

Name:

Hemos ID:

CSE-321 Programming Languages 2009
Midterm — Sample Solution

	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Total
Score						
Max	15	25	55	25	20	140

1 SML Programming [15 pts]

Question 1. [5 pts] Give a tail-recursive implementation of `fib` for computing Fibonacci numbers.

(Type) `fib: int -> int`

(Description)

`fib n` returns `fib (n - 1) + fib (n - 2)` when $n \geq 2$.

`fib n` returns 1 if $n = 0$ or $n = 1$.

(Invariant) $n \geq 0$.

```
fun fib n =
  let
    fun fib' a _ 0 = a
      | fib' a b n = fib' b (a + b) (n - 1)
  in
    fib' 1 1 n
  end
```

Question 2. [10 pts] Consider the signature `MATRIX` similar to the one that we have seen in Assignment 3.

```
signature MATRIX =
sig
  type t                (* type of square matrices *)
  val identity : int -> t  (* creates an identity matrix *)
  val dim : t -> int      (* dimension *)
  val ++ : t * t -> t     (* addition *)
  val ** : t * t -> t     (* multiplication *)
  val == : t * t -> bool  (* equality *)
end
```

- `t` denotes the type of square matrices.
- `identity n` returns an identity matrix of dimension n .
- `dim A` returns the dimension of matrix A .
- `++ (A1, A2)` adds two matrices A_1 and A_2 .
- `** (A1, A2)` multiplies two matrices A_1 and A_2 .
- `== (A1, A2)` returns `true` if two matrices A_1 and A_2 are equal and `false` otherwise.

The closure of a square matrix A is defined as $I + A + A^2 + A^3 + \dots$ where $I (= A^0)$ is the identity matrix. Alternatively the closure of A can be defined as $I + A + A^2 + \dots + A^i$ where i is the first positive integer such that $I + A + A^2 + \dots + A^i$ is equal to $I + A + A^2 + \dots + A^i + A^{i+1}$.

Implement the functor `ClosureFn` where the member `closure` computes the closure of a given matrix. `closure A` should terminate if A has a closure.

```

functor ClosureFn (Mat : MATRIX) :>
sig
  val closure : Mat.t -> Mat.t
end
=
struct
  fun closure m =
  let
    val one = Mat.identity (Mat.dim m)

    fun findClosure curr =
      let
        val next = Mat.++ (one, Mat.** (curr, m))

      in
        if Mat.== (curr, next) then curr
        else findClosure next
      end
  in
    findClosure one
  end
end

```

2 Inductive definitions [25 pts]

Question 1. [5 pts] Consider a system consisting of the following inference rules where $n \text{ nat}$ is a judgment meaning that n is a natural number:

$$\frac{}{0 \text{ nat}} \textit{Zero} \quad \frac{n \text{ nat}}{S \ n \text{ nat}} \textit{Succ}$$

Give an inference rule that is derivable:

$$\frac{\frac{n \text{ nat}}{S \ n \text{ nat}} \textit{Succ2}}{} \textit{Succ2}$$

Given an inference rule that is admissible, but not derivable:

$$\frac{\frac{S \ n \text{ nat}}{n \text{ nat}} \textit{Succ}^{-1}}{} \textit{Succ}^{-1}$$

Question 2. [20 pts] Consider the following system from the Course Notes where $s \text{ lparen}$ means that s is a string of matched parentheses.

$$\frac{}{\epsilon \text{ lparen}} \textit{Leps} \quad \frac{s_1 \text{ lparen} \quad s_2 \text{ lparen}}{(s_1) s_2 \text{ lparen}} \textit{Lseq}$$

Prove the following theorem. The proof does not proceed by rule induction on the judgment $\underbrace{((\dots(s \text{ lparen}))}_k$.

- Fill in the blank. Use as much space as you need.
- As is conventional in the Course Notes, place *conclusion* in the left and *justification* in the right.

Theorem 2.1. For any string s , if $\underbrace{((\dots(s \text{ lparen}))}_k$, then $\underbrace{((\dots(()s \text{ lparen}))}_k$.

Proof. By mathematical induction on k .

