Carlos G. Rivera Velazquez Lab7
ICOM4009-060

StudentID: 802-12-6420

Homework #7

Exercise 10.1 Scheme Constructions. You are to select and solve one or more exercises, preferably one from each of the 6 groups of exercises listed below and to convert their formalisation into a set of two or more scheme definitions:

1. **Exercises 2.1** and **5.7**: Documents
2. **Exercise 2.2:** Part Assemblies
3. Networks. Common to the next three exercises is that of a previous exercise, Exercise 4.1. Select one of either of:
   (a) **Exercises 2.3 and 5.3:** City Road Nets — Streets and Intersections
   (b) **Exercises 2.4 and 5.4:** Air Traffic Route Nets: Air Lanes and Airports
   (c) **Exercises 2.5 and 5.5:** Shipping Nets: Lanes and Harbours
4. **Exercise 2.6:** Robots
5. Languages. Select one of either of:
   (a) **Exercise 3.3:** Denotational Semantics: A Simple Bank
   (b) **Exercise 3.4:** Denotational Semantics: A Simple Banking Script Language
   (c) **Exercise 3.5:** Denotational (Continuation) Semantics: Language with Storable Label Values
   (d) **Exercise 3.6:** Denotational (Continuation) Semantics: A Proper Coroutine Programming Language
6. Systems. Select one of either of:
   (a) **Exercises 4.2 and 5.1:** Supermarkets
   (b) **Exercises 4.3 and 5.2:** Manufacturing

Exercises 3,4,5,6 were chosen for these assignment.

(3) Exercise 2.4: Define the abstract type of air traffic route nets, airports, and air lanes. Define observer functions that observe airports and air lanes from the net, airport capacity and air lane length. Axiomatise suitable air traffic route nets.

Solution:

Informal:

1. Each lane connects exactly two airports.
2. There may be several air lanes between any two airports
3. Air lanes are either two-way or one-way.
4. Air lanes have a length
5. Airports accommodate one or more aircrafts
6. An airport is characterised by the capacity of aircraft that may be parked in the airport tarmac.

Formal:

<u>type</u>

Net, Lane = (two-way | one-way), Airport, Aircraft

<u>value</u>

obs_Ls: (Net | Airport) → Lane-set

obs_As: (Net | Lane) → Airport-set

obs_Acs: Airport → Aircraft-set

connect: Net X Lane X Airport X Airport → Bool

capacity: Airport → Integer

length: Lane → Real

<u>axiom</u>

∀ l : Lane · **card** obs_As(l) == 2

∀ a, a': Airport · (**card** obs_Ls(a) ^ **card** obs_Ls(a') ≥ 1) ^ a' ≠ a

∀ a: Airport · **card** obs_Acs(a) ≤ capacity(a)

(3) Exercise 5.4: Now consider air traffic as consisting of moving aircraft. That is, aircrafts on the ground are not moving! An aircraft can only start from an airport.
It then enters air traffic by entering an air lane connected to that airport.
Normally an aircraft then moves continuously along a route (i.e., within a sequence of air lanes), and normally an aircraft leaves air traffic when, or by, landing in an airport, i.e., by leaving an air lane connected to that airport.
Abnormally an aircraft, or two, may leave air traffic by exploding in the air, for example, by collision.

1. Provide a definition of what an aircraft position, in the air, is.

2. Provide a type definition of the concept of air traffic. Assume a time interval, from time ts tart to time tenc [, over which air traffic is defined.

3. Impose suitable constraints on air traffic.

4. Define a function which applies to any air traffic and which yields the first time, after tstart> at which an aircraft collision occurs, i.e., when their locations "overlap".

5. Define another function which applies to any air traffic and which yields the first time, after ts tart> a * which a single aircraft explosion occurs, i.e., when the aircraft "suddenly" disappears from air traffic.

Solution:

Informal:

1. An airport can only start from an airport.
2. It then enters air traffic by entering an air lane connected to that airport.
3. An aircraft moves along a route.
4. An aircraft leaves air traffic by landing on an airport.
5. An aircraft may also leave air traffic by colliding with another plane.
6. If two aircrafts are in the same lane, then they should not have the same position.
7. Takeoff time should be less than landing time for one aircraft from one aircraft to another.

Formal:

<u>type</u>

Time, Airport, Lane, Aircraft, Net, TimeInterval, X, Y, Z,

Position = (X X Y X Z)

<u>value</u>

obs_pos: Net → Position-set

elapse_time: (Time X Time) → TimeInterval

collision_time: (Aircraft X Aircraft X Position X Position) → TimeInterval

start_flight: (Airpot X Aircraft) → Lane

landing: (Aircraft X Lane) → Airport

leave_air_traffic: (Aircraft X Lane) → Lane

position: Aircraft → Position

landing_time: (Aircraft X Airport) → Time

takeoff_time: (Aircraft X Airport) → Time axiom

$\forall$a, a':Aircraft, l: Lane · **card** obs_Acs(l) ≥ 2 ⇒ position(a) ≠ position(a')

$\forall$ac:Aircraft, a,a':Aircraft, st,et:Time · landing_time(a,ac) > takeoff_time(a',ac)

(4) Exercise 2.6: Describe, informally and formally, the way in which robots are put together, i.e., the geometry of the robot.

