CIS 500 — Software Foundations

Midterm I

(Standard and advanced versions together)

March 27, 2013

Answer key

1. (8 points) Indicate whether or not each of the following Hoare triples is valid by writing either "valid" or "invalid." Also, for those that are invalid, give a counter-example. The definition of Hoare triples is given on page 13, for reference.

(a)

```
{{ X=n /\ Y=m /\ Z=o }}
X ::= Y;
Y ::= Z;
Z ::= X
{{ X=n /\ Y=o /\ Z=m }}
```

Answer: Invalid: the state described in the precondition is a counterexample. On this state the output state should be such that `X=m`.

(b)

```
{{ X=1 \/ Z=0 }}
IFB  Z=0  THEN
     X ::= 0
THEN
     X ::= 1 - X
FI;
Z ::= 1
{{ X=0 /\ Z=1 }}
```

Answer: Valid.

(c)

```
{{ True }}
WHILE X > 0 DO
   X ::= X - 1;
   Y ::= X
END
{{ X=0 /\ Y=0 }}
```

Answer: Invalid, a counterexample is the state where `X=0` and `Y=1`.

(d)

```
{{ X>0 }}
WHILE X > 0 DO
   X ::= X + 1;
   Z ::= X - 1
END
{{ Z=X }}
```

Answer: Valid because the loop never terminates.

Grading scheme: 2 points each. 1 point for the counterexamples.

2. (12 points) Given the following programs, group together those that are equivalent in Imp by drawing boxes around their names. For example, if you think programs a through h are all equivalent to each other, but not to i, your answer should look like this: $\boxed{a, b, c, d, e, f, g, h}$ $\boxed{i}$.

The definition of program equivalence is repeated on page 13, for reference.

(a)
```
X ::= Y;
Y ::= Z;
X ::= 0
```

(b)
```
IFB Y > 3 THEN
  X ::= 2 * Y
ELSE
  X ::= 2 * Y
FI;
Y ::= X
```

(c)
```
WHILE X > 0 DO
   X ::= 0;
   SKIP
END
```

(d)
```
WHILE X > 0 DO
  X ::= X * Y + 1
END
```

(e)
```
X ::= 0;
Y ::= Z
```

(f)
```
X ::= Y;
WHILE X > 0 DO
  Y ::= X + 1;
  X ::= X - 1
END;
X ::= Y
```

(g)
```
Y ::= Z;
WHILE X > 0 DO
  X ::= X - 1;
  Y ::= Z
END
```

(h)
```
WHILE X <> X DO
  X ::= X + 1
END;
X::=0
```

(i)
```
X ::= 2 * Y;
Y ::= 2 * Y
```

Answer:
$\boxed{a, e, g}$ $\boxed{b, i}$ $\boxed{c, h}$ $\boxed{d}$ $\boxed{f}$

Grading scheme: 1 point off for each missing pair; 1 point off for each extraneous pair.

(There was actually a typo in part f: we meant to write "Y ::= Y - 1"; what we did write does not compute the same thing as (b) and (i), as we had intended. Many people missed this. We marked this as -1 point, and in general followed this rubric: if you had [a,e] and [g], that was -1; if you had [a], [e], and [g], that was -3 (three different pairs missed))

3. [**Standard**] (8 points) In the `Imp` chapter, we defined the evaluation semantics for Imp commands using Coq's `Inductive` facility for defining relations (in `Prop`)...

```
Inductive ceval : com -> state -> state -> Prop := ...
```

...instead of as a recursive `Fixpoint` (in `Set`):

```
Fixpoint ceval (st : state) (c : com) : state := ...
```

Briefly (1-3 sentences) explain why. *Answer:* The evaluation relation is not a total function – it's partial, because `WHILE` loops may diverge. But `Fixpoint` can only be used (at least, without awkward workarounds) to define total functions.

