



Uma Certificação em Coq do Algoritmo W Monádico

Rafael Castro

rafaelcgs10@gmail.com

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

22 de Agosto de 2019



Sumário

- Introdução
- Fundamentos
- Certificação do Algoritmo W

Introdução



Introdução - Sistemas de Tipos

- Sistemas de tipos são relações que ditam quais expressões podem ser associadas a quais tipos.
- Nas linguagens de programação, classificavam os dados para serem tratados corretamente pelo processador;
- Passaram a ser utilizados como uma técnica para identificar falhas na consistência de programas.
- Tipagem estática e dinâmica:
 - Estática: atribuições de tipo não mudam em *run time*; uma forma de verificação automática.
 - Dinâmica: atribuições de tipo **podem mudar** em *run time*; mais permissível; mais permissível a erros.

[1, 'a', objeto, [2]]



Introdução - Inferência de Tipos e Polimorfismo

- A **inferência de tipos** é o processo de encontrar a assinatura de tipo de um programa.
- Liberdade ao programador de escolher anotar ou não os tipos das funções.
- Polimorfismo é quando uma função pode assumir vários usos com diferentes tipos de dados. Em especial, o polimorfismo paramétrico é feito pela quantificação de variáveis de tipos.

reverse :: [a] → [a]



Introdução - Algoritmo para Inferência de Tipos

- O **algoritmo W** é um algoritmo para a inferência de tipos no sistema Damas-Milner.
Uma das bases para a inferência em Haskell e OCaml.
- *Idealmente* um algoritmo de inferência deve *representar* o que o seu sistema de tipos define.
- Formalmente isso é *Soundness* e *Completeness*.
- *Essas propriedades são diretamente refletidas nas implementações?*

Introdução - Provas mecanizadas

- O uso de assistentes de provas na pesquisa de linguagens de programação.
- Aproximam formalização e implementação.
- Reduzem o trabalho da verificação da prova.
- *PoplMark Challenge* e *Principles of Programming Languages* (POPL)



Introdução - O que é este trabalho?





Introdução - O que é este trabalho?

Uma Certificação em Coq do
Algoritmo W Monádico



Introdução - O que é este trabalho?

Uma Certificação em Coq do Algoritmo W Monádico

- Uma completa certificação de uma implementação monádica do **algoritmo W** no assistentes de provas Coq.
- Certificação do algoritmo de unificação.
- Uso da *Hoare State Monad* (HSM) para certificar o código monádico.



Introdução - Objetivos específicos

- Uma implementação certificada do **algoritmo de unificação**: consistência, completude e terminação.
- Modificar a *Hoare State Monad* para lidar com o efeito de exceção necessário no algoritmo W e avaliar as qualidades dessa técnica.
- Provar a consistência e a completude do algoritmo W monádico.

Fundamentos



Fundamentos - Damas-Milner e algoritmo W

- Damas-Milner: polimorfismo via *let*: forma restrita de polimorfismo.
- O algoritmo W é consistente e completo em relação as regras de Damas-Milner.
- Consistência (*Soundness*): O que o algoritmo W infere pode ser demostrado pelas regras de Damas-Milner.
- Completude (*Completeness*): Se o sistema Damas-Milner mostra que e tem o tipo τ , então o algoritmo infere para e o tipo τ' , de maneira que $\tau = \mathbb{R}(\tau')$ para alguma substituição \mathbb{R} .



Fundamentos - Trabalhos relacionados

- [Dubois and Ménissier-Morain, 1999] forneceram provas em Coq de consistência e completude do algoritmo W. As propriedades da unificação foram tomadas como axiomas e *binding* de variáveis é resolvido com *de Bruijn indices*.
- [Naraschewski and Nipkow, 1999] também apresentaram provas de consistência e completude do algoritmo W, mas em Isabelle/HOL. As propriedades da unificação também foram assumidas como axiomas e *de Bruijn indices* também foi utilizado.
- [Kothari and Caldwell, 2009] relataram um resultado parcial na certificação em Coq da extensão polimórfica do algoritmo de Wand.
- [Tan et al., 2015] verificaram a inferência de tipos de CakeML, uma linguagem baseada em SML (Standard ML) cujo compilador tem um *backend* completamente verificado [Kiam Tan et al., 2019]. As provas de consistência e completude da inferência de tipos foram feitas no assistente de provas HOL4.

