle$  for the sub-graph of  $G_\alpha(t)$ , where the set of r-edges is restricted  
 283 to those denoting residuals of  $s$ . Since there are no arbic  $\alpha$ -paths in  $G_\alpha(s)$ , the  $\beta$ -step can  
 284 be performed by means of capture-permitting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

291 Consider an  $a$ - or a  $c$ -edge from  $p'$  to  $q'$  in  $\langle G_\alpha(t) \rangle$  where  $p'$  denotes the position of a  
 292 variable  $y$  and  $q'$  the position of an application (in the case of an  $a$ -edge) or an abstraction  
 293 (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  
 294 distinguish following cases:

- 295 ■  $p', q'$  in the same component: we have the same edge from  $p$  to  $q$  in  $G_\alpha(s)$ .
- 296 ■  $q'$  in the context and  $p'$  in the body: then  $x \neq y$  (otherwise the  $y$  would have been  
 297 replaced by  $N$ ) and we have the same edge in  $G_\alpha(s)$  with 11 inserted at  $o$ .
- 298 ■  $q'$  in the context and  $p'$  in the argument: there is no variable capture so  $s(p)$  must occur  
 299 free in  $s|_q$ . Therefore, we have the same edge from  $p$  to  $q$  in  $G_\alpha(s)$ .

<sup>4</sup> 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  $\alpha$ -path.

300 ■  $q'$  in the body and  $p'$  in the argument: the origin of the  $a$ -edge/ $c$ -edge is an  $arb$ -path  
 301 from  $p$  to  $qq_1$ , for some  $q_1$ , followed by an  $a$ -edge/ $c$ -edge from  $qq_1$  to  $q$  in  $G_\alpha(s)$ .

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

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

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

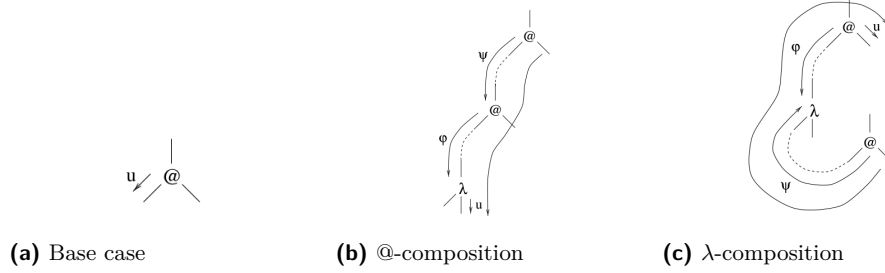
- 313 ■  $p'$  and  $q'$  are in the same component: then we have an  $r$ -edge from  $p$  to  $q$  in  $G_\alpha(s)$ .  
 314 ■ in all other cases: such an  $r$ -edge would denote a created redex in  $t$ . We have no such  
 315  $r$ -edge in  $\langle G_\alpha(t) \rangle$ .

316 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  
 317 in  $G_\alpha(s)$ .  $a$ -edges and  $c$ -edges map back to a path of shape  $\sigma_{arb}^* \cdot e$ , where  $e$  denotes an  
 318 edge of the same type and  $\sigma_{arb}$  an  $arb$ -path in  $G_\alpha(s)$ . An arbic  $\alpha$ -path in  $G_\alpha(t)$  has the  
 319 following shape  $(a'_1, r'_1, b'_1, \dots, a'_n, r'_n, b'_n, c')$ , where  $x_i$  denotes an  $x$ -edge  $(p_i, q_i)$ . If we replace  
 320 the edges in this path by the edges and paths they map back to we get a path of the shape  
 321  $(\sigma_{arb_1}^* \cdot a_1, r_1, b_1, \dots, \sigma_{arb_n}^* \cdot a_n, r_n, b_n, \sigma_{arb}^* \cdot c)$ , which would be an arbic  $\alpha$ -path in  $G_\alpha(s)$ . ◀

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

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

330 ► **Theorem 6.** *In finite developments  $\alpha$  can be avoided.*



■ **Figure 5** Well-balanced paths [3].

