Lecture 5

PROOF OBLIGATIONS, EVENT-B REFINEMENT

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 x 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* :∈ *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[E1, ..., En / x1, ..., xn]**

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 occured
- Example, the events

FEEDBANK ≜
moneybank := moneybank + amount

ROBBANK ≜
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(**any** v **where** G **then** S **end**)= ∃v. G∧ BA(S)
- BA(**when** G **then** S **end**) = G ∧ BA(S)
- BA(**begin** S **end**) = BA(S)

BA calculations for statements give:

- BA(v:=E)=(v’=E)
- BA(v:∈Q)=(v’∈Q)
- BA(v :| P(v’,v)) = P(v’,v)

Traces

- Execution of the events in the machine gives rise to a trace of states
- The relationship between two successive states is given by the before-after relationship of the event executed between the states
- Example of traces:
  - {*moneybank* =0, *moneybank* =1, *moneybank* =0, *moneybank* =1, ...}
  - {*moneybank* =0, *moneybank* =1, *moneybank* =2, *moneybank* =3, ...}

Verification

- Intuitively, we mean that a model is correct if every state in all traces satisfies the invariant
  - The events in the model cannot reach a state that violates the invariant
  - Model-checkers can check that this is the case for finite states spaces (that are sufficiently small)
- For proofs we use a stronger condition:
  - The invariant is assumed to hold before execution of an event and we prove that it holds afterwards (Invariant preservation)
  - The initialisation is proved to establish the invariant

Proofs

- Verification of Event-B models is mainly done with proofs
  - Proof Obligations (PO:s) (predicates) are generated for the models
  - Show that a predicate is true for all values on the variables
  - Automatic proof tools in the Rodin platform are based on the sequent calculus

On correctness of specifications

- Development in Event B relies on mathematical (logical) proofs
- In predicate logic we have:

FALSE ⇒ any statement

anything can be proved from false assumptions

- Hence, a contradictive specification can be implemented by any program
  - if the invariant evaluates to FALSE (it is unsatisfiable) then anything can be proved;
  - the Event-B specification is trivially ”correct”.
- In order to prohibit this, Event-B forces the developers to check that the first specification is correct and consistent.

Conditions for correct specification

In order to check the correctness of a specification, Event-B generates the following proof obligations

- **Conditions for the context**
  - checks that the axioms and theorems are well defined
- **Conditions for the invariant**
  - checks that the invariant and theorems are well defined
- **Conditions for the initialisation**
  - proves that the assignment statements in the initialisation establish a state that satisfies the invariant(s)
- **Conditions for the events**
  - proves that each event preserves the invariant(s)

Proof Obligations: Sequent representation

- Sequent representation of PO:

hypotheses

$\vdash$

goal

- The truth of the hypotheses leads to the truth of the goal.
- The symbol $\vdash$ is sometimes called *stile* or *turnstile*.
- If any of the hypotheses is $\bot$(FALSE) then any goal is trivially established.
- If the hypotheses are identically $\top$ (TRUE) then the hypotheses will be omitted.

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**)

The Rodin tool generates proof obligations to check that all expressions are well defined.

Consistency of a Machine

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

1. All it 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**)

Feasibility of non-determinism

- *Feasibility* states that the action of an event is always feasible whenever the event is enabled.
- In other words, there are always possible to find “after” values for the variables to satisfy the before-after predicate, e.g., before after predicate cannot be $x'>0 \wedge x'<0$

The feasibility proof obligation **evt/act/FIS** is as follows:

Axioms and theorems

Invariants and theorems

Guards of the event

$\vdash$

$\exists v'$. Before-after predicate

Invariant preservation

- An essential feature of an Event-B machine M is its invariants:
  - They show properties that hold in every reachable state of the machine.
- The invariant preservation proof obligation **evt/inv/INV** is as follows:

Axioms and theorems
Invariants and theorems
Guards of the event
Before-after predicate of the event
$\vdash$
Specific invariant with updated values