Grading scheme: full marks for any answer explaining the impossibility to embed the evaluation of the Turing Complete Imp language into a terminating Coq's function.

On non-correct answers, we took away a variable number of points depending on the severity of the misstatement and other arguments for the inductive definition given.

4. [**Standard**] (7 points) Each of the following inference rules expresses a claim about Hoare triples. If the claim is true for every choice of `c` and `P`, write "valid." Otherwise write "invalid" and give an example of a `c` and `P` for which it fails.

(a)

$$\frac{\{\!\{\, P \,\}\!\}\ c\ \{\!\{\, P \,\}\!\}}{\{\!\{\, P \,\}\!\}\ c;c\ \{\!\{\, P \,\}\!\}}$$

Answer: valid

(b)

$$\frac{\{\!\{\, P \,\}\!\}\ c;c\ \{\!\{\, P \,\}\!\}}{\{\!\{\, P \,\}\!\}\ c\ \{\!\{\, P \,\}\!\}}$$

Answer: invalid; for example, let `P` *be* $X = 0 \wedge Y = 1$ *and let* `c` *be* `Z ::= X; X ::= Y; Y ::= Z`.

Grading scheme: (a) 3 pt (-1pt for a wrong explanation of the "valid" answer). (b) 4 pt (2 for the invalid answer, 2 for the counter-example (1pt malus if this is not a Imp program))

5. [**Standard**] (12 points) In this exercise we consider extending Imp with "one-sided conditionals" of the form

```
IF1 b THEN c FI
```

where `b` is a boolean expression, and `c` is a command. If `b` evaluates to `false`, then `IF1 b THEN c FI` does nothing.

To formalize the extended language, we first add a clause to the definition of commands:

```
Inductive com : Type :=
  ...
  | CIf1 : bexp -> com -> com.

Notation "'IF1' b 'THEN' c 'FI'" := (CIf1 b c).
```

(a) Refer to the definition of `ceval` (page 13) for the evaluation relation of Imp. What rule(s) must be added to this definition to formalize the behavior of `IF1`? Write out the additional rule(s) in formal Coq notation.

Answer:

```
| E_If1True : forall b c st st',
    beval st b = true ->
    c / st || st' ->
    IF1 b THEN c FI / st || st'
| E_If1False : forall b c st,
    beval st b = false ->
    IF1 b THEN c FI / st || st
```

(b) Write a Hoare proof rule for one-sided conditionals. (For reference, the standard Hoare rules for Imp are provided on page 14.)

Try to come up with a rule that is both sound and as precise as possible. For full credit, make sure your rule can be used to prove the following valid Hoare triple:

```
{{ X + Y = Z }}
IF1 Y <> 0 THEN
  X ::= X + Y
FI
{{ X = Z }}
```

Answer:

$$\frac{\{\!\{ P \wedge b \}\!\}\ \texttt{c}\ \{\!\{ Q \}\!\} \qquad (P \wedge \sim b) \twoheadrightarrow Q}{\{\!\{ P \}\!\}\ \texttt{IF1 b THEN c FI}\ \{\!\{ Q \}\!\}} \quad (\texttt{hoare\_if1})$$

The following is also a sound rule for `IF1`, but it is not precise enough to handle the litmus test.

$$\frac{\{\!\{ P \wedge b \}\!\}\ \texttt{c}\ \{\!\{ P \}\!\}}{\{\!\{ P \}\!\}\ \texttt{IF1 b THEN c FI}\ \{\!\{ P \}\!\}} \quad (\texttt{hoare\_if1\_weak})$$

Grading scheme: For part (a), 6 points.

For part b, we accepted with full marks both $(P\wedge \sim b) \twoheadrightarrow Q$ *and* $\{\{P\wedge \sim b\}\}\texttt{SKIP}\{\{Q\}\}$

We also accepted with full marks the syntactically incorrect expression $\{\{P\wedge \sim b\}\} \twoheadrightarrow \{\{Q\}\}$.

