CS 320: Final Exam

Total: 100 pts

CS 320 Course Staff

Name: Nathan Mull

BU ID: 12345678

- You will have 120 minutes to complete this exam. Please read through every question carefully, some questions are easier than others.
- Please do not remove any pages from the exam.
- Please make very clear what your final solution for each problem (e.g., by surrounding it in a box). We reserve the right to mark off points if we cannot tell what your final solution is.
- You may use the pages marked *"This page is intentionally left blank."* for scratch work, but we **will not** consider any work on these pages when grading.
- Unless stated otherwise, you should only need the rules provided **in that problem** for your derivations.

This page is intentionally left blank.

Problem 1 (10 pts). Consider the following grammar:

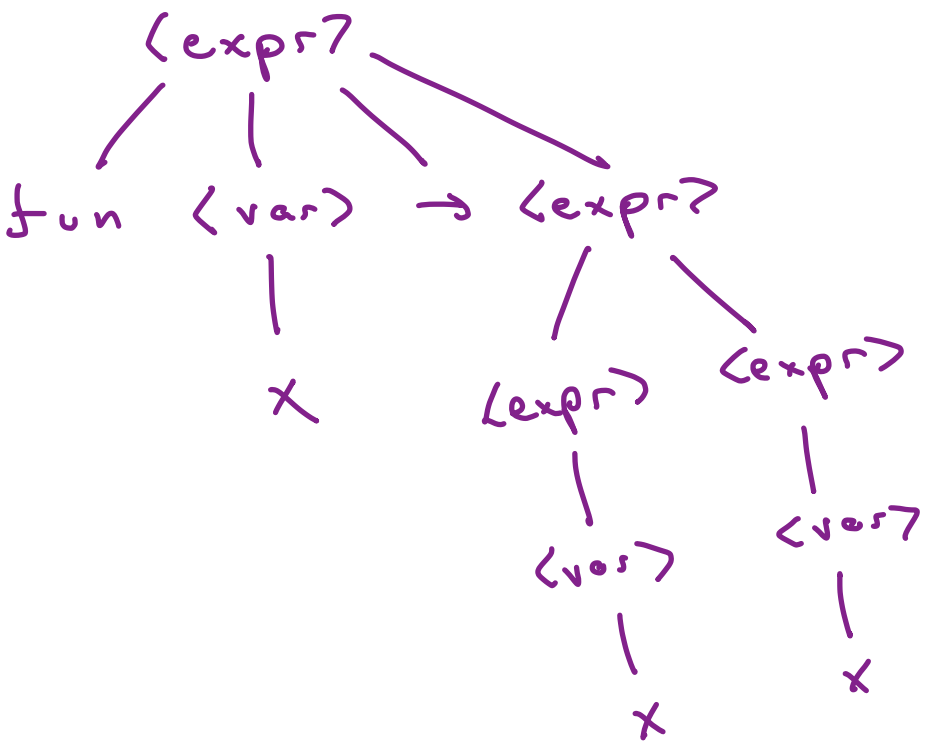
$$\langle expr \rangle ::= \mathsf{fun}\ \langle var \rangle \to \langle expr \rangle \mid \langle expr \rangle\ \langle expr \rangle \mid \langle var \rangle$$
$$\langle var \rangle ::= x$$

Is the grammar above ambiguous? If yes, present a sentence recognized by the grammar along with two distinct parse trees for the sentence. If not, explain why the grammar is unambiguous.

