* Q1.) part 1.) All cases are covered Why is one mark deducted

Indian Institute of Technology Delhi
Department of Computer Science and Engineering

Programming Languages

COL226 | 60 minutes | Minor II
March 24, 2018 | | Maximum Marks: 60

* Q 2.) Part 3.) Presentation is wrong but the logic is correct. please check

Open notes. Write your name, entry number and group at the top of each sheet in the blanks provided. Answer all questions in the space provided, in blue or black ink (no pencils, no red pens). Budget your time according to the marks. Do rough work on separate sheets.

3/10

Q0. (4+6=10 marks) **Associativity**.

1. Let $\gamma_1, \gamma_2, \gamma_3$ be tables (in $\mathcal{X} \rightharpoonup_{fin}$ *Answer*). By showing that we get the same answer for *any* input variable x, show that table augmentation (indeed function augmentation in general) is *associative*:

3

$\forall x \in dom(\gamma_1 \cup \gamma_2 \cup \gamma_3)$ $\quad \underbrace{\gamma_1[(\gamma_2[\gamma_3])]}_{P_1} = \underbrace{(\gamma_1[\gamma_2])[\gamma_3]}_{P_2}$

if $x \in dom(\gamma_3) \Rightarrow P_1 = \gamma_3(x)$ and $P_2 = \gamma_3(x) \Rightarrow P_1 = P_2$

if $x \notin dom(\gamma_3)$ and $x \in dom(\gamma_2) \Rightarrow P_1 = \gamma_2(x)$ and $\because x \in dom(\gamma_1[\gamma_2]) \therefore P_2 = \gamma_2(x) \Rightarrow P_1 = P_2$

if $x \notin dom(\gamma_3)$ and $x \notin dom(\gamma_2)$ and $x \in dom(\gamma_1) \Rightarrow P_1 = \gamma_1(x)$ and $P_2 = \gamma_1(x) \Rightarrow P_1 = P_2$

~~$\therefore$ in all cases $P_1 = P_2$~~

if $x \in dom(\gamma_3)$ and $x \in dom(\gamma_2) \Rightarrow P_1 = \gamma_3(x)$ (because γ_3 is overlapped)
and ($x \in dom(\gamma_1)$ or $x \notin dom(\gamma_1)$) similarly $P_2 = \gamma_3(x) \Rightarrow P_1 = P_2$

if $x \in dom(\gamma_1)$ and $x \in dom(\gamma_2)$ and $x \notin dom(\gamma_3)$ $\Rightarrow P_1 = (\gamma_2[\gamma_3])(x) = \gamma_2(x)$ because $x \notin dom(\gamma_3)$
similarly $P_2 = (\gamma_1[\gamma_2])(x) = \gamma_2(x)$

$\therefore$ in all cases $P_1 = P_2$ Show domains are equal.

2. Recall that two definitions are operationally equivalence if both elaborate to the same or equivalent tables. Show that *sequential composition* of definitions is *associative*:

$$\text{For all } d_1, d_2, d_3 \in \textit{Defs}: \quad d_1; (d_2; d_3) \approx_d (d_1; d_2); d_3$$

[Hint: Assume you have shown $\gamma' \vdash d_i \Longrightarrow \gamma_i$ for appropriate choice of γ' for each of the d_i. Using the Big-step inference rules, present the corresponding proofs as tree-shaped diagrammatic proofs]

0

Name: Siddhant Shingi Entry: 2016CS10370 Gp:

6/10

Q1. (2+3+2+3=10 marks) **Patterns: Syntax and Type-checking.** OCaml permits the left hand sides of definitions to be tuple patterns instead of merely variables (in fact, they can be even more complex patterns). For example, one can write

```
let (x,y) = (e1,e2);;
```

which binds variable x to the answer for e1 and y to the answer for e2. A slightly generalized form of simple definitions is now $p \stackrel{\Delta}{=} e$, where

$$p \in Pat ::= x \mid (x_1, \ldots, x_n)$$

Correspondingly, functions can be defined on tuples of arguments, so expression abstractions can be generalised to the form $\lambda p.e$.

1. Extend the function dv for the new definition form (mention any conditions):

1

$$dv(\,(x_1,\ldots,x_n) \stackrel{\Delta}{=} e\,) = \{x_1, x_2, \ldots, x_n\}$$

2. Complete the static semantic (*typing*) rule for the new kind of definition, assuming the variables in the patterns are all distinct.

0

$$\Gamma \vdash e : \tau_1 \times \tau_2 \cdots \tau_n \quad \frac{\Gamma \vdash x_1 \stackrel{\Delta}{=} e \Rightarrow \{x_1 : \tau_1\} \quad \Gamma \vdash x_2 \stackrel{\Delta}{=} e \Rightarrow \{x_2 : \tau_2\} \ \ldots \ \Gamma \vdash x_n \stackrel{\Delta}{=} e \Rightarrow \{x_n : \tau_n\}}{\Gamma \vdash (x_1,\ldots,x_n) \stackrel{\Delta}{=} e \triangleright \{x_1 : \tau_1\} \cup \{x_2 : \tau_2\} \cup \cdots \cup \{x_n : \tau_n\}}$$

3. Modify the function fv that returns the set of free variables (those variables which have an occurrence which is not bound) for the generalised abstraction form:

2

$$fv(\lambda(x_1,\ldots,x_n).e) = fv(e) - \{x_1, x_2, \ldots, x_n\}$$

4. Complete the static semantic (*typing*) rule for the generalised abstraction, assuming the variables in the patterns are all distinct.

