**Programming Languages [Fall 2010]**

**Test III**

**NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructions:**

1) This test is 10 pages in length.

2) You have 2 hours to complete and turn in this test.

3) Short-answer questions include a guideline for how many sentences to write. Respond in complete English sentences.

4) This test is closed books, notes, papers, friends, neighbors, phones, etc.

5) Use the backs of pages in this test packet for scratch work. If you write more than a final answer in the area next to a question, circle your final answer.

6) Write and sign the following:

“I pledge my Honor that I have not cheated, and will not cheat, on this test.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_1. [4 points]

What is *higher-order abstract syntax*? [1 sentence]

2. [4 points]

Define *high-level programming language*. [1 sentence]

3. [7 points]

In class we implemented *map* in terms of *foldr*: map F L ≡ foldr (fn(x,y)=>F(x)::y) [] L Now implement *foldr* in terms of *map* (without side effects), or if it can’t be done, briefly explain why not.

4. [14 points]

a) Write a curried ML function *f* that takes an int list *L* and an int *i* and returns the *i*th element of *L*. If *L* has fewer than *i* elements, or if *i* is less than 1, then *f* returns 0. E.g., f [9,8,7] 2 returns 8, while f [9,8,7] 4 and f [9,8,7] 0 return 0.

b) Rewrite your function from Part (a) in diML-AU.

5. [7 points]

Consider the following ML code: val f = map (fn(x,y)=>print(y))

a) Rewrite this code so that it defines f in an equivalent way but does not violate ML’s value restriction.

b) What type does f have?

6. [20 points]

State and prove Preservation for λST. You may assume without proof all the standard lemmas. You only need to prove two cases:

(1) (2) e1 → e1’

(λ(x:τ1):τ2=e)(v) → [v/x]e e1 e2 → e1’ e2

7. [21 points]

Consider a language L with boolean and reference types:

τ ::= bool | τ ref

e ::= true | false | if e1 then e2 else e3 | ref(e) | l | !e | e1:=e2

Expressions in L have the expected semantics, as discussed in class.

a) Is L Turing complete? Provide some intuition to justify your answer.

b) Define static semantics for L.

L has the following syntax:

τ ::= bool | τ ref

e ::= true | false | if e1 then e2 else e3 | ref(e) | l | !e | e1:=e2

c) Define dynamic semantics for L *without* using evaluation contexts.

L has the following syntax:

τ ::= bool | τ ref

e ::= true | false | if e1 then e2 else e3 | ref(e) | l | !e | e1:=e2

d) Define dynamic semantics for L using evaluation contexts.

8. [14 points]

Recall the following syntax for expressions in “extended diML”:

e ::= true | false | if e1 then e2 else e3 | n | e1+e2 | e1<e2 | fun f(x:τ1):τ2=e | e1 e2 | x | ( ) |

(e1,e2) | let(x1,x2)=e1 in e2 end | (inl e):τ1+τ2 | (inr e):τ1+τ2 | case e of (inl x1=>e1|inr x2=>e2) |

roll(e) | unroll(e) | l | ref(e) | !e | e1:=e2 | (e1; e2; …; en) | while(e1){e2} | fail | try e1 with e2 |

return e | {e}

a) State the Canonical Forms Lemma for extended diML.

b) Define evaluation contexts for extended diML.

9. [9 points]

Prove the following conjecture about diML, or prove that the conjecture is false.

*For all* e, Γ, *and* τ1 *there exists at most one* τ2 *such that* Γ├(fun f(x:τ1):τ2=e):τ1->τ2.

(As always, assume variables are implicitly renamed to be unique.)

**[Undergraduates stop here. The following problem is for graduate students.]**

10. [11 points]

For each of the following diML-AU expressions: If the expression is well typed, what is its type? If the expression is ill typed, briefly explain why.

a) roll(0)

b) roll(roll(unroll(unroll(roll(unroll(roll(0)))))))

c) roll( (inl 0):τ )

where τ = int +

(rec t is ((rec t is (int + rec t’ is (t + (t’\*t)))) +

(t \* rec t’’ is (int + rec t is (t’’ + (t\*t’’))))))