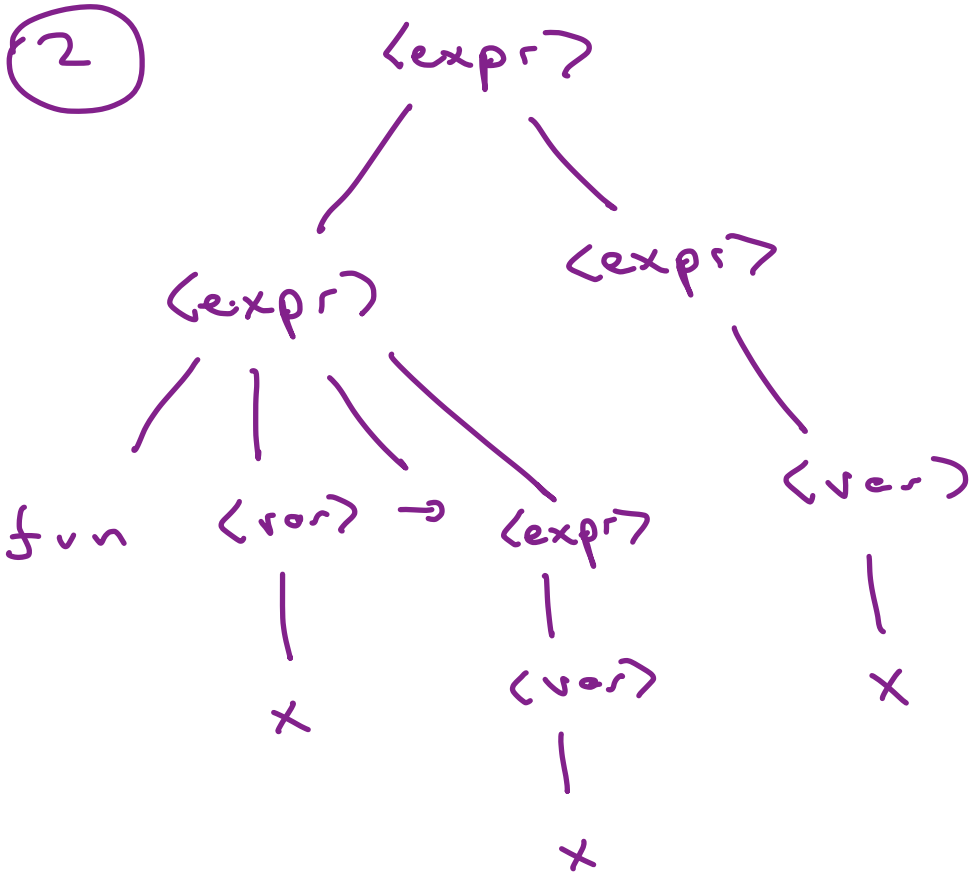
Solution.

fun x → x x

1



Solution. *(Continued)*



Problem 2 (15 pts). In this course, we considered two evaluation strategies: **call-by-value (CBV)** and **call-by-name (CBN)**. For CBV evaluation, we gave a substitution-based semantics (mini-project 1) and an environment-based semantics (mini-projects 2 and 3). For CBN evaluation, we only gave a substitution-based semantics. In the next 3 problems, we will consider an environment-based semantics for CBN evaluation, in particular, for let-expressions and function application.

With CBV evaluation, the environment binds variables to *values*, requiring us to evaluate the value of a variable (or function argument) before binding it. However, with CBN evaluation, we want to delay the evaluation of a variable until it is used. One possible solution would be to allow environments to bind variables to *expressions*. But since expressions can refer to variables, we need to implement lexical scoping, by *putting expressions into closures before binding* (in particular, note that closures can hold expressions other than function expressions):

$$\frac{\langle\ \mathcal{E}[x \mapsto (\!|\ \mathcal{E}\ ,\ e_1\ |\!)]\ ,\ e_2\ \rangle \Downarrow v}{\langle\ \mathcal{E}\ ,\ \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\ \rangle \Downarrow v}\ \text{CBN-LET} \qquad \frac{\mathcal{E}(x) = (\!|\ \mathcal{E}'\ ,\ e\ |\!) \qquad \langle\ \mathcal{E}'\ ,\ e\ \rangle \Downarrow v}{\langle\ \mathcal{E}\ ,\ x\ \rangle \Downarrow v}\ \text{CBN-VAR}$$

Let e denote the expression:

$$\mathsf{let}\ x = 2\ \mathsf{in}\ \mathsf{let}\ y = x + x\ \mathsf{in}\ \mathsf{let}\ x = 3\ \mathsf{in}\ y$$

Give a derivation, using the above rules, of $\langle\ \varnothing\ ,\ e\ \rangle \Downarrow v$ where v is the value of e according to these rules.