For a non-valid rule we took from 4 to 6 points away depending on the gravity of the error.

6. [**Advanced**] (12 points) In this question we consider extending Imp with `REPEAT` statements of the form

```
REPEAT c UNTIL b END
```

where `b` is a boolean expression, and `c` is a command. `REPEAT` behaves like `WHILE` except that the loop guard is checked *after* each execution of the body, with the loop repeating as long as the guard stays *false*. Because of this, the body will always execute at least once.

To formalize the extended language, we first add a clause to the definition of commands:

```
Inductive com : Type :=
  ...
  | CRepeat : com -> bexp -> com.

Notation "'REPEAT' e1 'UNTIL' b2 'END'" :=  (CRepeat e1 b2).
```

(a) Refer to the definition of `ceval` (page 13) for the evaluation relation of Imp. What rule(s) must be added to this definition to formalize the behavior of `REPEAT`? Write out the additional rule(s) in formal Coq notation.

Answer:

```
| E_RepeatEnd : forall st st' b1 c1,
    ceval st c1 st' ->
    beval st' b1 = true ->
    ceval st (CRepeat c1 b1) st'
| E_RepeatLoop : forall st st' st'' b1 c1,
    ceval st c1 st' ->
    beval st' b1 = false ->
    ceval st' (CRepeat c1 b1) st'' ->
    ceval st (CRepeat c1 b1) st''
```

(b) Write a Hoare proof rule for `REPEAT`.

Try to come up with a rule that is both sound and as precise as possible. For full credit, make sure your rule can be used to prove the following valid Hoare triple:

```
{{ Y <= m }}
REPEAT
  X ::= X + 1;
  IFB X <= m THEN Y ::= X ELSE SKIP END
UNTIL X > m END
{{ X > m /\ Y <= m }}
```

Answer:

```
Lemma hoare_repeat : forall P Q b c,
  {{ P }} c {{ Q }} ->
  {{ fun st => Q st /\ ~bassn b st }} c {{ Q }} ->
  {{ P }} (REPEAT c UNTIL b END) {{ fun st => Q st /\ bassn b st }}.
```

Grading scheme: Part a: 3 for each rule, deductions for minor mistakes or imprecision Part b: 3 if valid but not trivial, 6 for correct rule, 5 for rule that can prove example but not complete

7. **[Standard]** (18 points) The following Imp program calculates the sum and the difference of two numbers `n` and `m`:

```
  {{ n >= m /\ Y = n /\ X = m  }} ->>
Z ::= Y;
WHILE X > 0 DO
  X ::= X - 1;
  Y ::= Y + 1;
  Z ::= Z - 1
END
  {{ Z = n - m /\ Y = n + m }}
```

On the next page, add appropriate annotations to the program in the provided spaces to show that the Hoare triple given by the outermost pre- and post-conditions is valid. Use informal notations for mathematical formulae and assertions, but please be completely precise and pedantic in the way you apply the Hoare rules — i.e., write out assertions in *exactly* the form given by the rules (rather than logically equivalent ones). The provided blanks have been constructed so that, if you work backwards from the end of the program, you should only need to use the rule of consequence in the places indicated with `->>`.

The Hoare rules and the rules for well-formed decorated programs are provided on pages 14 and 15, for reference.

```
  {{ n >= m /\ Y=n /\ X=m  }} ->>
  {{ Y-X=n-m /\ Y+X=n+m}}
Z ::= Y;
  {{ Z-X=n-m /\ Y+X=n+m}}
WHILE X > 0 DO
  {{ Z-X=n-m /\ Y+X=n+m /\ X>0}} ->>
  {{ Z-1-(X-1)=n-m /\ Y+1+X-1=n+m}}
  X ::= X - 1;
  {{ Z-1-X=n-m /\ Y+1+X=n+m}}
  Y ::= Y + 1
  {{ Z-1-X=n-m /\ Y+X=n+m}}
  Z ::= Z - 1
  {{ Z-X=n-m /\ Y+X=n+m}}
END
  {{ Z-X=n-m /\ Y+X=n+m /\ X=0 }} ->>
  {{ Z=n-m /\ Y=n+m }}
```

