Suggested Solutions to Midterm Problems

1. Under what condition are $A[w/x][z/y]$ and $A[z/y][w/x]$ equal? Give an inductive proof of the equality (under that condition). (10%)

   Solution. A trivial condition is when x and y are the same variable (denoted by $x = y$) and w and z are the same ($w = z$); doing the same substitution twice consecutively is the same as doing it once. A less trivial condition is when $x \neq y$, $w \neq y$, and $z \neq x$. We shall give an inductive proof for this case. (It is possible to generalize the result for $A[t/x][u/y]$ and $A[u/y][t/x]$ where t and u are terms; the two formulae are equal if $x \neq y$, y is not free in t, and x is not free in u.)

   We first show that, for any term t, $t[w/x][z/y]$ and $t[z/y][w/x]$ are equal when $x \neq y$, $w \neq y$, and $z \neq x$:

   Base case: (1) constants: $c[w/x][z/y] = c = c[z/y][w/x]$. (2) variables: $x[w/x][z/y] = w[z/y] = w$ (as $w \neq y$) $= x[w/x] = (x[z/y])[w/x]$ (as $x \neq y$) $= x[z/y][w/x]$; analogously for y. For $v \neq x$ and $v \neq y$, $v[w/x][z/y] = v = v[z/y][w/x]$.

   Induction step: For every k-ary ($k > 0$) function f:

$$\begin{array}{cll} & f(t_1, t_2, \cdots, t_k)[w/x][z/y] & \\ = & f(t_1[w/x], t_2[w/x], \cdots, t_k[w/x])[z/y] & \text{, definition of substitution} \\ = & f(t_1[w/x][z/y], t_2[w/x][z/y], \cdots, t_k[w/x][z/y]) & \text{, definition of substitution} \\ = & f(t_1[z/y][w/x], t_2[z/y][w/x], \cdots, t_k[z/y][w/x]) & \text{, induction hypothesis} \\ = & f(t_1[z/y], t_2[z/y], \cdots, t_k[z/y])[w/x] & \text{, definition of substitution} \\ = & f(t_1, t_2, \cdots, t_k)[z/y][w/x] & \text{, definition of substitution} \end{array}$$

   We now prove that, for any formula A, $A[w/x][z/y]$ and $A[z/y][w/x]$ are equal when $x \neq y$, $w \neq y$, and $z \neq x$:

   Base case (atomic formulae): (1) $\bot[w/x][z/y] = \bot = \bot[z/y][w/x]$. (2) For every 0-ary predicate p, $p[w/x][z/y] = p = p[z/y][w/x]$. (3) For k-ary ($k > 0$) predicates, the proof is analogous to that for functions.

   Induction step (formulae): (1) Boolean combinations: we prove the case of $A \wedge B$; other cases are similar.

$$\begin{array}{cll} & (A \wedge B)[w/x][z/y] & \\ = & (A[w/x] \wedge B[w/x])[z/y] & \text{, definition of substitution} \\ = & A[w/x][z/y] \wedge B[w/x][z/y] & \text{, definition of substitution} \\ = & A[z/y][w/x] \wedge B[z/y][w/x] & \text{, induction hypothesis} \\ = & (A[z/y] \wedge B[z/y])[w/x] & \text{, definition of substitution} \\ = & (A \wedge B)[z/y][w/x] & \text{, definition of substitution} \end{array}$$

(2) Quantified formulae: we prove the two cases of $(\forall xA)$ and $(\forall vA)$ where $v \neq x$ and $v \neq y$; other cases are similar.

$$\begin{array}{lll}
 & (\forall xA)[w/x][z/y] & \\
= & (\forall xA)[z/y] & , x \text{ does not occur free in } (\forall xA) \\
= & \forall x(A[z/y]) & , x \neq y \\
= & (\forall x(A[z/y]))[w/x] & , x \text{ does not occur free in } (\forall x(A[z/y])) \\
= & (\forall xA)[z/y][w/x] & , \text{definition of substitution}
\end{array}$$

$$\begin{array}{lll}
 & (\forall vA)[w/x][z/y] & \\
= & (\forall v(A[w/x]))[z/y] & , v \neq x \\
= & \forall v(A[w/x][z/y]) & , v \neq y \\
= & \forall v(A[z/y][w/x]) & , \text{induction hypothesis} \\
= & (\forall v(A[z/y]))[w/x] & , v \neq x \\
= & (\forall vA)[z/y][w/x] & , v \neq y
\end{array}$$

□

2. Prove the following sequents using Gentzen's System LK. You may treat Γ and Δ in a sequent $\Gamma \vdash \Delta$ as *multisets* of formulas (to shorten the proof, as the Exchange rules will no longer be needed).

