Maybe a List would be better?: Correct by construction Maybe to List refactorings in a Haskell-like language

CSE3000 Research Project
José Carlos Padilla Cancio (j.c.padillacancio@student.tudelft.nl)
Responsible Professor: Jesper Cockx | Supervisor: Luka Miljak

TU Delft Delft University of Technology

1. Introduction

- Functional languages have referential transparency, i.e. **expressions correspond to a single value**
- Dependently typed languages, a form of functional programming, allow you to **write and verify proofs using their type checker.**
- You can have your cake and eat it too! **Write your function and formally verify it.**
- Goal of this RP is to apply this to refactorings in a **Haskell Like Language** (HLL) using the dependently typed language **Agda** [1]

```
safe-divM : Maybe Int -> Maybe Int -> Maybe Int
safe-divM _         (Just 0) = Nothing
safe-divM (Just x)  (Just y) = Just (x / y)
safe-divM _     _ = Nothing
```

→

```
safe-divL : [Int] -> [Int] -> [Int]
safe-divL _         [0] = []
safe-divL [x]   [y] = [x / y]
safe-divL _     _ = []
```

Figure 1. Example refactoring

2. Objectives

1. "Define a Haskell-like language in Agda"

2. "Write the refactorings of Maybe to List and formally verify its correctness"

3. Current work

Language Design choices

- Intrinsically-typed language
  - **Define type system and syntax at the same time**
- De Bruijn indices
  - **Variable references are all unique**
- Big step semantics
  - **Define pre-conditions and abstract small steps**

```
case x of
    Nothing -> e_n
    Just i -> e_j
```

Figure 2. Problematic refactoring

Refactoring

- Intrinsic typing
  - **Refactoring function is proof of well-typedness**
- Refactoring introduces new additions to Environment/Context
  - **Use Extend_Under data type to keep track of changes**

Proof

- Values can change post refactoring
  - Define $\mapsto_r$ relation to define how values should change
- Closure values may have different environment lengths and have different bodies
  - Use "weak extensional equivalence" to define valid closure values post refactoring.
  - "Equivalent" inputs give "equivalent" outputs

4. Primary takeaways

Techniques generalize well to other (data-oriented) refactorings

- Extend_Under_ data type
  - Keeps track of how structure of context changes
- $\mapsto_r$ relation
  - Defines "valid" updates for data-oriented refactorings
  - Definition in the case of closures weak but sufficient
- Intrinsic typing
  - Implicitly handles well-typedness

5. References

[1] Ana Bove, Peter Dybjer, and Ulf Norell. "A Brief Overview of Agda-A Functional Language with Dependent Types." In: TPHOLs. Vol. 5674. Springer. 2009, pp. 73–78.

[2] Philip Wadler, Wen Kokke, and Jeremy G. Siek. Programming Language Foundations in Agda. Aug. 2022. url: https://plfa.inf.ed.ac.uk/22.08/.