Um Algoritmo Verificado para Inferência de Tipos na Presença de Recursão Polimórfica

Rafael Castro

rafaelcgs10@gmail.com

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

15 de Fevereiro de 2019

Introdução - Sistemas de Tipos

- Sistemas de tipos são regras 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 regras para identificar falhas na consistência de programas

Introdução - Inferência de Tipos e Polimorfismo

- A inferência de tipos é o processo de encontrar a assinatura de tipo mais geral 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.

```
len :: [a] -> Int
```

- Existem situações onde a inferência de tipos é mais difícil que a verificação. Nesses casos, algoritmos podem rejeitar programas bem tipados.

Introdução - Recursão Polimórfica

- A recursão polimórfica acontece quando o tipo da chamada recursiva de uma função pode mudar ao longo das recursões.

```haskell
data Seq a = Nil | Cons a (Seq (a,a)).

len :: Seq a -> Int
len Nil = 0
len (Cons x s) = 1 + 2 * (len s)
```

- Haskell rejeita programas com recursão polimórfica se a assinatura de tipo não for explicitamente anotada.
- Diversos exemplos precisando de recursão polimórfica foram apresentados na literatura.

Introdução - Motivação e Objetivo

- O semi-algoritmo MMo proposto por (VASCONCELLOS; CAMARÃO, 2003) consegue tipar diversos casos de recursão polimórfica e recursão mútua.
- MMo não tem as suas propriedades fundamentais provadas: terminação (totalidade) e consistência (correto).
- **Objetivo**: investigar se o semi-algoritmo MMo tem as propriedades de totalidade e correção em relação ao seu sistema de tipos (especificação).
- A especificação em si também é um elemento de investigação.
- O método utilizado é a formalização do MMo e a sua especificação no assistente de provas Coq.

Fundamentos - Formalismos de Sistemas de Tipos

- Sistemas de tipos quando definidos formalmente podem garantir importantes propriedades da semântica de programas, como o famoso lema de Robin Milner "Well-typed programs cannot go wrong".
- Damas-Milner: polimorfismo via *let*. Base para diversas linguagens de programação funcional. Não permite recursão polimórfica pela regra *fix*.
- O primeiro sistema de tipos a permitir recursão polimórfica é o Sistema-F.
- Milner-Mycroft altera a regra de definições recursivas *fix* para permitir recursões polimórficas.

$$\frac{\Gamma, \mathtt{x} : \tau \vdash \mathtt{e} : \tau}{\Gamma \vdash \mathtt{fix}\ \mathtt{x}.\mathtt{e} : \tau}\,(\mathit{fix}) \qquad \frac{\Gamma, \mathtt{x} : \sigma \vdash \mathtt{e} : \sigma}{\Gamma \vdash \mathtt{fix}\ \mathtt{x}.\mathtt{e} : \sigma}\,(\mathit{fix}+)$$

Fundamentos - Inferência no Milner-Mycroft

- A inferência no sistema Milner-Mycroft é equivalente ao problema da semi-unificação.
- A semi-unificação é uma generalização da unificação.
- A semi-unificação é indecidível.
  - Casos decidíveis: apenas uma igualdade, qualquer aridade e no máximo uma variável livre, duas variáveis e semi-unificação acíclica.
- No caso geral não há como decidir a tipabilidade (*typability*) de um termo no sistema Milner-Mycroft.
- *Big type* vs *small type*.
- *Small types* no sistema Milner-Mycroft é decidível em tempo polinomial.
- Buscou-se alternativas para contornar a dificuldade deste problema.

Fundamentos - Propostas de Recursão Polimórfica

- Somente existem semi-algoritmos e algoritmos totais, mas incompletos pois rejeitam alguns programas bem tipados.
- O semi-algoritmo para resolver a semi-unificação apresentado em (HENGLEIN, 1993) foi utilizado na construção do semi-algoritmo W+ em (EMMS; LEIL, 1999).
    - Damas-Milner tem pior caso exponencial. Não acontece em situações práticas. Prática vs teoria no Milner-Mycroft.
- Diversas propostas para recursão polimórfica foram apresentadas na literatura com o uso de *intersection types*.

