

A Sample Event-B Model

Thai Son Hoang

February 25, 2013

This document provide a sample Event-B model using typeset using the `eventB` package.

1 Contexts

Our initial context `coursesCtx` contains a carrier set CRS denoting the set of courses that can be offered by the club. Moreover, `coursesCtx` includes a constant m denoting the maximum number of courses that the club can have at the same time. The context `coursesCtx` is as follows.

sets : CRS

constants : m

axioms :
axm0_1 : $\text{finite}(CRS)$
axm0_2 : $m \in \mathbb{N}1$
thm0_1 : $0 < m$
axm0_3 : $m \leq \text{card}(CRS)$

Note that we label the axioms and theorems with the prefixes denoting the role of the modelling elements, i.e., axm and thm , with some numbers. For example, $axm0_1$ denotes the first (i.e., 1) axiom for the initial model (i.e., 0). We apply this systematic labelling through out our development.

The assumption on CRS and m are captured by the axioms and theorems as follows. Axiom $axm0_1$ states that CRS is finite. Axiom $axm0_2$ states that m is a member of the set of natural numbers (i.e., m is a natural number). Finally, $axm0_3$ states that m cannot exceed the number of possible courses that can be offered by the club, represented as $\text{card}(CRS)$, the cardinality of CRS . A derived property of m is presented as theorem $thm0_1$.

$thm0_1/THM$ A proof obligation is generated for $thm0_1$ as follows. Notice that $axm0_3$ does not appear in the set of hypotheses for the obligation, since it is declared after $thm0_1$. By convention, each proof obligation is labelled according to the element involved and the name of the proof obligation rule. Here $thm0_1/THM$ indicates that it is a **THM** proof obligation for $thm0_1$.

...

The obligations can be trivially discharged since $\mathbb{N}1$ is the set of all positive natural numbers, i.e., $\{1, 2, \dots\}$.

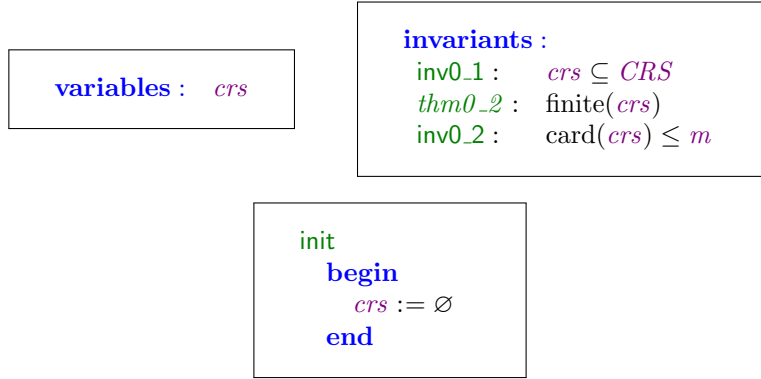
$axm0_3/WD$ It is required to prove that $axm0_3$ is well-defined. The corresponding proof obligation is as follows.

...

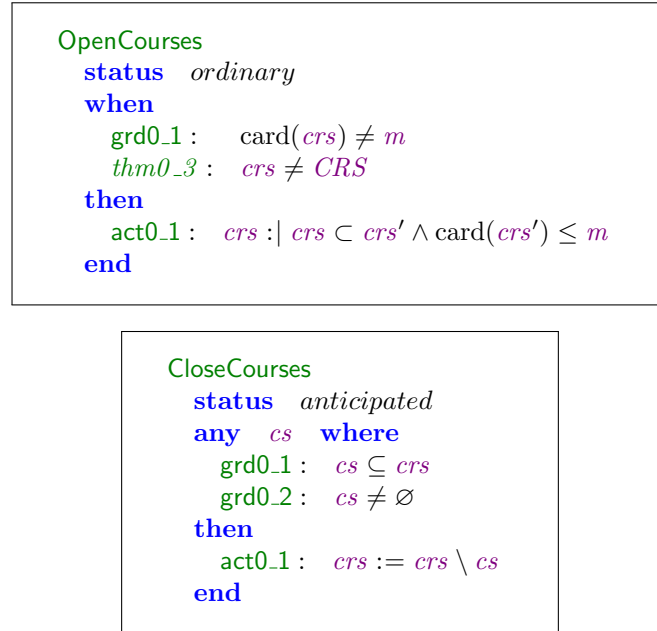
Since the goal appears amongst the hypotheses, the proof obligation can be discharged trivially. Note that the order of appearance of the axioms is important. In particular, `axm0_2` needs to be declared before `axm0_3`.

