Monad-Based Programming WS 2021

Assignment 3

Deadline for solutions: 10.12.2021

Exercise 1 Programming a Proof Assistant (20 Points)

The formal system of *natural deduction* operates with the judgements of the form

$$\Sigma; \Gamma \vdash \pi : A$$

which states that a (first-order) proposition A with free variables in Σ is derivable from the list of propositions Γ, and π is the *proof term*, encoding this derivation. Derivability of such judgment is interpreted as the fact that A provably follows from Γ.

Formally, for a fixed set of functional symbols F and a fixed set of predicate symbols P, we introduce *terms in context* with the rules

$$\frac{x \ \text{ in } \ \Sigma}{\Sigma \vdash x \ \ \mathit{term}} \ (\text{var-F}) \qquad \frac{\Sigma \vdash t_1 \ \ \mathit{term} \ \ \ldots \ \ \Sigma \vdash t_n \ \ \mathit{term}}{\Sigma \vdash f(t_1, \ldots, t_n) \ \ \mathit{term}} \ (\text{app-F})$$

Example: With $F = \{+\}$, $(x, y \vdash x + y \ \ \mathit{term})$ states that $x + y$ is a term over variables x and y.

Propositions in context (over F, P) are introduced with the rules:

$$\frac{\phi \in P \quad \Sigma \vdash t_1 \ \ \mathit{term} \ \ \ldots \ \ \Sigma \vdash t_n \ \ \mathit{term}}{\Sigma \vdash \phi(t_1, \ldots, t_n) \ \ \mathit{prop}} \ (\text{pred-F})$$

$$\frac{\Sigma \vdash B \ \ \mathit{prop} \ \ \Sigma \vdash B \ \ \mathit{prop}}{\Sigma \vdash A \wedge B \ \ \mathit{prop}} \ (\wedge F) \qquad \frac{}{\Sigma \vdash \top \ \ \mathit{prop}} \ (\top F)$$

$$\frac{\Sigma \vdash B \ \ \mathit{prop} \ \ \Sigma \vdash B \ \ \mathit{prop}}{\Sigma \vdash A \vee B \ \ \mathit{prop}} \ (\vee F) \qquad \frac{}{\Sigma \vdash \bot \ \ \mathit{prop}} \ (\bot F)$$

$$\frac{\Sigma \vdash B \ \ \mathit{prop} \ \ \Sigma \vdash B \ \ \mathit{prop}}{\Sigma \vdash A \Rightarrow B \ \ \mathit{prop}} \ (\Rightarrow F)$$

$$\frac{\Sigma, x \vdash A \ \ \mathit{prop}}{\Sigma \vdash \forall x.\, A \ \ \mathit{prop}} \ (\forall F) \qquad \frac{\Sigma, x \vdash A \ \ \mathit{prop}}{\Sigma \vdash \exists x.\, A \ \ \mathit{prop}} \ (\exists F)$$

Example: With $P = \{P\}$, $(\vdash \exists x.\, (P(x) \rightarrow \forall y.\ P(y)) \ \ \mathit{prop})$ is a proposition without free variables.

Derivable judgements are obtained with the following rules:

$$\frac{u\colon A \ \text{ in } \ \Gamma}{\Sigma;\Gamma \vdash u\colon A}\ (\text{hyp})$$

$$\frac{\Sigma;\Gamma \vdash \pi_1\colon A \qquad \Sigma;\Gamma \vdash \pi_2\colon B}{\Sigma;\Gamma \vdash (\pi_1,\pi_2)\colon A \wedge B}\ (\wedge I) \qquad \frac{}{\Sigma;\Gamma \vdash ()\colon \top}\ (\top I)$$

$$\frac{\Sigma;\Gamma \vdash \pi_1\colon A}{\Sigma;\Gamma \vdash \text{inl}\ \pi_1\colon A \vee B}\ (\vee I_1) \qquad \frac{\Sigma;\Gamma \vdash \pi_2\colon B}{\Sigma;\Gamma \vdash \text{inr}\ \pi_2\colon A \vee B}\ (\vee I_2)$$