Invariant preservation: example

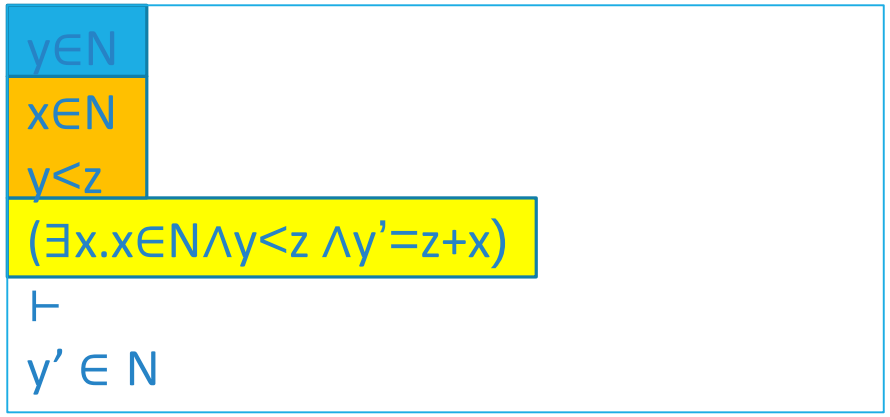
- Assume that we have an event **EVENT** and an invariant $y \in N$.

The event is given as

EVENT =
any x
where
$x \in N$
$y < z$
then $y := z+x$
end

and the invariant is given as $y \in N$

- BA(**EVENT**) = ∃v. Guards ∧ BA(action)



Invariant establishment

- *Invariant establishment* states that any possible state after initialisation given by the after predicate must satisfy the invariant *I*.
- The invariant establishment proof obligation **INIT/inv/INV** is as follows:

Axioms and theorems
Guards of the event
Before-after predicate of the event
⊢
Specific invariant with updated values

Contradicting events

An event can be contradicting (its correctness cannot be proved), if

- its guard is too weak