3

$$\frac{\Gamma \vdash x_1 : \tau_1, \Gamma \vdash x_2 : \tau_2 \ \cdots \ , \Gamma \vdash x_n : \tau_n \quad \Gamma[\{x_1 : \tau_1\} \cup \{x_2 : \tau_2\} \ldots \cup \{x_n : \tau_n\}] \vdash e_1 : \tau}{\Gamma \vdash \lambda(x_1,\ldots,x_n).e_1 : \tau_1 \times \tau_2 \times \cdots \tau_n \to \tau}$$

Q2. (4+3+3+3+3=16 marks) **Patterns: Operational Rules.**

1. Specify *pattern-matching*, a special case of unification, between a given pattern $p \in Pat$ and an answer a (hopefully a tuple). The operation $pm(p, a)$ yields a (unifying) substitution if one exists, by case analysis on p and a:
[You do NOT need to write OCaml code]

5/16

p	a	unifier.
~~$(x_1, x_2, \ldots, x_n)$~~	~~$(a_1, x_2, \ldots, x_n)$~~	~~id~~
$(x_1, x_2, \ldots, x_n)$	$(a_1, a_2, \ldots a_n)$	$\boxed{X \mapsto A}$ $\{x_i \to a_i\}_{i=1}^{n}$
$(c_1, c_2, \ldots, c_n)$	$(a_1, a_2, \ldots, a_m)$	FAIL ⇒ two constants cannot be mapped to each other.
$(a_1, a_2, \ldots, a_n)$	$(a_1, a_2, \ldots, a_n)$	id.
$(e_1, e_2, \ldots, e_n)$	$(a_1, a_2, \ldots, a_n)$	$\sigma_1 = mgu(e_1, a_1)$ $\sigma_2 = mgu(\sigma_1(e_2), \sigma_1(a_2))$ $\vdots$ $\sigma_n = mgu(\sigma_1 \ldots \sigma_{n-1}(e_n), \sigma_1 \ldots \sigma_{n-1}(a_n))$ then mgu will be $\sigma_1; \sigma_2; \ldots; \sigma_n$ if does not exists then FAIL.

where e_i is of type $f(t_1, t_2, \ldots t_k)$
operation in exp

2. Provide Big-step (Natural) semantics for the generalized definition of [Q1]. You may use $pm(p, a)$.

$$\frac{\gamma \vdash e \Rightarrow a}{\gamma \vdash p \triangleq e \Rightarrow \{p \mapsto a\}} \quad 1$$

3. Using the Principle of Correspondence, provide a compatible version of the Big-step (Natural) semantics for function call where the abstractions may be of the generalised form $\lambda p.e_1'$:

$$\frac{\langle\langle \gamma, e_1 \rangle\rangle \Rightarrow_n \langle\langle \gamma', \lambda p.e_1' \rangle\rangle \quad \langle\langle \gamma'[p \mapsto \langle\langle \gamma, e_2 \rangle\rangle], e_1' \rangle\rangle \Rightarrow a}{\gamma \vdash (e_1\ e_2) \Longrightarrow a}$$

×

4. How can one encode the general projection operator $\mathbf{proj}_i^n$, where $(1 \le i \le n)$ for projecting the i^{th} component of an n-tuple, using the generalized form of definition or abstraction?

3

~~using abstraction we can find the pass the tuple i.e $\lambda p.e_1$~~
~~and here $e_1 \Rightarrow x_i$ where x_i is ith~~

5. How can one encode parallel definition $x_1 \triangleq e_1 \| x_2 \triangleq e_2$ using the generalised form of definition?

$$\frac{\Gamma \vdash x_1 \triangleq e_1 \Rightarrow \{x_1 \Rightarrow e_1\} \quad \Gamma \vdash x_2 \triangleq e_2 \Rightarrow \{x_2 \Rightarrow e_2\}}{\Gamma \vdash x_1 \triangleq e_1 \| x_2 \triangleq e_2 \Rightarrow \{x_1 \Rightarrow e_1\}[\{x_2 \Rightarrow e_2\}]}$$

×

Q3. (3+5=8 marks) **SECD Machine.**

8/8

1. Consider new op-codes for the SECD machine: TUPLE(n) for *n-tuple expressions* $(e_1, \ldots, e_n)$ where $n \ge 2$, and PROJ(i, n) for *projection expressions* $\mathbf{proj}_i^{(n)}\ e$ where $(1 \le i \le n)$ for projecting the i^{th} component of a n-tuple. Now present rules for *compiling only the new expressions* into op-code sequences.

compile$((e_1, \ldots, e_n))$ = compile(e_1) @ compile(e_2) .. @ compile(e_n) @ [TUPLE(n)]

compile$(\mathbf{proj}_i^{(n)}\ e)$ = compile(e) @ [PROJ(i,n)]

