Constructive Logic (15-317), Fall 2015
Assignment 3: Pirates and Quantifiers

Contact: Michael Coblenz (mcoblenz@cs.cmu.edu)
Recitation C

Due Tuesday, September 29, 2015

Ahoy, matey! In case you are unaware, last Saturday (Sept. 19) was International Talk Like a Pirate Day. We will be observing this important holiday by solving problems relating to pirates. You might be surprised that pirates simply cannot exist without quantifiers; we will be exploring this important relationship in this assignment.

The Tutch portion of your work should be submitted electronically using the command

```
$ /afs/andrew/course/15/317/bin/submit -r hw3 <files...>
```

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

```
$ /afs/andrew/course/15/317/bin/status hw3
```

If you have trouble running either of these commands, email Anna, Michael, or Vincent.

The written portion of your work 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 Ship Construction (5 points)

"Set sail!" ordered Cap'n Platzer.

"No way," replied Mate Coblenz.

"What's wrong?" demanded Platzer.

"You're a new cap'n, remember? We don't have a ship yet."

"Okay, then, build me a ship forthwith!"

Mate Coblenz proposed the following rules for ship construction.

$$\frac{h\ \textbf{hull} \quad m\ \textbf{mast}}{s\ \textbf{ship}}\ ship - I$$

$$\frac{s\ \textbf{ship}}{h\ \textbf{hull}}\ ship - E_{hull}$$

$$\frac{s\ \textbf{ship}}{m\ \textbf{mast}}\ ship - E_{mast}$$

"But there's a problem. Where are we going to get a hull and a mast?" Coblenz asked.

"No problem; my friend Merlin can conjure those up out of thin air," replied Platzer.

$$\frac{}{m\ \textbf{mast}}\ mast - I$$

$$\frac{}{h\ \textbf{hull}}\ hull - I$$

Coblenz, working from the meager 15-317 budget, hired Long John to assemble the ship. Here is what the lad proved:

$$\dfrac{\dfrac{\dfrac{}{m_1\ \textbf{mast}}\ mast - I \quad \dfrac{}{h_1\ \textbf{hull}}\ hull - I}{s_1\ \textbf{ship}}\ ship - I \quad \dfrac{\dfrac{}{m_2\ \textbf{mast}}\ mast - I \quad \dfrac{}{h_2\ \textbf{hull}}\ hull - I}{s_2\ \textbf{ship}}\ ship - I}{h_1\ \textbf{hull}}\ ship - E_{hull} \qquad \dfrac{s_2\ \textbf{ship}}{m_2\ \textbf{mast}}\ ship - E_{mast}}{s_3\ \textbf{ship}}\ ship - I$$

Coblenz reviewed the construction plans. He immediately made Long John walk the plank. We won't be hearing from *that* old scumbucket again.

Task 1. Explain informally why this was Long John's last time building a ship. What was wrong with the construction plans?

Task 2. Rewrite Coblenz's rules so that even a landlubber using your rules will never make the same mistake Long John did. You should achieve the following objective: once parts have been assembled into a larger construction, then even if that construction is torn apart (via elimination rules), those parts cannot be re-used for assembly again. After all, if the parts were available originally, we shouldn't integrate them into a module just to raid that module for parts. Hint: this is *exactly* the same as the idea of verification vs. license to use, and you should use the same notation in your rules.

One way to think about this is that anything you are given is stronger than anything you compose yourself, but decomposition (i.e. elimination) preserves strength. A mast that you got via an introduction rule is somehow stronger than

a ship you built from pieces, and you can't disassemble something weak to get something strong.

2 Primitive Recursive Pirates (15 points)

Cap'n Platzer and his scurvy crew have captured a ship! Each captured pirate has the same amount g of gold. Cap'n Platzer wants to know how much gold he has looted, given p pirates.

Unfortunately, Cap'n Platzer just got into the pirate business, and he only has primitive recursion, natural numbers, and Tutch. Help him write some basic arithmetic functions so he can compute his loot. We don't really know what he's going to use factorials for, but he's eyeing the plank, so you'd better get started.

```
val plus : nat -> nat -> nat

val mult : nat -> nat -> nat

val fact : nat -> nat

val exp : nat -> nat -> nat

val gte : nat -> nat -> bool

val divide : nat -> nat -> (nat * nat)
```

Once you've implemented a function, you may use it in your solutions to others.

