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

EXAMINATIONS — 2012

END-OF-YEAR

SWEN 224
Formal Foundations of Programming
WITH ANSWERS

Time Allowed: 3 Hours

Instructions:

- Answer all **four** 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. Assertions and Specification [45 marks]

(a) [3 marks : 1 for each part] Consider the following assertion:

$$(x < u \wedge y < v) \vee (x > u \wedge y > v)$$

For each of the following combinations of value, say whether the assertion is true or false for those values:

(i) $x = 1, y = 5, u = 3, v = 8$ true

(ii) $x = 4, y = 6, u = 3, v = 3$ true

(iii) $x = 5, y = 3, u = 5, v = 3$ false

(b) [6 marks : 2 for each part] Consider the following assertion:

$$(\forall i : 1..|A| - 1 \bullet A[i] = 0 \Rightarrow A[i+1] \neq 0)$$

where A is assumed to be an array of numbers, with indexes starting from 1.

For each of the following arrays, say whether the assertion is true or false for that array:

(i) $A = (0, 2, 0)$ true

(ii) $A = (0, 0)$ false

(iii) $A = (0)$ true

(c) [20 marks : 5 for each part] For each of the following statements, write an *assertion* formalising the statement. Assume that x, y and z are integers, and A is an integer array of size N, with indexes starting from 1.

(i) x, y and z are all different.

$x \leq y \wedge y \neq z \wedge x \neq z$

(ii) A contains at least one occurrence of x.

$\exists i : 1 \leq i \leq N \bullet A[i] = z$

or $|\{i \mid 1 \leq i \leq N \wedge A[i] = z\}| \geq 1$

Ok to use $|A|$ instead of N; likewise in (iii) and (iv).

(iii) Some element of A occurs at least twice.

$\exists i, j : 1 \leq i, j \leq N \wedge i \neq j \bullet A[i] = A[j]$

or $\exists i, j : 1 \leq i < j \leq N \bullet A[i] \neq A[j]$

(iv) A contains exactly two occurrences of z.

$$\exists i, j : 1 \leq i < j \leq N : A[i] = z \wedge A[j] = z \wedge (\forall k : 1 \leq k \leq N \wedge k \neq i \wedge k \neq j : A[k] \neq z)$$
$$\text{or} \quad |\{i \mid 1 \leq i \leq N \wedge A[i] = z\}| = 2$$

(d) [16 marks : 8 for each part] Give a formal specification (pre- and postcondition) for each of the programs described below.

(i) A program that takes an array, A, of non-negative numbers and returns two outputs, s and z, where s is the sum of the elements of A, and z is the number of zero elements in A.

Pre: $(\forall i : 1.. \mid A \mid \bullet A[i] \geq 0)$

Post: $s = \sum_{i=1}^{|A|} A[i] \wedge n =| \{i \mid 1 \leq i \leq | A | \wedge A[i] = 0\} |$

(ii) A program that takes two arrays, A and B, where A and B are both in ascending order and have no elements in common, and returns an array C which is in ascending order and contains all of the elements of A and B.

Define $asc(X) = (\forall i : 1 \leq i <| X | \bullet X[i] < X[i+1])$
and $elts(X) = \{x \mid \exists i : 1 \leq i \leq | X | \bullet x = X[i]\}$

Pre: $asc(A) \wedge asc(B) \wedge elt(A) \cap elts(B) = \varnothing$

Post: $asc(C) \wedge elts(C) = elts(A) \cup elts(B)$

Question 2. Program Verification [45 marks]

(a) [18 marks : 6 for each part] For each of the following correctness assertions, write down the *verification condition(s)* that must hold in order for the correctness assertion to be valid, and give a brief explanation of why these verification conditions hold.

(i) $\{ k = 1 \}\, s := A[1]\, \{ s = \sum_{i=1}^{k} A[i] \}$

$k = 1 \Rightarrow A[1] = \sum_{i=1}^{k} A[i]$

This holds because $\sum_{i=1}^{1} A[i] = A[1]$.

