



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
Examiner: Prof. Tobias Nipkow

Date: Wednesday 23rd February, 2022
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 $\text{nil} := \lambda c n. n$

- 0
1
2
3
- a)* Define a function that appends two lists in this encoding, i.e. it should hold that
- $$\text{append } (\lambda c n. c x_1 (\dots (c x_k n) \dots)) (\lambda c n. c y_1 (\dots (c y_l n) \dots)) \rightarrow_{\beta}^* (\lambda c n. c x_1 (\dots (c x_k (c y_1 (\dots (c y_l n) \dots))) \dots))$$

```
append := (\lambda l_1 l_2. \lambda c n. l_1 c (l_2 c n))
```

- 0
1
2
3
- b)* Implement the append function again using fix.

```
null := (\lambda l. l (\lambda h t. false) true)
head := (\lambda l. l (\lambda h t. h) false)
append := fix (\lambda f l_1 l_2. if (null l_1) l_2 (cons (head l_1) (f (tail l_1) l_2)))
```

- 0
1
2
3
- c)* **BONUS:** Remember that we can prepend an element to a list with $\text{cons} := (\lambda x l. \lambda c n. c x (l c n))$. Define a function that computes the tail of a list, i.e. it should hold that

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

Hint: You might want to use pairs.

```
tail := (\lambda l. fst (l (\lambda x a. pair (snd a) (cons x (snd a))) (pair nil nil)))
```

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 .

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.

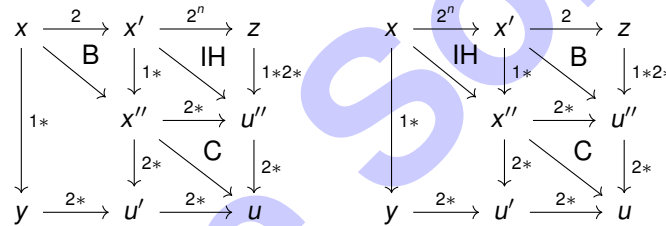
We prove

$$x \rightarrow_1^* y \wedge x \rightarrow_2^n z \Rightarrow \exists u. y \rightarrow_2^* u \wedge z \rightarrow_1^* \rightarrow_2^* u \quad (2.1)$$

for all x, y, z by induction on n .

Base case: If $n = 0$ then $u = y$ does the job.

Induction step: Assume (1) and assume $x \rightarrow_1^* y \wedge x \rightarrow_2^{n+1} z$. Thus there is an x' s.t. $x \rightarrow_2 x' \rightarrow_2^n z$. By B there is a u' s.t. $y \rightarrow_2^* u' \wedge x' \rightarrow_1^* \rightarrow_2^* u'$. Thus there is an x'' s.t. $x' \rightarrow_1 x'' \rightarrow_2^* u'$. By IH there is a u'' s.t. $x'' \rightarrow_2^* u'' \wedge z \rightarrow_1^* \rightarrow_2^* u''$. By C there is a u s.t. $u' \rightarrow_2^* u \wedge u'' \rightarrow_2^* u$. Thus $y \rightarrow_2^* u$ and $z \rightarrow_1^* \rightarrow_2^* u$.



0
1
2
3
4
5
6
7
8
9
10

Problem 3 Typing Lists (10 credits)

In this exercise, we add a primitive type of lists (and booleans) to the simply typed lambda calculus λ^{\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 $[\alpha]$ t1 t2` 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:

$$\frac{t_1 \rightarrow_{\beta} t'_1}{\text{cons}[\alpha] t_1 t_2 \rightarrow_{\beta} \text{cons}[\alpha] t'_1 t_2} \quad \frac{t_2 \rightarrow_{\beta} t'_2}{\text{cons}[\alpha] t_1 t_2 \rightarrow_{\beta} \text{cons}[\alpha] t_1 t'_2}$$

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

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

Answer the following questions:

0
1
2

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

$$\frac{t_1 \rightarrow_{\beta} t'_1}{\text{head}[\alpha] t_1 \rightarrow_{\beta} \text{head}[\alpha] t'_1} \quad \frac{t_1 \rightarrow_{\beta} t'_1}{\text{tail}[\alpha] t_1 \rightarrow_{\beta} \text{tail}[\alpha] t'_1} \quad \frac{t_1 \rightarrow_{\beta} t'_1}{\text{null}[\alpha] t_1 \rightarrow_{\beta} \text{null}[\alpha] t'_1}$$

0
1
2
3
4
5

b)* Consider the call-by-value reduction relation \rightarrow_{cbv} as defined in the lecture. Modify the above rules to obtain a call-by-value relation \rightarrow_{cbv} for λ^{\rightarrow} with lists.

