

EXAMINATIONS – 2008

END-OF-YEAR

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

Time Allowed: 3 Hours

Instructions: • There are six (6) questions, each worth 30 marks.

- Answer all of the 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. Finite acceptors

(a) [6 marks] Draw a transition diagram for an NFA (Nondeterministic Finite Acceptor) that accepts the language defined by the regular expression $a^*(ab|acd)^*a^*$.

You are not required to use a systematic construction. Your NFA should be as simple as possible, and may include null transitions.

(b) [4 marks] Give a trace (sequence of transitions) showing the behaviour of your NFA when given the string *aabacdaaa* as input. Show all possible states that the NFA could be in at each step.

(c) [8 marks] Draw a transition diagram for a DFA (Deterministic Finite Acceptor) equivalent to your NFA from part (a).

Explain how your DFA was constructed and how states of the DFA are related to those of the NFA.

(d) [2 marks] Give a trace (sequence of transitions) showing the behaviour of your DFA when given the string *abcda* as input.

(e) [10 marks] Prove that an NFA accepts a finite language if and only if its transition diagram, considered as a directed graph, contains no cycles consisting only of "useful" states. Recall that a state is "useful" if it lies on a path from the initial state to some accepting state. Explain why the result is false if the "useful" states condition is omitted.

Question 2. Describing languages

(a) [22 marks] Consider the following languages:

- (i) The set of strings over $\{a, b, c\}$ containing exactly one *a* and exactly one *b*.
- (ii) The set of strings of the form $a^m b^n$, where m < n.
- (iii) The set of all palindromes over $\{a, b, c\}$; i.e. the set of all strings $\alpha \in \{a, b, c\}^*$ such that $\alpha^R = \alpha$, where α^R is the reflection (or reverse) of α .
- (iv) The set of strings over $\{a, b, c, d\}$ such that no *c* occurs before an *a* and no *b* occurs after a *d*.

For each of the above languages: say whether the language is regular or not regular. If the language is regular, write a regular expression that defines it; otherwise, prove that it is not regular.

(b) [8 marks] The following grammars all define a simple language of arithmetic expressions with binary operators + and *, and numbers 1, 2 and 3 as atomic expressions:

G1:	$E \rightarrow E + E \mid T$ $T \rightarrow F * F \mid F$ $F \rightarrow 1 \mid 2 \mid 3$	G5:	$\begin{array}{rcl} E & \rightarrow & E+T & \mid T \\ T & \rightarrow & T*F & \mid F \\ F & \rightarrow & 1 & \mid 2 & \mid 3 \end{array}$
G2:	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	G6:	$\begin{array}{cccccccccccccccccccccccccccccccccccc$
G3:	$\begin{array}{rcl} E & \rightarrow & T+E & \mid & T \\ T & \rightarrow & F*T & \mid & F \\ F & \rightarrow & 1 & \mid & 2 & \mid & 3 \end{array}$	G7:	$\begin{array}{rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr$
G4:	$\begin{array}{rcl} E & \rightarrow & T+E & \mid & E \ast T & \mid & T \\ T & \rightarrow & 1 & \mid & 2 & \mid & 3 \end{array}$	G8:	$\begin{array}{rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr$

(i) [2 marks] Which grammar treats + and * as having the same precedence, and both being non-associative?

(ii) [2 marks] Which grammar treats + and * as having the same precedence, and both being left-associative?

(iii) [2 marks] Which grammar treats + and * as having different precedence, and both being right-associative?

(iv) [2 marks] Which grammar treats + and * as having different precedence and different-associativity?

Question 3. Parsing

Consider the following grammar:

$$\begin{array}{rcl} Stmt & \rightarrow & \text{if } Expr \text{ then } Stmt \text{ else } Stmt \text{ fi} \\ & | & \text{if } Expr \text{ then } Stmt \text{ fi} \\ & | & ID := ID \end{array}$$

$$Expr & \rightarrow & ID == ID$$

$$ID & \rightarrow & \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \dots$$

(a) [5 marks] For each of the following strings, discuss whether the above grammar produces it or not. Give *parse trees* as evidence where possible.

(i) if a == b then a := c ; a := b fi
(ii) if a == b then if b == c then a := c fi fi

(b) [5 marks] Compute the following sets:

- (i) *first*(*Expr*)
- (ii) follow(Stmt)

(c) [5 marks] By computing *first()* sets appropriately, show that this grammar is not LL(1).

(d) [5 marks] By *left-factoring*, convert this grammar to be LL(1).

