Change History

9am 1/03 Initial Version

Do not consult other students in working on this examination. If you have questions you may address them to the professor. You may consult the textbook, lecture notes, lecture slides, other reference material, and your own homeworks and solutions to homeworks provided by the professor.

Before turning in the exam, write and sign the following statement:

This paper and my electronic submissions represents my own work in accordance with University regulations.
Problem 1  [10 points] Consider the following language that supports both continuations and exceptions.

Variables  \( x \ ::= \ldots \)

Naturals  \( n \in \mathbb{N} \)

Types  \( \tau ::= \text{int} \mid \tau \text{ cont} \)

Expressions  \( e ::= n \mid + (e_1, e_2) \)

\( \mid \text{fail} \mid \text{try } e_1 \text{ ow } e_2 \)

\( \mid \text{throw } e_1 \text{ to } e_2 \mid \text{letcc } x \text{ in } e \mid K \)

Values  \( v ::= n \mid K \)

Frames  \( F ::= +(v_1, \square) \mid +(\square, e_2) \)

\( \mid \text{try } \square \text{ ow } e_2 \)

\( \mid \text{throw } v_1 \text{ to } \square \mid \text{throw } \square \text{ to } e_2 \)

Stacks  \( K ::= \cdot \mid F \triangleright K \)

\[ (K, +(e_1, e_2)) \mapsto +(+(e_2) \triangleright K, e_1) \text{ add1} \]

\[ (+(e_2) \triangleright K, v_1) \mapsto +(v_1, \square) \triangleright K, e_2) \text{ add2} \]

\[ n = n_1 + n_2 \]

\[ (+(n_1, \square) \triangleright K, n_2) \mapsto (K, n) \text{ add3} \]

\[ (K, \text{try } e_1 \text{ ow } e_2) \mapsto (\text{try } \square \text{ ow } e_2 \triangleright K, e_1) \text{ push} \]

\[ (F \neq \text{try } \square \text{ ow } e_2) \mapsto (K, \text{fail}) \text{ unwind} \]

\[ (\text{try } \square \text{ ow } e_2 \triangleright K, \text{fail}) \mapsto (K, e_2) \text{ catch} \]

\[ (K, \text{letcc } x \text{ in } e) \mapsto (K, [x \leftarrow K][e]) \text{ freeze} \]

\[ (K, \text{throw } e_1 \text{ to } e_2) \mapsto (\text{throw } \square \text{ to } e_2 \triangleright K, e_1) \text{ throw1} \]

\[ (\text{throw } \square \text{ to } e_2 \triangleright K, v_1) \mapsto (\text{throw } v_1 \text{ to } \square \triangleright K, e_2) \text{ throw2} \]

For each of the following expressions reduce the expression to a final state. Show the reduction step by step and label each step. For example:

\[ (. \text{, } +(1, 2), 3) \]

\[ \mapsto (+(1, 2) \triangleright . \text{, } +(1, 2)) \text{ add1} \]

\[ \mapsto (+((1, 2) \triangleright +(1, 2)) \triangleright , 1) \text{ add1} \]

\[ \mapsto (+((1, 2) \triangleright +(1, 2)) \triangleright , 2) \text{ add2} \]

\[ \mapsto +(3, 2) \triangleright , 3 \text{ add3} \]

\[ \mapsto (, 6) \text{ add3} \]

(a)  [1 point] (. , try + (+ (fail, 2), 3) ow 0)

(b)  [1 point] (. , +(1, letcc x in 2))

(c)  [2 points] (. , letcc x in try + (1, fail) ow (throw 0 to x))

(d)  [2 points] (. , try (letcc x in (try + (1, fail) ow (throw 2 to x))) ow 3)

(e)  [4 points] The language presented so far uses an inefficient implementation of exceptions. For this part of the problem you will complete the reformulation of the operational semantics so that it uses two
stacks. One stack represents the normal control stack and the other represents a stack of exception handlers. To do this properly expressions that deal with first-class continuations must be modified so they capture both the normal control stack as well as the handler stack. The semantics presented below is almost complete fill in the missing details marked with four in total.

