CS 494 SF Spring 2019 Final Practice Questions

<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

   Solution: provable for all Γ

2. Γ ⊢ x : Bool

   Solution: provable for some Γ

3. Γ ⊢ ite tru 5 tru : Nat

   Solution: not provable for any Γ

4. Γ ⊢ ite (isz x) 3 5 : Nat

   Solution: provable for some Γ

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

Solution:
Progress: For all e and T, if Γ ⊢ e : T, then either e is a value or e can take a step.
Preservation: For all e, e', and T, if Γ ⊢ e : T and e steps to e', then Γ ⊢ e' : T.

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

\[ \Gamma \vdash e : \text{Nat} \quad \Gamma \vdash e_1 : \text{Nat} \quad \Gamma \vdash e_2 : \text{Nat} \]

\[ \Gamma \vdash \text{ite} \ e \ e_1 e_2 : \text{Nat} \]

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.

Solution:
Progress will be affected: ite 7 8 9 can now be proved to have type Nat, but is not a value and cannot step.
Preservation will not be affected.
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.

**Solution: tru tru**

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

```coq
Fixpoint afi (e : tm) : list string :=
match e with
| x ⇒ [x]
| e_1 e_2 ⇒ afi e_1 ++ afi e_2
| λx : T. e ⇒ remove_all x (afi e)
| tru ⇒ []
| fls ⇒ []
| test e_1 e_2 e_3 ⇒ afi e_1 ++ afi e_2 ++ afi e_3
end.
```

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

(b) (0 points) What is the output of afi ((λx : Nat. x y) z)?

**Solution:** [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.

**Solution:** One possible solution is:

```coq
Inductive afi_rel : tm → list string → Prop :=
| afi_var : forall x, afi_rel x [x]
| afi_app : forall e_1 e_2 l_1 l_2, afi_rel e_1 l_1 → afi_rel e_2 l_2 → afi_rel (e_1 e_2) (l_1 ++ l_2)
| afi_lam : forall x T e l, afi_rel e l → afi_rel (λx : T. e) remove_all x l
| afi_tru : afi_rel tru []
| afi_fl : afi_rel fls []
| afi_test : forall e_1 e_2 e_3 l_1 l_2 l_3, afi_rel e_1 l_1 → afi_rel e_2 l_2 → afi_rel e_3 l_3 → afi_rel (test e_1 e_2 e_3) (l_1 ++ l_2 ++ l_3).
```
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.

Solution:

\[ \Gamma \vdash f : T_1 \rightarrow T_2 \quad \Gamma \vdash g : T_2 \rightarrow T_3 \]
\[ \Gamma \vdash f \ & \ g : T_1 \rightarrow T_3 \]
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:

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

Solution: SKIP ; ; \ast z ::= x / (x \mapsto 3, 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?

Solution: (x \mapsto 3, y \mapsto 10, z \mapsto 11), (10 \mapsto 3, 11 \mapsto 3), in 3 steps
Problem 5. (0 points)
Consider the following program:

\[
\begin{align*}
T_1 &::= \ast X;; \\
T_2 &::= \ast Y;; \\
\ast X &::= T_2;; \\
\ast Y &::= T_1
\end{align*}
\]

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.

Solution:
Precondition: \(\{X \mapsto a \land Y \mapsto b\}\)
Postcondition: \(\{X \mapsto b \land Y \mapsto a\}\)

(b) (0 points) Decorate the program with a proof of its pre- and postcondition.

Solution:
\[
\begin{align*}
\{X \mapsto a \land Y \mapsto b\} \\
T_1 &::= \ast X;; \\
\{X \mapsto a \land Y \mapsto b \land T_1 = a\} \\
T_2 &::= \ast Y;; \\
\{X \mapsto a \land Y \mapsto b \land T_1 = a \land T_2 = b\} \\
\ast X &::= T_2;; \\
\{X \mapsto T_2 \land Y \mapsto b \land T_1 = a \land T_2 = b\} \\
\ast Y &::= T_1 \\
\{X \mapsto T_2 \land Y \mapsto T_1 \land T_1 = a \land T_2 = b\} \rightarrow \\
\{X \mapsto b \land Y \mapsto a\}
\end{align*}
\]
A Type System for an Expression Language

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

<table>
<thead>
<tr>
<th>Γ(x) = T</th>
<th>n is a number</th>
<th>Γ ⊢ tru : Bool</th>
<th>Γ ⊢ fls : Bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Γ ⊢ e₁ : Nat</td>
<td>Γ ⊢ e₁ : Nat</td>
<td>Γ ⊢ e₁ : T</td>
<td>Γ ⊢ e₁ : T</td>
</tr>
<tr>
<td>Γ ⊢ pls e₁ e₂ : Nat</td>
<td>Γ ⊢ e₂ : Nat</td>
<td>Γ ⊢ e₂ : T</td>
<td>Γ ⊢ e₂ : T</td>
</tr>
</tbody>
</table>

B Small-Step Semantics of Simply-Typed Lambda Calculus

→ 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 fls. 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 → e₁'
e₁ e₂ → e₁' e₂
e₂ → e₂'
v₁ e₂ → v₁ e₂'
test tru e₁ e₂ → e₁

test fls e₁ e₂ → e₂
test e₁ e₂ → test e₁ e₂ e₁ e₂
e → e'

eval st a = v
x ::= a / st, m → SKIP / st(x := v), m
c₁ / st, m → e₁' / st', m'
c₁ ; c₂ / st, m → e₁' ; 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

D Separation Logic

{P} c {Q} means that if P holds before executing c, then Q will hold after executing c.
\[
\begin{align*}
\begin{array}{c}
\{P\} \text{ SKIP } \{P\} \\
\{Q[x \mapsto a]\} \quad x ::= a \quad \{Q\} \\
\{P\} \quad c_1 \quad \{Q\} \quad \{Q\} \quad c_2 \quad \{R\}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
P_1 \Rightarrow P_2 \\
\{P_2\} \quad c \quad \{Q_2\} \\
\{Q_2\} \quad \Rightarrow \quad Q_1
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
P \Rightarrow b \quad \{P\} \quad \text{TEST } b \quad \text{THEN } c_1 \quad \text{ELSE } \quad c_2 \quad \text{FI } \quad \{Q\}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\{P \land (e = \text{true})\} \quad c \quad \{P\} \\
\{P \land (e = \text{false})\} \quad \text{WHILE } x \text{ DO } c \text{ END } \\
x \neq y
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\{y \mapsto v\} \quad x ::= e \quad \{y \mapsto v \land x = v\}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\{x \mapsto v\} \quad *x ::= e \quad \{x \mapsto v\}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\{x \mapsto v\} \quad *x ::= e \quad \{x \mapsto e\}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\{x \mapsto v\} \quad *x ::= e \quad \{x \mapsto e\}
\end{array}
\end{align*}
\]
E  Scratch Space