2 Machines

We develop machine **m0** of the initial model, focusing on courses opening and closing. This machine sees context **coursesCtx** as developed in Section 1, hence as a result has access to the carrier set *CRS* and constant *m*. We model the set of opened courses by a variable, namely *crs*. Invariant `inv0_1` states that it is a subset of available courses *CRS*. A consequence of this invariant and of axiom `axm0_1` is that *crs* is finite, and this is stated in **m0** as theorem `thm0_2`. invariant `inv0_2` states that the number of opened courses, i.e., $\text{card}(\text{crs})$ is bounded above by *m*. Initially, all courses are closed hence *crs* is set to the empty set (\emptyset).



We model the opening and closing of courses using two events `OpenCourses` and `CloseCourses` as follows.



We choose purposely to model these events using different features of Event-B. In `OpenCourses`, we use a nondeterministic action to model the fact that some new courses are opened, i.e., $\text{crs} \subset \text{crs}'$, as long as the number of opened courses will not exceed its limit, i.e., $\text{card}(\text{crs}') \leq m$. The guard of the event states that the current number of opened courses has not yet reached the limit.

CloseCourses models the set of courses that are going to be closed using parameter *cs*. It is a non-empty set of currently opened courses which is captured by **CloseCourses**' guard. The action is modelled straightforwardly by removing *cs* out of the set *crs*.

We set the convergence status for **OpenCourses** and **CloseCourses** to be *ordinary* and *anticipated*, respectively. We delay the reasoning about the convergence of **CloseCourses** to later refinements. Our intention is to prove that there can be only finitely many occurrences of **CloseCourses** between any two **OpenCourses** events.

We present some of the obligations to illustrate what needs to be proved for the consistency of **m0**. We applied the proof obligation rules as showed earlier in this Section. Notice that we can take the axioms and theorems of the seen context **coursesCtx** as hypotheses in the proof obligations. For clarity, we show only parts of the hypotheses that are relevant for discharging the proof obligations. Moreover, we also show the proof obligations in their simplified form, e.g., when the events' assignments are deterministic.

thm0_2/THM This obligation is in order to ensure that *thm0_2* is derivable from previously declared invariants.

...

The proof obligation holds trivially since *crs* is a subset of a finite set, i.e., *CRS*.

init/inv0_2/INV This obligation ensures that the initialisation *init* establishes invariant *inv0_2*.

...

Since the cardinality of the empty set \emptyset is 0, the proof obligation holds trivially.

OpenCourses/*thm0_3*/THM This obligation ensures that *thm0_3* is derivable from the invariants and the previously declared guards of **OpenCourses**.

...

Informally, we can derive from the hypotheses that $\text{card}(\textit{crs}) < \text{card}(\textit{CRS})$, hence *crs* must be different from *CRS*.

OpenCourses/*act0_1*/FIS This obligation ensures that the nondeterministic assignment of **OpenCourses** is feasible when the event is enabled.

...

The reasoning about the proof obligation is as follows. Since *crs* is different from *CRS*, there exists an element *c* which is closed, i.e., not in *crs*. By adding *c* to the set of opened courses, we strictly increase the number of opened courses by 1. Moreover, the number of opened courses after executing the event is still within the limit since originally it is strictly below the limit.

