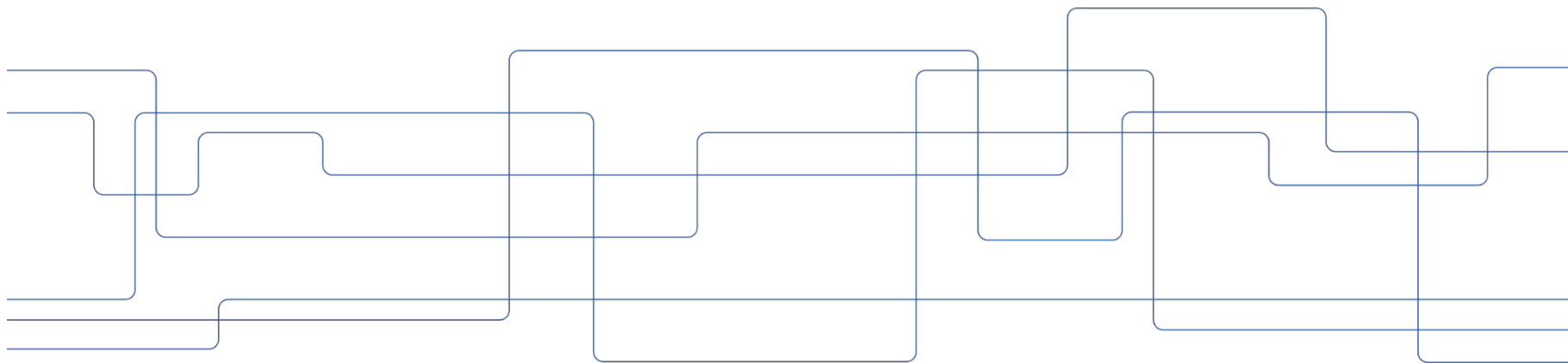




DD2460 Lecture 4.

Basics of modelling in Event-B

Elena Troubitsyna



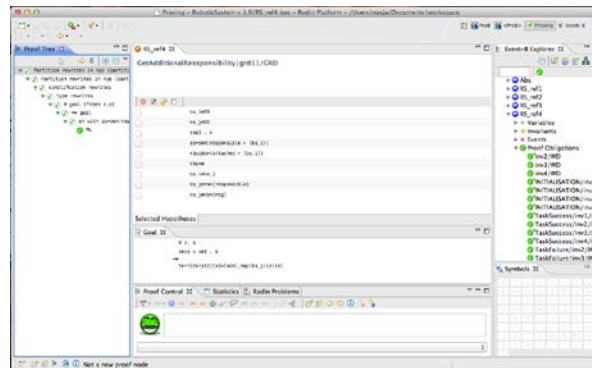
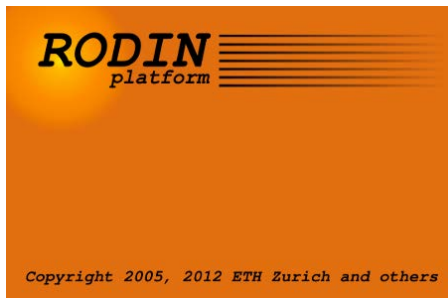


Lecture outline

- Structure of Event-B specification (project)

Event-B

- Event-B is a **formal framework** for modelling and verification of reactive systems
 - abstraction, decomposition and refinement to cope with model complexity
 - proof-based verification of functional correctness
 - requirements traceability
 - mature tool support (Rodin)





Development by refinement

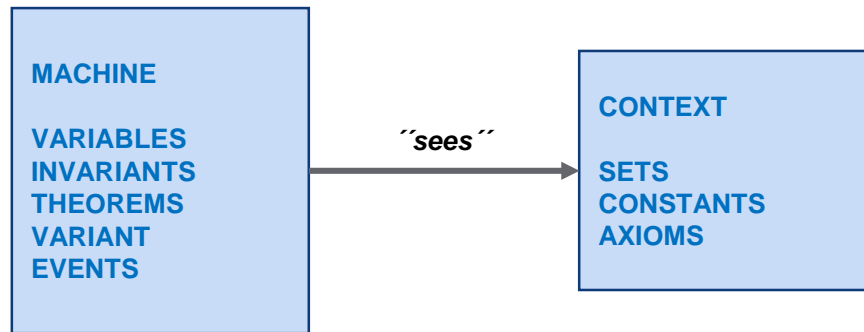
- Model-based approaches that support *correct-by-construction* system development **by refinement**
 - Idea of formal development by refinement:
 - To start with a very *abstract* model and gradually introduce system details by a number of correctness preserving steps called *refinements*
 - Each development (*refinement*) step is *proved to be correct* with respect to a more abstract specification
 - Abstract model is concise and simple, hence it can be fully understood and analysed
 - Complexity of the model increases gradually in a controlled way
-

Structure of Event-B model (specification)

- An Event-B **model** is made of several components.