Variables  
\( x ::= \ldots \)

Naturals  
\( n \in \mathbb{N} \)

Types  
\( \tau ::= \text{int} | \tau \text{cont} \)

Expressions  
\( e ::= n | +((e_1,e_2)) | \text{fail} | \text{try} e_1 \text{ow} e_2 | \text{throw} e_1 \text{to} e_2 \) 
\( | \text{letcc} \; x \in e \mid \text{CS} \)

Values  
\( v ::= n | \text{CS} \)

Frames  
\( F ::= +(v_1,\square) \mid +(\square, e_2) \) 
\( | \text{try} \; \square \text{ow} \; e_2 \) 
\( | \text{throw} \; v_1 \text{to} \; \square \mid \text{throw} \; \square \text{to} \; e_2 \)

Control Stacks  
\( K ::= \cdot \mid F \triangleright K \)

Handler Stacks  
\( H ::= \cdot \mid (K,e) \triangleright H \)

Control State  
\( CS ::= ??1?? \)

\[
\begin{align*}
(H,K,+(e_1,e_2)) & \mapsto (H,+(\square, e_2) \triangleright K, e_1) & \text{add1} \\
(H,+(\square, e_2) \triangleright K, v_1) & \mapsto (H,+(v_1,\square) \triangleright K, e_2) & \text{add2} \\
(H,+(n_1,\square) \triangleright K, n_2) & \mapsto (H,K,n) & \text{add3}
\end{align*}
\]

\[
\begin{align*}
(H,K,\text{try} e_1 \text{ow} e_2) & \mapsto ((K,e_2) \triangleright H,\text{try} \; \square \text{ow} \; e_2 \triangleright K, e_1) & \text{push} \\
((K,e_2) \triangleright H,\text{try} \; \square \text{ow} \; e_2 \triangleright K, v) & \mapsto (H,K,v) & \text{pop} \\
((K',e') \triangleright H,K,\text{fail}) & \mapsto (H,K',e') & \text{raise} \\
(H,K,\text{letcc} \; x \in e) & \mapsto (H,K,[x \gets ??2??]e) & \text{freeze} \\
(H,\text{throw} \; v \text{to} \; \square \triangleright K, ??3??) & \mapsto ??4?? & \text{thaw} \\
(H,K,\text{throw} \; e_1 \text{to} e_2) & \mapsto (H,\text{throw} \; \square \text{to} \; e_2 \triangleright K, e_1) & \text{throw1} \\
(H,\text{throw} \; \square \text{to} \; e_2 \triangleright K, v_1) & \mapsto (H,\text{throw} \; v_1 \text{to} \; \square \triangleright K, e_2) & \text{throw2}
\end{align*}
\]

Problem 2  [25 points]  

The abstract syntax for regular expressions is given below

Naturals  
\( n \in \mathbb{N} \)

Atoms  
\( a, b \in \Sigma \)

Strings  
\( s ::= \epsilon | a | s_1 . s_2 \)

Regular Expressions  
\( re ::= a | re_1 + re_2 | re_1 \cdot re_2 | re^n | re^* \)
Let \( \Sigma \) be some undefined alphabet of atoms. The variables \( s \) stand for strings of atoms where \( \epsilon \) is the empty string \( s_1.s_2 \) is concatenation of the strings \( s_1 \) with \( s_2 \). We define equality on strings as follows:

\[
\begin{align*}
    a &= a \\
    \epsilon.s &= s \\
    s.\epsilon &= s \\
    (s_1.s_2).s_3 &= s_1.(s_2.s_3) \\
    s_1.s_2 &= s'_1.s'_2 \quad \text{if } s_1 = s'_1 \text{ and } s_2 = s'_2
\end{align*}
\]

Given the notion of equality on strings above we define a meaning function \( \mathcal{E} \) on regular expressions as follows:

\[
\begin{align*}
    \mathcal{E}[a] &\equiv \{ s \mid s = a \} \\
    \mathcal{E}[re_1 + re_2] &\equiv \{ s \mid s \in \mathcal{E}[re_1] \lor s \in \mathcal{E}[re_2] \} \\
    \mathcal{E}[re_1 . re_2] &\equiv \{ s \mid \exists s_1, s_2. s_1 \in \mathcal{E}[re_1] \land s_2 \in \mathcal{E}[re_2] \land s = s_1.s_2 \} \\
    \mathcal{E}[re^0] &\equiv \{ s \mid s = \epsilon \} \\
    \mathcal{E}[re^{n+1}] &\equiv \{ s \mid \exists s_1, s_2. s_1 \in \mathcal{E}[re] \land s_2 \in \mathcal{E}[re^n] \land s = s_1.s_2 \} \\
    \mathcal{E}[re^*] &\equiv \{ s \mid \exists n.s \in \mathcal{E}[re^n] \}
\end{align*}
\]

The following equations can be shown to hold from the semantic of regular expressions.