Solution:

Informal:

1. A robot has a **base**, from which the **links** emanate.
2. A **link** is a rigid body, a single whole.
3. A **joint** is the connection between two or more links. It permits the orientation and position of the two links it connects to change.
4. A joint may be a **rotating joint** or a **linear joint**. A rotating joint defines an axis around which the two connected links may revolve. A linear joint allows one link to slide with respect to the other.
5. A **gripper** is basically the robot hand and is the last link. No further links emanate from the gripper.
6. An **arm** is a chain of links, from the base to a gripper.
7. Links have length, which is the perpendicular distance between its two joints.
8. Joint angles describe the rotation of a link with respect to another link.
9. Rotating joints have joint angles and linear joint have a link offset.
10. A joint can be either rotating or linear but not both.
11. A robot only has one base, at least one link and at least one gripper.
12. A joint must connect at least two links.
13. A gripper has one link.
14. An arm must have at least one link.
15. The base must have at least one arm, otherwise there wouldn't be a robot.

Formal:

type

Robot, Base, Link, Joint = (Rotating | Linear), Gripper, Arm

value

obs_As: (Robot | Base) → Arm-set

obs_Bs: (Robot | Link) → Base-set

obs_Gs: (Robot | Link) → Gripper-set

obs_Ls: (Arm | Joint | Gripper) → Link-set

obs_Js: (Arm | Link) → Joint-set

is_linear: Joint → Bool

is_rotating: Joint → Bool

connect: (Link X Link X Joint) → Bool

<u>axiom</u>

∀ j:Joint · is_linear(j) ^ is_rotating(j) == False

∀ r:Robot · **card** obs_As(r) ≥ 1 ^ **card** obs_Bs(r) == 1 ^ **card** obs_Gs(r) ≥ 1

∀ j:Joint · **card** obs_Ls(j) == 2

∀ g:Gripper · **card** obs_Ls(g) == 1

∀ a:Arm · **card** obs_Ls(a) ≥ 1

∀ b:Base · **card** obs_As(b) ≥ 1

(5) Exercise 3.3: Formalise the abstract types (i.e., sorts) of client names and account identifiers. Then formalise the concrete types of banks (contexts and states). Then define, still in RSL, the semantic functions which assign denotational meaning to transactions and to transaction sequences (the latter likely to relate to different clients).

<u>Solution:</u>

- Two or more clients may share accounts
- Clients may hold more than one account
- All accounts have clients
- The bank accepts the following transactions: create account, close account, deposit into account, withdraw from account, transfer between accounts, deliver a statement.
- The bank charges interest to negative accounts.

<u>type</u>

Client, Account = (Negative | Positive), Interest, Transaction, Bank

<u>value</u>

obs_Ac: (Bank | Client) → Account-set
obs_C: (Bank | Account) → Client-set
open: (Bank X Client) → Account
close: (Client X Account) → Bool
statement: (Client X Account) → Transaction
transfer: (Client X Client X Account X Account) → Bool
deposit: (Client X Account) → Bool
share: (Client X Client X Account) → Bool
interest: Account → Bool

<u>axiom</u>
∀a:Account · interest(a) == false ⇒ a ==positive
∀c:Client · **card** obs_Ac(c) ≥ 1
∀a:Account · **card** obs_C(a) ≥ 1

(6) Exercise 4.2: Supermarkets (I). You are asked to narrate and formalise a concept, such as you see it, of a supermarket, with shelves, price-tagged merchandise on shelves, a backup store from where near-empty or empty shelves can be replenished, consumers being in the supermarket, selecting merchandise from shelves and checking these out at a check counter. Assume each shelf to be typed with the merchandise it displays or is supposed to display.
What of the above, i.e., which entities of your model, constitute a (daily) context, and which constitutes the current state?

<u>Solution:</u>
Informal:
<u>Context:</u> The supermarket measures the quantity of the merchandise on shelves Qty, its capacity and the backup store's capacity and merchandise quantity. The supermarket also measures the merchandise being checked out at the check out.

<u>State:</u> The current quantity qty of the merchandise on the shelves and the current quantity of merchandise of the back up store.

We have an input channel to replenish the shelf and an output channel on the check counter. The shelves contains sensors to measure the shelf quantity, while the backup store contains both sensors (for measuring quantity) and actuators (to replenish shelf).

**Note: Full means that the shelf does not need to be replenished, it does not necessarily mean that is completely full. Empty does not mean that the shelf is fully empty, it means that the shelf needs to be replenished.

Formal:

<u>type</u>

Merchandise, Capacity, Qty, Check_Counter, Shelf = (full | empty), Backup_Store

<u>channel</u>

sc, bsc:Capacity, min,mout:Shelf

sq,bq: Qty, bout:Backup_Store

<u>value</u>

s:Supply, w:Withdraw, c:Checkout

system: Shelf X Backup_Store X Check_Counter → Unit

i_shelf: Shelf → read min out mout Unit

o_bstore: Backup_Store → read bq out bout Unit

(6) Exercise 5.1: Supermarkets (II). Reference is made to Exercise 4.2. Please read that exercise carefully. We assume here that you have also provided a solution to the questions asked. Consider "the day of a supermarket" to be a suitably discretised function from supermarkets to supermarkets. Assume that the cash registers start their day empty (no cash is changed). And assume that no deliveries are made during open hours, i.e., the day, to the backup store. Now write a well-formedness function over the "the day of a supermarket".

<u>Solution:</u>

<u>type</u>

T, Day, Hour = (open_hours | closed_hours), Minute, Second, Cash_Register, Backup_Store, Supermarket, at, mt

TT = aT X (Supermarket → (Hour-set X (Cash_Register → (mT) X (Backup_Store → (mT))) X aT

<u>value</u>
obs_Day: T → Day
obs_Hour: T → Hour
obs_Minute: T → Minute
obs_Second: T → Second
delivery: Hour → Boolean

<u>axiom</u>
∀h:Hour · (h == open_hours) ⇒ (delivery == false)

- Deliveries can't be made during open hours, to the backup store.
- This formalisation represents the day of a supermarket
- ***aT*** represents absolute time (beginning of the day and ending of the day), time goes from one absolute time to another.