The Safe Lambda Calculus - D.Phil. Erratum

William Blum

November 29, 2021

1 August 19, 2021, updated November 29, 2021

1. Remove lines 12-22 page 54. (Reported by Samuel Frontull). The remark that a version of the 'no-variable capture lemma' holds without the safe typing convention 3.13 (Lemma 3.17) is false and the argument is incorrect.

2. Small typo on line 2 with extra character 'm' in Definition 3.23.

3. A remark should be added to explain that in the safe lambda calculus (as defined in the thesis) substitution can indeed be implemented capture-permitting provided that the terms substituted is safe. When contracting a safe redex, however, capture-avoiding substitution is needed in the case $n > l$ (where there are less arguments than abstractions) if the body of the lambda in the redex is not itself safe.

   Definition 3.23 and Lemma 3.24 (contracting a safe redex preserves safety) remain valid but just like the TLCA paper mentions, we take the convention that substitution is implemented capture-permitting only when the substituted term itself is safe.

   In the stricter 'long' fragment of the safe calculus from Def 3.23, however, safe redexes can always be contracted with capture-permitting substitution. Indeed, the problematic case where the body of the lambda is unsafe cannot happen: Take the safe redex $R = (\lambda x_1..x_l x_{l+1}..x_n.M)N_1 \dots N_l$ so that $\Gamma \vdash R : T$ is a long safe term and M and each of the N_i are long-safe. Then consider the potentially problematic case $n > l$ and let $Q = \lambda x_{l+1}..x_n.M$ so that Q is also a term of type T. The definition of safe reduction gives: $R \to_{\beta_s} \lambda x_{l+1}..x_n.M[N_1..N_l/x_1..x_l]$. For a variable capture to occur in the substitution there must be a k and j such that $1 <= j <= l < k <= n$ where x_k occurs freely in N_j. But then x_k also occurs freely in R and therefore $|x_k| > |T| = |Q|$. But since Q takes an argument x_k we must have $|Q| > |x_k|$ which is a contradiction.

   Take Samuel Frontull's example $y : o \vdash (\lambda x^0 y^0.x)y$. It is a safe-redex (It is not a safe term but can occur within a larger safe term). The body $\lambda y^0.x$ of the redex is not safe itself so the safe redex cannot be contracted with capture-permitting substitution. However the η-equivalent term $\lambda x^o.(\lambda x^0 y^0.x)yz$ is also a safe-redex and is a long-safe term which can therefore be contracted with capture-permitting substitution without causing variable clashes.

   There are also other syntactic restrictions that eliminates the problematic case where capture-avoiding substitution is needed; homogeneity is one of them for instance.

4. *Remark about the two versions of the safe calculus:* In the thesis, we define 'Safe redex' and 'Safe beta reduction' on a more relaxed version of safe lambda calculus than the calculus I originally introduced in the TLCA paper, which I call 'long safe' in the thesis. When restricted to the long-safe fragment (Definition 3.31), the definitions coincide with TLCA paper.

   Recall that in the long safe system, the abstraction rule requires the body of the lambda to be a safe term, unlike the more relaxed version of Table 3.1 where the body can just be an 'almost safe application'.

   The rules from Table 3.1 capture the 'incremental binding of variables' aspect of the original safety restriction (i.e., if you consider AST tree where consecutive lambdas are merged into a single bulk AST node, then for safe terms you can retrieve the binder node of any variable by traversing the path from the variable to the root of the AST and stopping at the first lambda node with order greater than the order of the variable.) This is the version that gets studied in the last chapter of the thesis, and the one that gets characterized semantically by 'P-incremental' strategies.

   Observe that a long safe term that is not safe can always be turned into an eta-equivalent safe term: by just eta-expanding the body of abstractions that are *almost* safe applications. Such eta-expansion has the side-effect of instantiating fresh variable names, which partially defeats the benefit of the safety restriction. So in a way, the safety terms from Table 3.1 are not 'as safe' as the one from Def 3.31.

   Yet, it is still the preferred version of the calculus in the thesis because, contrary to the stricter long variant, this definition is sound with respect to eta-reduction (Proposition 3.37). This becomes important in Chapter 6 where I look at the game semantics model of the calculus. Such models are extensional (eta-expansion or eta-reduction do not alter the semantics of a term) and therefore they cannot possibly capture syntactic differences between long-safe term and safe terms. Hence, the game semantic characterization result from Chapter 5 (Theorem 5.19) concerns the more relaxed version of the calculus from Table 3.1 rather than just long-safety. The characterization result, however, implies that long safe terms are also denoted by P-incremental strategies.

   The variable-capture avoidance guarantees from 'Lemma 3.15' (pertaining to substitution rather than redex contraction) still apply to both versions of the calculi. But if one looks for a safe calculus where contracting safe redexes can always be done with capture permitting substitution, then long-safety is the one to pick. On the other hand, the definition from Table 3.1 is more appropriate for a semantic analysis of safety.

2 Acknowledgment

Credits to Samuel Frontull who reported conter-examples that led to this erratum.

3 Reference

[1] D.Phil. Thesis: The Safe Lambda Calculus 2009, Oxford University Research Archive `https://ora.ox.ac.uk/objects/uuid:537d45e0-01ac-4645-8aba-ce284ca02673/download_file?file_format=pdf&safe_filename=Blum%2Bthesis&type_of_work=Thesis`

[2] TLCA paper: The Safe Lambda Calculus (extended version) With C.-H. Luke Ong. Technical report. `https://william.famille-blum.org/research/tlca07-long.pdf`