Contents

1 Routes

1.1 Narrative

This system describes a airport connection system. The system state simply models the airports that are connected. So, dub is connected to cork, cork is connected to paris,etc. The operation we specify is, given a starting point (from?) and ending point (to?) to specify a possible route between these two AIRPORTs . Note the nature of the specification of the operation. The specification of the proposed route (route!) just states what rules this route! obeys, i.e. that each adjacent member of the sequence is connected according to the connected relation.

1.2 Basic Types and System State

$$[AIRPORT]$$

$$\begin{array}{|l}
\textit{AirSystem} \\
\hline
connected : AIRPORT \leftrightarrow AIRPORT \\
\hline
\end{array}$$

$$connected = \{dub \mapsto cork, cork \mapsto paris, cork \mapsto dub, paris \mapsto london, london \mapsto rome\}$$
$$route! = \langle dub, cork, paris, london, rome \rangle$$

1.3 Choose Route Operation

$$\begin{array}{|l}
\textit{ChoseRoute} \\
\hline
\Xi AirSystem \\
to?, from? : AIRPORT \\
route! : \mathrm{iseq}\, AIRPORT \\
\hline
(from?, to?) \in connected^{+} \\
\forall i : 1..(\# route! - 1) \bullet \\
\qquad (route!(i), route!(i+1)) \in connected \\
\hline
\end{array}$$

Note the precondition that states that there is an indirect connection between the starting point and destination. Also, note the use of the injective sequence iseq in the declaration of the *route*! sequence. This ensures that no AIRPORT will be visited more than once.

2 Stack

2.1 Narrative

This is the specification of a simple stack system. There is no limit on the size of the stack. We specify the system state and the two operations push and pop.

2.2 Basic Types and System State

$[X]$

$$\begin{array}{|l}
\textit{Stack} \\
\hline
s : \mathrm{seq}\, X \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{InitStack} \\
\hline
\textit{Stack}' \\
\hline
s' = \langle\ \rangle \\
\hline
\end{array}$$

2.3 Operations

$$\begin{array}{|l}
\textit{Push} \\
\hline
\Delta \textit{Stack} \\
x? : X \\
\hline
s' = s \frown \langle x? \rangle \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{Pop} \\
\hline
\Delta \textit{Stack} \\
x! : X \\
\hline
s = s' \frown \langle x! \rangle \\
\hline
\end{array}$$

Please note that you could chose whether to 'push' the element at the end of the sequence or the start of the sequence as long as you take this into account when 'popping'. Also please note that the specification of a Queue system would be very similar, the main difference being that the queue's version of pushing(enqueue) happens at the other end of the sequence from its version of popping (dequeue).

3 File System

3.1 Narrative

This specification deals with a File of characters and the manipulation of the characters in the file. The file is simple a sequence of CHAR's. We use a technique already seen in class and develop it a little. The technique breaks down the sequence into parts, these parts defined by e.g. their length and specfies how the new sequence is built from these parts. As is usual, the pre-conditions are stated first.

3.2 Basic Types and System State

$$[CHAR]$$

$$\begin{array}{|l}
\hline
\textit{FileSystem} \\
\hline
file : \mathrm{seq}\, CHAR \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{InitFileSystem} \\
\hline
FileSystem' \\
\hline
file = \langle\ \rangle \\
\hline
\end{array}$$

