Program Verification in Coq: Notes

1 Example: Plus

```
Module example.

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

Theorem plus_n_O : ∀ n:nat, n = n + 0.
Proof.
  intros n. induction n as [| n' IHn'].
  - reflexivity.
  - simpl. rewrite ← IHn'. reflexivity. Qed.
```

Example above from: Software Foundations Textbook. (https://softwarefoundations.cis.upenn.edu/)

```
End example.

Extraction Language Haskell.

Extraction example.
```

2 Example: Even Numbers

2.1 Example: Even numbers - 'How'

```
Fixpoint evenb (n : nat) : bool :=
match n with
| 0 ⇒ true
| 1 ⇒ false
| S (S n) ⇒ evenb n
end.
```

2.2 Example: Even numbers - 'What'

Definition *Even* (n : *nat*) : Prop :=
$\exists\ m,\ n = 2 \times m.$

2.3 Example: Even numbers - 'Correctness'

Lemma *evenb_correct* : $\forall\ n$, *evenb* n = *true* $\leftrightarrow$ *Even* n.
Abort. Proved below.

2.4 Example: Even numbers - 'Why'

Lemma *lemma_one* : $\forall\ n$, *evenb* (S n) = *negb* (*evenb* n).
Proof.
induction n as [|n'].
- Base case: $n = 0$
  reflexivity.
- Inductive case: for some n', $n = S\ n'$
  rewrite *IHn'*. simpl.
  destruct (*evenb* n') *eqn:SubCase*.
  + Sub-case (*evenb* n') = *true*
    reflexivity.
  + Sub-case (*evenb* n') = *false*
    reflexivity.

Qed.

2.5 Example: Even numbers - 'Why'

Require Import *Classical*.

Lemma *lemma_two* : $\forall\ n$, *Even* (S n) $\leftrightarrow \neg$ *Even* n.
Proof.
split.
- induction n as [|n'].
  + Base case: $n = 0$: unfold *Even*. intros. destruct *H*. destruct *x*. inversion *H*. simpl in *H*. rewrite $\leftarrow$ *plus_n_Sm* in *H*. discriminate *H*.
  + Inductive case: for some n', $n = S\ n'$: intros. unfold *not*. intros. apply *IHn'*. exact *H0*. unfold *Even*. destruct *H*. destruct *x*. simpl in *H*. discriminate *H*. $\exists$ *x*. simpl in *H*. rewrite $\leftarrow$ *plus_n_Sm* in *H*. inversion *H*. simpl. reflexivity.
- induction n. intros. unfold *Even* in *H*. *exfalso*. apply *H*. $\exists$ 0. reflexivity.
  intros. apply *NNPP*. unfold *not*. intros. apply *H*. apply *IHn*. unfold *not*. intros. apply *H0*. unfold *Even* in *H*. destruct *H1*. unfold *Even*. $\exists$ (S x). rewrite *H1*. simpl. rewrite $\leftarrow$ *plus_n_Sm*. reflexivity. Qed.

2.6 Example: Even numbers - 'Why'

```
Lemma evenb_correct : ∀ n, evenb n = true ↔ Even n.
Proof.
induction n.
 - firstorder. unfold Even. ∃ 0. reflexivity.
 - split.
    + intros. rewrite → lemma_one in H. rewrite lemma_two. unfold not. intros.
apply IHn in H0. rewrite H0 in H. simpl in H. discriminate H.
    + intros. apply lemma_two in H. rewrite lemma_one. destruct (evenb n) eqn:Case.
       × exfalso. apply H. apply IHn. reflexivity.
       × reflexivity.
Qed.
```

2.7 Example: Even numbers - 'What+How'

```
Program Definition evenb'
   (n : nat) : {b:bool | b = true ↔ Even n} :=
match n with
| 0 ⇒ true
| 1 ⇒ false
| S (S n) ⇒ evenb n
end.
```

2.8 Example: Even numbers - 'Why'

```
Next Obligation.
firstorder. ∃ 0. reflexivity. Qed.
Next Obligation.
firstorder. inversion H. destruct x. inversion H. simpl in H. rewrite ← plus_n_Sm
in H. inversion H. Qed.
Next Obligation.
induction n.
 - firstorder. unfold Even. ∃ 1. reflexivity.
 - split.
    + intros. rewrite → lemma_one in H. rewrite lemma_two. unfold not. intros.
apply IHn in H0. rewrite H0 in H. simpl in H. discriminate H.
    + intros. apply lemma_two in H. rewrite lemma_one. destruct (evenb n) eqn:Case.
       × exfalso. apply H. apply IHn. reflexivity.
       × reflexivity.
Qed.
```

3 Software Foundations

https://softwarefoundations.cis.upenn.edu/

4 Related

- VellVM http://www.cis.upenn.edu/~stevez/vellvm/
- CompCert http://compcert.inria.fr/
- Spark Ada (e.g. Air traffic management systems) http://www.adacore.com/sparkpro/
- Model Checking

5 DeepSpec

https://deepspec.org/