Constructive Logic (15-317), Fall 2009
Assignment 3: Natural Numbers and Classical Reasoning

William Lovas (wlovas@cs)

Out: Thursday, September 17, 2009
Due: Thursday, September 24, 2009 (before class)

This homework assignment aims to help draw connections between the material in this course and things you may be familiar with from prior mathematical experience. Natural numbers and induction are central to many areas of computer science, and these exercises will help you understand how they are defined and used from a formal perspective. Similarly, classical logic is pervasive in traditional mathematics, and these exercises will give you a chance to experience formal classical reasoning.

The Tutch portion of your work (Section 1) should be submitted electronically using the command

```
$ /afs/andrew/course/15/317/bin/submit -r hw03 <files...>
```

from any Andrew server. You may check the status of your submission by running the command

```
$ /afs/andrew/course/15/317/bin/status hw03
```

If you have trouble running either of these commands, email William.

The written portion of your work (Section 2) should be submitted at the beginning of class. If you are familiar with LaTeX, you are encouraged to use this document as a template for typesetting your solutions, but you may alternatively write your solutions *neatly* by hand.

1 Tutch Proofs (20 points)

Recall from recitation that Tutch's notation for the recursor $R(t, M_0, x.u.M_s(x,u))$ is a primitive recursion schema where the recursive copy of the function is bound locally:

```
rec t of f 0 => M0
       | f (s x) => Ms(x,f(x))
end
```

(Note the `end` keyword.) Using this, we defined addition as follows:

```
val plus : nat -> nat -> nat =
  fn x => fn y =>
    rec x of p 0 => y
           | p (s x') => s (p x')
    end;
```

Then we were able to prove properties of addition by induction over natural numbers, for instance, the property that if its second argument is zero, the result is the first argument:

```
proof plusN0N : !n:nat. (plus n 0 = n) =
begin
  [ n:nat;

    %%%% proceed by induction on n
    %% case n = 0.  TS: plus 0 0 = 0
    plus 0 0 = 0;

    %% case n = s n'.  TS: n':nat, plus n' 0 = n'
    %%                          |- plus (s n') 0 = (s n')
    [ n':nat, plus n' 0 = n';
      plus (s n') 0 = s n' ];

    %% apply induction to conclude:
    plus n 0 = n ];

  !n:nat. plus n 0 = n
end;
```

As usual, given the premises of an inference rule, Tutch allows you to derive the rule's conclusion; in this case, we're applying the rule of induction, $\mathsf{nat}E^{n',u}$, with the third premise being given by the hypothetical frame. While attempting to conclude an equality between natural numbers, Tutch reduces terms and employs the following rules of inference:

$$\frac{}{0 = 0}\ =_N I_0 \qquad \frac{n = n}{\mathsf{s}\, n = \mathsf{s}\, n}\ =_N I_{\mathsf{s}} \qquad \frac{0 = \mathsf{s}\, n}{J}\ =_N E_{0\mathsf{s}} \qquad \frac{\mathsf{s}\, n = 0}{J}\ =_N E_{\mathsf{s}0} \qquad \frac{\mathsf{s}\, m = \mathsf{s}\, n}{m = n}\ =_N E_{\mathsf{ss}}$$

As usual, you can run Tutch with the `-v` parameter to see which rules it's using to justify each line.

For more examples involving arithmetic, see Chapter 7 of the *Tutch User's Guide*.

Task 1 (3 pts)**.** Define multiplication in terms of addition.

```
val times : nat -> nat -> nat
```

Task 2 (3 pts)**.** Prove by induction that anything multiplied by zero is zero.

```
proof timesN00 : !n:nat. (times n 0 = 0)
```

To construct a proof term for an equality, you can access the rules above by using the constants `eq0`, `eqS (...)`, `eqE0S (...)`, `eqES0 (...)`, and `eqESS`.

