Name: *Entry:*

COL226: Programming Languages

Sun 16 May 2021 **Major** 2 hours Max marks 120

Instructions:
1. Download the paper.
2. Write your name and entry number in the designated space on top and *do not forget to sign the honour statement below.*
3. Answer the question(s) in the appropriate space provided starting from this page.
4. Scan the paper with your completed answer. *This paper should not take more than 2 hours to answer and you are given upto half-an-hour extra for downloading the paper and uploading your answers.*
5. Upload it on Gradescope 2002-COL226 page within the given time. *Make sure the first page with your name, entry no and signature is also the first page of your uploaded file*
6. Late submissions (within 2 minutes of submission deadline) on the portal will attract a penalty of 10% of the total marks allotted to the paper for each minute of delay and 20% for each minute of delay thereafter.
7. Email submissions after the closing of the portal will not be evaluated (You get a 0).
8. Uploads without the first page details (including signature) may be awarded 0 marks.

I abide by the Honour code that I have signed on my admission to IIT Delhi. I have neither given any help to anybody nor received any help from anybody or any site on the internet in solving the question(s) in this paper.

Signature: **Date:**

1. **[15 marks.]** Solve problem 4 of exercise 4.4. on page 359 in the Hyper-notes(2021-05-04).

2. **[3+((2x5)+2)=15 marks.]**

   (a) What is the distinguishing feature between leftmost-innermost-computation and the leftmost-outermost-computation rules? Explain with the help of the example

   ```
   fun OR (x, y) = if x then true else y
   ```

   (b) Consider the function

   ```
   fun M (x) = if x > 100 then x - 10 else M(M(x+11))
   ```

   i. Show the
   - leftmost-inner-most and
   - leftmost-outermost

   computations for `M(100)`.

   ii. How can the left-most outermost computation be made more efficient?

3. **[10+10=20 marks.]**

   (a) Function composition in mathematics defined by $(f \circ g)(x) = f(g(x))$ is associative. Define a combinator C for function composition and prove that function composition is associative i.e.

   $$(\mathsf{C}\ f\ ((\mathsf{C}\ g)\ h)) =_\beta (\mathsf{C}\ ((\mathsf{C}\ f)\ g)\ h)$$

   (b) However, programmers are more interested in the sequential composition of functions defined as $(f;g)(x) = g(f(x)$. Define another combinator D which captures sequential composition and prove that it is also associative.

4. **[(2x2)+(2x8)=20 marks.]** Church defined the truth values by the combinators $\mathsf{true} \overset{df}{=} \lambda\ x\ y[x]$ and $\mathsf{false} \overset{df}{=} \lambda\ x\ y[y]$ and the *if-then-else operator* by the combinator $\mathsf{ite} \overset{df}{=} \lambda\ x\ y\ z[((x\ y)\ z)]$. Shannon showed that all boolean operations may be represented using only the *if-then-else operator* and the two constants *true* and *false.*

   (a) Use the above combinators to define the boolean operations *and* and *or* as combinators.

(b) Prove that your definitions of *and* and *or* satisfy their respective truth-tables.

5. **[20 marks.]** Give a formal proof (with justification of each step) of the principal type scheme of the combinator $\mathsf{S} \overset{df}{=} \lambda x\ y\ z[((x\ z)\ (y\ z))]$.

6. **[10 marks.]** Assume the following command is added to the WHILE programming language. It is the standard for-loop found in most imperative programming languages.

$$S_0 \ \rightarrow\ \textbf{for}\ id\ :=\ E_1\ \textbf{to}\ E_2\ \textbf{do}\ S_1\ \textbf{endfor}$$

Assume that

- **for**, **to**, **do** and **endfor** are (new) keywords of the language,
- the two occurrences (E_1 and E_2) of the non-terminal E refer to expressions that evaluate to integer values and
- both the occurrences (S_0 and S_1) of the non-terminal symbol S refer to commands in the WHILE langauge.

Define a syntax-directed translation for this statement.

(a) You need to ensure that

- the values of E_1 and E_2 are evaluated initially. Assume they yield integer values i_1 and i_2.
- *id* is initially assigned the value of i_1,
- the for-loop terminates whenever the value of *id* exceeds the value i_2 and
- *id* is incremented by 1 after each execution of the body S_1 if and when the execution of S_1 terminates.

(b) You may use any of the attributes/variables/instructions and notation that are available in the hyper-notes without any further explanation. **Don't change any of them!**

(c) Any new attributes/variables that you introduce need to be defined precisely.

7. **[20 marks.]** The following is the definition of binary trees in Prolog/logic programming.

```
bintree(empty).
bintree(node(N, L, R)):- bintree(L),bintree(R).
```

Define the preorder-traversal of an arbitrary binary tree by a predicate

```
preorder(BT, L)
```

where `BT` is a variable denoting a binary tree and list `L` stands for preorder-traversal of the binary tree.