CSE-321 Programming Languages 2009
Midterm — Sample Solution

<table>
<thead>
<tr>
<th></th>
<th>Prob 1</th>
<th>Prob 2</th>
<th>Prob 3</th>
<th>Prob 4</th>
<th>Prob 5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>15</td>
<td>25</td>
<td>55</td>
<td>25</td>
<td>20</td>
<td>140</td>
</tr>
</tbody>
</table>
1 SML Programming [15 pts]

Question 1. [5 pts] Give a tail-recursive implementation of \texttt{fib} for computing Fibonacci numbers.

(Type) \texttt{fib: int -> int}

(Description)
\texttt{fib} \textit{n} returns \texttt{fib} \textit{(n-1)} + \texttt{fib} \textit{(n-2)} when \textit{n} \geq 2.
\texttt{fib} \textit{n} returns 1 if \textit{n} = 0 or \textit{n} = 1.

(Invariant) \textit{n} \geq 0.

fun fib \textit{n} =  
let  
fun fib’ \textit{a} \textit{b} \textit{0} = \textit{a}  
| fib’ \textit{a} \textit{b} \textit{n} = fib’ \textit{b} \textit{(a + b)} \textit{(n - 1)}  
in  
fib’ \textit{1} \textit{1} \textit{n}  
end

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

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

• \textit{t} denotes the type of square matrices.
• \texttt{identity} \textit{n} returns an identity matrix of dimension \textit{n}.
• \texttt{dim} \textit{A} returns the dimension of matrix \textit{A}.
• \texttt{++} \texttt{(A1, A2)} adds two matrices \textit{A1} and \textit{A2}.
• \texttt{**} \texttt{(A1, A2)} multiplies two matrices \textit{A1} and \textit{A2}.
• \texttt{==} \texttt{(A1, A2)} returns \texttt{true} if two matrices \textit{A1} and \textit{A2} are equal and \texttt{false} otherwise.

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

Implement the functor \texttt{ClosureFn} where the member \texttt{closure} computes the closure of a given matrix. \texttt{closure} \textit{A} should terminate if \textit{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:

\[
\begin{align*}
0 \text{ nat} \quad & \quad \text{Zero} \\
\frac{n \text{ nat}}{S \, n \text{ nat}} \quad & \quad \text{Succ}
\end{align*}
\]

Give an inference rule that is derivable:

\[
\frac{n \text{ nat}}{S \, S \, n \text{ nat}} \quad \text{Succ2}
\]

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

\[
\frac{S \, n \text{ nat}}{n \text{ nat}} \quad \text{Succ}^{-1}
\]

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.

\[
\begin{align*}
\epsilon \text{ lparen} \quad & \quad \text{Leps} \\
\frac{s_1 \text{ lparen} \quad s_2 \text{ lparen}}{(s_1) \quad s_2 \text{ lparen}} \quad & \quad \text{Lseq}
\end{align*}
\]

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

- 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 \( (\cdots (s \text{ lparen}, \) then \( (\cdots ()s \text{ lparen).

**Proof.** By mathematical induction on \( k \).

Case \( k = 0 \):

\[
\frac{s \text{ lparen}}{\text{assumption}}
\]

\[
\frac{\epsilon \text{ lparen} \quad \text{Leps} \quad \text{assumption} \quad s \text{ lparen}}{(s) \text{ lparen} \quad \text{Lseq}}
\]
Case $k = n$ where $n > 0$:

$$((\cdots (s \text{lparen}) \text{n}) \text{assumption}) \text{ by inversion and the rule Lseq}$$

$$((\cdots (s = (s_1)s_2) \text{lparen} \text{and} s_1 \text{lparen} \text{and} s_2 \text{lparen}) \text{n-1} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$$((\cdots (s = s_1)s_2) \text{n-1} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$$s = s_1's_2) \text{and} (\cdots (s_1 = s_1) \text{n-1} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$$((\cdots (())s_1 \text{lparen} \text{by IH on} s_1 \text{lparen} = (\cdots (s_1's_1 \text{lparen} \text{n-1} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$$((\cdots (())s_1 \text{lparen} s_2 \text{lparen} \text{Lseq (}((\cdots (())s_1 \text{lparen}) \text{n-1} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$$((\cdots (())s \text{lparen} \text{from (}((\cdots (())s_1 \text{lparen} \text{and} s = s_1's_2) \text{n} \text{from (}((\cdots (s = (s_1)s_2) \text{n})$$