2.1 Factorial

$$fact \in \mathbf{nat} \to \mathbf{nat}$$

Define *fact* using primitive recursion. Hint: use *mult* in your solution.

2.2 exp

$$exp \in \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}$$

Define *exp* using primitive recursion so that $exp\ x\ y = x^y$. Hint: use *mult* in your solution.

2.3 gte

$$\begin{aligned} gte &\in \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat} \\ gte\ 0\ 0 &= true \\ gte\ 0\ s(y') &= false \\ gte\ s(x')\ 0 &= true \\ gte\ s(x')\ (s(y')) &= gte\ x'\ y' \end{aligned}$$

Define *gte* using primitive recursion. If you use *mult* in your solution, you will be required to swab the deck every class for the rest of the semester.

2.4 Division

Consider a division function *divide* such that *divide* y x returns a pair of the quotient of x divided by y and the remainder of x divided by y. (Note that y is the first argument to *divide* and x the second argument.) If y is 0, it returns $\langle x, 0\rangle$:

$$divide \in \mathbf{nat} \to \mathbf{nat} \to (\mathbf{nat} \times \mathbf{nat})$$

Some examples:

$$\begin{aligned} divide\ 0\ t &\Rightarrow \cdots \Rightarrow \langle t, 0\rangle \\ divide\ s(0)\ t &\Rightarrow \cdots \Rightarrow \langle t, 0\rangle \\ divide\ s(s(0))\ 0 &\Rightarrow \cdots \Rightarrow \langle 0, 0\rangle \\ divide\ s(s(0))\ s(0) &\Rightarrow \cdots \Rightarrow \langle 0, s(0)\rangle \\ divide\ s(s(0))\ s(s(0)) &\Rightarrow \cdots \Rightarrow \langle s(0), 0\rangle \\ divide\ s(s(0))\ s(s(s(0))) &\Rightarrow \cdots \Rightarrow \langle s(0), s(0)\rangle \end{aligned}$$

Define *divide* using primitive recursion. Your solution may use *gte* and all constructs associated with pairs and booleans (see the Tutch manual).

3 Quantifiers

In the problems below, you can assume τ = **pirate**, $P(x)$ = **x has contracted scurvy**, and $Q(x)$ = **x is cleaning the cannons**. But if you ever use those assumptions, you're doing the problem wrong. The interpretation of R is left to your imagination. Your grader could probably use some entertainment, so feel free to write down your interpretation of R.

3.1 Tutch programming (15 points)

For each proposition below, prove its truth using Tutch. If its truth is unprovable, explain why (in your written solution). Eliminating existential quantifiers in Tutch results in a witness. As an example, note the line `c:t,~A(c)` in this proof:

```
proof ExNotImpNotAll : (?x:t. ~A(x)) => ~!x:t. A(x) = begin
[ ?x:t. ~A(x);
    [ !x:t. A(x);
        [ c: t, ~A(c);
            A(c);
            F ]; F ];
        ~!x:t. A(x) ];
    (?x:t. ~A(x)) => ~!x:t. A(x);
end;
```

You should read section 6 of http://www.cs.cmu.edu/ fp/courses/15317-f00/software/tutch.pdf so you understand how to use quantifiers in Tutch. It's only two pages long.

Task 3.

$$\forall x : \tau.\exists y : \tau.P(x) \supset P(y)$$

Task 4.

$$(\forall x : \tau.P(x) \supset Q(x)) \wedge (\forall x : \tau.Q(x) \supset R(x)) \supset (\forall x : \tau.P(x) \supset R(x))$$

Task 5.

$$(\forall x : \tau.P \vee Q(x)) \supset (P \vee \forall x : \tau.Q(x))$$

Task 6.

$$(P \supset \exists x : \tau.Q(x)) \supset (\exists x : \tau.P \supset Q(x))$$

3.2 Natural deduction (5 points)

For each proposition below, prove its truth using natural deduction. If its truth is unprovable, explain why (in your written solution). You should annotate every part of a derivation with the name of the inference rule. You should also annotate each hypothesis with some variable. Failure to do so may result in an ambiguous proof, in which case full credit will not be given.

Task 7.

$$(P \supset \forall x : \tau.Q(x)) \supset (\forall x : \tau.P \supset Q(x))$$

Task 8.

$$(\forall x : \tau.P \supset Q(x)) \supset (P \supset \forall x : \tau.Q(x))$$