Grading scheme: ~*2pt malus / wrong assertion (doesn't count propagation of errors)*

Loop condition at the loop exit (this should be $\sim (x > 0)$*). Having (x = 0) is 1pt malus, if this is the only mistake, or no malus if there are other mistake in the exercise*

8. (15 points) Suppose we've defined a Coq function `sort` that sorts lists of numbers. The following Imp program performs an analogous (though simpler) task: it sorts the numbers stored in the variables `X`, `Y`, and `Z`.

```
    {{ X=m /\ Y=n /\ Z=o }}
  WHILE X > Y \/ Y > Z DO
    IF X > Y THEN
      W := X;
      X := Y;
      Y := W
    ELSE
      SKIP
    FI;
    IF Y > Z THEN
      W := Y;
      Y := Z;
      Z := W
    ELSE
      SKIP
    FI
  END
    {{ sort[m,n,o] = [X,Y,Z] }}
```

On the next page, add appropriate annotations in the provided spaces to show that the Hoare triple given by the outermost pre- and post-conditions is valid.

The implication steps in your decoration may rely (silently) on the following facts about `sort`:

- If `l1` is a permutation of `l2` (i.e., they have the same elements, but perhaps not in the same order), then `sort l1 = sort l2`.
- If each element of `l` is less than or equal to the following element, then `sort l = l`.

```
    {{ X=m /\ Y=n /\ Z=o }} ->>
    {{ sort [m,n,o] = sort [X,Y,Z] }}
  WHILE X > Y \/ Y > Z DO
      {{ sort [m,n,o] = sort [X,Y,Z] /\ (X > Y \/ Y > Z) }} ->>
      {{ sort [m,n,o] = sort [X,Y,Z] }}
    IF X > Y THEN
        {{ sort [m,n,o] = sort [Y,X,Z] }} ->>
        {{ sort [m,n,o] = sort [X,Y,Z] }}
      W := X;
        {{ sort [m,n,o] = sort [Y,W,Z] }}
      X := Y;
        {{ sort [m,n,o] = sort [X,W,Z] }}
      Y := W
        {{ sort [m,n,o] = sort [X,Y,Z] }}
    ELSE
      SKIP
        {{ sort [m,n,o] = sort [X,Y,Z] }}
    FI;
      {{ sort [m,n,o] = sort [X,Y,Z] }} ->>
      {{ sort [m,n,o] = sort [X,Z,Y] }}
    IF Y > Z THEN
        {{ sort [m,n,o] = sort [X,Z,Y] }}
      W := Y;
        {{ sort [m,n,o] = sort [X,Z,W] }}
      Y := Z;
        {{ sort [m,n,o] = sort [X,Y,W] }}
      Z := W
        {{ sort [m,n,o] = sort [X,Y,Z] }}
    ELSE
      SKIP
        {{ sort [m,n,o] = sort [X,Y,Z] }}
    FI
      {{ sort [m,n,o] = sort [X,Y,Z] }}
  END
    {{ sort [m,n,o] = sort [X,Y,Z] /\ ~(X > Y \/ Y > Z) }} ->>
    {{ sort [m,n,o] = [X,Y,Z] }}
```

Grading scheme: 14 points minor mistakes, 12 points logic mistake but correct invariants, 8-12 correct sketch of invariants wrong use of some rules, 4-8 some ideas, 0-4 not clear idea.