(ii) $\{ \mathit{true} \}$ **if** $x > 0$ **then** $z := z + x$ **else** $z := z - x$ **fi** $\{ z \geq z_0 \}$

$\mathit{true} \wedge x > 0 \Rightarrow z + x \geq z$

i.e. $x > 0 \Rightarrow x \geq 0$

$\mathit{true} \wedge \neg\, (x > 0) \Rightarrow z - x \geq z$

i.e. $x \leq 0 \Rightarrow x \leq 0$

(iii) $\{ 0 \leq k <\!| A |\wedge\, s = \sum_{i=1}^{k} A[i] \}\, k := k{+}1;\ s := s{+}A[k]\, \{ 0 \leq k \leq\!| A |\wedge\, s = \sum_{i=1}^{k} A[i] \}$

$0 \leq k <\!| A |\wedge\, s = \sum_{i=1}^{k-1} A[i] \Rightarrow 0 \leq k+1 \leq\!| A |\wedge\, s + a[k+1] = \sum_{i=1}^{k+1} A[i]$

This holds because:

- $0 \leq k$ implies $0 \leq k+1$
- $k <\!| A |$ implies $k+1 <\!| A |$
- $s = \sum_{i=1}^{k-1} A[i]$ implies $s + a[k+1] = \sum_{i=1}^{k+1} A[i]$
  ($0 \leq k <\!| A |$ ensures that $a[k+1]$ is well-defined)

(b) The following correctness assertion says that the given loop sets r to n factorial, provided that n is positive, and k and r are initially both equal to one. Note that n, k and r are assumed to be integers, and that n factorial is the product of the first n positive integers, i.e. $n! = \times_{i=1}^{n} i$.

$$\{ n \geq 1 \wedge k = 1 \wedge r = 1 \}\ \textbf{while}\ k < n\ \textbf{do}\ k := k+1;\ r := r \times k\ \textbf{od}\ \{ r = n! \}$$

To verify the loop, we introduce the following *loop invariant*, which says that whenever execution reaches the top of the loop, k is between 1 and n (inclusive), and r is k factorial.

$$1 \leq k \leq n \wedge r = k!$$

(i) [18 marks: 6 for each condition and its justification] State the three verification conditions that must be proved in order to show that the loop is partially correct, and give a brief argument to show that each of them holds.

- The precondition for the loop implies the loop invariant.

  $$n \geq 1 \wedge k = 1 \wedge r = 1 \Rightarrow 1 \leq k \leq n \wedge r = k!$$

- The loop invariant is preserved when the loop test holds.

  $$\{\, 1 \leq k \leq n \wedge r = k! \wedge k < n \,\}\, k := k + 1;\ r := r \times k \,\{\, 1 \leq k \leq n \wedge r = k! \,\}$$

  To prove this, we must prove the verification condition:

  $$1 \leq k \leq n \wedge r = k! \wedge k < n \Rightarrow 1 \leq k + 1 \leq n \wedge r \times (k + 1) = (k + 1)!$$

- The loop invariant implies the postcondition when the loop test fails.

  $$1 \leq k \leq n \wedge r = k! \wedge k \not< n \Rightarrow r = n!$$

(ii) [4 marks] Give a brief argument to show that the method terminates properly. (You may ignore the possibility of arithmetic overflow.)

The program cannot loop forever because it terminates after n iterations. We can prove this formally using $n + 1 - i$ as a loop variant. This value decreases on each iteration of the loop, and cannot go negative, since the loop will exit when it becomes zero.

(c) [5 marks] Consider the following correctness assertion, which says that the given code sets m to the largest value in the array A, and n to the number of times that m occurs in A, provided that A is non-empty.

$\{\ |A| > 0\ \}$
$k := 1;\ m := A[1];\ n := 1;$
while $k < |A|$ **do**
  if $A[k] > m$ **then**
    $m := A[k];\ n := 1$
  else if $A[k] = m$ **then**
    $n := n + 1$
  fi fi;
  $k := k + 1$
od
$\{\, m = max_{i=1}^{|A|} A[i] \wedge n = |\,\{i \mid 1 \leq i \leq |A| \wedge m = A[i]\}\,| \,\}$

Give a loop invariant that could be used to verify this loop.

