Indian Institute of Technology Delhi
Department of Computer Science and Engineering

COL226 **Programming Languages** **Homework Assignment**

15 August 2020 24 hours (due: 16 August 2020 at 23:59) Maximum Marks: 90

Instructions:

1. Open book and notes. You may NOT consult with other people when doing this homework, either to help or seek help. If you present code that you found in a book or on the Web, you should acknowledge the source.
2. Submission of the homework will automatically imply you have accepted the honour code in its entirety.
3. It is suggested that you try to work out all the answers in rough before transferring them to the sheets which you will submit.
4. Write in blue or black ink and in the space provided, since we will be using gradescope for correction.
5. Note: If your circumstances are such that you cannot print out the question paper to write on, please answer each question on a clear sheet of paper. Answer each top-level question on a separate sheet of paper, with the question number and subparts clearly labelled. Make sure that your answers are in the same order as the questions, and if you do not attempt a question or part of a question, you should still write down the question number, leaving the answer blank.
6. Write your name, entry number and group at the top of each page.
7. Take a photograph or scan of each sheet in order, and combine them into a **single** PDF file.
8. Name the file "COL226-HW-*YourName-EntryNumber*.pdf". Failure to do so will result in your submission not being graded. If you are submitting answers not written in the space provided, i.e., on separate sheets, please add the word "-SEP" (in capital letters) to the name of your file before the ".pdf"
9. Upload your submission on moodle. If the file size is large, upload a zipped file.

You may look up any tutorial reference or text for understanding how Prolog programs execute. You need to be able to write small programs in Prolog and know how Prolog works, though precise syntactic correctness is not expected in this exercise.

For Q6, use the following user-defined data type definition of binary trees in OCaml:

```
type 'a tree = Leaf | Node of ' a * ('a tree) * ('a tree)
```

UNDERTAKING
I have understood the terms under which this assignment is being conducted, and I will act honestly, responsibly and fairly, and abide by the honour code.

Signature:

Date:

Place:

Q1. (6+4=10 marks) **Function Augmentation.** Let $f, g \in \mathcal{X} \rightarrow_{fin} A$. Define $f[g] : \mathcal{X} \rightarrow_{fin} A$ as

$$f[g](x) = \begin{cases} g(x) & \text{if } x \in dom(g) \\ f(x) & \text{if } x \in dom(f) - dom(g) \\ undef & \text{otherwise} \end{cases}$$

1. Prove from this definition that for any $f, g, h \in \mathcal{X} \rightarrow_{fin} A$, $(f[g])[h] = f[g[h]]$. [Recall that two functions are equal if their outputs on each input are equal.]

2. Prove that if $dom(f) \cap dom(g) = \emptyset$, then $f[g] = f \cup g$, where $f \cup g$ is defined as the union of the two functional relations. [Show first that the rhs is a finite-domain function.]

Q2. (2+8=10 marks) **Big-step Operational Semantics.**

1. Using the rules for $\gamma \vdash e \Longrightarrow_{\mathrm{e}} a$, *draw* a tree-structured *derivation* or *proof tree* for calculating the answer of the expression

$$\textbf{let } x \triangleq 5 \textbf{ in let } y \triangleq 4 \textbf{ in } (x + y) * z \textbf{ end end}$$

given the table $\gamma = \{x \mapsto 7, z \mapsto 3\}$,

2. Assume you are given derivation trees that prove $\gamma_i \vdash e_i \Longrightarrow_{\mathrm{e}} a_i$, for e_i and suitable γ_i, $\quad i \in \{1, 2, 3\}$, which are of the form

$$\begin{array}{c} \backslash \ \ \pi_i \ \ / \\ \gamma_i \vdash e_i \Longrightarrow_{\mathrm{e}} a_i \end{array}$$

Show, by drawing derivation trees, that the following two expressions *always* give the same answer:

$$\textbf{let } x \triangleq e_1 \textbf{ in let } y \triangleq e_2 \textbf{ in } e_3 \textbf{ end end} \quad \text{and} \quad \textbf{let } x \triangleq e_1; \ y \triangleq e_2 \textbf{ in } e_3 \textbf{ end}$$

Q3. (8+12=20 marks) **Operational Equivalence.** Two definitions are *operationally equivalent* if their elaboration results in the same table (or more generally, equivalent tables). Assuming that you have been given derivation trees

$$\begin{array}{c} \backslash \ \ \pi_i \ \ / \\ \gamma_i \vdash d_i \ \ \approx\!> \ \ \gamma_i' \end{array}$$

proving the relations $\gamma_i \vdash d_i \ \approx\!> \ \gamma_i'$, (for suitable choices of γ_i, γ_i'), show the following:

1. $\|$ is commutative, *i.e.*, $(d_1 \| d_2)$ and $(d_2 \| d_1)$ are operationally equivalent.

