CAVENDISH CAMPUS

School of Electronics and Computer Science

Modular Undergraduate Programme
Second Semester 2009 – 2010

Module Code: 3SFE618

Module Title: Formal Methods **[MARKING SCHEME]**

Date: Monday, 17th May 2010

Time: 10:00 – 12:00

Instructions to Candidates:

Answer ALL questions in Section A and TWO questions from Section B.
Section A is worth a total of 50 marks.
Each question in section B is worth 25 marks.

Section A

Answer ALL questions from this section

Question 1

Answers similar to the following are expected. Note: students may include a diagram of the 3 types of states.

(a) Three categories of system states: *valid* states, *initial* or *start* states & *error* or *invalid* states. [1 mark]

(b) The *state schema* models/represents the *"valid abstract states of the system"* being specified. [1 mark]

The state schema has two components: the information that makes up the state of the system, i.e. the variables and their types; & the constraints the information must satisfy, i.e., predicates. [1 mark]

(c) *State invariants* are the constraints/properties, in the state schema, that the states of the system are required to satisfy during its lifetime, i.e., all of the states it passes through during its execution should satisfy them. The valid states are those that satisfy the *state invariants*. [2 marks]

(d) The *initial state schema* defines the set of possible starting states of the system. Any initial state must also be a valid state, i.e., one that satisfies the state invariants in the *state schema*. [2 marks]

(e) *Pre-conditions* are predicates that determine the (valid) before states of the system in which the operation can successfully be completed, if they are not satisfied then the operation should not happen. [1 mark]

That is they characterize the *before* states that ensure that the new values assigned to the state variables by the operation will satisfy the after state's state invariants, i.e., ensures transitions from one valid state to another. [2 marks]

(f) A *total operation* is one which has an outcome defined for all possible states. In other words there is no legal state of the system for which the operation is not defined, i.e., *"all eventualities have been catered for"*. [2 marks]

[QUESTION Total 12]

Question 2

Schema Inclusion: allows the information defined in one schema to be included in another one. Thus for example large states can be decomposed into sub-states. **[2 marks]**

Schema Disjunction: allows schemas to be combined using $\vee$. E.g. used to combine the separate schemas for "success" & "errors" cases when defining "total" operations. **[2 marks]**

Δ Convention: allows the state in terms of primed & unprimed variables & invariants to be included when defining operations which change the state. **[3 marks]**

Ξ Convention: allows the state in terms of primed & unprimed variables & invariants, plus "no-change" constraints to be included when defining enquiry operations or "error" cases of operations which do not change the state. **[3 marks]**

[QUESTION Total 10]

Question 3

(a) The state schema:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CarsRus}} \\
\hline
stock : \mathbb{P}\, CAR \\
manufacturers : \mathbb{P}\, MANUFACTURER \\
registration : CAR ⤔ REGNO \\
make : CAR \nrightarrow MANUFACTURER \\
price : CAR \nrightarrow \mathbb{N} \\
\hline
\mathrm{ran}\ make \subseteq manufacturers \\
\hline
\end{array}
$$

[12 marks]

(b) Initial state schema, variables may or may not be primed:

$$
\begin{array}{|l}
\hline
\textit{InitialCarsRus} \\
\textit{CarsRus}' \\
\hline
\textit{stock}' = \{car1, car2\} \\
\textit{manufacturers}' = \{BMW, Toyota, Honda\} \\
\textit{registration}' = \{car1 \mapsto R52\_ABC, car2 \mapsto T58\_XYZ\} \\
\textit{make}' = \{car1 \mapsto BMW, car2 \mapsto Toyota\} \\
\textit{price}' = \{car1 \mapsto 7000, car2 \mapsto 5000\} \\
\hline
\end{array}
$$

[8 marks]

[QUESTION Total 20]

Question 4

Definition of *domain restriction* $S \lhd R$.
Generic parameters **[0.5 marks]**, signature [3 marks] ,
definition **[4.5 marks]**.

$$
\begin{array}{|l}
\hline
[X, Y] \\
\_ \lhd \_ : (\mathbb{P} X \times (X \leftrightarrow Y)) \rightarrow (X \leftrightarrow Y) \\
\hline
\forall R : X \leftrightarrow Y; \ S : \mathbb{P} X \bullet \\
\quad S \lhd R = \{ x : X; \ y : Y \mid x \mapsto y \in R \land x \in S \bullet x \mapsto y \} \\
\hline
\end{array}
$$

[8 marks]
[QUESTION Total 8]

Section B

Answer TWO questions from this section

Question 5

(a) **(i)** Invariants of $FilmRentalShop$:

1. $\forall c : CUSTOMER \bullet \#(rentedto \rhd \{\ c\ \}) \leq maxrentals$ — that no customer can borrow more than $maxrentals$ DVDs.
2. $inshop \cap \mathrm{dom}\ rentedto = \varnothing$ — a copy of a DVD can not be in the shop and rented out.
3. $\mathrm{dom}\ stock = inshop \cup \mathrm{dom}\ rentedto$ — the stock of copies of DVDs are those in the shop or those on rent.
4. $\mathrm{ran}\ rentedto \subseteq customers$ — only registered customers can borrow DVDs.

