CompCert, a fully-verified optimizing compiler for C

CertiKos, a fully verified hypervisor, for proving the correctness of subtle algorithms involving floating point numbers, and as the basis for CertiCrypt

CertiCrypt, an environment for reasoning about the security of cryptographic algorithms

Inductive day : Type :=  
  | monday  
  | tuesday  
  | wednesday  
  | thursday  
  | friday  
  | saturday  
  | sunday.

Definition orb (b1:bool) (b2:bool) : bool :=  
  match b1 with  
  | true ⇒ true  
  | false ⇒ b2  
  end.

Example test\_orb1: (orb true false) = true.  
Proof. simpl. reflexivity. Qed.

Notation "x && y" := (andb x y).

Check (S (S (S (S O)))).

Fixpoint eqb (n m : nat) : bool :=  
  match n with  
  | O ⇒ match m with  
         | O ⇒ true  
         | S m' ⇒ false  
         end  
  | S n' ⇒ match m with  
            | O ⇒ false  
            | S m' ⇒ eqb n' m'  
            end  
  end.

The keywords intros, simpl, and reflexivity are examples of *tactics*

Theorem plus\_id\_example : forall n m:nat,

n = m ->

n + n = m + m.

Proof.  
  (\* move both quantifiers into the context: \*)  
  intros n m.  
  (\* move the hypothesis into the context: \*)  
  intros H.  
  (\* rewrite the goal using the hypothesis: \*)  
  rewrite → H.  
  reflexivity. Qed.

Proof.  
  intros n. destruct n as [| n'] eqn:E.  
  - reflexivity.  
  - reflexivity. Qed.

List of Proved Theorems:

* Basics:
  + mult\_S\_1 : forall n m : nat, m = S n -> m \* (1 + n) = m \* m.
  + plus\_1\_neq\_0 : forall n : nat, (n + 1) =? 0 = false.
  + negb\_involutive : forall b : bool, negb (negb b) = b.
  + andb\_commutative : forall b c, andb b c = andb c b.
  + andb3\_exchange : forall b c d, andb (andb b c) d = andb (andb b d) c.
  + andb\_true\_elim2 : forall b c : bool, andb b c = true -> c = true.
  + identity\_fn\_applied\_twice : forall (f : bool -> bool),

(forall (x : bool), f x = x) -> forall (b : bool), f(f b) = b.

* + andb\_eq\_orb : forall (b c : bool), (andb b c = orb b c) -> b = c.
* Proof by Induction:
  + plus\_n\_O : forall n:nat, n = n + 0.
  + minus\_diag : forall n, minus n n = 0.
  + mult\_0\_r : forall n:nat, n \* 0 = 0.
  + plus\_n\_Sm : forall n m : nat, S (n + m) = n + (S m).
  + plus\_comm : forall n m : nat, n + m = m + n.
  + plus\_assoc : forall n m p : nat, n + (m + p) = (n + m) + p.
  + evenb\_S : forall n : nat, evenb (S n) = negb (evenb n).
  + plus\_swap : forall n m p : nat, n + (m + p) = m + (n + p).
  + mult\_comm : forall m n : nat, m \* n = n \* m.
  + mult\_1\_l : forall n:nat, 1 \* n = n.
  + mult\_plus\_distr\_r : forall n m p : nat, (n + m) \* p = (n \* p) + (m \* p).
  + mult\_assoc : forall n m p : nat, n \* (m \* p) = (n \* m) \* p.
  + eqb\_refl : forall n : nat, true = (n =? n).
  + bin\_to\_nat\_pres\_incr: forall n: bin, bin\_to\_nat (incr n) = S (bin\_to\_nat n).
  + nat\_bin\_nat : forall n, bin\_to\_nat (nat\_to\_bin n) = n.
* Lists:
  + app\_assoc : forall l1 l2 l3 : natlist, (l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).
  + app\_length : forall l1 l2 : natlist, length (l1 ++ l2) = (length l1) + (length l2).
  + rev\_length : forall l : natlist, length (rev l) = length l.
  + app\_nil\_r : forall l : natlist, l ++ [] = l.

Arguments nil {X}.  
Arguments cons {X} \_ \_.  
Arguments repeat {X} x count.

Fixpoint repeat''' {X : Type} (x : X) (count : nat) : list X :=  
  match count with  
  | 0 ⇒ nil  
  | S count' ⇒ cons x (repeat''' x count')  
  end.