PAPER CODE NO.	EXAMINER : Michael Fisher
COMP313	DEPARTMENT : Computer Science Tel. No. 4-6701

THE UNIVERSITY
of LIVERPOOL

MAY 2006 EXAMINATIONS

Bachelor of Arts : Year 3
Bachelor of Engineering : Year 3
Bachelor of Science : Year 3
Bachelor of Science : Year 4
No qualification aimed for : Year 1

Formal Methods

TIME ALLOWED : $2\frac{1}{2}$ hours

INSTRUCTIONS TO CANDIDATES

Answer **four** questions only.

If you attempt to answer more questions than the required number of questions (in any section), the marks awarded for the excess questions will be discarded (starting with your lowest mark).

THE UNIVERSITY
of LIVERPOOL

1. This question concerns the basic structures used within Z specifications.

   (a) In Z, what is the difference between a *total* function and a *surjection*? **[4]**

   (b) Given a bag

   $$B == [\![ford, toyota, fiat, ford, toyota, honda, ford]\!]$$

   then what is $\mathrm{dom}\, B$ and what is $\mathrm{ran}\, B$? How will $\mathrm{dom}\, B$ and $\mathrm{ran}\, B$ change if we add another '*fiat*' to B? **[8]**

   (c) If f and g are both functions of the same type, then under what circumstances (if any) will

   $$\mathrm{dom}(f \cup g) = (\mathrm{dom}\, f) \cup (\mathrm{dom}\, g) \ ?$$

   Give an appropriate example to illustrate your answer. **[9]**

   (d) Given $r : \mathbb{P}\{apple, banana, grape, peach\}$, write down all the values r can possibly have. **[4]**

2. We are developing a Z specification for a family tree and have developed the initial state space schema below (where *PEOPLE* is the set of all people):

   Ancestry

   $is\_parent\_of : PEOPLE \pfun \mathbb{P}\, PEOPLE$

   ...

   Here, $is\_parent\_of(fred) = \{emily, david\}$ if $(fred \mapsto \{emily, david\}) \in is\_parent\_of$

   (a) What invariant might we add to the above state space schema in order to ensure that no one is a parent of themselves? **[5]**

   (b) Write a Z specification for an *AddChild* operation, which adds a child for an already existing parent.

   The operation should take inputs $child? : PEOPLE$ and $parent? : PEOPLE$. **[8]**

   (c) How would you describe, using Z notation, the set of all people who are *grandparents*? **[7]**

   (d) Write a Z specification for a *GetChildren* operation, which returns the set of children for a given parent.

   The operation should take $parent? : PEOPLE$ as input and return $children! : \mathbb{P}\, PEOPLE$ as output. **[5]**

THE UNIVERSITY
of LIVERPOOL

3. This question concerns the fundamentals of Temporal Logic.

   (a) We wish to say that

   > "at some point in the future, it will always be the case that both x and y occur in the subsequent moment."

   How might we represent this in temporal logic? **[4]**

   (b) How does temporal logic extend classical logic? In your answer give an example of a statement that is more naturally represented in temporal, rather than classical, logic. **[8]**

   (c) Show, either by using formal semantics or by using a clear explanation, why, in a propositional, discrete, linear temporal logic, the formula $a \mathcal{U} \bigcirc b$ implies the formula $\bigcirc \Diamond b$. **[7]**

   (d) Give an overview of two different temporal logics, both in terms of their underlying model of time and in terms of the temporal operators used. **[6]**

4. This question concerns the relationship between temporal logic and programs.

   (a) Give a temporal formula capturing a semantics of the statement in a simple imperative programming language:

   ```
   (if (x>2) then x:=1 else x:=x+2); end
   ```

   [8]

   (b) What temporal formula would we write to describe the property that, at some point in the above program's execution, the variable `x` is guaranteed to have a value less than 5? Is this a liveness property, a safety property, or neither? **[5]**

   (c) Usually, temporal specifications of distributed systems comprise the specifications on the components, together with a *Comms* formula. For two distributed components, with temporal specifications $Spec_X$ and $Spec_Y$, what is the usual purpose of the *Comms* formula in

   $$Spec_A \wedge Spec_B \wedge Comms$$

   Give examples of three typical (and meaningful) types of formulae that we would use as *Comms* and explain what constraints these three represent. **[12]**

5. What is model checking, and why is it useful?

   Write an essay on this, bringing in as many elements as appropriate, including

   - the aim of model checking,
   - the theory behind model checking,
   - the uses of model checking,
   - the mechanisms for model checking,
   - the problems with model checking and
   - potential solutions to some of these problems.