Additional Rules Needed:

$$\frac{n \text{ is an integer literal}}{\langle\ \mathcal{E}\ ,\ n\ \rangle \Downarrow n}\ \text{INTEVAL} \qquad \frac{\langle\ \mathcal{E}\ ,\ e_1\ \rangle \Downarrow v_1 \qquad \langle\ \mathcal{E}\ ,\ e_2\ \rangle \Downarrow v_2}{\langle\ \mathcal{E}\ ,\ e_1 + e_2\ \rangle \Downarrow v_1 + v_2}\ \text{ADDEVAL}$$

Solution.

$\langle \emptyset, 2 \rangle \Downarrow 2$

$\langle \{x \mapsto (\emptyset, 2)\}, x \rangle \Downarrow 2$ $\quad$ $\langle \emptyset, 2 \rangle \Downarrow 2$

$\langle \{x \mapsto (\emptyset, 2)\}, x \rangle \Downarrow 2$

$\langle \{x \mapsto (\emptyset, 2)\}, x + x \rangle \Downarrow 4$

$\langle \{x \mapsto (\{x \mapsto (\emptyset, 2)\}, x+x)\}, 3 \rangle$, $y \rangle \Downarrow 4$

$y \mapsto (\{x \mapsto (\emptyset, 2)\}, x+x)$

$\langle \{x \mapsto (\emptyset, 2), y \mapsto (\{x \mapsto (\emptyset, 2)\}, x+x)\}, \mathsf{let}\ x = 3\ \mathsf{in}\ y \rangle \Downarrow 4$

$\langle \{x \mapsto (\emptyset, 2)\}, \mathsf{let}\ y = x + x\ \mathsf{in}\ \mathsf{let}\ x = 3\ \mathsf{in}\ y \rangle \Downarrow 4$

$\langle \emptyset, \mathsf{let}\ x = 2\ \mathsf{in}\ \mathsf{let}\ y = x + x\ \mathsf{in}\ \mathsf{let}\ x = 3\ \mathsf{in}\ y \rangle \Downarrow 4$

Solution. *(Continued)*

Problem 3 (15 pts). Recall that the rule for CBV evaluating function application in our environment-based semantics is given as follows (note that, for simplicity, we are using unnamed closures):

$$\frac{\langle\ \mathcal{E}\ ,\ e_1\ \rangle \Downarrow (\!|\ \mathcal{E}'\ ,\ \lambda x.e\ |\!) \qquad \langle\ \mathcal{E}\ ,\ e_2\ \rangle \Downarrow v_2 \qquad \langle\ \mathcal{E}'[x \mapsto v_2]\ ,\ e\ \rangle \Downarrow v}{\langle\ \mathcal{E}\ ,\ e_1\ e_2\ \rangle \Downarrow v}\ \text{CBV-App-Env}$$

Write the corresponding CBN-App-Env rule for CBN evaluating function application. Let e denote the following expression:

$$(\mathsf{fun}\ x \to 4)\ (1 + 2 + 3 + 4 + 5)$$

Use your rule to give a derivation of $\langle\ \varnothing\ ,\ e\ \rangle \Downarrow v$ where v is the value of e. (**Hint.** Remember that the point of CBN evaluation is that the argument of a function should *not* be evaluated if it does not appear in the body of the function.)

num rule

Additional Rule Needed:

$$\frac{}{\langle\ \mathcal{E}\ ,\ \mathsf{fun}\ x \to e\ \rangle \Downarrow (\!|\ \mathcal{E}\ ,\ \lambda x.e\ |\!)}\ \text{CBN-Fun-Env}$$

Solution.

$$\frac{\langle \mathcal{E}, e_1 \rangle \Downarrow (\!| \mathcal{E}', \lambda x.e |\!) \qquad \langle \mathcal{E}'[x \mapsto (\!| \mathcal{E}, e_2 |\!)], e \rangle \Downarrow v}{\langle \mathcal{E}, e_1\ e_2 \rangle \Downarrow v}$$

$$\frac{\langle \emptyset, \mathsf{fun}\ x \to e \rangle \Downarrow (\!| \emptyset, \lambda x.4 |\!) \qquad \langle \{x \mapsto (\!| \emptyset, 1+2+3+4+5 |\!)\}, 4 \rangle \Downarrow 4}{\langle \emptyset, (\mathsf{fun}\ x \to 4)(1+2+3+4+5) \rangle \Downarrow 4}$$

Solution. *(Continued)*

Problem 4 (10 pts). In this problem, we will consider the option type in OCaml, which is given by the following ADT:

```
type 'a option =
| None
| Some of 'a
```

