Indian Institute of Technology Delhi
Department of Computer Science and Engineering

COL226 **Programming Languages** **Take Home Major Exam**

30 August 2020 48 hours (due: 2 September 2020 at 12:00 Noon) Maximum Marks: 100

Instructions:

1. Open book and notes. You may NOT consult with other people when doing this exam, either to help or seek help. If you present material that you found in a book or on the Web, you should acknowledge the source. (This is a legitimate use of resources, but should be documented.)
2. You need to sign (with date and place indicated) and submit the undertaking on this page. Submission of the exam will imply you have accepted the honour code of IITD in its entirety.
3. You have adequate time to arrange for a printout of the question-answer paper, and to scan and upload it. No excuses will be entertained for late submission even by a few minutes.
4. It is suggested that you take a spare printout of the question paper (one-sided printing). It is also suggested that you try to work out all the answers in rough before transferring them to the sheets which you will submit.
5. Write in blue or black ink and in the space provided, since we will be using online systems and automated tools for correction. Otherwise the exam will not be graded.
6. Note: If your circumstances are such that you do not have access to a printer for a valid reason (you are in a containment zone, you are quarantined, floods, etc.) *and* there are no means (e.g., a neighbour or shop in the neighbourhood, etc.) to help you get a printout, *and* you therefore cannot print out the question paper to write on, you need to send me a prior email request with your current address and a clear and substantiated justification. This is NOT a general option available by default to all students. You need prior approval to submit in this manner. Your exam paper will not be graded otherwise.
7. If you are indeed in such circumstances, here are the instructions: 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.
8. Write your name, entry number and group at the top of each page.
9. Take a photograph or scan of each sheet in order, and combine them into a **single** PDF file.
10. Name the file "COL226-Major-*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"
11. Upload your submission on moodle. If the file size is large, upload a zipped file.

You may look up any tutorial reference or text to get started. However, your answers should conform to the notation and conventions used in this course.

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: Brand

Date: Palm

Place: Mat

Q1. (3+2+4+3+2+6=20 marks) **Pattern matching in definitions.** OCaml permits the left hand sides of definitions to be tuple patterns instead of merely variables. For example, one can write

```
let (x, (y, z) ) =  (3*17, (not true, 7+2) );
```

which binds variable `x` to the answer `51`, `y` to the answer `false` and `z` to the answer `9`. The generalised form of simple definitions is now $p \triangleq e$, where

$$p \in Pat ::= x \mid (p_1, \ldots, p_n)$$

All the variables appearing in a pattern are required to be distinct. Assume the function *vars* returns the set of variables in a given pattern or expression.

1. Formalise in mathematical notation the condition that all the variables in a pattern p are distinct (i.e., there are no repetitions).

$$varsdistinct(p) = \begin{cases} \text{true} & \text{if } p \equiv x \quad (0.5) \\ vars(p_i) \cap vars(p_j) = \emptyset \ (1 \le i \ne j \le n) & \text{if } p = (p_1, \ldots p_n) \quad (1.5) \\ \wedge (\forall i : 1 \le i \le n)\ varsdistinct(p_i) & (0.5) \end{cases} \quad (0.5) \quad \} 3$$

2. Extend the function $dv(d)$ which returns the *variables defined* in definition d, to include this generalised definition form (in mathematical notation):

$dv(x \triangleq e) = \{x\}$ (0.5)

$dv((p_1, \ldots, p_n) \triangleq e) = \bigcup_{i=1}^{n} vars(p_i)$ (1.5) } 2

3. Complete the static semantic (***typing***) rule for this new kind of definition, assuming the variables in the patterns are all distinct. [Hint: Define a function $pat\_typ(p, \tau)$, inductive on the structure of p, that associates variables in pattern p with the corresponding component type of τ.]

$pat\_typ(x, \tau) = \{x : \tau\}$

$pat\_typ((p_1, \ldots, p_n), (\tau_1 \times \ldots \times \tau_n)) = \bigcup_{i=1}^{n} pat\_typ(p_i, \tau_i)$ (1.5)

$pat\_typ(-,-)$ undefined otherwise (0.5) } 4

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash p \triangleq e :> pat\_typ(p, \tau)} \quad (2)$$

4. Provide Big-step (Natural) dynamic semantics (calculation rules) for this general form of definition. [Hint: use unification.]

$$\frac{\gamma \vdash e \Rightarrow a \quad (1) \qquad \sigma = mgu(p, a) \quad (1)}{\gamma \vdash p \triangleq e \Rightarrow\!\!> \sigma \quad (1)} \quad \} 3$$

5. How can one encode the projection operator $\mathbf{proj}_i^{(n)}$ using this general form of pattern-matched definitions?

