

Name:

Hemos ID:

CSE-321 Programming Languages 2006
Midterm

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5	Problem 6	Problem 7	Total
Score								
Max	20	15	15	15	15	10	10	100

1 SML Programming [20 pts]

Question 1. [5 pts] Give a tail recursive implementation `inorder` for inorder traversals of binary trees. Fill in the blank:

```
datatype 'a tree = Leaf of 'a | Node of 'a tree * 'a * 'a tree

(* inorder : 'a tree -> 'a list *)
fun inorder t =
  let
    (* inorder' : 'a tree -> 'a list -> 'a list *)
    fun inorder' (Leaf x) post =

      | inorder' (Node (left, x, right)) post =

    _____
  in
    inorder' t []
  end
```

Question 2. [5 pts] Rewrite expressions in the left column *without* using the `if/then/else` construct. You may use `not`, `andalso`, and `orelse`.

with if/then/else	without if/then/else
<code>if e then false else true</code>	
<code>if x then true else y</code>	
<code>if x then y else false</code>	
<code>if x then false else y</code>	
<code>if x then y else true</code>	

Question 3. [10 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 member : ''a set -> ''a -> bool
  val insert : ''a set -> ''a -> ''a set
  val remove : ''a set -> ''a -> ''a set
  val union : ''a set -> '' a set -> ''a set
end
```

- `empty` is an empty set.
- `singleton x` returns a singleton set consisting of x .
- `member s x` returns `true` if x is a member of s ; otherwise it returns `false`.
- `insert s x` adds x to the set s and returns the resultant set.
- `remove s x` removes x from the set s and returns the resultant set. If x is not a member of s , then `remove s x` returns s .
- `union s s'` returns the union of s and s' .

Give a functional representation of sets by implementing a structure `SetFun` of signature `SET`. In your answer, do not use the `if/then/else` construct; instead take advantage of the result from Question 2. Fill in the blank:

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

    val empty = _____

    fun singleton x = _____

    fun member s = _____

    fun insert s x = _____

    fun remove s x = _____

    fun union s s' = _____
  end
```

2 Reductions in the λ -calculus [15 pts]

Let us abbreviate an identity function $\lambda x_i. x_i$ as id_i . You will show the reduction sequence of $(\underline{\text{id}_1} \text{id}_2) (\text{id}_3 (\lambda z. \text{id}_4 z))$ under the call-by-name strategy (Question 1) and under the call-by-value strategy (Question 2).

Question 1. [5 pts] Show the reduction sequence under the call-by-name strategy. Underline the redex at each step. Do not expand id_i back to $\lambda x_i. x_i$.

$(\text{id}_1 \text{id}_2)$ $(\text{id}_3 (\lambda z. \text{id}_4 z))$

\mapsto

\mapsto

\mapsto

Question 2. [5 pts] Show the reduction sequence under the call-by-value strategy. Underline the redex at each step. Do not expand id_i back to $\lambda x_i. x_i$.

$(\text{id}_1 \text{id}_2)$ $(\text{id}_3 (\lambda z. \text{id}_4 z))$

\mapsto

\mapsto

\mapsto

Question 3. [5 pts] Fill in the blank with the result of applying α -conversion to the expression in the left. We have supplied a variable to be used in the conversion. If it is impossible to apply α -conversion using the given variable, write “impossible.”

- (example) $\lambda x. x \equiv_\alpha \lambda y. \underline{y}$

$$\lambda x. \lambda x'. x x' \equiv_\alpha \lambda x'. \underline{\hspace{4cm}}$$

$$\lambda x. \lambda x'. x x' x'' \equiv_\alpha \lambda x'. \underline{\hspace{4cm}}$$

$$\lambda x. \lambda x'. x x' x'' \equiv_\alpha \lambda x''. \underline{\hspace{4cm}}$$

3 Programming in the λ -calculus [15 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):

$$\begin{aligned}\hat{0} &= \lambda f. f^0 = \lambda f. \lambda x. x \\ \hat{1} &= \lambda f. f^1 = \lambda f. \lambda x. f x \\ &\dots \\ \hat{n} &= \lambda f. f^n = \lambda f. \lambda x. f f f \cdots f x\end{aligned}$$