Fundamentos - Assistentes de provas

- São programas para o desenvolvimento de provas formais.
- O núcleo é um verificador, que verifica a consistência lógica da prova.
- Os principais apelos são:
  - a verificação mecânica é rápida e evita as falhas humanas;
  - interatividade, permite visualizar informações sobre os estados da prova;
  - comandos para busca de teoremas e lemas para o progresso da prova;
  - automatização de provas com métodos não-deterministas;
  - potencialização da capacidade humana de realizar provas;
  - extração de programas verificados.

Fundamentos - Por que Coq?

- Existem dezenas: Automath, Agda, Twelf, ACL2, PVS, Minlog, Isabelle e Coq...
- O núcleo do Coq é o *Calculus of Inductive Constructions* (CIC). Extensão do *Calculus of Constructions* (CoC).
- CoC é um Cálculo Lambda polimórfico de ordem superior e com tipos dependentes.
- **Justificativa**:
  1. Frequente uso de Coq para formalizar aspectos de linguagens de programação. CoqPL.
  2. Projetos relevantes em Coq: CompCert é uma compilador de C; Simplicity é uma linguagem de programação para *blockchains*; Vellvm é uma formalização da LLVM em Coq.
  3. Materiais didáticos disponíveis!

Fundamentos - Terminação em Coq

- Coq é uma linguagem de programação funcional total.
- A garantia de terminação do Coq é uma conservadora verificação sintática: recursão primitiva.
- Coq conta com algumas maneiras de estender o conjunto de programas que podem ser implementados:
  1. Recursão bem-fundada (*Well-Founded Recursion*) por meio de uma relação bem-fundada (*Well-Founded Relation*). Uma relação bem-fundada.
  2. Recursão limitada (*Bounded Recursion*) através de um argumento, que representa o número de chamadas recursivas. Suficiente para que a computação finalize com o resultado correto.
  3. Recursão por iteração (*Recursion by Iteration*). Similar a recursão limitada. Definição de um funcional, que tem como um dos argumentos a função `f` que deseja-se implementar.

Fundamentos - Terminação em Coq

4. Recursão sobre um predicado (*Recursion on an Ad Hoc Predicate Section*). Representa o domínio da função. Recursão estrutural sobre provas do princípio indutivo deste predicado.
5. Recursão por mônada inspirada em Teoria de Domínios. Combinadores de ordem superior representam funções que não terminam. Uma obrigação de prova é a continuidade da computação na mônada.
6. Recursão geral baseada numa mônada de um tipo co-indutivo. É similar ao anterior, mas utiliza os tipos co-indutivos para representar a noção de não terminação, com a vantagem de não ter obrigações de provas.
7. Linguagem embutida: formalizar completamente uma linguagem de programação.

Fundamentos - Vantagens Terminação em Coq

- Somente as quatro primeiras técnicas permitem utilizar o mecanismo de avaliação interno do Coq.
- As técnicas 2 e 3 necessitam conhecer uma função capaz de computar o número de chamadas recursivas suficientes para uma dada entrada.
- A principal vantagem das últimas três técnicas é a possibilidade de implementar qualquer função recursiva e evitam provas.
- No aspecto da extração, a técnica mais eficiente é a primeira, pois as demais podem carregar (na extração) alguns lixos como os argumentos adicionais.

MMo - Diferenças básicas

O MMo funciona de maneira similar ao algoritmo *W*, ou seja, primeiro inferem-se os termos mais internos da expressão para seguir inferindo os mais externos.

1. As variáveis com sequência de índices. Detecção de dependências circulares em substituições.
2. Um contexto Γ é um conjunto de triplas (*variável*, *Kind_def*, *scheme*). Não há condição de consistência. *Typing context*.
3. o MMo retorna uma dupla com o tipo simples inferido τ e um contexto Γ.
4. Para cada definição num *BindGroup* infere-se o respectivo conjunto com os pares de identificadores e os seus respectivos tipos inferidos e requeridos (dado pelo símbolo Σ).

MMo - Inferência de grupo de definições

1. Computar o conjunto Σ com todos os tipos inferidos e requeridos do grupo.
2. Gera-se a partir de Σ um conjunto Ω de pares tipos inferidos (instâncias) e requeridos (τ', τ) que precisam ser unificados.
   - o tipo de cada suposição `x`$:\tau$ em cada um dos Γ em Σ precisa ser unificado com uma instância do seu respectivo tipo inferido τ' para `x`.
   - A função `supInst` troca cada variável de tipo α^s de τ' por uma nova instância $\alpha^{i::s}$.
3. Verificar a existência de dependências circulares na substituição $\mathbb{S}$ gerada pela unificação de Ω.
4. Caso exista dependência circular em $\mathbb{S}$, então o MMo termina com erro. Caso contrário, então aplica-se a $\mathbb{S}$ em Σ. Se $\mathbb{S}$ for efetiva, então volta-se para segunda etapa. Se não, para.

