TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2010

MID-TERM TEST

SWEN 224
Formal Foundations of Programming
WITH ANSWERS

Time Allowed: 90 minutes

Instructions: There are **four** (4) questions.

Answer **all** the questions.

The test will be marked out of **ninety** (90).

Calculators ARE NOT ALLOWED.

Non-electronic foreign language dictionaries are allowed.

No other reference material is allowed.

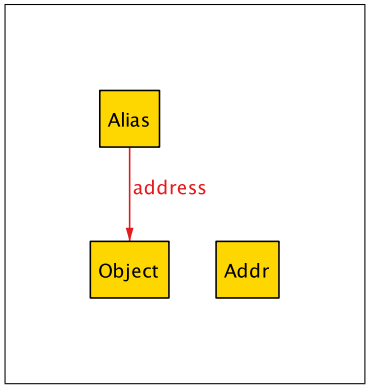
Question 1. Alloy Basics and Static Modelling [25 marks]

The following Alloy provides an incomplete model of an email address book that associates email addresses with names that are more convenient to use. An alias provides an alternative name for an address or name, while a group is a name used as a shorthand for a set of addresses and names. Thus, a name can refer to an address either directly or indirectly by referring to a name that refers to this address.

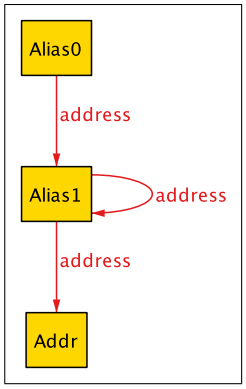
```
abstract sig Object {}
sig Addr extends Object {}
abstract sig Name extends Object { address: set Object }{ some address }
sig Alias, Group extends Name {}
pred p { all a: Alias | one a.address }
run {}
```

(a) [6 marks]

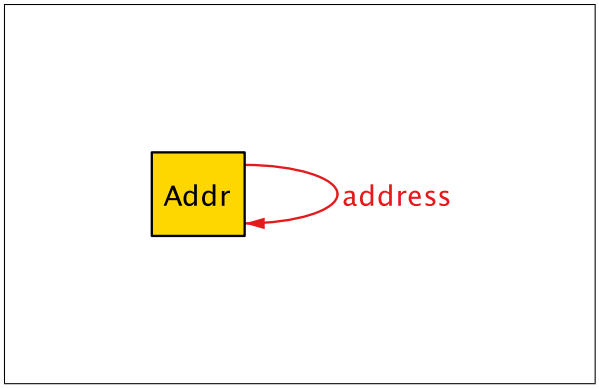
Which of the following instances could not be generated by the given run command? Give a brief explanation of your answer.



(a) Instance 1



(b) Instance 2



(c) Instance 3

1(a) is not a valid instance since Object is abstract (thus every Object is either a Name or an Addr).
1(c) is not a valid instance since `address` cannot relate an Addr to something.

Note that 1(b) is a valid instance. The predicate is not used by the run command and thus ignored.

(b) Write run commands for the following constraints, which one might add during an exploration of this model:

(i) [2 marks] There is a name but no address.

```
run { some Name and no Addr }
```

(ii) [2 marks] There is a least one name that is a member of a group.

```
run { some Name & Group.address }
```

(c) Consider the following instance:

$$\texttt{Addr} = \{A0, A1, A2\} \qquad \texttt{Alias} = \{Mum, Dad\} \quad \texttt{Group} = \{Family\}$$

$$\texttt{address} = \begin{array}{|c|c|}\hline Mum & A0 \\ \hline Dad & A1 \\ \hline Family & A2 \\ \hline Family & Mum \\ \hline Family & Dad \\ \hline \end{array}$$

(i) [1 mark] Compute `~address`

A0	*Mum*
A1	*Dad*
A2	*Family*
Mum	*Family*
Dad	*Family*

(ii) [2 marks] Compute `address.~address`

$$\texttt{address.\~{}address} = \begin{array}{|c|c|}\hline Mum & A0 \\ \hline Dad & A1 \\ \hline Family & A2 \\ \hline Family & Mum \\ \hline Family & Dad \\ \hline \end{array} . \begin{array}{|c|c|}\hline A0 & Mum \\ \hline A1 & Dad \\ \hline A2 & Family \\ \hline Mum & Family \\ \hline Dad & Family \\ \hline \end{array} = \begin{array}{|c|c|}\hline Mum & Mum \\ \hline Dad & Dad \\ \hline Family & Family \\ \hline \end{array}$$

(iii) [2 marks] Compute `address :> Addr`.

Mum	*A0*
Dad	*A1*
Family	*A2*

(iv) [2 marks] Compute `^address`

Mum	*A0*
Dad	*A1*
Family	*A2*
Family	*Mum*
Family	*Dad*
Family	*A0*
Family	*A1*

(d) Extending the Alloy Model

(i) [2 marks] Write a fact that ensures that a name cannot refer, neither directly nor indirectly, to itself.

```
fact { no n: Name | n in n.^address }
```

(ii) [3 marks] Write an Alloy function that returns the relation between a name and the addresses this name refers to, directly or indirectly.

```
fun res: Name->Addr { (^address) :> Addr }
```

(iii) [3 marks] Write an Alloy command to check whether all names eventually (directly or indirectly) resolve to an address.

```
check { all n: Name | some a: Addr | n->a in ^address }
```

Question 2. Dynamic Modelling in Alloy [20 marks]

(a) Local Versus Global State

(i) [6 marks] The Alloy model given below uses local state. Provide an alternative model of a file system that uses global state.

abstract sig Object { parent: **lone** Dir }
sig Dir, File **extends** Object {}
one sig Root **in** Dir {}{ **no** parent }
fact { Object **in** Root.*~parent }

```
abstract sig Object {}
sig Dir, File extends Object {}
sig FS {
  obj: set Object,
  root: obj & Dir,
  parent: (obj-root)->one (obj & Dir)
}{
  obj in root.*~parent
}
```

(ii) [4 marks] Briefly explain the differences between using local versus global state. What are the advantages and disadvantages of using one over the other?