$1 \leq k \leq |A| \wedge m = max_{i=1}^{k} A[i] \wedge n = |\,\{i \mid 1 \leq i \leq k \wedge m = A[i]\}\,|$

Question 3. Regular Languages [45 marks]

(a) [12 marks : 4 for each part] Write a *Regular Expression* to describe each of the following languages, over the alphabet $\{a, b, c\}$:

(i) All strings consisting of one or more a's, followed by one b, then zero or more c's.

aa^*bc^*

(ii) All strings containing exactly two b's.

$(a \mid c)^*b(a \mid c)^*b(a \mid c)^*$

(iii) All strings which do not contain three consecutive a's.

$(\lambda \mid a \mid aa)((b \mid c)(b \mid c)^*(a \mid aa))^*$

(b) Consider the NFA $M = (Q, q_I, A, N, F)$, where:

$$
\begin{aligned}
Q &= \{1, 2, 3, 4, 5, 6\} \\
q_I &= 1 \\
A &= \{\mathrm{a}, \mathrm{b}\} \\
N &= \{(1,a,2), (1,a,3), (2,b,2), (2,b,4), (2,b,5), (3,a,3), \\
&\quad\ (3,a,6), (3,b,5), (4,a,1), (4,a,5), (6,b,1), (6,b,5)\} \\
F &= \{2, 5\}
\end{aligned}
$$

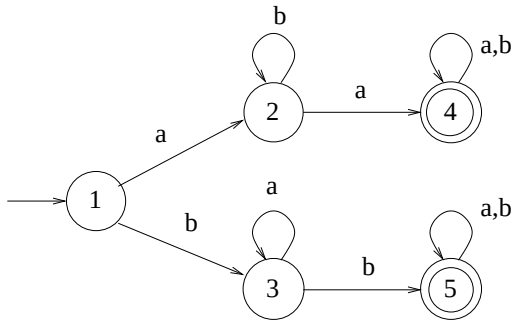
(i) [4 marks] Draw a transition diagram for M.

(ii) [5 marks] Show the sequence of configurations that M passes through in accepting the input *abbba*.

Note that you should show *all* states that M may be in after accepting part of the input.

States	*Input*	
$\{1\}$	*abbba*	
$\{\underline{2}, 3\}$	*bbba*	
$\{\underline{2}, 4, \underline{5}\}$	*bba*	
$\{\underline{2}, 4, \underline{5}\}$	*ba*	
$\{\underline{2}, 4, \underline{5}\}$	*a*	
$\{1, \underline{5}\}$	λ	*Accept*

(iii) [8 marks] Draw a transition diagram for the DFA obtained by applying the *subset construction* to M. Show the correspondence between states of the DFA and those of the NFA. You only need to show reachable states.



The state correspondence is:

DFAState	*NFAStates*
1	$\{1\}$
2	$\{1,2\}$
3	$\{1,2,3\}$
4	$\{1,2,3,4\}$
5	$\{1,2,3,4\}$

(c) [6 marks] Draw a transition diagram for an NFA$_\epsilon$ that accepts the language defined by the regular expression $a^*(b^*c^* \mid c^*d^*)^*a^*$.

(d) [10 marks] Given a DFA $M = (Q, q_I, A, N, F)$, show how to construct a DFA, $M' = (Q', q_I', A', N', F')$, which accepts the *complement* of $\mathcal{L}(M)$, i.e. the language consisting of all strings in A^* which are *not* accepted by M.

Give a brief argument to show that the resulting DFA does in fact accept the required language.

First, ensure that M is complete, adding an error state if necessary. Then define M' as:

$M' = (Q, q_I, A, N, Q - F)$

Question 4. Context-Free Languages [45 marks]

(a) Consider the language of simple method headers, where a method header consists of a method name, followed by a (possibly empty) list of argument declarations, separated by commas and enclosed in brackets. An argument declaration is an argument name followed by a colon and a type name. Method names, argument names and type names are identifiers. Identifiers should be treated as terminal symbols and do not need to be defined further.