In reality, if we have polymorphism and functions, we actually *never* need to introduce options. Instead, we can *encode* expressions of the option type as function expressions. If we replace

- None with the expression $\mathsf{fun}\ n \to \mathsf{fun}\ s \to n$
- $\mathsf{Some}(e)$ with the expression $\mathsf{fun}\ n \to \mathsf{fun}\ s \to s\ e$
- $\mathsf{match}\ e\ \mathsf{with} \mid \mathsf{None} \to e_1 \mid \mathsf{Some}(x) \to e_2$ with the expression $e\ e_1\ (\mathsf{fun}\ x \to e_2)$

then we get OCaml code that is, in some sense, equivalent. In the next few problems, we will look at how this encoding works by demonstrating this equivalence. First we need to show that the encoding produces well-typed expressions.

Let Γ be an arbitrary context. Determine a type τ and constraint $\mathcal{C}$ such that the judgment

$$\Gamma \vdash \mathsf{fun}\ n \to \mathsf{fun}\ s \to n : \tau \dashv \mathcal{C}$$

is derivable, and give the derivation of this judgment. Then use τ and $\mathcal{C}$ to determine the principle type of this expression in Γ.

Rules Needed:

$$\frac{(x:\tau) \in \Gamma}{\Gamma \vdash x : \tau \dashv \emptyset}\ \textsc{Var} \qquad \frac{\alpha \text{ is fresh} \qquad \Gamma, x:\alpha \vdash e : \tau \dashv \mathcal{C}}{\Gamma \vdash \mathsf{fun}\ x \to e : \alpha \to \tau \dashv \mathcal{C}}\ \textsc{Fun}$$

Solution.

$\forall \alpha.\, \forall \beta.\, \alpha \to \beta \to \alpha$

$$\frac{\Gamma, n:\alpha, s:\beta \vdash n : \alpha \dashv \emptyset}{\Gamma, n:\alpha \vdash \mathsf{fun}\ s \to n : \beta \to \alpha \dashv \emptyset}$$

$$\Gamma \vdash \mathsf{fun}\ n \to \mathsf{fun}\ s \to n : \alpha \to \beta \to \alpha \dashv \emptyset$$

Solution. *(Continued)*

$e : \tau$

Problem 5 (10 pts). Let Γ be an arbitrary context and suppose that $\Gamma \vdash e : \tau \dashv \mathcal{C}$ for an expression e, type τ and contraints $\mathcal{C}$. Determine a type τ' and constraint $\mathcal{C}'$ such that

$$\Gamma \vdash \mathsf{fun}\ n \to \mathsf{fun}\ s \to s\ e : \tau' \dashv \mathcal{C}'$$

is derivable, and give the derivation of this judgment. Then, assuming $\mathcal{C}$ has a unifier, use τ' and $\mathcal{C}'$ to determine the principle type of this expression in Γ.

Rules Needed:

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau \dashv \emptyset}\ \textsc{Var} \qquad \frac{\alpha \text{ is fresh} \qquad \Gamma, x : \alpha \vdash e : \tau \dashv \mathcal{C}}{\Gamma \vdash \mathsf{fun}\ x \to e : \alpha \to \tau \dashv \mathcal{C}}\ \textsc{Fun}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathcal{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathcal{C}_2 \qquad \alpha \text{ is fresh}}{\Gamma \vdash e_1\ e_2 : \alpha \dashv \tau_1 \doteq \tau_2 \to \alpha, \mathcal{C}_1, \mathcal{C}_2}\ \textsc{App}$$

Solution.

$\{x : \alpha\} \vdash \beta$

$\forall \alpha.\ \forall \beta.\ \alpha \to (\tau \to \beta) \to \gamma$

$$\frac{\Gamma, n : \alpha, s : \beta \vdash s : \beta \dashv \emptyset \qquad \Gamma, n : \alpha, s : \beta \vdash e : \tau \dashv \mathcal{C}}{\Gamma, n : \alpha, s : \beta \vdash s\ e : \beta \doteq \tau \to \beta, \mathcal{C}}$$

$$\frac{\Gamma, n : \alpha, s : \beta \vdash s\ e : \beta \doteq \tau \to \beta, \mathcal{C}}{\Gamma, n : \alpha \vdash \mathsf{fun}\ s \to s\ e : \beta \to \gamma \dashv \beta \doteq \tau \to \beta, \mathcal{C}}$$

$$\frac{\Gamma, n : \alpha \vdash \mathsf{fun}\ s \to s\ e : \beta \to \gamma \dashv \beta \doteq \tau \to \beta, \mathcal{C}}{\Gamma \vdash \mathsf{fun}\ n \to \mathsf{fun}\ s \to s\ e : \alpha \to \beta \to \gamma \dashv \beta \doteq \tau \to \beta, \mathcal{C}}$$