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

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

337 **4  $\alpha$ -paths—A Sound Characterisation For  $\alpha$**

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

343 **Legal Paths.**

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

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

- 352 ■ (base case) The path  $(p, p1)$  with  $M(p) = @$  is a wbp.
- 353 ■ (*@-composition*) let  $\psi, \varphi$  be two composable wbps of type @-@ and @- $\lambda$ , respectively. Then  
 354  $\psi \cdot \varphi \cdot u$  is a wbp, where  $u = (p, p1)$  with  $p$  the position of the final abstraction of  $\varphi$ .
- 355 ■ ( *$\lambda$ -composition*) Let  $\varphi = (p, \dots, p_n)$  a wbp of type @- $\lambda$  and  $\psi = \sigma_a \cdot (\sigma_b)^r$  with  $\sigma_a$  a wbp  
 356 of type @-v ending at position  $q$  and  $\sigma_b = (p_n, q)$  a *b*-edge in  $G_{\alpha}(M)$ . Then  $\psi \cdot (\varphi)^r \cdot u$ ,  
 357 where  $u = (p, p2)$ , is a wbp.

358 Legal paths impose a legality constraint on the well-balanced paths, restricting the call  
 359 and return paths of *cycles*. Next, we recall the definition of a cycle. Let  $\varphi$  be a wbp. A  
 360 subpath  $\psi$  of  $\varphi$  is an *elementary @-cycle* of  $\psi$  (over an @-node) when (i) it starts and ends  
 361 with the argument edge of the @-node and (ii) is internal to the argument N of the application  
 362 corresponding to the @-node (i.e., does not traverse any variable that occurs free in N). The  
 363 set of *@-cycles* of  $\varphi$  (over an @-node) and of the *v-cycles* of  $\varphi$  (over the occurrence v of a

variable) is defined inductively, as follows: (i) every elementary @-cycle of  $\zeta$  is an @-cycle; (ii) *v-cycle*: every cyclic subpath of  $\zeta$  of the form  $(v)^r \cdot (\phi)^r \cdot \psi \cdot \phi \cdot v$ , where  $\phi = (p_2, \dots, q_n)$  is a wbp,  $\psi$  is an @-cycle and  $v = (p_1, p_2)$  a *b*-edge, is an *v*-cycle; (iii) *@-cycle*: every subpath  $\psi$  of  $\zeta$  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 *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 @- $\lambda$ , cf. [3].

► **Proposition 7** ([3, Corollary 6.2.26]). *Let  $\psi$  be an @-cycle of  $\phi$  over an @-node. The wbp  $\phi$  can be uniquely decomposed as:  $\phi = \zeta_1 \lambda (\zeta_2)^r @ \psi @ \zeta_3 \lambda \zeta_4$ ,<sup>5</sup> where  $\zeta_2$  (call-path) and  $\zeta_3$  (return-path) are wbps of type @- $\lambda$ .*

Considering the statement of the proposition, the last label of  $\zeta_1$  and the first label of  $\zeta_4$  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  $\lambda$ -term  $M$  there is a legal path of type @- $\lambda$  in  $M$ .*

It follows that for any (created) redex along a reduction sequence starting from a  $\lambda$ -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 $\alpha$ -avoidance via $\alpha$ -paths.

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

► **Definition 10.** *Let  $M$  be  $\lambda$ -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 *v-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.

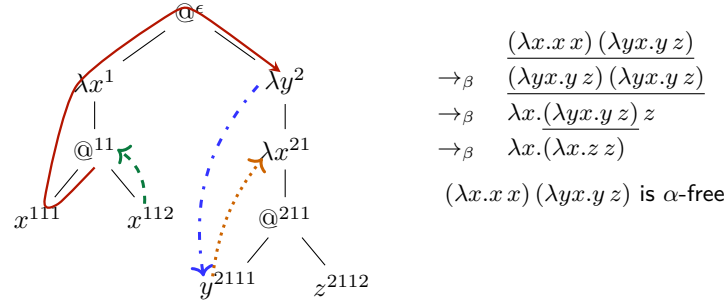
► **Definition 11.** *The set of *albic*-paths of  $M$  is inductively defined:*

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



<sup>5</sup> We use the  $\lambda$ - and @-symbol to point out the start- and end-nodes of the different wbps.

<sup>6</sup> 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.



■ **Figure 6**  $\alpha$ -paths overapproximate the need for  $\alpha$ .

400 Finally, based on Definitions 10 and 11 we can define  $\alpha$ -paths.

401 ► **Definition 12** ( $\alpha$ -path). Let  $\psi$  be an albic-path of  $\lambda$ -term  $M$ . If  $\psi$  starts at a variable  $x$   
402 and ends at a  $\lambda$ -node  $\lambda y$ , where  $x = y$ , then the  $v$ - $\lambda$ -path  $\psi$  is called an  $\alpha$ -path.

