**QUESTION 1 [12 Marks]**

1. Explain by giving the THREE (3) reasons how using formal methods allows to overcome errors at software development.

[4 Marks]

* Increase confidence in the correctness/robustness/security of the system
* Refinement and clarify customer’s requirements
* reveal ambiguity, inconsistency, incompleteness of specifications
* Provide basis for mathematical verifying equivalence between specification and implementation

1. You are given the following requirement:

*“Users should have easy to use interface to access the payment system”*

1. Discuss TWO (2) reasons why this requirement is not clear or complete.

[3 Marks]

* + - Not clear if users of the system include all the possible types of users of system or its limited set (e.g. registered, not registered).
    - Not clear the statement “easy to use”, which has own meaning for different stakeholders.
    - Not clear, what means “access” and to what functions of payment?
    - Not clear, what means interface, is it GUI?

1. Considering your answer in (a), rewrite the requirement in natural language by avoiding the ambiguity of meaning.

[3 Marks]

*Registered users of the payment system should have simple and intuitive GUI-based interface to assess the function of specification payment details*.

1. Find two unambiguous (but natural sounding) sentences equivalent to the sentence *The man saw the woman with a telescope*, the first where the man has the telescope, the second where the woman has the telescope.

[2 Marks]

*The man saw in a telescope the woman*

*The man saw the woman having a telescope*

**QUESTION 2 [23 Marks]**

1. For what kind of systems TLA is good? State THREE (3) types of systems.

[3 Marks]

* Distributed
* Concurrent
* Asynchronous
* Real-time

1. Explain the principles of work of model checking algorithm, implemented in TLC. How TLC calculates amount of distinct states?

[7 Marks]

TLC is a TLA Model Checker, i.e. the program to find errors in TLA specifications. TLC can check that the specification satisfies (implies) a large class of TLA formulas. To find errors in a specification TLC trying to verify that it satisfies the needed properties. TLC works by generating behaviours that satisfy the specification. To do this, it must be given in the form of the model of the specification. To define a model, we should assign values to the specification's constant parameters. Model checking algorithm tries to find all the reachable states, that is, all states that can occur in behaviours satisfying the formula . For example we have a system, which described by two variables x and y. In TLA notation:

VARIABLES x, y

We can fix all the possible states of the system by specification of the type invariant property

TypeInvariant == /\ x \in {1, 2, 3}

/\ y \in {4, 5, 6}

In this case the general amount of distinct states will be calculated as product of the power of sets {1,2,3} and {4,5,6}, i.e. |{1,2,3}| \* |{4,5,6}| = 3\*3 = 9

Initial predicate is used to specify initial values and also check for type invariance, e.g.

Init == /\ x = 1

/\ y = 4

/\ TypeInvariant

1. What does it mean behaviour of system in term of TLA? Give ONE (1) example of specification of behaviour of a system.

[3 Marks]

We represent the behavior of a system as a sequence of states, where a state is defined by assignment of values to variables. Let the variable hr represent the clock's display. We can present a behavior of the clock as the sequence



where e.g. [hr = 11] is a state in which the variable hr has the value 11.

In TLA, the next state of a system expressed with using primed variables, e.g. hr`= hr+1. This allows to set the relation between the values of variables in the previous (old) state and the next (new) state of a system.

1. For each following temporal formula give ONE (1) example of situation, there the formula can be applied.



[2 Marks]

Temporal operator □ (box) is used to express that all the steps are satisfied by the formula P. A formula □P, where P is a state predicate, is true of a behaviour iff P is true in every state of the behaviour.



This formula is true on behaviour, iff each step is a [N]v step.

[2 Marks]



[2 Marks]

TLA operator ◊ we read as eventually. ◊F is defined to equal to ¬□¬F. It asserts that F is not always false, which means that F is true at some time.



[2 Marks]

This formula implies that *F is infinitely often true*. F is true then or in some later time.

 This formula implies that *F is infinitely always true*, i.e. F becomes true and remains true thereafter.

[2 Marks]

**QUESTION 3 [40 Marks]**

Develop TLA specifications of the following situations. You can use TLATEX or ASCII notation. Please comment your statements.

**1. Task about a cat, a mouse and a cheese.**

When a rain comes, the cat goes in the room or in the basement. When the cat is in the room, the mouse goes in a hole, and the cheese is put in the fridge. If the cheese on the table, and the cat in the basement, then mouse in the room. Now comes the rain, and cheese on the table.

[20 Marks]

EXTENDS Naturals, TLC

\\* **Define Cat, Mouse, Cheese as Variables**

VARIABLES Cat, Mouse, Cheese

\\* **Define Room, Base, Table, Hole, Fridge as Constants**

CONSTANT Room, Base, Table, Hole, Fridge

\\* **Definition of type invariant and initial condition**

TypeInvariant == /\ Cheese \in {Table, Fridge}

/\ Cat \in {Room, Base}

/\ Mouse \in {Room, Hole}

Init == /\ Cheese = Table

/\ TypeInvariant

\\* **Definition of actions**

Next1 == /\ Cat = Room

/\ Mouse' = Hole

/\ Cheese' = Fridge

/\ UNCHANGED <<Cat>>

Next2 == /\ Cat = Base

/\ Mouse' = Room

/\ Cheese = Table

/\ UNCHANGED <<Cat, Cheese>>

\\* **Specification and theorem to prove**

Spec == Init /\ [][Next1 \/ Next2 ] \_ <<Cat, Mouse, Cheese>>

THEOREM Spec => []TypeInvariant

**2. Modelling mechanical movement**

Develop TLA specification of linear mechanical movement of a material point, bounded in the box with coordinates of borders x=0 and x=5. When a point touches a border it reverse the direction of movement. The movement is discrete – each step of material point is one unit of distance.

