

MFES - Verificação Formal de Software

MEI 2012/13

Exercícios de Coq

- Apresente provas em Coq de, no mínimo, 3 alíneas de cada uma das partes.
- Na resolução dos exercícios da parte I, evite aplicar táticas automáticas.
- A resolução dos exercícios deve ser enviada, por mail, até ao dia **10 de Junho**.

Parte I - Raciocínio Lógico

1. Prove as seguintes tautologias da lógica proposicional:

- (a) $(A \vee B) \vee C \rightarrow A \vee (B \vee C)$
- (b) $(B \rightarrow C) \rightarrow A \vee B \rightarrow A \vee C$
- (c) $(A \wedge B) \wedge C \rightarrow A \wedge (B \wedge C)$
- (d) $A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$
- (e) $(A \wedge B) \vee (A \wedge C) \leftrightarrow A \wedge (B \vee C)$
- (f) $(A \vee B) \wedge (A \vee C) \leftrightarrow A \vee (B \wedge C)$

2. Prove os seguintes teoremas da lógica de primeira ordem:

- (a) $(\exists x.P(x) \wedge Q(x)) \rightarrow (\exists x.P(x)) \wedge (\exists x.Q(x))$
- (b) $(\exists x.\forall y.P(x, y)) \rightarrow \forall y.\exists x.P(x, y)$
- (c) $(\exists x.P(x) \rightarrow (\forall x.\forall y.P(x) \rightarrow Q(y)) \rightarrow \forall y.Q(y)$
- (d) $(\forall x.Q(x) \rightarrow R(x)) \rightarrow (\exists x.P(x) \wedge Q(x)) \rightarrow \exists x.P(x) \wedge R(x)$
- (e) $(\forall x.P(x) \rightarrow Q(x)) \rightarrow (\exists x.P(x)) \rightarrow \exists y.Q(y)$
- (f) $(\exists x.P(x)) \vee (\exists x.Q(x)) \leftrightarrow (\exists x.P(x) \vee Q(x))$

3. Assumindo o *princípio do meio excluído* como axioma, prove que:

- (a) $((A \rightarrow B) \rightarrow A) \rightarrow A$ (*lema de Pierce*)
- (b) $\neg\neg A \rightarrow A$
- (c) $\neg\forall x.P(x) \rightarrow \exists x.\neg P(x)$

Parte II - Raciocínio Indutivo

Os exercícios que se seguem requerem as bibliotecas `ZArith` e `Lists`.

1. Defina a função `sum` que calcula o somatório de uma lista de inteiros e prove as seguintes propriedades:

- (a) `forall l1 l2, sum (l1 ++ l2) = sum l1 + sum l2`
- (b) `forall l, sum (rev l) = sum l`
- (c) `forall l1 l2, Prefix l1 l2 -> sum l1 <= sum l2` (Nota: `Prefix` está definido em 4.)

2. Prove as seguintes propriedades sobre listas:

- (a) `forall (A:Type) (l:list A), length (rev l) = length l`
- (b) `forall (A B:Type) (f:A->B) (l:list A), length (map f l) = length l`
- (c) `forall (A B:Type) (f:A->B) (l:list A), rev (map f l) = map f (rev l)`

3. 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:

- (a) `forall (A:Type) (x:A) (l:list A), In x l -> In x (rev l)`
- (b) `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`
- (c) `forall (A:Type) (x:A) (l : list A), In x l -> exists l1, exists l2, l = l1 ++ (x::l2)`

4. 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).
```

Prove as seguintes propriedades:

- (a) `forall (A:Type) (l1 l2:list A), Prefix l1 l2 -> length l1 <= length l2`
- (b) `forall (A B:Type) (f: A->B) (l1 l2:list A), Prefix l1 l2 -> Prefix (map f l1) (map f l2)`
- (c) `forall (A:Type) (l1 l2:list A), Prefix l1 (l1++l2)`
- (d) `forall (A:Type) (l1 l2:list A) (x:A), Prefix l1 l2 -> In x l1 -> In x l2`
- (e) A relação `Prefix` é uma relação de ordem:
 - i. `forall (A:Type) (l:list A), Prefix l l`
 - ii. `forall (A:Type) (l1 l2 l3:list A), Prefix l1 l2 /\ Prefix l2 l3 -> Prefix l1 l3`
 - iii. `forall (A:Type) (l1 l2:list A), Prefix l1 l2 /\ Prefix l2 l1 -> l1=l2`

5. Considere o seguinte predicado definido indutivamente:

```
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:

- (a) `forall (A:Type) (l1 l2:list A), SubList l1 l2 -> SubList (rev l1) (rev l2)`
- (b) `forall (A:Type) (x:A) (l1 l2:list A), SubList l1 l2 -> In x l1 -> In x l2`
- (c) `forall (A:Type) (x:A) (l1 l2:list A), Prefix l1 l2 -> SubList l1 l2`
- (d) `forall (A:Type)(l1 l2 l3 l4:list A), SubList l1 l2 -> SubList l3 l4 -> SubList (l1++l3) (l2++l4)`
- (e) `forall (A B:Type) (f:A->B) (l1 l2:list A), SubList l1 l2 -> SubList (map f l1) (map f l2)`