

Monadic W in Coq

Rafael Castro

rafaelcgs10@gmail.com

Departamento de Ciência da Computação
Centro de Ciências e Tecnológicas
Universidade do Estado de Santa Catarina

25 de Set de 2020

Contents

- Introduction
- Foundations
- The Monadic W in Coq

Introduction



Introduction - A type inference algorithm

- What is type inference?
- The **algorithm W** is a classical type inference algorithm, which was originally designed for the Damas-Milner type system.
The original inspiration for the systems of languages like Haskell and OCaml.
- *Ideally* a type inference algorithm must represent its type system defines.
- Formally this is called *Soundness* and *Completeness*.



Introduction - What this work is about?





Introduction - What this work is about?

This a Coq certification of the monadic algorithm
 $W!$



Introduction - What this work is about?

This a Coq certification of the monadic algorithm W!

- We bundle together the following related work:
 - Algorithm W in Coq (work by Dubois)
 - Type unification in Coq (work by Ribeiro and Camarão)
 - The Hoare State Monad (work by Swierstra)

Foundations



Foundations - Damas-Milner and algorithm W

What is certifying algorithm W in Coq?

- *Soundness*: What algorithm W infers it can be demonstrated using the Damas-Milner typing rules
- *Completeness*: if Damas-Milner shows that an expression e has a type τ , then W infers for e a type τ' , such that $\tau = \mathbb{R}(\tau')$ for some substitution \mathbb{R} .

Foundations - Hoare State Monad

- *Hoare State Monad* (HSM): presented by [Swierstra, 2009]
A variation of the usual state monad.
- Useful for certifying stateful programs.
- In Coq: HoareState p a q: : a precondition, a returning value and a post-condition.

```
Program Definition HoareState (pre : Pre) (a : Type) (post : Post a) : Type :=
  forall i : {t : st | pre t}, {(x, f) : a * st | post i x f}.
```

```
Program Definition ret (a : Type) : forall x,
  @HoareState top a (fun i y f => i = f /\ y = x) := fun x s => (x, s).
```

```
Program Definition bind : forall a b P1 P2 Q1 Q2,
  (@HoareState P1 a Q1) -> (forall (x : a), @HoareState (P2 x) b (Q2 x)) ->
  @HoareState (fun s1 => P1 s1 /\ forall x s2, Q1 s1 x s2 -> P2 x s2)
    b
    (fun s1 y s3 => exists x, exists s2, Q1 s1 x s2 /\ Q2 x s2 y s3) :=
      fun a b P1 P2 Q1 Q2 c1 c2 s1 => match c1 s1 as y with
        | (x, s2) => c2 x s2
      end.
```

Figura: The Hoare State Monad.

Foundations - Hoare Exception-State Monad

- Algorithm W has two side effects: state and exception.
- Hoare Exception-State Monad (HESM):*
add the *sum* type to the definition of Hoare State.

```
Program Definition HoareState (B:Prop) (pre:Pre) (a:Type) (post:Post a) : Type :=  
forall i : {t : st | pre t},  
{e : sum (prod a st) B | match e with  
| inl (x, f) => post (proj1_sig i) x f  
| inr _ => True  
end}.
```

```
Program Definition bind : forall a b P1 P2 Q1 Q2 B,  
(@HoareState B P1 a Q1) -> (forall (x:a), @HoareState B (P2 x) b (Q2 x)) ->  
@HoareState B (fun s1 => P1 s1 /\ forall x s2, Q1 s1 x s2 -> P2 x s2) b  
(fun s1 y s3 => exists x, exists s2, Q1 s1 x s2 /\ Q2 x s2 y s3) :=  
fun B a b P1 P2 Q1 Q2 c1 c2 s1 => match c1 s1 as y with  
| inl (x, s2) => c2 x s2  
| inr R =>_  
end.
```

Figura: The Hoare Exception-State Monad.

Certification of algorithm W

Certification of algorithm W - Damas-Milner in Coq

- The work of [Dubois and Ménissier-Morain, 1999] is our main reference.
- Terms and types (simple and polymorphic)
- Typing rules of Damas-Milner in the *syntax-directed* version, given by the inductive type `has_type`.
 - Each proving tree has a `uniq` format for each lambda term.

