

## Exercise Sheet 7 — Example Solution

### Z-Specification of CSS — Part II

#### The System-State of CSS

The following schema *ClassSchedulingSystem* provides a preliminary model (version V1) of the state-space of our class-scheduling-system, which we will extend and refine in the final specification.<sup>1</sup>

**section** *stateV1* **parents** *classSchedule*

*INIT* ::= *Valid* | *Invalid*

<p><i>ClassSchedulingSystem</i></p> <p><i>eventOfferTable</i> : <i>EventOfferTable</i> <i>roomTable</i> : <i>RoomTable</i> <i>classScheduleTable</i> : <i>ClassScheduleTable</i> <i>classScheduleTableInit</i> : <i>INIT</i></p>
--

#### Remarks:

- Recall that in the example solution of Exercise Sheet 6 we defined:

$$\mathit{ClassScheduleTable} == \mathit{ClassScheduleID} \leftrightarrow \mathit{ClassSchedule}$$

where

$$\mathit{ClassSchedule} == \mathit{EventOfferID} \leftrightarrow \mathbb{P}(\mathit{DayBeginDuration} \times \mathit{RoomID})$$

Hence, *classScheduleTable* is a table that contains ‘illegal’ class-schedules in the sense of the characterizing predicate *ClassSchedules*; this also extends to the state-space modeling above and represents a problem. Since we cannot really foresee what will be needed by the operations that actually create entries into the class-schedule table, we will stick to this straightforward approach in order to study the design in several steps.

- *classScheduleTableInit* is a flag that indicates whether class-schedules have been generated or not. It corresponds *exactly* to the INIT-variable in the statechart-diagram in the example solution of Exercise Sheet 4.

---

<sup>1</sup>The following has been produced using ZETA (integrated into xemacs), both for the latex source and for Z type-checking. This part of the specification builds up on the previous part (in example solution of Exercise Sheet 6): the Z-section *stateV1* imports the section *classSchedule*.

# Specification of the Conflicts

## section *conflicts1* parents *classSchedule*

We now formalize auxiliary functions that compute room-features conflicts (violations of the requirements enforced on a room) and instructor conflicts (violations of the fact that an instructor cannot take part in two or more events during the same time-frame).

The main idea is to compute the cardinality of sets of tuples, which characterize the conflicts.

The system assigns, within a schedule, a pair of room and date to an event. A room-features conflict occurs when the required features (e.g. Beamer and Projector) of an event are not a subset of the features of the assigned room, or when the number of expected attendants is higher than the capacity of the room.

We formally define the Z-function *RoomFeatureConflicts* as follows:

$$\begin{array}{|l}
 \hline
 \textit{RoomFeatureConflicts} : \textit{ClassSchedule} \times \textit{RoomTable} \times \textit{EventOfferTable} \rightarrow \mathbb{N} \\
 \hline
 \forall s : \textit{ClassSchedule}; r : \textit{RoomTable}; eo : \textit{EventOfferTable} \bullet \\
 \textit{RoomFeatureConflicts}(s, r, eo) = \\
 \# \{ eoid : \text{dom } s; rid : \text{dom } r \mid \exists dbd : \textit{DayBeginDuration} \bullet (dbd, rid) \in (s \textit{ eoid}) \wedge \\
 ((eo \textit{ eoid}).\textit{Features}) \not\subseteq (r \textit{ rid}).\textit{Features} \\
 \vee ((eo \textit{ eoid}).\textit{ExpectedAttendants}) > (r \textit{ rid}).\textit{Capacity} \} \\
 \hline
 \end{array}$$

The following is an alternative, more refined, formalization:

$$\begin{array}{|l}
 \hline
 \textit{RoomFeatureConflicts2} : \textit{ClassSchedule} \times \textit{RoomTable} \times \textit{EventOfferTable} \rightarrow \mathbb{N} \\
 \hline
 \forall s : \textit{ClassSchedule}; r : \textit{RoomTable}; eo : \textit{EventOfferTable} \bullet \\
 \textit{RoomFeatureConflicts2}(s, r, eo) = \\
 \# \{ eoid : \text{dom } s; rid : \text{dom } r; dbd : \textit{DayBeginDuration} \mid (dbd, rid) \in (s \textit{ eoid}) \wedge \\
 ((eo \textit{ eoid}).\textit{Features}) \not\subseteq (r \textit{ rid}).\textit{Features} \\
 \vee ((eo \textit{ eoid}).\textit{ExpectedAttendants}) > (r \textit{ rid}).\textit{Capacity} \} \\
 \hline
 \end{array}$$

Both formalizations are possible according to the specification, and we illustrate the differences between them by means of an example: let *schedule* be such that

$$\begin{array}{l}
 \textit{schedule} \textit{ EO13} = \{ (\langle \textit{Day} = \textit{Mo}, \textit{Begin} = 9, \textit{Duration} = 2 \rangle, 052 - 02 - 017), \\
 (\langle \textit{Day} = \textit{Th}, \textit{Begin} = 9, \textit{Duration} = 2 \rangle, 052 - 02 - 017) \}
 \end{array}$$

where the features of room 052–02–017 do not meet the requirements of *EO13*. The function *RoomFeatureConflicts2* counts this conflict twice, but *RoomFeatureConflicts* counts it only once.

For simplicity we choose to work with the first formalization. Note also that we could give yet another formalization where the number of missing features is explicitly counted, thus weighing a room-features conflict where two requirements (e.g. beamer and projector) are not met more than a room-features conflict where only one requirement (e.g. projector) is not met.