403 Inductively, we can conclude that the absence of  $\alpha$ -paths implies that  $\alpha$ -avoidance.

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

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

409 Further,  $\alpha$ -path freeness is preserved by  $\beta$ -reduction.

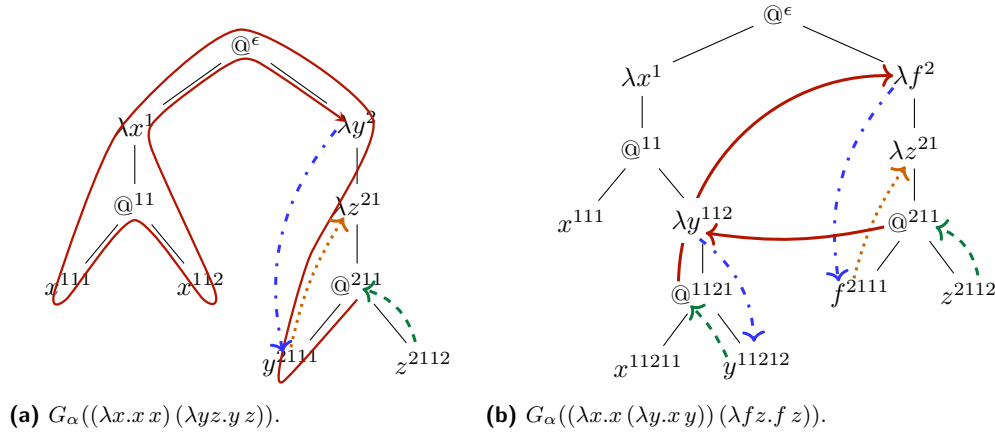
410 ► **Lemma 14** ( $\beta$ -invariance).  $\rightarrow_\beta$  preserves  $\alpha$ -path-freeness.

411 **Proof.** The proof proceeds the same way as the proof of Lemma 5. We restrict ourselves  
412 to the most interesting parts here. Again, we use primed variables ( $p', q'$ ) to range over  
413 positions in the target term  $t$ , indicating the positions they trace back to in the source  
414 term  $s$ , by unpriming ( $p, q$ ). Let  $s \rightarrow_\beta t$ .  $r$ -edges and  $b$ -edges in  $\langle G_\alpha(t) \rangle$  map back to  
415 an edge of the same type in  $G_\alpha(s)$ .  $a$ -edges and  $c$ -edges map back to a path of shape  
416  $\sigma_{alb}^* \cdot e$ , where  $e$  denotes an edge of the same type and  $\sigma_{alb}$  an alb-path in  $G_\alpha(s)$ . An  $\alpha$ -path  
417 in  $t$  has the following shape  $(a'_1, l'_1, b'_1, \dots, a'_n, l'_n, b'_n, c')$ , where  $x_i$  denotes an  $x$ -edge/legal  
418 path from  $p_i$  to  $q_i$ . If we replace  $a$ -edges and  $c$ -edges by the path the map back to we get  
419  $(\sigma_{alb_1}^* \cdot a_1, l_1, b_1, \dots, \sigma_{alb_n}^* \cdot a_n, l_n, b_n, \sigma_{alb}^* \cdot c)$ , where  $\sigma_{alb_i}^* \cdot x_i$  in  $s$  connects the same positions  
420 as the corresponding  $x$ -edge in  $t$ . It follows that if we have an  $\alpha$ -path in  $t$ , then we have an  
421  $\alpha$ -path  $s$ . ◀

422 ► **Theorem 15.** Let  $M$  be a  $\lambda$ -term. If  $M$  contains no  $\alpha$ -path, then  $M$  avoids  $\alpha$ .

423 **Proof.** Assume  $M$  contains no  $\alpha$ -path. Due to Lemma 14,  $\alpha$ -path freeness is preserved by  
424  $\beta$ -reduction. Then it follows by Lemma 13 that capture-permitting substitutions can be  
425 employed in place of capture-avoiding ones. Thus  $M$  avoids  $\alpha$ . ◀

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



■ **Figure 7** Unremovable  $\alpha$ -paths.