[1 mark each]

[SUBPART Total 4]

(ii) Pre-conditions of $RentFilm\_Success$:

1. $dvd? \in inshop$ — the copy is in the shop.
2. $customer? \in customers$ — the customer is a registered customer.
3. $\#(rentedto \rhd \{\ customer?\ \}) < maxrentals$ — the customer has not reached the $maxrentals$ rental limit.

[1 mark each]

[SUBPART Total 3]

[PART Total 7]

(b) The following is what is expected:

$$
\begin{array}{|l}
\hline
\textit{RentFilm\_Ok} \\
\Delta FilmRentalShop \\
\Xi FilmDataBase \\
dvd? : DVD \\
customer? : CUSTOMER \\
\hline
dvd? \in inshop \\
customer? \in customers \\
\#(rentedto \rhd \{ \ customer? \ \}) < maxrentals \\
rentedto' = rentedto \oplus \{ \ dvd? \mapsto customer? \ \} \\
inshop' = inshop \setminus \{ \ dvd? \ \} \\
\hline
\end{array}
$$

[6 marks]

$$ReportSuccess \mathrel{\widehat=} [\ report! : REPORT \mid report! = OK \ ]$$

[1 mark]

$$RentFilm\_Success \mathrel{\widehat=} RentFilm\_Ok \land ReportSuccess$$

[1 mark]

[PART Total 8]

(c) **(i)** Derive the error case pre-conditions by negating those of the successful operation. **[1 mark]** The actual pre-conditions (see schemas below). **[2 marks]**

[SUBPART Total 3]

(ii) The following is what is expected:

New types:

$$
\begin{array}{rl}
REPORT ::= & OK \\
& \mid Error\_NotAvailable \\
& \mid Error\_NotACustomer \\
& \mid Error\_AtRentingLimit
\end{array}
$$

[2 marks]

Error Case schemas:

$$
\begin{array}{|l}
\hline
\textit{RentFilm\_NotAvailable} \\
\Xi FilmRentalShop \\
dvd? : DVD \\
customer? : CUSTOMER \\
report! : REPORT \\
\hline
dvd? \notin inshop \\
report! = Error\_NotAvailable \\
\hline
\end{array}
$$

[1 mark]

$$
\begin{array}{|l}
\hline
\textit{RentFilm\_NotACustomer} \\
\Xi FilmRentalShop \\
dvd? : DVD \\
customer? : CUSTOMER \\
report! : REPORT \\
\hline
customer? \notin customers \\
report! = Error\_NotACustomer \\
\hline
\end{array}
$$

[1 mark]

$$
\begin{array}{|l}
\hline
\textit{RentFilm\_AtRentingLimit} \\
\Xi FilmRentalShop \\
dvd? : DVD \\
customer? : CUSTOMER \\
report! : REPORT \\
\hline
\#(rentedto \rhd \{\ customer?\ \}) = maxrentals \\
report! = Error\_AtRentingLimit \\
\hline
\end{array}
$$

[1 mark]

$$
\begin{aligned}
RentFilm \mathrel{\widehat{=}} & \; RentFilm\_Success \\
& \lor RentFilm\_NotAvailable \\
& \lor RentFilm\_NotACustomer \\
& \lor RentFilm\_AtRentingLimit
\end{aligned}
$$

[2 marks]

[SUBPART Total 7]

[PART Total 10]

[QUESTION Total 25]

Question 6

Something similar to the following is expected. The main differences are likely to be in the definitions of types $CELL$ & $WINDOW$; & how panel areas are defined. Marks will be awarded accordingly.

(a)

$$CELL == \mathbb{N} \times \mathbb{N}$$
$$WINDOW == \mathbb{N} \leftrightarrow \mathbb{N}$$

[2 marks]

$$\begin{array}{|l} WindowWidth, WindowHeight : \mathbb{N} \\ \hline WindowWidth = 80 \\ WindowHeight = 24 \end{array}$$

[2 marks]

$$\begin{array}{|l} Window : WINDOW \\ \hline Window = 1 \mathrel{..} WindowWidth \times 1 \mathrel{..} WindowHeight \end{array}$$

[3 marks]

[PART Total 7]

(b) Students should use the relational operators to define panel areas.

$$\begin{array}{|l} NavigationPanel : WINDOW \\ \hline NavigationPanel = (1 \mathrel{..} 10) \lhd Window \end{array}$$

[3 marks]

$$\begin{array}{|l} TitlePanel : WINDOW \\ \hline TitlePanel = ((11 \mathrel{..} 69) \lhd Window) \rhd (1 \mathrel{..} 4) \end{array}$$