An instructor conflict occurs when a class-schedule requires an instructor to take part in two (or more) events at the same time (i.e. during the same time-frame). In order to define formally the function *InstructorConflicts* we exploit the auxiliary function *TimeFrame* (of the example solution of Exercise Sheet 6):

$$\begin{array}{|l}
 \hline
 \textit{InstructorConflicts} : \textit{ClassSchedule} \times \textit{EventOfferTable} \rightarrow \mathbb{N} \\
 \hline
 \forall s : \textit{ClassSchedule}; eo : \textit{EventOfferTable} \bullet \\
 \textit{InstructorConflicts}(s, eo) = \\
 \# \{ eoid1 : \text{dom } s; eoid2 : \text{dom } s; i : \textit{Instructor} \mid \\
 eoid1 \neq eoid2 \\
 \wedge i \in (eo \textit{ eoid1}).\textit{Instructors} \\
 \wedge i \in (eo \textit{ eoid2}).\textit{Instructors} \\
 \wedge \textit{TimeFrame}\{sid : s \textit{ eoid1} \bullet \textit{first } sid\} \cap \textit{TimeFrame}\{sid : s \textit{ eoid2} \bullet \textit{first } sid\} \neq \emptyset \} \text{div } 2 \\
 \hline
 \end{array}$$

**Remark:** we have to divide by 2 since  $eid1$  and  $eid2$  both occur in  $s : ClassSchedule$ , so that the instructor conflicts within the  $DayBeginDurations$  of  $eid1$  and  $eid2$  are counted twice.

## The Operations of the Class-Scheduling-System

**section**  $operationsV1$  **parents**  $stateV1$

Operations model transitions in the state-space of the system. They correspond *exactly* to the transitions in the example solution of Exercise Sheet 4. We now formalize (preliminary versions, based on the preliminary version of the state space, of) the operations add, delete and update for both rooms and event-offers. Note that we take a ‘cautious’ approach: whenever there is danger than an operation produce inconsistencies in the generated class-schedules, we delete all of them and start from scratch (by setting  $classScheduleTable' = \emptyset$ ). The final specification will contain more refined operations.

— <i>AddRoom</i> —
$\Delta ClassSchedulingSystem$
$r? : Room$
$\exists rid : RoomID \bullet (rid \notin \text{dom } roomTable) \wedge (roomTable' = roomTable \cup \{rid \mapsto r?\})$
$eventOfferTable' = eventOfferTable$
$classScheduleTable' = \emptyset$
$classScheduleTableInit = Invalid$

— <i>DeleteRoom</i> —
$\Delta ClassSchedulingSystem$
$rid? : RoomID$
$rid? \in \text{dom } roomTable$
$roomTable' = roomTable \setminus \{rid? \mapsto roomTable\ rid?\}$
$eventOfferTable' = eventOfferTable$
$classScheduleTable' = \emptyset$
$classScheduleTableInit' = Invalid$

— <i>UpdateRoom</i> —
$\Delta ClassSchedulingSystem$
$rid? : RoomID$
$r? : Room$
$rid? \in \text{dom } roomTable$
$roomTable' = roomTable \setminus \{rid? \mapsto roomTable\ rid?\} \cup \{rid? \mapsto r?\}$
$eventOfferTable' = eventOfferTable$
$classScheduleTable' = \emptyset$
$classScheduleTableInit' = Invalid$

Note that we can equivalently choose to work with the overwrite operator  $\oplus$  to define:

$$roomTable' = roomTable \oplus \{rid? \mapsto r?\}$$

The operations for adding, deleting and updating event-offers are similar to the corresponding ones for rooms.

— *AddEventOffer* —

$\Delta$  *ClassSchedulingSystem*

$eo? : \text{EventOffer}$

$\exists eoid : \text{EventOfferID} \bullet (eoid \notin \text{dom } \text{eventOfferTable})$   
 $\wedge (\text{eventOfferTable}' = \text{eventOfferTable} \cup \{eoid \mapsto eo?\})$   
 $\text{roomTable}' = \text{roomTable}$   
 $\text{classScheduleTable}' = \emptyset$   
 $\text{classScheduleTableInit}' = \text{Invalid}$

— *DeleteEventOffer* —

$\Delta$  *ClassSchedulingSystem*

$eoid? : \text{EventOfferID}$

$eoid? \in \text{dom } \text{eventOfferTable}$   
 $\text{eventOfferTable}' = \text{eventOfferTable} \setminus \{eoid? \mapsto \text{eventOfferTable } eoid?\}$   
 $\text{roomTable}' = \text{roomTable}$   
 $\text{classScheduleTable}' = \emptyset$   
 $\text{classScheduleTableInit}' = \text{Invalid}$

— *UpdateEventOffer* —

$\Delta$  *ClassSchedulingSystem*

$eoid? : \text{EventOfferID}$

$e? : \text{EventOffer}$

$(eoid? \in \text{dom } \text{eventOfferTable})$   
 $\text{eventOfferTable}' = \text{eventOfferTable} \setminus \{eoid? \mapsto \text{eventOfferTable } eoid?\} \cup \{eoid? \mapsto e?\}$   
 $\text{roomTable}' = \text{roomTable}$   
 $\text{classScheduleTable}' = \emptyset$   
 $\text{classScheduleTableInit}' = \text{Invalid}$

As for *UpdateRoom*, we can equivalently choose to work with the overwrite operator  $\oplus$  to define:

$$\text{eventOfferTable}' = \text{eventOfferTable} \oplus \{eoid? \mapsto e?\}$$