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

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

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

#### 443 Undecidability.

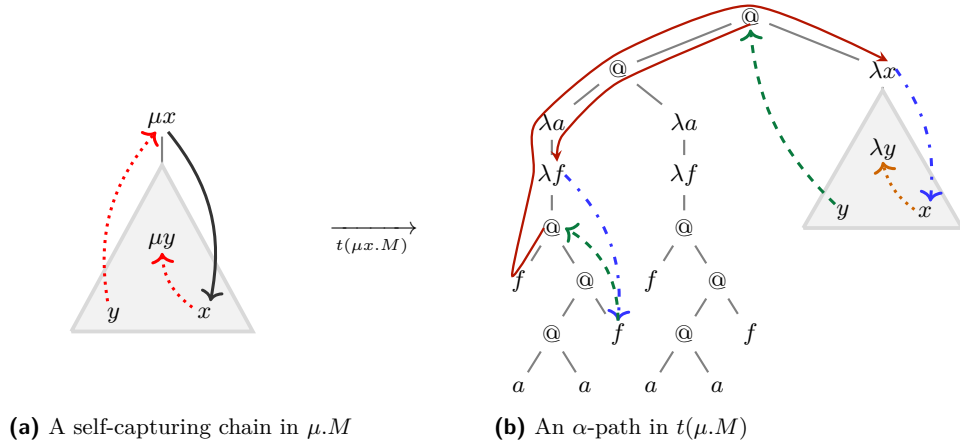
444 Arbitrary  $\lambda$ -terms may have an unbounded set of legal paths, all of them characterising  
 445 a different virtual redex. For such terms, making a prediction about the need for  $\alpha$  via  
 446  $\alpha$ -paths is not feasible. This problem is even undecidable for leftmost–outermost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

448 ► **Theorem 17.**  *$\alpha$ -avoidance is undecidable for the leftmost–outermost reduction strategy.*

449 **Proof.** In proof, we employ a reduction from Post’s correspondence problem (PCP short),  
 450 whose undecidability is well-known [32]. Recall that PCP asks whether for an arbitrary finite  
 451 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  $\{a, b\}$ , there exists indices  
 452  $i_j \in \{1, 2, \dots, n\}$  such that

$$453 \quad s_{i_1} s_{i_2} \dots s_{i_k} = s'_{i_1} s'_{i_2} \dots s'_{i_k} \quad k \geq 1.$$

454 It is not difficult to define  $\lambda$ -terms for (i) strings **aa**, **bb**, namely  $AA := \lambda a b x.a(a x)$  and  
 455  $BB := \lambda a b x.b(b x)$ , respectively; (ii) *conditionals* (denoted as *ITE*); (iii) *pairs* (*PAIRS*) and  
 456 (iv)  $n$  particular PCP (*PCP*), such that the  $\lambda$ -term *PCP* takes an (encoding) of list of pairs  
 457 as input and recursively combines them, until a solution is produced (if it exists at all). As



■ **Figure 8** A self-capturing chain in  $\mu$  is an  $\alpha$ -path in  $\Lambda_\mu$ .

458 the leftmost–outermost strategy is normalising for the  $\lambda$ -calculus, this solution can be found  
 459 by this strategy. Now, consider the following program

460  $(ITE (PCP PAIRS) AA BB) (\lambda xyz.(xz) y)$ ,

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

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

472 In the sequel, we clarify the ancestry of  $\alpha$ -paths wrt. the concept of *chains* in the  $\mu$ -  
 473 calculus, cf. [17]. Further, we briefly detail our tool **Alpha** that can be used to compute and  
 474 illustrate  $\alpha$ -paths.

#### 475 Interpretation of $\mu$ in the $\lambda$ -calculus.

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

$$\begin{aligned}
 481 \quad t(\mu x.M) &:= AA(\lambda x.t(M)) \rightarrow_\beta (\lambda f.f(AAf))(\lambda x.t(M)) \rightarrow_\beta (\lambda x.t(M))(AA(\lambda x.t(M))) \\
 482 &\rightarrow_\beta t(M)[x := AA(\lambda x.t(M))] = t(M)[x := t(\mu x.M)] \\
 483 &= t(M)[x := \mu x.M].
 \end{aligned}$$

484 In  $\Lambda_\mu$  we can use  $\alpha$ -paths to characterise  $\alpha$ . We sketch the argument that an  $\alpha$ -path in  
 485  $t(M)$  for some  $\mu$ -term  $M$  correspond a self-capturing chain in  $M$ . Note that, as reducing

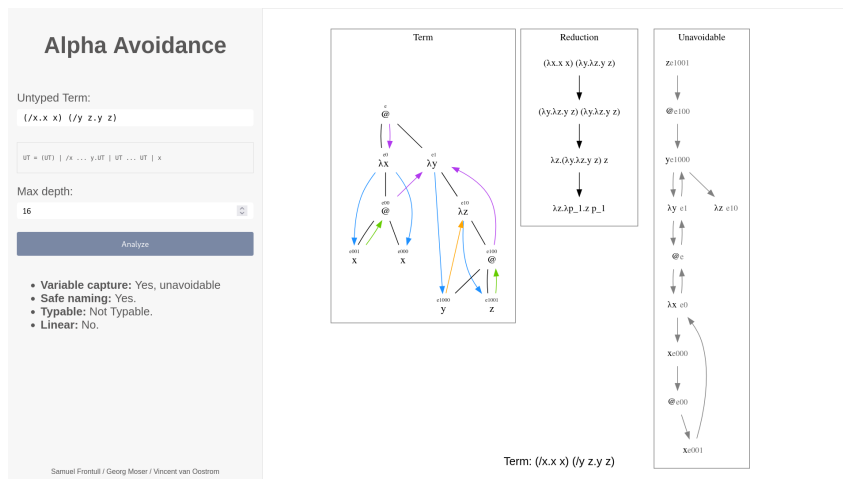


Figure 9  $\alpha$ -avoidance tool web-interface.

