University of Massachusetts Lowell — Comp 3010: Organization of Programming Languages

Comp 3010 Mock Final Exam

Dec 06, 2024

---

1. Which *one* of the following expressions is equal to the beta reduction of $(\lambda w.\, \lambda a.\, a\; w)\; (\lambda w.\, a\; w)$?

a) $\lambda a.\, a\; (\lambda w.\, a\; w)$

⇒ b) $\lambda b.\, b\; (\lambda w.\, a\; w)$

c) $\lambda a.\, a\; w$

d) $\lambda w.\, a\; w$

> **Answer:** *You have to alpha-convert $\lambda a.\, a\; w$ to $\lambda b.\, b\; w$ and then carry out the substitution for β-reduction.*

2. Consider the small-step evaluation of the following lambda calculus term under call-by-value and call-by-name semantics. (See Appendix A.)

$$(\lambda x.\, \lambda y.\, x\; x)\; ((\lambda z.\, z\; z)\; (\lambda w.\, w))$$

Which one of the following statements is true?

a) Neither CBV nor CBN evaluation terminate.

b) Both CBV and CBN evaluations terminate and take the same number of steps.

c) Both CBV and CBN evaluations terminate but CBN evaluation takes more steps.

⇒ d) Both CBV and CBN evaluations terminate but CBV evaluation takes more steps.

> **Answer:** *In CBV evaluation, we first reduce the argument $((\lambda z.\, z\; z)\; (\lambda w.\, w))$ to a value and then perform β-reduction whereas in CBN evaluation strategy, we directly proceed to β-reduction.*

3. Consider the following inference rules, which define sets S and T of strings (where $\cdot$ is string concatenation).

$$\frac{}{\text{“a”} \in S} \qquad \frac{s \in S}{\text{“a”} \cdot s \cdot \text{“b”} \in S} \qquad \frac{s \in S \quad t \in T}{s \cdot t \in S} \qquad \frac{}{\text{“ba”} \in T} \qquad \frac{t \in T}{\text{“b”} \cdot t \cdot \text{“a”} \in T}$$

Which *one* of the following properties holds of all strings $s \in S$?

a) The length of s is even.

⇒ b) The number of “a” characters in s is one more than the number of “b” characters in s.

c) String s is of the form “a . . . ab . . . b” (i.e., some number of “a”s followed by some number of “b”s).

d) String s is of the form “a . . . ab . . . ba . . . a” (i.e., some number of “a”s followed by some number of “b”s followed by some number of “a”s).

Answer: *The element s always terminates with "a"; so the resulting string has one "a" more than "b".*

4. Consider the term

$$(\lambda x.x)((\lambda y.y)\ 5)$$

Recall that a redex is a sub-term that will be evaluated immediately. What is the redex in the above term? Select all that apply.

⇒ a) $((\lambda y.y)\ 5)$ under call-by-value evaluation.

⇒ b) $(\lambda x.x)((\lambda y.y)\ 5)$ under call-by-name evaluation.

c) $((\lambda y.y)\ 5)$ under both call-by-name and call-by-value evaluation.

d) There is no redex.

5. Consider

$$\Omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$$

Recall that a term said to be in a normal form if it cannot be evaluated further. Which of the following is true.

a) Ω has a normal form under call-by-name evaluation.

b) Ω has a normal form under call-by-value evaluation.

c) Ω has a normal form under both call-by-name and call-by-value evaluation.

⇒ d) Ω does not have a normal form under either call-by-name or call-by-value evaluation.

6. Consider a large-step **call-by-name** semantics for the lambda calculus. Which *one* of the following is an appropriate rule for application?

a) $$\frac{e_2 \Downarrow v' \qquad e_1\{v'/x\} \Downarrow v}{(\lambda x.\, e_1)\; e_2 \Downarrow v}$$

b) $$\frac{e_1\{e_2/x\} \Downarrow v}{(\lambda x.\, e_1)\; e_2 \Downarrow v}$$

c) $$\frac{e_1 \Downarrow \lambda x.\, e \qquad e_2 \Downarrow v' \qquad e\{v'/x\} \Downarrow v}{e_1\; e_2 \Downarrow v}$$

