Don't *bind* yourself to *do* notation!

Timen Zandbergen[1] Responsible Professor: Jesper Cockx[1] Supervisor: Luka Miljak[1]

[1]EEMCS TU Delft

Background

- Refactoring is the practice of changing the code of a program without changing its observed behavior[1].
- Many automated refactorings contain bugs, even in widely used IDEs[2].
- Formal verification is cheaper than traditional methods of creating high assurance systems[3].
- Agda is a proof assistant and programming language, using dependent types to specify properties[4].

Proof of correctness

$$✓ : \gamma \models L \downarrow v \to (\text{rmDoEnv } \gamma) \models (\text{rmDo } L) \downarrow (\text{rmDoValue } v)$$

γ an environment over Γ
Γ a typing context
L a language construct ($\Gamma \vdash$ A)
A some type in the HLL
v some value of type A

Figure 1. The function signature of the proof of correctness. It proves that for any language construct that reduces to a certain value, the refactored version reduces to either the same value or a closure with the refactoring applied to it.

The proof of equivalence pre- and post- refactoring. `rmDoValue` is needed because closures may have a modified body. `rmDoEnv` is needed because a closure could either originate from the environment outside the refactor, or be constructed within the refactored program.

Because `rmDoEnv` and `rmDoValue` only affect the bodies of closures, and ✓ proves that all non-closure values are unaffected by the refactoring, the resulting closures are also contextually equivalent.

Example code

```
bindChain : ∅ ⊢ 𝕋maybe 𝕋n
bindChain =
  Just (num 1) >>=
  (ƛ (Just (num 1) >>=
  (ƛ Just (# 1 + # 0))))

doChain : ∅ ⊢ 𝕋maybe 𝕋n
doChain =
  do← Just (num 1) ⁀
  do← Just (num 1) ⁀
  Just ( # 1 + # 0)

exChain : bindChain ≡ (rmDo doChain)
exChain = refl
```

Figure 2. Two example functions written in the Haskell-like language

An example function pre- and post- refactoring. These two programs both return the same value, `num 2`, but use different constructs to go about it.

The representation used is intrinsically well typed, making it impossible to create a construct that has a runtime type error. This is achieved using de Bruijn representation[5], alleviating the problems associated with string names.

Key takeaways

- Big-step reduction more aligned with proof construction than small-step reductions.
- Environments facilitate cleaner induction.
- Closures are modified, meaning an equivalence needs to be established.
- Converting the reduction avoids problems with determinism and terms.

Future work

- Support a larger subset of Haskell.
- Support more refactorings, compose them to larger refactorings[6].
- Correct parser and printer rather than modify only AST.

References

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. USA: Addison-Wesley Longman Publishing Co., Inc., 1999, ISBN: 0201485672.

[2] M. Gligoric, F. Behrang, Y. Li, J. Overbey, M. Hafiz, and D. Marinov, "Systematic testing of refactoring engines on real software projects," in *ECOOP 2013 – Object-Oriented Programming*, G. Castagna, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 629–653, ISBN: 978-3-642-39038-8.

[3] G. Klein, J. Andronick, K. Elphinstone, *et al.*, "Comprehensive formal verification of an os microkernel," *ACM Trans. Comput. Syst.*, vol. 32, no. 1, Feb. 2014, ISSN: 0734-2071. DOI: 10.1145/2560537.

[4] Agda Development Team, *Agda 2.6.3 documentation*, 2023. [Online]. Available: https://agda.readthedocs.io/en/v2.6.3/.

[5] N. de Bruijn, "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem," *Indagationes Mathematicae*, vol. 75, no. 5, pp. 381–392, 1972. DOI: 10.1016/1385-7258(72)90034-0.

[6] D. Horpácsi, J. Kőszegi, and S. Thompson, "Towards trustworthy refactoring in erlang," in *Proceedings of the Fourth International Workshop on Verification and Program Transformation, Eindhoven, The Netherlands, 2nd April 2016*, G. Hamilton, A. Lisitsa, and A. P. Nemytykh, Eds., ser. Electronic Proceedings in Theoretical Computer Science, vol. 216, Open Publishing Association, 2016, pp. 83–103. DOI: 10.4204/EPTCS.216.5.

T.Zandbergen-2@student.tudelft.nl