Solution. *(Continued)*

Problem 6 (15 pts). Next we need to show that encoding is correct with respect to evaluation. Recall the encoding of options:

- None is encoded as $\mathsf{fun}\ n \to \mathsf{fun}\ s \to n$
- $\mathsf{Some}(e)$ is encoded as $\mathsf{fun}\ n \to \mathsf{fun}\ s \to s\ e$
- $\mathsf{match}\ e\ \mathsf{with} \mid \mathsf{None} \to e_1 \mid \mathsf{Some}(x) \to e_2$ is encoded as $e\ e_1\ (\mathsf{fun}\ x \to e_2)$

Based on this, give the encoding for the expression

$$\mathsf{match\ None\ with} \mid \mathsf{None} \to e_1 \mid \mathsf{Some}(x) \to e_2$$

where e_1 and e_2 are arbitrary expressions. Let e denote the expression you wrote down. Assuming that $\langle\ \mathcal{E}\ ,\ e_1\ \rangle \Downarrow v_1$, give a derivation of $\langle\ \mathcal{E}\ ,\ e\ \rangle \Downarrow v_1$.

Rules Needed:

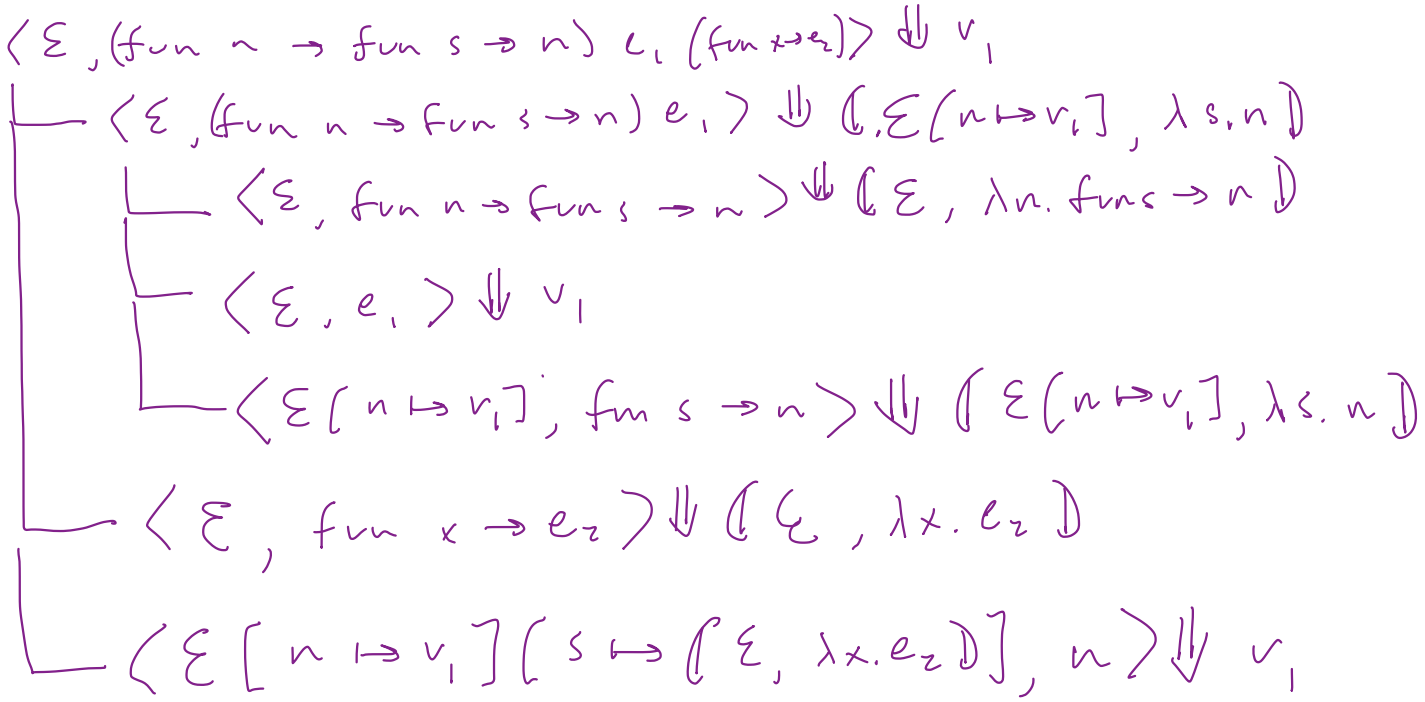
$$\frac{x \text{ is a variable}}{\langle\ \mathcal{E}\ ,\ x\ \rangle \Downarrow \mathcal{E}(x)}\ \textsc{VarEval} \qquad \frac{}{\langle\ \mathcal{E}\ ,\ \mathsf{fun}\ x \to e\ \rangle \Downarrow (\!|\ \mathcal{E}\ ,\ \lambda x.e\ |\!)}\ \textsc{FunEval}$$