A specification consists of

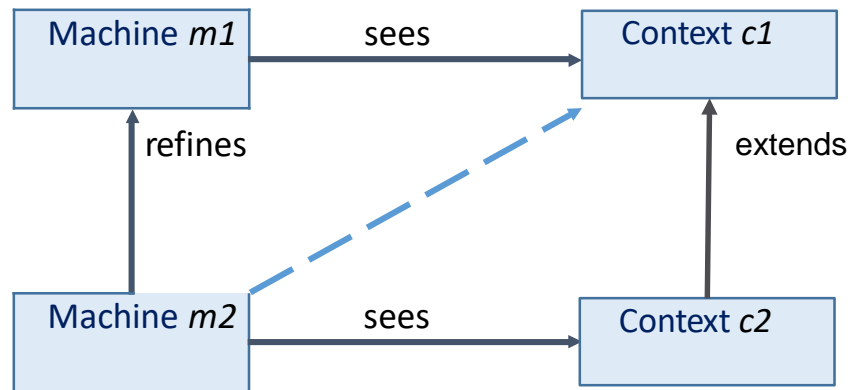
- a *static* part, specified in a [context](#), and
- a *dynamic* part, specified in a [machine](#).





Relationship between machines and contexts

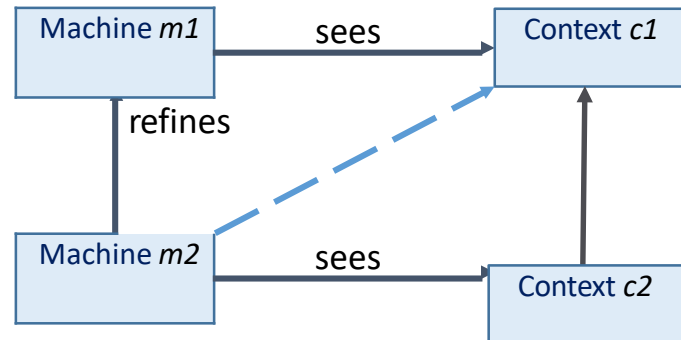
- **Contexts** contain the static structure of a discrete system (constants and axioms)
- **Machines** contain the dynamic structure of a discrete system (variables, invariants, and events)



- *Machines see contexts*
- *Contexts can be extended*
- *Machines can be refined*

Visibility Rules

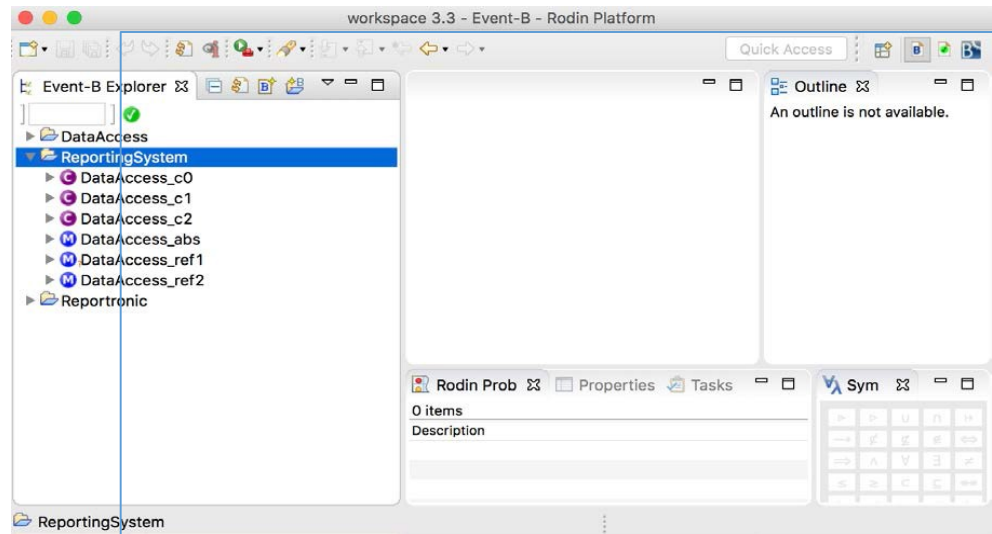
- A machine can see several contexts (or no context at all).
- A context may extend several contexts (or no context at all).
- A machine implicitly sees all contexts extended by a seen context.
- A machine only sees a context either explicitly or implicitly.
- A machine only refines at most one other machine.
- No cycle in the "refines" or "extends" relationships.





