DTU CIVILINGENIØREKSAMEN **Page 1 of 6 pages**

Written examination on June 10, 1998 **Course number: 49238**

Course name: Formal Software Specification

Permitted resources: Any

problem 1	:	28%
problem 2	:	32%
problem 3	:	40%
TOTAL	:	100%

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

Problem 1.

This problem concerns a hospital information system. At a hospital there are employed doctors and nurses, and there are patients. We assume that employees cannot be patients. The hospital is organised in uniquely named wards. Each doctor, nurse and patient is attached to one and only one ward and is identified by a unique person number. Each patient has a medical record. The format of medical records, person numbers and ward names is not given.

Assume given a type "Record" for medical records, a type "PersonNr" for person numbers and type "WardId" for ward names.

Question 1.1 (15%)
Define a type, "Hospital", which can be used to represent the given information. The definition must be model-oriented.
Hint: Define "Hospital" as a subtype using a well-formedness predicate, which you also define. 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 (3%)
Define a function, "is_in", which investigates whether a given person is a patient of a given hospital.

Question 1.3 (5%)
Define a function, "discharge", which discharges a given patient from a given hospital.

Question 1.4 (5%)
Define a function, "doctors", which returns the set of doctors in a given ward at a given hospital.

Problem 2.

This problem concerns syntax and semantics of a very simple specification language. A *specification* consists of 0, 1, or more type declarations, followed by 0, 1, or more constant declarations. A *type declaration* introduces the name of a type. A *constant declaration* introduces the name of a constant and states the name of the type that the constant should have.

A concrete syntax example:

types
t1,
t2,
t3
constants
c1 : t3,
c2 : t1

Question 2.1 (5%)
Define an abstract syntax for specifications.

Question 2.2 (11%)
A specification is well-formed, if

1. it does not contain two declarations of the same name, and
2. each type name used in the constant declarations is introduced in one of the type declarations.

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

In the following, assume that "Val" is a type of values which is not further specified.

A *valid interpretation* of a well-formed specification is a datastructure, which to each declared type name associates a set of values, and to each declared constant name associates a value from the set of values associated with the (name of) the type of the constant. The *semantics* of a well-formed specification is the set of valid interpretations of the specification.

(The problem is continued on next page.)

Question 2.3 (5%)
Define a type which can represent interpretations of specifications.

Question 2.4 (11%)
Define a function which gives the semantics of a well-formed specification.

Problem 3.

This problem concerns a specification of a simple railway system for a single-track railway line.

The track is divided into sections, numbered in succession from 0 to *max*, where *max* is a not further specified natural number.
Uniquely named trains run on the track. At the two ends of the track there are robust barriers, so that the trains cannot move further. It is assumed that the length of the trains is less than the length of the sections. Hence, a train is either situated on a single section with a number *s* (in which case the train is said to have a *single position* consisting of *s*) or on two sections with numbers *s1* and *s2*, where s2 = s1 + 1, (in which case the train is said to have a *double position* consisting of *s1* and *s2*). A train can have one of two possible *running directions*: *increasing* or *decreasing* direction of the section numbers.

Assume given a type "TrainId" for train names.

Question 3.1 (2%)
Declare "max" and define a type "Sectionnr" of section numbers.

Question 3.2 (6%)
Define a type, "Position", which can represent the possible positions of a train.
Hint: Define "Position" as a subtype of a variant type, which you also define.

Question 3.3 (2%)
Define a type, "Direction", which can represent the possible running directions of a train.

(The problem is continued on next page.)

The *state* of the railway is characterized by the position and running direction of the trains, and is changed, when the trains move or change running direction. A state is said to be *safe*, if no two distinct trains are fully or partly on the same section. (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 railway states, and a number of associated functions operating on states, including the following basic (non-derived) observer and generator functions:

- An observer function "direction" which, given a train and a state, gives the running direction of the train.
- An observer function "position" which, given a train and a state, gives the position of the train.
- A generator function "reverse", which, given a train and a state, generates a new state corresponding to the event that the train change running direction.
- A generator function "move", which, given a train and a state, generates a new (possibly unsafe) state corresponding to the event that the train moves to the next position in its running direction (even if that might imply a train collision), i.e. from a double position to next single position, or from a single position to next double position. However, if the train is situated on an end section with direction towards the end then the train must keep its position. It should not be specified, how "move" behaves for an unsafe state.

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, "direction" and "position".

Question 3.6 (4%)
Use the given observer functions to define a function, "safe", which for a given state gives **true**, if the state is safe.

(The problem is continued on next page.)

Question 3.7 (1%)
Declare the generator functions, "reverse" and "move".

Question 3.8 (19%)
Formulate observer-generator axioms as described in "Lecture Notes on the RAISE Development Method".

Question 3.9 (4%)
Define a generator function "safe_move", which has same functionality as "move", except in the case where "move" would give an unsafe state. In such cases "safe_move" must imply that the train keeps its position.