   (a) $\vdash (A \to B) \to ((A \to \neg B) \to \neg A)$ (10%)

   Solution. Left as an exercise (since most of you were able to find a proof). □

   (b) $\vdash \forall x(P(x) \land Q(x)) \to \exists yP(y) \land \forall zQ(z)$ (10%)

   Solution.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(w) \vdash P(w)}\,(\mathit{axiom})}{P(w) \land Q(w) \vdash P(w)\ \{w \text{ as } t \text{ in } P(y)[t/y]\}}\,(\land L)}{P(w) \land Q(w)\ \{= (P(x) \land Q(x))[w/x]\} \vdash \exists yP(y)}\,(\exists R)}{\forall x(P(x) \land Q(x)) \vdash \exists yP(y)}\,(\forall L) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Q(w) \vdash Q(w)}\,(\mathit{axiom})}{P(w) \land Q(w) \vdash Q(w)}\,(\land L)}{\forall x(P(x) \land Q(x)) \vdash Q(w)}\,(\forall L)}{\forall x(P(x) \land Q(x)) \vdash \forall zQ(z)}\,(\forall R)}{\dfrac{\exists x(P(x) \land Q(x)) \vdash \exists yP(y) \land \forall zQ(z)}{\vdash \forall x(P(x) \land Q(x)) \to \exists yP(y) \land \forall zQ(z)}\,(\to R)}\,(\land R)$$

□

3. Recall that a set Γ of propositions is *inconsistent* if, for any proposition B, the sequent $\Gamma \vdash B$ is provable (using the propositional part of System LK).

   Prove that, if there is some proposition A such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then Γ is inconsistent. (Hint: think of the Cut rule and the sequents "$\Gamma \vdash \neg\neg A$" and "$\Gamma, \neg\neg A \vdash B$", for an arbitrary proposition B) (10%)

   Solution. We give a proof of $\Gamma \vdash B$ for an arbitrary B under the assumption that there is some proposition A such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable. To shorten the proof, we shall take the multiset view of sequents.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{assumed proof}}{\Gamma \vdash A}}{\Gamma, \neg A \vdash}(\neg L)}{\Gamma \vdash \neg\neg A}(\neg R) \qquad \dfrac{\dfrac{\dfrac{\text{assumed proof}}{\Gamma \vdash \neg A}}{\Gamma, \neg\neg A \vdash}(\neg L)}{\Gamma, \neg\neg A \vdash B}(WR)}{\Gamma, \Gamma \vdash B}(\textit{Cut}\ (\neg\neg A))}{\Gamma \vdash B}(\textit{Repeated CL})$$

□

4. Give a pair of pre-condition and post-condition that specifies the correctness requirement of the following code segment for finding the smallest element of an array of numbers. Prove that the code indeed satisfies the requirement.

```
$min, i := A[1], 1;$
while $i < n$ do begin
    $i := i + 1;$
    if $A[i] < min$ then $min := A[i];$
end
```

To handle multiple assignments of the form "$x, y := E, F$", the following axiom may be added to Hoare Logic:

$$\overline{\{P[E, F/x, y]\}\ x, y := E, F\ \{P\}}$$

Note: $A[t, u/x, y]$ ($T[t, u/x, y]$) denotes the formula (term) obtained from *simultaneously* substituting term t for free occurrences of x and term u for y in formula A (term T). For example, $(x + 1 > y)[y, x/x, y] = y + 1 > x$. (20%)

Solution. The program finds (and assigns to min) the smallest element of array A if array A has at least one element ($n \geq 1$). This correctness requirement may be specified by the following pair of pre and post-conditions.

Pre-condition: $n \geq 1$.

Post-condition: $(\exists j : 1 \leq j \leq n : min = A[j]) \wedge (\forall k : 1 \leq k \leq n : min \leq A[k])$ (or $\exists j(1 \leq j \leq n \wedge min = A[j]) \wedge \forall k(1 \leq k \leq n \rightarrow min \leq A[k])$, using the syntax defined in class).

We annotate the program as follows to show the main steps of the correctness proof.

$\{n \geq 1\}$

```
$S_1$: $min, i := A[1], 1$;
$\{1 \leq i \leq n \wedge (\exists j : 1 \leq j \leq i : min = A[j]) \wedge (\forall k : 1 \leq k \leq i : min \leq A[k])\}$
$S_2$: while $i < n$ do begin
            $i := i + 1$;
            if $A[i] < min$ then $min := A[i]$;
      end
$\{i \geq n \wedge 1 \leq i \leq n \wedge (\exists j : 1 \leq j \leq i : min = A[j]) \wedge (\forall k : 1 \leq k \leq i : min \leq A[k])\}$
```

The last assertion in the annotation implies the post-condition of the correctness requirement. By the rule of (sequential) composition, the rule of consequence, and the preceding implication, correctness of the program follows from correctness of the above annotation. Let $P(i)$ denote $\exists j : 1 \leq j \leq i : min = A[j]$ and $Q(i)$ denote $\forall k : 1 \leq k \leq i : min \leq A[k]$. We are now left with two proof obligations (proving the correctness of two Hoare triples):

$\{n \geq 1\}\ S_1\ \{1 \leq i \leq n \wedge P(i) \wedge Q(i)\}$

and

$\{1 \leq i \leq n \wedge P(i) \wedge Q(i)\}\ S_2\ \{i \geq n \wedge 1 \leq i \leq n \wedge P(i) \wedge Q(i)\}$.