3.3 Operations

$$\begin{array}{|l}
\hline
\textit{DeleteChars} \\
\hline
\Delta FileSystem \\
from?, to? : \mathbb{N}_1 \\
\hline
1 \leq from? \leq to? \leq \# file \\
\exists before, toBeDeleted, after : seqCHAR \mid \\
\quad \# before = from? - 1 \land \\
\quad \# toBeDeleted = to? - from? + 1 \\
\qquad \bullet\ file = before \frown toBeDeleted \frown after \land \\
\qquad\quad file = before \frown after \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\ InsertChars\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta FileSystem \\
positionToInsert? : \mathbb{N}_1 \\
toBeInserted? : \mathrm{seq}\, CHAR \\
\hline
positionToInsert? \leq \# file \\
\exists before, after : seqCHAR \mid \\
\quad \# before = positionToInsert? - 1 \wedge \\
\qquad\qquad \bullet\ file = before \frown after \wedge \\
\qquad\qquad\quad file' = before \frown toBeInserted? \frown after \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\ PositionFirstOccurenceOfChar\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Xi FileSystem \\
ch? : CHAR \\
position? : \mathbb{N}_1 \\
\hline
ch? \in ran\ file \\
\exists before, after : seqCHAR \mid \\
\quad ch? \notin ran\ before \qquad\qquad \wedge \\
\quad position? = \# before + 1 \quad \wedge \\
\qquad\qquad \bullet\ file = before \frown \langle ch? \rangle \frown after \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\ PositionFirstOccurenceOfSequence\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Xi FileSystem \\
positionFound! : \mathbb{N}_1 \\
lookingFor? : \mathrm{seq}\, CHAR \\
\hline
\exists before, after : seqCHAR \mid \\
\quad (\neg\ \exists a, b : \mathrm{seq}\, CHAR \bullet \\
\qquad\quad before = a \frown lookingFor? \frown b) \wedge \\
\qquad\qquad positionFound! = \# before + 1 \\
\qquad\qquad\quad \bullet\ file = before \frown lookingFor? \frown after \\
\hline
\end{array}$$

4 Allocation of undergraduate projects

4.1 Narrative of the System

A third level college requires a computerised system to manage the allocation of the individual projects undertaken by its final-year degree students. Each student must be allocated to a personal supervisor from the lecturing staff.

Each lecturer has a maximum number of students which s/he is free to supervise. Each student and lecturer must list their topics of interest in descending order of their interest in the topic.

The system must then attempt to allocate students to supervisors (one at a time) in order to “maximise contentedness” with the allocations assigned. The concept of this “contentedness” is difficult but we define it as follows:

- We allocate students one at a time in the order they are presented to us.
- We decide that the priority is to allocate the student currently under consideration his/her most desired choice from lecturers who are currently available (i.e. those who have still not been assigned their full complement of students), the student being allocated to the lecturer who has this topic at the highest on his/her list of preferences.
- If more than one lecturer has the topic at the same level of priority, an arbitrary choice of supervisor can be made from these lecturers.
-

We are thus putting the students’ wishes above those of the lecturers.

4.2 Basic Types and System State

$[PERSON]$	the set of all people.
$[TOPIC]$	the set of all academic areas of interest.

$$\begin{array}{|l}
\multicolumn{1}{l}{ProjectAllocation} \\ \hline
studentInterests, lecturerInterests : PERSON \mathrel{+\!\!\!\!\rightarrow} \operatorname{iseq} TOPIC \\
allocations : PERSON \mathrel{+\!\!\!\!\rightarrow} PERSON \\
maxPlaces : PERSON \mathrel{+\!\!\!\!\rightarrow} \mathbb{N}_1 \\ \hline
\operatorname{dom} studentInterests \cap \operatorname{dom} lecturerInterests = \varnothing \\
\operatorname{dom} allocations \subseteq \operatorname{dom} studentInterests \\
\operatorname{ran} allocations \subseteq \operatorname{dom} lecturerInterests \\
\operatorname{dom} maxPlaces = \operatorname{dom} lecturerInterests \\
\forall lec : \operatorname{dom} maxPlaces \bullet \\
\qquad \#(allocation \rhd \{lec\}) \leq maxPlaceslec \\ \hline
\end{array}$$

4.3 Initial System State

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{InitProjectAllocation}} \\ \hline
\mathit{ProjectAllocation}' \\ \hline
\mathit{lecturerInterests}' = \varnothing \\
\mathit{studentInterests}' = \varnothing \\ \hline
\end{array}
$$

4.4 Operations

We now add specify the successful cases of operations to add a student to the system , add a lecturer to the system, allocate a supervisor to a student, deallocate a supervisor from a student, remove a topic from a lecturers preference and output the set of lecturers available for the supervision of a given topic.