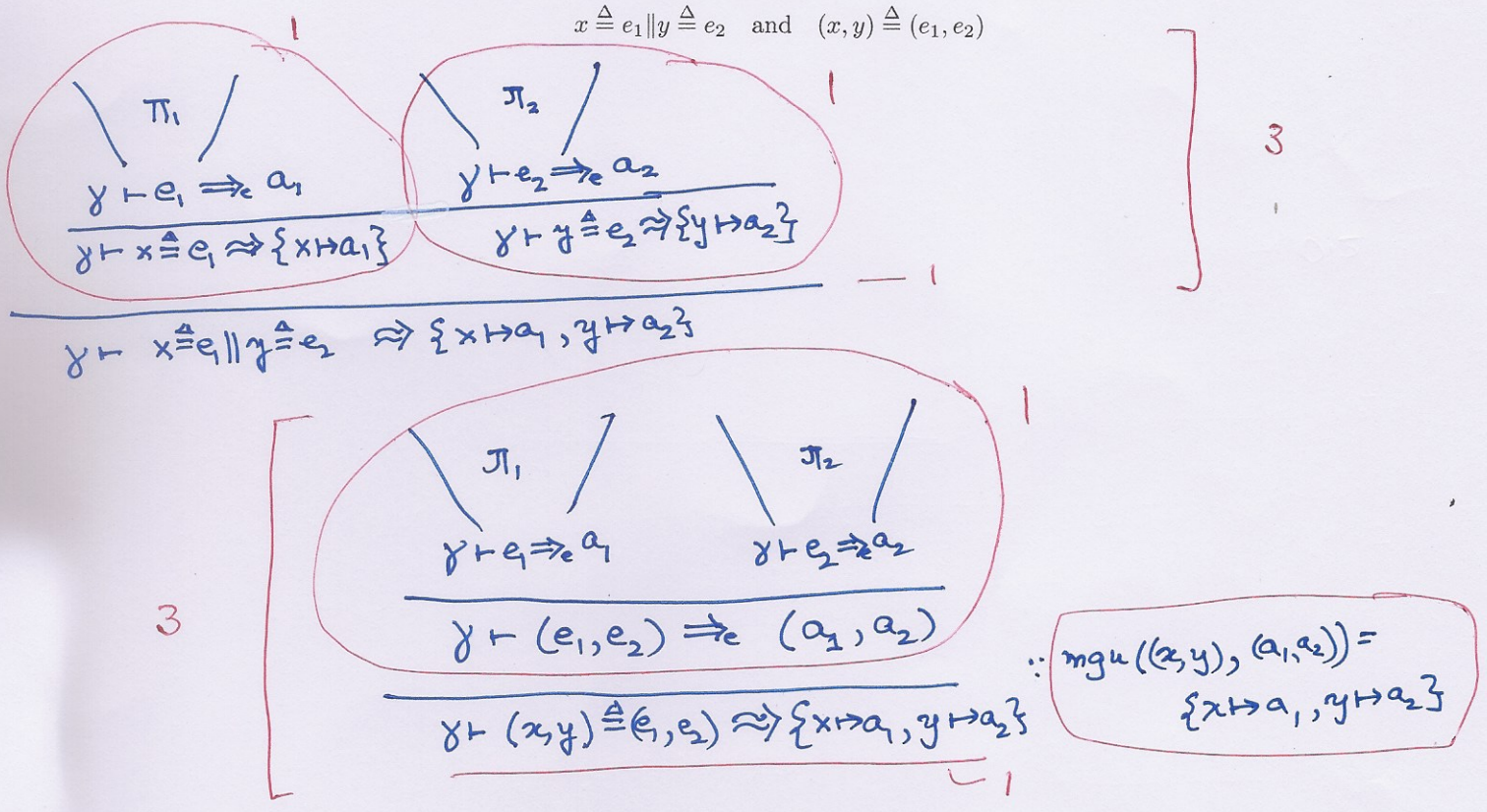
$proj_i^{(n)}$ e encoded let $(x_1, \ldots x_n) = e$ in x_i end] 2