$\square$
\section{\lam-Calculus [55 pts]}

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

\[
\begin{align*}
((\lambda x_1.x_1)(\lambda x_2.x_2)) ((\lambda x_3.x_3)(\lambda z.z.z)) & \quad \mapsto \quad (\lambda x_2.x_2)((\lambda x_3.x_3)(\lambda z.z.z)) \\
(\lambda x_3.x_3)(\lambda z.z.z) & \quad \mapsto \quad (\lambda z.z.z)
\end{align*}
\]

\textbf{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 $x$.

\[
[\ForEach x.e] = \begin{cases} 
  e & \text{if } x \neq y \\
\end{cases}
\]

\[
[\ForEach y.e] = \begin{cases} 
  y & \text{if } x \neq y \\
\end{cases}
\]

\[
[\ForEach [e_1.e_2]] = \begin{cases} 
  [\ForEach [e_1]] [\ForEach [e_2]] & \text{if } x \neq y \\
\end{cases}
\]

\[
[\ForEach \lambda x.e] = \lambda x. [\ForEach e] 
\]

\[
[\ForEach \lambda y.e] = \lambda y. [\ForEach [e_1]] & \text{if } x \neq y, y \notin \text{FV}(e_1) \\
\]

\[
[\ForEach \lambda y.e] = \lambda y. [\ForEach [e_1]] [y\rightarrow z] & \text{if } x \neq y, y \notin \text{FV}(e_1), z \neq x, z \notin \text{FV}(e_1) \\
\]

6
Question 3. [5 pts] A Church numeral encodes a natural number \( n \) as a \( \lambda \)-abstraction \( \hat{n} \) which takes a function \( f \) and returns \( f^n = f \circ f \cdots \circ f \) (\( n \) times):

\[
\hat{n} = \lambda f. f^n = \lambda f. \lambda x. f f f \cdots f x
\]

Define an exponentiation function \( \exp \) such that \( \exp \hat{m} \hat{n} \) evaluates to \( \hat{m^n} \).

\[
\exp = \lambda m. \lambda n. n \ m
\]

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

- \( \text{halve} \) \( \hat{2} \ast k \) returns \( \hat{k} \).
- \( \text{halve} \) \( \hat{2} \ast k + 1 \) returns \( \hat{k} \).

You may use the following pre-defined constructs: \( \text{zero} \), \( \text{succ} \), and \( \text{pair}/\text{fst}/\text{snd} \).

- \( \text{zero} \) encodes the natural number zero.
  \[
  \text{zero} = \hat{0} = \lambda f. \lambda x. x
  \]

- \( \text{succ} \) finds the successor of a given natural number.
  \[
  \text{succ} = \lambda \hat{n}. \lambda f. \lambda x. \hat{n} f (f x)
  \]

- \( \text{pair} \) creates a pair of two expressions, and \( \text{fst} \) and \( \text{snd} \) are projection operators.
  \[
  \text{pair} = \lambda x. \lambda y. \lambda b. b x y
  \]
  \[
  \text{fst} = \lambda p. (\lambda t. \lambda f. t)
  \]
  \[
  \text{snd} = \lambda p. (\lambda t. \lambda f. t)
  \]

\[
\text{halve} = \lambda \hat{n}. \text{fst} (\hat{n} (\lambda p. \text{pair} (\text{snd} p) (\text{succ} (\text{fst} p))))(|\text{pair zero zero}|)
\]
Question 5. [10 pts] This question assumes types \texttt{var} and \texttt{expr} that we have seen in Assignment 4:

\begin{verbatim}
  type var = string
  datatype exp =
   Var of var
   | Lam of var * exp
   | App of exp * exp
\end{verbatim}

Suppose that we have two functions \texttt{subst} and \texttt{isValue}:

- \texttt{subst : expr -> var -> expr -> expr}
  
  \texttt{subst e' x e} returns \([e'/x]e\).