Now, consider the following grammar:

 $Expr \rightarrow Expr + Term | Term$ $Term \rightarrow ID$ $ID \rightarrow a | b | c | \dots$

(e) [5 marks] The above grammar is *left-recursive*. Discuss why this means it is not LL(1).

(f) [5 marks] Show how the grammar can be rewritten to eliminate left-recursion. Indicate any differences in the parse trees produced, compared with the original grammar.

COMP 202 / SWEN 202

[30 marks]

Question 4. FlowChart Programs

(a) Indicate whether each of the following FlowChart programs is *syntactically correct* or not. If it is not correct, indicate why; otherwise, briefly describe what it does.

(i) [3 marks]

	input x;
	input y;
	if $x < y$ then 1 else 2;
1:	x := y;
	goto 3;
2:	skip;
(ii) [3 marks]	
	input x;
	if $x < 0$ then 1 else 2;
1:	x := -x;
2:	skip;
	output x;
(iii) [3 marks]	
	input x;
	input y;
	if $x < 0$ then 1 else 4;
1:	if $y < 0$ then 2 else 3;
2:	z := x + y;
3:	goto 5;
4:	z := 0;
5:	output z;

(b) Consider the following two FlowChart programs.

	x := 1;		x := 1;
	y := 0;		y := 0;
1:	if $x < 3$ then 2 else 3;	1:	if <i>x</i> < 3 then 2 else 3;
2:	x := x + 1;	2:	x := x + 1;
	y := y + 1;		y := x - 1;
	output y;		output y;
	goto 2;		goto 2;
3:	skip;	3:	skip;
	(Program A)		(Program B)

(i) [3 marks] Give an execution trace for program A.

(ii) [4 marks] Briefly discuss whether these programs are *weakly equivalent*, *strongly equivalent* or *not equivalent*.

(c) The FlowChart language is to be extended with support for *arrays*. This allows programs such as the following, where an element of the array *a* is overwritten with zero:

$$a := \langle 0, 1, 2, 3 \rangle;$$

input *x*;
 $a[x] := 0;$

(i) [2 marks] Give a condition which ensures any statement of the form "v[i] := e" will not cause the program to abort.

(d) The semantics for assignments of the form "v := e" is given by:

$$\frac{P(pc) = (l_1 : v := e) \quad n = \mathcal{V}(e, S)}{(P, pc, S) \longrightarrow (P, pc+1, S[v := n])} \quad [\text{ASSIGN}]$$

(i) [2 marks] On the following FlowChart Machine state, apply the above ASSIGN rule to produce the next state:

$$(\langle 1: x := y \rangle, 1, \{x \mapsto 2, y \mapsto 3\})$$

(ii) [4 marks] Give a similar rule which governs assignments of the form " $v[i] := e^{n}$. Note, if v is an array, then v[i := n] returns v with element i updated to hold the constant n.

(e) [6 marks] For each of the numbered statements in the following program, write an appropriate *assertion* which always holds true before that statement is executed. Indicate whether the postcondition will be satisfied after the return statement is executed.

PRE:
POST: ret
$$\ge 0$$

absum(*list*) begin
 $i := 1;$
 $r := 0;$
1: if $i \le |list|$ then 2 else 6;
2: if *list*[*i*] < 0 then 3 else 4;
3: $r := r - list[i];$
goto 5;
4: $r := r + list[i];$
5: $i := i + 1;$
goto 1;
6: return r;
end

Question 5. Alloy

The statement "One Man's Ceiling is Another Man's Floor", taken from a song by Paul Simon, inspired the following Alloy model:

```
sig Platform {}
sig Man {ceiling, floor: Platform}
fact above { all m: Man | some n: Man | m.ceiling = n.floor }
pred below { all m: Man | some n: Man | m.floor = n.ceiling }
pred woleb { some n: Man | all m: Man | m.floor = n.ceiling }
```

(a) Consider the following instance of the model:

$$\begin{aligned} \mathtt{Man} &= \{M0, M1\} \\ \mathtt{Platform} &= \{P0, P1\} \\ \mathtt{ceiling} &= \mathtt{floor} = M0 \text{->}P0 + M1 \text{->}P1 = \boxed{\begin{array}{c} M0 & P0 \\ \hline M1 & P1 \end{array}} \end{aligned}$$

(i) [2 marks] Draw a visualisation (graph representation as Alloy would do) of this model instance.

(ii) [1 mark] Compute M0.ceiling.