The rest should be just a simple exercise of the Hoare logic. □

5. In a virtual department store, customers purchase products from various stores. The department also allows customers to make gift packages (say, Alice's gift from Bob) so that other customers may choose them.
   Consider the following classes:

```
class Shoe extends Product {
  public Shoe (int p, String v);
  public int price ();
  public String vendor ();
  // ...
};

class Clothing extends Product {
  public Clothing (int p, String v);
  public int price ();
  public String vendor ();
  // ...
}

Shoe nike (1000, "Nike");
Clothing levis (800, "Levi's");
```

(a) Use the Composite pattern to design two classes: `Product` as component and `Gift` as composite. Write down their class definitions. (5%)

Solution.

```
class Product {
  public int price ();
  public String vendor ();
  public void add (Product);
  public void remove (Product);
};

class Gift extends Product {
  public add (Product) { /* ... */ }
  public void remove (Product) { /* ... */ }

  private Product products[];
};
```

□

(b) Suppose Alice's gift contains `nike` and `levis`. Write down the class definition of `Alicesgift`. (3%)

Solution.

```
class Alicesgift extends Gift {
  public Alicegift () {
    add (nike);
    add (levis);
  }
};
```

□

(c) Use the Prototype pattern to design the class `Catalog` that contains methods `MakeNike`, `MakeLevis`, `MakeAlicesGift`. Redefine classes `Product`, `Shoe`, `Clothing` if necessary. (5%)

Solution.

```
class Shoe extends Product {
  // ...
  public Shoe clone ();
}

class Clothing extends Product {
  // ...
  public Clothing clone ();
}

class Gift extends Product {
  // ...
  public Gift clone ();
}

class Catalog {
  public Catalog (Shoe s, Clothing c, Gift g) {
    _shoe = s; _clothing = c; _gift = g;
  }
  public Shoe MakeNike () {
    _shoe.clone ();
  }
  public Clothing MakeLevis () {
    _clothing.clone ();
  }
  public Gift MakeGift () {
    _gift.clone ();
  }

  private Shoe _shoe;
  private Clothing _clothing;
  private Gift _gift;
}
```

□

6. Consider the following definitions:

```
class A {
  public void foo () {};
  public void bar () {};
};

class B : A {
  public void foo () ;
  public void bar ();
}

class C {
  public void foo () { a.foo (); };
  public void bar () { a.bar (); };
  private A a;
}
```

(a) Is class `B` a subclass of class `A`? (2%)

Answer: Yes.

(b) Is type `B` a subtype of type `A`? (2%)

Answer: Yes.

(c) Is class `C` a subclass of class `A`? (2%)

Answer: No.

(d) Is type `C` a subtype of type `A`? (2%)

Answer: Yes.

(e) If a pattern is defined by using class inheritance (like `B`), we call it

[✓ class pattern] ___object pattern (2%)

(f) If a pattern is defined by using forwarding (like `C`), we call it

___class pattern [✓ object pattern] (2%)

7. Suppose we would like to implement class `Collection`:

```
class Collection {
  public void add (Node n);
  public void remove ();
  public Node peek ();
  // ...
};
```

But there are several randomly access data structures that may be used in the implementation. For instance, we may have linked lists:

```
class LinkedList {
  public void insert (int idx, Node n);
  public void delete (int idx);
  public Node first ();
  public Node last ();
};
```

Or arrays:

```
class Array {
  public void insert (int idx, Node n);
  public void delete (int idx);
  public Node first ();
  public Node last ();
};
```

(a) Apply the Bridge pattern to define the class `RADS` that allows `Collection` to change the underlying randomly access data structure implementations. Redefine the class `Collection` if necessary. (5%)

Solution.

```
class RADS {
  public void insert (int idx, Node n);
  public void delete (int idx);
  public Node first ();
  public Node last ();
};

class Collection {
  public Collection (RADS r)
    { _rads = r; }
  // ...
  private _rads;
};
```

□

(b) Apply the Factory method pattern to define and implement the `Collection` subclasses `Stack` and `Queue` by the classes `Array` and `LinkedList` respectively. (4%)

Solution.

```
class Stack extends Collection {
  public Stack ()
    { super (new Array); top = 0; }
  public void add (Node n)
    { insert (top++, n); }
  public void remove ()
    { delete (--top); }
  public Node peek ()
    { return last (); }

  private int top;
};

class Queue extends Collection {
  public Stack ()
    { super (new LinkedList); head = tail = 0; }
  public void add (Node n)
    { insert (tail++, n); }
  public void remove ()
    { delete (head++); }
  public Node peek ()
    { return first (); }

  private int head, tail;
};
```

□

(c) Apply the Abstract Factory pattern to define `Stack` and `Queue` so that the client can choose randomly access data structures while creating `Stack` or `Queue` objects. (6%)

Solution.

```
class Stack extends Collection {
  public Stack (RADS factory)
    { super (factory); top = 0; }
  // ...
};

class Queue extends Collection {
  public Queue (RADS factory)
    { super (factory); head = tail = 0; }
  // ...
};

Stack arrayStack (new Array);
Stack linkedListStack (new LinkedList);
Stack arrayQueue (new Array);
Stack linkedListQueue (new LinkedList);
```

□