Note: The rules of part a) need not be modified.

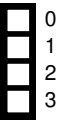
$$\frac{v_1 \text{ value} \quad t_2 \rightarrow_{\text{cbv}} t'_2}{\text{cons}[\alpha] v_1 t_2 \rightarrow_{\text{cbv}} \text{cons}[\alpha] v_1 t'_2}$$

$$\frac{v_1, v_2 \text{ values}}{\text{null}[\alpha] (\text{cons}[\beta] v_1 v_2) \rightarrow_{\text{cbv}} \text{false}}$$

$$\frac{v_1, v_2 \text{ values}}{\text{head}[\alpha] (\text{cons}[\beta] v_1 v_2) \rightarrow_{\text{cbv}} v_1} \quad \frac{v_1, v_2 \text{ values}}{\text{tail}[\alpha] (\text{cons}[\beta] v_1 v_2) \rightarrow_{\text{cbv}} v_2}$$

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

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


$$\Gamma \vdash \text{nil}[\alpha] : \text{list } \alpha$$
$$\frac{\Gamma \vdash t_1 : \alpha \quad \Gamma \vdash t_2 : \text{list } \alpha}{\Gamma \vdash \text{cons}[\alpha] t_1 t_2 : \text{list } \alpha}$$
$$\frac{\Gamma \vdash t : \text{list } \alpha}{\Gamma \vdash \text{null } t : \text{bool}}$$
$$\frac{\Gamma \vdash t : \text{list } \alpha}{\Gamma \vdash \text{head } t : \alpha}$$
$$\frac{\Gamma \vdash t : \text{list } \alpha}{\Gamma \vdash \text{tail } t : \text{list } \alpha}$$

Sample Solution

Problem 4 Type Inference for Let (5 credits)

Consider λ^{\rightarrow} extended with let and consider the following the typing problem

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

where A is a type variable.

- 0 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.

1

2

$$\sigma = \forall B. (A \rightarrow B) \rightarrow B$$

- 0 b) Draw the type derivation tree for

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

1

2

3

Of course, with the correct type for $?\tau$.

Use only the introduction and elimination rules for \rightarrow and \forall and the standard assumption rule

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

$$\frac{\frac{\Gamma \vdash y : \sigma}{\Gamma \vdash y : (A \rightarrow A) \rightarrow A} \quad \forall E \quad \frac{\Gamma, v : B \vdash x : A}{\Gamma \vdash \lambda v. x : A \rightarrow A} \rightarrow I}{\Gamma \vdash (y (\lambda v. x)) : A} \rightarrow E$$

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

$$\frac{\Gamma, \neg A \vdash_C \perp}{\Gamma \vdash_C A} \text{CCONTR}$$

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:

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

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 $\Gamma \vdash_C A \Rightarrow \Gamma^* \vdash_I A^*$ 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 CCONTR and $\vee I_1$.

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 (*).

0
1
2
3
4
5
6
7
8
9

We prove $\Gamma \vdash_C A \Rightarrow \Gamma^* \vdash_I A^*$ by induction on the derivation of $\Gamma \vdash_C A$.

- **Case CCONTR**

We get $\Gamma^*, (\neg A)^* \vdash_I \perp^* = \Gamma^*, \neg(A^*) \vdash_I \perp$ as an induction hypothesis, assume $\Gamma \vdash_C A$ and need to show that $\Gamma^* \vdash_I A^*$. We justify this with the proof below. In the proof the fact (*) is applicable to A^* since A^* is negative as stated above.

$$\frac{\frac{\text{I.H.}}{\Gamma^*, \neg(A^*) \vdash_I \perp}}{\Gamma^* \vdash_I \neg\neg(A^*)} (*)}{\Gamma^* \vdash_I A^*} (*)$$

- **Case $\vee I_1$**

For some arbitrary but fixed A and B , we assume $\Gamma \vdash_C A$ and need to show $\Gamma^* \vdash_I (A \vee B)^* = \Gamma^* \vdash_I \neg(\neg(A^*) \wedge \neg(B^*))$. Furthermore, we have the induction hypothesis $\Gamma^* \vdash_I A^*$. We give the following derivation for the goal:

