Name: Hemos ID:

CSE-321 Programming Languages 2007
Midterm

	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Prob 6	Prob 7	Total
Score								
Max	13	11	26	5	10	20	15	100

1 SML Programming [13 pts]

Question 1. [**3 pts**] Give a tail-recursive implementation of `fact` for the factorial function. Perhaps you will need two lines of code.

```
fun fact n =
   let

     ________________________________

   in ________________________________________________
     fact' n 1
   end
```

Question 2. [**5 pts**] Exploit mutable references in SML to implement a factorial function of type `int -> int`. You may use the `fn` keyword, but not the `fun` keyword. That is, do not use the built-in mechanism for building recursive functions in SML. Your program should evaluate to a factorial function that returns $n!$ if its argument n is positive, *i.e.*, $n > 0$. Perhaps you will need three or four lines of code.

```
let

  __________________________________________

  ________________________________________

  ______________________________________________________________________
in

  _______
end
```

Question 3. [**5 pts**] A signature `SET` for sets is given as follows:

```
signature SET =
sig
    type 'a set
    val empty : ''a set
    val singleton : ''a -> ''a set
    val union : ''a set -> '' a set -> ''a set
    val intersection : ''a set -> '' a set -> ''a set
    val diff : ''a set -> '' a set -> ''a set
end
```

- `empty` is an empty set.
- `singleton` x returns a singleton set consisting of x.
- `union` s s' returns the union of s and s'.
- `intersection` s s' returns the intersection of s and s'.
- `diff` s s' returns the difference of s and s': the set of elements in s but not in s'.

Give a functional representation of sets by implementing a structure `SetFun` of signature `SET`. You may not use the `if/then/else` construct. Instead use `not`, `andalso`, and `orelse`.

```
structure SetFun : SET where type 'a set = 'a -> bool =
  struct
    type 'a set = 'a -> bool

    val empty = ______________________

    fun singleton x = ________________________

    fun union s s' = ____________________________________

    fun intersection s s' = _____________________________________

    fun diff s s' = ___________________________________________
   end
```

2 True/false questions [11 pts]

For true/false questions, a wrong answer gives a penalty equal to the points assigned to the question. Given an answer only if you are convinced!

Question 1. [**1 pts**] A derivable rule is always admissible. True or false?

____

Question 2. [**1 pts**] When reducing a closed expression, we may need to use α-conversions. True or false?

____

Question 3. [**1 pts**] We can prove $\lambda x.\, e \equiv_\alpha \lambda y.\, e'$ when $x \neq y$ and $y \in FV(e)$ where $FV(e)$ calculates the set of free variables in e. True or false?

____

Question 4. [**1 pts**] Given a function f in the untyped λ-calculus, we write f^n for the function applying f exactly n times, *i.e.*, $f^n = f \circ f \cdots \circ f$ (n times). A fixed point of f is also a fixed point of f^n if $n \geq 1$. True or false?

____

Question 5. [**1 pts**] In the presence of an abort expression $\mathsf{abort}_A\ e$, type safety of the simply typed λ-calculus continues to hold. True or false?

____

Question 6. [**2 pts**] The fixed point construct $\mathsf{fix}\ x\!:\!A.\, e$ makes every type inhabited in the simply typed λ-calculus. True or false?

____

Question 7. [**1 pts**] If an algorithmic typing judgment covers all possible cases of well-typed expressions, it is said to be "sound." True or false?

____

Question 8. [**3 pts**] If a language has no static type system, it cannot be a safe language. True or false?

____

3 Short answers [26 pts]

Question 1. [4 pts] Show the reduction sequence of the following expression under the call-by-name strategy.

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

$\mapsto$ ____________________

$\mapsto$ ____________________

$\mapsto$ ____________________

$\mapsto$ ____________________

Question 2. [2 pts] Suppose that v_1, v_2, and v_3 are all values of type A in the simply typed λ-calculus. Assuming the *lazy* reduction strategy, how many steps does it take to fully reduce to a value the following expression?

$$\mathsf{fst}\ \Big((\lambda x\!:\!A \times A.\, \mathsf{fst}\ x)\ \big((\lambda x\!:\!A.\, x)\ v_1, v_2\big),\quad v_3\Big)$$

(Given a reduction sequence $e \mapsto e' \mapsto e'' \mapsto v$, we say that it takes three steps to fully reduce e, for example.)