An Event-B project

- A **project** contains the complete mathematical development.
- Contains two kinds of components: **Contexts** and **Machines**.
- Projects and components are listed in the tool





Context part

- A context with name **Context1** has the following form:

CONTEXT Context1

SETS ⟨list of carrier sets⟩ **CONSTANTS**

⟨list of constants⟩ **AXIOMS** ⟨list of
labelled axioms⟩ **THEOREMS*** ⟨list of
labelled theorems⟩

END

- the context contains the static part of a model
- the context defines sets, constants that can be used in several different machines
- different properties for the sets and constants are given in **axioms-clause**

Context part

- A context with name **Context1** has the following form:

CONTEXT Context1

SETS ⟨list of carrier sets⟩

CONSTANTS ⟨list of constants⟩

AXIOMS ⟨list of labelled axioms⟩

THEOREMS* ⟨list of labelled theorems⟩

END

- A context has a unique name
- **sets-clause** contains the non-empty carrier sets
- **constants-clause** contains constants
 - they can be read but not assigned values
- **axioms-clause** lists the predicates that should hold for the constants
 - Defines types and logical properties
 - Hypotheses in all proof obligations
- **theorems-clause** lists the theorems
 - They have to be proved within the context



Context part

- A context with name **Context1** has the following form:

```
CONTEXT Context1 EXTENDS Context0 SETS ⟨list  
    of carrier sets⟩  
  
CONSTANTS ⟨list of constants⟩  
  
AXIOMS ⟨list of labelled axioms⟩  
  
THEOREMS* ⟨list of labelled theorems⟩  
  
END
```

- A context can extend another context
- **extends-clause** defines it



Example on context

```
CONTEXT UniversityContext  
  SETS STUDENTS ...  
  CONSTANTS max_course_capacity  
  AXIOMS  
    axm1: max_course_capacity  $\in \mathbb{Z}$   
    axm2: max_course_capacity  $\leq 30$   
    ...  
END
```



Machine part

- A machine with name **Machine1** has the following form:

MACHINE Machine1

SEES* ⟨list of context names⟩

VARIABLES ⟨list of variables⟩

INVARIANTS ⟨list of labelled invariants⟩

EVENTS ⟨list of events⟩

END

- A machine defines the *dynamic behaviour* of a model through events that are guarded by and act on the variables.
- **Machine-clause** gives the name of the machine
- **Sees-clause** lists the contexts that the machine can see
- **Variables-clause** gives state of the module that can be modified locally in the machine



Machine part

- A machine with name **Machine1** has the following form:

MACHINE Machine1

SEES* ⟨list of context names⟩

VARIABLES ⟨list of variables⟩

INVARIANTS ⟨list of labelled invariants⟩

EVENTS ⟨list of events⟩

END

- **Invariants-clause** lists the predicates that must hold for the variables and gluing invariants
 - The types of the variables
 - Restriction and relations between variables
- **Event-clause** contains the relevant events that change the state of the machine while preserving the invariant
 - An event describes the relationships between the state before the event takes place and just afterwards
 - Event **INITIALISATION** gives initial values to the variables and establishes the invariants



Remark on difference between model and a program

- Machines should not be thought of as programs
 - although they might be implemented by software.
- The machine models a state and the events represent behaviour that could occur
 - the conditions that must apply if an event is to fire; and
 - the effect the event has on the state.
- All communication occurs through the state.
 - machine gives a representation of possible behaviours of some system. The system might contain non-software components.



Example of a machine

MACHINE *RegistrationSystem*

SEES *UniversityContext*

VARIABLES registered enrolled

INVARIANTS

inv1: registered \subseteq STUDENTS

inv2: enrolled \subseteq STUDENTS

inv3: enrolled \subseteq registered

...

EVENTS \langle list of events \rangle

END

- Registered students form a subset of abstract set STUDENTS defined in CONTEXT
- inv3: only registered students can be enrolled



An Event-B model

- A model can contain:
 - Only contexts (represents a pure mathematical structure)
 - Only machines (the model is not parametrised)
 - Both machines and contexts

- All machines and context identifiers must be distinct in the same model (project)



Event-B events

MACHINE Machine1

SEES* ⟨list of context names⟩

VARIABLES ⟨list of variables⟩

INVARIANTS ⟨list of labelled invariants⟩

EVENTS ⟨list of events⟩

END

- All events represent transitions
- The events modify the state of the system (values of variables)
 - An event is a state transition in a discrete dynamic system.
- They are executed in one (atomic) step
 - Only one event fires at a time.
- An event essentially consists of guards and actions.

