You have 50 minutes 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 8 pages to the exam. Once the exam begins, please verify that you have all 8 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 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!
<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (25 points)
Suppose we are defining a programming language with the following abstract syntax:

\[
\text{Inductive } \text{exp} := \\
\mid \text{Zero} \\
\mid \text{One} \\
\mid \text{Flip } (e : \text{exp}) \\
\mid \text{Guard } (e1 : \text{exp}) (e2 : \text{exp}).
\]

The intended behavior of the language is as follows:
- Zero evaluates to 0
- One evaluates to 1
- Flip \( e \) evaluates to 0 if the value of \( e \) is non-zero and 1 otherwise
- Guard \( e1 \) \( e2 \) computes the value of \( e2 \) if the value of \( e1 \) is non-zero, and evaluates to 0 otherwise

(a) (10 points) Write a function \( \text{eval} : \text{exp} \rightarrow \text{nat} \) that evaluates programs in this language.

**Solution:**
\[
\text{Fixpoint } \text{eval} \ e : \text{exp} : \text{nat} := \\
\begin{array}{l}
\text{match } e \text{ with} \\
\mid \text{Zero } \Rightarrow 0 \\
\mid \text{One } \Rightarrow 1 \\
\mid \text{Flip } e' \Rightarrow \text{match } e' \text{ with } 0 \Rightarrow 1 \mid _- \Rightarrow 0 \text{ end} \\
\mid \text{Guard } e1 e2 \Rightarrow \text{match } e1 \text{ with } 0 \Rightarrow 0 \mid _- \Rightarrow \text{eval } e2 \text{ end}
\end{array}
\]

(b) (15 points) Write an inductive relation \( \text{evalR} : \text{exp} \rightarrow \text{nat} \rightarrow \text{Prop} \) that expresses the same behavior as a relation, in the big-step style. Do **not** use your \( \text{eval} \) function from the previous part.

**Solution:**
\[
\text{Inductive } \text{evalR} : \text{exp} \rightarrow \text{nat} \rightarrow \text{Prop} := \\
\mid \text{eval_Zero} : \text{evalR } \text{Zero} 0 \\
\mid \text{eval_One} : \text{evalR } \text{One} 1 \\
\mid \text{eval_Flip0} e (\text{He} : \text{evalR } e \ 0) : \text{evalR } (\text{Flip } e) 1 \\
\mid \text{eval_Flip} e n (\text{He} : \text{evalR } e \ n) (\text{Hn} : n \ll 0) : \text{evalR } (\text{Flip } e) 0 \\
\mid \text{eval_Guard0} e1 e2 (\text{He} : \text{evalR } e1 \ 0) : \text{evalR } (\text{Guard } e1 e2) 0 \\
\mid \text{eval_Guard} e1 e2 n1 n2 (\text{He1} : \text{evalR } e1 \ n1) (\text{Hn1} : n1 \ll 0) (\text{He2} : \text{evalR } e2 \ n2) : \\
\quad \text{evalR } (\text{Guard } e1 e2) n2
\]

Page 3
Problem 2. (25 points)
The big-step semantics of Imp are given in Appendix 2. All values $v$ are natural numbers. Which of the following statements are provable? You will receive partial credit for wrong answers if you explain your reasoning.

(a) (3 points) $X ::= 3 ;; X ::= 4 \not\Downarrow_{st} (X \mapsto 3)$

Solution: Not provable.

(b) (3 points) $X ::= 3 ;; X ::= 4 \not\Downarrow_{st} (X \mapsto 4)$

Solution: Provable.

(c) (4 points) \(\text{TEST } X < Y \text{ THEN } X ::= X - Y \text{ ELSE } X ::= X - X \FI / \not\Downarrow_{st} (X \mapsto 0)\)

Solution: Provable: in either case, $X$ will be set to 0.

(d) (5 points) For every program $c$ and starting state $st$, there is a state $st'$ such that $c \not\Downarrow_{st} st'$.