____

Question 3. [3 pts] Encode the boolean type bool and its constructs true, false, and $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ using the sum type $A+A$, the unit type unit, and their constructs.

bool = ____________________

true = ____________________

false = ____________________

$\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ = ____________________

Question 4. [2 pts] Give an expression in the extended simply typed λ-calculus that denotes a recursive function f of type $A \rightarrow B$ whose formal argument is x and whose body is e.

____

Question 5. [3 pts] Show the reduction sequence of the expression $\text{!ref } (\lambda x\!:\!A.\, x)$ in the simply typed λ-calculus with mutable references. The reduction begins with an empty store and uses a location l when allocating a reference. The reduction judgment has the form $e \mid \psi \mapsto e' \mid \psi'$ where a store ψ is a collection of bindings of the form $l \mapsto v$.

$\text{!ref } (\lambda x\!:\!A.\, x) \mid \cdot \quad \mapsto$ ______________________________

Question 6. [3 pts] Complete the rule for the store typing judgment $\psi :: \Psi$ in the simply typed λ-calculus with mutable references.

$$\frac{dom(\Psi) = dom(\psi)}{\psi :: \Psi}\ \mathsf{Store}$$

Question 7. [6 pts] Consider the environment semantics (using the environment evaluation judgment $\eta \vdash e \hookrightarrow v$) for the simply typed λ-calculus with a base type bool:

$$\begin{array}{rrcl}
\mathsf{type} & A & ::= & P \mid A \to A \\
\mathsf{base\ type} & P & ::= & \mathsf{bool} \\
\mathsf{expression} & e & ::= & x \mid \lambda x\!:\!A.\, e \mid e\ e \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \\
\mathsf{environment} & \eta & ::= & \cdot \mid \eta, x \hookrightarrow v
\end{array}$$

Give an inductive definition of values:

____

Write the environment evaluation rule for applications:

$$\frac{}{\eta \vdash e_1\ e_2 \hookrightarrow v}\ \mathbf{App}_{\mathsf{e}}$$

Question 8. [3 pts] What is the language construct in C++ that realizes parametric polymorphism, although it is "a terribly hacked and inadequate feature" from our point of view?

____

4 Inductive definition [5 pts]

Suppose that we use a sequence of digits **0** and **1** as a binary representation of a natural number. As usual, the rightmost digit corresponds to the least significant bit and the leftmost digit corresponds to the most significant bit. For example, **1101** denotes a natural number $2^3 + 2^2 + 2^0 = 13$. A syntactic category bin for such sequences of digits can be inductively defined in several ways, but we use the following definition:

$$\mathsf{bin} \quad b \quad ::= \quad \mathbf{0} \mid \mathbf{1} \mid b\mathbf{0} \mid b\mathbf{1}$$

We wish to inductively define a syntactic category pbin for sequences of digits that denote *positive* natural numbers and also do not have a leading **0**. For example, **1101** belongs to pbin, but **01101** does not because it has a leading **0**. **0** does not belong to pbin, either, because it does not denote a positive natural number.

Question 1. [**3 pts**] Give an inductive definition of pbin. You may not introduce auxiliary syntactic categories.

$$\mathsf{pbin} \quad p \quad ::= \quad \underline{\hspace{6cm}}$$

Question 2. [**2 pts**] Give an inductive definition of a function *num* which takes a sequence p belonging to pbin and returns its corresponding decimal number. For example, we have $num(\mathbf{10}) = 2$ and $num(\mathbf{1101}) = 13$.

5 Programming in the λ-calculus [10 pts]

A Church numeral encodes a natural number n as a λ-abstraction $\hat{n}$ which takes a function f and returns $f^n = f \circ f \cdots \circ f$ (n times):

$$\hat{n} \;=\; \lambda f.\, f^n \;=\; \lambda f.\, \lambda x.\, f\ f\ f\ \cdots f\ x$$

The goal of this problem is to define a logarithm function log which finds the logarithm in base 2 of a given non-zero natural number (encoded as a Church numeral).

