Semantics, WS 2011-2012: Solution for Assignment 3

Prof. Dr. Gert Smolka, Dr. Chad Brown

Exercise 3.1 Prove the following.

```
Goal forall X:Prop, ~X <-> ~~~X.
```

Solution to Exercise 3.1

```
Goal forall X:Prop, ~X <-> ~~~X.
intros X. split.
intros A B. exact (B A).
intros A B. apply A. intros C. exact (C B).
Qed.
```

Exercise 3.2 Prove the following.

a)

```
Goal False <-> forall Z:Prop, Z.
```

b)

```
Goal forall X:Prop, ~X <-> forall Z:Prop, X -> Z.
```

c)

```
Goal forall X:Type, forall x y:X, x = y <-> forall p:X -> Prop, p x -> p y.
```

d)

```
Goal forall X Y:Prop, X /\ Y <-> forall Z:Prop, (X -> Y -> Z) -> Z.
```

e)

```
Goal forall X Y:Prop, X \/ Y <-> forall Z:Prop, (X -> Z) -> (Y -> Z) -> Z.
```

Solution to Exercise 3.2

```
Goal False <-> forall Z:Prop, Z.
split.
intros A. contradiction A.
intros A. exact (A False).
Qed.

Goal forall X:Prop, ~X <-> forall Z:Prop, X -> Z.
intros X. split.
intros A Z B. contradiction (A B).
intros A. exact (A False).
Qed.

Goal forall X:Type, forall x y:X, x = y <-> forall p:X -> Prop, p x -> p y.
intros X x y. split.
intros A. rewrite A. intros p B. exact B.
intros A. apply (A (fun z => x = z)). reflexivity.
Qed.

Goal forall X Y:Prop, X /\ Y <-> forall Z:Prop, (X -> Y -> Z) -> Z.
intros X Y. split.
intros A Z B. destruct A as [x y]. exact (B x y).
intros A. apply (A (X /\ Y)). intros x y. split. exact x. exact y.
Qed.

Goal forall X Y:Prop, X \/ Y <-> forall Z:Prop, (X -> Z) -> (Y -> Z) -> Z.
intros X Y. split.
intros A Z B C. destruct A as [x|y]. exact (B x). exact (C y).
intros A. apply (A (X \/ Y)).
intros x. left. exact x.
intros y. right. exact y.
Qed.
```

Exercise 3.3 Prove the following.

a)

```
Goal forall X Y:Prop, ~(X \/ Y) <-> ~X /\ ~Y.
```

b)

```
Goal forall X Y Z:Prop, (X \/ (Y /\ Z)) <-> (X \/ Y) /\ (X \/ Z).
```

Solution to Exercise 3.3

```
Goal forall X Y:Prop, ~(X \/ Y) <-> ~X /\ ~Y.
intros X Y. split.
intros A. split.
intros x. apply A. left. exact x.
intros y. apply A. right. exact y.
intros A B. destruct A as [A1 A2]. destruct B as [x|y].
exact (A1 x).
exact (A2 y).
Qed.
```

```
Goal forall X Y Z:Prop, (X \/ (Y /\ Z)) <-> (X \/ Y) /\ (X \/ Z).
intros X Y Z. split.
intros A. destruct A as [x|B].
split. left. exact x. left. exact x.
destruct B as [y z]. split.
right. exact y.
right. exact z.
intros A. destruct A as [A1 A2].
destruct A1 as [x|y].
left. exact x.
destruct A2 as [x|z].
left. exact x.
right. split. exact y. exact z.
Qed.
```

Exercise 3.4 Prove the following. (This exercise may be tough.)

```
Goal (forall X:Prop, ~~X -> X) -> (forall X:Prop, X \/ ~X).
```

Solution to Exercise 3.4

```
Goal (forall X:Prop, ~~X -> X) -> (forall X:Prop, X \/ ~X).
intros A X. apply (A (X \/ ~X)).
intros B. apply B.
right. intros x.
apply B. left. exact x.
Qed.
```

Exercise 3.5 Prove the following.

a)

```
Goal forall p:nat -> Prop, forall x:nat, p 0 -> (forall x:nat, p x -> p (S x)) -> p x.
```

b)

```
Goal forall X:Type, forall p: list X -> Prop, forall xs:list X, p nil ->
 (forall x:X, forall xs: list X, p xs -> p (x :: xs)) -> p xs.
```

Solution to Exercise 3.5

```
Goal forall p:nat -> Prop, forall x:nat, p 0 -> (forall x:nat, p x -> p (S x)) -> p x.
intros p x A B. induction x.
exact A.
exact (B x IHx).
Qed.

Goal forall X:Type, forall p: list X -> Prop, forall xs:list X, p nil ->
 (forall x:X, forall xs: list X, p xs -> p (x :: xs)) -> p xs.
intros X p xs A B.
induction xs.
exact A.
exact (B a xs IHxs).
Qed.
```

Exercise 3.6 Extend the compiler correctness development with an operator for subtraction.

Solution to Exercise 3.6 All one needs to do is extend the definition of binop and evalBinop. The rest of the development remains the same.

```
Inductive binop : Type := Plus | Times | Minus.

Definition evalBinop (b : binop) : nat -> nat -> nat :=
  match b with
    | Plus => plus
    | Times => mult
    | Minus => minus
  end.
```

Exercise 3.7 (Challenge) Write a decompilation function that recovers an expression from the program it compiles to and prove the correctness of your function.

Exercise 3.8 Consider the following alternative definition of a compiler.

```
Fixpoint compile' (e : exp) : prog :=
  match e with
    | Const n => iConst n :: nil
    | Binop b e1 e2 => compile' e1 ++ compile' e2 ++ iBinop b :: nil
  end.
```

Consider the binary operators for addition (+), multiplication (*) and subtraction (-). What is the maximum set of these three operators for which this compiler is correct?

Solution to Exercise 3.8 The compiler is correct if addition and multiplication are included, but not if subtraction is included. If subtraction were included, then the expression corresponding to $1 - 0$ would be compiled into a program that runs on the empty stack and ends with the stack containing 0 instead of 1.