For example, the following are sentences in this language, assuming that sequences of letters are identifiers:

- `f()`
- `sum(x: int, y: int)`
- `draw(a: float, b: char, c: bool)`

(i) [10 marks] Write a *Context Free Grammar* to describe this language. You are not required to give a full formal definition of this grammar — just write the list of rules.

$S \to id(T)$
$T \to \lambda \mid U$
$U \to id : id \mid id : id, U$

(ii) [5 marks] Draw a parse tree for the following method header using your grammar:

```
sum(x: int, y: int)
```

(b) Consider the following grammar, where "!", "$", "#", "(", ")" and "@" are terminal symbols:

$S \to T\,!$
$T \to T\,\$\,T \mid T\,\#\,T \mid (\,T\,) \mid @$

(i) [6 marks] Demonstrate that the grammar is ambiguous by drawing two different parse trees for "@#@$@!".

(ii) [8 marks] Write an equivalent, unambiguous grammar, treating $ as *left-associative*, # as *right-associative*, and giving # higher precedence than $.

$S \to T\,!$
$T \to T\,\$\,U \mid U$
$U \to V\,\#\,U \mid V$
$V \to @ \mid (\,S\,)$

(iii) [10 marks: 6 for the grammar, 4 for the proof] Write an LL(1) grammar which is equivalent to the given grammar, and show that it is LL(1). Note that in writing this grammar, you do **not** need to consider associativity or precedence.

$$
\begin{array}{lcl}
S & \rightarrow & T\,! \\
T & \rightarrow & @\,T' \mid (\,T\,) \\
T' & \rightarrow & \lambda \mid \$\,T \mid \#\,T
\end{array}
$$

In the definition of T, $\mathit{first}(@T') = \{@\}$ and $\mathit{first}((\,T\,)) = \{(\}$, which are disjoint.

In the definition of T', $\mathit{first}(\lambda) = \{\}$, $\mathit{first}(\$T) = \{\$\}$, and $\mathit{first}(\#T') = \{\#\}$, which are pairwise disjoint.

$T' \Rightarrow^* \lambda$, so we have to show that $\mathit{first}(T') \cap \mathit{follow}(T') = \varnothing$. Now, $\mathit{first}(T') = \{\$, \#\}$ and $\mathit{follow}(T') = \{!,)\}$, which are indeed disjoint.

Thus, the grammar is LL(1).

(c) [6 marks] Let $G = (V_N, V_T, S, P)$ be a Context Free Grammar, and let N be a non-terminal symbol in V_N such that $N \neq S$ and N does not occur on the left-hand side of any rule in P.

Show that deleting N from V_N and deleting from P any rule containing N does not alter the language defined by the grammar.

If N is not S and does not appear on the left-hand side of any rule in P, then N cannot occur in any parse tree constructed from G. Therefore, if we deleted N and any rules containing N from the grammar, we will still be able to construct the same parse trees.

More formally, let G' be the grammar obtained from G by deleting N from V_N and deleting from P any rule containing N; i.e. $G' = (V_N - \{N\}, V_T, S, P')$, where $P' = P - \{X \rightarrow \alpha \in P \mid N \in \alpha\}$ (where $N \in \alpha$ is shorthand for $\exists\, i \in 1.. \mid \alpha \mid\bullet\ \alpha_i = N$). We need to show that $\mathcal{L}(G) = \mathcal{L}(G')$.

Let w be a string in $\mathcal{L}(G)$, then there is a parse tree T for w from G. But since N is not S and does not appear on the left-hand side of any rule in P, N cannot occur in T. Thus, T is also a parse tree for w from G', which means that w is in $\mathcal{L}(G)$.

Similarly, if w is a string in $\mathcal{L}(G')$, there is a parse tree T for w from G'. In this case, T is trivially also a parse tree for w from G, because every nonterminal and rule of G' is also in G. So w is in $\mathcal{L}(G)$.

****