- $\mathsf{log}\ \widehat{k}$ evaluates to $\widehat{n}$ if $2^n \leq k < 2^{n+1}$.
- log never takes $\widehat{0}$ as an argument. Hence the result of evaluating $\mathsf{log}\ \widehat{0}$ is unspecified.

Your answers may use the following pre-defined constructs: zero, one, succ, $\mathsf{if/then/else}$, pair, eq, halve, and fix.

- zero and one encode natural numbers zero and one, respectively.

$$\begin{array}{rcccl} \mathsf{zero} & = & \hat{0} & = & \lambda f.\, \lambda x.\, x \\ \mathsf{one} & = & \hat{1} & = & \lambda f.\, \lambda x.\, f\ x \end{array}$$

- succ finds the successor of a given natural number.

$$\mathsf{succ} \;=\; \lambda \widehat{n}.\, \lambda f.\, \lambda x.\, \widehat{n}\ f\ (f\ x)$$

- $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ is a conditional construct.

$$\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \;=\; e\ e_1\ e_2$$

- pair creates a pair of two expressions, and fst and snd are projection operators.

$$\begin{array}{rcl} \mathsf{pair} & = & \lambda x.\, \lambda y.\, \lambda b.\, b\ x\ y \\ \mathsf{fst} & = & \lambda p.\, p\ (\lambda t.\, \lambda f.\, t) \\ \mathsf{snd} & = & \lambda p.\, p\ (\lambda t.\, \lambda f.\, f) \end{array}$$

- eq tests two natural numbers for equality.

$$\mathsf{eq} \;=\; \lambda x.\, \lambda y.\, \mathsf{and}\ (\mathsf{isZero}\ (x\ \mathsf{pred}\ y))\ (\mathsf{isZero}\ (y\ \mathsf{pred}\ x))$$

- $\mathsf{halve}\ \widehat{2 * k}$ returns $\widehat{k}$.
  $\mathsf{halve}\ \widehat{2 * k + 1}$ returns $\widehat{k}$.

$$\mathsf{halve} \;=\; \lambda \widehat{n}.\, \mathsf{fst}\ (\widehat{n}\ (\lambda p.\, \mathsf{pair}\ (\mathsf{snd}\ p)\ (\mathsf{succ}\ (\mathsf{fst}\ p)))(\mathsf{pair}\ \mathsf{zero}\ \mathsf{zero}))$$

- fix is the fixed point combinator.

$$\mathsf{fix} \;=\; \lambda F.\, (\lambda f.\, F\ \lambda x.\, (f\ f\ x))\ (\lambda f.\, F\ \lambda x.\, (f\ f\ x))$$

These constructs use the following auxiliary constructs, which you do not need:

$$\begin{array}{rcl}
\mathsf{tt} & = & \lambda t.\, \lambda f.\, t \\
\mathsf{ff} & = & \lambda t.\, \lambda f.\, f \\
\mathsf{and} & = & \lambda x.\, \lambda y.\, x\ y\ \mathsf{ff} \\
\mathsf{isZero} & = & \lambda x.\, x\ (\lambda y.\, \mathsf{ff})\ \mathsf{tt} \\
\mathsf{next} & = & \lambda p.\, \mathsf{pair}\ (\mathsf{snd}\ p)\ (\mathsf{succ}\ (\mathsf{snd}\ p)) \\
\mathsf{pred} & = & \lambda \widehat{n}.\, \mathsf{fst}\ (\widehat{n}\ \mathsf{next}\ (\mathsf{pair}\ \mathsf{zero}\ \mathsf{zero}))
\end{array}$$

Question 1. [3 pts] Use the fixed point combinator to define log. You may use the above pre-defined constructs, but do not expand them into their definitions.

$\mathsf{log}\ =$ ____

Question 2. [7 pts] Define log without using the fixed point combinator. You may use the above pre-defined constructs, but do not expand them into their definitions. (You are not allowed to rewrite your answer to the previous question by expanding fix into its definition!)