9. [**Advanced**] (18 points) The following program implements "slow multiplication" in Imp.

```
    {{ True }}
  Y ::= 0;
  Z ::= 0;
  WHILE Y < n DO
```

```
      X ::= 0;
      WHILE X < m DO
        Z ::= Z + 1;
        X := X + 1
      END;
      Y ::= Y + 1
    END
      {{ Z = n*m }}
```

On the next page, add appropriate annotations to the program in the provided spaces to show that the Hoare triple given by the outermost pre- and post-conditions is valid.

```
      {{ True }} ->>
      {{ 0 = 0*m /\ 0 <= n }}
    Y ::= 0;
      {{ 0 = Y*m /\ Y <= n }}
    Z ::= 0;
      {{ Z = Y*m /\ Y <= n }}
    WHILE Y < n DO
        {{ Z = Y*m /\ Y <= n /\ Y < n }} ->>
        {{ Z = Y*m + 0 /\ (Y+1) <= n /\ 0 <= m }}
      X ::= 0;
        {{ Z = Y*m + X /\ (Y+1) <= n /\ X <= m }}
      WHILE X < m DO
          {{ Z = Y*m + X /\ (Y+1) <= n /\ X <= m /\ X < m}} ->>
          {{ Z + 1 = Y*m + X + 1 /\ (Y+1) <= n /\ X+1 <= m }}
        Z ::= Z + 1;
          {{ Z = Y*m + X + 1 /\ (Y+1) <= n /\ X+1 <= m }}
        X := X + 1
          {{ Z = Y*m + X /\ (Y+1) <= n /\ X <= m }}
      END;
        {{ Z = Y*m + X /\ (Y+1) <= n /\ X <= m /\ ~(X < m) }} ->>
        {{ Z = (Y+1)*m /\ (Y+1) <= n }}
      Y ::= Y + 1
        {{ Z = Y*m /\ Y <= n }}
    END
      {{ Z = Y*m /\ Y <= n /\ ~(Y < n) }} ->>
      {{ Z = n*m }}
```

Grading scheme: 17 points for minor mistakes, 15 points logic mistake but correct invariants, 10-15 correct sketch of invariants wrong use of some rules, 4-9 some ideas, 0-4 not clear idea.

10. [**Advanced**] (15 points) Recall the Hoare logic rule for `WHILE` loops:

$$\frac{\{\!\{\, P \wedge b \,\}\!\}\ \mathtt{c}\ \{\!\{\, P \,\}\!\}}{\{\!\{\, P \,\}\!\}\ \mathtt{WHILE\ b\ DO\ c\ END}\ \{\!\{\, P \wedge \sim b \,\}\!\}} \quad (\mathtt{hoare\_while})$$

Write a careful informal proof of its correctness.

Answer:

Suppose `st` is a state satisfying `P` and that `(WHILE b DO c END) / st` $\Downarrow$ `st'`. Proceed by induction on a derivation of `(WHILE b DO c END) / st` $\Downarrow$ `st'`. Because of the form of the program, there are just two cases to consider:

(a) `(WHILE b DO c END) / st` $\Downarrow$ `st'` by rule `E_WhileEnd`, with `st' = st` and `beval st b = false`. We know `P st'` by assumption, and the assertion $(\sim b)$ *st* follows by definition from the fact that `beval st b = false`, so `st'` satisfies the required postcondition.

(b) `(WHILE b DO c END) / st` $\Downarrow$ `st'` by rule `E_WhileLoop`, with `beval st b = true` and `c / st` $\Downarrow$ `st1` and `(WHILE b DO c END) / st1` $\Downarrow$ `st'`. By the first premise (using the fact that `beval st b = true` implies the assertion `b st`, plus the assumption that `P` holds for `st` and the definition of validity for Hoare triples), we have `P st1`. Now, by the IH (for the second premise), the assertion $P \wedge \sim b$ holds for the state `st'`, as required.