MMo - Dependências circulares

- Uma dependência circular ocorre quando uma substituição troca uma variável de tipo α^{s} por um tipo τ, o qual ocorre uma variável de tipo $\alpha^{s'}$ tal que s seja uma subsequência de s'. Ex: $[\mathtt{a}^{1,0} \to \mathtt{b}/\mathtt{a}^{0}]$.
- A relação de ser subsequência pode ser indireta, portanto é necessário fazer uma verificação transitiva na substituição.

MMo - Exemplo inferência de grupo

```
data Seq a = Nil | Cons a (Seq (a,a)).
len :: Seq a -> Int
len Nil = 0
len (Cons x s) = 1 + 2 * (len s)
```

Σ:

$\{(\texttt{Nil}, (\texttt{Seq a}, \emptyset)),$
$(\texttt{Cons}, (\texttt{b} \to \texttt{Seq (b, b)} \to \texttt{Seq b}, \emptyset)),$
$(\texttt{len}, (\texttt{Seq c} \to \texttt{Int},$
$\quad \{(+):\texttt{Int} \to \texttt{Int} \to \texttt{Int}, \texttt{len}:\texttt{Seq(c, c)} \to \texttt{Int}\}))\}$

Ω:

$\{(\texttt{Seq c}^0 \to \texttt{Int}, \texttt{Seq(c, c)} \to \texttt{Int})\}$

$\mathbb{S}$: $[(\texttt{c, c})/\texttt{c}^0]$

Não tem dependência circular e não modifica os tipos de Σ (ignorando renomeações de variáveis)

Formalização em Coq - Detalhes da formalização

- Formalização parcial e simplificada (sem grupos).
- Trabalho similar para o algoritmo W por (DUBOIS; MÉNISSIER-MORAIN, 1999).
- Uso da formalização da unificação feita por (RIBEIRO; CAMARÃO, 2016).
- Termos, tipos, *schemes* formalizados como tipos indutivos.
- Para garantir que dois tipos generalizados sejam sintaticamente iguais, a respeito da equivalência α, utiliza-se a notação *De Bruijn* nos *schemes*.
- Uso de mônadas de estados e falhas.

Formalização em Coq - Aspectos da terminação

- Coq é uma linguagem de programação funcional total.
- De todas as funções do MMo, a que computa o conjunto Σ é a única que não é de recursão primitiva.
- Até o momento não se conhece alguma relação bem fundada a respeito da efetividade da substituição.
- Utilizar alguma forma de limite de iteração.
- Trabalho similar para o algoritmo W por (DUBOIS; MÉNISSIER-MORAIN, 1999).
- Uso da formalização da unificação feita por (RIBEIRO; CAMARÃO, 2016).
  - Seria necessário descobrir como computar esse número ou, então, associar esse valor às regras de inferência.

Formalização em Coq - Aspectos da consistência

- Uso das regras de Damas-Milner em *syntax-directed* em DUBOIS; MÉNISSIER-MORAIN, 1999).
  - Regras da forma $\Gamma \vdash e : \tau$.
  - Regras de especialização (*spec*) e generalização (*gen*) são fundidas, respectivamente, com as regras (*var*) e (*let*).
- Regra (*fix*+) na forma *syntax-directed*.
- As regras de inferência são representadas em Coq como um tipo (proposição) indutivo.
- O tipo indutivo é uma certificação que pode ser diretamente utilizada num tipo dependente no MMo:

```
Definition infer_dep : forall (e : term) (G : ctx),
    Infer ({tau : ty &
     {g : ctx | has_type (apply_subst_ctx s G) e tau}})
```

Formalização em Coq - Estabilidade da substituição

- O lema da estabilidade da substituição é uma propriedade clássica em sistemas de tipos e é fundamental na prova da consistência.
- Se é verdade que $\Gamma \vdash$ e $: \tau$, então para qualquer substituição $\mathbb{S}$ tem-se $\mathbb{S}\Gamma \vdash$ e $: \mathbb{S}\tau$.
- Casos monomórficos são fáceis. Além do caso polimórfico do `let_ht` há também o caso do `fix_ht`.

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.