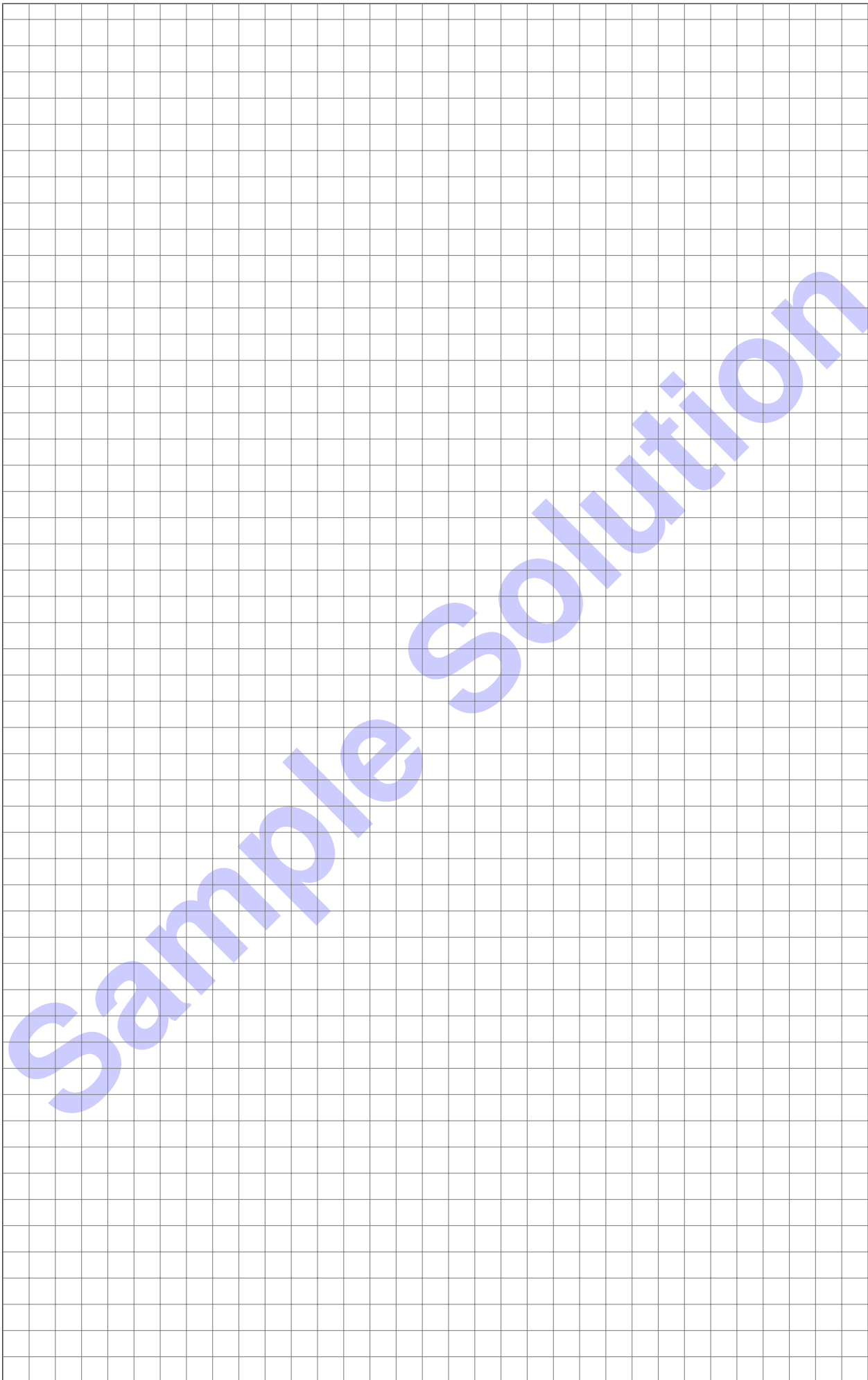
$$\frac{\frac{\text{I.H.}}{\Gamma' \vdash_I A^*} \quad \frac{\Gamma' \vdash_I \neg(A^*) \wedge \neg(B^*)}{\Gamma' \vdash_I \neg(A^*)} \wedge E_1}{\Gamma' \vdash_I \neg(A^*) \wedge \neg(B^*) \vdash_I \perp} \wedge E_1}{\Gamma^* \vdash_I \neg(\neg(A^*) \wedge \neg(B^*))}$$

Sample Solution

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

A large grid of graph paper, consisting of 20 columns and 30 rows of small squares, intended for writing solutions. A large, light blue watermark reading "Sample Solution" is oriented diagonally across the grid from the bottom-left to the top-right.

Sample Solution



Sample Solution