Question 1. [5 pts] Define an operation `double` for doubling a given natural number. Specifically `double \hat{n}` returns $\widehat{2 * n}$. Fill in the blank:

$$\text{double} = \lambda \hat{n}. \underline{\hspace{10em}}$$

Question 2. [10 pts] Define an operation `halve` for halving a given natural number. Specifically `halve \hat{n}` returns $\widehat{n/2}$:

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

For defining `halve`, you want to exploit the encoding of pairs in the Course Notes:

$$\begin{aligned}\text{pair} &= \lambda x. \lambda y. \lambda b. b x y \\ \text{fst} &= \lambda p. p (\lambda t. \lambda f. t) \\ \text{snd} &= \lambda p. p (\lambda t. \lambda f. f)\end{aligned}$$

Use `pair`, `fst`, and `snd` without expanding them into the above definition. To make your answer more readable, you also want to use `zero` for a natural number zero and `succ` for finding the successor to a given natural number:

$$\begin{aligned}\text{zero} &= \hat{0} = \lambda f. \lambda x. x \\ \text{succ} &= \lambda \hat{n}. \lambda f. \lambda x. \hat{n} f (f x)\end{aligned}$$

Fill in the blank:

$$\text{halve} = \lambda \hat{n}. \underline{\hspace{10em}}$$

4 A weird reduction strategy [15 pts]

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

type	$A ::= P \mid A \rightarrow A$
base type	P
expression	$e ::= x \mid \lambda x:A. e \mid e e$
value	$v ::= \lambda x:A. e$

We will develop a weird strategy specified as follows:

- Given an application $e_1 e_2$, we first reduce e_2 .
- After reducing e_2 to a value, we reduce e_1 .
- When e_1 reduces to a λ -abstraction, we apply the β -reduction.

Question 1. [5 pts] Give the rules for the reduction judgment $e \mapsto e'$ under the weird reduction strategy. You need three rules.

_____ \mapsto _____ \mapsto _____ \mapsto

Question 2. [5 pts] Give the rules for the evaluation judgment $e \hookrightarrow v$ under the weird reduction strategy. You need two rules.

_____ \hookrightarrow _____ \hookrightarrow

Question 3. [5 pts] Give the definition of evaluation contexts corresponding to the weird reduction strategy:

evaluation context $\kappa ::=$

5 Substitution theorem [15 pts]

Prove the substitution theorem for the following fragment of the simply typed λ -calculus:

type	$A ::= P \mid A \rightarrow A$
base type	P
expression	$e ::= x \mid \lambda x:A. e \mid e e$
typing context	$\Gamma ::= \cdot \mid \Gamma, x : A$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \text{Var} \quad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x:A. e : A \rightarrow B} \rightarrow I \quad \frac{\Gamma \vdash e : A \rightarrow B \quad \Gamma \vdash e' : A}{\Gamma \vdash e e' : B} \rightarrow E$$

Theorem (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 $\Gamma, x : A \vdash e' : C$. We assume that all variables in a typing context are distinct. We also assume that variable clashes never occur in the rule $\rightarrow I$. That is, x in the rule $\rightarrow I$ is always a fresh variable.

Fill in the blank:

Case $\frac{y : C \in \Gamma}{\Gamma, x : A \vdash y : C} \text{Var}$ where $e' = y$

$$\frac{\Gamma \vdash y : C}{\Gamma \vdash [e/x]y : C} \text{ from } \underline{\hspace{2cm}} \text{ and the rule } \underline{\hspace{2cm}} \text{ from } x \neq y$$

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

$$\frac{\Gamma \vdash e : A}{[e/x]x = e} \text{ from the assumption}$$

Case $\frac{\Gamma, x : A, y : B_1 \vdash e'' : B_2}{\Gamma, x : A \vdash \lambda y : B_1. e'' : B_1 \rightarrow B_2} \rightarrow I$ where $e' = \lambda y : B_1. e''$ and $C = B_1 \rightarrow B_2$