Formal definitions for Imp

Syntax

```
Inductive aexp : Type := | ANum : nat -> aexp | AId : id -> aexp |
APlus : aexp -> aexp -> aexp | AMinus : aexp -> aexp -> aexp | AMult :
aexp -> aexp -> aexp.

Inductive bexp : Type :=
  | BTrue : bexp
  | BFalse : bexp
  | BEq : aexp -> aexp -> bexp
  | BLe : aexp -> aexp -> bexp
  | BNot : bexp -> bexp
  | BAnd : bexp -> bexp -> bexp.

Inductive com : Type :=
  | CSkip : com
  | CAss : id -> aexp -> com
  | CSeq : com -> com -> com
  | CIf : bexp -> com -> com -> com
  | CWhile : bexp -> com -> com.

Notation "'SKIP'" :=
  CSkip.
Notation "l '::=' a" :=
  (CAss l a) (at level 60).
Notation "c1 ; c2" :=
  (CSeq c1 c2) (at level 80, right associativity).
Notation "'WHILE' b 'DO' c 'END'" :=
  (CWhile b c) (at level 80, right associativity).
Notation "'IFB' e1 'THEN' e2 'ELSE' e3 'FI'" :=
  (CIf e1 e2 e3) (at level 80, right associativity).
```

Evaluation relation

```
Inductive ceval : com -> state -> state -> Prop :=
  | E_Skip : forall st,
      SKIP / st || st
  | E_Ass  : forall st a1 n X,
      aeval st a1 = n ->
      (X ::= a1) / st || (update st X n)
  | E_Seq : forall c1 c2 st st' st'',
      c1 / st  || st' ->
      c2 / st' || st'' ->
      (c1 ; c2) / st || st''
  | E_IfTrue : forall st st' b1 c1 c2,
      beval st b1 = true ->
      c1 / st || st' ->
      (IFB b1 THEN c1 ELSE c2 FI) / st || st'
  | E_IfFalse : forall st st' b1 c1 c2,
      beval st b1 = false ->
      c2 / st || st' ->
      (IFB b1 THEN c1 ELSE c2 FI) / st || st'
  | E_WhileEnd : forall b1 st c1,
      beval st b1 = false ->
      (WHILE b1 DO c1 END) / st || st
  | E_WhileLoop : forall st st' st'' b1 c1,
      beval st b1 = true ->
      c1 / st || st' ->
      (WHILE b1 DO c1 END) / st' || st'' ->
      (WHILE b1 DO c1 END) / st || st''

  where "c1 '/' st '||' st'" := (ceval c1 st st').
```

Program equivalence

```
Definition bequiv (b1 b2 : bexp) : Prop :=
  forall (st:state), beval st b1 = beval st b2.

Definition cequiv (c1 c2 : com) : Prop :=
  forall (st st' : state),
    (c1 / st || st') <-> (c2 / st || st').
```

Hoare triples

```
Definition hoare_triple (P:Assertion) (c:com) (Q:Assertion) : Prop :=
  forall st st', c / st || st' -> P st  -> Q st'.

Notation "{{ P }} c {{ Q }}" := (hoare_triple P c Q).
```

Implication on assertions

```
Definition assert_implies (P Q : Assertion) : Prop :=
  forall st, P st -> Q st.

Notation "P ->> Q" := (assert_implies P Q) (at level 80).
```

(ASCII `->>` is typeset as a hollow arrow in the rules below.)

Hoare logic rules

$$\frac{}{\{\!\{\,\texttt{assn\_sub X a}\ Q\,\}\!\}\ \texttt{X := a}\ \{\!\{\,Q\,\}\!\}}\quad(\texttt{hoare\_asgn})$$

$$\frac{}{\{\!\{\,P\,\}\!\}\ \texttt{SKIP}\ \{\!\{\,P\,\}\!\}}\quad(\texttt{hoare\_skip})$$

$$\frac{\begin{array}{c}\{\!\{\,P\,\}\!\}\ \texttt{c1}\ \{\!\{\,Q\,\}\!\}\\ \{\!\{\,Q\,\}\!\}\ \texttt{c2}\ \{\!\{\,R\,\}\!\}\end{array}}{\{\!\{\,P\,\}\!\}\ \texttt{c1; c2}\ \{\!\{\,R\,\}\!\}}\quad(\texttt{hoare\_seq})$$

