CSE-433 Assignment: First Order Logic

gla@postech

In this assignment, you will prove theorems in pure first-order logic (Section `FirstOrder`).

Tactics for universal and existential quantifications

The following table shows tactics for universal and existential quantifications in first-order logic:

	$\forall$ (`forall`)	$\exists$ (`exists`)
Introduction	`intro`	`exists`
Elimination	`apply`, `apply ... with` *term*$_1$ *term*$_2$ $\cdots$ *term*$_n$	`elim`, `destruct`

We transcribe the proofs of the following theorems (given in the Course Notes) into Coq:

$$(\forall x.A \wedge B) \supset (\forall x.A) \wedge (\forall x.B) \ \textit{true}$$
$$\exists x.\neg A \supset \neg\forall x.A \ \textit{true}$$
$$\forall y.(\forall x.A) \supset (\exists x.A) \ \textit{true}$$

```
Section FirstOrder.
Variable Term : Set.
Variables A B : Term -> Prop.

Theorem forall_and :
(forall x : Term, A x /\ B x) -> (forall x : Term, A x)  /\ (forall x : Term, B x).
Proof.
intro w.
split; (intro a; elim (w a); intros; assumption).
Qed.

Theorem exist_neg : (exists x : Term, ~ A x) -> (~ forall x : Term, A x).
Proof.
intro w; intro z; elim w; intros a y; elim y; apply z.
Qed.

Theorem not_weird : forall y : Term, (forall x : Term, A x) -> (exists x : Term, A x).
Proof.
intro a; intro w; exists a; apply w.
Qed.
End FirstOrder.
```

We can exploit the `destruct` tactic to simplify the proofs of the first two theorems:

```
Theorem forall_and :
(forall x : Term, A x /\ B x) -> (forall x : Term, A x)  /\ (forall x : Term, B x).
Proof.
intro H.
split; intro x; destruct (H x) as [Ha Hb]; assumption.
Qed.

Theorem exist_neg : (exists x : Term, ~ A x) -> (~ forall x : Term, A x).
```

```
Proof.
intros H H'.
destruct H as [x Hx].
exact (Hx (H' x)).
Qed.
```

First we declare a set `Term` which we will use as the set of terms:

```
Variable Term : Set.
```

We do not actually specify elements of the set `Term` because pure first-order logic does not assume a particular set of terms.

Next we declare two predicates `A` and `B`:

```
Variables A B : Term -> Prop.
```

`A` and `B` are both given type `Term -> Prop` to indicate that they are parameterized over elements of the set `Term`, or terms. We write `A x` for the proposition that `A` instantiates to when applied to term `x`. Note that all term variables in my Coq program are assigned type `Term` so that they can be used as arguments to `A` and `B`.

Sets, propositions, and types

You might well be confused about the differences between `Set` for sets, `Prop` for propositions, and `Type` for types in Coq. To tell the truth, these are all types *and also* terms in Coq — what a convoluted system it is! For now, we only need the following facts. The invariant is that everything in Coq has its type!

- A proof term M, or equivalently a proof, has a certain type A, and we call A a proposition. So we have a relation $M : A$.
- A proposition A has type `Prop`, and we call `Prop` a *sort* in order to differentiate it from types in the general sense. So we have a relation $A :$ `Prop`, which literally says that A belongs to the set `Prop` of propositions.
- A term t has a certain type τ, and we call τ a datatype. So we have a relation $t : \tau$.
- A datatype τ has type `Set`, and we also call `Set` a *sort* in order to differentiate it from types in general sense. So we have a relation $\tau :$ `Set`, which literally says that τ belongs to the set `Set` of datatypes.
- Both `Type` and `Set` have type `Type`!

We can summarize the above relations as follows:

term t	:	datatype τ	:	`Set`	:	`Type`
proof term M	:	proposition A	:	`Prop`	:	`Type`

Declarations and definitions

The following table shows how to declare term variables with only their datatypes, and how to define term variables with terms as well as their datatypes. Global declarations and definitions are exported to the outside of sections (beginning with `Section` and ending with `End`), while local declarations and definitions are not.

	declaration	definition
global	`Parameter v :` τ, `Parameters`	`Definition c :` τ `:=` t.
local	`Variable v :` τ, `Variables`	`Let c :` τ `:=` t.

