Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Lochbihler and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 2: Natural Deduction and Recursion

Submission deadline: March 3rd, 2014 (before 11:00 am)

Please, in addition to your name, mark your submission with your exercise group, i.e., tutor name and day of group (Tuesday or Wednesday). If you submit your solutions via email, please start the email's subject with `[FMFP]`. In case you are not yet enrolled in an exercise group, send an email to `omaric@inf.ethz.ch`.

Assignment 1:

Explain whether you can α-convert the following pairs of formulas into one another:

(a) $\forall x. p(x) \wedge q(y)$ and $\forall y. p(y) \wedge q(x)$

(b) $\forall x. \forall y. p(x,y) \wedge p(x,z) \wedge \exists z. p(x,z)$ and $\forall y. \forall x. p(y,x) \wedge p(y,z) \wedge \exists z. p(y,z)$

Consider the following formula. Instantiate the outermost bound variable x with $f(y)$:

(c) $\forall x. p(y) \wedge p(x) \wedge (\forall y. q(x,y)) \wedge \exists x. p(x)$

Assignment 2:

For each of the following formulas, find two structures with universe $\{a,b,c\}$ and nonempty relations. One that satisfies the formula and another one that does not satisfy it.

(a) $(\exists x.\, p(x)) \wedge (\exists y.\, q(y)) \rightarrow (\exists x.\, p(x) \wedge q(x))$

(b) $\forall x.\, (\exists y.\, r(x,y) \wedge q(y)) \rightarrow (\forall y.\, r(x,y) \rightarrow q(y))$

(c) $\forall x.\, \forall y.\, r(x,y) \rightarrow r(y,x) \rightarrow x = y$

Hint: All the structures $\mathcal{A}_i$ are of the form $\mathcal{A}_i = (U_{\mathcal{A}_i}, I_{\mathcal{A}_i})$ for $U_{\mathcal{A}_i} = \{a,b,c\}$ and $I_{\mathcal{A}_i}(s) = \{\ldots\}$, for each of the predicate symbols s occurring in the respective formula.

Assignment 3:

In this exercise, we work with intuitionistic predicate logic. The corresponding natural deduction rules are listed below.

$$\frac{}{\Gamma, A \vdash A}\ \textit{axiom} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to E$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot E \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg I \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash B} \neg E$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge EL \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge ER$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee IL \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee IR \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C} \vee E$$

$$\frac{\Gamma \vdash A(x)}{\Gamma \vdash \forall x.\, A(x)} \forall I^{*} \qquad \frac{\Gamma \vdash \forall x.\, A(x)}{\Gamma \vdash A(t)} \forall E$$

$$\frac{\Gamma \vdash A(t)}{\Gamma \vdash \exists x.\, A(x)} \exists I \qquad \frac{\Gamma \vdash \exists x.\, A(x) \qquad \Gamma, A(x) \vdash B}{\Gamma \vdash B} \exists E^{**}$$

Side conditions: (*) x does not occur free in any formula in Γ and (**) x neither occurs free in any formula in Γ nor in B.

Recall that $\to$ is right-associative, while $\wedge$ and $\vee$ are left-associative. Moreover, $\neg$ binds stronger than $\wedge$, which binds stronger than $\vee$, which in turn binds stronger than $\to$. Also note that the scope of the quantifiers extends as far to the right as possible.

Prove that the following statements are valid in intuitionistic predicate logic for formula P and Q.

(a) $((\exists x.\, P(x)) \to Q) \to \forall x.\, P(x) \to Q$, where x does not occur free in Q.

(b) $(\exists x.\, P(x) \wedge Q(x)) \to (\exists x.\, P(x)) \wedge (\exists y.\, Q(y))$

(c) $(\forall x.\, P(x) \to Q(x)) \to \forall x.\, \neg Q(x) \to \neg P(x)$

Assignment 4:

In this assignment you will develop a Haskell program based on Newton's method for calculating the square root of a nonnegative `Double`. Since the type `Double` is of limited precision, your square root function `root :: Double -> Double` will compute the square root up to some suitably small error `eps`.

(a) Write a function `improve :: Double -> Double -> Double` that improves your approximation. The first argument is the number from which you want to calculate the square root and the second argument is the approximation you have calculated so far. Newton's method says that if y_n is an approximation of $\sqrt{x}$ then

$$y_{n+1} = \frac{y_n + x/y_n}{2}$$

is a better approximation.

(b) Write a function `goodEnough :: Double -> Double -> Bool` that checks whether your approximation is in the error bound `eps :: Double`. More precisely, the approximation y is good enough, compared to the prior approximation y', if

$$|\frac{y - y'}{y'}| < \texttt{eps}$$

You can chose, e.g., `eps` as 0.001.

(c) Use the functions `improve` and `goodEnough` for writing the function `root`. As the first approximation y_0 you can use, e.g., $y_0 = 1$.

(d) Now extend this to work with I/O. That is, write a main function that asks the user for a number to compute the root of. Your main function then prints out the resulting square root for positive input, using the definition of `root` above, and starts over. For negative input your program aborts.

Assignment 5:

Write a Haskell function `cntChange :: Int -> Int` that computes the number of ways to change any given amount of money (expressed in Rappen) by using CHF coins.
Hint: Think recursively. The number of ways to change amount a using n different kinds of coins is equal to the sum of

- the number of ways to change a using all but the first kind of coin, and
- the number of ways to change amount $a - d$ using the n kinds of coins, where d is the denomination of the first kind of coin.

Test your program thoroughly!

Assignment 6 (headache of the week):

Consider the following informal argumentation that the function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = x$, for all $x \in \mathbb{R}$ is continuous, that is, f fulfills the property

$$\forall c.\, \forall \epsilon.\, \epsilon > 0 \to \exists \delta.\, \delta > 0 \land \forall x.\, |x - c| < \delta \to |f(x) - f(c)| < \epsilon\,.$$

Claim: If $f : \mathbb{R} \to \mathbb{R}$ is $f(x) = x$, for all $x \in \mathbb{R}$, then f is continuous.
Proof: Let c be an arbitrary real number. Let ϵ be an arbitrary positive real number. Choose $\delta = \epsilon$. Note that $\delta > 0$ since $\epsilon > 0$. Let x be an arbitrary real number. Assume that $|x - c| < \delta$. As $f(x) = x$ and $f(c) = c$, it follows that $|f(x) - f(c)| < \epsilon$, because we chose $\delta = \epsilon$. **Q.E.D.**

Turn the above argumentation into a formal proof, that is, use the proof rules from the lecture to derive

$$\vdash \big(\forall x.\, f(x) = x\big) \rightarrow \forall c.\, \forall \epsilon.\, \epsilon > 0 \rightarrow \exists \delta.\, \delta > 0 \land \forall x.\, |x - c| < \delta \rightarrow |f(x) - f(c)| < \epsilon\,.$$

Make yourself clear how the proof rules in your derivation reflect the informal reasoning steps.
Hint: You do not need any properties about subtraction and the ordering over the reals. Treat $-$, $|\cdot|$, and $<$ as function and predicate symbols. In particular, $-$ is a binary function symbol, $|\cdot|$ a unary function symbol, and $<$ a binary predicate symbol. The symbols $-$ and $<$ are written in infix notation.