Constructive Logic (15-317), Spring 2020
Assignment 2: Come to terms with proofs

Due Tuesday, Jan 28, 2020

You will be using Tutch for the majority of this assignment, but there is also a written portion. This assignment is due at 11:59 on the above date and must be submitted electronically via Autolab and Gradescope. Submit the written portion of your homework to gradescope as a **pdf** and the code portion as a **tar** archive containing **hw2_task1**{**a, b**}**.tut**, **hw2_task2a, b, c, d**, and **hw2_task3**{**a, b, c, d**}**.tut**.

1 Tutch Proofs

A Tutch is programming language for writing and checking simple proofs. A proof in Tutch is made much like a proof in natural deduction. Each introduction and elimination rule has a counterpart in Tutch. To use most rules, all that is required is for you to have shown all of the prerequisites for that rule in scope and then you can just claim any consequent of the rule. Making assumptions is done like below:

```
proof A => A =
begin
[A;          % assume u
 A           % use u
];
A => A;      % Implication introduction with u
end;
```

To show an $\vee$ and use it, you must take each side of the or and derive from each of them the same conclusion. Then you may conclude whatever you derived in both branches. The only time you'll use square brackets without an implication immediately following it is in an $\vee$ elimination which will look like below:

```
proof (A | B) => T =
begin
[A | B;              % assume u
  [A;                % assume w
    T                % verum intro
  ];
  [B;                % assume w'
    T                % verum intro
  ];
  T                  % or elimination
];
A | B => T;          % Implication introduction
end;
```

For more examples, see Chapter 4 of the Tutch User's Guide or the recitation code.

Task 1 (4 points)**.** Begin by giving a proof of the following theorems using Tutch.

a. `proof reuse : ((A => B) & (A => C)) => A => (B & C);`

c. `proof toptobottom : (A => T) & (F => A);`

Tutch allows you to annotate your proof with proof terms by declaring it with `annotated proof`. An annotated proof is just like a regular Tutch proof, but each line, normally `A`, is annotated with the term that justifies it, making it `M : A`.

```
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;
```

The proof terms are very similar to the ones given in lecture and are summarized in Section A.2.1 of the Guide.

Task 2 (8 points)**.** Give annotated proofs for the following theorems using Tutch.

a. `annotated proof reuse : ((A => B) & (A => C)) => A => (B & C);`

b. `annotated proof andmap : (A & B) => ((A => C) & (B => D)) => (C & D);`

c. `annotated proof M : ((A | B) => C) => (A => C) & (B => C);`

d. `annotated proof dual : ~(A | B) <=> (~A & ~B);`

It is also possible to give a term without the propositions. To give a program as a proof in Tutch, declare it with `term` rather than `proof`:

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

The syntax for terms is, coincidentally, precisely the syntax that we have used for annotating proof terms in lecture.

Task 3 (8 points)**.** Give proof terms for the following theorems using Tutch.

a. `term toptobottom : (A => T) & (F => A);`

b. `term ormap : (A | B) => ((A => C) & (B => D)) => (C | D);`

c. `term M : ((A | B) => C) => (A => C) & (B => C);`

d. `term classy : (A | ~A) => (~~A => A);`

On a machine with Tutch installed, you can check your progress against a particular requirements file by running

```
$ tutch -r ./hw2_1a.req hw2_1a.tut
```

Substituting `1a` for the appropriate task number and letter to denote the problem.

Task 4 (4 points)**.** Give a brief description of your solution to problem 3c in the perspective of a function. What does it do?

2 I thunk therefore I am

Task 5 (8 points)**.** Consider a unary connective $*$ defined by the following rules:

$$\dfrac{\dfrac{}{\top\ \mathit{true}}\ u \\ \vdots \\ \mathit{A\ true}}{*\mathit{A\ true}}\ *I^u \qquad \dfrac{*\mathit{A\ true} \quad \top\ \mathit{true}}{\mathit{A\ true}}\ *E$$

1. Under what condition relative to $A\ \mathit{true}$ is $*A\ \mathit{true}$ derivable?
2. Using $\mathbf{thunk}(\mathbf{u}.\mathbf{M})$ as the constructor, give (the) appropriate intro rule(s) for $\mathbf{thunk}(u.M) : *A$.
3. Using $\mathbf{thunk}(u.M)N$ as the destructor, give (the) appropriate elim rule(s) for $\mathbf{thunk}(u.M)N : A$.
4. Can $*$ have a reduction rule[1]? If not, explain why, otherwise write out a reduction rule for $*$.

3 Reduce, Reuse, Recycle

Recall from lecture that we can simulate a computation by applying reduction rules to a proof term until no more reduction rules may be applied.

Task 6 (8 points)**.** Take M as the proof term in your solution to Task3c. Consider the term

$$\mathbf{fn}\, w : C \wedge C \Rightarrow M(\mathbf{fn}\, v : A \vee B \Rightarrow \mathbf{case}\, v\ \mathbf{of}\ \mathbf{inl}\, l \Rightarrow \mathbf{fst}(w) \mid \mathbf{inr}\, r \Rightarrow \mathbf{snd}(w))$$

1. Write a series of reduction steps from this term until no more steps may be applied[2]. You can rename the variables or add primes to avoid capture.
2. What proposition is this term a proof of?

Submitting your solutions

Please generate a tarball containing your solution files by running

```
$ tar cf hw2.tar hw2_task1a.tut hw2_task1b.tut hw2_task2a.tut \
hw2_task2b hw2_task2c.tut hw2_task2d.tut \
hw2_task3a.tut hw2_task3b.tut hw2_task3c.tut hw2_task3d.tut
```

and submit the resulting `hw2.tar` file to Autolab. Then submit your pdf with answers to task 3 and 4 to gradescope.

[1] remember that a reduction rule takes a destructor for connective and reduces it to a simpler term.
[2] remember to apply reuction rules to subterms