$\Rightarrow$ d) $$\frac{e_1 \Downarrow \lambda x.\, e \qquad e\{e_2/x\} \Downarrow v}{e_1\; e_2 \Downarrow v}$$

Answer: *Choice (d) does not evaluate argument and substitutes e_2 eagerly where choice (c) represents evaluates the argument e_2 to v' before carrying out the substitution (and thus follows CBV evaluation strategy).*

7. Consider translating a lambda calculus with booleans and a conditional expression to the pure lambda calculus. We will use CBV semantics for both the source and target languages. The translation of booleans is given by the following.

$$\mathcal{T}[\![\mathsf{true}]\!] = \lambda x.\, \lambda y.\, x$$
$$\mathcal{T}[\![\mathsf{false}]\!] = \lambda x.\, \lambda y.\, y$$

Which *one* of the following is an appropriate translation of $\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$?
(Assume that $x, y \notin FV(e_1) \cup FV(e_2) \cup FV(e_3)$. Hint: note that the evaluation of $\mathsf{if\ false\ then}\ \Omega\ \mathsf{else}\ 0$ should terminate, where $\Omega \triangleq (\lambda x.\, x\; x)\; (\lambda x.\, x\; x)$ is a non-terminating expression.)

a) $\mathcal{T}[\![\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!] = \mathcal{T}[\![e_1]\!]\ \mathcal{T}[\![e_2]\!]\ \mathcal{T}[\![e_3]\!]$.

b) $\mathcal{T}[\![\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!] = \mathcal{T}[\![e_1]\!]\ \ (\lambda x.\, \mathcal{T}[\![e_2]\!])\ \ (\lambda y.\, \mathcal{T}[\![e_3]\!])$

$\Rightarrow$ c) $\mathcal{T}[\![\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!] = (\mathcal{T}[\![e_1]\!]\ \ (\lambda x.\, \mathcal{T}[\![e_2]\!])\ \ (\lambda y.\, \mathcal{T}[\![e_3]\!]))\ \ \lambda y.\, y$

d) $\mathcal{T}[\![\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!] = (\lambda x.\, \lambda y.\, \mathcal{T}[\![e_1]\!])\ \mathcal{T}[\![e_2]\!]\ \mathcal{T}[\![e_3]\!]$

Answer: *In choice (c), the branches are not evaluated as they are lifted to functions and thus aren't evaluated until they are applied to some argument. This prevents from terms such as ω getting evaluated regardless of the value of the condition.*

8. Suppose that type inference on a program has produced the following constraint set.

$$\{A = B \times C, \quad B = A \to C, \quad C = \textbf{int}\}$$

If we run the unification algorithm on this set of constraints, what will be the result?

a) The unification algorithm will fail to terminate.

⇒ b) The unification algorithm will terminate with failure.

c) The unification algorithm will produce a unification equivalent to

$$\begin{aligned}[&A \mapsto (\textbf{int} \to \textbf{int}) \times \textbf{int},\\ &B \mapsto \textbf{int} \to \textbf{int},\\ &C \mapsto \textbf{int}]\end{aligned}$$

d) The unification algorithm will produce a unification equivalent to

$$\begin{aligned}[&A \mapsto (A \to \textbf{int}) \times \textbf{int},\\ &B \mapsto A \to \textbf{int},\\ &C \mapsto \textbf{int}]\end{aligned}$$

Answer: *Unification results in circular constraints such as $A = A \to C \times C$. This will not match any of the cases in the unification algorithm; it thus terminates with failure.*

9. Which *two* of the following are valid typing judgments? Circle **both** valid typing judgments.

⇒ a) $f : \textbf{int} \times \textbf{int} \vdash 40 + (\#1\ f) : \textbf{int}$

b) $f : \textbf{int} \times \textbf{int} \vdash 40 + (\#1\ f)\ (\#2\ f) : \textbf{int}$

c) $f : \textbf{int} \to \textbf{int} \vdash 40 + f\ x : \textbf{int}$

⇒ d) $f : \textbf{int} \times (\textbf{int} \to \textbf{int}) \vdash 40 + (\#2\ f)\ (\#1\ f) : \textbf{int}$

Answer:

*In choice (b), the sub-term $(\#1\ f)\ (\#2\ f)$ is ill-typed because $\#f$ has type **int** that is not a function type, and yet it is applied to $(\#2\ f)$. This does not match with the premises of* T-APP *rule.*

In choice (c), f has the right function type but x is not well-typed since the typing environment does not contain x.