Hence they are said to be **guarded** events.

 - **Event = *guard + action***



Guarded events

Event = *guard* + *action*

- An event essentially consists of guards and actions.
 - the guards define the necessary conditions for the event to be enabled
 - the actions define the way the variables of the machine are modified

- A guarded event can be executed only when its guard is true
 - if more than one guard is true, one of them is non-deterministically chosen
 - if no guards are true, the specification will terminate.

Event structure

```
Event1 =                                // event name
  any
    x1 x2                                // event parameters
  where
    G1                                  // event guards
    G2
    ...
  then
    v1 := exp1 // event actions
    v2 := exp2
    ...
end
```

- Events are identified by unique names
- **any-clause** lists the parameters (or local variables) of the event
- **where-clause** contains the guards of the event, i.e., the conditions for the event to be enabled
- **then-clause** lists the actions of the event



Events: general form

- A machine can contain arbitrary many events
- An event can have one of the following forms:

Event1 =

any

x1, x2

where

G1

G2

...

then

v1 := exp1

v2 := exp2

...

end

Event2 =

when

G1

G2

...

then

v1 := exp1

v2 := exp2

...

end

Event3 =

begin

v1 := exp1

v2 := exp2

...

end



Event guards

- A guards is a predicate that specifies enabling conditions under which events may occur
- Example, booking a room for the lecture
 - grd1:** $lec \in LECTURERS$
 - grd2:** $lh \in LHALL$
- Guards should be strong enough to ensure invariants are maintained by the actions of an event but not too strong that they prevent desirable behaviour



Event actions

- An action describes the ways one or several state variables are modified by the occurrence of an event
- An action might be either deterministic or non-deterministic
- There are three principle constructions — that Event B calls **substitutions** — for changing the state of a machine:

- $x := e$ // x becomes equal to the value of e
This rule may be used recursively to assign to any number of variables.
- $x: | P$ // x becomes such that it satisfies the **before-after** predicate P
- $x: \in S$ // x becomes in the set S



Event actions cnt.

$x := e$ and $x: | P$ can be extended to *multiple assignment*:

$x, y := e1, e2$ and $x, y: | P$,

and recursively to many variables.

- The variables must be distinct! ~~$x, x, z := e1, e2, e2$~~
- Note: all assignments can be written in the form: $x, y: | P$



Deterministic actions

- Here is the form of some deterministic actions on variables x , y and z :

Event example

...

Then

act1: $x := x + y$

act2: $y := y + x - z$

...

- Notice that variables x and y should be distinct.
- Actions are supposed to be “performed” in parallel.
- Variables x and y are assigned to $x + y$ and $y + x - z$, respectively.
- Variable z is used but not modified by these actions



Non-deterministic actions

- A non-deterministic actions of the form

$$\bullet x : | P$$

// *x becomes such that it satisfies the before-after predicate P*

- The before-after-predicate gives the condition that holds just before the action takes place. It may contain all variables of the machine.

Non-deterministic actions (example)

$$\bullet x, y : | \ x' > x \wedge y' < x'$$

- On the LHS of operator $:|$, we have two distinct variables
- On the RHS, we have a *before-after predicate*
- The RHS contains occurrences of x and y (before values) and primed occurrences x' and y' (after values)
- As a result (in this example):
 - x is assigned a value greater than its previous value
 - y is assigned a value smaller than that, x' , assigned to x

Non-deterministic action (cont.)

$$x : \in S$$

- The variable is assigned an arbitrary element of the set S .
- This form is a special form of the previous one.

- Example 1:

$$\text{act1: } x : \in \{x + 1, y - 2, z + 3\}$$

Here x is assigned any value from the set $\{x + 1, y - 2, z + 3\}$

- Example 2:

$$\text{act2: } st : \in STUDENTS$$

Here st is assigned any value from the set $STUDENTS$



Non-deterministic action (another example)

- **act1:** $x : \in \{a \mid 0 < a \wedge a < 50\}$

Here x is assigned any arbitrary number between 1 and 49.



Simple example: Coffee club

- Lets create a model of a coffee club.
- For the coffee club we require a moneybank that stores money used by the coffee club.
- **REQ1:** a money bank for storing and reclaiming finite, non-negative funds for a coffee club;
- **REQ2:** an operation for adding money to the money bank;
- **REQ3:** an operation for removing money from the money bank; cannot remove more than money bank contains.