$$\frac{\begin{array}{c}\{\!\{\,P \wedge b\,\}\!\}\ \texttt{c1}\ \{\!\{\,Q\,\}\!\}\\ \{\!\{\,P \wedge \sim b\,\}\!\}\ \texttt{c2}\ \{\!\{\,Q\,\}\!\}\end{array}}{\{\!\{\,P\,\}\!\}\ \texttt{IFB b THEN c1 ELSE c2 FI}\ \{\!\{\,Q\,\}\!\}}\quad(\texttt{hoare\_if})$$

$$\frac{\{\!\{\,P \wedge b\,\}\!\}\ \texttt{c}\ \{\!\{\,P\,\}\!\}}{\{\!\{\,P\,\}\!\}\ \texttt{WHILE b DO c END}\ \{\!\{\,P \wedge \sim b\,\}\!\}}\quad(\texttt{hoare\_while})$$

$$\frac{\begin{array}{c}\{\!\{\,P'\,\}\!\}\ \texttt{c}\ \{\!\{\,Q'\,\}\!\}\\ P \rightarrowtriangle P'\\ Q' \rightarrowtriangle Q\end{array}}{\{\!\{\,P\,\}\!\}\ \texttt{c}\ \{\!\{\,Q\,\}\!\}}\quad(\texttt{hoare\_consequence})$$

$$\frac{\begin{array}{c}\{\!\{\,P'\,\}\!\}\ \texttt{c}\ \{\!\{\,Q\,\}\!\}\\ P \rightarrowtriangle P'\end{array}}{\{\!\{\,P\,\}\!\}\ \texttt{c}\ \{\!\{\,Q\,\}\!\}}\quad(\texttt{hoare\_consequence\_pre})$$

$$\frac{\begin{array}{c}\{\!\{\,P\,\}\!\}\ \texttt{c}\ \{\!\{\,Q'\,\}\!\}\\ Q' \rightarrowtriangle Q\end{array}}{\{\!\{\,P\,\}\!\}\ \texttt{c}\ \{\!\{\,Q\,\}\!\}}\quad(\texttt{hoare\_consequence\_post})$$

Decorated programs

(a) SKIP is locally consistent if its precondition and postcondition are the same:

```
{{ P }}
SKIP
{{ P }}
```

(b) The sequential composition of c1 and c2 is locally consistent (with respect to assertions P and R) if c1 is locally consistent (with respect to P and Q) and c2 is locally consistent (with respect to Q and R):

```
{{ P }}
c1;
{{ Q }}
c2
{{ R }}
```

(c) An assignment is locally consistent if its precondition is the appropriate substitution of its postcondition:

```
{{ P [X |-> a] }}
X ::= a
{{ P }}
```

(d) A conditional is locally consistent (with respect to assertions P and Q) if the assertions at the top of its "then" and "else" branches are exactly P /\ b and P /\ ~b and if its "then" branch is locally consistent (with respect to P /\ b and Q) and its "else" branch is locally consistent (with respect to P /\ ~b and Q):

```
{{ P }}
IFB b THEN
  {{ P /\ b }}
  c1
  {{ Q }}
ELSE
  {{ P /\ ~b }}
  c2
  {{ Q }}
FI
{{ Q }}
```

(e) A while loop with precondition `P` is locally consistent if its postcondition is `P /\ ~b` and if the pre- and postconditions of its body are exactly `P /\ b` and `P`:

```
{{ P }}
WHILE b DO
  {{ P /\ b }}
  c1
  {{ P }}
END
{{ P /\ ~b }}
```

(f) A pair of assertions separated by `->>` is locally consistent if the first implies the second (in all states):

```
{{ P }} ->>
{{ P' }}
```