ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

J-R. Abrial, T.S. Hoang and M. Schmalz

DEPLOY Block Course

(ETH Zurich 9-11 April, 2008)

Exercise Sheet 1: Requirement Document and Introduction to the Rodin Platform

Description of Work

The first aim of this exercise is to come up with a clean requirement document for a simple access control system of a building. You are given the informal descriptions of the system as follows. It is quite clumsy and poorly written (on purpose).

The second aim of this exercise is to be familiar with the Modeling interface and the Interactive Proving interface of the Rodin Platform. The example that we choose to use in this exercise is the development of search programs in the slides "A Summary of the Event-B Modeling Notation".

The last exercise to develop an algorithm to find the maximum number within an array is intended to be a challenge exercise.

An Access Control System

Here is an informal description of the system.

> The system consists of the following parts:
>
> - a set of rooms,
> - among them, a special room called "hallway".
> - Rooms are connected by doors.
> - The hallway is connected to all rooms.
> - A person will be authorized to be in certain rooms.

Out of the above description, you need to produce a clean requirement document. You are suggested to use the taxonomy of requirements:

- EQP for equipment,
- FUN for functional,
- SAF for safety properties.

For this exercise, you should work in group of 2 or 3 people.

Solution:

We provide our requirements document. It is not perfect, but we hope to give you some helpful inspirations.

Our access control system consists of rooms, doors connecting these rooms, and people who move between the rooms.

EQP1	The system consists of *rooms*, *doors*, and *people*.

There is a special room called "hallway".

EQP2	We call a special room *hallway*.

The rooms in the building are connected by doors.

EQP3	A *door* connects two rooms.

A door connects two rooms, but it never connects a room to itself. Moreover, the connection between rooms is symmetric. if a room 1 is connected with room 2, then vice-versa, the room 2 must be connected with room 1.

EQP4	No door connects a room to itself.

EQP5	If a door connects room 1 with room 2, then it also connects room 2 with room 1.

The hallway is connected to all rooms but itself.

EQP6	The hallway is connected to all rooms but itself.

A person is in at most one room of the building at a time.

FUN1	A person is in at most one room.

To enter a room in the building, people first go through the hallway. Vice-versa, people have to leave the building via the hallway.

FUN2	A person enters and leaves the building via the hallway.

Once inside the building, people can move from one room to another room connected to each other.

FUN3	If a person is in room 1 and room 1 is connected with room 2, then the person can go to room 2.

However, there are some authorisations that permit people to be in some rooms but not into others.

FUN4	A person is allowed to be in certain rooms.

Since the people enter and leave the building via the hallway, every person must be allowed to be in the hallway.

FUN5	Each person is allowed to be in the hallway.

A critical property of the system is that there is no unauthorised access. In fact, this is somehow unrealistic: unauthorised access sometimes happens, even in prisons. We also leave it completely open how the access control is realized (keycards, fingerprints, RFID, guards, . . .). See the following therefore as a simplification.

SAF1	If a person is in a room, then she is authorized to be in that room.

Development of Search Algorithms

Proving the Remaining Obligations

In the development "search" that was distributed to you along with the summary slides, there are some proof obligations that are not discharged. Use the interactive proving facility of the Rodin Platform to discharge those obligations.

- Model **m_0b**: *search/act1/FIS*.
- Model **m_1a**: *search/grd1/WD*, *progress/grd1/WD* and *progress/inv2/INV*.
- Model **m_1b**: *search/grd1/WD*, *progress/grd1/WD*, *progress/inv2/INV* and *progress/VAR*.

A useful point to know here is how to reuse proofs (or rather part of the proof) by "Copy/Paste".

- Right click on the proof tree node where you want to copy the proof sub-tree starting from that node and choose "Copy".
- Right click on the proof tree node where you want to re-use the copied proof tree on and choose "Paste".

Solution:

See the development within the Rodin Platform *search_proved.zip*.

A Different Search Algorithm

This is the exercise from the last slide (slide 33) of the set of slides "A Summary of the Event-B Modeling Notation".

Create a new refinement of **m_0a** or **m_0b** in order to obtain the following final program:

$i, j := 1, n+1$;	**initialisation**
WHILE $f(j-1) \neq v$ DO	
$j := j-1$	**progress**
END ;	
$i := j-1$	**search**

Write down on paper the following:

- guard strengthening proof obligations if any, and
- simulation proof obligations if any

for the event *search* of the new machine. Compare the result with the proof obligations generated by the Rodin Platform.

Solution:

See the machines **m_1a** and **m_1b** in the development within the Rodin Platform *search_backward.zip*.