486 Turing's fixed point combinator  $AA$  itself does not cause any capturing problems, we do not  
 487 introduce "new  $\alpha$ -problems". Thus, we only need to characterise the paths that correspond  
 488 to reductions at the root of a reduct of  $AA(\lambda x.M)$  to characterise the need for  $\alpha$  in  $\Lambda_\mu$ .  
 489 For that, we observe that a pair of connected binding and capturing links in  $\mu$  correspond  
 490 to an *alb*-path in  $\Lambda_\mu$  and a self-capturing chain to an  $\alpha$ -path. Figure 8 illustrates this  
 491 correspondence.

### 492 Implementation.

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

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

- 499 1) **alpha free**, if no  $\alpha$ -paths were found and the calculation is terminated;
  - 500 2) **alpha can be avoided**, if  $\alpha$ -paths were found (but no unremovable  $\alpha$ -paths); in this  
 501 case, the tools prints an  $\alpha$ -equivalent term for which the calculation is  $\alpha$ -free;
  - 502 3) **alpha is unavoidable**, if unremovable  $\alpha$ -paths have been found;
- 503 or returns **maybe**, if no  $\alpha$ -paths have been found, but the computation has not been terminated  
 504 (the maximum depth has been reached). Recall that the problem is undecidable, cf. Section 4.<sup>7</sup>  
 505 Listing 1 shows an exemplary output of the command line tool.

### Listing 1 Church encoding of $3^2$

```
506 $ dune exec bin/main.exe "(/f0x.f0(f0x))0(/f0x.f0(f0(f0x))) "
```

507 alpha can be avoided:

```
508 (/f x.f (f x)) (/f p_12.f (f (f p_12)))
```

509

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

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

513 **5**  $\alpha$ -avoidance in Affine, Safe and Weak  $\lambda$ -Calculi

514 In this section, we show how  $\alpha$ -paths can be applied to analyse the need for  $\alpha$  in restricted  
515  $\lambda$ -calculi.

516 **5.1 The affine  $\lambda$ -calculus**

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

520 **► Definition 18.** *The set  $\Lambda_{AFF}$  of affine  $\lambda$ -terms is a subset of  $\Lambda$  and inductively defined as*  
521 *follows:*

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

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

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

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

- 532 1.  $p, q$  both in the context: Then as  $p' = p$  and  $q' = q$  so by assumption we have  $p' \prec q'$ .
- 533 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  
534 we have  $os_1 = p' \prec q' = os_2$ .
- 535 3.  $p = o2s_1$  and  $q = o2s_2$  both in the argument: Then from  $p \prec q$  we know that  $s_1 \prec s_2$   
536 and we have  $ots_1 = p' \prec q' = ots_2$ .
- 537 4.  $p$  is in the context and  $q = o11s$  in the body. Then  $p' = p$  and  $q' = os$  and since  $p \prec q$  we  
538 also have  $p' \prec q'$ .
- 539 5.  $p$  is in the context and  $q = o2s$  in the argument. Then  $p' = p$  and  $q' = oqs$ . Since we  
540 know that  $p \prec o$  (because it is in the context), we also have  $p' \prec q'$ .

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

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

548 **► Lemma 20.** *Let  $M$  be an arbitrary term in  $\Lambda_{AFF}$ . There are no unremovable  $\alpha$ -paths  
549 in  $M$ .*

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

551 **► Theorem 21.** *In the affine  $\lambda$ -calculus  $\alpha$  can be avoided.*

