TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI



EXAMINATIONS - 2009

END-OF-YEAR

COMP 202 / SWEN 202 Formal Methods of Computer Science / Formal Foundations of Software Engineering

Time Allowed: 3 Hours

Instructions: • Answer all five questions.

- The exam will be marked out of one hundred and eighty (180).
- Calculators ARE NOT ALLOWED.
- Non-electronic Foreign language dictionaries are allowed.
- No other reference material is allowed.

Question 1. Alloy

Consider the following Alloy specification, which models items taken on a trip. Since some of the items may be containers, such as bags or suitcases, we also record whether one item is inside other item.

```
sig Item {}
sig Item {}
sig Trip {
   items: set Item,
   isInside: items->items
}
pred loneLoc[t: Trip] {
   all i: t.items | lone t.isInside[i]
}
pred transLoc[t: Trip] {
   t.isInside = ^(t.isInside)
}
pred addItem[t,t': Trip, i: Item] {
   t'.items = t.items + i
}
```

Now consider the following instance of this model:

Item	=	{Bag,Coin,Wallet}					
Trip	=	$\{T0, T1, T2\}$					
items	_	T0BagT0WalletT1BagT1WalletT2BagT2CoinT2Wallet					
isInside	=	T0WalletBagT1WalletBagT1BagWalletT2WalletBagT2CoinBagT2CoinWallet					

[40 marks]

- (a) [13 marks] Understanding Alloy.
- (i) [1 mark] Compute T2.isInside.

Wallet	Bag
Coin	Bag
Coin	Wallet

(ii) [1 mark] Compute ^(T2.isInside).



(iii) [2 marks] Is the predicate loneLoc[T2] true for this instance? Briefly explain why or why not.

Predicate loneLoc[T2] is not true for this instance since *Coin* is both inside *Bag* and *Wallet* for trip T2.

(iv) [2 marks] In your own words, describe which trips satisfy the predicate loneLoc.

A trip satisfies predicate loneLoc if every item taken to that trip is inside at most one item for that trip.

(v) [2 marks] Is the predicate transLoc[T2] true for this instance? Briefly explain why or why not.

The predicate transLoc[*T2*] is true for this instance since *T2*.isInside = (T2.isInside) (see answers to questions i and ii).

(vi) [2 marks] In your own words, describe which trips satisfy the predicate transLoc.

A trip satisfies predicate transLoc if the relation isInside for this trip is equivalent to the transitive closure of the isInside relation for this trip. That is, if an item *i* taken to a trip is inside an item *j*, which in turn is inside an item *k*, then *i* must also be inside *k* for this trip.

(vii) [1 mark] Compute T0.items + Wallet.



(viii) [2 marks] Is the predicate addItem[T0, T0, Wallet] true for this instance? Briefly explain why or why not.

The predicate addItem[T0, T0, Wallet] is true for this instance since T0.items = T0.items + Wallet (see answer to question vii).

(b) [7 marks] Writing Alloy.

(i) [1 mark] Provide a run command that shows only trips for which the predicate loneLoc holds.

run { all t: Trip | loneLoc[t] }

(ii) [2 marks] Write a predicate called noCycles that is true if, for a given trip, an item cannot be inside itself (neither directly nor indirectly).

pred noCycles[t: Trip] { all i: Item | i->i !in ^(t.isInside) }

(iii) [4 marks] Can a trip with an item that both contains an item and is inside another item satisfy both predicates loneLoc and transLoc at once? Write an Alloy assertion that can be used to check your answer.

Assuming cycles are not allowed, such a trip cannot exist.

(c) [8 marks] Adding items.

(i) [3 marks] Write an Alloy command to check that the operation addItem preserves the invariant noCycles.

```
check { all t,t': Trip, i: Item |
    noCycles[t] and addItem[t,t',i] implies noCycles[t'] }
```

(ii) [3 marks] Does the operation addItem preserve the invariant noCycles? Explain why or why not.