CloseCourses/*inv0_2*/INV This obligation is simplified accordingly since the assignment is deterministic. The purpose of the obligation is to prove that **CloseCourses** maintains invariant *inv0_2*.

...

Since removing some courses *cs* from the set of opened courses *crs* can only reduce its number, the proof obligation can be trivially discharged.

DLF/THM The deadlock-freeness condition is encoded as theorem *DLF* of machine **m0**, which results in the following proof obligation.

...

We reason as follows. If $\text{card}(crs) \neq m$, the goal trivially holds. Otherwise, i.e., $\text{card}(crs) = m$, since $m \neq 0$, we have that $crs \neq \emptyset$. As a result, we can prove that $\exists cs. cs \subseteq crs \wedge cs \neq \emptyset$ by instantiating cs with crs itself.

3 Context Extension

3.1 Context membersCtx

This is an initial context (i.e., does not extend any other context) containing a carrier sets *MEM*. *MEM* represents the set of club members, with an axiom stating that it is finite.

sets : <i>MEM</i>	axioms : axm1_1 : finite(<i>MEM</i>)
-------------------	-------------------------------------------

3.2 Context participantsCtx

This context extends the previously defined context **membersCtx** and is as follows.

constants : <i>PRTCPT</i>	axioms : axm1_2 : <i>PRTCPT</i> \subseteq <i>MEM</i> thm1_1 : finite(<i>PRTCPT</i>)
---------------------------	-----------------------------------------------------------------------------------------------

Constant *PRTCPT* denotes the set of participants which must be members of the club (axm1_2). Theorem *thm1_1* states that there can be only a finite number of participants, which gives rise to the following trivial proof obligation.

...

An important point is that axiom *axm1_1* of the abstract context **membersCtx** appears as a hypothesis in the proof obligation.

3.3 Context instructorsCtx

This context extends two contexts **coursesCtx** and **membersCtx**, and introduces two constants, namely *INSTR* and *instrs*. *INSTR* models the set of instructors which are members of the club (axm1_3). Constant *instrs* models the relationship between courses and instructors and is constrained by *axm1_4*: it is a *total function* from *CRS* to *INSTR*.

constants : <i>INSTR</i> , <i>instrs</i>
axioms : axm1_3 : <i>INSTR</i> \subseteq <i>MEM</i> axm1_4 : <i>instrs</i> $\in CRS \rightarrow INSTR$

4 Machine Refinement

4.0.1 Machine m1

Machine **m1** sees contexts **instructorsCtx** and **participantsCtx**. As a result, it implicitly sees **coursesCtx** and **membersCtx**. Variable *crs* is retained in this refinement. An additional variable *prtcpts* representing information about course participants is introduced. Invariant **inv1.1** models *prtcpts* as a relation between the set of opened courses *crs* and the set of participants *PRTCPT*. Invariant **inv1.2** states that for every opened course *c*, the instructor of that course, i.e., *instrs(c)*, is not amongst its participants, represented by *prtcpts*[\{c\}].

variables : *crs*, *prtcpts*

invariants :
inv1.1 : *prtcpts* \in *crs* \leftrightarrow *PRTCPT*
inv1.2 : $\forall c \cdot c \in crs \Rightarrow instrs(c) \notin prtcpts[\{c\}]$

init
 begin
 ...
prtcpts := \emptyset
 end

Initially, there are no opened courses hence *prtcpts* is assigned to be \emptyset . The original abstract event **OpenCourses** stays unchanged in this refinement, while an additional assignment is added to **CloseCourses** to update *prtcpts* by removing the information about the set of closing courses *cs* from it.