552 **Proof.** Due to Lemma 20 it only remains to prove that for every affine  $\lambda$ -term  $M$  there  
553 exists an affine  $\lambda$ -term  $N$  such that  $M \equiv_{\alpha} N$  and  $N$  avoids  $\alpha$ . This, however, follows from  
554 Lemma 19 in conjunction with Lemma 16.  $\blacktriangleleft$



$$\begin{array}{c}
\text{(var)} \frac{}{x : A \vdash_s x : A} \quad \text{(const)} \frac{}{\vdash_s f : A} \quad f : A \in \Xi \quad \text{(wk)} \frac{\Gamma \vdash_s M : A}{\Delta \vdash_s M : A} \quad \Gamma \subset \Delta \quad \text{(\delta)} \frac{\Gamma \vdash_s M : A}{\Gamma \vdash_{asa} M : A} \\
\text{(app}_{asa}\text{)} \frac{\Gamma \vdash_{asa} M : A \rightarrow B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_{asa} M N : B} \quad \text{(app)} \frac{\Gamma \vdash_{asa} M : A \rightarrow B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_s M N : B} \quad \text{ord } B \leq \text{ord } \Gamma \\
\text{(abs)} \frac{\Gamma, x_1 : A_1, \dots, x_n : A_n \vdash_{asa} M : B}{\Gamma \vdash_s \lambda x_1^{A_1} \dots x_n^{A_n}. M : (A_1, \dots, A_n, B)} \quad \text{ord } (A_1, \dots, A_n, B) \leq \text{ord } \Gamma
\end{array}$$

■ **Figure 10** The safe  $\lambda$ -calculus [9].

## 5.2 The safe $\lambda$ -calculus

555

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

563

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

568

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

574

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

576

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

578

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

586

In the safe  $\lambda$ -calculus, consecutive redexes are contracted simultaneously, as the standard  
587  $\beta$ -reduction does not preserve safety [9, Section 3.1.2]. This requires a notion of simultaneous  
588 substitution. The definitions of simultaneous capture-permitting and simultaneous capture-  
589 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)
$x_i$	$N_i$	$N_i$
$y$	$y$	$y$
$e_1 e_2$	$e_1[\overline{N}/\overline{x}] e_2[\overline{N}/\overline{x}]$	$e_1[\overline{N}/\overline{x}] e_2[\overline{N}/\overline{x}]$
$(\lambda \overline{y}.e)[\overline{N}/\overline{x}]$	$\lambda \overline{y}.e[\overline{N}'/x']$ 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 $e, \overline{N}$	$\lambda \overline{y}.e[\overline{N}'/x']$ where $\overline{x}' = \overline{x} - \overline{y}$

590 ► **Definition 24** (safe redex [9, Definition 3.21]). *An untyped safe redex is an untyped almost*  
 591 *safe application (a succession of several standard redexes) of the form  $(\lambda x_1 \dots x_n.M) N_1 \dots N_l$*   
 592 *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.*

593 ► **Definition 25** (safe redex contraction). *The relation  $\beta_s$  is defined on the set of safe redexes*  
 594 *as follows:*

$$595 \quad \beta_s = \{(\lambda x_1 \dots x_n.M) N_1 \dots N_l \mapsto (\lambda x_{l+1} \dots x_n.M)[N_1 \dots N_l/x_1 \dots x_l] \mid n > l\}$$

$$596 \quad \cup \{(\lambda x_1 \dots x_n.M) N_1 \dots N_l \mapsto M[N_1 \dots N_n/x_1 \dots x_n] N_{n+1} \dots N_l \mid n \leq l\}$$

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

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

601 ► **Definition 26.** *The safe  $\beta$ -reduction, written as  $\rightarrow_{\beta_s}$ , is the compatible closure of the*  
 602 *relation  $\beta_s$  with respect to the formation rules of the safe  $\lambda$ -calculus.*

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

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

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

616 A more restrictive set of rules is needed to resolve this issue. These rules are depicted  
 617 in Figure 11.<sup>10</sup> In this system we dropped the "almost safety" and allow to type only

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

<sup>9</sup> Compare to the errata published at <https://github.com/blumu/dphil.thesis/blob/erratum/Current/thesis-erratum/dphilerratum.pdf>.

<sup>10</sup> Simultaneous substitutions coincide with the singleton substitutions from Table 1 in the case  $|\overline{x}| = 1$ .

$$\begin{array}{c}
\text{(var)} \frac{}{\{x : A\} \vdash_{s\alpha} x : A} \quad \text{(const)} \frac{}{\vdash_{s\alpha} f : A} \quad f : A \in \Xi \quad \text{(wk)} \frac{\Gamma' \vdash_{s\alpha} M : A}{\Gamma \vdash_{s\alpha} M : A} \quad \Gamma' \subset \Gamma \\
\text{(app)} \frac{\Gamma \vdash_{s\alpha} M : (A_1, \dots, A_n, B) \quad \Gamma_{\geq m} \vdash_{s\alpha} N_1 : A_1 \quad \dots \quad \Gamma_{\geq m} \vdash_{s\alpha} N_j : B_j}{\Gamma \vdash_{s\alpha} M N_1 \dots N_j : B} \quad m = \text{ord } B \\
\text{(abs)} \frac{\Gamma_{\geq m} \cup \{x_1 : A_1, \dots, x_n : A_n\} \vdash_{s\alpha} M : B}{\Gamma \vdash_{s\alpha} \lambda x_1 \dots x_n. M : (A_1, \dots, A_n, B)} \quad m = \text{ord}(A_1, \dots, A_n, B)
\end{array}$$

■ **Figure 11** An  $\alpha$ -avoiding safe  $\lambda$ -calculus.

