Monadic W in Coq

CASS 2020
Coq Andes Summer School

Rafael Castro G. Silva; Cristiano Vasconcellos; Karina Girardi Roggia
from
Santa Catarina State University

The Hoare Exception-State Monad

The *Hoare State Monad* was introduced by Swierstra as a variant of the usual state monad that can be easily applied to the verification of stateful programs in Coq. This monad, in the Coq language, is a definition of a `Type` called `HoareState p a q`, such that the parameters are in order: the precondition, the type of the return value and the postcondition. This formulation allows Hoare-Floyd reasoning for programs in the state monad, however as pointed out by Gibbons and others is not possible to directly apply this technique to others monads.

The algorithm W needs to deal with two kinds of effects: state and exception. The HSM can only handle states, so we propose a simple modification to it in order to also capture type inference errors: we simply embed the `sum` type to the definition of `HoareState`. Our modification doesn't create much additional complexity, therefore reading its original paper and our source code is enough to understand our modified version. We call this monad Hoare *Exception-State Monad* (HESM).

The Monadic W in Coq

This central part of our work is based on the certified unification presented by Ribeiro and Camarão (referenced here as *original unification*) and the algorithm W certification presented by Dubois (references here as *original work*). In regard of the used unification algorithm, we basically change the original unification to operate over a pair of types instead of list of pair of types and change how the application operation of the substitution works in order to obtain some useful lemmas to the completeness proof of the algorithm W. Those changes had some impact in the termination proof that we don't have enough space to discuss here.

Our implementation of algorithm W and its proofs are from scratch in order to work with Coq versions 8.9* and, also, so we could reformulate it in the HESM. The instance of the HESM that we use is called `Infer`, with the type `id` (Coq's $\mathbb{N}$) to count the type variable states and the type `InferFailure` to track down all possible inference errors. In Figure 1 the type signature of algorithm W is imposing the precondition that the initial state `i` in the monad is new in relation to the given context (`new_tv_ctx G i`) and the poscondition requires several conditions related to the initial state, final state `f` and the resulting computation `r` of type `ty * substitution`. The penultimate postcondition is soundness and the last is completeness.

```
Program Fixpoint W (e:term) (G:ctx) :
Infer (λ i ⇒ new_tv_ctx G i) (ty * substitution)
      (λ i r f ⇒ i ≤ f ∧ new_tv_subst (snd r) f ∧ new_tv_ty (fst r) f ∧
      new_tv_ctx (apply_subst_ctx (snd r) G) f ∧ (snd r)(G) ⊢ e : (fst r) ∧
      completeness e G (fst r) (snd r) i) :=
```

Figure: The left side of the monadic W in Coq.

Proof Technique Analysis

Since our development is most related to the one made by Dubois, we will provide a comparison between them to demonstrate the advantages of the HESM. The HESM is a method for dependent type programming (see Figure 1) and, therefore, it avoids the use of awkward lemmas stating the results (and the monad state) of algorithm W and other subroutines.

The information presented in the Coq's context are similar to the suppositions made during a proof with paper and pen of algorithm W, for example inductive hypothesis about the posconditions of each recursive call and hypothesis about the new type variables[a]. On the other hand, the original work had to develop many separated lemmas in order to obtain those hypothesis.

The original work uses explicit argument passing to deal with the generation of fresh variables, but, unfortunately, many lemmas to reason about it were necessary. For example,

```
∀ (e:term) (st st':id) (G:ctx) (tau:ty) (S:substitution),
  W st G e = Some tau S st' → st ≤ st'
```

guarantees that the algorithm W can only increase the state argument or keep it. In the original work, lemmas stating this invariant were necessary for different parts of the algorithm (e.g. the unification), but in our formulation it's already part of the main certification.

[a] Provide by the precondition and the postcondition.

Conclusion

This study case is a complete formalization of algorithm W and its proofs of soundness and completeness for the Damas-Milner type system. Therefore, it doesn't rely on unification axioms as previous work for this algorithm, thus it was possible to extract to Haskell an executable code.

Even though other mechanized proofs for this algorithm have already been done, none of them follows the usual implementation with a monad and use it as part of the method of proving. The Hoare State Monad allows the certification of stateful programs using Hoare-Floyd reasoning, but for this study case a simple extension was introduce to also handle exceptions.

Our main contribution concerns the use of *The Hoare State Monad* in the certification, which was useful to avoid some odd lemmas and definitions, moreover, this monad provided a sufficient set of hypothesis to proceed with the completeness proof. Side contributions are: a modification on the Ribeiro and Camarão unification to use it in algorithm W, a modification on the *The Hoare State Monad* to handle failure and a revitalization of the Dubois work. This whole formalization counts 2055 lines of definitions, 3804 lines of proofs, 143 definitions and 254 lemmas[a].

[a] Excluding the LibTactics file.