- the guard or the action is incorrect
- the invariant is too strong (too precise) (states properties that are not maintained by the events)
- the invariant is too weak (does not give enough assumptions for the proof to succeed)
- the invariant simply is wrong

Consistency of a Machine

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

1. All it 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**)
7. **Deadlock freeness (DLF)**

Deadlock freeness

- For non-terminating systems we need to guarantee that there is always an enabled event, i.e. that the system is deadlock free
- The deadlock freeness proof obligation **DLF** is as follows:

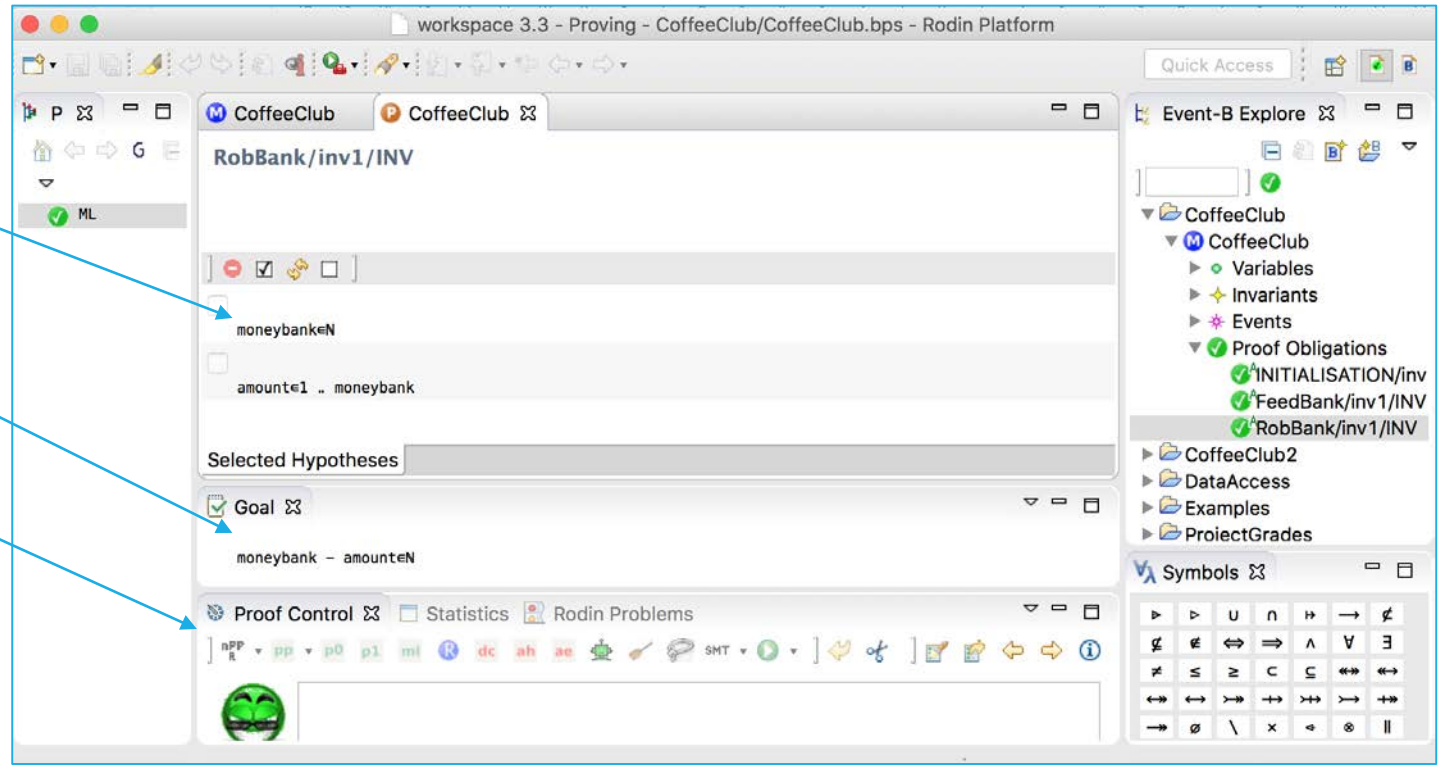
Axioms and theorems
Invariants and theorems
$\vdash$
Disjunction of the guards of the events