CloseCourses
status *anticipated*
any *cs* **where**
 ...
then
 ...
act1.2 : *prtcpts* := *cs* \triangleleft *prtcpts*
end

A new event is added, namely **Register**, to model the registration of a participant *p* for an opened course *c*. The guard of the event ensures that *p* is not the instructor of the course (**grd1.3**) and is not yet registered for the course (**grd1.4**). The action of the event update *prtcpts* accordingly by adding the mapping *c* \mapsto *p* to it.

```

Register
status convergent
any p, c where
  grd1.1 : p ∈ PRTCPT
  grd1.2 : c ∈ crs
  grd1.3 : p ≠ instrs(c)
  grd1.4 : c ↦ p ∉ prtcpts
then
  act1.1 : prtcpts := prtcpts ∪ {c ↦ p}
end

```

We attempt to prove that **Register** is convergent and **CloseCourses** is anticipated using the following variant.

```

variant : (crs × PRTCPT) \ prtcpts

```

The variant is a set of mappings, each links an opened course to a participant who has *not* registered for the respective course.

We present some of the important proof obligations for **m1**. For events **OpenCourses** and **CloseCourses**, proof obligations are trivial.

CloseCourses/inv1.2/INV This obligation is to ensure that **inv1.2** is maintained by **CloseCourses**. The obligation is trivial, in particular, given that $c \notin cs$, we have $(cs \triangleleft prtcpts)[\{c\}]$ is the same as $prtcpts[\{c\}]$.

...

Register/inv1.1/INV This obligation is to guarantee that **inv1.1** is maintained by the new event **Register**.

...

FIN This obligation is to ensure that the declared variant used for proving convergence of events is finite. This is trivial, since the set of opened course *crs* and the set of participants *PRTCPT* are both finite.

...

CloseCourses/VAR This proof obligation ensures that anticipated event **CloseCourses** does not increase the variant.

...

Register/VAR This proof obligation ensures that the convergent event **Register** decreases the variant. This is trivial since a new mapping $c \mapsto p$ is added to *prtcpts*, effectively increasing *prtcpts*, hence strictly decreasing the variant.

...

4.1 Machine m2

We perform a data refinement by replacing abstract variables *crs* and *prtcpts* by a new concrete variable *atnds*. This machine does not explicitly model any requirements: it implicitly inherits requirements from previous abstract machines. As stated in invariant *inv2_1*, *atnds* is a *partial function* from *CRS* to some set of participants (i.e., member of $\mathbb{P}(\text{PRTCPT})$). Invariants *inv2_2* and *inv2_3* act as gluing invariants, linking abstract variables *crs* and *prtcpts* with concrete variable *atnds*. Invariant *inv2_2* specifies that *crs* is the domain *atnds*. Invariant *inv2_3* states that for every opened courses *c*, the set of participants attending that course represented abstractly as *prtcpts*[[*c*]] is the same as *atnds*(*c*).

variables : *atnds*

invariants :
inv2_1 : *atnds* $\in CRS \mapsto \mathbb{P}(\text{PRTCPT})$
inv2_2 : *crs* = dom(*atnds*)
inv2_3 : $\forall c. c \in crs \Rightarrow prtcpts[\{c\}] = atnds(c)$
thm2_1 : finite(*atnds*)

init
 begin
 atnds := \emptyset
 end

We illustrate our data refinement by the following example. Assume that the available courses *CRS* are $\{c_1, c_2, c_3\}$, with *c*₁ and *c*₂ being opened, i.e., *crs* = $\{c_1, c_2\}$. Assume that *c*₁ has no participants, and *p*₁ and *p*₂ are attending *c*₂. Abstract variable *prtcpts* hence contains two mappings as follows $\{c_2 \mapsto p_1, c_2 \mapsto p_2\}$. The same information can be represented by the concrete variable *atnds* as follows $\{c_1 \mapsto \emptyset, c_2 \mapsto \{p_1, p_2\}\}$.

We refine the events using data refinement as follows. Event *OpenCourses* is refined by *OpenCourse* where one course (instead of possibly several courses) is opened at a time. The course that is opening is represented by the concrete parameter *c*.