Case $k = 0$:

$s \text{ lparen}$	assumption
$\frac{\frac{}{\epsilon \text{ lparen}} \textit{Leps} \quad \frac{\textit{assumption}}{s \text{ lparen}}}{() s \text{ lparen}} \textit{Lseq}$	

Case $k = n$ where $n > 0$:

$$\begin{array}{c}
 \underbrace{((\dots(s \text{ lparen} \\
 n \\
 \hline
 \underbrace{((\dots(s = (s_1)s_2 \text{ and } s_1 \text{ lparen and } s_2 \text{ lparen} \\
 n \\
 \hline
 \underbrace{(\dots(s = s_1)s_2}_{n-1} \qquad \text{from } \underbrace{((\dots(s = (s_1)s_2)}_n \\
 \hline
 s = s'_1s_2 \text{ and } \underbrace{(\dots(s'_1 = s_1}_{n-1} \qquad \text{from } \underbrace{(\dots(s = s_1)s_2}_{n-1} \\
 \hline
 \underbrace{(\dots((s'_1 \text{ lparen} \\
 n \qquad \text{by IH on } s_1 \text{ lparen} = \underbrace{(\dots(s'_1 \text{ lparen} \\
 n-1 \\
 \hline
 \underbrace{(\dots((s'_1 \text{ lparen } s_2 \text{ lparen} \\
 n \\
 \hline
 \underbrace{((\dots((s'_1) s_2 \text{ lparen}}_n \quad Lseq \\
 \hline
 \underbrace{((\dots((s \text{ lparen} \\
 n \qquad \text{from } \underbrace{((\dots((s'_1) s_2 \text{ lparen}}_n \text{ and } s = s'_1s_2 \\
 \hline
 \square
 \end{array}$$

3 λ -Calculus [55 pts]

Question 1. [5 pts] Show the reduction sequence under the call-by-name strategy. Underline the redex at each step.

$$\begin{aligned}
 & ((\lambda x_1. x_1) (\lambda x_2. x_2)) ((\lambda x_3. x_3) (\lambda z. z z)) \\
 \mapsto & \underline{(\lambda x_2. x_2) ((\lambda x_3. x_3) (\lambda z. z z))} \\
 \mapsto & \underline{(\lambda x_3. x_3) (\lambda z. z z)} \\
 \mapsto & \underline{(\lambda z. z z)}
 \end{aligned}$$

Question 2. [5 pts] Complete the inductive definition of substitution. You may use $[x \leftrightarrow y]e$ for the expression obtained by replacing all occurrences of x in e by y and all occurrences of y in e by x .

$$[e/x]x = \underline{e}$$

$$[e/x]y = \underline{y} \quad \text{if } x \neq y$$

$$[e/x](e_1 e_2) = \underline{[e/x]e_1 [e/x]e_2}$$

$$[e'/x]\lambda x. e = \underline{\lambda x. e}$$

$$[e'/x]\lambda y. e = \underline{\lambda y. [e'/x]e} \quad \text{if } x \neq y, y \notin FV(e')$$

$$[e'/x]\lambda y. e = \lambda z. \underline{[e'/x][y \leftrightarrow z]e} \quad \begin{array}{l} \text{if } x \neq y, y \in FV(e') \\ \text{where } z \neq y, z \notin FV(e), z \neq x, z \notin FV(e') \end{array}$$

Question 3. [5 pts] A Church numeral encodes a natural number n as a λ -abstraction \widehat{n} which takes a function f and returns $f^n = f \circ f \cdots \circ f$ (n times):

$$\widehat{n} = \lambda f. f^n = \lambda f. \lambda x. f f f \cdots f x$$

Define an exponentiation function `exp` such that `exp \widehat{m} \widehat{n}` evaluates to $\widehat{m^n}$.

$$\text{exp} = \underline{\lambda m. \lambda n. n m}$$

Question 4. [10 pts] Define a function `halve` which halves a given natural number (encoded as a Church numeral):

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

You may use the following pre-defined constructs: `zero`, `succ`, and `pair/fst/snd`.

- `zero` encodes the natural number zero.

$$\text{zero} = \widehat{0} = \lambda f. \lambda x. x$$

- `succ` finds the successor of a given natural number.

$$\text{succ} = \lambda \widehat{n}. \lambda f. \lambda x. \widehat{n} f (f x)$$

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

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

$$\text{halve} = \underline{\lambda \widehat{n}. \text{fst} (\widehat{n} (\lambda p. \text{pair} (\text{snd } p) (\text{succ} (\text{fst } p)))) (\text{pair } \text{zero } \text{zero})}$$

Question 5. [10 pts] This question assumes types `var` and `expr` that we have seen in Assignment 4:

```

type var = string
datatype exp =
  Var of var
| Lam of var * exp
| App of exp * exp

```

Suppose that we have two functions `subst` and `isValue`:

- `subst : expr -> var -> expr -> expr`
`subst e' x e` returns $[e'/x]e$.
- `isValue : expr -> bool`
`isValue e` returns `true` if e is a value and `false` otherwise.

