<table>
<thead>
<tr>
<th>Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NetID:</td>
<td></td>
</tr>
</tbody>
</table>

- You have 2 hours to complete this exam.
- This is a closed-notes exam.
- Do not share anything with other students. Do not talk to other students. Do not look other students’ exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.
- Including this cover sheet and rules at the end, there are 9 pages to the exam. Once the exam begins, please verify that you have all 9 pages.
- Please write your name and NetID in the spaces above.
- Show your work. Partial credit will be given for incomplete answers.
- The pages at the end of the exam contain definitions and inference rules for various systems. You may detach these pages. If you do, please turn them in with the rest of your exam.
- If you finish with time remaining, check your work!
Problem 1. (0 points)
The typing rules for a simple expression language are given in Appendix A.

(a) (0 points) For each of the following typing judgments, indicate whether it is provable for all values of
Γ, no values of Γ, or some values of Γ.
1. Γ ⊢ 5 : Nat
2. Γ ⊢ x : Bool
3. Γ ⊢ ite tru 5 tru : Nat
4. Γ ⊢ ite (isz x) 3 5 : Nat

(b) (0 points) State progress and preservation theorems for this type system.

(c) (0 points) Suppose we added the following typing rule to the system:

\[
\frac{Γ ⊢ e : Nat \quad Γ ⊢ e_1 : Nat \quad Γ ⊢ e_2 : Nat}{Γ ⊢ \text{ite} \ e \ e_1 \ e_2 : Nat}
\]

Would this affect progress? Would this affect preservation? For each “yes” answer, give an example of
a term that might no longer have the affected property.
Problem 2. (0 points)
The small-step semantics for the simply-typed lambda calculus (STLC) are given in Appendix B.

(a) (0 points) Are there any stuck terms in the language? If so, give an example.

The function afi, defined as follows, computes a list of the free variables in a term:

\[
\text{Fixpoint } \text{afi } (e : \text{tm}) : \text{list string } := \\
\text{match } e \text{ with } \\
| x \Rightarrow [x] \\
| e_1 e_2 \Rightarrow \text{afi } e_1 ++ \text{afi } e_2 \\
| \lambda x : T. e \Rightarrow \text{remove all } x \text{ (afi } e) \\
| \text{tru } \Rightarrow [] \\
| \text{fls } \Rightarrow [] \\
| \text{test } e_1 e_2 e_3 \Rightarrow \text{afi } e_1 ++ \text{afi } e_2 ++ \text{afi } e_3 \\
\text{end.}
\]

where remove all x l removes all occurrences of x from the list l.

(b) (0 points) What is the output of afi ((\lambda x : \text{Nat}. x y) z)?

(c) (0 points) Write an inductive relation that computes the same value as afi. For full credit, your relation should not call the afi function.
Problem 3. (0 points)
The typing rule for application in STLC is:

$$\Gamma \vdash e_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash e_2 : T_1$$  
$$\Gamma \vdash e_1 \ e_2 : T_2$$

Suppose we wanted to add an operator $\&$ to the language such that $f \ & g$ returns a function that applies $f$ to its argument and then applies $g$ to the result, that is, $(f \ & g) \ x$ is equal to $g \ (f \ x)$. Write a typing rule for the $\&$ operator.
Problem 4. (0 points)
The semantics for a simple imperative language with memory are given in Appendix C.
(a) (0 points) Write the next step taken by the configuration:

\[ x ::= \ast y ;; \ast z ::= x / (x \mapsto 0, y \mapsto 10, z \mapsto 11), (10 \mapsto 3, 11 \mapsto 4) \]

(b) (0 points) What are the resulting state and memory after the above program has executed completely (that is, stepped until the command becomes SKIP)? How many steps does it take to get there?
Problem 5. (0 points)
Consider the following program:

T1 ::= *X;;
T2 ::= *Y;;
*X ::= T2;;
*Y ::= T1

The rules of separation logic for this language are given in Appendix D.
(a) (0 points) Write an informative pre- and postcondition for the program in separation logic.

(b) (0 points) Decorate the program with a proof of its pre- and postcondition.
A Type System for an Expression Language

Γ ⊢ e : T means that in the type context Γ, the expression e has type T.