\[
\begin{align*}
    \mathcal{E}[re_1 + re_2] &\equiv \mathcal{E}[re_2 + re_1] \\
    \mathcal{E}[re_1 . (re_2 + re_3)] &\equiv \mathcal{E}[(re_1 . re_2) + (re_1 . re_3)] \\
    \mathcal{E}[re . re^n] &\equiv \mathcal{E}[re^{n+1}] \\
    \mathcal{E}[re . re^n] &\equiv \mathcal{E}[re^n . re]
\end{align*}
\]

Let us define the width of a regular expression as a pair of natural numbers that gives the length of the shortest string in the set defined by the regular expression and the length of the longest string in the set or \( \infty \) if there is no longest string. For example:

\[
\begin{align*}
    \mathcal{W}[a] &\equiv (1, 1) \\
    \mathcal{W}[a + \epsilon] &\equiv (0, 1) \\
    \mathcal{W}[a^n] &\equiv (n, n) \\
    \mathcal{W}[(a + \epsilon)^n] &\equiv (0, n) \\
    \mathcal{W}[(a + \epsilon)^*] &\equiv (0, \infty)
\end{align*}
\]

(a) [15 points] For this part of the problem you should define a semantic function, \( \mathcal{W} \), that computes the width of a regular expression. The mathematical functions \( \min \) and \( \max \) will be useful in your definition. Note that \( \max(n, \infty) = \infty \) and \( \min(n, \infty) = n \). The functions \( \text{fst} \) and \( \text{snd} \) where \( \text{fst}(x, y) = x \) and \( \text{snd}(x, y) = y \) will also be useful.

(b) [2 points] For the remaining problems show that each hold for the semantic function \( \mathcal{W} \) which you defined.

\[
\mathcal{W}[re_1 + re_2] \equiv \mathcal{W}[re_2 + re_1]
\]

(c) [2 points]

\[
\mathcal{W}[re_1 . (re_2 + re_3)] \equiv \mathcal{W}[(re_1 . re_2) + (re_1 . re_3)]
\]

(d) [3 points] Hint: There are two “obvious” ways of defining the semantics for \( \mathcal{W}[re^n] \). You should choose the definition that makes the next two rules easy to prove.

\[
\mathcal{W}[re . re^n] \equiv \mathcal{W}[re^{n+1}]
\]

4
(e) [3 points]

\[ \mathcal{W}[re \cdot re^n] \cong \mathcal{W}[re^n \cdot re] \]

Problem 3 [10 points] Using the following rules

\[
\begin{align*}
\tau &<: \tau \\
\tau_1 &<: \tau_3 \quad \tau_3 &<: \tau_2 \\
\tau_1 &<: \tau_2 \\
m &> n \\
\tau_1 \ast \ldots \ast \tau_m &<: \tau_1 \ast \ldots \ast \tau_n \\
\forall 1 \leq i \leq n \quad \tau_i' &<: \tau_i \\
\tau_i' \ast \ldots \ast \tau_n' &<: \tau_1 \ast \ldots \ast \tau_n \\
\forall 1 \leq i \leq n \quad \tau_i' &<: \tau_i \\
\{a : \tau_1, \ldots, z : \tau_m\} &<: \{a : \tau_1, \ldots, z : \tau_n\} \\
\{a : \tau_1', \ldots, z : \tau_n'\} &<: \{a : \tau_1, \ldots, z : \tau_n\} \\
\tau_1 &<: \tau_1' \\
\tau_2 &<: \tau_2 \\
\tau_i' &<: \tau_i \\
\tau_1' \rightarrow \tau_2' &<: \tau_1 \rightarrow \tau_2 \\
\tau' = \tau \\
\tau'' \ref &<: \tau \ref
\end{align*}
\]

and assuming

\[ D <: B \quad E <: B \quad B <: A \quad C <: A \]

where A, B, C, D, and E are distinct:

(a) [3 points] Define rules that allow for both width and depth subtyping for binary sums. The rules should describe valid relationships between the following pairs of types.

