classmate

Date 28.1.2020
Page 1

FUNCTIONAL PROG. IN COQ

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

Definition next_weekday (d: day) : day :=
  match d with
  | monday => tuesday
  | tuesday => wednesday
  | wednesday => thursday
  | thursday => friday
  | friday => monday
  | saturday => monday
  | sunday => monday
  end.

Compute (next_weekday friday).
// monday : day

Compute (next_weekday (next_weekday saturday)).
// tuesday: day
```

Date 28.1.2020
Page 2

Example test_next_weekday:
(next_weekday (next_weekday saturday)) = tuesday.
Proof. simpl. reflexivity. Qed.

```
Inductive bool : Type :=
  | true
  | false.

Definition negb (b:bool) : bool :=
  match b with
  | true => false
  | false => true
  end.

Definition andb (b1:bool) (b2:bool) : bool :=
  match b1 with
  | true => b2
  | false => false
  end.

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.

Example test_orb2 : (orb false false) = false.
Proof. simpl. reflexifity. Qed.

Example test_orb3 : (orb false true) = true.
Proof. simpl. reflexivity. Qed.

Example test_orb4 : (orb true true) = true.
Proof. simpl. reflexivity. Qed.


Notation "x && y" := (andb x y).
Notation "x || y" := (orb x y).

Example test_orb5 : false || false || false || true
  = true.
Proof. simpl. reflexivity. Qed.
```

NOTE: simple tactic won't expand multi-level function calls. for that we may use unfold <fn>.

```
Definition nandb (b1 : bool) (b2 : bool) : bool :=
  match b1 with
  | false => true
  | true  => (negb b2)
  end.
```

Example test_nandb1 : (nandb true false) = true.
Proof. reflexivity. Qed.

Example test_nandb2: (nandb false false) = true.
Proof. reflexivity. Qed.

Example test_nandb3 : (nandb false true) = true.
Proof. reflexivity. Qed.

Example test_nandb4: (nandb true true) = false.
Proof. reflexivity. Qed.

```
Definition andb3 (b1 : bool) (b2 : bool) (b3 : bool) : bool :=
  match b1 with
  | true  => (andb b2 b3)
  | false => false
  end.
```

... examples.

TYPES

Check true.
(* ===> true : bool *)

Check (negb true).
(* ===> negb true : bool *)

```
Check negb.
(* ===> negb : bool -> bool *)
```

NEW TYPES FROM OLD

```
Inductive rgb : Type :=
  | red
  | green
  | blue.

Inductive color : Type :=
  | black
  | white
  | primary (p : rgb).

Definition monochrome (c : color) : bool :=
  match c with
  | black => true
  | white => true
  | primary q => false
  end.

Definition isred (c : color) : bool :=
  match c with
  | black => false
  | white => false
  | primary red => true
  | primary _ => false
  end.
```

TUPLES.

```
Inductive bit : Type :=
  | B0
  | B1.

Inductive nybble : Type :=
  | bits (b0 b1 b2 b3 : bit).

Check (bits B1 B0 B1 B0).
(* ==> bits B1 B0 B1 B0 : nybble *)

Definition all-zero (nb : nybble) : bool :=
  match nb with
  | (bits B0 B0 B0 B0) => true
  | (bits _ _ _ _) => false
  end.

Compute (all-zero (bits B1 B0 B1 B0)).
(* ===> false : bool *)

Compute (all-zero (bits B0 B0 B0 B0)).
(* ===> true : bool *)
```

MODULES

```
Module NatPlayground.
```

NUMBERS

```
Inductive nat : Type :=
  | O
  | S (n : nat).

Inductive nat' : Type :=
  | stop
  | tick (foo : nat').

Definition pred (n : nat) : nat :=
  match n with
  | O   => O
  | S n' => n'
  end.

End NatPlayground.

Check (S (S (S (S O)))).
(* ===> 4 : nat *)
```

```
Definition minustwo (n : nat) : nat :=
  match n with
  | O => O
  | S O => O
  | S (S n') => n'
  end.

Compute (minustwo 4).
(* ===> 2 : nat *)

Check S.
Check pred.
Check minustwo.

Fixpoint evenb (n : nat) : bool :=
  match n with
  | O => true
  | S O => false
  | S (S n') => (evenb n')
  end.

Definition oddb (n : nat) : bool :=
  (negb (evenb n)).
```

```
Example test_oddb1 : oddb 1 = true.
Proof. simpl. reflexivity. Qed.

Example test_oddb2 : oddb 4 = false.
Proof. simpl. reflexivity. Qed.


Module NatPlayground2.

Fixpoint plus (n : nat) (m : nat) : nat :=
  match n with
  | O => m
  | S n' => S (plus n' m)
  end.

Compute (plus 3 2).


Fixpoint mult (n m : nat) : nat :=
  match n with
  | O => O
  | S n' => (plus m (mult n' m))
  end.

Example test_mult1: (mult 3 3) = 9.
Proof. simpl. reflexivity. Qed.
```

```
Fixpoint minus (n m : nat) : nat :=
  match n, m with
  | O, _ => O
  | S_, O => n
  | S n', S m' => (minus n' m')
  end.

End NatPlayground2.

Fixpoint exp (base power : nat) : nat :=
  match power with
  | O => S O
  | S p => (mult base (exp base p))
  end.


Fixpoint factorial (n : nat) : nat :=
  match n with
  | O => S O ;
  | S n' => (mult n (factorial n'))
  end.


Example test_factorial1 : (factorial 3) = 6.
Proof. reflexivity. Qed.

Example test_factorial2 : (factorial 5) = (mult 10 12).
Proof. reflexivity. Qed.
```

```
Notation "x+y" := (plus x y)
                  (at level 50, left associativity).
                  : nat_scope.

Notation "x-y" := (minus x y)
                  (at level 50, left associativity)
                  : nat_scope

Notation "x * y" := (mult x y)
                  (at level 40, left associativity)
                  : nat_scope.

Check ((0+1)+1).

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

```
Fixpoint leb (n m : nat) : nat :=
  match n with
  | O => match m with true
         | O => ~~false~~ true
         | S _ => true
  | S n' => match m with
         | O => false
         | S m' => (leb n' m')
         end
  end.

Definition ltb (n m : nat)
  match (leb n m) with
  | false => false
  | true => match (leb m n) with
            | true => false
            | false => true
            end
  end.

Theorem plus_O_n : forall n : nat, O+n=n.
Proof.
  intros n.
  simpl.
  reflexivity.
Qed.
```