Example cnt: moneybank variable

MACHINE CoffeeClub

VARIABLES *moneybank*

// The machine state is represented by the variable, *moneybank*,
denoting the money bank for the coffee club.

INVARIANTS

inv1: *moneybank* $\in \mathbb{N}$ // **REQ1:** *moneybank* must not be negative.

// here \in set membership

// \mathbb{N} - the set of natural numbers

\mathbb{N} NAT the set of natural numbers = non-negative integers



Model events

...

EVENTS

INITIALISATION \triangleq

then

act1: *moneybank* := 0 // we initialise *moneybank* to 0

end

FEEDBACK \triangleq // REQ2: adding to *moneybank*

any *amount*

where

grd1: *amount* $\in \mathbb{N}_1$ // \mathbb{N}_1 is used rather than \mathbb{N} to prevent the event firing
uselessly if *amount* = 0

then

act1: *moneybank* := *moneybank* + *amount*

end



Example cnt.: Model events (cont.)

```
...  
ROBBANK  $\triangleq$  // REQ3: removing money from moneybank  
  any amount  
  where  
    grd1: amount  $\in$  1 .. moneybank // The amount must not exceed the contents of money-  
                                         bank and we don't need to uselessly remove an amount of 0  
  then  
    act1: moneybank := moneybank - amount  
  end  
END
```



CoffeeClub model

Machine CoffeeClub

Variables *moneybank*

inv1: *moneybank* $\in \mathbb{N}$

Events

INITIALISATION \triangleq

then

act1: *moneybank* := 0

end

FEEDBANK \triangleq

any *amount*

where

grd1: *amount* $\in \mathbb{N}_1$

then

act1: *moneybank* := *moneybank* +

amount

end

ROBBANK \triangleq

any *amount*

where

grd1: *amount* $\in 1 \dots \text{moneybank}$

then

act1: *moneybank* := *moneybank* - *amount*

end

END



Coffee club model in Rodin Editor

MACHINE

CoffeeClub

VARIABLES

moneybank

INVARIANTS

inv1 : moneybank $\in \mathbb{N}$

EVENTS

INITIALISATION \triangleq

STATUS

ordinary

BEGIN

act1 : moneybank $\models 0$

END

FeedBank \triangleq

STATUS

ordinary

ANY

amount

WHERE

grd1 : amount ≥ 0

THEN

act1 : moneybank \models moneybank + amount

END

RobBank \triangleq

STATUS

ordinary

ANY

amount

WHERE

grd1 : amount $\in 1..moneybank$

THEN

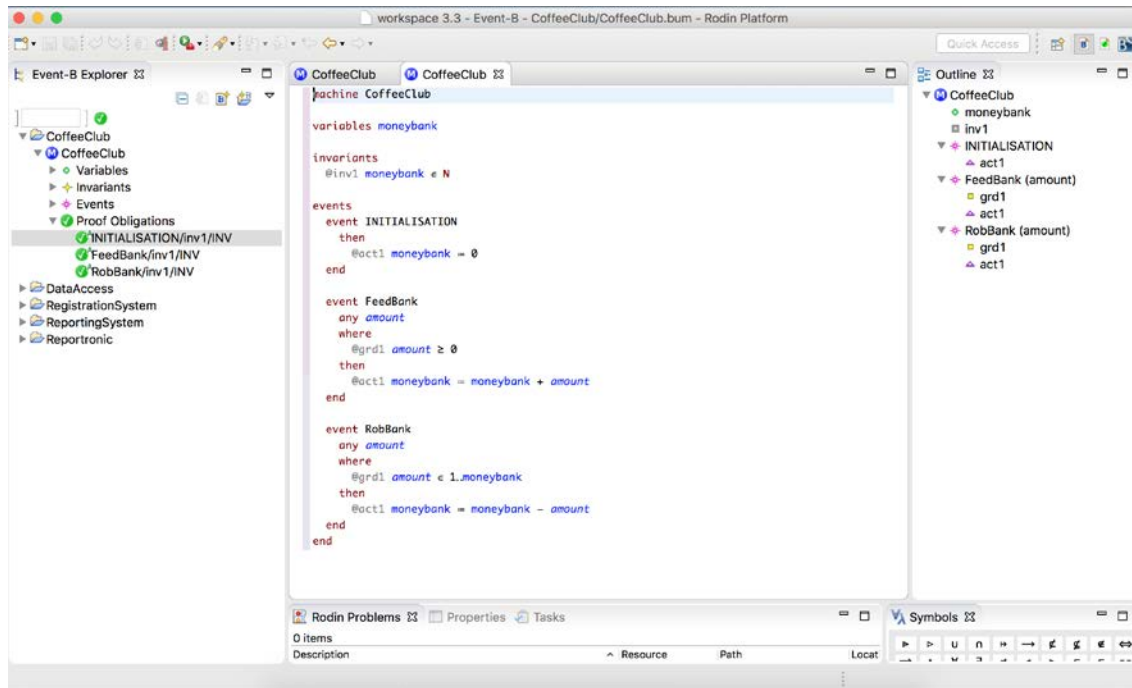
act1 : moneybank \models moneybank - amount