10. Suppose we have a typed lambda calculus with references, where the types are given by the following grammar.

$$\tau ::= \textbf{int} \mid \tau_1 \to \tau_2 \mid \tau\ \textbf{ref}$$

True or false: there exists a well-typed expression e such that evaluation of e produces a store

$$[\ell_1 \mapsto \ell_2, \ell_2 \mapsto \ell_1],$$

for some locations ℓ_1 and ℓ_2.

a) True

⇒ b) False

Answer: *The store*

$$[\ell_1 \mapsto \ell_2, \ell_2 \mapsto \ell_1],$$

*leads to circular typing. Suppose ℓ_1 is given the type **int ref**, then from the assign typing rule, it should be the case that ℓ_2 has type **int**, but since ℓ_2 is itself a reference, it cannot have the type **int**.*

11. Consider the *unify* algorithm used in type inference. Circle the substitution that could possibly be returned by the algorithm on $\mathit{unify}(\{Z \equiv U \to W, X \equiv Y, Y \equiv Z\})$.

⇒ a) $[Y \mapsto U \to W] \circ [X \mapsto Y] \circ [Z \mapsto U \to W]$

b) $[Y \mapsto U \to W, X \mapsto Y, Z \mapsto U \to W]$

c) $[]$

d) $[Y \mapsto U \to W, X \mapsto U \to W]$

Answer: *Follow the unification algorithm—if there is more than one substitution, it always returns the composition as shown in choice (a).*

12. Consider the following lambda-calculus program with references. Assume CBV semantics.

$$\begin{array}{l} \mathsf{let}\ x = \mathsf{ref}\ 7\ \mathsf{in} \\ \mathsf{let}\ y = (x := \lambda z.\,(!x\ z) + z)\ \mathsf{in} \\ !x\ 35 \end{array}$$

Which *one* of the following statements is true?

a) This program will terminate.

⇒ b) This program will diverge (i.e., not terminate).

c) This program will get stuck.

Answer:
Assume ℓ gets allocated when ***ref*** *7 is evaluated. Then follow the evaluation of* LET *and* ASSIGN *rules, we finally have the term $!\ell\ 35$ with store $[\ell \mapsto (\lambda z.\,(!\ell\ z) + z)]$. Following the* DEREF *rule, we have $(\lambda z.\,(!\ell\ z) + z)\ 35$. After β-reduction, we have $(!ell\ 35) + 35$. The lookup thus continues and the evaluation diverges.*

13. Show why $[x \mapsto \textbf{bool} \rightarrow \textbf{int}] \vdash (\lambda y\!:\!\textbf{int}.\ x\ y)\ \textbf{true}$ does not hold.

Answer: *Left as an exercise.*

14. Consider a translation of a language with integers and arithmetic where addition is translated (incorrectly) as follows. (Note that $\Omega \triangleq (\lambda x.\,x\ x)\ (\lambda x.\,x\ x)$ is a non-terminating expression.)

$$\begin{aligned} \mathcal{T}[\![e_1 + e_2]\!] = \ & \mathsf{let}\ m = \mathcal{T}[\![e_1]\!]\ \mathsf{in} \\ & \mathsf{let}\ n = \mathcal{T}[\![e_2]\!]\ \mathsf{in} \\ & \mathsf{if}\ n = 0\ \mathsf{then}\ \Omega\ \mathsf{else}\ m + n \end{aligned}$$

Assuming all other language features are translated correctly, this translation is sound but not complete. (See Appendix B for the definitions of soundness and completeness of translation.)

Using an example, explain why the translation is not complete.

⇒ a)

Consider $e = 1 + 0$. In the source language, it evaluates to 1 whereas it does not evaluate to a value in the target.