[25]

6. Consider the following Promela code describing a three process system where:

   - process `A` sends information to process `B` via channel `a2b`,
   - process `B` sends information to process `C` via channel `b2c`, and
   - process `C` sends information to process `A` via channel `c2a`.

```
proctype A (chan in, out)
{
        int total;
        total = 0;                          /* initial state */
S1:     total = (total+3)%8;
        out!total;
        printf("A sent %d\n", total);
        in?total;
        assert(total != 4);                 /* assertion */
        printf("A received %d\n", total);
        if
        :: (total != 0) -> goto S1;
        :: (total == 0) -> out!total
        fi }

proctype B (chan in, out)
{
        int total;
S1:     in?total;
        printf("B received %d\n", total);
        if
        :: (total != 0) -> total = (total+3)%8; out!total;
                           printf("B sent %d\n", total);
                           goto S1;
        :: (total == 0) -> out!total
        fi }

proctype C (chan in, out)
{
        int total;
S1:     in?total;
        printf("C received %d\n", total);
        if
        :: (total != 0) -> total = (total+3)%8; out!total;
                           printf("C sent %d\n", total);
                           goto S1;
        :: (total == 0) -> out!total
        fi }

init {  chan a2b = [1] of { int };
        chan b2c = [1] of { int };
        chan c2a = [1] of { int };
        atomic { run A(c2a, a2b); run B(a2b, b2c); run C(b2c, c2a) }
     }
```

Note that `%` is the modulo arithmetic operator. So, for example `(17%8)=1` and `(15%8)=7`.

(a) If we execute this program what sequence of outputs can we expect? **[10]**

(b) Will the assertion in process `A` succeed? Explain your answer. **[8]**

(c) What will happen if we change the assertion in process `A` to be

```
assert(total != 1)
```

[7]

Glossary of Z notation

Names

a,b	identifiers
d,e	declarations (e.g., $a : A; b, \ldots : B \ldots$)
f,g	functions
m,n	numbers
p,q	predicates
s,t	sequences
x,y	expressions
A,B	sets
C,D	bags
Q,R	relations
S,T	schemas
X	schema text (e.g., d, $d \mid p$ or S)

Definitions

$a == x$	Abbreviated definition
$a ::= b \mid \ldots$	Data type definition (or $a ::= b\langle\langle x \rangle\rangle \mid \ldots$)
$[a]$	Introduction of a given set (or $[a, \ldots]$)
$a\_$	Prefix operator
$\_a$	Postfix operator
$\_a\_$	Infix operator

Logic

$true$	Logical true constant
$false$	Logical false constant
$\neg\, p$	Logical negation
$p \land q$	Logical conjunction
$p \lor q$	Logical disjunction
$p \Rightarrow q$	Logical implication ($\neg\, p \lor q$)
$p \Leftrightarrow q$	Logical equivalence ($p \Rightarrow q \land q \Rightarrow p$)
$\forall X \bullet q$	Universal quantification
$\exists X \bullet q$	Existential quantification
$\exists_1 X \bullet q$	Unique existential quantification
let $a == x; \ldots \bullet p$	Local definition

Sets and expressions

$x = y$	Equality of expressions
$x \neq y$	Inequality ($\neg\, (x = y)$)
$x \in A$	Set membership
$x \notin A$	Non-membership ($\neg\, (x \in A)$)
$\varnothing$	Empty set
$A \subseteq B$	Set inclusion
$A \subset B$	Strict set inclusion ($A \subseteq B \land A \neq B$)
$\{x, y, \ldots\}$	Set of elements
$\{X \bullet x\}$	Set comprehension
$\lambda X \bullet x$	Lambda-expression – function
$\mu X \bullet x$	Mu-expression – unique value
let $a == x; \ldots \bullet y$	Local definition
if p then x else y	Conditional expression
$(x, y, \ldots)$	Ordered tuple
$A \times B \times \ldots$	Cartesian product
$\mathbb{P}\, A$	Power set (set of subsets)
$\mathbb{P}_1\, A$	Non-empty power set
$\mathbb{F}\, A$	Set of finite subsets
$\mathbb{F}_1\, A$	Non-empty set of finite subsets
$A \cap B$	Set intersection
$A \cup B$	Set union
$A \setminus B$	Set difference
$\bigcup A$	Generalized union of a set of sets
$\bigcap A$	Generalized intersection of a set of sets
$first\ x$	First element of an ordered pair
$second\ x$	Second element of an ordered pair
$\#A$	Size of a finite set

Relations

$A \leftrightarrow B$	Relation ($\mathbb{P}(A \times B)$)
$a \mapsto b$	Maplet ((a, b))
dom R	Domain of a relation
ran R	Range of a relation
id A	Identity relation
$Q \fatsemi R$	Forward relational composition
$Q \circ R$	Backward relational composition ($R \fatsemi Q$)
$A \lhd R$	Domain restriction
$A \lhd\!\!\!\!- R$	Domain anti-restriction
$A \rhd R$	Range restriction
$A \rhd\!\!\!\!- R$	Range anti-restriction
$R(\!\mid A \mid\!)$	Relational image
$iter\ n\ R$	Relation composed n times
R^n	Same as $iter\ n\ R$
$R^\sim$	Inverse of relation (R^{-1})
R^*	Reflexive-transitive closure
R^+	Irreflexive-transitive closure
$Q \oplus R$	Relational overriding ($(\text{dom}\, R \lhd\!\!\!\!- Q) \cup R$)
$a\ \underline{R}\ b$	Infix relation

