Constructive Logic (15-317), Fall 2009
Assignment 2: Quantifiers and Proof Terms

William Lovas (wlovas@cs)

Out: Thursday, September 10, 2009
Due: Thursday, September 17, 2009 (before class)

Previously, your work in this course has concerned only the propositional fragment of intuitionistic logic. In this homework, you will delve into the exciting world of first-order logic by solving problems involving the quantifiers $\forall$ and $\exists$. Furthermore, you'll have a chance to explore the Curry-Howard correspondence between logic and computation by writing proof terms representing your deductions. Finally, you'll continue to expand your understanding of harmony in definitions of logics by playing with a new quantifier.

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

```
$ /afs/andrew/course/15/317/bin/submit -r hw02 <files...>
```

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

```
$ /afs/andrew/course/15/317/bin/status hw02
```

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 and Proof Terms (25 points)

Tutch allows you to give an *annotated proof* for a proposition by declaring it with `annotated proof`. An annotated proof is just like a regular Tutch proof, but each line `A` is annotated with the term that justifies it `M : A`. Such an annotated proof is essentially a typing derivation for the proof term at its conclusion. Here's a simple example showing that conjunction is commutative:

```
annotated proof andComm : A & B => B & A =
begin
  [ u : A & B;
    snd u : B;
    fst u : A;
    (snd u, fst u) : B & A ];
  fn u => (snd u, fst u) : A & B => B & A
end;
```

Since a proof term determines the structure of the proof, Tutch also allows you to give just the proof term, by declaring it with `term`:

```
term andComm : A & B => B & A =
  fn u => (snd u, fst u);
```

For more examples, see Chapter 4 of the *Tutch User's Guide*. The proof terms are very similar to the ones given in lecture and are summarized in Section A.2.1 of the *Guide*.

Task 1 (6 pts)**.** Prove the theorem $(A \vee C) \wedge (B \supset C) \supset (A \supset B) \supset C$ using Tutch. Give a proof, an annotated proof, and a proof term.

```
proof implOr : (A | C) & (B => C)  =>  (A => B) => C
annotated proof implOr : (A | C) & (B => C)  =>  (A => B) => C
term implOr : (A | C) & (B => C)  =>  (A => B) => C
```

Task 2 (13 pts)**.** Prove the following theorems using Tutch, and provide proof terms.

```
proof curry : (A & B => C) => (A => B => C)
proof qcurry : ((?x:t. B(x)) => C) => (!x:t. B(x) => C)

term curry : (A & B => C) => (A => B => C)
term qcurry : ((?x:t. B(x)) => C) => (!x:t. B(x) => C)

proof compose : (!x:t. A(x) => B(x))
             => (!x:t. B(x) => C(x))
             => !x:t. A(x) => C(x)

term compose : (!x:t. A(x) => B(x))
            => (!x:t. B(x) => C(x))
            => !x:t. A(x) => C(x)
```

Task 3 (6 pts)**.** Prove the following theorems using Tutch.

```
proof distribAllAnd
      : (!x:t. A(x) & B(x)) <=> (!x:t. A(x)) & (!x:t. B(x))
proof distribExAnd1
      : (?x:t. A(x) & B(x)) => (?x:t. A(x)) & (?x:t. B(x))
```

On Andrew machines, you can check your progress against the requirements file `/afs/andrew/course/15/317/req/hw02.req` by running the command

```
$ /afs/andrew/course/15/317/bin/tutch -r hw02 <files...>
```

2 A Mixed-Up Quantifier (15 points)

In recitation, we saw that we could not prove $(\forall x{:}\tau.\, A(x)) \supset \exists x{:}\tau.\, A(x)$ *true*—our universal quantifier permits the domain of quantification to be empty! We were able to hack around this by proving a different proposition, but suppose we wanted to directly define a universal quantifier $\blacktriangledown x{:}\tau.\, A(x)$ that did *not* permit vacuous quantification. What would such a quantifier look like?

Its introduction rule $\blacktriangledown I^a$ is similar to $\forall I^a$: we must prove $A(a)$ for a new parameter $a : \tau$. However, to ensure the domain of quantification is non-empty, we must also supply an element $t : \tau$.

$$\dfrac{t : \tau \qquad \begin{array}{c}\overline{a : \tau} \\ \vdots \\ A(a)\ \mathit{true}\end{array}}{\blacktriangledown x{:}\tau.\, A(x)\ \mathit{true}}\ \blacktriangledown I^a$$

Then we can give two elimination rules, one with an existential character and one with a universal character.

$$\dfrac{\blacktriangledown x{:}\tau.\, A(x)\ \mathit{true} \qquad \begin{array}{c}\overline{a : \tau} \\ \vdots \\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \blacktriangledown E^a_\exists \qquad\qquad \dfrac{\blacktriangledown x{:}\tau.\, A(x)\ \mathit{true} \qquad t : \tau}{A(t)\ \mathit{true}}\ \blacktriangledown E_\forall$$

As usual, rules that introduce a parameter restrict its scope to the premise in which it is introduced. In particular, in the elimination rule $\blacktriangledown E^a_\exists$, the parameter a may not appear in the conclusion of the rule, C *true*.

Task 4 (2 pts)**.** Show that this captures our intuition about what non-vacuous universal quantification should mean by giving a deduction of $(\blacktriangledown x{:}\tau.\, A(x)) \supset \exists x{:}\tau.\, A(x)$ *true*.

Task 5 (4 pts)**.** Are these elimination rules locally sound? If so, give a local reduction for each elimination rule; if one does not exist, explain why.

Task 6 (4 pts)**.** Are these elimination rules locally complete? If so, give a local expansion for for an arbitrary deduction of $\blacktriangledown x{:}\tau.\, A(x)$; if one does not exist, explain why.

Task 7 (5 pts)**.** Make up a proof term for each of the rules, and express the local reductions and local expansions you found above using proof terms.