Verificação Formal (2021/22)

Coq (2)

A indução é uma técnica que nos permite definir e raciocinar com objectos que são *estruturados* (de uma forma bem fundada) e *finitos* (embora arbitrariamente grandes). A indução explora a natureza finita e estruturada desses objectos para superar a sua complexidade arbitrária.

Nesta aula iremos trabalhar com funções e predicados definidos indutivamente, sendo um dos seus objectivos o desenvolvimento de provas por indução. Adicionalmente iremos ilustar alguns aspectos práticos mais avançados do sistema Coq, e explorar novas tácticas de prova e maneiras de as combinar.

O website do Coq disponibiliza toda a documentação.

1 Aspectos práticos

Comece por invocar o Coq e carregue o ficheiro `lesson2a.v`.

Neste ficheiro vão sendo feitas experiências com pequenos exemplos (acompanhando o exposto nos slides) sobre os eliminadores do tipo `nat`, assim como experiências sobre aspectos práticos do Coq: notação, obrecarga, escopo de interpretação, argumentos implícitos, comandos de pesquisa de lemas, etc.

Execute, passo a passo, as instruções deste ficheiro e analize o seu efeito. Atente nos comentários lá colocados e no efeito execução dos comandos.

2 Provas sobre listas e números naturais

Carregue agora ficheiro `lesson2b.v`. Esse ficheiro faz uso da biblioteca `List`, e nele se apresentam algumas definições e provas de várias propriedades sobre listas.

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.

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

Crie agora um novo ficheiro Coq para desenvolver os problemas que a seguir se apresentam.

3 Funções sobre listas

Vamos usar a biblioteca `List`.

Começe por definir uma função `sum` que calcula o somatório de uma lista de números naturais. Experimente aplica-la a listas concretas e verifique o resultado obtido.

Analise agora a definição das funções `++`, `rev`, `length` e `map`, já definidas na biblioteca `List` e seguinte propriedades:

1. `forall l1 l2, sum (l1 ++ l2) = sum l1 + sum l2`
2. `forall (A:Type) (l:list A), length (rev l) = length l`
3. `forall (A B:Type) (f:A->B) (l:list A), rev (map f l) = map f (rev l)`

4 Predicados sobre listas

Nos exemplos do ficheiro `lesson2b.v` tem a definição do predicado `In`, que testa se um elemento pertence a uma lista, definido como uma função recursiva:

```
Fixpoint In (A:Set) (a:A) (l:list A) {struct l} : Prop :=
    match l with
      | nil => False
      | cons x xs => x = a \/ In a xs
    end.
```

Essa é uma abordagem possível, mas pouco usual em Coq. O que é aconselhado fazer na codificação em Coq de proprieades é defini-las como um tipo inductivo (e não como uma função recursiva como foi feito acima). A justificação para esta opção deve-se ao facto de ser muito mais simples desenvolver provas que envolvam essas propriedades, uma vez que o Coq gera automaticamente os princípios de indução para esses tipos, e ficamos assim com mais ferramentas para desenvolver as provas.

Considere o seguinte predicado definido indutivamente:

```
Inductive In (A:Type) (y:A) : list A -> Prop :=
| InHead : forall xs:list A, In y (cons y xs)
| InTail : forall (x:A) (xs:list A), In y xs -> In y (cons x xs).
```

Prove as seguintes propriedades:

1. `forall (A:Type) (a b : A) (l : list A), In b (a :: l) -> a = b \/ In b l`
2. `forall (A:Type) (l1 l2: list A) (x:A), In x l1 \/ In x l2 -> In x (l1 ++ l2)`
3. `forall (A:Type) (x:A) (l:list A), In x l -> In x (rev l)`
4. `forall (A B:Type) (y:B) (f:A->B) (l:list A), In y (map f l) -> exists x, In x l /\ y = f x`

5 Prefix

Considere o seguinte predicado definido indutivamente:

```
Inductive Prefix (A:Type) : list A -> list A -> Prop :=
| PreNil : forall l:list A, Prefix nil l
| PreCons : forall (x:A) (l1 l2:list A), Prefix l1 l2 -> Prefix (x::l1) (x::l2).
```

Como o nome já sugere, `(Prefix l1 l2)` indica que a lista `l1` é um prefixo da lista `l2`.

Prove as seguintes propriedades:

1. `forall (A:Type) (l1 l2:list A), Prefix l1 l2 -> length l1 <= length l2`
2. `forall l1 l2, Prefix l1 l2 -> sum l1 <= sum l2`
3. `forall (A:Type) (l1 l2:list A) (x:A),  (In x l1) /\ (Prefix l1 l2) -> In x l2`

6 Sublist

Considere o seguinte predicado que codifica a relação de sublista:

```
Inductive SubList (A:Type) : list A -> list A -> Prop :=
| SLnil : forall l:list A, SubList nil l
| SLcons1 : forall (x:A) (l1 l2:list A), SubList l1 l2 -> SubList (x::l1) (x::l2)
| SLcons2 : forall (x:A) (l1 l2:list A), SubList l1 l2 -> SubList l1 (x::l2).
```

Prove as seguintes propriedades:

1. `SubList (1::3::nil) (3::1::2::3::4::nil)`
2. `forall (A:Type)(l1 l2 l3 l4:list A),`
   `SubList l1 l2 -> SubList l3 l4 -> SubList (l1++l3) (l2++l4)`
3. `forall (A:Type) (l1 l2:list A), SubList l1 l2 -> SubList (rev l1) (rev l2)`
4. `forall (A:Type) (x:A) (l1 l2:list A), SubList l1 l2 -> In x l1 -> In x l2`