Solution: Not provable: infinite WHILE loops will not evaluate to any state.

For the next two questions, recall that two programs are equivalent if, for all starting states, they both evaluate to the same result state.

(e) (5 points) For any boolean expression $b$ and commands $c_1$ and $c_2$, the following two programs are equivalent:

\[
\text{TEST } b \text{ THEN } c_1 \text{ ELSE SKIP } \FI ;; \quad \text{TEST } b \text{ THEN SKIP ELSE } c_2 \FI
\]

and

\[
\text{TEST } b \text{ THEN } c_1 \text{ ELSE } c_2 \FI
\]

Solution: Not provable: $c_1$ may change the value of variables used in $b$, making it possible for both $c_1$ and $c_2$ to be executed in the first program.

(f) (5 points) For any $c$, the following two programs are equivalent:

\[
\text{WHILE } \sim (X = 1) \DO
\quad c ;;
\quad X ::= 1
\END
\]

and

\[
\text{TEST } X = 1 \text{ THEN } \not\Downarrow_{st} \FI ;;
\quad X ::= 1
\]

Solution: Provable. If $X$ is 1 in the initial state, both programs do not execute $c$ and end in a state where $X$ is 1. If $X$ is not 1 in the initial state, both programs execute $c$ once and then set $X$ to 1.
Problem 3. (25 points)

The small-step semantics of Imp are given in Appendix 3.

(a) (10 points) Which of the following statements are provable?

1. $X ::= 3 ; ; Y ::= 4 / st \rightarrow \text{SKIP} ; ; Y ::= 4 / st(X \rightarrow 3)$

2. $X ::= 3 ; ; Y ::= 4 / st \rightarrow X ::= 3 ; ; \text{SKIP} / st(Y \rightarrow 4)$

3. $X ::= 3 ; ; Y ::= 4 / st \rightarrow Y ::= 4 / st(X \rightarrow 3)$

4. $X ::= 3 ; ; Y ::= 4 / st \rightarrow^* \text{SKIP} ; ; Y ::= 4 / st(X \rightarrow 3)$

5. $X ::= 3 ; ; Y ::= 4 / st \rightarrow^* X ::= 3 ; ; \text{SKIP} / st(Y \rightarrow 4)$

6. $X ::= 3 ; ; Y ::= 4 / st \rightarrow^* Y ::= 4 / st(X \rightarrow 3)$

Solution: 1, 4, 6

(b) (15 points) Suppose we wanted to add a DO-WHILE loop to Imp, such that the command $\text{DO } c \text{ WHILE } b \text{ END}$ executes $c$ first, then repeats if $b$ is true. Write one or more small-step rules giving the semantics of $\text{DO } c \text{ WHILE } b \text{ END}$. For full credit, do not use the ordinary WHILE loop in your answer.

Solution:

$\text{DO } c \text{ WHILE } b \text{ END} / st \rightarrow c ; ; \text{TEST } b \text{ THEN DO } c \text{ WHILE } b \text{ END ELSE SKIP FI} / st$
Problem 4. (25 points)
Consider the following program:

\[
\begin{align*}
R & := 1 ; ; \\
\text{WHILE } Y > 0 \text{ DO} \\
R & := R * X ; ; \\
Y & := Y - 1 \\
\text{END ; ;} \\
R & := R + Z
\end{align*}
\]

(a) (5 points) What does this program compute? Write an informative pre- and postcondition for the program.

Solution:

Precondition: \(X = a \land Y = b \land Z = c\)
Postcondition: \(R = a^b + c\)

(b) (20 points) The rules of Hoare logic are given in Appendix 4. Starting with your pre- and postcondition from the previous part, decorate the program by writing an assertion between each pair of lines to form a valid sequence of Hoare triples. You will need to come up with an invariant for the loop. Make sure to write all necessary implications and check that they hold.