$\mathsf{log}\ =\ \lambda \widehat{n}.$ ____

6 Complete call-by-name reduction [20 pts]

Consider the following fragment of the simply typed λ-calculus:

$$\begin{array}{rrcl} \textsf{type} & A & ::= & P \mid A \rightarrow A \\ \textsf{base type} & P & & \\ \textsf{expression} & e & ::= & x \mid \lambda x\!:\!A.\, e \mid e\ e \\ \textsf{value} & v & ::= & \lambda x\!:\!A.\, e \end{array}$$

Under the *call-by-name* (CBN) strategy, an expression reduces to a value using two reduction rules below:

$$\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ \mathit{Lam} \qquad \frac{}{(\lambda x\!:\!A.\, e)\ e' \mapsto [e'/x]e}\ \mathit{App}$$

Note that the second subexpression in an application (*e.g.*, e_2 in $e_1\ e_2$) is not reduced immediately.

In this problem, we will consider a variant of the CBN strategy, called the *complete CBN strategy*, in which we attempt to reduce the expression e in $\lambda x\!:\!A.\, e$ before applying the rule *App*. As a result, we reduce $(\lambda x\!:\!A.\, e)\ e'$ to $[e'/x]e$ by the rule *App* only when the function body e is a normal form. Recall that an expression e is said to be a normal form if no reduction rule is applicable, *i.e.*, if there is no e' such that $e \mapsto e'$. Thus, if a reduction sequence terminates, it must end up with a normal form.

Under the complete CBN strategy, a normal form is not necessarily a value. For example, $\lambda x\!:\!A.\, x\ (\lambda y\!:\!B.\, y)$ is a normal form (because there is no e' such that $\lambda x\!:\!A.\, x\ (\lambda y\!:\!B.\, y) \mapsto e'$) and also a value, whereas $x\ y$ is a normal form (because there is no e' such that $x\ y \mapsto e'$) but not a value. Conversely a value is not necessarily a normal form. For example, $\lambda x\!:\!A.\, (\lambda y\!:\!B.\, y)\ x$ is a value but not a normal form because its body $(\lambda y\!:\!B.\, y)\ x$ reduces to another expression x, as shown in $\lambda x\!:\!A.\, (\lambda y\!:\!B.\, y)\ x \mapsto \lambda x\!:\!A.\, x$. We call a normal form that is a value as a *value normal form*, and a normal form that is not a value as a *non-value normal form*.

In order to syntactically distinguish the two kinds of normal forms, we introduce two new syntactic categories:

$$\begin{array}{rrcl} \textsf{non-value normal form} & \mathit{xnf} & ::= & x \mid \mathit{xnf}\ e \\ \textsf{normal form} & \mathit{nf} & ::= & \mathit{xnf} \mid \lambda x\!:\!A.\, \mathit{nf} \end{array}$$

Examples of non-value normal forms are $x\ (\lambda y\!:\!A.\, y)$ and $x\ y$. Note that a non-value normal form can always be written as $x\ e_1\ e_2\ \cdots\ e_n$. A normal form nf is either a non-value normal form xnf or a value normal form $\lambda x\!:\!A.\, \mathit{nf}'$. Note that the body of a value normal form $\lambda x\!:\!A.\, \mathit{nf}$ is just a normal form, not necessarily another value normal form.

Question 1. [6 pts] Give the rules for the reduction judgment $e \mapsto e'$. You need three rules.

$$\frac{}{\quad \mapsto \quad} \qquad \frac{}{\quad \mapsto \quad}$$

$$\frac{}{\quad \mapsto \quad}$$

Question 2. **[4 pts]** Give the rules for the evaluation judgment $e \hookrightarrow nf$ which means that an expression e evaluates to a normal form nf. You need three rules and we provide one.

$$\frac{}{nf \hookrightarrow nf} \qquad \frac{}{\quad\hookrightarrow\quad}$$

$$\frac{}{\quad\hookrightarrow\quad}$$

Question 3. **[4 pts]** Give the definition of evaluation contexts corresponding to the complete CBN strategy.

$$\text{evaluation context} \quad \kappa \quad ::= \quad \underline{\qquad\qquad\qquad}$$

