Answers to Exercises Toegepaste Logica 99-00

1. a.

$$\dfrac{\dfrac{\dfrac{(A\to B\to B\to C)^x \quad \dfrac{\dfrac{A^u}{A\to A}\,I[u]\to}{A}\,E\to A^y}{B\to B\to C}\,E\to \quad B^z}{B\to C}\,E\to \quad B^z}{\dfrac{\dfrac{C}{B\to C}\,I[z]\to}{A\to B\to C}\,I[y]\to}{(A\to B\to B\to C)\to A\to B\to C}\,I[x]\to$$

b.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x\in \mathsf{Var}_{A\to B\to B\to C}}{x : A\to B\to B\to C} \quad \dfrac{\dfrac{\dfrac{u\in \mathsf{Var}_A}{u:A} \quad \dfrac{y\in \mathsf{Var}_A}{y:A}}{\lambda u{:}A.\, u : A\to A}}{(\lambda u{:}A.\, u)\, y : A}}{x\,((\lambda u{:}A.\, u)\, y) : B\to B\to C} \quad \dfrac{z\in \mathsf{Var}_B}{z:B}}{x\,((\lambda u{:}A.\, u)\, y)\, z : B\to C} \quad \dfrac{z\in \mathsf{Var}_B}{z:B}}{x\,((\lambda u{:}A.\, u)\, y)\, z\, z : C}}{\lambda z{:}B.\, x\,((\lambda u{:}A.\, u)\, y)\, z\, z : B\to C}}{\lambda y{:}A.\, \lambda z{:}B.\, x\,((\lambda u{:}A.\, u)\, y)\, z\, z : A\to B\to C}}{\lambda x{:}A\to B\to B\to C.\, \lambda y{:}A.\, \lambda z{:}B.\, x\,((\lambda u{:}A.\, u)\, y)\, z\, z : (A\to B\to B\to C)\to A\to B\to C}$$

c. $\lambda x{:}A\to B\to B\to C.\, \lambda y{:}A.\, \lambda z{:}B.\, x\,((\lambda u{:}A.\, u)\, y)\, z\, z \to_\beta$
$\lambda x{:}A\to B\to B\to C.\, \lambda y{:}A.\, \lambda z{:}B.\, x\, y\, z\, z$

d.

$$\dfrac{\dfrac{\dfrac{\dfrac{(A\to B\to B\to C)^x \quad A^y}{B\to B\to C}\,E\to \quad B^z}{B\to C}\,E\to \quad B^z}{\dfrac{\dfrac{C}{B\to C}\,I[z]\to}{A\to B\to C}\,I[y]\to}{(A\to B\to B\to C)\to A\to B\to C}\,I[x]\to$$

2. a. Three possible ways to extend intuitionistic logic to classical logic are as follows:

   i. Add the double negation rule:

   $$\dfrac{\neg\neg A}{A}$$

   ii. Add $A\vee\neg A$ as an axiom.

iii. Add Peirce's Law: $((A \to B) \to A) \to A$ as an axiom.

b. Formulas in minimal logic correspond to types of simply typed lambda calculus as follows:

- a propositional variable corresponds to a type variable,
- the connective $\to$ corrseponds to the type constructor $\to$, so a formula of the form $A \to B$ corresponds to the type $A \to B$.

Proofs in minimal logic correpond to type derivations in simply typed lambda calculus as follows:

- an assumption corresponds to a type variable,
- implication introduction corresponds to abstraction,
- implication elimination corresponds to application.

A detour in minimal logic corresponds to a beta-redex in simply typed lambda calculus. Elimination of a detour corresponds to a beta-reduction step according to the beta-reduction rule: $(\lambda x{:}A.\, M)\, N \to_\beta M[x := N]$.

(a) A type A is said to be inhabited if there is a term M with $M : A$. The inhabitation problem is the question whether, given a type A, a closed term M exists with $M : A$. This question corresponds in minimal logic to the question whether the formula A is a tautology. So inhabitation corresponds to provability.