2. Finally present the new SECD machine *execution rules for only these two new op-codes.*

$$\left\langle\left\langle \begin{array}{|c|} a_n \\ \vdots \\ a_1 \\ \hline S \end{array}, \text{TUPPLE}(n) :: C, \boxed{\gamma}, \boxed{D} \right\rangle\right\rangle \Longrightarrow \left\langle\left\langle \begin{array}{|c|} (a_1, \cdots, a_n) \\ \hline S \end{array}, C, \boxed{\gamma}, \boxed{D} \right\rangle\right\rangle$$

$$\left\langle\left\langle \begin{array}{|c|} (a_1, \cdots, a_n) \\ \hline S \end{array}, \text{PROJ}(i,n) :: C, \boxed{\gamma}, \boxed{D} \right\rangle\right\rangle \Longrightarrow \left\langle\left\langle \begin{array}{|c|} a_i \\ \hline S \end{array}, C, \boxed{\gamma}, \boxed{D} \right\rangle\right\rangle$$

Q4. (16 marks) **Subject reduction.** This theorem shows that in strongly- and statically-typed languages, calculation results in an answer whose type is unchanged, and hence type checking/inference done at

$\lambda p.e_1$

$p \Rightarrow$ ~~tuple of~~ (tuple, i)

$e_1 \Rightarrow$ ith element of the tuple

i.e e_1 ~~conta~~ returns only the ith element of the tuple.
rest all the variables are not bound in $\lambda p.e_1$

×

Name: Siddhant Shirgi   Entry: 2016CS10510 Gp: 1

10.5 / 16

compile time guarantees freedom from type errors at run time. You are to prove the theorem for the following language (*only the following and no other expressions are to be considered in this exam*):

$$e \in Exp ::= x \mid (e_1, \ldots, e_n) \mid \mathbf{proj}_i^{(n)}\, e$$

A table γ is *type-faithful* to type assumption Γ if $dom(\gamma) \subseteq dom(\Gamma)$ and for all $x \in dom(\gamma)$: $\Gamma \vdash \gamma(x) : \Gamma(x)$.

Prove by induction that for all $e \in Exp$, for all Γ, γ where γ is type-faithful to Γ, for all types τ and all answers a: *if* $\Gamma \vdash e : \tau$ *and* $\gamma \vdash e \Longrightarrow a$, *then* $\Gamma \vdash a : \tau$

Base cases:

when $e = x$. i.e variables.

$\therefore$ if $\Gamma \vdash x : \Gamma(x)$

and $\gamma \vdash x \Rightarrow \gamma(x)$

then $\Gamma \vdash \gamma(x) : \Gamma(x)$ 4

(*) hence by induction the subject Representation is proved.

Induction Hypothesis:

for ~~any~~ all $e \in Exp$ of height h, for all Γ, γ, for all a, τ

γ is type faithful to type assumption Γ

if $\Gamma \vdash e : \tau$ and $\gamma \vdash e \Rightarrow a$ then $\Gamma \vdash a : \tau$

2.5

Induction Cases:

let e be an exp with height $= n+1$

if e is of the form $(e_1, \ldots, e_n)$

where $\text{height}(e_1, e_2, \ldots, e_n) \le n$

$\therefore \Gamma \vdash e : \tau_1 \times \tau_2 \times \ldots \tau_n$ why?

if $\Gamma \vdash e_1 : \tau_1,\ \Gamma \vdash e_2 : \tau_2 \ldots,\ \Gamma \vdash e_n : \tau_n$

$\gamma \vdash e \Rightarrow (a_1, a_2, \ldots, a_n)$ why?

if $\gamma \vdash e_1 \Rightarrow a_1,\ \gamma \vdash e_2 \Rightarrow a_2, \ldots, \gamma \vdash e_n \Rightarrow a_n$

3.5

$\therefore \Gamma \vdash (a_1, a_2, \ldots, a_n) : \tau_1 \times \tau_2 \times \ldots \times \tau_n$ why?

if $\Gamma \vdash a_1 \Rightarrow \tau_1,\ \Gamma \vdash a_2 \Rightarrow \tau_2 \ldots \Gamma \vdash a_n \Rightarrow \tau_n$ is valid. [Logical presentation, please]

and using IH also follows

so $\Gamma \vdash (a_1, \ldots a_n) : \tau_1 \times \tau_2 \ldots \times \tau_n$

~~the~~ IH.

if e is of form $\text{Proj}_i^{(n)} e_1$ where $\text{height}(e_1) \le n$

$\therefore \Gamma \vdash e : \tau_i$

if $\Gamma \vdash e_1 : \tau_1, \ldots \Gamma \vdash e_i : \tau_i$ X   0.5

(*) $\gamma \vdash e \Rightarrow a_i$

if $\gamma \vdash e_1 \Rightarrow a_1$ X $\ldots \gamma \vdash e_i \Rightarrow a_i$

$\therefore \Gamma \vdash a_i : \tau_i$ if $\Gamma \vdash a_1 : \tau_1 \ldots \Gamma \vdash a_i : \tau_i$

using IH.