Proof in Rodin

When discharging a proof in Rodin, three views are available:

- hypotheses,
- goal and
- proof control that provides

access to a range of provers and also tactics for manual proof.

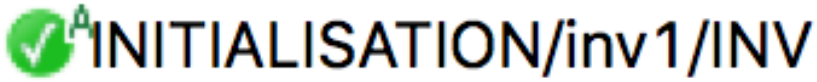


Labelling of proof obligations

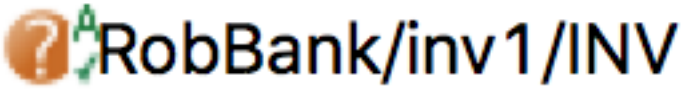
- Proof obligations are labelled by Rodin as follows:

event-name/predicate id/proof type id

- For example:



Green color indicates - a proof obligation to show that the **INITIALISATION** event satisfies the invariant labelled with inv1.



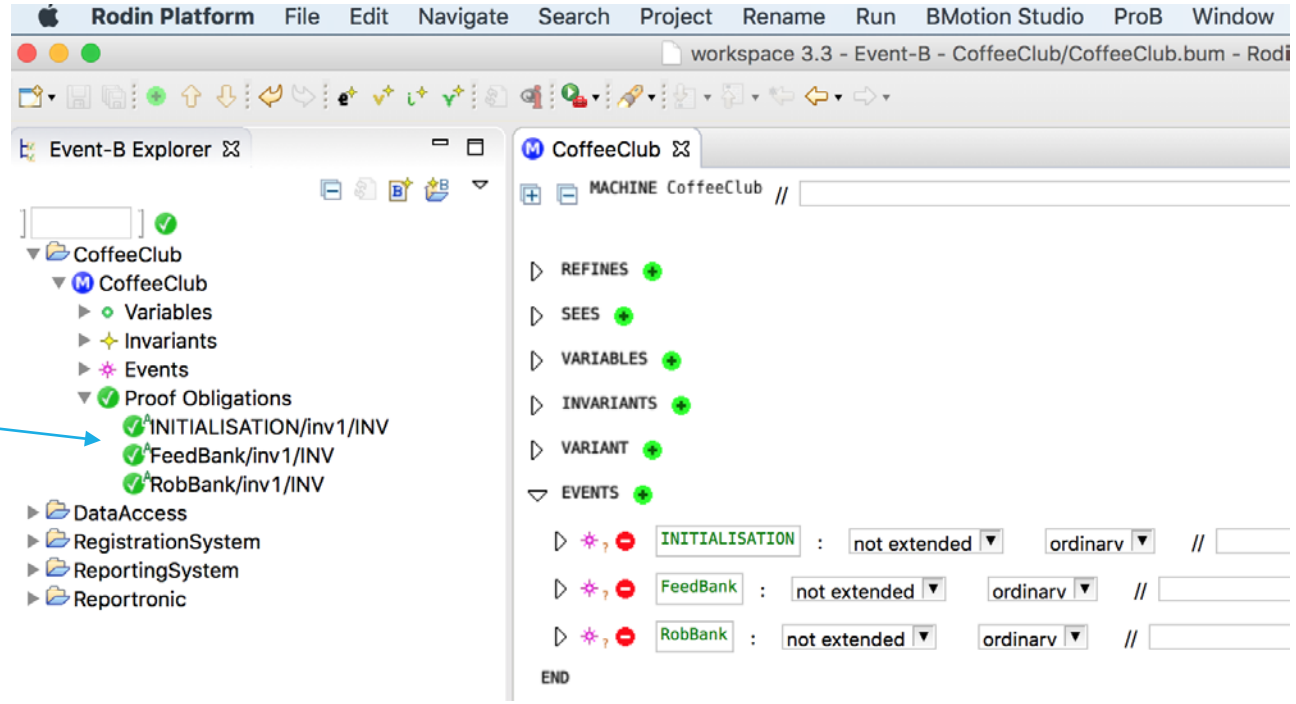
Red (brown) color indicates - a proof obligation to show that the **RobBank** event does not satisfy the invariant labelled with inv1.

CoffeeClub model (from Lecture 3)

MACHINE **CoffeeClub**
VARIABLES *moneybank*
inv1: *moneybank* ∈ ℕ
EVENTS
INITIALISATION ≙
then
act1: *moneybank*:= 0
end
FEEDBANK ≙
any *amount*
where
grd1: *amount* ∈ $\mathbb{N}_1$
then
act1: *moneybank* := *moneybank* + *amount*
end
ROBBANK ≙
any *amount*
where
grd1: *amount* ∈ *1* .. *moneybank*
then
act1: *moneybank* := *moneybank* - *amount*
end
END

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.



Proving prove obligations

INITIALISATION/inv1/INV:

$\top$

$\vdash$

$0 \in \mathbb{N}$

Invariants
inv1: *moneybank* $\in \mathbb{N}$
…
INITIALISATION $\triangleq$
then
act1: *moneybank*:= 0
end

Assuming nothing, ($\top$ or true), show that 0 is an element of the set of natural numbers. Clearly, it is true.

Verifying that the **INITIALISATION**, moneybank := 0, establishes the invariant *moneybank* $\in \mathbb{N}$.

Proving prove obligations

FEEDBANK/inv1/INV:

$moneybank \in \mathbb{N}$

$amount \in \mathbb{N}_1$

$\vdash$

$moneybank + amount \in \mathbb{N}$

Invariants
inv1: $moneybank \in \mathbb{N}$
…
FEEDBANK ≙
any *amount*
where
grd1: $amount \in \mathbb{N}_1$
then
act1: *moneybank := moneybank + amount*
end

Verifying that the actions of **FEEDBANK**, *moneybank := moneybank + amount*, maintains the invariant, $moneybank \in \mathbb{N}$.
This is clearly true as both *moneybank* and *amount* are natural numbers.