- \texttt{isValue : expr -> bool}
  
  \texttt{isValue e} returns true if \(e\) is a value and false otherwise.

Below is a function \texttt{step} of type \texttt{expr -> expr} such that \texttt{step e} returns \(e'\) if \(e\) reduces to \(e\) and raises \texttt{Stuck} otherwise.

\begin{verbatim}
  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
\end{verbatim}

We write \(e \mapsto e'\) if \(e\) reduces to \(e'\). Give exactly three reduction rules corresponding to the above definition of \texttt{step}.

\begin{align*}
  e_2 \mapsto e'_2 & \quad e_1 \mapsto e'_1 & \quad (\lambda x : A. e) \mapsto [v/x] e
\end{align*}

\[ \quad e_1 \ e_2 \mapsto e_1 \ e'_2 \quad e_1 \ v \mapsto e'_1 \ v \]
Question 6. [5 pts] Convert the following expression to a de Bruijn expression.

\[ \lambda x. \lambda y. (\lambda z. (\lambda u. x y z u) (x y z)) (\lambda w. w) \]

\[ \equiv_{dB} \lambda. \lambda. (\lambda. (\lambda. 3 2 1 0) (2 1 0)) (\lambda. 0) \]

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

\[
\text{de Bruijn expression} \quad M ::= n \mid \lambda. M \mid M M \\
\text{de Bruijn index} \quad n ::= 0 \mid 1 \mid 2 \mid \cdots
\]

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.

\[
\tau_i^n(N_1 N_2) = \tau_i^n(N_1) \tau_i^n(N_2) \\
\tau_i^n(\lambda. N) = \lambda. \tau_{i+1}^n(N) \\
\tau_i^n(m) = m + n \quad \text{if } m \geq i \\
\tau_i^n(m) = m \quad \text{if } m < i
\]

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.

\[
FV(M) = FV_0(M) \\
FV_i(M_1 M_2) = FV_i(M_1) \cup FV_i(M_2) \\
FV_i(\lambda. M) = FV_{i+1}(M) \\
FV_i(m) = \{m\} \text{ if } m \geq i \\
FV_i(m) = \{\} \text{ if } m < i
\]
4 Simply-typed $\lambda$-calculus [25 pts]

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

\[
\begin{align*}
    e \mapsto e' & \quad \iff \quad e \text{ reduces to } e' \\
    \Gamma \vdash e : A & \quad \iff \quad \text{expression } e \text{ has type } A \text{ under typing context } \Gamma
\end{align*}
\]

State the weakening property of typing judgments:

(Weakening). If $\Gamma \vdash e : C$, then $\Gamma, x : A \vdash e : C$.

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

(Progress).

If $\cdot \vdash e : A$ for some type $A$, then either $e$ is a value or there exists $e'$ such that $e \mapsto e'$.

(Type preservation).

If $\Gamma \vdash e : A$ and $e \mapsto e'$, then $\Gamma \vdash e' : A$.

**Question 2.** [5 pts] Consider the extension of the simply-typed $\lambda$-calculus with sum types:

\[
\begin{align*}
    \text{type} & \quad A \ ::= \quad \cdots \quad | \quad A + A \\
    \text{expression} & \quad e \ ::= \quad \cdots \quad | \quad \text{inl}_A e \quad | \quad \text{inr}_A e \quad | \quad \text{case } e \text{ of } \text{inl } x. e \quad | \quad \text{inr } x. e
\end{align*}
\]

