Université Bordeaux 1. Master d'informatique.
Logique INF462

Décembre 2010

Sujet de M. Castéran Tous documents autorisés ; durée conseillée : 1h30.
Les 3 exercices sont indépendants

À lire avant d'écrire quoi que ce soit *Le critère d'évaluation sera la qualité et la clarté de vos explications. Les commandes* Coq *non accompagnées d'explications ne seront pas prises en compte ; il vaut mieux expliquer simplement la structure de vos preuves ; par exemple vous pouvez donner des explications de la forme «* En appliquant telle tactique, les nouveaux buts suivants sont engendrés *». Si vous oubliez la syntaxe précise d'une tactique, signalez-le,*

1 Exercice (3pts / 10)

On considère deux variables `P` et `Q` de type `Prop`. Donner des preuves (sans utiliser de tactique automatique) des énoncés suivants :

```
Lemma L1 : (P -> ~P) -> (P -> Q).

Lemma L2 : ((P ∨ ~Q) ∧ Q) -> P.
```

2 Exercice (2 pts / 10)

On considère les declarations suivantes :

```
Section S2.
 Variables (A B C: Set)
           (f : A -> B)
           (g : B -> C).

 Hypothesis gf_surjective : forall c:C, exists a:A, g (f a) = c.
```

2.1

Prouver le théorème suivant :

```
Theorem g_surjective : forall  c:C, exists b:B, g b = c.
```

3 Exercice (5 pts / 10)

3.1

Que calculent les fonctions suivantes ?

```
Require Import List.

Fixpoint take_first n (l:list nat) :=
match n, l with 0, _ => nil
              | S p, nil => nil
              | S p, a::l' => a :: take_first p l'
end.

Fixpoint take_last n (l:list nat) :=
match n, l with 0, l' => l'
              | S p, nil => nil
              | S p, a::l' =>  take_last p l'
end.
```

3.2

Écrire une version *polymorphe* de ces deux fonctions (c'est-à-dire fonctionnant sur des listes ayant n'importe quel type d'éléments).

3.3

On remarque qu'en concaténant les résultats de `take_first` et `take_last` (pour une même liste et un même argument numérique), on obtient la liste d'origine. Énoncer cette propriété en *Coq* (avec vos versions polymorphes), et en donner la preuve.