618 applications that provide enough arguments to abstractions. More precisely, if an argument  
619 of order  $k$  is provided, the arguments of all abstracted variables of order  $k$  and higher must  
620 be provided. In this way, we avoid free variables ending up in the scope of abstractions of the  
621 same order during reduction. This avoids potential variable capture, since it can be assumed  
622 that free variables are always of a higher order than the abstractions they enter the scope of.  
623 Therefore, according to the safe variable typing convention, they are named differently.<sup>11</sup>

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

628 In the following Lemma 29 we show that the safe  $\lambda$ -calculus of Figure 11 avoids  $\alpha$  by  
629 reasoning with  $\alpha$ -paths. This can be done by interpreting safe  $\lambda$ -terms as ordinary terms.

630 ► **Theorem 29.** *In the safe  $\lambda$ -calculus no variable capture can occur, provided that the safe*  
631 *variable typing convention is adopted.*

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

### 641 5.3 The weak $\lambda$ -calculus

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

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

646 ► **Definition 30** (*weak  $\lambda$ -reduction [39, Definition 3.1]*). A particular occurrence of a redex  $R$   
 647 in a  $\lambda$ -term  $M$  will be called *weak in  $M$*  iff no variable-occurrence free in  $R$  is bound in  $M$ .  
 648 A weak  $\beta$ -contraction in  $M$  is the contraction of a  $\beta$ -redex-occurrence that is weak in  $M$ .

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

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

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

666 In sum,  $\alpha$ -avoidance is immediate from the definitions.

667 ► **Theorem 32.** *In the weak  $\lambda$ -calculus  $\alpha$  can be avoided.*

## 668 **6 Conclusion**

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

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

688 ——— **References** ———

- 689 1 Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus.  
690 *Information and Computation*, 105(2):159–267, 1993. doi:10.1006/inco.1993.1044.
- 691 2 Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. Paths in the lambda-  
692 calculus. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science*  
693 *(LICS '94), Paris, France, July 4-7, 1994*, pages 426–436. IEEE Computer Society, 1994.  
694 doi:10.1109/LICS.1994.316048.
- 695 3 Andrea Asperti and Stefano Guerrini. *The optimal implementation of functional programming*  
696 *languages*, volume 45 of *Cambridge tracts in theoretical computer science*. Cambridge University  
697 Press, 1998.
- 698 4 Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the  $\lambda$ -calculus.  
699 *Theoretical Computer Science*, 142(2):277–297, 1995. doi:10.1016/0304-3975(94)00279-7.
- 700 5 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and  
701 Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming,*  
702 *ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 263–274. ACM, 2013. doi:  
703 10.1145/2500365.2500606.
- 704 6 Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies*  
705 *in Logic and the Foundations of Mathematics*. North-Holland, 2nd revised edition, 1984.  
706 doi:10.2307/2274112.
- 707 7 Henk P. Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types.*  
708 *Perspectives in logic*. Cambridge University Press, 2013.
- 709 8 Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the Weak Lambda-Calculus.  
710 In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer,  
711 editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan*  
712 *Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer*  
713 *Science*, pages 70–87. Springer, 2005. doi:10.1007/11601548\_7.
- 714 9 William Blum. *The Safe Lambda Calculus*. PhD thesis, Oxford University, UK, 2009.
- 715 10 William Blum and C.-H. Luke Ong. The Safe Lambda Calculus. In *TLCA*, pages 39–53,  
716 Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 717 11 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer*  
718 *Science*, Volume 5, Issue 1, February 2009. doi:10.48550/arXiv.0901.2399.
- 719 12 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for  
720 automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes*  
721 *Mathematicae*, 75(5):381–392, 1972. Proc. 19th International Conference on Automated  
722 Deduction. doi:10.1016/1385-7258(72)90034-0.
- 723 13 Felice Cardone and J. Roger Hindley. Lambda-Calculus and Combinators in the 20th Century.  
724 In *Logic from Russell to Church*, 2009. doi:10.1016/S1874-5857(09)70018-4.
- 725 14 Alonzo Church and John Barkley Rosser. Some properties of conversion. *Trans-*  
726 *actions of the American Mathematical Society*, 39:472–482, 1936. doi:10.1090/  
727 S0002-9947-1936-1501858-0.
- 728 15 Haskell B. Curry. *Combinatory Logic*. Amsterdam: North-Holland Pub. Co., 1958.
- 729 16 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20(2):95–207,  
730 1982. doi:10.1016/0304-3975(82)90009-3.
- 731 17 Jörg Endrullis, Clemens Grabmayer, Jan Willem Klop, and Vincent van Oostrom. On equal  
732  $\mu$ -terms. *Theoretical Computer Science*, 412(28):3175–3202, 2011. doi:10.1016/j.tcs.2011.  
733 04.011.
- 734 18 Samuel Frontull. Alpha Avoidance. Master’s thesis, University of Innsbruck, 2021.
- 735 19 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders.  
736 In *Proc. 14th LICS*, pages 214–224, 1999. doi:10.1109/LICS.1999.782617.
- 737 20 J. Roger Hindley. Reductions of Residuals are Finite. *Transactions of the American Mathem-*  
738 *atical Society*, 240:345–361, 1978. doi:10.2307/1998825.

