Logic and Verification
Informatics
Technical University of Munich

Eexam
Place student sticker here

Note:

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

Lambda Calculus

Exam:	IN2358 / Endterm	**Date:**	Wednesday 23rd February, 2022
Examiner:	Prof. Tobias Nipkow	**Time:**	08:00 – 09:30

	P 1	P 2	P 3	P 4	P 5
I					

Working instructions

- This exam consists of **12 pages** with a total of **5 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 43 credits. There is one bonus exercises on this exam which means that the grading scale for 40 credits (instead of 43 credits) will be applied.
- Detaching pages from the exam is prohibited.
- Allowed resources:
  - one **A4 sheet** with hand-written notes on both sides
  - one **analog dictionary** English ↔ native language **without annotations**
- Subproblems marked by * can be solved without results of previous subproblems.
- Do not write with red or green colors nor use pencils.
- Physically turn off all electronic devices, put them into your bag and close the bag.

Left room from ________ to ________ / Early submission at ________

Problem 1 List programming (9 credits)

Recall the fold encoding from the exercise sheet where a list $[x, y, z]$ is represented as $\lambda c\ n.\ c\ x\ (c\ y\ (c\ z\ n))$. Accordingly, the empty list is defined as nil $:= \lambda c\ n.\ n$

0 1 2 3

a)* Define a function that appends two lists in this encoding, i.e. it should hold that

$$\text{append}\ (\lambda c\ n.\ c\ x_1\ (\ldots(c\ x_k n)\ldots))\ (\lambda c\ n.\ c\ y_1\ (\ldots(c\ y_l\ n)\ldots)) \to_\beta^* (\lambda c\ n.\ c\ x_1\ (\ldots(c\ x_k\ (c\ y_1\ (\ldots(c\ y_l\ n)\ldots)))\ldots)).$$

0 1 2 3

b)* Implement the append function again using fix.

0 1 2 3

c)* **BONUS**: Remember that we can prepend an element to a list with cons $:= (\lambda x\ l.\ \lambda c\ n.\ c\ x\ (l\ c\ n))$. Define a function that computes the tail of a list, i.e. it should hold that

$$\text{tail}\ (\text{cons}\ x\ l) \to_\beta^* l.$$

Hint: You might want to use pairs.

Problem 2 **Confluence (10 credits)**

Let $\rightarrow_1 \subseteq A \times A$ and $\rightarrow_2 \subseteq A \times A$. Let all variables below range over A. Notation: $x \rightarrow_1^* \rightarrow_2^* z$ means $\exists y.\ x \rightarrow_1^* y \wedge y \rightarrow_2^* z$.

Assume

B: $x \rightarrow_1^* y \wedge x \rightarrow_2 z \Rightarrow \exists u.\ y \rightarrow_2^* u \wedge z \rightarrow_1^* \rightarrow_2^* u$ for all x, y, z

C: $\rightarrow_2$ is confluent

Prove that $x \rightarrow_1^* y \wedge x \rightarrow_2^* z \Rightarrow \exists u.\ y \rightarrow_2^* u \wedge z \rightarrow_1^* \rightarrow_2^* u$ for all x, y, z.

☐ 0
☐ 1
☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
☐ 8
☐ 9
☐ 10

The proof must be given in the standard verbal style. However, it is very helpful to draw a diagram, in particular as a starting point.

Problem 3 Typing Lists (10 credits)

In this exercise, we add a primitive type of lists (and booleans) to the simply typed lambda calculus $\lambda^{\rightarrow}$. The type bool can be constructed using either of the constants true or false. The corresponding typing rules are unsurprisingly

$$\Gamma \vdash \text{true} : \text{bool} \quad \text{and} \quad \Gamma \vdash \text{false} : \text{bool}.$$

For lists we need to add quite a number of constants to the calculus. Similarly to the function abstraction $\lambda x : \tau.\ t$ where the type τ of x is explicitly annotated, each of the constants takes a type of list elements α as an argument. We have

- the empty list nil$[\alpha]$,
- the list constructor cons$[\alpha]$ t_1 t_2 that prepends the term t_1 to the list t_2,
- the function null$[\alpha]$ t that tests whether the list represented by t is empty,
- the function head$[\alpha]$ t that extracts the head of the list t, and
- the function tail$[\alpha]$ t that returns the tail of the list t.