Fundamentos - Hoare State Monad

- *Hoare State Monad* (HSM): apresentada por [Swierstra, 2009]
Uma variante da usual mònada de estados.
- Verificação de programas com efeitos de estado.
- Um tipo HoareState $p \ a \ q : a$ pré-condição, o tipo do valor de retorno e a pós-condição.

```
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: A Mônada de Estado de Hoare.

Fundamentos - Hoare Exception-State Monad

- O algoritmo W tem dois tipos de efeitos: estado e exceção.
- *Hoare Exception-State Monad* (HESM):
embute-se o tipo *sum* na definição de HoareState.

```
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) -> (@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: A Mônada de Estado-Exceção de Hoare.



Certificação do algoritmo W



Certificação do algoritmo W - Unificação I

- O algoritmo de unificação [Robinson, 1965] é parte fundamental do algoritmo W.
- Modifica-se a certificação da unificação realizada por [Ribeiro and Camarão, 2016]:
 - A operação da substituição é de reformulada para ser não-incremental.
 - O algoritmo é reescrito para unificar somente dois tipos, ao invés de uma lista de tipos.
- A unificação não é recursão estrutural. A prova de terminação é feita por uma relação bem-fundada sobre uma ordenação lexicográfica.
- Essas modificações resultaram em algumas alterações não triviais na prova de terminação.



Certificação do algoritmo W - Unificação II

- Os tipos (monomórficos) são definidos em Coq como tipos induktivos, onde os identificadores são números naturais.
- A substituição de tipos é definida em Coq como uma lista de duplas de identificadores de tipo e tipos.
Acompanhado de uma operação de aplicação da substituição.



Certificação do algoritmo W - Damas-Milner em Coq I

- O trabalho de [Dubois and Ménissier-Morain, 1999] é a principal referência desta formalização.
- Termos e tipos da linguagem de Damas-Milner são definidos em Coq como tipos indutivos.
- A linguagem dos tipos é dividida em duas:
 - Tipos monomórficos.
 - Tipos polimórficos: possuem um construtor a mais para representar variáveis quantificadas.



Certificação do algoritmo W - Damas-Milner em Coq II

- Contextos são representados como listas de duplas de identificadores e tipos polimórficos.
- As regras de tipagem do sistema Damas-Milner são utilizadas na versão *syntax-directed*, pelo o tipo indutivo `has_type`.
- Essas modificações resultaram em algumas alterações não triviais na prova de terminação.
 - Arvores de provas tem um único formato para cada termo `lambda`
 - Quantificação e instanciação estão embutidas, respectivamente, nas regras para variáveis e `lets`.



Certificação do algoritmo W - Instanciação de tipos

- Tipos polimórficos são instanciados para tipos monomórficos por meio de uma operação de substituição especial chamada.
- Definiu-se uma forma de substituição: `inst_subst` é apenas uma lista de tipos monomórficos, ao invés de uma lista de associatividade.
- O número n na variável quantificada associada com o tipo encontrado na n -ésima posição em `inst_subst`.

Certificação do algoritmo W - Generalização de tipos

A problemática da generalização

- α e β são duas variáveis de tipo sintaticamente distintas.
- Não ocorrem livres no contexto considerado.
- Os tipos $\alpha \rightarrow \alpha$ and $\beta \rightarrow \beta$ são quantificados, respectivamente, para $\forall\alpha, \alpha \rightarrow \alpha$ e $\forall\beta, \beta \rightarrow \beta$.
- Sintaticamente diferentes, mas representam exatamente o mesmo conjunto de tipos.

Certificação do algoritmo W - Generalização de tipos

A problemática da generalização

- α e β são duas variáveis de tipo sintaticamente distintas.
- Não ocorrem livres no contexto considerado.
- Os tipos $\alpha \rightarrow \alpha$ and $\beta \rightarrow \beta$ são quantificados, respectivamente, para $\forall\alpha, \alpha \rightarrow \alpha$ e $\forall\beta, \beta \rightarrow \beta$.
- Sintaticamente diferentes, mas representam exatamente o mesmo conjunto de tipos.
- *Binding* de variáveis costuma dificultar provas.
Utiliza-se *de Bruijn indices* para lidar com as variáveis quantificadas e evitar lidar com equivalência α .
- A função de `gen_ty` quantifica as variáveis de tipo são quantificados linearmente: se var n é a m -ésima variável de tipo, então essa é convertida em `sc_gen m`.



Certificação do algoritmo W - A algoritmo W monádico

- O algoritmo W foi implementado de forma monádica com a HESM.
- A certificação é dada pela assinatura com tipos dependentes:

```
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) :=
```



Certificação do algoritmo W - Prova de consistência

- Enunciado por `has_type` sobre o resultado do algoritmo W.
- Trivial para a maioria dos casos:
 - O caso da aplicação: requer o lema da estabilidade da substituição.
 - O caso do termo *let*: requer vários lemas e definições auxiliares, incluindo sobre listas disjuntas¹, sub-listas e substituições de renomeação.

