Verificação Formal (2023/24)

Coq (2)

Crie um ficheiro Coq e começe por carregar a biblioteca `List`.

```
Require Import List.
```

1 Funções sobre listas

Defina 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 as funções `++`, `rev`, `length` e `map`, já definidas na biblioteca `List` e prove as seguintes 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)`

2 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.
```

Uma outra abordagem é definir o predicado `In` como um tipo indutivo. 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`

3 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`

4 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`