$$\frac{\langle\ \mathcal{E}\ ,\ e_1\ \rangle \Downarrow (\!|\ \mathcal{E}'\ ,\ \lambda x.e\ |\!) \qquad \langle\ \mathcal{E}\ ,\ e_2\ \rangle \Downarrow v_2 \qquad \langle\ \mathcal{E}'[x \mapsto v_2]\ ,\ e\ \rangle \Downarrow v}{\langle\ \mathcal{E}\ ,\ e_1\ e_2\ \rangle \Downarrow v}\ \textsc{AppEval}$$

Solution.

(fun n → fun s → n) e_1 (fun x → e_2)

Solution. *(Continued)*



Problem 7 (15 pts). Based on the same encoding of options, give the encoding for the expression

$$\mathsf{match}\ \mathsf{Some}(e')\ \mathsf{with}\ |\ \mathsf{None} \to e_1\ |\ \mathsf{Some}(x) \to e_2$$

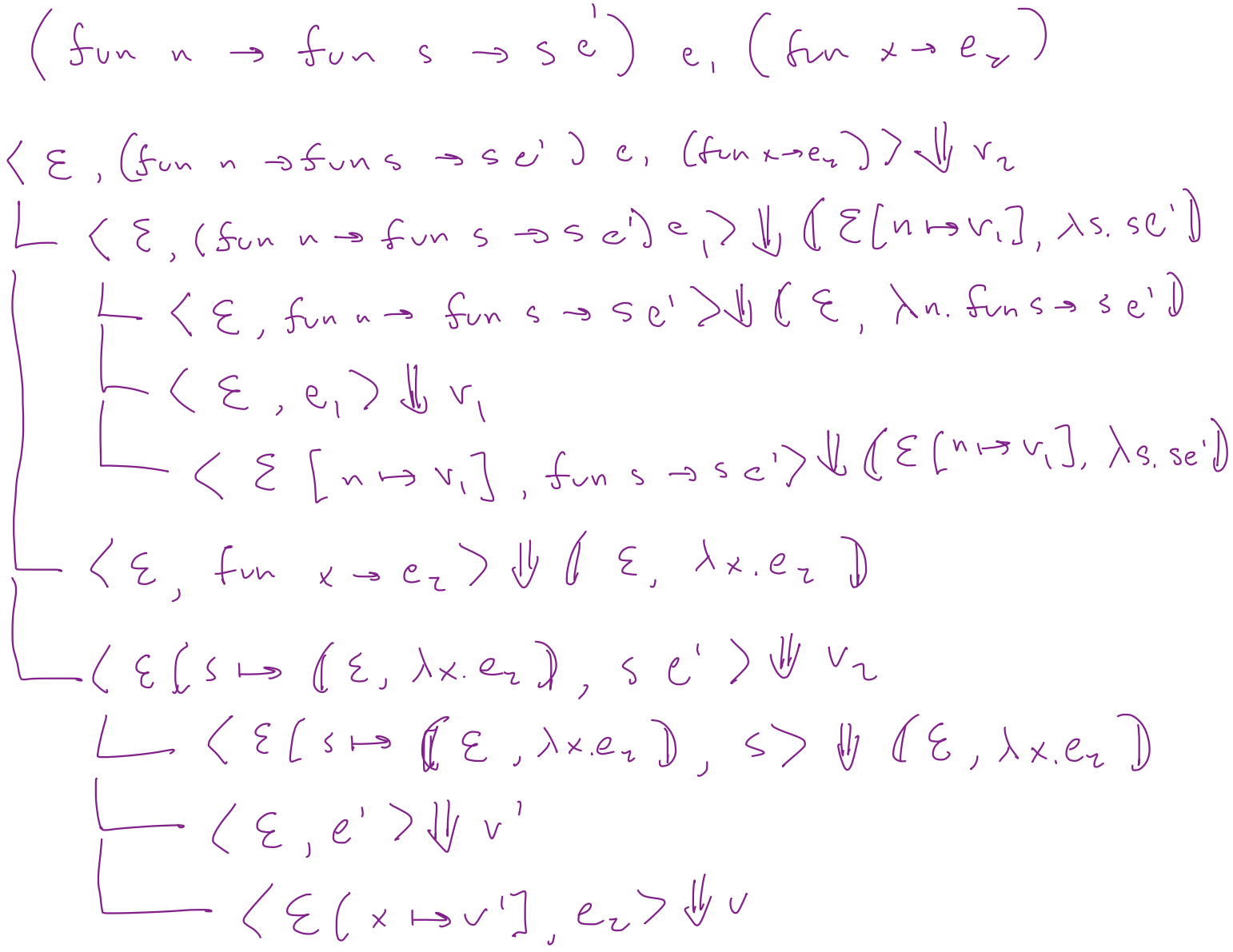
where e_1 and e_2 are arbitrary expressions. Let e denote the expression you wrote down. Assuming that $\langle\ \mathcal{E}\ ,\ e'\ \rangle \Downarrow v'$ and that $\langle\ \mathcal{E}[x \mapsto v']\ ,\ e_2\ \rangle \Downarrow v_2$, give a derivation of $\langle\ \mathcal{E}\ ,\ e\ \rangle \Downarrow v_2$.

Rules Needed:

$$\frac{x \text{ is a variable}}{\langle\ \mathcal{E}\ ,\ x\ \rangle \Downarrow \mathcal{E}(x)}\ \textsc{VarEval} \qquad \frac{}{\langle\ \mathcal{E}\ ,\ \mathsf{fun}\ x \to e\ \rangle \Downarrow (\!|\ \mathcal{E}\ ,\ \lambda x.e\ |\!)}\ \textsc{FunEval}$$