15. Consider the translation in the previous question. A student puts the following argument.

Consider $e = 1 + 0$. The translated expression is as shown below.

$$\begin{aligned}\mathcal{T}[\![1+0]\!] = \; & \textsf{let } m = \mathcal{T}[\![1]\!] \textsf{ in} \\ & \textsf{let } n = \mathcal{T}[\![0]\!] \textsf{ in} \\ & \textsf{if } n = 0 \textsf{ then } \Omega \textsf{ else } 1 + 0\end{aligned}$$

Evaluating $\mathcal{T}[\![1+0]\!]$ will yield Ω whereas evaluating $(1 + 0)$ yields 1. Since $\Omega \neq 1$, this implies the translation is not sound as well.

What is the flaw in the argument?

⇒ a)

Ω is not a value. Hence, the premise of the soundness is false. Since false implies anything, the lemma holds and $\mathcal{T}[\![1+0]\!]$ is not a counter example.

16. Suppose we made a mistake in the typing judgments for simply-typed lambda calculus and replaced the rule T-App with the following rule (leaving all other rules unchanged):

$$\text{T-APP}\ \frac{\Gamma \vdash e_1 : \tau \to \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau}$$

The resulting type system is no longer sound. However, the progress lemma holds. (See Appendix D for a statement of type soundness, and the type preservation and progress lemmas.)

Using an example, explain where the preservation lemma fails.

⇒ a)

Consider $(\lambda x.\,\textsf{true})\ 1$. According to the new typing rule, the type of the expression is **int**. However, after taking a step, it evaluates to true that has the type **bool**. Since **int** and **bool** are different types, the type preservation lemma fails.

Appendix A Pure lambda calculus semantics

Syntax:

$$e ::= x \mid \lambda x.\, e \mid e_1\ e_2$$
$$v ::= \lambda x.\, e$$

Call-by-value Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{e_2 \longrightarrow e_2'}{v\ e_2 \longrightarrow v\ e_2'} \qquad \frac{}{(\lambda x.\, e)\ v \longrightarrow e\{v/x\}}$$

Call-by-name Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$

Full beta-reduction Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{e_2 \longrightarrow e_2'}{e_1\ e_2 \longrightarrow e_1\ e_2'} \qquad \frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$

Normal-order Evaluation Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \text{ where } e_1 \text{ is not of the form } \lambda x.\, e \qquad \frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$

Appendix B Adequacy of translation

A translation is sound if every target evaluation represents a source evaluation:

Soundness: $\forall e \in \mathbf{Exp}_{\text{src}}.$ if $\mathcal{T}[\![e]\!] \longrightarrow^*_{\text{trg}} v'$ then $\exists v.\ e \longrightarrow^*_{\text{src}} v$ and v' equivalent to v

A translation is complete if every source evaluation has a target evaluation.

Completeness: $\forall e \in \mathbf{Exp}_{\text{src}}.$ if $e \longrightarrow^*_{\text{src}} v$ then $\exists v'.\ \mathcal{T}[\![e]\!] \longrightarrow^*_{\text{trg}} v'$ and v' equivalent to v

Appendix C Environment Semantics

$$\frac{}{\langle x, \rho\rangle \Downarrow \rho(x)} \qquad \frac{}{\langle n, \rho\rangle \Downarrow n} \qquad \frac{\langle e_1, \rho\rangle \Downarrow n_1 \quad \langle e_2, \rho\rangle \Downarrow n_2}{\langle e_1 + e_2, \rho\rangle \Downarrow n}\ n = n_1 + n_2$$

$$\frac{}{\langle \lambda x.\, e, \rho\rangle \Downarrow (\lambda x.\, e, \rho)} \qquad \frac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho_{lex}[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

Appendix D Type soundness

Theorem (Type soundness). *If* $\vdash e : \tau$ *and* $e \longrightarrow^* e'$ *then either* e' *is a value, or there exists* e'' *such that* $e' \longrightarrow e''$.

Lemma (Preservation). *If* $\vdash e : \tau$ *and* $e \longrightarrow e'$ *then* $\vdash e' : \tau$.

Lemma (Progress). *If* $\vdash e : \tau$ *then either* e *is a value or there exists an* e' *such that* $e \longrightarrow e'$.