Proving prove obligations

ROBBANK/inv1/INV:

$moneybank \in \mathbb{N}$

$amount \in 1 .. moneybank$

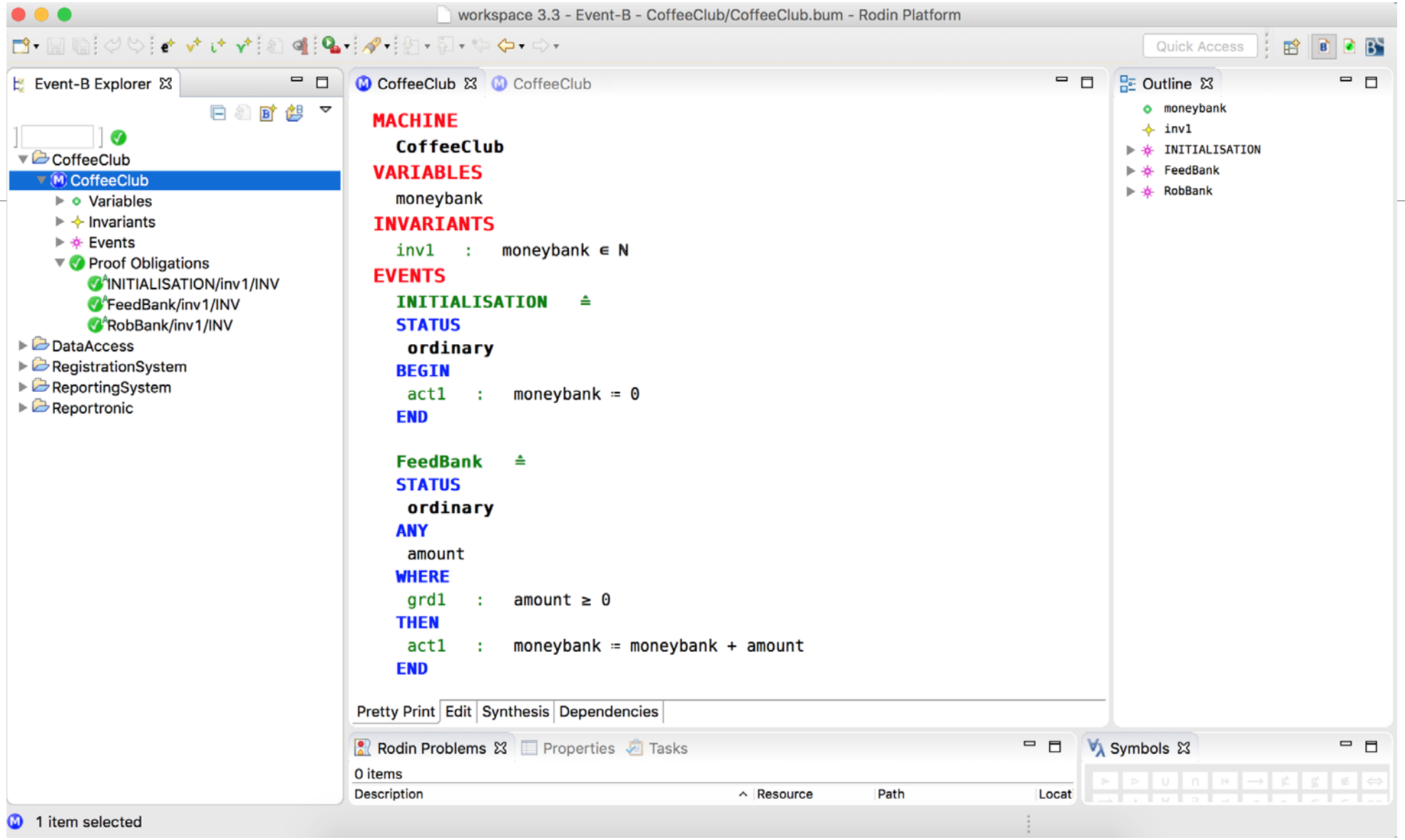
$\vdash$

$moneybank - amount \in \mathbb{N}$

Invariants
inv1: $moneybank \in \mathbb{N}$
…
RODBANK $\triangleq$
any *amount*
where
grd1: $amount \in 1 .. moneybank$
then
act1: $moneybank := moneybank - amount$
end

Similar to the preceding POs, but this time verifying that $moneybank \in N$ is maintained by the action $moneybank := moneybank - amount$.