Functions

$A \pfun B$	Partial functions
$A \fun B$	Total functions
$A \pinj B$	Partial injections
$A \inj B$	Total injections
$A \psur B$	Partial surjections
$A \sur B$	Total surjections
$A \bij B$	Bijective functions
$A \mathrel{+\!\!\!\rightarrow} B$	Finite partial functions
$A \mathrel{\rightarrowtail\!\!\!\!+} B$	Finite partial injections
$f\ x$	Function application (or $f(x)$)

Numbers

$\mathbb{Z}$	Set of integers
$\mathbb{N}$	Set of natural numbers $\{0, 1, 2, ...\}$
$\mathbb{N}_1$	Set of non-zero natural numbers $(\mathbb{N} \setminus \{0\})$
$m + n$	Addition
$m - n$	Subtraction
$m * n$	Multiplication
$m \operatorname{div} n$	Division
$m \bmod n$	Modulo arithmetic
$m \leq n$	Less than or equal
$m < n$	Less than
$m \geq n$	Greater than or equal
$m > n$	Greater than
$succ\, n$	Successor function $\{0 \mapsto 1, 1 \mapsto 2, ...\}$
$m \,..\, n$	Number range
$min\, A$	Minimum of a set of numbers
$max\, A$	Maximum of a set of numbers

Sequences

$\operatorname{seq} A$	Set of finite sequences
$\operatorname{seq}_1 A$	Set of non-empty finite sequences
$\operatorname{iseq} A$	Set of finite injective sequences
$\langle\rangle$	Empty sequence
$\langle x, y, ...\rangle$	Sequence $\{1 \mapsto x, 2 \mapsto y, ...\}$
$s \frown t$	Sequence concatenation
$\frown / s$	Distributed sequence concatenation
$head\, s$	First element of sequence ($s(1)$)
$tail\, s$	All but the head element of a sequence
$last\, s$	Last element of sequence ($s(\#s)$)
$front\, s$	All but the last element of a sequence
$rev\, s$	Reverse a sequence
$squash\, f$	Compact a function to a sequence
$A \upharpoonright s$	Sequence extraction ($squash(A \lhd s)$)
$s \upharpoonright A$	Sequence filtering ($squash(s \rhd A)$)
$s \operatorname{prefix} t$	Sequence prefix relation ($s \frown v = t$)
$s \operatorname{suffix} t$	Sequence suffix relation ($u \frown s = t$)
$s \operatorname{in} t$	Sequence segment relation ($u \frown s \frown v = t$)
$\operatorname{disjoint} A$	Disjointness of an indexed family of sets
$A \operatorname{partition} B$	Partition an indexed family of sets

Bags

$\operatorname{bag} A$	Set of bags or multisets ($A \nrightarrow \mathbb{N}_1$)
$[\![]\!]$	Empty bag
$[\![x, y, ...]\!]$	Bag $\{x \mapsto 1, y \mapsto 1, ...\}$
$count\, C\, x$	Multiplicity of an element in a bag
$C \sharp x$	Same as $count\, C\, x$
$n \otimes C$	Bag scaling of multiplicity
$x \sqsubseteq C$	Bag membership
$C \sqsubseteq D$	Sub-bag relation
$C \uplus D$	Bag union
$C \setminus\!\!\setminus D$	Bag difference
$items\, s$	Bag of elements in a sequence

Schema notation

Vertical schema.

S
d
p

New lines denote ';' and '$\land$'. The schema name and predicate part are optional. The schema may subsequently be referenced by name in the document.

Axiomatic definition.

d
p

The definitions may be non-unique. The predicate part is optional. The definitions apply globally in the document.

$[a, ...]$ **Generic definition.**

d
p

The generic parameters are optional. The definitions must be unique. The definitions apply globally in the document.

$S \mathrel{\widehat=} [X]$	Horizontal schema
$[T; ... \mid ...]$	Schema inclusion
$z.a$	Component selection (given $z : S$)
θS	Tuple of components
$\neg\, S$	Schema negation
$\operatorname{pre} S$	Schema precondition
$S \land T$	Schema conjunction
$S \lor T$	Schema disjunction
$S \Rightarrow T$	Schema implication
$S \Leftrightarrow T$	Schema equivalence
$S \setminus (a, ...)$	Hiding of component(s)
$S \upharpoonright T$	Projection of components
$S \fatsemi T$	Schema composition (S then T)
$S \gg T$	Schema piping (S outputs to T inputs)
$S[a/b, ...]$	Schema component renaming (b becomes a, etc.)
$\forall X \bullet S$	Schema universal quantification
$\exists X \bullet S$	Schema existential quantification
$\exists_1 X \bullet S$	Schema unique existential quantification

Conventions

$a?$	Input to an operation
$a!$	Output from an operation
a	State component before an operation
a'	State component after an operation
S	State schema before an operation
S'	State schema after an operation
ΔS	Change of state (normally $S \land S'$)
ΞS	No change of state (normally $[S \land S' \mid \theta S = \theta S']$)

Jonathan P. Bowen
Oxford University Computing Laboratory
Wolfson Building, Parks Road, OXFORD OX1 3QD, UK
Email: Jonathan.Bowen@comlab.ox.ac.uk