UPPSALA UNIVERSITET
Matematiska institutionen
Erik Palmgren

LABORATORY EXERCISE 2
= ASSIGNMENT 3
Applied Logic, Fall 2010
Computer Assisted Theorem Proving
2010-09-23

Laboratory exercise 2 = Assignment 3

The second laboratory exercise is about using some built in tactics and libraries of Coq. You may work in groups of up to three persons. In the below Coq scripts are five theorems to be proved in Coq (problem 1-5). Run the scripts and fill in the missing proofs. You need to consult the documentation for the standard library (see: coq.inria.fr)

Hand in finished and annotated proofs at the latest October 7, 2010.

```
Section Omega_tactics_lab.

Require Import ZArith.

Open Scope Z_scope.

(* Try to prove following with the help of the omega
tactic *)

Theorem problem1:
(exists u:Z,(forall x y:Z,
  y+x <6 /\ y < x+3 /\ 2*x < 5*y -> x<u /\ y <u)).
(* proof ? *)

Definition P (u:Z) :=
(forall x y:Z,
  y+x <6 /\ y < x+3 /\ 2*x < 5*y -> x<u /\ y <u).

Definition Q (x y u:Z) :=
y+x <6 /\ y < x+3 /\ 2*x < 5*y -> x<u /\ y <u.

Theorem problem2: (forall u v: Z, u <v /\ P u -> P v).
(* proof ? *)
```

```
(* Prove there is a least u satisfying P u. *)

Theorem problem3: (exists u:Z, P u /\
     (forall v:Z, P v -> u <= v)).
(* proof ? *)
```

Here we introduce a new inductive data structure and some operations on it.

```
Section Trees.

Require Import List.

Open Scope list_scope.

(* Binary trees with elements from A as leaves *)

Inductive bt (A:Set) : Set :=
  leaf (a: A) | node (l: (bt A)) (r: (bt A)).

(* Prove an induction scheme using the induction
command *)

Theorem bt_IND (A:Set)(P:(bt A) -> Prop):
(forall a:A, P (leaf A a)) ->
(forall l:(bt A), (forall r:(bt A),
 (P l) -> (P r) -> (P (node A l r)))) ->
  (forall t: (bt A), P t).
intros A P.
intros Hbase Hstep.
induction t.
apply Hbase.
apply Hstep.
assumption.
assumption.
Qed.
```

```
Fixpoint twist (A:Set) (t : bt A) :=
match t with
|(leaf a) => (leaf A a)
|(node l r) => (node A (twist A r) (twist A l))
end.

Fixpoint flatten (A:Set) (t: bt A) : (list A) :=
match t with
|(leaf a) => a::nil
|(node l r) => (flatten A l) ++ (flatten A r)
end.

Variable U:Set.
Variables a b c d: U.

Definition tree1 : (bt U) := (node U (node U (leaf U a) (leaf U b))
(node U (leaf U c) (leaf U d))).

Eval compute in (twist U tree1).
Eval compute in (flatten U tree1).

(* rev = reverse list operation *)

Eval compute in (rev (a::b::c::nil)).
Eval compute in (flatten U (twist U tree1)).

Theorem problem4: (forall A:Set,
(forall t: (bt A), (twist A (twist A t)) = t)).
intro A.
induction t.
(* simplify using computation rules *)
simpl.
 (* Finish the proof! *)


Theorem problem5: (forall A:Set,
(forall t: (bt A),
(flatten A (twist A t)) = (rev (flatten A t)))).
(* Proof ? *)
```