15-317: Constructive Logic, Fall 2020
Assignment 2: Tutch, Constructivity & Harmony!

Instructor: André Platzer
TAs: Zhibo Chen, Avery Cowan, Julia Gu, Akshina Gupta, Ethan Rosenthal

Due: Thursday, February 18, 2021, 11:59 pm

The assignments in this course must be submitted electronically through Gradescope. Written homework PDFs and coding SML files will both go to Gradescope. For this homework, submit two files:

- `hw2.pdf` (your written solutions)
- `hw2.sml` (your coding solutions)

1 More Proofs? Deduce that!

Task 1 (12 points)**.** Prove the following theorems using natural deduction logic in SML. Remember that $\neg A$ is syntatic sugar for $A \supset F$. You can look at `support/nd_examples.sml` for reference natural deduction proof trees.

a. `prove absurdity:` $A \wedge \neg A \supset B$

b. `prove sCombinator:` $(A \supset B) \supset (A \supset B \supset C) \supset (A \supset C)$

c. `prove deMorgin:` $\neg(A \vee B) \supset \neg A \wedge \neg B$

d. `prove deMorgout:` $\neg A \wedge \neg B \supset \neg(A \vee B)$

You can compile your code by running the following command in your command line. Keep in mind that this will only tell you if your proof is well forrmatted and will not check for correctness:

```
$ smlnj -m sources.cm
```

You can pretty print your natural deduction proofs by running the following command in your repl. This should make it easier for you to check your work:

```
>> Out.print_nd Homework2.{proof_name_here}
```

2 Trees are Programs

Task 2 (12 points)**.** Prove the following theorems using the proof-as-program logic in SML. You can look at `support/pap_examples.sml` for reference proofs as programs proof trees.

a. `prove deMorgagain:` $\neg A \wedge \neg B \supset \neg(A \vee B)$

b. `prove toptobottom:` $(A \supset \top) \wedge (\bot \supset A)$

c. `prove reuse:` $((A \supset B) \wedge (A \supset C)) \supset (A \supset B \wedge C)$

d. `prove ormap:` $((A \vee B) \supset C) \supset (A \supset C) \wedge (B \supset C)$

You can compile your code the same way as for natural deduction. You can pretty print your proof-as-program proof trees by running the following command in your repl:

```
>> Out.print_pap Homework2.{proof_name_here}
```

3 Functions are Proofs

Task 3 (8 points)**.** For this task, you will be directly writing the code that inhabits the corresponding type for a proposition. For each proposition, either submit `SOME(v)` where `v` is a value of that type or leave it as `NONE` if the proposition is unprovable[1]. Make sure that you are not using the proof tree infrastructure.

We provide you with the `void` type and `abort` function to deal with falsehood. Similarly, you have access to the built-in structure `Either` in order to deal with $\vee$.

a. `prove curry:` $(A \wedge B \supset C) \supset A \supset B \supset C$

b. `prove abba:` $((A \supset B) \supset B) \supset A$

c. `prove contrapositive:` $(A \supset B) \supset (\neg B \supset \neg A)$

d. `prove exclusion:` $((A \vee B) \wedge \neg A) \supset B$

[1] Proving the totality of functions using exceptions or recursion is nontrivial so do not use exceptions or recursion for this task

4 I thunk therefore I am

Task 4 (8 points)**.** Consider a unary connective $\circ$ defined by the following rules:

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

1. Under what condition relative to A *true* is $\circ A$ *true* derivable?
2. Using $\mathbf{thunk}(\mathbf{u}.\mathbf{M})$ as the constructor, give (the) appropriate intro rule(s) for $\mathbf{thunk}(u.M) : \circ A$.
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 $\circ$ have a reduction rule[2]? If not, explain why, otherwise write out a reduction rule for $\circ$.
5. Why might a programming language or programmer want to use thunks in code?[3]

[2]Remember that a reduction rule takes a destructor for a connective and reduces it to a simpler term. The destructor should be unique from those of existing connectives

[3]Any reasonable guess is fine