END

END



Demo of Rodin platform





workspace 3.3 - Event-B - CoffeeClub/CoffeeClub.bum - Rodin Platform

Event-B Explorer

- CoffeeClub
 - Variables
 - Invariants
 - Events
 - Proof Obligations
 - INITIALISATION/inv1/INV
 - FeedBank/inv1/INV
 - RobBank/inv1/INV
 - DataAccess
 - RegistrationSystem
 - ReportingSystem
 - Reportronic

CoffeeClub

MACHINE

CoffeeClub

VARIABLES

moneybank

INVARIANTS

inv1 : moneybank \in N

EVENTS

INITIALISATION \triangleq

STATUS

ordinary

BEGIN

act1 : moneybank = 0

END

FeedBank \triangleq

STATUS

ordinary

ANY

amount

WHERE

grd1 : amount \geq 0

THEN

act1 : moneybank = moneybank + amount

END

Pretty Print Edit Synthesis Dependencies

Rodin Problems Properties Tasks

0 items

Description	Resource	Path	Locat
-------------	----------	------	-------

Outline

- moneybank
- inv1
- INITIALISATION
- FeedBank
- RobBank

Symbols

0 items

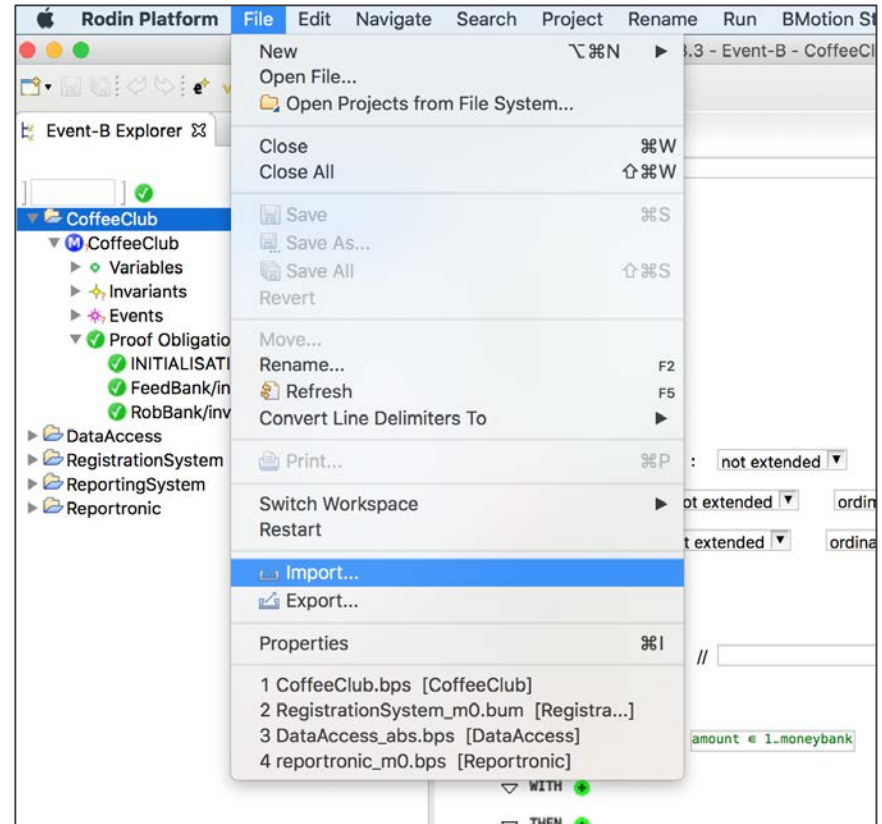
Navigation icons: back, forward, search, etc.