Certification of algorithm W - Monadic W

- The monadic algorithm W was implemented using the HESM.
- The certification is given by the type signature:

```
Program Fixpoint W (e:term) (G:ctx) {struct e} :  
Infer (fun i => new_tv_ctx G i) (ty * substitution)  
  (fun i x f => i <= f /\ new_tv_subst (snd x) f /\ new_tv_ty (fst x) f /\  
    new_tv_ctx (apply_subst_ctx (snd x) G) f /\  
    has_type (apply_subst_ctx ((snd x)) G) e (fst x) /\  
    completeness e G (fst x) ((snd x)) i) :=
```

Certification of algorithm W - Completeness definition

Given Γ and e , let Γ' be an instance of Γ and σ a scheme such that

$$\Gamma' \vdash e : \sigma,$$

then

- ① $W(\Gamma, e)$ computes successfully.
- ② Se $W(\Gamma, e) = (\mathbb{S}, \tau)$ therefore, for some type substitution \mathbb{R} ,

$$\Gamma' = \mathbb{R}\mathbb{S}\Gamma \text{ e } \mathbb{R}\text{gen}(\mathbb{S}\Gamma, \tau) \geq \sigma.$$

Certification of algorithm W - Completeness definition

Given Γ and e , let Γ' be a instance of Γ and σ a scheme such that

$$\Gamma' \vdash e : \sigma,$$

then

- ① $W(\Gamma, e)$ computes successfully.
- ② Se $W(\Gamma, e) = (\mathbb{S}, \tau)$ therefore, for some type substitution \mathbb{R} ,

$$\Gamma' = \mathbb{R} \mathbb{S} \Gamma \text{ e } \mathbb{R} \text{gen}(\mathbb{S} \Gamma, \tau) \geq \sigma.$$

The reformulation of completeness is:

```

Definition completeness (e:term) (G:ctx) (tau:ty) (s:substitution) (st:id) :=
  forall (tau':ty) (phi:substitution),
  has_type (apply_subst_ctx phi G) e tau' ->
  exists s', tau' = apply_subst s' tau /\
  (forall x:id, x < st ->
    apply_subst phi (var x) = apply_subst s' (apply_subst s (var x))).
  
```

Certificação do algoritmo W - Prova de completude

- A HESM é útil nesta prova, pois permite raciocinar sobre o estado do contador.
- É necessário apresentar (computar) quem é a substituição s' .
- O caso da aplicação se utiliza do fato da unificação computada ser a mais geral.
- A maior dificuldade novamente é o caso do termo *let*, que requer raciocínio sobre a criação de novas variáveis de tipo e a relação de mais geral entre tipos.

Certificação do algoritmo W - Análise da prova

- A HESM é um tipo dependente: o programa, cujo tipo é a sua especificação, e a sua prova são desenvolvidos na mesma etapa.
- Program permite que elementos da prova sejam postergados como obrigações de provas e a escrita do programa seja feita como em tipos simples, evitando casamentos de padrão dependentes.
- As informações presentes no contexto são similares as suposições feitas na prova com papel e caneta do algoritmo W [Damas, 1984]
- Evitou-se diversos lemas auxiliares *deselegantes* relacionados ao estado do contador e lemas que são hipóteses de indução.

Conclusão

- Algumas modificações não triviais no certificação do algoritmo de unificação apresentado por [Ribeiro and Camarão, 2016].
- Modificou-se a *Hoare State Monad* para lidar com exceções.
- Apresentou-se uma certificação completa em Coq do algoritmo, cujo código pode ser extraído e executado.
- Uso da HESM para a certificação de um algoritmo de inferência de tipos.

Dificuldades

- A certificação de algoritmos de inferência de tipos requer um grande corpo de lemas e definições.
- *Binding* de variáveis é difícil de lidar.
- Os casos polimórficos são os mais complicados.

References I

-  Damas, L. M. M. (1984).
Type Assignment in Programming Languages.
(CST-33-85).
-  Dubois, C. and Ménissier-Morain, V. (1999).
Certification of a Type Inference Tool for ML: Damas-Milner
within Coq.
Journal of Automated Reasoning, 23(3-4):319–346.
-  Ribeiro, R. and Camarão, C. (2016).
A mechanized textbook proof of a type unification algorithm.
In Cornélio, M. and Roscoe, B., editors, *Formal Methods: Foundations and Applications*, Cham.



References II

-  Swierstra, W. (2009).
The Hoare State Monad.
Technology, pages 440–451.