This is not quite so simple, but the guard $amount \in 1 .. moneybank$ ensures that amount ≤ moneybank and hence the invariant is maintained.


workspace 3.3 - Event-B - CoffeeClub/CoffeeClub.bum - Rodin Platform
Quick Access
Event-B Explorer
CoffeeClub
CoffeeClub
Variables
Invariants
Events
Proof Obligations
INITIALISATION/inv1/INV
FeedBank/inv1/INV
RobBank/inv1/INV
DataAccess
RegistrationSystem
ReportingSystem
Reportronic
CoffeeClub
CoffeeClub
MACHINE
CoffeeClub
VARIABLES
moneybank
INVARIANTS
inv1 : moneybank ∈ N
EVENTS
INITIALISATION ≙
STATUS
ordinary
BEGIN
act1 : moneybank ≔ 0
END
FeedBank ≙
STATUS
ordinary
ANY
amount
WHERE
grd1 : amount ≥ 0
THEN
act1 : moneybank ≔ moneybank + amount
END
Pretty Print
Edit
Synthesis
Dependencies
Outline
moneybank
inv1
INITIALISATION
FeedBank
RobBank
Rodin Problems
Properties
Tasks
0 items
Description
Resource
Path
Locat
Symbols
1 item selected

Refinement relation

- The refinement relation relates the state of the machine *being refined* to the state of the refinement machine.
- The refinement relation is expressed explicitly or implicitly in the invariant of a refinement.

Refinement rules

Refinement requires consistency:

> *behaviour of a refined event must be acceptable behaviour of the unrefined event in the unrefined model.*

Informal examples of refinement

Examples of refinement of a car:

Case 1: Suppose you wanted to modify your car by installing a GPS navigation machine. This would be a refinement of your current car and you would expect that the rest of the car is not affected by the refinement. This refinement is an example of an **extension**, that is the rest of the car is not changed and the refinement is an addition.

Case 2: Suppose, on the other hand you wanted to change your car from right-hand drive to left-hand drive. This is clearly not an extension like the above modification. It should be a refinement in that the functionality of the car is not changed, but there certainly are significant changes to the algorithm. This would probably be an algorithmic refinement: the behaviour of the car does not change, but the steering and other controls have changed positions and changed connections to the rest of the car, so the algorithm at some level has changed.

Refinement rules

The following rules apply to refinement:

- **strengthen guards and invariants:** guards and invariants can be strengthened, provided overall functionality is not reduced;
- **nondeterminism can be reduced:** where a model offers choice, then the choice can be reduced — but not increased — in the refinement;
- **the state can be augmented by an orthogonal state:** new state variables, whose values do not affect the existing state, may be added.

Refinement

- Consistent with the refinement rules, a single event may be refined by multiple events, or conversely, multiple events may be refined by a single event.
- *New events:* As well as refinements of the events of the refined machine, the refinement may introduce new events.
  - The new events must NOT change variables inherited from the state of the refined machine. This is a restriction that recognises that a machine state can be modified only by the events of that machine, or their refinements.

Refinement of the CoffeeClub model

The **CoffeeClub** (see Lecture 3) we define a ***moneybank*** that models an amount (of money), and events that describe adding to (**FEEDBANK event)** or taking from (**ROBBANK event)** the amount modelled by moneybank. We want to add new functionality.

The new requirements are:

ID	Requirements
REQ4	A facility for members to join the coffee club; each member has a distinct membership id.
REQ5	Members have an account that cannot go into debt;
REQ6	An operation that enables a member to add money to their account;
REQ7	Money added to a members account is also added to the club money bank;
REQ8	An operation that sets the price for a cup of coffee;
REQ9	An operation that enables a member to buy a cup of coffee; the member's account is reduced by the cost of a cup of coffee;
REQ10	Money reduced from a members account is also deleted from the club money bank;

Refinement of the CoffeeClub model

We will introduce variables ***members***, ***accounts*** and ***coffeeprice*** and events that correspond to

- **a new member joining the club:** each member of the club is represented by a unique identifier that is arbitrarily chosen from an abstract set *MEMBERS*;
- **a member adding money to their account:** each member has an account, to which they can add "money";
- **a member buying a cup of coffee:** there will be a variable, ***coffeeprice***, representing the cost of a cup of coffee, and each member can buy a cup of coffee provided they have enough money in their account.

The value of all money added to/removed from accounts is added to/removed from ***moneybank***.