Below is the new version of the machine **m_1a**.

machine
m_1a
refines
m_0a
sees
ctx_0
variables
i
j
invariants and thms.
inv1 : $j \in 1 \mathinner{..} n+1$
inv2 : $v \notin f[j \mathinner{..} n]$
theorems
thm1 : $v \in f[1 \mathinner{..} j-1]$
variant
j
events
...
end

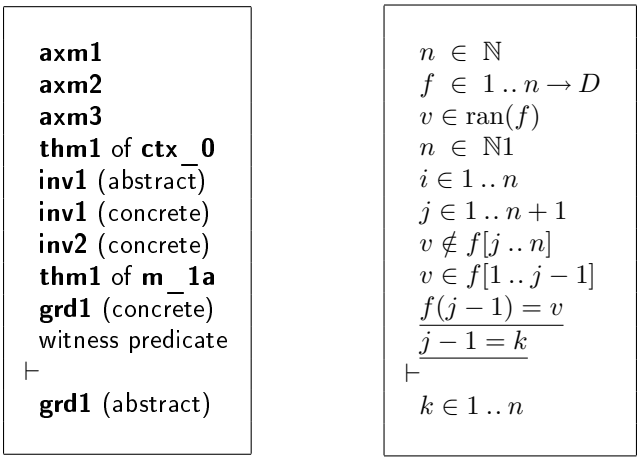
initialisation $\widehat{=}$
status
ordinary
then
act1 : $i := 1$
act2 : $j := n+1$
end

search $\widehat{=}$
status
ordinary
refines
search
when
grd1 : $f(j-1) = v$
with
k : $j-1 = k$
then
act1 : $i := j-1$
end

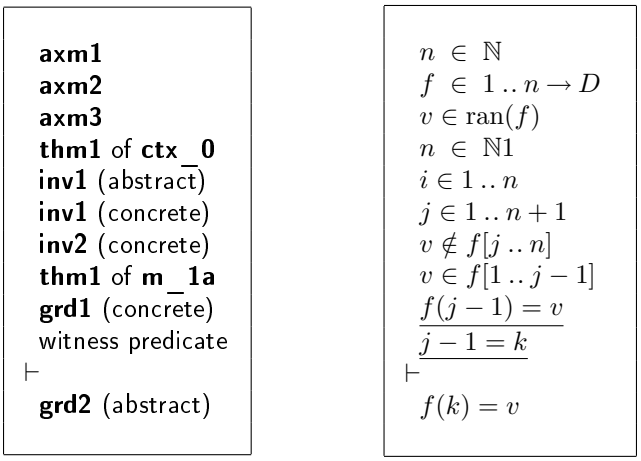
progress $\widehat{=}$
status
convergent
when
grd1 : $f(j-1) \neq v$
then
act1 : $j := j-1$
end

(abstract-)**search** $\widehat{=}$
status
ordinary
any
k
where
grd1 : $k \in 1 \mathinner{..} n$
grd2 : $f(k) = v$
then
act1 : $i := k$
end

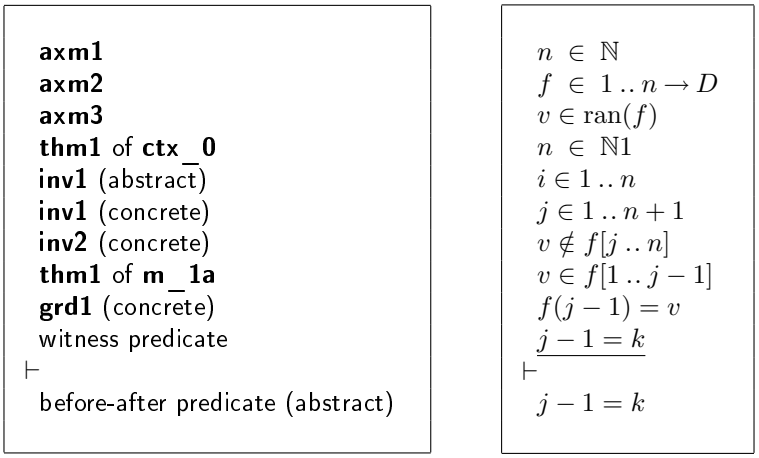
And here is the proof obligation *search/grd1/GRD*



And here is the proof obligation *search/grd2/GRD*



And here is the proof obligation *search/act1/SIM*



Finding the Maximum Number of An Array (Challenge Exercise)

In this exercise, we develop a program to find the maximum number of an array. Assume that we have an array "a" which has "n" elements, with the indexes starting from 1 to n. The algorithm to find the maximum number within an array (or rather the index of the maximum number) is as follows.

We use to indexes x and y where x starting from 1 and y is initially n. At each step of the algorithm, we do the following:

- if $x = y$ then x is the index of the maximum number.
- otherwise, i.e. $x \neq y$, we compare the value of a at indexes x and y and do the following:
  - if $a(x) < a(y)$, increase x by 1.
  - otherwise, decrease y by 1.

Formalize the above algorithm in Event-B and establish the sequential program out of it. Do not forget to prove the convergence of new events in a refinement.

Solution:

See the development within the Rodin Platform archive *maximum.zip*.