6. Assume you are given derivation trees that prove $\gamma \vdash e_i \Longrightarrow_e a_i$, for given γ, e_i, a_i $\quad i \in \{1,2\}$. Suppose these trees are of the form

$$\frac{\backslash\ \pi_i\ /}{\gamma_i \vdash e_i \Longrightarrow_e a_i}$$

Show, by drawing derivation trees using the big-step rules, that the following two definitions *always* elaborate to the same table:

$$x \triangleq e_1 \| y \triangleq e_2 \quad \text{and} \quad (x,y) \triangleq (e_1, e_2)$$



Q2. (2+3+4+3+3+5=20 marks) **First-class Definitions.**

Consider treating the syntactic category *Def* as first-class expressions in the language. This extension introduces constructs of the form $\{\!|\ d\ |\!\}$, called records or *structures*. If they are first-class, structures can be named using the usual definition mechanism. Components of a structure are accessed using the dot notation, e.g., $x.y$ denotes the field/component y of the structure named x. Structures can have components which are themselves structures.

1. Extend the syntax of expressions *Exp* to include structures and their components.

$e \in Exp ::= n \mid x \mid \ldots \mid \{\!|\ d\ |\!\} \mid e.x$ — 1 — 1 — 2

$d \in Def ::= \ldots$

2. Modify the syntax of definitions *Def* to include structure definitions.

$d \in Def ::= \cancel{x \triangleq e} \mid \ldots \mid i \triangleq e$ — 1

$i \in Names ::= x \mid i.x$ — 2 — 3

3. Can the field of a structure, e.g., $x.y$ to be on the left-side of a definition? If yes, then what are the scoping issues and implementation issues to consider? If no, then why not?

4 { 3 [YES, it can.
Scoping: need to be able to support temporary changes to fields of a structure, and to be able to determine that two structure expressions refer to the same/different structures
1 [Implementation: Need to implement a stacking discipline on structure fields, to support temporary changes

4. Since we wish structures and their fields to follow a strong typing discipline, we need to provide structures with a notion corresponding to a type (let's call this a *sig*). Give a mathematical characterisation of a *sig* for a structure, written as $\{\|\_\|\}$

2 [$\Gamma \in \text{TypAssmpn} \stackrel{\text{def}}{=} \text{Names} \to_{\text{fin}} \text{Type}$
1 [sigs are characterised as $\{\|\ \Gamma\ \|\}$ } 3

5. Provide a type-checking rule for structures and their components (clearly indicate any conditions that must hold):

$$\frac{\Gamma \vdash d \Rightarrow \Gamma_1}{\Gamma \vdash \{\!|\ d\ |\!\} : \{\|\ \Gamma_1\ \|\}} \quad 1.5$$

$$\frac{\Gamma \vdash e : \{\|\ \Gamma_1\ \|\}}{\Gamma \vdash e.x : \Gamma_1(x)} \quad -1.5 \qquad \Big] 3$$

6. Provide big-step operational semantic rules for structures and fields assuming an eager (call-by-value) semantics (clearly indicate any conditions that need to hold.)

$a \in \text{Ans} ::= \ldots \mid \{\!|\ \gamma\ |\!\}$ where $\gamma \in \text{Table}$ −1

$$\frac{\gamma \vdash d \Rightarrow \gamma_1}{\gamma \vdash \{\!|\ d\ |\!\} \Longrightarrow_e \{\!|\ \gamma_1\ |\!\}} \Big] 2$$

$$\frac{\gamma \vdash e \Rightarrow_e \{\!|\ \gamma_1\ |\!\}}{\gamma \vdash e.x \Longrightarrow_e \gamma_1(x)} \Big] 2 \qquad \Big\} 5$$

Q3. (2+2+2+4+6=16 marks) **Principle of Abstraction.** Let us now employ the Principle of Abstraction on structures. A parameterised structure is of the form $\lambda p.m$, where m is a structure and p is a pattern. For simplicity, you may consider only patterns that are single variables. Suppose parameterised structures are first-class elements in the language.