(b) Modelling Operations

Consider the following model of a Map that associates unique keys with values.

sig Key, Value {}
sig Map {
  keys: **set** Key,
  values: keys −> Value
}

pred inv[m:Map] {
  all k:Key | **lone** m.values[k]
}

(i) [6 marks] Provide an operation that models adding a new key-to-value mapping and preserves the invariant `inv` provided.

```
// m refers to the pre-state
// m' refers to the post-state
// k and v are inputs
pred add[m,m': Map, k: Key, v: Value] {
  k !in m.keys
  m'.keys = m.keys + k
  m'.values = m.values + k->v
}
```

(ii) [4 marks] Write an Alloy command to check whether your operation given in **(i)** preserves invariant `inv`.

```
check { all m,m': Map, k: Key, v: Value |
  inv[m] and add[m,m',k,v] implies inv[m']
}
```

Question 3. JML [20 marks]

(a) Understanding JML Specification

Do the following methods correctly implement their specification? Give a brief explanation why you think they do or do not.

(i) [2 marks]

```
//@ requires x != 0;
//@ ensures \result == 1 || \result == 2;
int foo(int x) {
  return 2;
}
```

Yes, since the method always returns 2, which satisfies the postcondition.

(ii) [2 marks]

```
//@ requires a != null;
//@ ensures (\forall int k; 0 < k && k < a.length; a[k-1] <= a[k]);
void sort(int[] a) {
  for (int i = 0; i < a.length; i++) {
    a[i] = 0;
  }
}
```

Yes, since the array a contains just zeros when the method returns and thus satisfies the postcondition.

(iii) [2 marks]

```
//@ requires true;
//@ ensures false;
boolean bar() {
  return false;
}
```

No, since the postcondition cannot be satisfied.
Note that it would satisfy the postcondition `\result == false`.

(b) Writing JML Specifications

Write a JML specification for each of the following methods.

(i) [4 marks] `int find(int[] a, int value)`

Given a non-null integer array `a` and an integer `value` that is guaranteed to occur in `a`, method `find` returns the first index position for the given `value` in `a`.

```
//@ requires a != null;
//@ requires (\exists int i; 0 <= i && i < a.length; a[i] == value);
```

```
//@ ensures a[\result] == value;
//@ ensures (\forall int i; 0 <= i && i < \result; a[i] != value);
```

(ii) [5 marks] `int[] replace(int[] a, int x, int y)`

Given integers `x` and `y`, and a non-null integer array `a`, the array returned by method `replace` is the same as `a` with all occurrences of `x` replaced by `y`.

```
//@ requires a != null;
//@ ensures \result != null && \result.length == a.length;
/*@ ensures (\forall int i; 0 <= i && i<a.length;
       (a[i] == x && \result[i] == y) || (a[i] != x && \result[i]==a[i])); */
```

(c) Class Invariants

(i) [5 marks] Explain what a class invariant is, and what it means for a method to preserve a class invariant.

Question 4. Program Verification [25 marks]

(a) [10 marks]

Prove that the following program satisfies its specification by annotating it with appropriate JML assertions.

```
//@ requires true;
//@ ensures \result <= x && \result <= y;
int min(int x, int y) {
  int r = x;
  if (0 < y - r) {
    r = x;
  }
  else {
    r = y;
  }
  return r;
}
```

```
//@ ensures \result <= x && \result <= y;
int min(int x, int y) {
  //@ assert true;
  //@ assert_redundantly x == x;
  int r = x;
  //@ assert r == x;
  if (0 < y - r) {
    //@ assert 0 < y - r;
    //@ assert r == x;
    //@ assert_redundantly x <= x && x <= y;
    r = x;
    //@ assert r <= x && r <= y;
  }
  else {
    //@ assert ! (0 < y - r);
    //@ assert r == x;
    //@ assert_redundantly y <= x && y <= y;
    r = y;
    //@ assert r <= x && r <= y;
  }
  //@ assert r <= x && r <= y;
  return r;
}
```

(b) [15 marks]

Consider a Java method for computing the minimum element of a non-empty array:

```
//@ requires a != null;
//@ requires a.length > 0;
```

```
//@ ensures (\exists int k; 0 <= k && k < a.length; \result == a[k]);
//@ ensures (\forall int k; 0 <= k && k < a.length; \result <= a[k]);
int min(int[] a) {
  int r = a[0];
  int i = 1;
  while (i < a.length) {
    if (a[i] < r) {
      r = a[i];
    }
    i = i + 1;
  }
  return r;
}
```

(i) [6 marks] Provide a loop invariant that may be used in showing that the above method satisfies its specification.

(ii) [5 marks] Explain the steps that are required to prove that your loop invariant from part **(i)** is correct. You do *not* need to give the proof.

(iii) [4 marks] Assuming your loop invariant from part **(i)** is correct, demonstrate that it is strong enough to prove that the method satisfies its specification.

****