Solution:

\[
\begin{align*}
\{X = a \land Y = b \land Z = c\} & \implies \{1 = a^{(b - 1)} \land Z = c\} \\
R & := 1 ; ; \\
\{R = a^{(b - 1)} \land Z = c\} & \implies \{R * X = a^{(b - 1)} \land Z = c\} \\
\text{WHILE } Y > 0 \text{ DO} \\
\{R = a^{(b - 1)} \land Z = c\} & \implies \{R * X = a^{(b - 1)} \land Z = c\} \\
R & := R * X \\
\{R = a^{(b - 1)} \land Z = c\} & \implies \{R = a^{(b - 1)} \land Z = c\} \\
Y & := Y - 1 \\
\{R = a^{(b - 1)} \land Z = c\} & \implies \{R * X = a^{(b - 1)} \land Z = c\} \\
\text{END ; ;} \\
\{R + Z = a^b + c\} & \implies \\
R & := R + Z \\
\{R = a^b + c\}
\end{align*}
\]
1 Syntax of Imp

Arithmetic expressions: 
\[ a := n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \]

Boolean expressions: 
\[ b := \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \sim b \mid b \& \& b \]

Commands: 
\[ c := \text{SKIP} \mid x := a \mid c_1 ;; c_2 \mid \text{TEST} \ b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} \mid \text{WHILE } b \text{ DO } c \text{ END} \]

where \( n \) is a natural number, \( x \) is a variable name.

2 Big-Step Semantics of Imp

\[ c / st \Downarrow st' \] means that the command \( c \), when executed starting in the state \( st \), results in the state \( st' \).

\[
\begin{align*}
\text{aeval} \ st \ a &= v \\
\text{x ::= a} / st \Downarrow st(x \mapsto v)
\end{align*}
\]

\[
\begin{align*}
\text{beval} \ st \ b &= \text{true} \\
\text{TEST} \ b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} / st \Downarrow st'
\end{align*}
\]

\[
\begin{align*}
\text{beval} \ st \ b &= \text{false} \\
\text{WHILE} \ b \text{ DO } c \text{ END} / st \Downarrow st'
\end{align*}
\]

3 Small-Step Semantics of Imp

\[ c / st \rightarrow c' / st' \] means that the command \( c \), when executed starting in the state \( st \), takes a single step to the new command \( c' \) in the state \( st' \).

\[
\begin{align*}
\text{aeval} \ st \ a &= v \\
\text{x ::= a} / st \rightarrow \text{SKIP} / st(x \mapsto v)
\end{align*}
\]

\[
\begin{align*}
\text{beval} \ st \ b &= \text{true} \\
\text{TEST} \ b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} / st \rightarrow c_1 / st
\end{align*}
\]

\[
\begin{align*}
\text{beval} \ st \ b &= \text{false} \\
\text{WHILE} \ b \text{ DO } c \text{ END} / st \rightarrow \text{TEST} \ b \text{ THEN } c \text{ ELSE } \text{SKIP} / st
\end{align*}
\]
4 Floyd-Hoare Logic for Imp

\(\{P\} c \{Q\}\) means that if \(P\) holds before executing \(c\), then \(Q\) will hold after executing \(c\).

\[
\begin{array}{c}
\{P\} \text{SKIP} \{P\} \\
\{Q[x \mapsto a]\} x ::= a \{Q\} \\
\{P\} \text{c}_1 \{Q\} \quad \{Q\} \text{c}_2 \{R\} \\
P_1 \Rightarrow P_2 \quad \{P_2\} \text{c} \{Q_2\} \quad Q_2 \Rightarrow Q_1 \\
\{P_1\} \text{c} \{Q_1\} \\
\{P\} \text{TEST} b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI } \{Q\} \\
\{P\} \text{WHILE } e \text{ DO } c \text{ END } \{P \land (e = \text{false})\}
\end{array}
\]

5 Scratch Space