1. Extend the syntax of expressions to admit parameterised structures:

$e ::= \ldots \mid \lambda p.e \mid e_1\ e_2 \mid \{\!|\ d\ |\!\} \mid e.x$

$\lambda p.e$: 1 ; $e_1\ e_2$: 1 ; 2

$\{\!|\ d\ |\!\} \mid e.x$: from prev. question

2. How should the syntax of definitions in *Def* be modified?

NO EXTENSION OR CHANGE —— (2)
$i \triangleq e$ already there

3. Suppose we allowed structures to be parameters of such abstract structures, then what ~~is the~~ changes are required in the type system, i.e., in the characterisation of *Type*?

$\tau \in Type ::= \ldots\ldots \mid \{\| \Gamma \|\}$ —— 2

$\Gamma \in TypAssumptn \stackrel{def}{=} Names \rightarrow_{fin} Type$ (earlier question)

4. Provide type-checking rules for parameterised structures and their instantiation:

NO CHANGE! from functions.

$$\frac{\Gamma[x:\tau_1] \vdash m : \tau_2}{\Gamma \vdash \lambda x.m : \tau_1 \rightarrow \tau_2} \quad 2$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \quad 2$$

4

5. Provide big-step operational semantic rules for creating and instantiating parameterised structures, assuming an eager (call-by-value) semantics:

$$\overline{\langle\langle \gamma, \lambda p.e \rangle\rangle \Rightarrow_e \langle\langle \gamma, \lambda p.e \rangle\rangle}$$ — NO CHANGE except λp. —— 1

[FROM Q1]

$$\frac{\langle\langle \gamma, e_1 \rangle\rangle \Rightarrow_e \langle\langle \gamma_1, \lambda p.e_1' \rangle\rangle \quad (1) \qquad \langle\langle \gamma, e_2 \rangle\rangle \Rightarrow_e a_2 \quad (0.5) \qquad \sigma = mgu(p, a_2) \quad (1) \qquad \langle\langle \gamma_1[\sigma], e_1' \rangle\rangle \Rightarrow_e a \quad (2)}{\langle\langle \gamma, e_1\ e_2 \rangle\rangle \Rightarrow_e a \quad (0.5)}$$

5

6

Q4. (10+10+2=22 marks) **Closures.** A closure is a data structure used to avoid an expensive operation of syntactic substitution in an expression, by looking up a table instead. A closure "really" represents an expression, which we obtain by recursively replacing each variable in the closure's expression by the expression (answer) to which that variable is mapped in the closure's table. Recall that *Tables* and *Clos*ures are mutually recursively defined. In the call-by-name semantics, *Table*s map variables to closures, and closures are pairs consisting of a table and an expression:

$$Table \stackrel{def}{=} \mathcal{X} \rightarrow_{fin} Clos \qquad Clos \stackrel{def}{=} Table \times Exp$$

Consider a minimal functional language of expressions:

$$e \in Exp ::= x \mid n \mid e_1 + e_2 \mid \lambda x.e_1 \mid (e_1 \sqcup\!\sqcup\, e_2)$$

where $(e_1 \sqcup\!\sqcup\, e_2)$ denotes applying function represented by expression e_1 to the argument represented by e_2.

1. Define a function *unroll* : *Clos* → *Exp*, which recursively unrolls a closure to yield the corresponding expression.

10

$$\begin{cases} unroll(\ll \gamma, n \gg) = n & -1 \\ unroll(\ll \gamma, x \gg) = \begin{cases} unroll(\gamma(x)) & \text{if } x \in dom(\gamma) \\ x & \text{if } x \notin dom(\gamma) \end{cases} & -3 \\ unroll(\ll \gamma, e_1 + e_2 \gg) = unroll(\ll \gamma, e_1 \gg) + unroll(\ll \gamma, e_2 \gg) & -1 \\ unroll(\ll \gamma, \lambda x.e \gg) = \lambda x.\ unroll(\ll \gamma \setminus \{x\}, e \gg) & -3 \\ unroll(\ll \gamma, e_1 \sqcup e_2 \gg) = unroll(\ll \gamma, e_1 \gg) \sqcup unroll(\ll \gamma, e_2 \gg) & -2 \end{cases}$$