4.4.1 Adding a student to the system

The inputs required for this operation are a student and that students list of topic preferences in descending order.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AddStudent}} \\ \hline
\Delta \mathit{ProjectAllocation} \\
s? : \mathit{PERSON} \\
ts? : \mathrm{iseq}\, \mathit{TOPIC} \\ \hline
s? \notin (\mathrm{dom}\, \mathit{studentInterests} \cup \mathrm{dom}\, \mathit{lecturerInterests}) \\
\\
\mathit{studentInterests}' = \mathit{studentInterests} \cup \{s? \mapsto ts?\} \\
\\
\mathit{lecturerInterests}' = \mathit{lecturerInterests} \\
\mathit{allocations}' = \mathit{allocations} \\
\mathit{maxPlaces}' = \mathit{maxPlaces} \\ \hline
\end{array}
$$

4.4.2 Adding a lecturer to the system

The inputs required for this operation are a lecturer, the list of topics which that lecturer is prepared to supervise and the maximum number of students which the lecturer may supervise (in descending order of preference)

$$\begin{array}{l}
\textit{AddLecturer} \\
\hline
\Delta \mathit{ProjectAllocation} \\
\mathit{lect?} : \mathit{PERSON} \\
\mathit{ts?} : \operatorname{iseq} \mathit{TOPIC} \\
\mathit{maxAlloc?} : \mathbb{N} \\
\hline
\mathit{lect?} \notin (\operatorname{dom} \mathit{studentInterests} \cup \operatorname{dom} \mathit{lecturerInterests}) \\
\\
\mathit{lecturerInterests}' = \mathit{lecturerInterests} \cup \{\mathit{lect?} \mapsto \mathit{ts?}\} \\
\mathit{maxPlaces}' = \mathit{maxPlaces} \cup \{\mathit{lect?} \mapsto \mathit{maxAlloc?}\} \\
\\
\mathit{studentInterests}' = \mathit{studentInterests} \\
\mathit{allocations}' = \mathit{allocations} \\
\hline
\end{array}$$

4.4.3 Allocating a student to a supervisor

The input to this operation is the student to be allocated. The operation must allocate the student to a supervisor in such a way that the student gets to do the highest priority topic from his/her list for which a supervisor is available. (***'Available'*** means that the topic appears in the preference list of at least one supervisor who still has places left for supervisions). The student is to be allocated to the lecturer who has this topic highest on his/her list of preferences.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Allocate}} \\
\hline
\Delta \mathit{ProjectAllocation} \\
\mathit{stud?} : \mathit{PERSON} \\
\\
\hline
\mathit{stud?} \notin \operatorname{dom} \mathit{studentInterests} \\
\mathit{stud?} \notin \operatorname{dom} \mathit{allocations} \\
\\
\exists\, \mathit{sup} : \operatorname{dom} \mathit{lecturerInterests};\ t : \mathit{TOPIC};\ i, j : \mathbb{N} \mid \\
\qquad \mathit{maxPlaces}(\mathit{sup}) > \#(\mathit{allocations}^{\sim} (\!| \{\mathit{sup}\} |\!)) \ \wedge \\
\qquad i \mapsto t \in \mathit{studentInterests}(\mathit{stud?}) \wedge \\
\qquad j \mapsto t \in \mathit{lecturerInterests}(\mathit{sup}) \qquad \bullet \\
\qquad\quad (\\
\qquad\quad\ \forall\, \mathit{lect} : \operatorname{dom} \mathit{lecturerInterests};\ k : \mathbb{N} \mid \\
\qquad\qquad\qquad \mathit{maxPlaces}(\mathit{lect}) > \#(\mathit{allocation}^{\sim} (\!| \{\mathit{lect}\} |\!)) \bullet \\
\qquad\quad\ (\\
\qquad\qquad (k \mapsto t \in \mathit{lecturerInterestslect} \Rightarrow k \geq j) \wedge \\
\qquad\qquad (\operatorname{ran}(1..i - 1 \lhd \mathit{studentInterests}(\mathit{stud?}))) \quad \cap \\
\qquad\qquad (\operatorname{ran}(\mathit{lecturerInterests}\ \mathit{lect})) \qquad\qquad\qquad = \varnothing \\
\qquad\quad\ ) \qquad\qquad\quad \wedge \\
\qquad\quad\ \mathit{allocations}' = \mathit{allocations} \cup \{\mathit{stud?} \mapsto \mathit{sup}\} \\
\qquad) \\
\\
\mathit{studentInterests}' = \mathit{studentInterests} \\
\mathit{lecturerInterests}' = \mathit{lecturerInterests} \\
\mathit{maxPlaces}' = \mathit{maxPlaces} \\
\hline
\end{array}
$$