Task 3 (2 pts)**.** Give a proof term for `timesN00` using a recursion schema.

```
term timesN00 : !n:nat. (times n 0 = 0)
```

Task 4 (12 pts)**.** Show that equality on natural numbers is symmetric by proving the following theorems in Tutch using induction.

```
proof symmEq0 : !n:nat. (0 = n => n = 0)
proof symmEqS : !m:nat. (!x:nat. m = x => x = m)
                     => !n:nat. (s m = n => n = s m)
proof symmEq : !m:nat. !n:nat. (m = n => n = m)
```

Pro Tip: You can use any proposition you've proven as a lemma simply by stating the proposition in your proof. In the above, your proof of `symmEq` can appeal to `symmEq0` and `symmEqS`.

On Andrew machines, you can check your progress against the requirements file `/afs/andrew/course/15/317/req/hw03.req` by running the command

```
$ /afs/andrew/course/15/317/bin/tutch -r hw03 <files...>
```

2 Classical Logic (20 points)

2.1 Principle of Bivalence

Since classically, every proposition is either true or false, classical logicians often make use of the following bivalence principle:

$$\dfrac{\dfrac{}{A\ true}\,u \qquad \dfrac{}{A\ false}\,k}{\begin{array}{cc}\vdots & \vdots \\ J & J\end{array}}{\scriptstyle Cases^{u,k}}$$

Task 5 (4 pts)**.** Show how this principle can be derived using the rules given in lecture. You may use any rules, including ones we showed derivable.

2.2 Natural Deduction for Falsehood

Although we effectively defined $A\ \mathit{false}$ as a judgement-level macro instead of defining it by giving rules to conclude it, many quite "natural" inference rules are derivable given our definition.

Task 6 (10 pts)**.** Using our definition of falsehood as $A\ \mathit{false} = \begin{array}{c} A\ \mathit{true} \\ \vdots \\ \# \end{array}$, show that the following rules are derivable:

$$\frac{A\ \mathit{false} \quad B\ \mathit{false}}{A \vee B\ \mathit{false}}\ \vee_f I \qquad \frac{A \vee B\ \mathit{false}}{A\ \mathit{false}}\ \vee_f E_L \qquad \frac{A \vee B\ \mathit{false}}{B\ \mathit{false}}\ \vee_f E_R$$

$$\frac{A\ \mathit{false}}{A \wedge B\ \mathit{false}}\ \wedge_f I_L \qquad \frac{B\ \mathit{false}}{A \wedge B\ \mathit{false}}\ \wedge_f I_R$$

Extra Credit Task 1 (3 extra credit pts)**.** Can you show the following rule derivable?

$$\frac{A \wedge B\ \mathit{false} \qquad \begin{array}{c} \overline{A\ \mathit{false}}\ k \\ \vdots \\ J \end{array} \qquad \begin{array}{c} \overline{B\ \mathit{false}}\ \ell \\ \vdots \\ J \end{array}}{J}\ \wedge_f E^{k,\ell}$$

If not, can you explain why?

2.3 The Prover's Dilemma

A humorous example of how classical logic sometimes runs counter to our intuitions is the Drinker's Paradox:

Drinker's Paradox: $\exists x{:}\tau.\, (D(x) \supset \forall y{:}\tau.\, D(y))$

In words, there's someone at the pub for whom if he drinks, then everybody drinks—someone's always the life of the party.

In our setting, we still can't prove this even using classical reasoning, since our domains of quantification may still be empty. However, there's a propositional analogue of the Drinker's Paradox that we might call the Prover's Dilemma:

Prover's Dilemma: $(A \supset A \wedge B) \vee (B \supset A \wedge B)$

We call it so because we can imagine someone wanting to prove both A and B, and the proposition above seemingly gives the prover a tantalizing time-saving shortcut: one of the results he requires implies both of them! The dilemma is to figure out which.

Task 7 (3 pts)**.** Prove a related theorem using classical reasoning: $(A \supset B) \vee (B \supset A)$.

Task 8 (3 pts)**.** Using the above theorem as a lemma (i.e. assuming a deduction of it), give a deduction of the Prover's Dilemma: $(A \supset A \wedge B) \vee (B \supset A \wedge B)$.