2. Since a closure really represents an expression, we now wish to adapt the "has type" relation $\Gamma \vdash e : \tau$ to cover closures as well, i.e., allow cases for $\Gamma \vdash \ll \gamma, e \gg : \tau$. Complete the rules for the following closures, clearly indicating any special conditions:

10

$$\begin{cases} 1 - \dfrac{}{\Gamma \vdash \ll \gamma, n \gg : int} \\ 1 - \dfrac{\Gamma \vdash \ll \gamma, e_1 \gg : int \quad \Gamma \vdash \ll \gamma, e_2 \gg : int}{\Gamma \vdash \ll \gamma, e_1 + e_2 \gg : int} \\ 2 - \dfrac{\Gamma \vdash \gamma(x) : \tau}{\Gamma \vdash \ll \gamma, x \gg : \tau} \text{ if } x \in dom(\gamma) \quad \left[\dfrac{x \notin dom(\gamma)}{\Gamma \vdash \ll \gamma, x \gg : \Gamma(x)} \right] \\ 3 - \dfrac{\Gamma[x : \tau_1] \vdash \ll \gamma, e \gg : \tau_2}{\Gamma \vdash \ll \gamma, \lambda x.e \gg : \tau_1 \to \tau_2} \quad x \notin dom(\gamma) \\ 2 - \dfrac{\Gamma \vdash \ll \gamma, e_1 \gg : \tau_1 \to \tau_2 \quad \Gamma \vdash \ll \gamma, e_2 \gg : \tau_1}{\Gamma \vdash \ll \gamma, (e_1 \sqcup e_2) \gg : \tau_2} \end{cases}$$

3. Will these typing rules change if we switch to a call-by-name (lazy) evaluation strategy instead of an call-by-value (eager) strategy? If yes, how; if no, why not?

NO, THEY DO NOT CHANGE

BECAUSE TYPE-CHECKING IS INDEPENDENT OF EVALUATION STRATEGY IN STRONGLY, STATICALLY TYPED LANGUAGES.

