DTU CIVILINGENIØREKSAMEN **Page 1 of 6 pages**

Written examination on June 2, 1999 **Course number: 49238**

Course name: Formal Software Specification

Permitted resources: Written materials, only

problem 1	:	23%
problem 2	:	37%
problem 3	:	40%
TOTAL	:	100%

All specifications, definitions, etc., must be written in the specification language RSL.

Problem 1.

This problem concerns a reservation system for a theatre. For each planned performance at the theatre, it should be possible to reserve seats for people. Performances, seat numbers and person identifications are not further specified.

Question 1.1 (15%)
Define a type, "Database", which can be used to represent the given information. The definition must be model-oriented. If you find it necessary to introduce assumptions in addition to the ones mentioned above, then also state these in the solution.

The functions in the following questions must all be applicative (i.e. without use of variables), and when relevant, appropriate pre conditions must be defined.

Question 1.2 (4%)
Define a function, "free_seats", which given a performance and a database, returns the set of free seats at that performance.

Question 1.3 (4%)
Define a function, "mk_reservation", which reserves a given, free seat at a performance for a person.

Problem 2.

This problem concerns syntax and semantics of a language for writing systems of equations. A *system of equations* consists of 0, 1, or more equations in a given order. Each *equation* has a left-hand side and a right-hand side. The left-hand side is an identifier and the right-hand side is an expression. An *expression* is either (1) an integer constant, (2) an identifier or (3) an addition expression. A concrete syntax example:

```
x = 17
y = x + 5
z = 1
w = y + x + 2
```

Question 2.1 (5%)
Define an abstract syntax for a system of equations.

Question 2.2 (10%)
A system of equations is well-formed, if

1. the identifiers on the left-hand sides are distinct, and
2. identifiers appearing on the right-hand side of equation number i have been introduced on the left-hand side of one of the previous equations.

Define a static semantics for the language, i.e. define a function which checks whether a system of equations is well-formed.

An *interpretation* of a system of equations is a data structure, which associates an integer to each identifier appearing in the system of equations. A *solution* to a system of equations is an interpretation that satisfies each of the equations.

Question 2.3 (2%)
Define a type which can represent such interpretations.

(The problem is continued on next page.)

Question 2.4 (10%)

A well-formed system of equations has exactly one solution which is said to be its *semantics*. Define a function, "sem1", which gives the semantics of a well-formed system of equations.

Question 2.5 (10%)

Now the language is changed such that a system of equations is not required to be well-formed. In this case a system of equations may have 0, 1 or more solutions, and its *semantics* is the set of all its solutions. Define a function, "sem2", which gives the new semantics of a system of equations.

Problem 3.

This problem concerns a specification of a lift system. The lift is travelling between floors number 1 and *max*, where *max* is some constant greater than 1. Hence, at any time the lift cage is either situated at one of the floors or between two consecutive floors. At each floor there is a door which can be opened and closed. (In this problem, we do *not* consider buttons to call the lift.)

Question 3.1 (2%)
Specify "max" and define a type "Floor" of floors.

Question 3.2 (2%)
Define a type, "DoorPosition", which can represent the possible positions of a door.

Question 3.3 (6%)
Define a type, "LiftPosition", which can represent the possible positions of the lift cage.

The *state* of the lift system is characterised by the position of the lift cage and the positions of the doors, and is changed, when the lift is moving or one of the doors is opened or closed. (Note that the concept of "state" in this connection has nothing to do with RSL variables.)

The following questions concern an algebraic, applicative specification of a datatype "State" of lift system states, and a number of associated functions operating on states, including the following basic (non-derived) observer and generator functions:

- An observer function "door_position", which, given a floor and a state, gives the position of the door at that floor.
- An observer function "lift_position", which, given a state, gives the position of the lift cage.
- A generator function "open_door", which, given a floor and a state, generates a new state corresponding to the event that the door has been opened.
- A generator function "move_up", which, given a state, generates a new state corresponding to the event that the lift cage has been moved one position upwards, i.e. either from a floor to a position between that floor and the floor above that, or from a position between two consecutive floors to the uppermost of these two floors. However, if the lift cage is situated at the topmost floor (floor number *max*), it must keep its position.

(The problem is continued on next page.)

If you find it necessary to make additional assumptions, then state them in the solution.

Question 3.4 (1%)
Declare a sort "State" of states.

Question 3.5 (1%)
Declare the observer functions, "door_position" and "lift_position".

Question 3.6 (1%)
Declare the generator functions, "open_door" and "move_up".

Question 3.7 (19%)
Formulate observer-generator axioms.

Question 3.8 (4%)
A state is said to be *safe*, if at each floor the door at that floor is open only when the lift is situated at that floor. Use the given observer functions to define a function, "safe", which for a given state gives **true**, if and only if the state is safe.

Question 3.9 (4%)
Define a generator function "safe_open_door", which has same functionality as "open_door", except in the case where "open_door" would give an unsafe state. In such cases "safe_open_door" should keep the state unchanged.