Below is a function `step` of type `expr -> expr` such that `step e` returns e' if e reduces to e' and raises `Stuck` otherwise.

```

fun step (App (Lam (x, e), e2)) =
  if isValue e2 then subst e2 x e
  else App (Lam (x, e), step e2)
| step (App (e1, e2)) =
  if isValue e2 then App (step e1, e2)
  else App (e1, step e2)
| step _ = raise Stuck

```

We write $e \mapsto e'$ if e reduces to e' . Give exactly three reduction rules corresponding to the above definition of `step`.

$$\frac{e_2 \mapsto e'_2}{e_1 e_2 \mapsto e_1 e'_2} \qquad
 \frac{e_1 \mapsto e'_1}{e_1 v \mapsto e'_1 v} \qquad
 \frac{}{(\lambda x:A. e) v \mapsto [v/x]e}$$

Question 6. [5 pts] Convert the following expression to a de Bruijn expression.

$\lambda x. \lambda y. (\lambda z. (\lambda u. x y z u) (x y z)) (\lambda w. w)$

$$\equiv_{\text{dB}} \underline{\lambda. \lambda. (\lambda. (\lambda. 3 \ 2 \ 1 \ 0) (2 \ 1 \ 0)) (\lambda. 0)}$$

Question 7. [5 pts] Following is the definition of de Bruijn expressions:

$$\begin{array}{ll} \text{de Bruijn expression} & M ::= n \mid \lambda. M \mid M M \\ \text{de Bruijn index} & n ::= 0 \mid 1 \mid 2 \mid \dots \end{array}$$

Complete the definition of $\tau_i^n(N)$, as given in the Course Notes, for shifting by n (i.e., incrementing by n) all de Bruijn indexes in N corresponding to free variables, where a de Bruijn index m in N such that $m < i$ does not count as a free variable.

$$\begin{array}{ll} \tau_i^n(N_1 \ N_2) & = \underline{\tau_i^n(N_1) \ \tau_i^n(N_2)} \\ \tau_i^n(\lambda. N) & = \underline{\lambda. \tau_{i+1}^n(N)} \\ \tau_i^n(m) & = \underline{m + n} && \text{if } m \geq i \\ \tau_i^n(m) & = \underline{m} && \text{if } m < i \end{array}$$

Question 8. [10 pts] Define a mapping $FV(M)$ that finds the set of de Bruijn indexes corresponding to free variables in M . Here are a few examples:

- $FV(\lambda. 0 \ 1 \ 2) = \{1, 2\}$
- $FV(\lambda. \lambda. 0 \ 1 \ 2) = \{2\}$
- $FV(\lambda. 0 \ 1 \ (\lambda. 0 \ 2)) = \{1, 2\}$
- $FV(\lambda. \lambda. \lambda. 0 \ 1 \ 2) = \{\}$

Perhaps you will need an auxiliary function and use it in the definition of $FV(M)$. If you introduce an auxiliary function, briefly state its meaning.

$$\begin{array}{ll} FV(M) & = \underline{FV_0(M)} \\ \underline{FV_i(M_1 \ M_2)} & = \underline{FV_i(M_1) \cup FV_i(M_2)} \\ \underline{FV_i(\lambda. M)} & = \underline{FV_{i+1}(M)} \\ \underline{FV_i(m)} & = \underline{\{m\} \text{ if } m \geq i} \\ \underline{FV_i(m)} & = \underline{\{\} \text{ if } m < i} \end{array}$$

4 Simply-typed λ -calculus [25 pts]

Question 1. [10 pts] We use the following reduction and typing judgments in formulating the semantics of the simply-typed λ -calculus:

$$\begin{aligned} e \mapsto e' &\Leftrightarrow e \text{ reduces to } e' \\ \Gamma \vdash e : A &\Leftrightarrow \text{expression } e \text{ has type } A \text{ under typing context } \Gamma \end{aligned}$$

State the weakening property of typing judgments:

(Weakening). $\frac{\Gamma \vdash e : C}{\Gamma, x : A \vdash e : C}$.