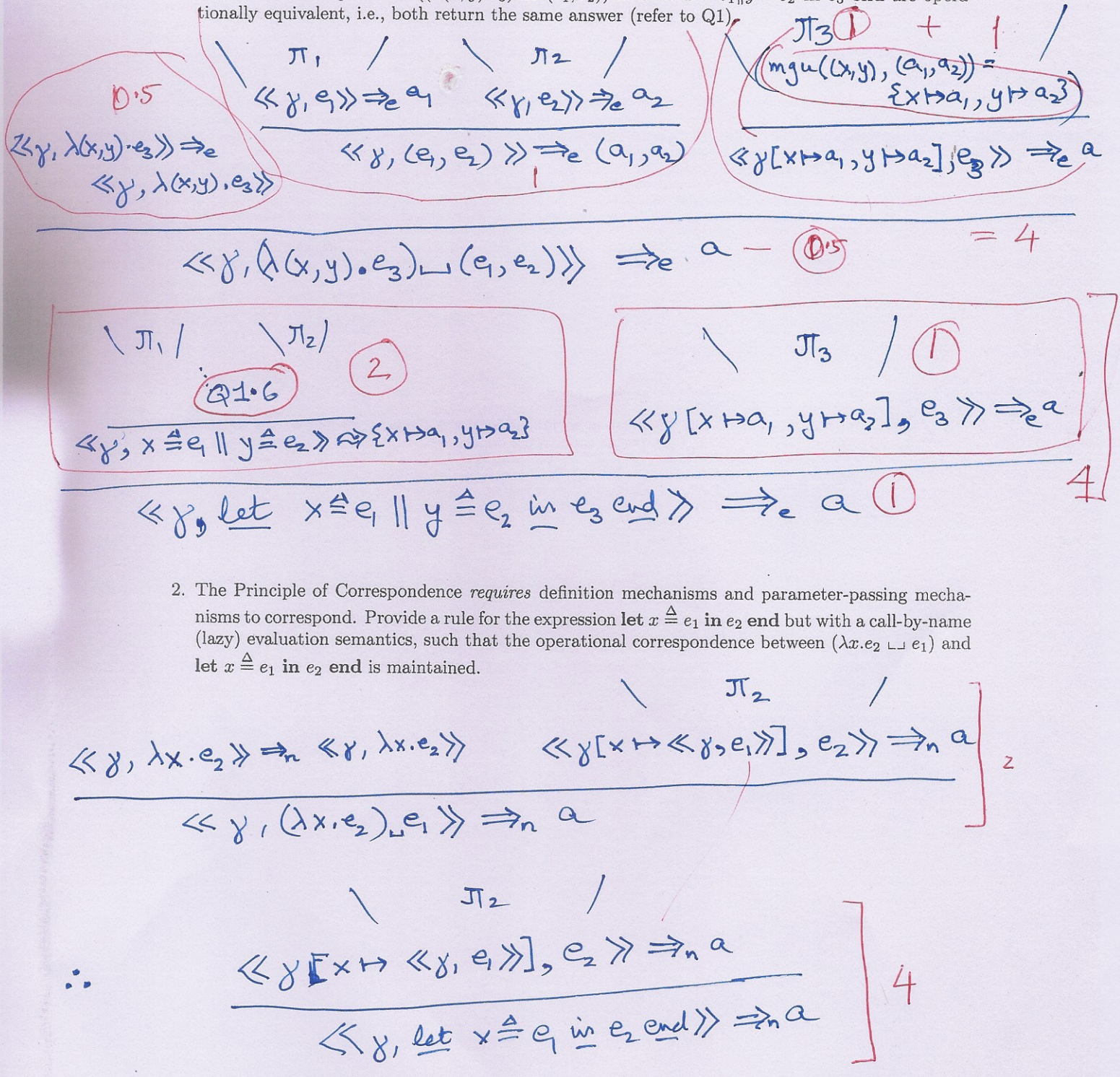
2 –

Q5. (8+6=14 marks) **Principle of Correspondence.**

In the following, we will use the closure-style presentation of the expression calculation relation, e.g., writing $\ll \gamma, e \gg \Longrightarrow_e a$ instead of $\gamma \vdash e \Longrightarrow_e a$.

1. Suppose we are given derivation trees π_1, π_2 and π_3 that prove the transitions $\ll \gamma, e_1 \gg \Longrightarrow_e a_1$, $\ll \gamma, e_2 \gg \Longrightarrow_e a_2$ and $\ll \gamma[x \mapsto a_1, y \mapsto a_2], e_3 \gg \Longrightarrow_e a$ respectively.

   Show that the expressions $((\lambda(x,y).e_3) \llcorner\!\lrcorner (e_1, e_2))$ and **let** $x \triangleq e_1 \| y \triangleq e_2$ **in** e_3 **end** are operationally equivalent, i.e., both return the same answer (refer to Q1).



2. The Principle of Correspondence *requires* definition mechanisms and parameter-passing mechanisms to correspond. Provide a rule for the expression **let** $x \triangleq e_1$ **in** e_2 **end** but with a call-by-name (lazy) evaluation semantics, such that the operational correspondence between $(\lambda x.e_2 \llcorner\!\lrcorner e_1)$ and **let** $x \triangleq e_1$ **in** e_2 **end** is maintained.

Q6. (6+2=8 marks) **Extending Subject Reduction.**

1. State (**no proof required!**) a modified form of the Subject Reduction theorem so that it covers the closure-style expression calculation relation $\ll \gamma, e \gg \Longrightarrow_e a$.

Γ, γ agree if for all $x \in dom(\gamma)$ $\Gamma \vdash \gamma(x) : \Gamma(x)$ (AS BEFORE). LET Γ, γ agree. 1

SUPPOSE $e' = unroll(\ll \gamma, e \gg)$ 2 and $\Gamma \vdash e' : \tau$ 1

IF $\ll \gamma, e \gg \Longrightarrow_e a$, then $\Gamma \vdash a : \tau$ 1

(if a is a closure, use the relation defined in Q4.2).

6

2. Will the Subject Reduction Theorem still hold if in the statement above, $\ll \gamma, e \gg \Longrightarrow_e a$ is replaced by $\ll \gamma, e \gg \Longrightarrow_n a$? If yes, why; if no, why not?

2 NO, IT DOES NOT CHANGE, SINCE EVALUATION/CALCULATION STRATEGY IS INDEPENDENT OF TYPING