Hype for Types
HW 01
Due 21 January 2020

This homework will require you to translate between English descriptions of typing rules and formal inference rules. For example, we have the following typing rule for product types:

> If `x` has type σ and `y` has type τ, then `(x,y)` has type $\sigma * \tau$

which is represented by the formal inference rule

$$\frac{\Gamma \vdash \texttt{x} : \sigma \quad \Gamma \vdash \texttt{y} : \tau}{\Gamma \vdash \texttt{(x,y)} : \sigma * \tau} \quad .$$

So, we'll start by giving you either the English description or the formal rule, and asking you to produce the other. Then, we'll move on to giving multiple rules governing a more complex type.

If you're trying to LaTeX this, an easy way to typeset this is by using the `proof` package (i.e. `usepackage{proof}`). Then, you can produce the above inference rule with

```
\infer[]
    {\Gamma\vdash\texttt{(x,y)}:\sigma * \tau}
    {\Gamma\vdash\texttt{x}:\sigma & \Gamma\vdash\texttt{y}:\tau}
```

Question 1 (Required)
What does the following rule say in English?

$$\frac{\Gamma \vdash \texttt{f} : \sigma \to \tau \quad \Gamma \vdash \texttt{t} : \sigma}{\Gamma \vdash \texttt{(f t)} : \tau}$$

Question 2 (Required)
Write the following rule as a formal inference rule.

> If, in context Γ, `b` is of type `bool`, and both `e1` and `e2` are of type τ (in Γ), then the expression `if b then e1 else e2` is of type τ

Question 3 (Required)

For any type τ, we define the type τ `option` by the following rules:

- In any context, `NONE` : τ `option`
- If `x` is of type τ in context Γ, `SOME(x)` is of type τ `option` in context Γ
- If
  - `e1` : σ in context Γ
  - `e2` : σ in context $\Gamma, \texttt{x} : \tau$
  - `opt` : τ `option` in context Γ

  then

  (`case opt of NONE => e1 | (SOME x) => e2`)

  has type σ in context Γ

Write these rules as formal inference rules

Question 4 (Optional)

Write inference rules for the type `bool` of booleans. You should include the values of type bool, rule(s) for getting booleans from other types (e.g. if `x` and `y` are of the same type, then `x=y` is a boolean[1]), and rules for using booleans (e.g. what's an appropriate rule for `if...then...else` expressions?).

[1] Don't worry about equality types

Question 5 (Optional)

Come up with an inference rule governing how to typecheck `let...in...end` expressions with only a single declaration between the `let` and the `in`, e.g.

```
let
  val x = e
in
  y
end
```