State two theorems, progress and type preservation, constituting type safety:

(Progress).

$\frac{\text{If } \cdot \vdash e : A \text{ for some type } A,}{}$

$\text{then either } e \text{ is a value or there exists } e' \text{ such that } e \mapsto e'.$

(Type preservation).

$\frac{\text{If } \Gamma \vdash e : A \text{ and } e \mapsto e', \text{ then } \Gamma \vdash e' : A.}{}$

Question 2. [5 pts] Consider the extension of the simply-typed λ -calculus with sum types:

$$\begin{array}{ll} \text{type} & A ::= \dots \mid A + A \\ \text{expression} & e ::= \dots \mid \text{inl}_A e \mid \text{inr}_A e \mid \text{case } e \text{ of inl } x. e \mid \text{inr } x. e \end{array}$$

Write the typing rule for $\text{case } e \text{ of inl } x. e \mid \text{inr } x. e$:

$$\frac{\Gamma \vdash e : A_1 + A_2 \quad \Gamma, x_1 : A_1 \vdash e_1 : C \quad \Gamma, x_2 : A_2 \vdash e_2 : C}{\Gamma \vdash \text{case } e \text{ of inl } x_1. e_1 \mid \text{inr } x_2. e_2 : C} \text{+E}$$

Question 3. [5 pts] Specify the lazy reduction strategy for the constructs for sum types. You should extend the definition of values and give reduction rules that maintain type safety.

$$\text{value } v ::= \dots \mid \underline{\text{inl}_A e \mid \text{inr}_A e}$$

$$\frac{e \mapsto e'}{\text{case } e \text{ of inl } x_1. e_1 \mid \text{inr } x_2. e_2 \mapsto \text{case } e' \text{ of inl } x_1. e_1 \mid \text{inr } x_2. e_2}$$

$$\frac{}{\text{case inl}_A e \text{ of inl } x_1. e_1 \mid \text{inr } x_2. e_2 \mapsto [e/x_1]e_1}$$

$$\frac{}{\text{case inr}_A e \text{ of inl } x_1. e_1 \mid \text{inr } x_2. e_2 \mapsto [e/x_2]e_2}$$

Question 4. [5 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 .

$$\underline{\text{fix } f : A \rightarrow B. \lambda x : A. e}$$

5 Substitution [20 pts]

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

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

$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$	Var	$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \rightarrow I$	$\rightarrow E$	$\frac{\Gamma \vdash e : A \rightarrow B \quad \Gamma \vdash e' : A}{\Gamma \vdash e e' : B}$
$\frac{e_1 \mapsto e'_1}{e_1 e_2 \mapsto e'_1 e_2}$	Lam	$\frac{e_2 \mapsto e'_2}{(\lambda x : A. e) e_2 \mapsto (\lambda x : A. e) e'_2}$	Arg	$\frac{}{(\lambda x : A. e) v \mapsto [v/x]e}$ App

Fill in the blank 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.

Lemma 5.1 (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 consider only two cases shown below. In the first 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 $\frac{\Gamma, x : A, y : C_1 \vdash e'' : C_2}{\Gamma, x : A \vdash \lambda y : C_1. e'' : C_1 \rightarrow C_2} \rightarrow I$ where $e' = \lambda y : C_1. e''$ and $C = C_1 \rightarrow C_2$:

$\Gamma, y : C_1 \vdash [e/x]e'' : C_2$	by induction hypothesis
$\Gamma \vdash \lambda y : C_1. [e/x]e'' : C_1 \rightarrow C_2$	by the rule $\rightarrow I$
$[e/x]\lambda y : C_1. e'' = \lambda y : C_1. [e/x]e''$	from $y \notin FV(e)$ and $x \neq y$
$\Gamma \vdash [e/x]\lambda y : C_1. e'' : C_1 \rightarrow C_2$	

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 $e' = e_1 e_2$:

$\Gamma \vdash [e/x]e_1 : B \rightarrow C$	by IH on $\Gamma, x : A \vdash e_1 : B \rightarrow C$
$\Gamma \vdash [e/x]e_2 : B$	by IH on $\Gamma, x : A \vdash e_2 : B$
$\Gamma \vdash [e/x]e_1 [e/x]e_2 : C$	by the rule $\rightarrow E$
$\Gamma \vdash [e/x](e_1 e_2) : C$	from $[e/x](e_1 e_2) = [e/x]e_1 [e/x]e_2$

□