Translating Type Class Laws from Agda2hs to QuickCheck

Giovanni Grandi

Supervisors: Jesper Cockx, Nathaniel Burke

1. Agda2Hs

agda2hs [1] allows compiling a subset Agda code into readable Haskell

```
data Vec (a : Type) : (@0 n : Nat) → Type where
  Nil : Vec a 0
  Cons : {@0 n : Nat} → a → Vec a n → Vec a (suc n)
{-# COMPILE AGDA2HS Vec #-}
```

```
data Vec a = Nil | Cons a (Vec a)
```

Agda allows expressing stricter types for stronger guarantees and proving statements about code directly in the same language

2. QuickCheck

QuickCheck [2] allows property based testing in Haskell. We can write code and then test specifications on random inputs

```
add :: Nat -> Nat -> Nat
add Zero y = y
add (Suc x) y = add x y

prop_add_commutative (x :: Nat) (y :: Nat) = add x y === add y x
```

QuickCheck runs the property a number of times with random inputs until it can find a counterexample.

```
ghci> quickCheck prop_add_commutative
*** Failed! Falsified (after 2 tests):
0
1
```

This allows quickly checking if a property holds

3. Type classes

Type classes are the way to implement ad-hoc polymorphism in Agda and Haskell. Some examples are:

```
record Eq (a : Type) : Type where
  field _==_ : a → a → Bool
```

Eq which provides a way to check if two values are equal

```
record Show (a : Type) : Type where
  field show : a → String
```

Show which allows turning a type into a String

```
record Monoid (a : Type) : Type where
  field mempty : a
        _<>_ : a → a → Bool
```

Monoid which allows combining two values of the same type

Often type classes have a series of laws which represents how a lawful instance should behave.

4. Example Translation

An example of how my code creates properties based on laws:

1. add the laws pragma to a definition

```
postulate instance
  iLawfulFunctorMaybe : IsLawfulFunctor Maybe
{-# COMPILE AGDA2HS iLawfulFunctorMaybe laws #-}
```

2. lookup the type of the definition

```
record IsLawfulFunctor (F : Type → Type) ⦃ iFuncF : Functor F ⦄ : Type₁ where
  field
    identity : (fa : F a) → (fmap id) fa ≡ id fa
    composition : (fa : F a) (f : a → b) (g : b → c)
      → fmap (g ∘ f) fa ≡ (fmap g ∘ fmap f) fa
```

3. for each of the fields, infer its type

```
{a b c : Type} (fa : Maybe a) (f : a → b) (g : b → c) →
  fmap (g ∘ f) fa ≡ (fmap g ∘ fmap f) fa
```

4. substitute all of the types with Integer

```
(fa : Maybe Integer) (f : Integer → Integer) (g : Integer → Integer) →
  fmap (g ∘ f) fa ≡ (fmap g ∘ fmap f) fa
```

5. Translate the signature into Haskell

```
prop_composition (fa :: Maybe Integer)
  (Fun _ (f :: Integer -> Integer)) (Fun _ (g :: Integer -> Integer))
  = fmap (g . f) fa == (fmap g . fmap f) fa
```

5. Analysis

The generation was evaluated based on some real life Haskell examples:

- the ListT monad transformer from the transformers package
- a type class based on the signatures of different data structures to extract a sorted list
- a variety of the postulated type classes in the agda2hs base library

This analysis found that:

- Finding counterexamples was useful for difficult to prove type class laws
- using Dec to check the result of a type class law was useful for predicates that cannot be expressed in terms of _≡_
- There are still some limitations around functions, preconditions and arbitrary types to generate properties for all the laws in agda2hs base

6. Conclusions

- It is possible to translate type class laws into QuickCheck properties
- Using a property based test before trying to prove a type class law was useful for verifying there were no bugs in the definition

Future work:

- Modify the laws pragma to allow passing in arbitrary types
- Support generating properties for laws with preconditions
- Allow emitting a different notion of equality to support functions
- Supporting mixing both functions and laws in the same type class

Recommendations for Agda2Hs:

- Add support for quotient types

TU Delft

Bibliography: [1] J. Cockx, O. Melkonian, L. Escot, J. Chapman, and U. Norell, “Reasonable Agda is correct Haskell: writing verified Haskell using agda2hs,” in Proceedings of the 15th ACM SIGPLAN International Haskell Symposium, in Haskell 2022. New York, NY, USA: Association for Computing Machinery, Sep. 2022, pp. 108–122. doi: 10.1145/3546189.3549920.
[2] K. Claessen and J. Hughes, “QuickCheck: a lightweight tool for random testing of Haskell programs,” in Proceedings of the fifth ACM SIGPLAN international conference on Functional programming, in ICFP '00. New York, NY, USA: Association for Computing Machinery, Sep. 2000, pp. 268–279. doi: 10.1145/351240.351266.