[20 Marks]

**EXTENDS Integers**

\\* Distance and velocity

**VARIABLES x, v**

\\* Point cannot leave the box

\\* So this is type invariant

TypeInvariant == /\ x <= 5

/\ x >= 0

Init == /\ x = 0

/\ v = 1

/\ TypeInvariant

Next == /\ x' = x + v

/\ IF ((x' = 0) \/ (x' = 5)) THEN v' = -v ELSE v' = v

Spec == Init /\ [][Next]\_<<x, v>>

THEOREM Spec => [] TypeInvariant

**QUESTION 4 [27 Marks]**

Develop model of a simple document control system in Z notation.

People who work together need to share their work, but there are many occasions for misunderstandings and confusion. Errors can be introduced when two people working on the same file of software code and make changes that conflict with each other. We need involve the computer to help prevent such errors: this is the purpose of a document control system.

Here are informal requirements:

* If a user wants to check out a document in order to change the document and the user has the permission to change it, and nobody else is changing the document at the moment, then that user may check the document out.
* As soon as a user has checked out a document for editing, everyone else is disallowed from checking out the document (but people with read permission still can read it).
* When the user is done editing the document, it should be checked in, allowing another user to check it out.

The Z model begins by introducing two basic sets that hold everything of interest in this universe, namely people and documents:

|  |
| --- |
| [PERSON, DOCUMENT] |

Some people have permission to change particular documents. We can model that as a relation on documents and people:

|  |  |  |
| --- | --- | --- |
|  |  | permission: DOCUMENT \rel PERSON |

This relation is just a set of pairs of the form (document, person). For example, Doug can change the specification, Aki and Doug can change the design, and Aki and Phil can change the code.

|  |  |  |
| --- | --- | --- |
|  |  | doug, aki, phil: PERSON |
|  | spec, design, code: DOCUMENT |
| permission = { (spec, doug), (design, doug), (design, aki), (code, aki), (code, phil) } | |

The state of the system is another relation of the same type, this one saying which documents are actually checked out to whom. The central requirement is that a document can only be checked out to one person at a time, so in this case the relation is a function: it associates each object in the domain with a single object in the range.

|  |  |  |
| --- | --- | --- |
|  | Documents |  |
|  |  |
|  |  |
| checked\_out: DOCUMENT \pfun PERSON | |
|  |  |
|  |  |
|  |  |
| checked\_out \subseteq permission | |
|  |  |

checked\_out is a partial function, indicated by the stroke through the arrow \pfun. This means that some documents might not be checked out to anybody. The predicate says that documents can only be checked out to people who have permission to change them.

A possible state of our system occurs when Doug has checked out the specification and design, and Phil has checked out the code.

|  |
| --- |
| checked\_out = { (design, doug), (spec, doug), (code, phil) } |

Lets define two operations that change the state, CheckOut and CheckIn. CheckOut has two input parameters, the person p? and the document d?

|  |  |  |
| --- | --- | --- |
|  | CheckOut |  |
|  |  |
|  |  |
| \Delta Documents | |
| p?: PERSON | |
| d?: DOCUMENT | |
|  |  |
|  |  |
|  |  |
| d? \notin **dom** checked\_out | |
| (d?, p?) \in permission | |
| checked\_out' = checked\_out \cup {(d?, p?)} | |
|  |  |

CheckOut has two preconditions. They are the predicates that contain no primed “after”' variables. First, document d? can't already be checked out: it can't be in the domain of checked\_out. Moreover, the person doing the checking out needs permission: (d?, p?) must belong to permission. If the preconditions are satisfied we add the pair (d?,p?) to checked\_out; this prevents anyone else from checking out d?.

We must account for cases where the preconditions are not satisfied. There are two preconditions, so there must be two such cases. CheckedOut says that the document is already checked out: $d? \in **dom**checked\_out. Unauthorized$ expresses that the person does not have permission: (d?,p?) \notin permission. In both cases nothing gets checked out and the state of the system does not change: \Xi Documents.

|  |  |  |
| --- | --- | --- |
|  | Unauthorized |  |
|  |  |
|  |  |
| \Xi Documents | |
| p?: PERSON | |
| d?: DOCUMENT | |
|  |  |
|  |  |
|  |  |
| (d?,p?) \notin permission | |
|  |  |

The total operation T\_CheckOut covers all three possibilities.

|  |
| --- |
| T\_CheckOut \defs CheckOut \lor CheckedOut \lor Unauthorized |