Constructive Logic (15-317), Fall 2016
Assignment 3: Sequents and Quantifiers

Contact: Evan Cavallo (ecavallo@cs.cmu.edu)

Due Tuesday, September 27, 2016

This assignment is due at the beginning of class on the above date and must be submitted electronically via autolab. Submit your homework as a **tar** archive containing two files: **hw3.pdf** (your written solutions) and **hw3.tut** (your Tutch solutions).

1 Sequents for Natural Deduction

In this section we will look at another style of notation for natural deduction, based on *sequents*. For a list of judgments $\Gamma = \mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$ and single judgment $\mathcal{J}$, the sequent $\Gamma \vdash \mathcal{J}$, read "$\Gamma$ entails $\mathcal{J}$," is another way of writing the hypothetical judgment

$$\begin{array}{c} \mathcal{J}_1 \quad \cdots \quad \mathcal{J}_n \\ \vdots \\ \mathcal{J} \end{array}$$

We can rewrite the rules of natural deduction to use this notation. For example, here are the rules for implication:

$$\frac{\Gamma, A \ \mathit{true} \vdash B \ \mathit{true}}{\Gamma \vdash A \supset B \ \mathit{true}} \supset I \qquad \frac{\Gamma \vdash A \supset B \ \mathit{true} \quad \Gamma \vdash A \ \mathit{true}}{\Gamma \vdash B \ \mathit{true}} \supset E$$

We no longer have a floating A *true* assumption in the introduction rule, but instead simply add it to the context of hypotheses. This notation is often clearer and easier to work with for metatheory, since the hypotheses in scope are made explicit. On the other hand, it can get quite cluttered when the context is large.

Note: this is not *the sequent calculus*, which is an entirely different logical system. It's just an alternative notation for the usual natural deduction rules. This exercise is meant to clarify the different between natural deduction and sequent calculus.

Task 1 (6 points)**.** Rewrite the rules for the connectives $\wedge$ and $\vee$ using this notation.

Task 2 (3 points)**.** State the substitution principle for judgments in this notation.

2 Terms and Quantifiers

2.1 Tutch, Quantified

Task 3 (8 points)**.** For each proposition below, prove its truth using Tutch. 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 can find documentation for quantifiers in Tutch at `https://www.andrew.cmu.edu/course/15-317/software/tutch/doc/html/tutch_6.html`.

1. `proof apply : (!x:t.A(x) => B(x)) => (!x:t.A(x)) => (!x:t.B(x));`
2. `proof instance : (!x:t.A(x)) & (?y:t.B(y)) => ?z:t.A(z);`
3. `proof swap : (?y:t.!x:t.A(x,y)) => (!x:t.?y:t.A(x,y));`
4. `proof frobenius : (R & ?x:t.Q(x)) <=> ?x:t.(R & Q(x));`

2.2 Lawvere Equality

Consider the following rules for a connective $\sim$. The judgment $t \sim_\tau s \ \mathit{true}$ is intended to express that t and s are equal terms of type τ.

$$\frac{t : \tau}{t \sim_\tau t \ \mathit{true}} \sim I \qquad \frac{t \sim_\tau s \ \mathit{true}}{t : \tau} \sim E_{\text{type}} \qquad \frac{t \sim_\tau s \ \mathit{true} \quad \begin{array}{c}\overline{a : \tau} \\ \vdots \\ P(a,a) \ \mathit{true}\end{array}}{P(t,s) \ \mathit{true}} \sim E^a_{\text{prop}}$$

Task 4 (3 points)**.** Show that the rule

$$\frac{s \sim_\tau t \; \mathit{true} \quad Q(s) \; \mathit{true}}{Q(t) \; \mathit{true}} \; \text{Leibniz}$$

is derivable in this system.

Task 5 (4 points)**.** Give natural deduction proofs of the following judgments:

1. $\forall x{:}\tau.\forall y{:}\tau.(x \sim_\tau y) \supset (y \sim_\tau x) \; \mathit{true}$
2. $(\forall x{:}\tau.\forall y{:}\tau.x \sim_\tau y) \supset (\exists z{:}\tau.P(z)) \supset (\forall z{:}\tau.P(z)) \; \mathit{true}$

You may use the derived rule Leibniz in your proofs.

Task 6 (3 points)**.** Show that the connective $\sim$ is locally sound and complete by giving all relevant local reductions and local expansions.

3 Data Types

Recall the introduction rules, recursion elimination rule, and induction elimination rule for the natural number type:

$$\frac{}{\mathtt{z} : \mathtt{nat}} \; \mathtt{nat}I_{\mathtt{z}} \qquad \frac{n : \mathtt{nat}}{\mathtt{s}\; n : \mathtt{nat}} \; \mathtt{nat}I_{\mathtt{s}} \qquad \frac{\begin{array}{ccc} n : \mathtt{nat} & t_0 : \tau & \begin{array}{c}\overline{x : \mathtt{nat}} \; \overline{r : \tau} \\ \vdots \\ t_s : \tau\end{array}\end{array}}{R(n, t_0, x.r.t_s) : \tau} \; \mathtt{nat}E^{x,r}$$

$$\frac{\begin{array}{ccc} n : \mathtt{nat} & C(\mathtt{z}) \; \mathit{true} & \begin{array}{c}\overline{x : \mathtt{nat}} \; \overline{C(x) \; \mathit{true}}^{\,u} \\ \vdots \\ C(\mathtt{s}\; x)\end{array}\end{array}}{C(n) \; \mathit{true}} \; \mathtt{nat}E^{x,u}$$

Task 7 (3 points)**.** Prove the following judgment in natural deduction.

$$\forall x{:}\mathtt{nat}.\neg(x \sim_{\mathtt{nat}} \mathtt{z}) \supset \exists y{:}\mathtt{nat}.(x \sim_{\mathtt{nat}} \mathtt{s}\; y) \; \mathit{true}$$

Task 8 (6 points)**.** Define the following primitive recursive functions in Tutch. You can find documentation for programming with types at `https://www.andrew.cmu.edu/course/15-317/software/tutch/doc/html/tutch_5.html`.

```
val plus : nat -> nat -> nat
val mult : nat -> nat -> nat
val factorial : nat -> nat
```

Task 9 (4 points)**.** The natural numbers are a simple example of an inductively defined type. Other inductively defined types include lists and trees. Just like natural numbers, lists and trees are constructed by successively applying constructors. Consider the following introduction rules for a type $\texttt{tree}_\kappa$ of trees with leaves of type κ.

$$\frac{k : \kappa}{\texttt{leaf}(k) : \texttt{tree}_\kappa}\ \texttt{tree}I_{\texttt{leaf}} \qquad \frac{l : \texttt{tree}_\kappa \quad r : \texttt{tree}_\kappa}{\texttt{node}(l; r) : \texttt{tree}_\kappa}\ \texttt{tree}I_{\texttt{node}}$$

A recursion rule gives us a way to define functions out of an inductive type, while an elimination rule allows us to prove theorems about it. Define a recursion rule and an elimination rule for the type $\texttt{tree}_\kappa$.