CSE-433: Inductive Datatypes

gla@postech

In this assignment, you will practice inductive datatypes and Leibniz equality in Coq. We list some commands and tactics that you may need for this assignment. Examples of using these commands and tactics are also given.

See `coq4.v` for problem 1 (30 points) and problem 2 (30 points). See `coq4-2.v` for problem 3 (10 points), problem 4 (10 points), problem 5 (10 points), and problem 6 (10 points).

1 Proving properties of `plus`

Prove the following lemmas in Coq.

```
Lemma plus_O_n : forall n:nat, n = plus O n.
Lemma plus_n_O : forall n:nat, n = plus n O.
Lemma plus_n_S : forall n m:nat, S (plus n m) = plus n (S m).
Lemma plus_com : forall n m:nat, plus n m = plus m n.
Lemma plus_assoc : forall (m n l:nat), plus (plus m n) l = plus m (plus n l).
```

You want to use the following tactics in your proofs.

- `induction`
  When applied to a goal of the form `forall x:T, A(x)`, it creates new subgoals according to the inductive definition of type `T`. The difference from the `destruct` tactic is that the `induction` tactic applies the elimination rule based on induction and hence automatically creates induction hypotheses.
  `induction n`

- `rewrite`
  `rewrite e` requires `e` to be of type `forall (x1:T1) (x2:T2) ... (xn:Tn), a = b`. Then applying `rewrite e` to a goal of the form `P(a)` rewrites it as `P(b)`.
  `rewrite Heq`
  `rewrite <- plus_n_O`
  `rewrite -> plus_n_O` (which is equivalent to `rewrite plus_n_O`)
  `rewrite <- (plus_n_O n0)`
  `rewrite -> (plus_n_O n0)` (which is equivalent to `rewrite (plus_n_O n0)`)
  `rewrite Heq in H1`
  Note that if `rewrite plus_n_O` is applicable to several parts of the current goal, Coq applies the tacics to the first matching part by default. You can circumvent this limitation by providing a more specific proof term, for example, by using `rewrite (plus_n_O n0)`.

- `simpl`
  `simpl` simplifies terms in the current goal using the definition of its subterms. For example, it simplifies `plus O n` to `n`.
  `simpl`
  `simpl plus`
  `simpl plus at 1`

- `reflexivity`
  Applying this tactic to a goal of `t1 = t2` immediately completes the proof if `t1` and `t2` can be converted to each other (*e.g.*, `6*6=9*4`) by the `simpl` tactic.

- `replace`
  `replace e with e'` replaces `e` in the current goal by `e'` and creates a new goal `e' = e`.
  ```
  replace (f 1) with 0
  replace (f 1) with (f 0)
  ```
  You can complete this assignment without using the `replace` tactic, if you make good use of the `rewrite` tactic.

2 Proving $2 * \Sigma_{i=0}^{n} i = n + n * n$

Prove the following theorem in Coq.

```
Theorem sum_n_plus : forall n:nat, double (sum_n n) = plus n (mult n n).
```

Your proof may use any lemma from the previous part. You will need to introduce extras lemmas to complete the proof. The sample solution, for examples, introduces three lemmas, one of which is:

```
Lemma double_plus2 : forall n:nat, double n = plus n n.
```

3 Proving properties of `plus` using the default datatype `nat`

You prove the previous properties of `plus` using the default datatype `nat` provided by Coq.

4 Proving $2 * \Sigma_{i=0}^{n} i = n + n * n$ using the default datatype `nat`

As we have redefined the function `double` (which is no longer a recursive function), you need the tactic `unfold` in your proof. Apply this tactic to expand `double e` to `e + e`.

- `unfold`
  `unfold x` expands `x` into its definition.
  ```
  unfold double
  unfold element at 1
  unfold element in H
  ```

5 Proving $2 * \Sigma_{i=0}^{n} i = n + n * n$ using the `Arith` library

The goal is to familiarize yourself with the two commands `SearchPattern` and `SearchRewrite` which enable you to find theorems of particular form that have already been proven and are available in the library.

- `SearchPattern`
- `SearchRewrite`

6 Proving $2 * \Sigma_{i=0}^{n} i = n + n * n$ using `ring_simplify`

The goal is to learn the tactic `ring_simplify` and `ring`.

- `ring_simplify`
- `ring`