Write the typing rule for case $e$ of inl $x. e$ | inr $x. e$:

\[
\begin{array}{c}
\Gamma \vdash e : A_1 + A_2 \\
\Gamma, x_1 : A_1 \vdash e_1 : C \\
\Gamma, x_2 : A_2 \vdash e_2 : C
\end{array}
\quad +E
\begin{array}{c}
\Gamma \vdash \text{case } e \text{ of } \text{inl } x_1. e_1 \mid \text{inr } x_2. e_2 : C
\end{array}
\]
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.

\[
\text{name} \quad v ::= \cdots \mid \text{inl}_A e \mid \text{inr}_A e
\]

\[
e \mapsto e'
\]

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

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

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

Question 4. [5 pts] Give an expression in the extended simply typed \(\lambda\)-calculus that denotes a recursive function \(f\) of type \(A \rightarrow B\) whose formal argument is \(x\) and whose body is \(e\).

\[
\text{fix } f : A \rightarrow B. \lambda x : A. e
\]
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     | $A ::= P | A \rightarrow A$ |
|----------|----------------------------|
| base type| $P$                        |
| expression| $e ::= x | \lambda x : A. e | e e$ |
| value    | $v ::= \lambda x : A. e$   |
| typing context| $\Gamma ::= \cdot | \Gamma, x : A$ |

\[
\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A. e : A \rightarrow B} \quad \frac{\Gamma \vdash e : A \rightarrow B \quad \Gamma \vdash e' : A}{\Gamma \vdash e : \Gamma} \quad \frac{(\lambda x : A. e) e_2 \mapsto (\lambda x : A. e) e'_2}{\text{App}}
\]

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

\[
\frac{\text{Arg} \quad (\lambda x : A. e) v \mapsto [v/x]e}{\text{Var}}
\]

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 $y \notin \text{FV}(e)$ and $y \neq x$. If $y \in \text{FV}(e)$ or $y = x$, we can always choose a different variable by applying an α-conversion to $\lambda y : C_1. e''$.

**Case** $\frac{\Gamma, y : C_1 \vdash [e/x]e'' : C_2}{\Gamma, x : A \vdash \lambda y : C_1. e'' : C_1 \rightarrow C_2} \rightarrow I$ where $e' = \lambda y : C_1. e''$ and $C = C_1 \rightarrow C_2$:

\[
\frac{\Gamma, y : C_1 \vdash [e/x]e'' : C_2}{\Gamma \vdash \lambda y : C_1. [e/x]e'' : C_1 \rightarrow C_2} \quad \text{by induction hypothesis}
\]

\[
\frac{\Gamma \vdash \lambda y : C_1. [e/x]e'' : C_1 \rightarrow C_2}{\Gamma \vdash [e/x] \lambda y : C_1. e'' : C_1 \rightarrow C_2} \quad \text{by the rule } \rightarrow I
\]

\[
\frac{\Gamma \vdash [e/x] \lambda y : C_1. e'' : C_1 \rightarrow C_2}{\Gamma \vdash [e/x] \lambda y : C_1. e'' : C_1 \rightarrow C_2} \quad \text{from } y \notin \text{FV}(e) \quad \text{and } x \neq y
\]

**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 $e' = e_1 e_2$:

\[
\frac{\Gamma \vdash [e/x] e_1 : B \rightarrow C}{\Gamma \vdash [e/x] e_1 : B \rightarrow C} \quad \text{by IH on } \Gamma, x : A \vdash e_1 : B \rightarrow C
\]

\[
\frac{\Gamma \vdash [e/x] e_2 : B}{\Gamma \vdash [e/x] e_2 : B} \quad \text{by IH on } \Gamma, x : A \vdash e_2 : B
\]

\[
\frac{\Gamma \vdash [e/x] e_1 [e/x] e_2 : C}{\Gamma \vdash [e/x] e_1 [e/x] e_2 : C} \quad \text{by the rule } \rightarrow E
\]

\[
\frac{\Gamma \vdash [e/x] (e_1 e_2) : C}{\Gamma \vdash [e/x] (e_1 e_2) : C} \quad \text{from } [e/x] (e_1 e_2) = [e/x] e_1 [e/x] e_2
\]