$$\frac{\Sigma;\Gamma, x\colon A \vdash \pi\colon B}{\Sigma;\Gamma \vdash \lambda x.\,\pi\colon A \Rightarrow B}\ (\Rightarrow I)$$

$$\frac{\Sigma, x;\Gamma \vdash \pi\colon A}{\Sigma;\Gamma \vdash \Lambda x.\,\pi\colon \forall x.\,A}\ (\forall I) \qquad \frac{\Sigma \vdash t \ \ \text{term} \qquad \Sigma;\Gamma \vdash \pi[t/x]\colon A[t/x]}{\Sigma;\Gamma \vdash (t,\pi)\colon \exists x.\,A}\ (\exists I)$$

$$\frac{\Sigma;\Gamma \vdash \pi\colon A \wedge B}{\Sigma;\Gamma \vdash \text{fst}\ \pi\colon A}\ (\wedge E_1) \qquad \frac{\Sigma;\Gamma \vdash \pi\colon A \wedge B}{\Sigma;\Gamma \vdash \text{snd}\ \pi\colon B}\ (\wedge E_2)$$

$$\frac{\Sigma;\Gamma \vdash \pi\colon A \vee B \qquad \Sigma;\Gamma, x\colon A \vdash \pi_1\colon C \qquad \Sigma;\Gamma, y\colon B \vdash \pi_2\colon C}{\Sigma;\Gamma \vdash \text{case}\ \pi\ \text{inl}\ x \mapsto \pi_1; \text{inr}\ y \mapsto \pi_2\colon C}\ (\vee E)$$

$$\frac{\Sigma;\Gamma \vdash \pi\colon \bot}{\Sigma;\Gamma \vdash \text{absurd}\ \pi\colon C}\ (\bot E)$$

$$\frac{\Sigma;\Gamma \vdash \pi_1\colon A \Rightarrow B \qquad \Sigma;\Gamma \vdash \pi_2\colon A}{\Sigma;\Gamma \vdash \pi_1\pi_2\colon B}\ (\Rightarrow E) \qquad \frac{\Sigma;\Gamma \vdash \pi\colon \forall x.\,A \qquad \Sigma \vdash t \ \ \textit{term}}{\Sigma;\Gamma \vdash \pi[t]\colon A[t/x]}\ (\forall E)$$

$$\frac{\Sigma;\Gamma \vdash \pi_1\colon \exists x.\,A \qquad \Sigma, x;\Gamma, u\colon A \vdash \pi_2\colon C}{\Sigma;\Gamma \vdash \text{let}\ (x,u) = \pi_1\ \textit{in}\ \pi_2\colon C}\ (\exists E)$$

The syntax of proof terms is designed so as to indicate how proofs of the judgements in the conclusions are obtained from the proofs of the judgements in the premises. Consider, for example, the conclusion of $(\vee E)$:

$$\Sigma;\Gamma \vdash \text{case}\ \pi\ \text{inl}\ x \mapsto \pi_1; \text{inr}\ y \mapsto \pi_2\colon C.$$

This says: to obtain a proof of a proposition C (from the premises Γ), take a proof π of the disjunction $A \vee B$; if π proves A, bind the corresponding proof to x and call π_1 for which the current assumptions Γ suffice, since the only additional one (the assumption that A is true) is proven by x; if π proves B, bind the corresponding proof to y and call π_2 in an analogous manner.

Alternatively, we can view the last set of rules as term formation rules for a generalization of simply typed λ-calculus – the proof terms are generalized λ-terms. This connection between λ-calculus and proof theory is called the *Curry-Howard correspondence*. We add the following rule of *double negation elimination*, in order to fully capture classical logic (even though this rule largely disrupts the Curry-Howard correspondence):

$$\frac{\Sigma;\Gamma \vdash \pi\colon (C \Rightarrow \bot) \Rightarrow \bot}{\Sigma;\Gamma \vdash \mathit{nne}(\pi)\colon C}\ (\neg\neg E)$$

(we thus assume that negation $\neg A$ is encoded as $A \Rightarrow \bot$).

