Guião para aula laboratorial de Verificação Formal (2019/20)

Coq (3)

O mecanismo de extracção de programas do Coq permite escrever programas funcionais dentro do Coq; usar a lógica de Coq para provar propriedades de correcção sobre esses programas; e, em seguida, fazer a extração do programa para OCaml ou Haskell.

Esta aula é dedicada à especificação e prova de correcção de programas em Coq.

O website do Coq disponibiliza toda a documentação, assim como diversos livros e tutoriais que fornecem uma introdução rápida ao sistema Coq.

1 Programando e provando

Existem duas abordagens para definir funções e fornecer provas de que elas satisfazem uma determinada especificação:

- Definir essas funções com uma *especificação fraca* e, em seguida, adicionar *lemas complementares* que garantam a sua correcção. Por exemplo, definimos uma função $f : A \to B$ e provamos que $\forall x : A, R\, x\, (f\, x)$, onde R é uma relação que codifica o comportamento de entrada/saída pretendido da função.
- Fornecer uma *especificação forte* da função: o tipo dessa função determina directamente que a entrada é um valor x do tipo A e que a saída é a combinação de um valor v do tipo B e uma prova de que v satisfaz $R\, x\, v$. Esta abordagem usa tipos Σ para exprimir as restrições de especificação no tipo de uma função (restringindo o tipo do codomínio aos valores que satisfazem a especificação).

  Ao provar o teorema da especificação forte estamos a construir um habitante (um termo) deste tipo. É então possível extrair o conteúdo computacional dessa prova e assim obter uma função que satisfaz a especificação.

Carregue ficheiro `lesson3.v`. Esse ficheiro ilusta com pequenos exemplos as duas abordagens acima descritas.

Execute, passo a passo, as instruções deste ficheiro e analize o seu efeito. Atente nos comentários lá colocados e no efeito da aplicação de cada táctica de prova na evolução do estado da prova. Consulte o reference manual do Coq, quando necessário

Apresente as provas para os exercícios propostos (em comentário) ao longo do ficheiro.

2 Insertion sort

Analise o caso de estudo do *insertion sort*, onde o programa (Haskell) é extraído da prova de sua especificação.

Apresente as provas para os exercícios propostos (em comentário) ao longo do ficheiro. Execute as instruções deste ficheiro e analize o seu efeito. Atente no efeito da aplicação de cada táctica, tática automática e combinação de tácticas de prova na evolução do estado da prova.

3 Recursão não estrutural

Verificar a conversibilidade entre tipos pode exigir computação com funções recursivas. Por esse motivo, a combinação da não-terminação com tipos dependentes faz com que o *type checking* não seja decidível. Assim, Coq não permite codificar funções que não terminem (para que o *type checking* não possa divergir), nem funções parciais (para que o *type checking* não possa quebrar).

Para impor a decidibilidade do *type checking*, o Coq exige que as funções recursivas sejam codificadas em termos de recursores ou permite formas restritas de expressões de ponto fixo. A maneira usual de assegurar a terminação de expressões de ponto fixo é impor restrições sintáticas, restringindo todas as chamadas recursivas a serem aplicadas a termos estruturalmente menores do que o argumento formal da função.

A partir da versão 8.1, o Coq disponibiliza o comando `Function` que permite codificar diretamente funções recursivas gerais. O comando `Function` é parametrizado como uma *função de medida* que especifica como o argumento "decresce" nas chamadas recursivas da função. Ao definir uma função deste modo, são geradas obrigações de prova que devem ser verificadas para garantir a terminação.

Analise o exemplo da divisão Euclideana e da função merge. Analise também a prova de correcção apresentada para a divisão Euclideana.

4 Exercícios

1. Apresente a prova dos seguintes lemas enunciados no ficheiro `lesson3.v`

   (a) `Lemma Perm_cons :  forall a l1 l2, Perm l1 l2 -> Perm (a::l1) (a::l2).`

   (b) `Lemma Perm_cons_cons :  forall x y l, Perm (x::y::l) (y::x::l).`

2. Apresente a formulação em Coq do teorema que garante a correção da função *merge* definida acima. Não precisa de o provar.

3. Apresente uma especificação forte adquada a cada uma das funções abaixo indicadas. Faça a prova desses teoremas (especificação) e, por fim, proceda à extracção do respectivo programa Haskell.

   (a) Uma função que calcula o comprimento de uma lista.

   (b) Uma função que recebe uma lista de pares, em que a segunda componente dos pares é um número natural, e produz o somatório de todos esses números naturais.