- 739 **21** J. Roger Hindley. BCK-Combinators and Linear lambda-Terms have Types. *Theoretical*  
740 *Computer Science*, 64(1):97–105, 1989. doi:10.1016/0304-3975(89)90100-X.
- 741 **22** Martin Hyland. A simple proof of the Church–Rosser theorem. Oxford University, UK, 1973.
- 742 **23** Bart Jacobs. Semantics of lambda-I and of other substructure lambda calculi. In Marc Bezem  
743 and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 195–208, Berlin,  
744 Heidelberg, 1993. Springer Berlin Heidelberg. doi:10.1007/BFb0037107.
- 745 **24** Jan W. Klop. *Combinatory reduction systems*. PhD thesis, Rijksuniversiteit Utrecht, 1980.
- 746 **25** Dexter Kozen. Results on the propositional  $\mu$ -calculus. *Theoretical Computer Science*, 27:333–  
747 354, 1983. doi:10.1016/0304-3975(82)90125-6.
- 748 **26** Clemens Kupke, Johannes Marti, and Yde Venema. Size measures and alphabetic equivalence  
749 in the  $\mu$ -calculus. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual*  
750 *ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*,  
751 pages 18:1–18:13. ACM, 2022. doi:10.1145/3531130.3533339.
- 752 **27** Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional*  
753 *Programming*, 14(6):623–633, 2004. doi:10.1017/S0956796804005131.
- 754 **28** Jolie G. de Miranda. *Structures generated by higher-order grammars and the safety constraint*.  
755 PhD thesis, University of Oxford, UK, 2006.
- 756 **29** Maxwell H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals*  
757 *of mathematics*, 43(2):223–243, 1942. doi:10.2307/1968867.
- 758 **30** Vincent van Oostrom and Roel de Vrijer. Four equivalent equivalences of reductions. *Electronic*  
759 *Notes in Theoretical Computer Science*, 70(6):21–61, 2002. WRS 2002, 2nd International  
760 Workshop on Reduction Strategies in Rewriting and Programming - Final Proceedings (FLoC  
761 Satellite Event). doi:10.1016/S1571-0661(04)80599-1.
- 762 **31** Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice  
763 Hall, January 1987.
- 764 **32** Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American*  
765 *Mathematical Society*, 52(4):264 – 268, 1946. doi:10.2307/2267252.
- 766 **33** John Barkley Rosser. Review: H. B. Curry, A New Proof of the Church–Rosser Theorem.  
767 *Journal of Symbolic Logic*, 21(4):377–378, 1956.
- 768 **34** David E. Schroer. *The Church–Rosser Theorem*. PhD thesis, Cornell University, 1965.
- 769 **35** Rick Statman. On the complexity of alpha conversion. *The Journal of Symbolic Logic*,  
770 72(4):1197–1203, 2007. doi:10.2178/jsl1/1203350781.
- 771 **36** Terese. *Term rewriting systems*. Cambridge University Press, 2003.
- 772 **37** Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis,  
773 University of Oxford, 1971.
- 774 **38** Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. *Journal of*  
775 *Functional Programming*, 26:e21, 2016. doi:10.1017/S095679681600023X.
- 776 **39** Naim Çağman and J. Roger Hindley. Combinatory weak reduction in lambda calculus.  
777 *Theoretical Computer Science*, 198(1-2):239–247, 1998. doi:10.1016/S0304-3975(97)00250-8.