4.4.4 Deallocating a student from a supervisor

The input to this operation is the student to be allocated. The precondition is that the student is actually allocated to a supervisor. We should (and do) not need to had the supervisor as an input. We calculate that as part of the operation.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{DeAllocate}} \\
\hline
\Delta \textit{ProjectAllocation} \\
\textit{stud?} : \textit{PERSON} \\
\hline
\exists\, \textit{sup} : \operatorname{dom} \textit{lecturerInterests} \quad \bullet \\
\qquad (\textit{stud?} \mapsto \textit{sup} \in \textit{allocations} \ \wedge \\
\qquad\ \textit{allocations}' = \textit{allocations} \setminus \{\textit{stud?} \mapsto \textit{sup}\}) \\
\\
\textit{studentInterests}' = \textit{studentInterests} \\
\textit{lecturerInterests}' = \textit{lecturerInterests} \\
\textit{maxPlaces}' = \textit{maxPlaces} \\
\hline
\end{array}
$$

Exercise: Try to rewrite this without the use of the $\exists$ - (**Hint** try domain anti-restriction)

4.4.5 Removing a topic from a lecturer's preference list

The lecturer must already be in the system and the topic must be in the lecturer's preference list. If the lecturer is already allocated to supervise this topic, it's too bad the lecturer has made the agreement (i.e. we don't make any consequent allocation changes). However, the lecturer will not take any further students on this topic.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RemoveLecsTopic}} \\
\hline
\Delta \textit{ProjectAllocation} \\
\textit{lect?} : \textit{PERSON} \\
\textit{t?} : \textit{TOPIC} \\
\hline
\textit{lect?} \in \operatorname{dom} \textit{lecturerInterests} \\
\textit{t?} \in \operatorname{ran}\, (\textit{lecturerInterests}(\textit{lect?}) \\
\\
\textit{lecturerInterests}' = \textit{lecturerInterests} \ \oplus \\
\qquad\qquad \{\textit{lect?} \mapsto \textit{squash}\ (\textit{lecturerInterests}(\textit{lect?}) \rhd \{\textit{t?}\})\} \\
\textit{studentInterests}' = \textit{studentInterests} \\
\textit{allocations}' = \textit{allocations} \\
\textit{maxPlaces}' = \textit{maxPlaces} \\
\hline
\end{array}
$$

4.4.6 The set of all lecturers available for supervision of a given topic

For a given lecturer to be a member of this set, the topic must be in that lecturer's list of preferences and the lecturer must still have supervision places available.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{LecturersAvailable}}\\
\hline
\Xi \mathit{ProjectAllocation}\\
t? : \mathit{TOPIC}\\
ps! : \mathbb{P}\,\mathit{PERSON}\\
\hline
ps! = \{p : \operatorname{dom} \mathit{lecturerInterests} \mid\\
\qquad\qquad t? \in \operatorname{ran}(\mathit{lectInterests}(p))\\
\qquad\qquad \mathit{maxPlaces}(p) > \#(\mathit{allocation}^{\sim} \llparenthesis \{p\} \rrparenthesis)\\
\qquad\qquad\qquad\qquad\qquad\qquad \bullet\ p\}\\
\hline
\end{array}
$$

4.4.7 Exercises on Project Allocation System

1. Write a predicate which specifies a state where there are no unallocated students.

2. Write a predicate which specifies a state where all students in the system are unallocated.

3. Write a schema for an operation to remove a student *stud*? from the system.

4. Write a schema for an operation to remove a lecturer *lect*?from the system.

5. Write a Z expression for the set of all the students allocated to a given supervisor *sup*?.

6. Write a Z expression for the set of all students with the same supervisor as a given student *p*?.

7. Write a schema for an operation to add a new topic to a given lecturer's (*lect*?) priority list at a given position. If the position is greater than the length of the list, the topic should be added to the back of the list.

5 Genealogical Database

5.1 Narrative of the System:

A database is required to keep track of genealogical relationships between horses (family trees). It would be possible to represent the required relationships (parent, grandparent etc) separately, but this would unnecessarily complicate the specification. Instead, we represent the minimum information necessary to be able to define operations to output any required relationships. The most fundamental genealogical relationship is that of parent to child,, and this, together with the sex of the horses in the database will be enough to enable us to specify all the operations we require.

