Proving correctness of refactoring tuples to records

A correct-by-construction approach on a Haskell-like language

Jeroen Bastenhof – J.Bastenhof@student.tudelft.nl

1. Agda

- Dependently typed functional programming language.
- Can be used as a proof assistant.
- Formal verification.

2. Motivation

- How to ensure a refactoring is correct?
- Extensive testing increases confidence, but provides no explicit guarantees as opposed to a formal verification [1].
- Less attention for functional languages as opposed to class-based, object-oriented languages [2].

3. Refactoring tuples to records

- Tuples are less explicit compared to records.
- Record field accessors provide layer of abstraction.
- Not possible to distinguish tuples during refactoring when they have the same signature, but different representation.
- Is proving the correctness of tuple to record refactoring feasible?

```haskell
-- Before: less expressive and extensible
lastName :: (String, String, Int) -> String
lastName (_, s, _) = s

-- After: more expressive and extensible
data Person = Person { initials :: String
                     , lastName :: String
                     , age :: Int
                     }
```

Listing 1. Tuple to record refactoring example based on work by Miran [3].

4. The Haskell-like language (HLL)

- Intrinsically-well-typed language constructed in Agda that forms the foundation of the refactoring operation and the proofs.

```
data _,_⊢_ : Ctx → DataCtx → Type → Set where
    ...
    tuple : TypeResolver Γ Γᵈ ts
        ------------------------
        → Γ , Γᵈ ⊢ tupleT ts
    tLookup : Γ , Γᵈ ⊢ tupleT ts → t ∈ ts
        ---------------------------------
        → Γ , Γᵈ ⊢ t
    recInst : (recDecl ts) ∈ Γᵈ → TypeResolver Γ Γᵈ ts
        ----------------------------------------------
        → Γ , Γᵈ ⊢ recT ts
    rLookup : Γ , Γᵈ ⊢ recT ts → t ∈ ts
        ------------------------------
        → Γ , Γᵈ ⊢ t
```

Listing 3. Tuple and record-related constructors for the HLL.

```
data _,_⊢_⇓_ : Env Γ → (Γᵈ : DataCtx) → (Γ , Γᵈ ⊢ t) → Value t → Set where
    ...
    ⇓tuple : ReductionResolver γ Γᵈ tr vs
        --------------------------------
        → γ , Γᵈ ⊢ tuple tr ⇓ tuple vs
    ⇓tLookup : γ , Γᵈ ⊢ e ⇓ tuple vs
        ---------------------------------------------
        → γ , Γᵈ ⊢ (tLookup e x) ⇓ poly-list-lookup vs x
    ⇓recInst : ReductionResolver γ Γᵈ tr vs
        -----------------------------------
        → γ , Γᵈ ⊢ recInst x tr ⇓ rec vs
    ⇓rLookup : γ , Γᵈ ⊢ e ⇓ rec vs
        ------------------------------------------
        → γ , Γᵈ ⊢ (rLookup e x) ⇓ poly-list-lookup vs x
```

Listing 2. Big-step semantics for tuple and record-related constructs.

5. Apply refactoring to the HLL

- **Refactoring operation:** for all tuples, generate a record declaration and replace the tuple by a record instance of that declaration. All declarations are globally known.
- Make a sub-expression aware of its context by providing a trace. This helps to update existing record declaration lookups and generate new ones for situations similar to **figure 1.**

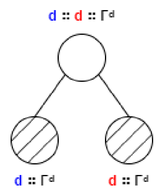

Figure 1. Example of the refactored declaration context at different levels of a language construct. Dashed circles indicate refactored tuples.

6. Proving correctness and conclusions

- Due to the use of an intrinsically-typed language, the refactoring operation is also a well-typedness proof.
- Prove that the refactoring replaces **all** tuples by mapping a refactored expression to a construct that does not support tuples.
- Relation construct is used to show how the refactoring alters the evaluated value of an expression.
- Successfully constructing these proofs leads to a positive feasibility indication.
- Techniques can be reused for other functional programming languages that share the notion of tuples and records (e.g., Erlang).

7. References

[1] D. Horpácsi, J. Kőszegi, and S. Thompson, “Towards Trustworthy Refactoring in Erlang,” Electronic Proceedings in Theoretical Computer Science, vol. 216, pp. 83–103, 7 2016.
[2] E. A. AlOmar, M. W. Mkaouer, C. Newman, and A. Ouni, “On preserving the behavior in software refactoring: A systematic mapping study,” Information and Software Technology, vol. 140, p. 106675, 2021.
[3] L. Miran, Learn you a Haskell for great good!: A beginner’s guide, ch. Making Our Own Types and Type Classes. No Starch Press, 2012.

Supervisors: J.G.H. Cockx, L. Miljak

TUDelft