1 item selected



Model importing

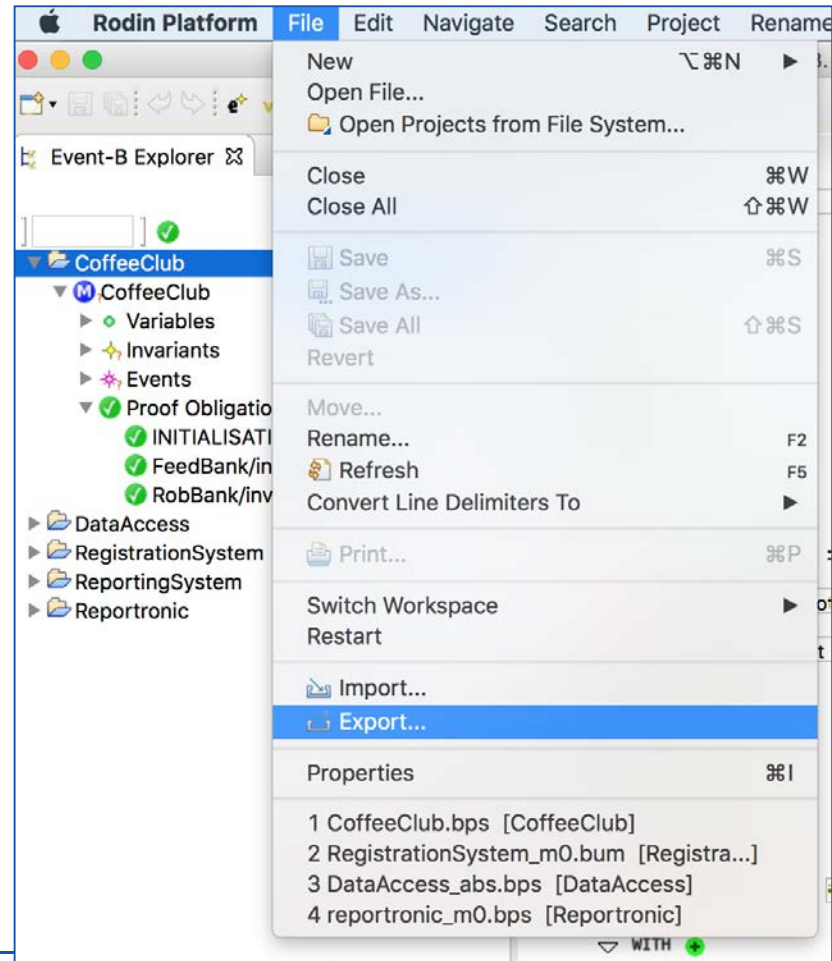
- A “.zip” file corresponding to a project which has been exported elsewhere can be imported locally.
- **File > Import** from the menubar
- In the import wizard select **General > Existing Projects into Workspace** and click **Next**.
- Then choose the **Select archive file** option and hit the **Browse...** button to find the zip file that you want to import.
- Now click **Finish**.






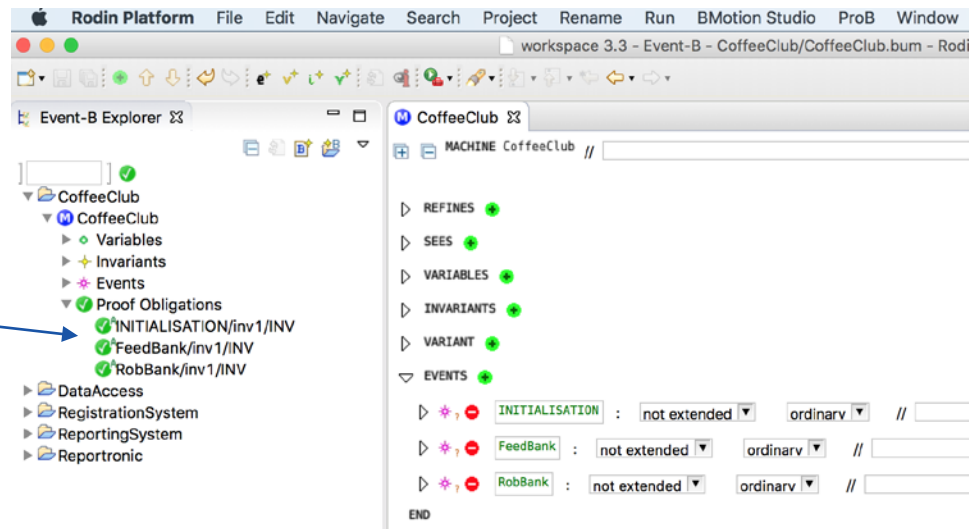
Model exporting

- Exporting a project is the operation by which you can construct automatically a ".zip" file containing the entire project.
- It then becomes a project like the other ones which were created locally.
- In order to export a project, select it and then select on **File > Export...** from the menubar.
- The **Export** wizard will pop up. In this window, select **General, Archive File** and click the **Next** button.
- Specify the path and name of the archive file into which you want to export your project and finally select **Finish**.