The operation addItem does not preserve the invariant noCycles because the *isInside* relation is not constrained by the operation so can introduce cycles.

(iii) [2 marks] The given addItem operation allows an item to be added that is already an item of the trip. How could you improve the operation so that it only allows new items to be added that are not yet items of the trip?

By adding the constraint/precondition i !in t.items to the operation.

(d) [12 marks] Packing items.

(i) [5 marks] Write an operation that models putting an item into another item that preserves the invariants noCycles and loneLoc.

```
pred put[t,t': Trip, item, into: Item] {
  item != into // Can't put into itself.
  no t.isInside[item] // Not yet inside something.
  into->item !in ^(t.isInside) // Item to put into cannot be inside item.
  t'.items = t.items
  t'.isInside = t.isInside + item->into
}
```

(ii) [7 marks] Write an operation that models putting an item into another item that preserves the invariants noCycles and transLoc.

```
pred put[t,t': Trip, item, into: Item] {
  item != into // Can't put into itself.
  no t.isInside[item] // Not yet inside something.
  item !in t.isInside[into] // Item to put into cannot be inside item.
  t'.items = t.items
  t'.isInside = ^(t.isInside + item->into)
}
```

Question 2. JML

[40 marks]

(a) [10 marks] **Pre- and Postconditions.**

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

(i) [2 marks]

```
//@ requires true;
//@ ensures true;
int magicNumber() {
  return 42;
}
```

Yes, since the method always terminates and true (the postcondition) is satisfied when the method finishes.

(ii) [2 marks]

```
//@ requires true;
//@ ensures false;
int magicNumber() {
  return 42;
}
```

(iii) [2 marks]

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

No, since the postcondition is not (and cannot be) satisfied.

```
(iv) [2 marks]
```

```
//@ requires size > 5;
//@ ensures \result != null && \result.length > 5;
int[] intArray(int size) {
  return new int[size];
}
```

Yes, since the method always returns a non-null array of size greater than 5 provided that an integer greater than 5 was provided as an input.

```
(v) [2 marks]
```

```
//@ requires a != null;
/*@ ensures a != null &&
    (\ forall int i; 0 < i && i < a.length; a[i] >= a[i-1]); @*/
```

```
void sort(int[] a) {
  for(int i = 0; i < a.length; i++) {
     a[i] = i;
   }
}</pre>
```

Yes, since provided a non-null array is given as argument, the method returns a non-null array that is sorted.

(b) [12 marks] Class Invariants.

Consider the following Java class to represent the time of day using 24 hour format.

```
public class TimeOfDay
{
    private int hour;
    private int min;
    public TimeOfDay(int hour, int min) { ... }
    public void setHour(int hour) { ... }
    public void setMinute(int min) { ... }
}
```

(i) [2 marks] Explain what a class invariant is.

(ii) [2 marks] Give a class invariant for the above TimeOfDay class using JML notation, which restricts the values of hour and min to 24 hour format.

(iii) [4 marks] Suppose you want to check class invariants at run-time but you do not have JML tools installed. Add assertions using Java's assert keyword into the TimeOfDay class that check the constraints imposed by the invariant from part (ii), making clear exactly where in the code your assertions are to be added.

(iv) [2 marks] Assume you implement a class that extends TimeOfDay. What are the requirements on the class invariant for this subclass?

(v) [2 marks] Escj performs compile time checking of JML annotations. Sometimes escj warns about code that is correct with respect to the JML annotations, that is, it gives false positives. Do false positives occur when you use jmlrac to perform run-time checking of JML annotations? Explain your answer.

(c) [18 marks] Loop Invariants and Variants.

Consider the following Java method:

```
public static boolean binarySearch(int[] a, int target)
  int gazeUp = a.length - 1;
  int gazeDown = 0;
  while (gazeUp >= gazeDown) {
    int gaze = gazeDown+(gazeUp-gazeDown)/2;
    if (a[gaze] == target) {
      return true;
    }
    if (target < a[gaze]) {
      gazeUp = gaze - 1;
    } else {
      gazeDown = gaze+1;
    }
  }
  return false;
}
```

This method implements a binary search. It takes a sorted integer array and a target integer as input, and returns true if the target integer is one of the integers in the provided array, and false otherwise.