Γ ⊢ x : T
Γ ⊢ n : Nat
Γ ⊢ tru : Bool
Γ ⊢ fals : Bool

Γ ⊢ e1 : Nat
Γ ⊢ e2 : Nat
Γ ⊢ pls e1 e2 : Nat
Γ ⊢ isz e : Bool
Γ ⊢ e : Nat
Γ ⊢ tru e : Bool
Γ ⊢ fals e : Bool
Γ ⊢ e1 : Nat
Γ ⊢ e2 : Nat
Γ ⊢ellite e e e1 e2 : T

B Small-Step Semantics of Simply-Typed Lambda Calculus

e → e' means that the λ-term e steps to the term e' in one step. Values are either abstractions λx : T. e or boolean values tru or fals. In the following rules, any term named v must be a value, while other terms may not be values.

(λx : T. e) v → [x := v]e
e1 e2 → e1' e2
v1 e2 → v1 e2'

test tru e1 e2 → e1
test fals e1 e2 → e2
e → e'

test e1 e2 → test e' e1 e2

C Small-Step Semantics of Imp with Memory

c / st, m → c' / st', m' means that the command c, when executed starting in the state st and memory m, takes a single step to the new command c' in the state st' and memory m'.

aeval st a = v

x := a / st, m → SKIP / st(x := v), m
c1 / st, m → c1' / st', m'
c1 ; c2 / st, m → c1' ; c2 / st', m'

beval st b = true

TEST b THEN c1 ELSE c2 FI / st, m → c1 / st, m

beval st b = false

TEST b THEN c1 ELSE c2 FI / st, m → c2 / st, m

WHILE b DO c END / st, m → TEST b THEN c ;; WHILE b DO c END ELSE SKIP FI / st, m

aeval st a = ℓ m(ℓ) = v

x := *a / st, m → SKIP / st(x := v), m

aeval st a1 = ℓ aeval st a2 = v

*a1 := a2 / st, m → SKIP / st, m(ℓ := v)

D Separation Logic

{P} c {Q} means that if P holds before executing c, then Q will hold after executing c.
\[
\frac{\{P\} \text{ SKIP } \{P\}}{
\{P\} \text{ SKIP } \{P\} \quad \{Q[x \mapsto a]\} \ x := a \ \{Q\} \quad \{P\} \quad \{Q\} \ c_1 \ \{Q\} \ c_2 \ \{R\} \quad \{P\} \ c_1 \ ; \ c_2 \ \{R\} \quad \{P\} \ c_1 \ ; \ c_2 \ \{R\} \\
\frac{P_1 \Rightarrow P_2 \quad \{P_2\} \ c \ \{Q_2\} \quad Q_2 \Rightarrow Q_1}{\{P\} \ c \ \{Q_1\}} \\
\frac{\{P \land (e = \text{true})\} \ c \ \{P\}}{\{P\} \ \text{WHILE } e \ \text{DO } c \ \text{END} \ \{P \land (e = \text{false})\}} \\
\frac{x \neq y}{\{y \mapsto v\} \ x := *y \ \{y \mapsto v \land x = v\}} \\
\frac{\{x \mapsto v\} \ *x := c \ \{x \mapsto c\}}{\{P \ c \ \{Q\}\} \ \text{TEST } b \ \text{THEN } c_1 \ \text{ELSE} \ c_2 \ \text{FI} \ \{Q\}} \\
\frac{x \neq y}{\{y \mapsto v\} \ x := *y \ \{y \mapsto v \land x = v\}} \\
\frac{\{P \ c \ \{Q\}\} \ \text{WHILE } e \ \text{DO } c \ \text{END} \ \{P \land (e = \text{false})\}}{\{P \ c \ \{Q\}\} \ \text{WHILE } e \ \text{DO } c \ \text{END} \ \{P \land (e = \text{false})\}} \\
\frac{x \neq y}{\{y \mapsto v\} \ x := *y \ \{y \mapsto v \land x = v\}} \\
\frac{\{P \ c \ \{Q\}\} \ \text{WHILE } e \ \text{DO } c \ \text{END} \ \{P \land (e = \text{false})\}}{\{P \ c \ \{Q\}\} \ \text{WHILE } e \ \text{DO } c \ \text{END} \ \{P \land (e = \text{false})\}} \\
\frac{x \neq y}{\{y \mapsto v\} \ x := *y \ \{y \mapsto v \land x = v\}}
\]
E  Scratch Space