$$\frac{\langle\ \mathcal{E}\ ,\ e_1\ \rangle \Downarrow (\!|\ \mathcal{E}'\ ,\ \lambda x.e\ |\!) \qquad \langle\ \mathcal{E}\ ,\ e_2\ \rangle \Downarrow v_2 \qquad \langle\ \mathcal{E}'[x \mapsto v_2]\ ,\ e\ \rangle \Downarrow v}{\langle\ \mathcal{E}\ ,\ e_1\ e_2\ \rangle \Downarrow v}\ \textsc{AppEval}$$

Solution.



Solution. *(Continued)*

Problem 8 (10 pts). Define a function called `replicate` in OCaml that takes a list ℓ, a function f and a (positive) number n as input and outputs a list where $f^1, f^2, f^3, \ldots, f^n$ are applied (in this order) to each element of ℓ:

$$\texttt{val replicate} : \text{'a list} \to (\text{'a} \to \text{'a}) \to \text{int} \to \text{'a list}$$

e.g., `replicate` $[1;5;8;9]$ f $2 = [f(1); f(f(1)); f(5); f(f(5)); f(8); f(f(8)); f(9); f(f(9))]$. The behavior of the function is undefined if $n \leq 0$.

Solution.

```
let r x f =
  let rec go n =
    if n = 1
    then [f x]
    else List.map f (x :: go n-1)
  in go

let replicate l f n
  List.concat
    (List.map (fun x -> r x f n) l)
```

Solution. *(Continued)*