Question 4. **[6 pts]** Give the definition of frames and the rules for the state transition judgment $s \mapsto_{\mathsf{C}} s'$ for the abstract machine C. $\sigma \blacktriangleright e$ means that the machine is currently reducing $\sigma[\![e]\!]$, but has yet to analyze e. $\sigma \blacktriangleleft nf$ means that the machine is currently reducing $\sigma[\![nf]\!]$ and has already analyzed nf; that is, it is returning nf to the top frame of σ. Fill in the blank:

$$\begin{array}{lcll}
\text{frame} & \phi & ::= & \underline{\qquad\qquad\qquad} \\
\text{stack} & \sigma & ::= & \square \mid \sigma; \phi \\
\text{state} & s & ::= & \sigma \blacktriangleright e \mid \sigma \blacktriangleleft nf
\end{array}$$

$$\frac{}{\sigma \blacktriangleright nf \mapsto_{\mathsf{C}}} \; Nf_{\mathsf{C}}$$

$$\frac{}{\sigma \blacktriangleright e_1\ e_2 \mapsto_{\mathsf{C}}} \; Lam_{\mathsf{C}}$$

$$\frac{}{\sigma \blacktriangleright \lambda x\!:\!A.\, e \mapsto_{\mathsf{C}}} \; BodyA_{\mathsf{C}}$$

$$\frac{}{\sigma; \lambda x\!:\!A.\, \square \blacktriangleleft nf \mapsto_{\mathsf{C}}} \; BodyR_{\mathsf{C}}$$

$$\frac{}{\sigma; \square\ e_2 \blacktriangleleft xnf \mapsto_{\mathsf{C}}} \; Xnf_{\mathsf{C}}$$

$$\frac{}{\sigma; \square\ e_2 \blacktriangleleft \lambda x\!:\!A.\, nf \mapsto_{\mathsf{C}}} \; App_{\mathsf{C}}$$

7 Type preservation [15 pts]

In this problem, we use the following fragment of the simply typed λ-calculus. We do not consider base types.

$$
\begin{array}{rrcl}
\textsf{type} & A & ::= & P \mid A \to A \\
\textsf{base type} & P & & \\
\textsf{expression} & e & ::= & x \mid \lambda x\!:\!A.\, e \mid e\ e \\
\textsf{value} & v & ::= & \lambda x\!:\!A.\, e \\
\textsf{typing context} & \Gamma & ::= & \cdot \mid \Gamma, x : A
\end{array}
$$

$$
\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \textsf{Var} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x\!:\!A.\, e : A \to B} \to\!\textsf{I} \qquad \frac{\Gamma \vdash e : A \to B \quad \Gamma \vdash e' : A}{\Gamma \vdash e\ e' : B} \to\!\textsf{E}
$$

$$
\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ Lam \qquad \frac{e_2 \mapsto e_2'}{(\lambda x\!:\!A.\, e)\ e_2 \mapsto (\lambda x\!:\!A.\, e)\ e_2'}\ Arg \qquad \frac{}{(\lambda x\!:\!A.\, e)\ v \mapsto [v/x]e}\ App
$$

Question 1. [**5 pts**] Fill in the blank in the next page to complete the proof of the substitution lemma. We assume that a typing context is an unordered set and that variables in a typing context are all distinct.

Question 2. [**10 pts**] Fill in the blank in the page after to complete the proof of the type preservation theorem. Unlike the proof given in the Course Notes, we apply rule induction to $\Gamma \vdash e : A$ instead of $e \mapsto e'$. You may use Lemmas 7.2 (Substitution) and 7.1 (Inversion).

Lemma 7.1 (Inversion). *Suppose* $\Gamma \vdash e : C$.
If $e = x$*, then* $x : C \in \Gamma$.
If $e = \lambda x\!:\!A.\, e'$*, then* $C = A \to B$ *and* $\Gamma, x : A \vdash e' : B$ *for some type* B.
If $e = e_1\ e_2$*, then* $\Gamma \vdash e_1 : A \to C$ *and* $\Gamma \vdash e_2 : A$ *for some type* A.

Lemma 7.2 (Substitution). *If* $\Gamma \vdash e : A$ *and* $\Gamma, x : A \vdash e' : C$*, then* $\Gamma \vdash [e/x]e' : C$*.*

Proof. By rule induction on the judgment $\Gamma, x : A \vdash e' : C$. In the third case, we assume (without loss of generality) that y is a fresh variable such that $y \notin FV(e)$ and $y \neq x$. If $y \in FV(e)$ or $y = x$, we can always choose a different variable by applying an α-conversion to $\lambda y\!:\!C_1.\, e''$.