- \( \tau_1 \) and \( \tau_1 + \tau_2 \)
- \( \tau_2 \) and \( \tau_1 + \tau_2 \)
- \( \tau_1 + \tau_2 \) and \( \tau_3 + \tau_4 \)

(b) [1 point] Using your answer from Part (a) as well as the other subtyping rules given fill in the box with a type (A,B,C,D, or E) that will make the equation true or state that no such type will satisfy the equation.

\[ A \rightarrow (B \rightarrow D) <: E \rightarrow (\square \rightarrow B) \]

(c) [2 point]

\[ (A \ast A) \rightarrow (B \rightarrow (B \rightarrow B)) <: (A \ast B \ast E) \rightarrow (C \rightarrow (\square \rightarrow A)) \]

(d) [2 points]

\[ ((E \ast D \ast A) \rightarrow \{a : \square\}) \rightarrow C <: ((B \ast A) \rightarrow \{a : A, b : C\}) \rightarrow A \]

(e) [2 points]

\[ \square \rightarrow ((B + E) \ast A) \rightarrow D <: B \rightarrow (C \ast D) \rightarrow (B + E) \]

Problem 4 [10 points] Given the following ML datatypes

\begin{verbatim}
datatype int_tree = Leaf of int | Node of (int_tree * int * int_tree)
datatype bool_list = Nil | Cons of (bool * bool_list)
datatype result = None | Tree of int_tree | List of bool_list | Pair of (result * result)
\end{verbatim}

(a) [6 points] Write out types using the following language of types that are equivalent to the datatypes.

\[
\begin{align*}
\text{type vars} & \quad \text{tv} \ ::= \ldots \\
\tau & ::= \text{int} \mid \text{unit} \mid (\tau_1 + \tau_2) \mid (\tau_1 \ast \tau_2) \mid (\tau_1 \rightarrow \tau_2) \mid (\text{rec tv is } \tau) \mid \text{tv}
\end{align*}
\]

(b) [4 points] Given your representation of the type, convert the ML value to an equivalent value based on your answer to part (a). Use the syntax presented in Chapter 19 of Harper to represent your value.

\[ \text{Pair(List(Cons(true,Nil)),Tree(Leaf(3)))} \]
Problem 5  [10 points]

structure Set : sig
  type set
  1: val empty : set
  2: (* The empty set is the set S such that ∀ x. x ∉ S. *)
  3: val member : (int * set) -> bool (* The function member(x, S) returns true iff x ∈ S. *)
  4: val singleton : int -> set (* The function singleton(x) returns a set {x} *)
  5: val remove : (int * set) -> set (* The function remove(x, S) returns a set S' such that x ∉ S'. *)
  6: (* The function union(S1, S2) returns a set {x | x ∈ S1 ∨ x ∈ S2} *)
  7: val intersect : (set * set) -> set (* The function intersect(S1, S2) returns a set {x | x ∈ S1 ∧ x ∈ S2} *)
end

Given the SML signature above and the definitions used in Chapter 9 of Mitchell answer the following questions:

(a) [2 points] Which lines are part of the interface?
(b) [2 points] Which lines are part of the specification?
(c) [2 points] Which functions or constants are constructors?
(d) [2 points] Which functions or constants are operators?
(e) [2 points] Which functions or constants are observers?

Problem 6  [25 points] Below is the abstract syntax for a Featherweight-TAL. A simple typed-assembly language.

Variables  x ::= ...
Function Names  f ::= ...
Integers  i ∈ Z
Booleans  b ::= true | false
Values  v ::= b | i
Types  τ ::= int | bool
Operations  op ::= + | − | <
Primitive Expressions  pe ::= x | v
Expressions  e ::= let x = pe1 op pe2 in e
             | let x = f(pe) in e
             | if pe then e1 else e2
             | ret(pe)
Function Declaration  fd ::= fun f(x : τ1) → τ2 is e
Program P ::= fd; P | e

Notice that there are no complex expressions of the form (1 + (2 + 3)). Instead we would have to represent the above as the expression

let t1 = 2 + 3 in
let t2 = 1 + t1 in
ret (t2)
which names every temporary value that is computed in the left-to-right evaluation of \((1 + (2 + 3))\). A program to compute the 25th Fibonacci number is as follows:

```plaintext
fun fib(n:int):int is
    let test = n < 2 in
    if test then ret(1)
    else let t1 = n - 1 in
        let t2 = n - 2 in
        let t3 = fib(t1) in
        let t4 = fib(t2) in
        let t5 = t3 + t4 in
        ret t5;
    let ans = fib(25) in
    ret ans
```

The directory `ftal` contains a small-step operational semantics and a static semantics. For this problem you must reverse engineer the ML code in order to complete the mathematical description below. **Hint:** If you discover an ambiguity that cannot be answered by looking at the reference implementation ask for clarification. The code is not a complete mathematical specification and some “obvious” details must be added in the specification that are not explicitly in the code.