2. ";" is associative, *i.e.*, $(d_1; d_2); d_3$ and $d_1; (d_2; d_3)$ are operationally equivalent.

Q4. (2+3+3+2=10 marks) **Expressiveness.**

1. Give an example, involving only one variable definition in each d_i, where $(d_1 \| d_2)$ *cannot* be expressed as d_1; d_2 or d_2; d_1 (without introducing any extra variables).

2. Under what conditions can $(d_1 \| d_2)$ be expressed as d_1; d_2 (without introducing an extra variable)? Give an example.

3. Under what conditions is $(d_1 \| d_2)$ equivalent to both d_1; d_2 and also to d_2; d_1? Give an example.

4. OCaml allows definitions of the form $(x_1, \ldots, x_n) \stackrel{\Delta}{=} e$, where the the x_i are distinct variables and $e : \tau_1 * \ldots * \tau_n$. Give an example with $n = 2$ to show what may go wrong if the x_i are not distinct.

Q5. (4+6=10 marks) **Subterms**. Consider terms t_1, t_2 in the Σ-algebra $\mathcal{T}_\Sigma(\mathcal{X})$, for some signature Σ. We say that t_1 is a *subterm* of t_2, if t_1 appears somewhere within t_2, when terms are visualised as trees.

1. Formalize the notion of t_1 being a sub-term of t_2 by providing an *inductively defined relational specification* $subterm(t_2, t_1)$. [Note: you are to write this in mathematical notation, not as an OCaml/Prolog program. Hint: Any term t is trivially a sub-term of itself.]

2. We can refine the notion of sub-term to also indicate the *position* in t_2 at which an occurrence of sub-term t_1 appears by providing the *path* p from the root of t_2 to that occurrence of t_1. A position/path is a list of indices (natural numbers), where each index indicates which "child" of the current node to follow. For example, $[2; 1]$ describes the path starting from the root, going to the node that is the second child of the root, and then going the first child of that node. Not each list of indexes may be a legal path in a given tree. Write a mathematical specification of the relation $subterm(t_2, p, t_1)$, which says that the term t_1 is the subterm of t_2 at position p. [Note: you need to specify this relation for terms in the term-algebra of an arbitrary signature Σ.]

Q6. (5+7+8=20 marks) **From functional to relational programs and back.** Any function f written in OCaml using case analysis (*i.e.*, the `match` construct) can easily turned into a Prolog program p_f by following the simple rules: (a) if f takes k arguments, let p_f take $k+1$ arguments, with the last argument standing for the result; (b) convert each parameter and local variable in OCaml to begin with a Capital letter in Prolog; (c) convert each constructor in OCaml that starts with a Capital letter into a constructor starting with a small letter in Prolog; (d) write each case in a match as a clause in Prolog, with `->` written as `:-`; (e) replace each nested call to $g(e_1, \ldots, e_n)$ in the OCaml program by a fresh Prolog variable Z and conjoin on the right side, a call to predicate $p_g(e'_1, \ldots, e'_n, Z)$ until there are no nested function calls, where the e'_i are transformed versions of the e_i.

1. Write an OCaml program *subtree*$(t2, p)$ for the data type `'a tree`, which returns the subtree at position p in binary tree $t2$ (assuming this position is a valid position; raise an exception otherwise).

2. Now convert the program into a Prolog program `subterm`$(T2, P, T1)$, failing if P is not a valid position in $T2$.

3. Since the above Prolog program defines a predicate, it can express multiple functional programs when run in different input-output modes of the parameters. Considering `subterm`$(T2, P, T1)$ in "in-out-in" mode, write the equivalent OCaml program *positions*$(t2, t1)$ which returns the *list* of all positions in $t2$ at which subtree $t1$ appears. [Note: if it does not appear, return the empty list.]

Q7. (10 marks) **Extending Subject Reduction.** State (**no proof required!**) an extended form of the Subject Reduction theorem to encompass the mutually recursive syntactic categories of expressions *Exp* and definitions *Def*, in terms of the relations $\Gamma \vdash e : \tau$, $\Gamma \vdash d :> \Gamma'$, $\gamma \vdash e \Longrightarrow_{\mathrm{e}} a$, and $\gamma \vdash d \ \approx\!> \ \gamma'$.