(b) Type checking: is P a term of type A? The corresponding question in minimal logic is proof checking: is P a proof of the formula A?

3. a.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{plus} : \mathsf{Nat}\to\mathsf{Nat}\to\mathsf{Nat}} \quad \dfrac{x \in \mathsf{Var}_{\mathsf{Nat}}}{x : \mathsf{Nat}} \quad \dfrac{n : \mathsf{Var}_{\mathsf{Nat}}}{n : \mathsf{Nat}}}{\mathsf{plus}\ x\ n : \mathsf{Nat}}}{\lambda y{:}\mathsf{Nat}.\ \mathsf{plus}\ x\ n : \mathsf{Nat}\to\mathsf{Nat}} \quad \dfrac{m \in \mathsf{Var}_{\mathsf{Nat}}}{m : \mathsf{Nat}}}{\lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{plus}\ x\ n : \mathsf{Nat}\to\mathsf{Nat}\to\mathsf{Nat}} \quad \mathsf{z} : \mathsf{Nat}}{\mathsf{r}_{\mathsf{Nat}}(\mathsf{z}, \lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{plus}\ x\ n, m) : \mathsf{Nat}}}{\lambda n{:}\mathsf{Nat}.\ \mathsf{r}_{\mathsf{Nat}}(\mathsf{z}, \lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{plus}\ x\ n, m) : \mathsf{Nat}\to\mathsf{Nat}}}{\lambda m{:}\mathsf{Nat}.\ \lambda n{:}\mathsf{Nat}.\ \mathsf{r}_{\mathsf{Nat}}(\mathsf{z}, \lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{plus}\ x\ n, m) : \mathsf{Nat}\to\mathsf{Nat}\to\mathsf{Nat}}$$

b.

$$\begin{array}{ll}
\mathsf{plus}\ \mathsf{z}\ q & =_{\mathbf{T}} \\
(\lambda m{:}\mathsf{Nat}.\ \lambda n{:}\mathsf{Nat}.\ \mathsf{r}_{\mathsf{Nat}}(n, \lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{s}(x), m))\ \mathsf{z}\ q & =_{\mathbf{T}} \\
\mathsf{r}_{\mathsf{Nat}}(q, \lambda x{:}\mathsf{Nat}.\ \lambda y{:}\mathsf{Nat}.\ \mathsf{s}(x), \mathsf{z}) & =_{\mathbf{T}} \\
q &
\end{array}$$

$$
\begin{array}{ll}
\mathsf{plus}\ \mathsf{s}(p)\ q & =_{\mathbf{T}} \\
(\lambda m{:}\mathsf{Nat}.\,\lambda n{:}\mathsf{Nat}.\,\mathsf{r}_{\mathtt{Nat}}(n, \lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x), m))\,(\mathsf{s}(p))\,q & =_{\mathbf{T}} \\
\mathsf{r}_{\mathtt{Nat}}(q, \lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x), \mathsf{s}(p)) & =_{\mathbf{T}} \\
(\lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x))\,\mathsf{r}_{\mathtt{Nat}}(q, \lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x), p)\,p & =_{\mathbf{T}} \\
\mathsf{s}(\mathsf{r}_{\mathtt{Nat}}(q, \lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x)), p) & =_{\mathbf{T}} \\
\mathsf{s}((\lambda m{:}\mathsf{Nat}.\,\lambda n{:}\mathsf{Nat}.\,\mathsf{r}_{\mathtt{Nat}}(n, \lambda x{:}\mathsf{Nat}.\,\lambda y{:}\mathsf{Nat}.\,\mathsf{s}(x), m))\,p\,q) & =_{\mathbf{T}} \\
\mathsf{s}(\mathsf{plus}\ p\ q) &
\end{array}
$$

4. a. $\lambda x{:}A.\,(M\,x) \to_\eta M$ with x not free in M.

   b. A type variable b is of the form $A_1 \to \ldots \to A_n \to b$, with $n = 0$. If B is of the form $B_1 \to \ldots \to B_k \to b$ with b a type variable, then $A \to B$ is of the form $A \to B_1 \to \ldots \to B_k \to b$ with b a type variable.

   c. See page 45.

