Practical Verification of a Free Monad Instance

Luka Janjić (L.Janjic@student.tudelft.nl)

Supervisors: Jesper Cockx and Lucas Escot

Computer Science and Engineering – TU Delft

Background

Free Monad

```
data Free f a = Pure a | Free (f (Free f a))
```

- Finer structuring of large, monolithic monads such as IO
- Convenient way of embedding domain specific languages

Agda

- **Total**
  - → all programs terminate
- **Purely functional**
  - → everything is a function
- **Dependently typed**
  - → type system also a logic system

Strict Positivity

- A restriction on data types in Agda
- Prevents direct translation of Free
- Parameter **f** is not strictly positive

Containers

$$[\![S \triangleright P]\!]\ X = \Sigma[\ s \in S\ ]\ (P\ s \rightarrow X)$$

- Uniform way of representing strictly positive types
- Determined by types S (shape) and P(s) (positions)
- Allows for reasoning about the most general case permissible by Agda

Research Question

How can a free monad data type Free be formally verified using Agda2HS?

- How can the definition of Free and its monad instance be precisely translated to Agda?
- Can we, and how, verify the monad laws on the traslation?
- Can the translation be made such that Agda2HS translates it the original Haskell definition?

Research Method

1. Port a portion of Haskell's free monad library to Agda

```
data Free (F : Container00) (A : Set) : Set
where
    pure : A → Free F A
    free : ⟦ F ⟧ (Free F A) → Free F A
```

2. State and prove monad laws

```
monad-right-id (free fa) =
   begin
      free (fmap (_>>= pure) fa)
   =⟨ cong free (fmapBindToReturnIsId fa) ⟩
      free fa
    end
```

3. Transpile the Agda code back to Haskell using Agda2HS
4. Verify that the generated Haskell code is equivalent to the one from the original library

Results

Translated Free to Agda, representing the first parameter by a container:

```
data Free (F : Container00) (A : Set) : Set
where
    pure : A → Free F A
    free : ⟦ F ⟧ (Free F A) → Free F A
```

Reached the limits of Agda2HS:

- The resulting formulation of **Free** cannot be handled by Agda2HS
- More fine grained control over the translation is required

Desired translation basis:

```
data Free (F : Set → Set) (A : Set) : Set
where
    pure : A → Free F A
    free : F (Free F A) → Free F A
```

Proved that the monad laws hold for the Free data type

```
monad-left-id
 : (a : A) → (f : A → Free F B)
 → (return a >>= f) ≡ f a

monad-right-id : (m : Free F A)
 → m >>= return ≡ m

monad-assoc : (m : Free F A)
 (g : A → Free F B) (h : B → Free F C) →
(m >>= g) >>= h ≡ m >>= (λ x → g x >>= h)
```

Conclusions

Current capabilities of Agda2HS are not nuanced enough to handle alternative representation of parameters

```
data Free (F : Container00) (A : Set) : Set
where
    pure : A → Free F A
    free : ⟦ F ⟧ (Free F A) → Free F A
```

- The **F**'s type must be translated as a more general type `Set → Set`
- The "instantiation" of **F** (i.e. ⟦_⟧) must be omitted in the translation

Suggested improvement to the tool

- Include an *annotation* declaring an alternative representation is used in a definition
- Two mandatory arguments:
  - What type is replaced (**`Container00`**)
  - By what is it replaced (**`Set`** `→ Set`)
- One optional argument for declaring the "instantiation" function (⟦_⟧)

Example syntax

```
@R{Container00} {Set → Set} {⟦_⟧}
data Free (F : Container00)
          (A : Set) : Set where
    pure : A → Free F A
    free : ⟦ F ⟧ (Free F A) → Free F A
```