We define the following β-reduction rules for the list constants:

$$\frac{t_1 \rightarrow_\beta t_1'}{\text{cons}[\alpha]\ t_1\ t_2 \rightarrow_\beta \text{cons}[\alpha]\ t_1'\ t_2} \qquad \frac{t_2 \rightarrow_\beta t_2'}{\text{cons}[\alpha]\ t_1\ t_2 \rightarrow_\beta \text{cons}[\alpha]\ t_1\ t_2'}$$

$$\frac{}{\text{null}[\alpha]\ (\text{nil}[\beta]) \rightarrow_\beta \text{true}} \qquad \frac{}{\text{null}[\alpha]\ (\text{cons}[\beta]\ t_1\ t_2) \rightarrow_\beta \text{false}}$$

$$\frac{}{\text{head}[\alpha]\ (\text{cons}[\beta]\ t_1\ t_2) \rightarrow_\beta t_1} \qquad \frac{}{\text{tail}[\alpha]\ (\text{cons}[\beta]\ t_1\ t_2) \rightarrow_\beta t_2}$$

Answer the following questions:

0 1 2

a)* Three reduction rules are missing; state them.

0 1 2 3 4 5

b)* Consider the call-by-value reduction relation $\rightarrow_{\text{cbv}}$ as defined in the lecture. Modify the above rules to obtain a call-by-value relation $\rightarrow_{\text{cbv}}$ for $\lambda^{\rightarrow}$ with lists.
Note: The rules of part a) need not be modified.

Since we are working with $\lambda^{\to}$, we not only want to evaluate lists but also type them.

c)* We use list α to denote the type of lists with elements of type α. Give the typing rules for the list constants.
Hint: The types of head[α] (nil[α]) and tail[α] (nil[α]) should be α and list α, respectively.

☐ 0
☐ 1
☐ 2
☐ 3

Problem 4 **Type Inference for Let (5 credits)**

Consider $\lambda^{\rightarrow}$ extended with `let` and consider the following the typing problem

$$x : A \vdash \texttt{let}\ \ y = \lambda z.\,(z\ x)\ \ \texttt{in}\ \ (y\ (\lambda v.\,x)) : ?\tau$$

where A is a type variable.

0 1 2

a)* Find a most general type schema σ with $x : A \vdash \lambda z.\,(z\ x) : \sigma$. You may, but you do not need to draw a type derivation tree.

0 1 2 3

b) Draw the type derivation tree for

$$x : A, y : \sigma \vdash (y\ (\lambda v.\,x)) : ?\tau$$

Of course, with the correct type for $?\tau$.
Use only the introduction and elimination rules for $\rightarrow$ and $\forall$ and the standard assumption rule

$$\Gamma \vdash x : \tau \quad \text{where} \quad \Gamma(x) = \tau.$$

Problem 5 **Logic (9 credits)**

In this exercise, we consider intuitionistic logic with negation, conjunction, and disjunction, but without implication. We write $\Gamma \vdash_I A$ to mean that A is provable from Γ in *intuitionistic* logic. To obtain *classical* logic, we add the classical contradiction rule

$$\frac{\Gamma, \neg A \vdash_C \bot}{\Gamma \vdash_C A}\ \textsc{CContr}$$

where we take $\Gamma \vdash_C A$ to mean that A is provable from Γ in classical logic. Furthermore, we define a function $-^*$ that takes formulas to formulas:

$$\begin{array}{rll}
\bot^* & = \bot & \\
p^* & = \neg\neg p & \text{for atomic } p \\
(\neg A)^* & = \neg(A^*) & \\
(A \wedge B)^* & = A^* \wedge B^* & \\
(A \vee B)^* & = \neg(\neg(A^*) \wedge \neg(B^*)) &
\end{array}$$

Fact: It can be shown that A^* is negative for any formula A. Recall that we call a formula negative if propositional variables P only occur in negated form $\neg P$ in the formula.

We want to prove that classical logic can be embedded into intuitionistic logic in the sense that $\Gamma \vdash_C A \implies \Gamma^* \vdash_I A^*$ where Γ^* means that we apply $-^*$ pointwise.

Show the statement by *induction on the derivation* of $\Gamma \vdash_C A$. You only need to consider the cases where $\Gamma \vdash_C A$ was proved by the rules $\textsc{CContr}$ and $\vee I_1$.

☐ 0
☐ 1
☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
☐ 8
☐ 9

Note: In the homework we proved that if A is negative, $\Gamma \vdash_I \neg\neg A$ implies $\Gamma \vdash_I A$. You may refer to this fact as $(*)$.

Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.