# Programming Languages [Fall 2018]
# 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) This test is closed books, notes, laptops, phones, smartwatches, friends, neighbors, etc.

4) Use the backs of pages in this test packet for scratch work.

5) Write and sign the following: "I pledge my Honor that I have not cheated, and will not cheat, on this test."

______________________________________________________________________

______________________________________________________________________

Signed: ______________________________________________

1. [15 points]

a) Implement a list-reversing function ($\alpha$ list$\rightarrow\alpha$ list) in ML that runs in linear time, subject to the constraints of Assignment I (e.g., no library-function calls).

b) Implement a 1-line list-reversing function ($\alpha$ list$\rightarrow\alpha$ list) in ML, subject to the constraints of Assignment II (e.g., no recursion except through map/fold function calls).

c) In diML extended with binary product, binary sum, and iso-recursive types: Define types for rolled and unrolled int-lists and then implement an int-list-reversing function.

2. [70 points]

Consider a strongly typed programming language B having first-order abstract syntax:

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\quad\quad \tau ::= \text{bool} \mid \text{unit}$

a) Define higher-order abstract syntax for B.

b) Define static semantics for B. Well-typed while-loops have unit type in B.

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x=e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\tau ::= \text{bool} \mid \text{unit}$

c) Define CBV and CBN dynamic semantics for B, without using evaluation contexts.

d) Define full-β dynamic semantics for B, *with* evaluation contexts.

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x{=}e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\qquad \tau ::= \text{bool} \mid \text{unit}$

e) Show a B program P that evaluates differently in CBV and CBN evaluation strategies. Illustrate the difference by tracing P's evaluation in each strategy, underlining redexes.

f) On the following 1-3 pages, state all the lemmas, theorems, and corollary needed to prove type safety for B, and prove one *inductive* case of each. *If the proof does not use induction, then you need not prove any cases*. Your statements and proofs should assume that B has a CBV dynamic semantics defined without evaluation contexts. Also assume that the transitive rule for judgment form $e \rightarrow^* e'$ has premises $e \rightarrow^* e_1$ and $e_1 \rightarrow e'$.

Hint: You should show Weakening, Inversion, Canonical Forms, and Substitution lemmas, Progress and Preservation theorems, a Type-safety corollary, and 5 proof cases.

In addition, for every lemma, theorem, and corollary that you prove a case of, such that the lemma/theorem/corollary is stated in the form of *if A and B then C*, also disprove the converse of that lemma/theorem/corollary by defining a counterexample such that C and B are true, but A is not. For example, if you just proved a case of a lemma L stating that *for all e and e', if $e \rightarrow e'$ and $e{=}e'$ then $e'{=}e$*, you could disprove the converse of L with the counterexample $e{=}e'{=}()$, because then $e'{=}e$ and $e{=}e'$ are true, but $e \rightarrow e'$ is not.

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\qquad$ $\tau ::= \text{bool} \mid \text{unit}$

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\quad\quad\quad$ $\tau ::= \text{bool} \mid \text{unit}$

$e ::= \text{true} \mid \text{false} \mid x \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{while}(e_1)\{e_2\} \mid ()$ $\qquad \tau ::= \text{bool} \mid \text{unit}$

3. [15 points]

In diML, decomposition-for-types says that if $\Gamma \vdash E[e]:\tau$ then $\exists\tau'$:($\Gamma \vdash e:\tau'$ and $\Gamma \vdash E[\tau']: \tau$). Now consider a PL called diMLR, which is diML extended with return expressions, as defined in class. Prove the appropriate decomposition-for-types lemma for diMLR, but only show one case of the proof: E=return E'. Also define any rules cited in your proof.

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

4.  [7 points]
Define a rule, as we did in class, for typing a memory containing pointers and arrays. The judgment form is M:A.