(i) [4 marks] Give a JML specification (precondition and postcondition) that formalises the above description of the binarySearch method.

(ii) [8 marks] Provide a loop invariant and explain how it can be used to prove that the method satisfies its specification. You do *not* need to give the proof.

(iii) [6 marks] Give a loop variant and an argument (informal proof) to show that the method terminates.

Question 3. Regular Languages

(a) [5 marks] Write a regular expression that defines the set of all strings over $\{a, b, c\}$ whose length is a multiple of three.

(b) [10 marks] Draw a transition diagram for a finite acceptor that recognises all strings over {1,2,3,\$} which start and end with a \$, and in which any two \$'s are separated by a non-empty ascending sequence of digits (for example the strings "\$123\$" and "\$1\$3\$13\$" are in this languages, but "\$\$" and "\$312\$" are not).

(c) [25 marks] Consider the NFA $M = (Q, q_I, A, N, F)$, where:

• $Q = \{1, 2, 3, 4\}$	N	а	b	С
• $q_I = 1$	1	{1,2,3}	{1,4}	Ø
	2	Ø	{2,4}	Ø
• $A = \{a, b, c\}$	3	Ø	Ø	{3,4}
• $F = \{4\}$	4	Ø	Ø	{3,4}

and *N* is given by the table on the right. For example, this means that $N(1, a) = \{1, 2, 3\}$ and $N(3, b) = \emptyset$ (the empty set).

(i) [5 marks] Draw a transition diagram for *M*.

(ii) [5 marks] Show a sequence of configurations giving all states that *M* could be in at each step while reading the input "*aaabbcc*".

(iii) [10 marks] Draw a transition diagram for a DFA which is equivalent to *M*. Explain the relationship between the states of your DFA and those of *M*.

(iv) [5 marks] Write a regular expression that defines the language accepted by *M*.

Question 4. Context Free Languages

[40 marks]

Consider the following grammar:

$$S \rightarrow if B then S |$$

$$S; S |$$

$$\{S\} |$$

$$A$$

$$A \rightarrow a1 | a2 | a3$$

$$B \rightarrow b1 | b2 | b3$$

(a) [6 marks] Draw a parse tree for each of the following sentences:

```
(i) if b1 then {a1;a2}
```

(ii) if *b*1 then if *b*2 then *a*1

(**b**) [6 marks] Explain, with reference to the following sentence, what it means for a grammar to be *ambiguous*:

if *b*1 **then** *a*1;*a*2

(c) [10 marks] Explain, with reference to the above grammar, what it means for a grammar to be in LL(1) form. Write an equivalent grammar in LL(1) form and show that it is an LL(1) grammar.

(d) [18 marks] Explain how you would construct a recursive descent parser from your grammar in part (c), and give pseudo-code for the resulting parser.

Question 5. Program Equivalence

[20 marks]

(a) [5 marks] Explain briefly what it means for two programs to be:

- (i) strongly equivalent
- (ii) weakly equivalent

(b) [15 marks] It is possible to remove all **if** statements from a While program, replacing them by **while** statements as follows.

Each if statement, if *B* then S_1 else S_2 fi, is replaced by the code fragment:

b := true;while b and B do $S'_1; b := false$ od; while b and not B do $S'_2; b := false$ od

where: • S'_1 and S'_1 are the results of applying the transformation to S_1 and S_2 , respectively;

- a new boolean variable *b*, not occurring anywhere else in the program, is used for each **if** statement translated; and
- all other statements remain unchanged.

(i) [10 marks] Show that the result of applying this transformation to a program *P* is weakly equivalent to *P*.

(ii) [5 marks] Explain why the result of applying this transformation to a program *P* is not strongly equivalent to *P*.