5.2 Basic Types and System State

$$\begin{array}{ll} [HORSE] & \text{the set of all people.} \\ GENDER ::= male \mid female & \text{Free type.} \end{array}$$

$$\begin{array}{|l} \hline \textit{GenDB} \\ parent : HORSE \leftrightarrow HORSE \\ sex : HORSE \nrightarrow GENDER \\ \hline (\operatorname{dom} parent \cup \operatorname{ran} parent) \subseteq \operatorname{dom} sex \\ \forall\, h : HORSE \bullet \\ \qquad (h, h) \notin parent^{+} \\ \forall\, p, q, r : HORSE \bullet \\ \qquad (\{(p, q), (p, r)\} \subseteq parent \wedge q \neq r) \\ \qquad \Rightarrow \quad sex(q) \neq sex(p) \\ \hline \end{array}$$

5.3 Initial System State

$$\begin{array}{|l} \hline \textit{InitGenDB} \\ GenDB' \\ \hline parent' = \varnothing \\ sex' = \varnothing \\ \hline \end{array}$$

5.4 Operations

We now add specify the successful cases of a few operations.

5.4.1 Adding a person to the database

The inputs required for this operation are a horse and its gender.

$AddHorse$

$$\begin{array}{l}
\Delta GenDB \\
horse? : HORSE \\
gen? : GENDER \\
\hline
horse? \notin \operatorname{dom} sex \\
\\
sex' = sex \cup \{horse? \mapsto gen?\} \\
parent' = parent
\end{array}$$

5.4.2 Adding a parent/offspring relationship to the database

The inputs required for this operation are a offspring and a potential parent.

$AddRel$

$$\begin{array}{l}
\Delta GenDB \\
parentHorse?, childHorse? : HORSE \\
\\
\hline
\{parentHorse?, childHorse?\} \subseteq \operatorname{dom} sex \\
parentHorse? \mapsto childHorse? \notin parent \\
childHorse? \mapsto parentHorse? \notin parent \\
\#(\{childHorse?\} \lhd parent) \leq 1 horse? \notin \operatorname{dom} sex \\
\\
\forall\, h : HORSE \bullet \\
\qquad\qquad (childHorse?, h) \in parent \Rightarrow sex(h) \neq sex(parentHorse?) \\
horse? \notin \operatorname{dom} sex \\
\\
\\
parent' = parent \cup (childHorse?, parentHorse?) \\
sex' = sex
\end{array}$$

5.4.3 Changing the name of a person in the database

The inputs to this operation are the old name and the new name.

$$\begin{array}{l}
\textit{ChangeName} \\
\hline
\Delta GenDB \\
oldName?, newName? : HORSE \\
\hline
oldName? \in \mathrm{dom}\, sex \\
newName? \notin domsex \\
\\
sex' = (\{oldName?\} \lhd sex) \cup \{new? \mapsto sex(oldName?)\} \\
parent' = (\ \{oldName?\} \lhd parent \rhd \{oldName?\}) \\
\qquad\qquad \cup \{h : HORSE \mid h \in parent \llparenthesis \{oldName?\} \rrparenthesis \bullet newName? \mapsto h\} \\
\qquad\qquad \cup \{h : HORSE \mid h \in parent^{\sim} \llparenthesis \{oldName?\} \rrparenthesis \bullet h \mapsto newName?\} \\
\hline
\end{array}$$

5.4.4 Changing the sex of a person in the database

The input to this operation is the parent whose sex we wish to change.

$$\begin{array}{l}
\textit{ChangeSex} \\
\hline
\Delta GenDB \\
horse? : HORSE \\
\hline
horse? \in \mathrm{dom}\, sex \\
newName? \notin domsex \\
\\
sex' = sex \oplus \\
\qquad \{h : HORSE;\ s : GENDER \mid h \in (parent^{\sim} \mathbin{;} parent)^{+} \llparenthesis \{horse?\} \rrparenthesis \wedge \\
\qquad\qquad\qquad (s \neq sex(h)) \bullet h \mapsto s\} \\
\hline
\end{array}$$

5.5 Exercises on Genealogical Database

1. Specify operations to return the set of all horses who have the following relationships to a given horse h?

   (a) The parents of horse h?

   (b) The grandparents of horse h?

   (c) The grandchildren of horse h?

   (d) The descendants of horse h?

   (e) The siblings of horse h?

   (f) The aunts of horse h? (aunt is defined as a parent's female sibling)

2. Give a Z expression for the set of all horses in the database who have no relatives in the database.

3. Give a Z expression for the set of all horses in the database who have no siblings in the database.