¹Cujo predicado chama-se `are_disjoints`



Certificação do algoritmo W - Substituição de renomeação

- A substituição de renomeação é uma substituição com tais características em que:
 - Cada variável no domínio (*id*) mapeia para uma variável (*id*).
 - O domínio e a imagem da substituição são disjuntos.
 - Duas distintas variáveis no domínio têm diferentes imagens.
- Definido pelo tipo `ren_subst`, implementado como uma lista de associatividade entre identificadores.
- Condições são formalizadas no tipo indutivo: `is_rename_subst`.
- A prova do caso *let*² requer a demonstração da existência de uma substituição de renomeação em especial: o domínio é uma lista de variáveis 1 e que não mapeia para as listas de variáveis 11 e 12.

²Para a prova de consistência do algoritmo W e do lema da estabilidade da tipagem sobre a substituição



Certificação do algoritmo W - Estabilidade da substituição

- O lema da estabilidade da substituição é uma propriedade clássica em sistemas de tipos.
- Se é verdade que $\Gamma \vdash e : \tau$, então para qualquer substituição $\$$ tem-se $\$ \Gamma \vdash e : \τ .
- Casos monomórficos são fáceis.
- O caso polimórfico do `let_ht` requer uma prova sobre a comutação de uma generalização de tipo (de algum τ) com uma substituição de tipo s .
Condição: condição: a substituição s não pode estar relacionada com as variáveis a serem quantificadas τ
- Computa-se uma substituição ρ que renomeia as variáveis de tipo em τ que devem ser generalizadas em novas variáveis de tipos que não ocorrem na substituição s e não são livres no contexto considerado.

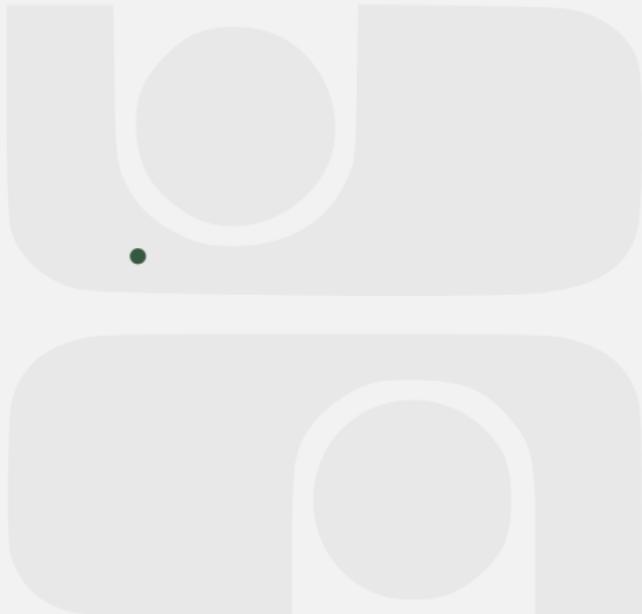
Certificação do algoritmo W - Prova de completude

A definição de completude é:

```
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))).
```

- 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, as quais são definidas para tipos monomórficos (*new_tv_ty*), para *schemes* (*new_tv_schm*), para contextos (*new_tv_ctx*) e para substituições (*new_tv_subst*).

Certificação do algoritmo W - Análise da prova





Conclusão

- Fundamentos teóricos para formalização do MMo.
- Assistentes de provas Coq.
- Como o MMo funciona.
- Formalização do MMo parcial.
- Principais dificuldades encontradas.
- Alternativas para a prova de terminação.

References I

-  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.
-  Kiam Tan, Y., Myreen, M. O., Kumar, R., Fox, A., Owens, S., and Norrish, M. (2019).
The Verified CakeML Compiler Backend.
Journal of Functional Programming, 29.
-  Kothari, S. and Caldwell, J. L. (2009).
Toward a machine-certified correctness proof of Wand's type reconstruction algorithm.

References II

-  Naraschewski, W. and Nipkow, T. (1999).
Type Inference Verified: Algorithm script W sign in Isabelle/HOL.
Journal of Automated Reasoning, 23(3-4):299–318.
-  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.
-  Robinson, J. A. (1965).
A machine-oriented logic based on the resolution principle.
J. ACM, 12(1):23–41.

References III

-  Swierstra, W. (2009).
The Hoare State Monad.
Technology, pages 440–451.
-  Tan, Y. K., Owens, S., and Kumar, R. (2015).
A verified type system for CakeML.
pages 1–12.