(iii) [1 mark] Compute M1.floor.

(iv) [2 marks] Is the predicate called below true for this instance? Give a brief explanation of why or why not.

(v) [2 marks] Is the predicate called woleb true for this instance? Give a brief explanation of why or why not.

(b) Consider the following instance of the model:

$$Man = \{M0, M1, M2\}$$

$$Platform = \{P0, P1, P2\}$$

$$ceiling = M0 ->P1 + M1 ->P2 + M2 ->P1 = \boxed{\begin{array}{c} M0 & P1 \\ M1 & P2 \\ M2 & P1 \end{array}}$$

$$floor = M0 ->P0 + M1 ->P1 + M2 ->P2 = \boxed{\begin{array}{c} M0 & P0 \\ M1 & P1 \\ M2 & P2 \end{array}}$$

7

[30 marks]

(i) [2 marks] Draw a visualisation (graph representation as Alloy would do) of this model instance.

(ii) [2 marks] Compute floor. ~ceiling.

(iii) [2 marks] Compute ^(floor.~ceiling).

(iv) [2 marks] Compute floor. ~ceiling.Man.

(v) [2 marks] In your own words, describe what floor. ~ceiling. Man represents.

(vi) [2 marks] Is the predicate called below true for this instance? Give a brief explanation of why or why not.

(c) Extending the Alloy specification

(i) [2 marks] Add a fact to the Alloy model given above that makes sure that ceiling and floor are distinct for each Man.

(ii) [3 marks] Write an Alloy function called fun_above (with no arguments) that returns a binary relation between a platform and any platform directly above it. A platform q is directly above a platform p, i.e. p->q in fun_above, if and only if there is a Man who has Platform p as floor and Platform q as ceiling.

(iii) [3 marks] Write a predicate called notAboveItself that is true if no Platform is (directly or indirectly) above itself. In other words, predicate notAboveItself should be true if and only if no platform is directly above itself or is directly above a Platform that is directly above itself, etc.

(iv) [2 marks] What would happen if you executed the command run notAboveItself using the Alloy analyser? Give a brief explanation of your answer.

Question 6. An Alloy Model for Finite Acceptors

[30 marks]

A *Nondeterministic Finite Acceptor* (NFA) is defined as a tuple (Q, q, A, N, F), where Q is a non-empty finite set of states, $q \in Q$ is the initial state, A is a finite set of symbols, $N \subseteq Q \times A \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final states.

Consider the following incomplete Alloy model for NFAs:

sig Symbol {}
sig State { transitions: Symbol -> State }

(a) [5 marks] Extend the Alloy model given above with a representation for start and final states according to the NFA definition given. Make sure that each NFA model instance has exactly one initial state.

(b) Provide definitions (predicates) for the following classes of NFAs:

(i) [2 marks] NFAs with only one state.

(ii) [2 marks] NFAs with no final states.

(iii) [2 marks] NFAs for which the initial state is not final.

(iv) [2 marks] NFAs without *self-loops*, i.e. transitions that start and end at the same state.

(v) [2 marks] *Deterministic Finite Acceptors,* that is NFAs that do not have multiple transitions that start from the same state and are labelled with the same symbol.

(vi) [2 marks] NFAs that have a transition to a final state from each non-final state.

(vii) [3 marks] NFAs for which all states are *reachable*, that is, can be reached from the initial state by following transitions only. More formally: Given a NFA (Q, q, A, N, F), a state $s \in Q$ is reachable if there is a sequence of states q_0, \ldots, q_n such that $q_0 = q$, $q_n = s$, and for each $0 \le i < n$ there is a symbol $a \in A$ such (q_i, a, q_{i+1}) $\in N$.

(c) Next you want to model relations and operations on NFAs but the current model is static and does not allow such dynamic behaviour.

(i) [5 marks] Rewrite the static model given above to provide a dynamic model that supports multiple NFAs. To support operations on NFAs, your new model should allow NFAs to have states and symbols in common, but with different start and final states as well as different transition relations.

(ii) [2 marks] Provide a predicate that checks whether two NFAs have the same set of states and the same set of symbols but different initial states.

(iii) [3 marks] Provide an Alloy operation that, given a NFA, a state q, and a symbol a, adds to the given NFA a transition labelled a from the initial state to state q. More formally: Given an NFA (Q, q, A, N, F), a state $s \in Q$ and a symbol $a \in A$, your operation should construct the NFA: (Q, q, A, $N \cup \{(q, a, s)\}, F$).