It turns out that these definitions and declarations can be used not only for terms but also for proof terms and even for datatypes and propositions! For example, we have seen an example of declaring a datatype like

```
Variable Term : Set.
```

or declaring a proposition like

```
Variable P : Prop.
```

For proofs and proof terms, Coq provides the following specialized forms for declarations and definitions. An opaque definition hides its proof M and makes only `H` and A visible for later use. A transparent definition makes visible its proof M as well. If you do not understand what the difference is, just use opaque definitions in your Coq program and you will never run into trouble!

	declaration	definition
global	`Axiom H :` A (`Parameter H :` A — not recommended)	`Lemma H :` A`. Proof` M`.` — opaque `Theorem H :` A`. Proof` M`.` — opaque (`Definition H :` A `:=` M`.` — transparent, not recommended)
local	`Hypothesis H :` A, `Hypotheses` (`Variable H :` A — not recommended)	`Let H :` A `:=` M`.` — transparent

`apply`, `elim`, `destruct`, and `exact`

In general, arguments to these tactics can be proof terms as long as they have proper types. Here are a few examples.

- `apply (Ltn O (S O)).`
  Instead of specifying a label, we use a proof term `Ltn O (S O)`.
- `elim (EM (exists x, ~ P x)).`
  Instead of specifying a label, we use a proof term `EM (exists x, ~ P x)`
- `exact (Eqi a).`
  Instead of specifying a label, we use use a proof term `Eqi a`.

1 Properties of natural numbers (50 points)

We use the following axioms to characterize natural numbers (which are all given in the Course Notes).

$$\frac{}{Nat(\mathbf{0})\ true}\ Zero \qquad \frac{}{\forall x.Nat(x) \supset Nat(\mathbf{s}(x))\ true}\ Succ$$

$$\frac{}{\forall x.Eq(x,x)\ true}\ Eq_i \qquad \frac{}{\forall x.\forall y.\forall z.(Eq(x,y) \wedge Eq(x,z)) \supset Eq(y,z)\ true}\ Eq_t$$

$$\frac{}{\forall x.Lt(x,\mathbf{s}(x))\ true}\ Lt_s \qquad \frac{}{\forall x.\forall y.Eq(x,y) \supset \neg Lt(x,y)\ true}\ Lt_\neg$$

We translate these axioms into Coq declarations as follows:

```
Variable Term : Set.

Variable O : Term.
Variable S : Term -> Term.

Variable Nat : Term -> Prop.
Variable Eq : Term -> Term -> Prop.
Variable Lt : Term -> Term -> Prop.
```

```
Hypothesis Zero : Nat O.
Hypothesis Succ : forall x : Term, Nat x -> Nat (S x).
Hypothesis Eqi : forall x : Term, Eq x x.
Hypothesis Eqt : forall (x : Term) (y : Term) (z: Term), (Eq x y /\ Eq x z) -> Eq y z.
Hypothesis Lts : forall x : Term, Lt x (S x).
Hypothesis Ltn : forall (x : Term) (y : Term), Eq x y -> ~ Lt x y.
```

Proofs of the following theorems are given in the Course Notes. Translate them into Coq.

$$\forall x.Nat(x) \supset (\exists y.Nat(y) \wedge Eq(x,y))\ \mathit{true}$$
$$\forall x.\forall y.Eq(x,y) \supset Eq(y,x)\ \mathit{true}$$
$$\neg\exists x.Eq(x,\mathbf{0}) \wedge Eq(x,\mathbf{s}(\mathbf{0}))\ \mathit{true}$$

2 More properties of natural numbers (50 points)

Prove the following theorems in Coq.

$$\forall x.Nat(x) \supset Nat(\mathbf{s}(\mathbf{s}(x)))\ \mathit{true}$$
$$\forall x.\forall y.Lt(x,y) \supset \neg Eq(x,y)\ \mathit{true}$$
$$\neg\exists x.\exists y.Eq(x,y) \wedge Lt(x,y)\ \mathit{true}$$