### Variable Typing

\[
\Gamma ::= \cdot \mid \Gamma[x \mapsto \tau]
\]

### Function Types

\[
\sigma ::= \tau_1 \rightarrow \tau_2
\]

### Function Typing

\[
\Phi ::= \cdot \mid \Phi[f \mapsto \sigma]
\]

### Function Table

\[
FT ::= \cdot \mid FT[f \mapsto fd]
\]

### Stack

\[
k ::= \cdot \mid (x, e) \triangleright k
\]

### Program State

\[
S ::= (FT, k, e)
\]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S \rightarrow S')</td>
<td>Single-step relation</td>
</tr>
<tr>
<td>(S \rightarrow^* S')</td>
<td>Multi-step relation</td>
</tr>
<tr>
<td>(\Gamma \vdash pe : \tau)</td>
<td>Well-formed primitive expressions</td>
</tr>
<tr>
<td>(\Phi; \Gamma \vdash e : \tau)</td>
<td>Well-formed expressions</td>
</tr>
<tr>
<td>(\Phi \vdash fd \ okay)</td>
<td>Well-formed function</td>
</tr>
<tr>
<td>(\Phi \vdash FT \ okay)</td>
<td>Well-formed function table</td>
</tr>
<tr>
<td>(\Phi \vdash k : \tau \ stack)</td>
<td>Well-formed stack</td>
</tr>
<tr>
<td>(\Phi \vdash S \ okay)</td>
<td>Well-formed program state</td>
</tr>
</tbody>
</table>

(a) \(S \rightarrow S'\) \[4 points\] **Complete the rules for the relation** \(S \rightarrow S'\)

(b) \(\Gamma \vdash pe : \tau\) \[3 points\] **Complete the rules for the relation** \(\Gamma \vdash pe : \tau\)

(c) \(\Phi; \Gamma \vdash e : \tau\) \[4 points\] **Complete the rules for the relation** \(\Phi; \Gamma \vdash e : \tau\)
Complete the rules for the relation \( \Phi \vdash fd \ ok \)

\[
\Phi \vdash FT \ ok
\]

\[
\Phi \vdash \cdot \ ok \quad \frac{\Phi \vdash fd \ ok \quad \Phi \vdash FT \ ok}{\Phi \vdash \cdot \ ok} \quad \frac{\Phi \vdash \cdot \ ok}{\Phi \vdash FT[fd \mapsto f] \ ok}
\]

\[
\Phi \vdash k : \tau \ stack
\]

\[
\Phi \vdash \cdot : \tau \ stack \quad \frac{\Phi ; \cdot \mapsto \tau \ stack \quad \Phi \vdash k : \tau' \ stack}{\Phi \vdash \cdot : \tau \ stack} \quad \frac{\Phi \vdash (x, e) \mapsto k : \tau \ stack}{\Phi \vdash (x, e) \mapsto k : \tau \ stack}
\]

\[
\Phi \vdash (FT, k, e) \ ok
\]

\[
\Phi \vdash FT \ ok \quad \Phi \vdash k : \tau \ stack \quad \Phi ; \vdash e : \tau \quad \frac{\Phi \vdash (FT, k, e) \ ok}{\Phi \vdash (FT, k, e) \ ok}
\]

Complete the statements of the following lemmas and theorems. Note you do not need to prove them just fill in the blanks with the appropriate relations defined previously.

(e) [2 points] **Type Soundness Theorem:**
If \( \) and \( \), then either

1. \( \)
2. \( \)

(f) [2 points] **Progress Lemma:**
If \( \), then either

1. \( \)
2. \( \)

(g) [2 points] **Preservation Lemma:**
If \( \) and \( \), then \( \)

(h) [2 points] **Canonical Forms Lemma:**
If \( \), then

1. \( \)
2. \( \)

(i) [2 points] **Substitution Lemma:**
If \( \) and \( \), then \( \)

**Problem 7** [10 points] The following questions refer to the files in lst. Submit modified version of all the files on-line. They require Java version 1.5 to compile. The directory contains a file lst/list.sml which is a reference implementation of a simple list library in SML. There are also four different partially implemented Java versions which attempt to implement the SML library in Java.

(a) [2 points] Complete the implementation of ListA.java

(b) [2 points] Complete the implementation of ListB.java
(c) [2 points] Complete the implementation of ListC.java

(d) [2 points] Complete the implementation of ListD.java

(e) [2 points] ListC.java and ListD.java represent internally how generics are implemented by the Java compiler. The ListD version of the functions map and length are more efficient because they avoid introducing extra coercions. Ignoring efficiency issues why might the ListA/ListC interface be preferred to the ListB/ListD interface.