Foundations of Software Science (ソフトウェア基礎科学/ソフトウエア基礎)

Reporting Assignment

Deadline (firm): Dec. 13th, 2019 (23:59 JST)

How to submit: Make a PDF with LaTeX or Word, and submit it by email to the following mail address with the subject "`FSS 2019: Report 2`".

`kztk@ecei.tohoku.ac.jp`

Both report and email must contain your name and student ID.
I do not accept handwritten reports.

Grading: This report assignment is a substitute for a midterm exam. Let r be the score of this report, then the score of your midterm exam will be $\min(100, r)$. Thus, you do not need to answer all the questions. The minimal requirement would be answering 6 or 7 easy questions or 1 laborious question.

I. Consider the natural numbers defined by the following BNF

$$m, n ::= \mathsf{Z} \mid \mathsf{S}(n)$$

and the function *add* defined inductively as follows.

$$\begin{array}{lcl} add(Z, m) & = & m \\ add(S(n), m) & = & S(add(n, m)) \end{array}$$

(1) **(10 points)** Compute $add(\mathsf{S}(\mathsf{S}(\mathsf{Z})), \mathsf{S}(\mathsf{Z}))$.

(2) **(10 points)** Prove that Z is the left unit of *add*, i.e., $add(\mathsf{Z}, n) = n$ for all natural numbers n.

(3) **(20 points)** Prove by induction that Z is the right unit of *add*, i.e., $add(n, \mathsf{Z}) = n$ for all natural numbers n.

(4) **(20 points)** Prove by induction that $add(n, \mathsf{S}(m)) = \mathsf{S}(add(n, m))$ for all natural numbers n and m.

(5) **(35 points)** Prove by induction that *add* is associative.

II. Consider the untyped λ-calculus lectured in class.

(1) **(20 points)** Define *append* to be the following λ-term.

$$\mathit{append} = \lambda x.\lambda y.\lambda c.\lambda n.x\ c\ (y\ c\ n)$$

Let A_1, A_2, B_1 and B_2 be λ-terms in normal form. Reduce

$$\mathit{append}\ (\lambda c.\lambda n.c\ A_1\ (c\ A_2\ n))\ (\lambda c.\lambda n.c\ B_1\ (c\ B_2\ n))$$

to normal form. Write down each reduction step and underline the redexes chosen in the reduction sequence.

(2) **(25 points)** Terms like $\lambda c.\lambda n.c\ A_1\ (c\ A_2\ n)$ above are called Church lists. Specifically, $\lambda c.\lambda n.n$ represents the empty list, and $\lambda c.\lambda n.c\ A_1\ (c\ A_2\ (\ldots(c\ A_k\ n)\ldots))$ represents the list $[A_1, A_2, \ldots, A_k]$. Give λ-terms *head* and *tail* that have the following reduction sequences if $k > 0$.

$$\mathit{head}\ (\lambda c.\lambda n.c\ A_1\ (c\ A_2\ (\ldots(c\ A_k\ n)\ldots))) \longrightarrow^* A_1$$
$$\mathit{tail}\ (\lambda c.\lambda n.c\ A_1\ (c\ A_2\ (\ldots(c\ A_k\ n)\ldots))) \longrightarrow^* \lambda c.\lambda n.c\ A_2\ (\ldots(c\ A_k\ n)\ldots)$$

We do not specify the behavior of *head* and *tail* for $\lambda c.\lambda n.n$.

(3) **(20 points)** Give a λ-term *null* that checks whether an input Church list is empty or not. For example, the function behaves as below.

$$\mathit{null}\ (\lambda c.\lambda n.c\ A_1\ (c\ A_2\ (\ldots(c\ A_k\ n)\ldots))) \longrightarrow^* \begin{cases} \lambda x.\lambda y.x & (k = 0) \\ \lambda x.\lambda y.y & (k > 0) \end{cases}$$

(4) **(100 points)** Instead of having arbitrary λ-abstractions, a fixed set of predefined functions is known to be sufficient for Turing-completeness. What is the set? Justify your answer.

III. Consider the simply typed λ-calculus with sums and products lectured in class.

(1) **(15 points)** Give the type of the following λ-term.

$$\lambda s.\lambda z.s\ (s\ z)$$

Justify your answer by writing down its typing derivation.

(2) **(15 points)** Give a λ-term that has the type $(A \to B \to C) \to B \to A \to C$ under the empty type environment, where A, B, and C are some base types. Write down its typing derivation.

(3) **(15 points)** Check your answer to the previous question by writing a program and then checking its type. You may use a functional programming language such as OCaml or Haskell that contains simply-typed λ-calculus.

(4) **(30 points)** Let A, B, C and R be some base types. Give λ-terms that have the following types, respectively.

- $A \to B \to A$
- $(A \to B \to C) \to (A \to B) \to A \to C.$
- $A \to ((A \to R) \to R)$
- $(A \to B) \to ((A \to R) \to R) \to ((B \to R) \to R)$
- $((A \to R) \to R) \to (A \to ((B \to R) \to R)) \to ((B \to R) \to R)$

Check your answer by writing programs and checking their types in OCaml or Haskell.

(5) **(30 points)** Let us write $M_R\ A$ for $(A \to R) \to R$. Then, give a λ-term that has the following type.

$$((A \to M_R\ B) \to M_R\ A) \to M_R\ A$$

Check your answer by writing a program and checking its type in OCaml or Haskell.

(6) **(50 points)** Let A and B be base types. Then, it is impossible to give a term of type $((A \to B) \to A) \to A$. Explain why. (hint: Peirce's law)

(7) **(50 points)** It is known that simply-typed λ-calculus is *not* powerful to express Church numerals. Give a concrete example for this.

(8) **(110 points)** Investigate an extension of simply-typed λ-calculus such as System F. Summarize its definition and properties such as progress, subjection reduction, and Curry-Howard correspondence, and give appropriate citations.

(9) **(120 points)** Prove that every well-typed λ-term has a normal form.

IV. **(120 points)** Write a program that computes a sequence $M_1 \longrightarrow_\beta M_2 \longrightarrow_\beta M_3 \longrightarrow_\beta \ldots$ for a given untyped λ-term M_1.

V. **(150 points)** Write a program for *either one* of the following problems about simply-typed λ-calculus.

- Given a term M and a type τ, check whether $\emptyset \vdash M : \tau$ holds or not.
- Given a type τ, find M, if any, such that $\emptyset \vdash M : \tau$.
- Given a term M, find a type τ, if any, such that $\emptyset \vdash M : \tau$.