Case $\dfrac{y : C \in \Gamma, x : A}{\Gamma, x : A \vdash y : C}\,\mathsf{Var}$ where $e' = y$ and $y : C \in \Gamma$:

$\Gamma \vdash y : C$ — from $y : C \in \Gamma$

$[e/x]y = y$ — from $x \neq y$

____

Case $\dfrac{}{\Gamma, x : A \vdash x : A}\,\mathsf{Var}$ where $e' = x$ and $C = A$:

____________________ — assumption

________________________ — $[e/x]x = e$

Case $\dfrac{\Gamma, x : A, y : C_1 \vdash e'' : C_2}{\Gamma, x : A \vdash \lambda y\!:\!C_1.\, e'' : C_1 \to C_2}\,\to\!\mathsf{I}$ where $e' = \lambda y\!:\!C_1.\, e''$ and $C = C_1 \to C_2$:

______________________________ — by induction hypothesis

____________________________________ — by the rule $\to\!\mathsf{I}$

$[e/x]\lambda y\!:\!C_1.\, e'' =$ ________________________ — from $y \notin FV(e)$ and $x \neq y$

____

Case $\dfrac{\Gamma, x : A \vdash e_1 : B \to C \quad \Gamma, x : A \vdash e_2 : B}{\Gamma, x : A \vdash e_1\ e_2 : C}\,\to\!\mathsf{E}$ where $e' = e_1\ e_2$:

________________________ by induction hypothesis on ______________________________

__________________________ by induction hypothesis on __________________________

______________________________ — by the rule $\to\!\mathsf{E}$

$\Gamma \vdash [e/x](e_1\ e_2) : C$ — from ____________________________________

$\square$

Theorem 7.3 (Type preservation). *If* $\Gamma \vdash e : A$ *and* $e \mapsto e'$*, then* $\Gamma \vdash e' : A$*.*

Proof. By rule induction on the judgment $\Gamma \vdash e : A$.

Case $\dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{Var}$ where $e = x$:
There is no expression e' such that $x \mapsto e'$, so we do not need to consider this case.

Case $\dfrac{\Gamma, x : A_1 \vdash e'' : A_2}{\Gamma \vdash \lambda x{:}A_1.\, e'' : A_1 \rightarrow A_2}\,{\rightarrow}\mathsf{I}$ where $e = \lambda x{:}A_1.\, e''$ and $A = A_1 \rightarrow A_2$:
There is no expression e' such that $\lambda x{:}A_1.\, e'' \mapsto e'$, so we do not need to consider this case.

Case $\dfrac{\Gamma \vdash e_1 : C \rightarrow A \quad \Gamma \vdash e_2 : C}{\Gamma \vdash e_1\ e_2 : A}\,{\rightarrow}\mathsf{E}$ where $e = e_1\ e_2$:
There are three subcases depending on the reduction rule used in the derivation of $e \mapsto e'$. Note that if e_1 is a λ-abstraction, it must have the form $\lambda x{:}C.\, e''$ by Lemma 7.1 with $\Gamma \vdash e_1 : C \rightarrow A$.

Subcase $\dfrac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\,\mathit{Lam}$ where $e' = e_1'\ e_2$:

____

by induction hypothesis on ____________________ with _______________

________________________ from ____

Subcase $\dfrac{e_2 \mapsto e_2'}{(\lambda x{:}C.\, e'')\ e_2 \mapsto (\lambda x{:}C.\, e'')\ e_2'}\,\mathit{Lam}$ where $e_1 = \lambda x{:}C.\, e''$ and $e' = (\lambda x{:}C.\, e'')\ e_2'$:

____

by induction hypothesis on ________________ with _______________

__________________________ from ____

Subcase $\dfrac{}{(\lambda x{:}C.\, e'')\ v \mapsto [v/x]e''}\,\mathit{App}$ where $e_1 = \lambda x{:}C.\, e''$ and $e_2 = v$ and $e' = [v/x]e''$:

___________________________ by Lemma 7.1 with __________________________

____

by Lemma 7.2 with _______________ and _____________________ □