_____ by IH on the premise

_____ by the rule $\rightarrow I$

$$[e/x]\lambda y : B_1. e'' = \lambda y : B_1. [e/x]e'' \text{ from } \underline{\hspace{2cm}} \text{ and } \underline{\hspace{2cm}}$$

Case $\frac{\Gamma, x : A \vdash e_1 : B \rightarrow C \quad \Gamma, x : A \vdash e_2 : B}{\Gamma, x : A \vdash e_1 e_2 : C} \rightarrow E$ where $e' = e_1 e_2$

by IH on the first premise

by IH on the second premise

$\Gamma \vdash [e/x]e_1 [e/x]e_2 : C$

by the rule $\rightarrow E$

by the definition of substitution

□

6 Transitivity [10 pts]

In a reduction sequence judgment $e \mapsto^* e'$, we use \mapsto^* for the reflexive and transitive closure of \mapsto . That is, $e \mapsto^* e'$ holds if $e \mapsto e_1 \mapsto \dots \mapsto e_n = e'$ where $n \geq 0$. Then we would expect that $e \mapsto^* e'$ and $e' \mapsto^* e''$ together imply $e \mapsto^* e''$, since we obtain a proof of $e \mapsto^* e''$ simply by concatenating $e \mapsto e_1 \mapsto \dots \mapsto e_n = e'$ and $e' \mapsto e'_1 \mapsto \dots \mapsto e'_m = e''$:

$$e \mapsto e_1 \mapsto \dots \mapsto e_n = e' \mapsto e'_1 \mapsto \dots \mapsto e'_m = e''$$

You will prove this transitivity property of \mapsto^* under the following inductive definition:

$$\frac{}{e \mapsto^* e} \text{ Refl} \quad \frac{e \mapsto e'' \quad e'' \mapsto^* e'}{e \mapsto^* e'} \text{ Trans}$$

Theorem (Transitivity). *If $e \mapsto^* e'$ and $e' \mapsto^* e''$, then $e \mapsto^* e''$.*

Fill in the blank below and complete the proof:

Proof. By rule induction on the judgment _____.

Case $\frac{}{\quad} \mapsto^* \langle \quad \rangle$ where _____

assumption

from _____

Case $\frac{}{\quad} \mapsto^* \langle \quad \rangle$

assumption

by _____

from _____

□

7 Abstract machine C [10 pts]

Consider the following fragment of the simply typed λ -calculus for the call-by-value strategy:

type	$A ::= P \mid A \rightarrow A$
base type	P
expression	$e ::= x \mid \lambda x:A. e \mid e e \mid \text{fix } x:A. e$
value	$v ::= \lambda x:A. e \mid$
frame	$\phi ::= \square e \mid v \square \mid$
stack	$\sigma ::= \square \mid \sigma; \phi$
state	$s ::= \sigma \blacktriangleright e \mid \sigma \blacktriangleleft v$

The goal of this problem is to write the rules for the state transition judgment $s \mapsto_C s'$ for the abstract machine C. For your reference, we give the rules for the reduction judgment $e \mapsto e'$ below:

$$\frac{e_1 \mapsto e'_1}{e_1 e_2 \mapsto e'_1 e_2} \text{Lam} \quad \frac{e_2 \mapsto e'_2}{v e_2 \mapsto v e'_2} \text{Arg} \quad \frac{}{(\lambda x:A. e) v \mapsto [v/x]e} \text{App} \quad \frac{}{\text{fix } x:A. e \mapsto [\text{fix } x:A. e/x]e} \text{Fix}$$

Fill in the blank and complete each rule:

$$\frac{}{\sigma \blacktriangleright v \mapsto_C} \text{Val}_C$$

$$\frac{}{\sigma \blacktriangleright e_1 e_2 \mapsto_C} \text{Lam}_C$$

$$\frac{}{\sigma; \square e_2 \blacktriangleleft v \mapsto_C} \text{Arg}_C$$

$$\frac{}{\sigma; (\lambda x:A. e) \square \blacktriangleleft v \mapsto_C} \text{App}_C$$

$$\frac{}{\sigma \blacktriangleright \text{fix } x:A. e \mapsto_C} \text{Fix}_C$$