5. a.

```
Inductive natlist : Set := nil : natlist |
                           cons : nat -> natlist -> natlist
```

   Here a new `Set` is declared with name `natlist`. Its constructors are `nil` and `cons`. It is an inductive definition, that is, `natlist` is the smallest set that contains `nil` and is closed under the use of `cons`.

   b. The type of $\texttt{natlist}_{\texttt{ind}}$ is

```
(P : natlist -> Prop)
( P nil ) ->
((l : natlist) (P l) -> (n : nat) (P (cons n l))) ->
(l : natlist) (P l)
```

   c. If a property `P` holds for the empty list, and we have the following: if the property `P` holds for the list `l` and then it holds for the list `cons n l` for evry natural number `n`, then the property `P` holds for every list `l`

6. a. A type A is interpreted as the set of strongly normalizing terms of type A.

   b. The proof proceeds by induction on the derivation of $M : B$. See page 57.

7. a. We assume a base type Terms. Then a function symbol f of arity n is represented by a distinguished variable f of type $\mathsf{Terms} \to \ldots \to \mathsf{Terms} \to \mathsf{Terms}$ with $n + 1$ times Terms.

   b. Also here we assume a base type Terms (the same as in a). Then a predicate symbol r of arity n is interpreted as a distinguished variable r of type $\mathsf{Terms} \to \ldots \to \mathsf{Terms} \to \star$ with n times Terms.

c.

- A formula of the form $r\ M_1 \ldots M_n$ corresponds to a type of the form $r\ M_1 \ldots M_n$, with r the predicate symbol interpreted as a variable.
- A formula of the form $A \rightarrow B$ corresponds to a type $\Pi x{:}A.\ B$.
- A formula of the form $\forall x.\ A$ corresponds to a type $\Pi x{:}\mathsf{Terms}.\ A$.

d. The universe `Set` corresponds to $\star$. The universe `Prop` corresponds to $\star$. The universe `Type` corresponds to $\Box$.

e.

f. The conversion rule is necessary because types may contain terms. This is for instance the case in $\mathsf{Natlist}\,((\lambda x{:}\mathsf{Nat}.\ x)\,5)$. With the conversion rule, this type can be used as the type $\mathsf{Natlist}\,5$.

8. a. The identity functions $\lambda x{:}\mathsf{Nat}.\ x$ on natural numbers and $\lambda x{:}\mathsf{Bool}.\ x$ on booleans both do the same thing namely nothing. It is then handy to have to define this function only once instead of twice. This can be done by defining a polymorphic identity function: $\lambda a{:}\star\,.\,\lambda x{:}a.\ x$. From this polymorphic identity function we can obtain the identity on natural numbers by applying it to Nat as follows: $(\lambda a{:}\star\,.\,\lambda x{:}a.\ x)\,\mathsf{Nat} \rightarrow_\beta \lambda x{:}\mathsf{Nat}.\ x$ and the identity on booleans by applying it to Bool as follows: $(\lambda a{:}\star\,.\,\lambda x{:}a.\ x)\,\mathsf{Bool} \rightarrow_\beta \lambda x{:}\mathsf{Bool}.\ x$.

b.

9. a. Program extraction is the technique to extract a program or algorithm from the correctness proof of a specification. For instance, the specification of a sorting algorithm is that for every list l a list l' exists such that l' is ordered and moreover l' is a permutation of l. Program extraction yields from the proof of correctness of a specification an algorithm that transforms a list l into a list l' that is ordered and that is moreover a permutation of l.

b. The tactic `Program` is used to do something like the opposite of program extraction. A program is converted into a skeleton of a proof of its correctness. The tactic generates the goals that remain to be proved.