Proof Obligations for CoffeeClub

- CoffeeClub is a very simple model and the POs are correspondingly simple.
- As a consequence the POs are easily discharged automatically by the provers in the Rodin tool.
- The following POs are generated for the above machine.

- Notice that all POs concerned with maintenance of INV 1 are verifying that REQ1 is satisfied.





The semantic of the events

- In order to formally be able to control that the events do, what they are supposed to do, we have to give them as exact mathematical **semantics**
 - The events change the local state (the variables) of a specification.
 - An event describes the relationship between the before-state and the after-state.
-



The state of a specification

- The state of a specification (an Event-B machine) can be a combination of all possible values of its variables
- The state of the specification can be modelled as the value of the cartesian product of these variable types
 - For example if we have 2 variables of type natural numbers, then the state is $N \times N$ (all possible pairs of natural numbers)
- The events of the specification change the state

Different kinds of state changes

- Deterministic
 - **1-1** relationship between initial and final states – guaranteed to reach the final state
 - Ex. ... **then** *moneybank* := *moneybank* + *amount* **end**
 - Non-deterministic
 - **1-n** relationship
 - may reach different final states – Ex. ... **then** $n \in 1..5$ **end**
 - Non-executable
 - in “waiting mode”
 - Ex. **when** $n > 2$ **then** $n := n + 1$ **end**
 - Non-terminating
 - **1-0** relationship
 - guaranteed not to reach a final state
-



Substitution

- Substitution is a central concept in Event-B
 - Substitution refers to substituting a free variable with an expression
 - Substitution of variable x in predicate P with expression E is denoted: $P[E/x]$
 - A variable in an expression is free, if it is not bound by a quantifier
 - The variables in E should not become bound after the substitution
 - This can be generalised to multiple substitution $P[E_1, \dots, E_n / x_1, \dots, x_n]$
-

Before-after-predicate

- Every event can be associated with a ***before-after predicate***
- The predicate gives the relationship between the values of the variable right before (**n**) and right after (**n'**) the event has occurred
- Example, the events

FEEDBANK \triangleq
 $moneybank := moneybank + amount$

RODBANK \triangleq
 $moneybank := moneybank - amount$

- correspond to the following before-after-predicates

$moneybank' = moneybank + amount$

$moneybank' = moneybank - amount$



Before-after predicates

- Before-after predicates (BA) for events can be calculated based on the following:
 - $BA(\text{any } v \text{ where } G \text{ then } S \text{ end}) = \exists v. G \wedge BA(S)$
 - $BA(\text{when } G \text{ then } S \text{ end}) = G \wedge BA(S)$
 - $BA(\text{begin } S \text{ end}) = BA(S)$

BA calculations for statements give:

- $BA(v := E) = (v' = E)$
 - $BA(v \in Q) = (v' \in Q)$
 - $BA(v :| P(v', v)) = P(v', v)$
-



Consistency of Contexts

- In order to be consistent the Event-B context must satisfy the following properties:
 1. All its axioms must be well defined (**axm/WD**)
 2. All its theorems must be well defined (**thm/WD**)
 3. All its theorems must be proved (**thm/THM**)
-



Consistency of a Machine

In order to be consistent a machine must satisfy the following conditions:

1. All its invariants and theorems must be well defined (**inv/WD** and **thm/WD**)
 2. All its event guards and actions must be well defined (**grd/WD** and **act/WD**)
 3. All its nondeterministic events must be feasible (**evt/act/FIS**)
 4. All its theorems must be proved (**thm/THM**)
 5. All invariants must be established by the initialisation (**INIT/inv/INV**)
 6. All invariants must be preserved by all events (**evt/inv/INV**)
-



Verification of model correctness and safety

- Recall that we discussed that safety requirements are typically defined as “Always”
- If the specification contains the invariant defining safety requirements then by verifying correctness of the specification, we at the same time verify safety
- Proofs allows us to investigate all possible execution traces
- Observe: we do not need to explicitly define all possible states (infinite number) but still are able to proof the desired properties
- But if the specification does not contain any useful invariants (e.g., only types of variables) then modelling exercise becomes pointless



Wrap-up

- We learn about the structure of Event-B specification
- Investigated what is the CONTEXT and MACHINE
- Studied the structure of events
- Learnt about invariants
- Learnt about the verification of Event-B models
- Established connection between safety requirements, invariants and proofs

Questions?