[3 marks]

$$\begin{array}{|l} CartPanel : WINDOW \\ \hline CartPanel = ((70 \mathrel{..} 80) \lhd Window) \rhd (1 \mathrel{..} 4) \end{array}$$

[3 marks]

$$\begin{array}{|l} GoodsPanel : WINDOW \\ \hline GoodsPanel = ((11 \mathrel{..} 69) \lhd Window) \rhd (5 \mathrel{..} 24) \end{array}$$

[3 marks]

[PART Total 12]

(c) $MouseLocationTest$ operation:

$$PANEL ::= Navigation\_Panel \mid Title\_Panel \mid Cart\_Panel \mid Goods\_Panel$$

[1 mark]

$$\begin{array}{|l} \multicolumn{1}{l}{\text{\_\_ } MouseLocationTest \text{ \_\_\_\_\_\_\_\_}} \\ mouseLocation? : CELL \\ panel! : PANEL \\ \hline (mouseLocation? \in NavigationPanel \land panel! = Navigation\_Panel) \\ \quad \lor (mouseLocation? \in TitlePanel \land panel! = Title\_Panel) \\ \quad \lor (mouseLocation? \in CartPanel \land panel! = Cart\_Panel) \\ \quad \lor (mouseLocation? \in GoodsPanel \land panel! = Goods\_Panel) \\ \hline \end{array}$$

[5 marks]

[PART Total 6]

[QUESTION Total 25]

Question 7

(a) (Marks will be awarded accordingly if sensible but different answers are given.)

1. Both ZTC & ZANS provides syntax & type checking of Z specifications.
2. ZTC produces a type report of specification.
3. Both accept three input modes two of which are text based, as well as $\LaTeX\,2_\varepsilon$.
4. ZANS facilitates the validation of specifications via animation, by helping to find logical errors.

[PART Total 4]

(b) The following corrections are expected, but different correct answers will be marked accordingly.

Line 21: the type mismatch is caused by `subseteq` (i.e. $\subseteq$), which expects the lhs & rhs to be sets. Correction: it should be `in` (i.e. $\in$). **[2 marks]**

Line 32: `DownKey` is undefined because of a typo, see line 7 (`DownKEY`). Correction: change so both are the same, e.g., change version on line 7 to `DownKey`. **[1 mark]**

Line 37: this error is caused by a missing ";" at the end of the line, which ZTC expects. Correction: add a ";" to the end of the line. **[1 mark]**

Line 46: type mismatch is caused by the omission of the free type constructor. Correction: change to `report! = error(OnLastRow_CanNotMove_Down)`. **[1 mark]**

Line 53: syntax error caused by schema name `Down_ERRORS` which is defined as `Down_Errors` on line 51. Correction: change so `Down_Errors`. **[1 mark]**

Line 61: type mismatch is caused by the use of set brackets around ordered pair. Correction: replace { and } by (and) respectively. **[1 mark]**

Line 69: `position'` is undefined because only the state schema `Cursor` has been included on line 65. Correction: `Delta Cursor` ($\Delta Cursor$). **[2 marks]**

Line 77: `report!` is undefined because by it has been incorrectly declared on line 74, `report`. Correction: change declaration on line 74 to `report!`. **[1 mark]**

Reached .. parsing: caused by the omission of "`end specification`" at the end of the file. Correction: add it. **[1 mark]**

[PART Total 11]

(c) The following additions and modifications are expected, but different correct answers will be marked accordingly.

1. Replace the type definitions of *ERROR* & *REPORT* with:

```
REPORT ::=   Success | OnLastRow_CanNotMove_Down
             | AtHome_CanNotMove_Left
```

   since ZANS can not deal with general free types. **[2 marks]**
   In addition all of the error reporting would need to be changed as well, e.g., replace `report! = error(OnLastRow_CanNotMove_Down)` with `report! = OnLastRow_CanNotMove_Down`. **[1 mark]**

2. Provide an explicit definition for global defs., e.g., define values for example to the following values in the axiom schemas:

```
numbcols = 5
numbrows = 5
SCREEN = { 1, 2, 3, 4, 5 } & { 1, 2, 3, 4, 5 }
HomePosition = (1, 1)
```

   Otherwise ZANS could not execute the operations. **[3 marks]**

3. Insertion of the two ZANS pragmas:

```
%% state-schema  Cursor
%% init-schema   InitialCursor
```

   Indicates which are the state & initial-state schemas. **[2 marks]**

4. Re-define the initial-state schema:

```
--- InitialCursor --------------------
|  Cursor'
|-------------------------------
| position' = HomePosition
-------------------------------------
```

[**1 mark**]
ZANS starts the animation by executing the initial-state scheme & therefore expects an operation schema, ie. it must define the after state variables. [**1 mark**]

[**PART Total 10**]

[**QUESTION Total 25**]