

VDM-SL: Problems

John Fitzgerald (adapted by Olga Pacheco)

December 2008

Problem 1 An airline is introducing a new computing system for allocating seats to passengers when the passengers arrive at the airport. Each seat on a flight has a seat number, a position (either by a window, by the aisle, or in the middle) and a passenger name (a character string). For the purpose of this system, a flight is a collection of seats with no two distinct seats on the flight having the same seat number.

1. Define the types “Position”, “Seat” and “Flight”.
2. Give an explicit definition of a function “freeseats” which, given a flight, returns the set of unallocated seats.
3. The passenger will be asked which seat position (aisle, window or middle) they prefer. They will be allocated a seat which matches their preference, provided one is available. Give an implicit specification of a function “goodseat” which, given a flight and a preferred position, returns an unallocated seat in the preferred position. Your specification should deal with the possibility that there may be no unallocated seats in the preferred position.

Problem 2 An hospital uses a computer-based system to send ambulances to incident scenes. The system maintains lists of requests for ambulances. A request consists of a description of the incident (a string) and a field indicating the level of urgency of the request: a request may be non-urgent, urgent or emergency.

1. Give VDM-SL definitions of data types “Request” and “List” to represent requests and lists of requests respectively.
2. Define a function which adds a request to the tail end of a list, returning the updated list.
3. Give an explicit definition of a function “firstemergency” which, given a list, returns the first emergency request in the list and returns nil if no such request is present in the list.
4. Give an implicit specification of “firstemergency”.

Problem 3 We are concerned with the formal specification of part of the software controlling a chemical processing plant. The plant is divided into streams. Each stream is a sequence of reactors. Each reactor has a set of batches of chemicals currently in the reactor and a maximum number of batches allowed in the reactor at any time. Each batch has an identifier and contents type. The representations of identifiers and contents types are immaterial.

1. Define the types “Stream”, “Reactor”, “BatchId”, “Batch”.
2. Define a function “batchesinstream” which returns the set of batches in a given stream.
3. Define an invariant on “Stream” expressing the following requirements formally:
 - (a) No two batches in the stream have the same identifier.
 - (b) No batches occurs in more than one reactor.
 - (c) No reactor has more batches than its maximum.

Problem 4 A computing system is to be developed to control the placement of containers of hazardous waste in a storage building. Each container has a unique identifier which is recorded along with an indication of the class of waste (high-level or low-level) stored in the container. A formal specification of the system is under development. The enumerated type `WasteClass` models the two classes of waste, while the composite type `Container` models containers:

types

```
WasteClass = <HIGH> | <LOW>;

Container :: cid : ContId
           class : WasteClass;

ContId = seq of char;
```

The store consists of a three-dimensional array of storage locations. Each location is modelled by an x-coordinate, a y-coordinate and a z-coordinate, giving a location’s position with respect to an origin (0,0,0). The array is 10 locations long in the x and y directions and 5 locations long in the z direction.

The composite type `Location` models locations. The type `Store` is defined to model the overall store. It maps locations to the containers stored at each location:

```

Location :: x: nat
           y: nat
           z: nat
inv 1 == x>=0 and x<10 and y>=0 and y<10 and z>=0 and z<5;

Store = map Location to [Container]
inv s == ...

```

1. Define an invariant for **Store** which records the restrictions that no two containers in the store have the same identifier and that no two locations hold the same container.

2. Define the function

```
adjacent: Location * Location * Store -> bool,
```

which returns **true** if and only if its first two arguments are adjacent locations in the third argument (a store).

3. A store is said to be safe if the following conditions are satisfied:

- No containers containing high-level waste are located around the outside of the store (e.g $x=0$ or $x=9$, or $z=4, \dots$).
- No two containers of high-level waste are in adjacent locations.

Define a Boolean function **safe_store** which returns **true** if and only if the store is safe. Add a conjunct to the invariant on **Store** which records this constraint.

4. Give an explicit definition of a function **add_to_store** which, given a container, a store and a location, returns the updated store with the container placed at that location. Include a pre-condition to ensure that the resulting store will not break the invariant on **Store**.

5. Define a function:

```
adjacent_containers: ContId * Store -> set of Container
```

which given a container identifier returns the set of containers adjacent to that container.

Problem 5 We want to develop a monitoring system for tracking air traffic. The system maintains a record of the structure of the air space in a certain area, the flight plans of aircraft passing through the area and the progress of the aircraft through their planned flights. The system has to be developed to a high level of integrity and a formal model of the system is to be constructed in order to permit analysis of the safety requirements at an early stage in development through the validation process. The airspace is divided into regions. Regions are linked together, so that aircraft pass from one region to another. Aircraft travel through the area in one direction (from left to right) and aircraft in a given region may only travel into a region to the right of the current one. Regions are labelled by numbers. Each link is modelled as a pair of region identifiers. Aircraft using a link pass from the “from”

region to the “to” region. An area is modelled as a set of links (e.g. {mk_Link(1,3), mk_Link(1,4), mk_Link(3,5), mk_Link(5,7)}).

```
RId = nat1
```

```
Link :: fromlink: RId
      tolink: RId;
```

A flight plan is a sequence of regions through which an aircraft will fly. Flight plans are modelled as sequences of region identifiers (e.g. [1,3,5,7]). The position of an aircraft, i.e. how far it has progressed through its plan is given as a natural number, i.e. the index of its current region within its flight plan. The plan and position of an aircraft is combined into a flight record:

```
FlightRecord :: plan: seq of RId
              position: nat1
inv mk_FlightRecord(pl,pos) == pos in set inds pl;
```

Flights are identified by flight identifiers:

```
FId = token;
```

The system is modelled as a composite object containing the area and the flight records:

```
ATCSystem :: area: set of Link
          flights: map FId to FlightRecord;
```

1. A plan is said to be feasible in a given system if it only involves the use of links in the space component of the system. Define a boolean function which, given a set of links and a plan, returns true if the plan is viable in the set of links and false otherwise.
2. Define a function which adds a new aircraft to an ATCSystem, given the flight identifier and its flight plan. The function should place the new aircraft in the first region of its flight plan. Remember to ensure that the aircraft identifier is unique and the plan is viable.
3. Define a function `move: ATCSystem * FId -> ATCSystem` which moves a flight forward one step in its plan. If the flight reaches the end of the plan, it should be removed from the system.
4. Define a function which, given the identifier of a starting region and the identifier of the finishing region, returns a viable route from the starting region to the ending region. Your function should return a special error value if no viable route exists.

5. Define a function which, given an ATCSystem and a region identifier, returns the set of flight identifiers for aircraft currently in a given region.
6. Aircraft traffic control regulations require that no more than ten aircraft should be in any region at any time. Add an invariant which ensures that this condition is respected in the system. Modify the functions `move` and `add`, in order to ensure the invariant. When a move or the addition of a new aircraft is not possible, the system is not updated.
7. Suppose the air traffic control regulations are modified so that a new safety regulation is introduced. For each region, the total number of aircraft within that region and all of the regions ahead of it must be less than 40. Define a function `ahead` which, given an ATCSystem and the identifier of a region, returns the set of identifiers of regions ahead of the given region. Define a function `safe` which, given an ATCSystem, returns true if and only if it satisfies the new safety constraint.