OpenCourses
 when
 grd0_1 : card(*crs*) $\neq m$
 thm0_1 : *crs* $\neq CRS$
 then
 act0_1 : *crs* :| *crs* $\subset crs' \wedge \text{card}(crs') \leq m$
 end

OpenCourse
 refines *OpenCourses*
 any *c* where
 grd2_1 : *c* $\notin \text{dom}(atnds)$
 grd2_2 : card(*atnds*) $\neq m$
 with
 crs' = *crs* $\cup \{c\}$
 then
 act2_1 : *atnds*(*c*) := \emptyset
 end

The concrete guards ensure that *c* is a closed course and the number of opened course (card(*atnds*)) has not reached the limit *m*. The action of *OpenCourse* sets the initial participants for the newly opened course *c* to be the empty set. In order to prove the refinement relationship between *OpenCourse* and *OpenCourses*, we need to give the witness for the after value of the disappearing variable *crs'*. In this case, it is specified as *crs'* = *crs* $\cup \{c\}$, i.e., adding the newly opened course *c* to the original set of opened courses *crs*.

Abstract event *CloseCourses* is refined by concrete event *CloseCourse*, where one course *c* (instead of possibly several courses *cs*) is closed at a time. The guard and action of concrete event *CloseCourse* are as expected.

```

CloseCourses
status anticipated
any cs where
  grd0.1 : cs ⊆ crs
  grd0.2 : cs ≠ ∅
then
  act0.1 : crs := crs \ cs
  act2 : prtcpts := cs ≪ prtcpts
end

```

```

CloseCourse
status convergent
refines CloseCourses
any c where
  grd2.1 : c ∈ dom(atnds)
with
  cs = {c}
then
  act2.1 : atnds := {c} ≪ atnds
end

```

We need to give the witness for the disappearing abstract parameter cs . It is specified straightforwardly as $cs = \{c\}$. Notice also that we change the convergence status of **CloseCourse** from *anticipated* to *convergent*. We use the following variant to prove that **CloseCourse** is convergent.

```

variant : card(atnds)

```

The variant represents the number of mappings in $atnds$, and since it is a partial function, it is also the same as the number of elements in its domain, i.e., $\text{card}(atnds) = \text{card}(\text{dom}(atnds))$. As a result, the variant represent the number of opened courses.

Event **Register** is refined as follows¹, such that references to crs and $prtcpts$ in guard and action are replaced by references to $atnds$.

```

(abs_)Register
any p, c where
  grd1.1 : p ∈ PRTCPT
  grd1.2 : c ∈ crs
  grd1.3 : p ≠ instrs(c)
  grd1.4 : c ↦ p ∉ prtcpts
then
  act1.1 : prtcpts := prtcpts ∪ {c ↦ p}
end

```

```

(cnc_)Register
refines Register
any p, c where
  grd2.1 : p ∈ PRTCPT
  grd2.2 : c ∈ dom(attendees)
  grd2.3 : p ≠ instrs(c)
  grd2.4 : p ∉ atnds(c)
then
  act2.1 : atnds(c) := atnds(c) ∪ {p}
end

```

We now show some proof obligations for proving the refinement of **m1** by **m2**.

OpenCourse/act0.1/SIM This proof obligation ensures that the action **act0.1** of abstract event **OpenCourses** can simulate the action of concrete event **OpenCourse**. Notice the use of the witness for crs' as a hypothesis in the obligation.

...

CloseCourse/grd0.1/GRD This proof obligation ensures that the guard of concrete event **CloseCourse** is stronger than the abstract guard **grd0.1** of abstract event **CloseCourses**. Note the use of the witness for cs as a hypothesis in the obligation.

...

CloseCourse/NAT This proof obligation ensures that the variant is a natural number when **CloseCourse** is enabled.

...

¹We use prefixes **(abs_)** and **(cnc_)** to denote the abstract and concrete version of the event, accordingly.

CloseCourse/VAR This proof obligation ensures that the variant is strictly decreased by **CloseCourse**. The obligation is trivial since the variant represents the number of opened courses and **CloseCourse** closes one of them.

...