Your task: Design and program an interactive proof assistant in Haskell, implementing the above rules of natural deduction. Mind the following:

- No need to implement the entire system in detail as it stands. In particular, no need to implement formation rules for $\Sigma \vdash t \;\; \mathit{term}$ and $\Sigma \vdash A \;\; \mathit{prop}$. It suffices to introduce a handful of functional and predicate symbols statically and specify terms and propositions over them by the corresponding grammars.
- It is not necessary to implement proof terms – the entire calculus perfectly works if they are completely omitted.
- Use the state monad (transformer) to store the current proof state, which is a stack of pending judgements. Every rule transforms such a stack as follows: it pops the top judgement, matches it with the conclusion of the rule, and pushes the premises of the rule back on the stack. The proof is finished once the stack becomes empty.
- The interaction with the user can be organised as follows: the user is asked to provide the name of a rule and possibly further input, such as a term in the $(\exists I)$ rule. The rule is either applied, if possible, or an error message is displayed. This is repeated in a loop.
- You can take orientation from Coq's tactic language. E.g. $(\Rightarrow I)$ corresponds to `intro` and $(\wedge I)$ to `split`, etc.

Test your implementation by proving

1. the excluded middle law $P() \vee (P() \Rightarrow \bot)$;
2. propositional de Morgan laws: $((P() \vee Q()) \Rightarrow \bot) \Rightarrow (P() \Rightarrow \bot) \wedge (Q() \Rightarrow \bot)$ and $((P() \wedge Q()) \Rightarrow \bot) \Rightarrow (P() \Rightarrow \bot) \vee (Q() \Rightarrow \bot)$;
3. the following de Morgan law for quantifiers: $(\exists x.\, P(x) \Rightarrow \bot) \Rightarrow \forall x.\, (P(x) \Rightarrow \bot)$.

Hint: Note that you cannot derive (1) without using $(\neg\neg E)$. Think of (sparing) use of $(\neg\neg E)$ for proving other formulas.

Exercise 2 Not Quite Contextual Equivalence, Continued (5 Points)

Recall two implementations of Fibonacci numbers from Assignment 2:

```
fib 0 = 1
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)

fib2 0 = (1 , 1)
fib2 n = let (fn_1, fn) = fib2 (n - 1) in (fn, fn_1 + fn)
```

Show that PCF terms for `fib` and `fst $ fib2` are denotationally equal.

Hint: Obtain recursive equations for $[\![f]\!]$, $[\![f_1]\!]$, $[\![f_2]\!]$ and use them to show that $[\![f]\!] = [\![f_1]\!]$ by induction over the input, where f, f_1, f_2 are the PCF terms for `fib`, `fst $ fib2` and `snd $ fib2` respectively.

Exercise 3 The Diagonal Identity (5 Points)

Given a domain D and a continuous function $f\colon D \times D \to D$, the following *diagonal identity* holds true:

$$\mu x.\, \mu y.\, f(x,y) = \mu x.\, f(x,x)$$

where $\mu x.\, t$ abbreviates $\mu\,(\lambda x.\, t)$. This law is essentially responsible for the fact that nested recursion loops (in pretty much *any* programming language) can be equivalently replaced by a single recursion loop. In order to prove it, we need to show the inequalities

$$\begin{aligned} \mu x.\, \mu y.\, f(x,y) &\sqsubseteq \mu x.\, f(x,x) \\ \mu x.\, f(x,x) &\sqsubseteq \mu x.\, \mu y.\, f(x,y) \end{aligned}$$

Prove one of them.

Hint: One of these inequations can be deduced merely from the fact that μ calculates *least fixpoints*, that is, always: $\mu x.\, t = t[\mu x.\, t/x]$ and $\mu x.\, t$ is the least element with this property.