CoffeeClub: Context

```
CONTEXT Members
SETS
        MEMBERS                 // REQ4
AXIOMS
        axm1: finite(MEMBERS)
        axm2: MEMBERS≠ ∅
END
```

- An abstract set MEMBERS, which we will use as the source of unique identifiers for members.
- The set is not given a specific size (cardinality), but it is declared to be finite, meaning that it does have a size (cardinality) that is a natural number. Sets are potentially infinite, unless declared otherwise. Note that in Event-B infinity is not a natural number.

Refinement: Concepts

Concepts of refinement:

- **Extension** is one type of refinement in which the state is extended by adding new variables and the functionality of some existing events are extended by the addition of new actions.
- In **extended** mode of an event, only the new parameters, guards and actions are displayed in this presentation. That is, only the actual extensions refined events are shown. This is done to make the extension clearer. In some of the following events no changes are shown indicating that the event is not affected by the extension.

Refinement of **CoffeeClub** machine

MACHINE **MemberShip** **REFINES** **CoffeeClub**
SEES **Members**
VARIABLES
moneybank // ”old” variable from the **CoffeeClub** model
members // REQ4: the set of current id members
accounts // REQ5: the member accounts
coffeeprice // REQ8: the price of a cup of coffee
INVARIANTS
inv1: $moneybank \in \mathbb{N}$ // ”old” invariant from the **CoffeeClub** model
inv2: $members \subseteq MEMBERS$ // REQ4: each member has unique id
inv3: $accounts \in members \rightarrow \mathbb{N}$ // REQ5: **each** member has an account (total function)
inv4: $coffeeprice \in \mathbb{N}$ // REQ8: any price other than free!
…

MemberShip machine (cont.)

INITIALISATION *event is extended here meaning that we add new actions to initialise variables of this refinement step but we keep that we have defined early.*

EVENTS
INITIALISATION: extended ≙
    then
        act2: *members* := ∅ // empty set of members
        act3: *accounts* := ∅ // empty set of accounts
        act4: *coffeeprice* :∈ N1 // initial coffee price set to arbitrary non-zero value
    end
SETPRICE ≙ // new event modelling REQ8: operation sets the price for a cup of coffee
    any amount
    where
        grd1: *amount* ∈ ℕ**1** // any value from the set of natural numbers, non-zero
    then
        act1: *coffeeprice* := amount
    end

MemberShip machine (cont.)

```
NEWMEMBER ≙                    // new event modelling REQ8
    any m
    where
        grd1: m ∈ MEMBERS \ members     // choose an unused element
                                           of MEMBERS
    then
        act1: members := members ∪ {m}   // add new member m to the
                                           members set
        act2: accounts(m) := 0           // initial value of the member's
                                           account is 0
    end
```

Notation

Shorthand
$f(x) := e$ is a shorthand for $f := f \lhd \{x \mapsto e\}$

MemberShip machine (cont.)

```
CONTRIBUTE ≙     // REQ6: a member can add money to own account
      any  amount
           m
      where
           grd1: amount ∈ N1              // any positive amount
           grd2: m ∈ members              // any member of a club
      then
           act1: accounts(m) := accounts(m) + amount       // this amount is added
                                          to the account of the member m
           act2: moneybank := moneybank + amount  // moneybank  is
                                                  updated REQ7
      end
```

MemberShip machine (cont.)

```
BUYCOFFEE  ≙          // REQ9: An operation that enables a member to buy a cup of coffee;
    any m
    where
        grd1: m ∈ members
        grd2: accounts(m) ≥ coffeeprice
    then
        act1: accounts(m) := accounts(m) – coffeeprice
            // the member's account is reduced by the cost of a cup of coffee;
        act2: moneybank := moneybank - coffeeprice
            // moneybank  is updated as required by REQ10

    end
```

MemberShip machine (cont.)

Neither **FEEDBANK** nor **ROBBANK** are affected by that model extension. So, these events are stay the same:

```
...
FEEDBANK extended ≙
    any
    where
    then
    end
ROBBANK extended ≙
    any
    where
    then
    end
```

Model verification via generating proof obligations

The proof obligations contain a surprise:

Contribute/moneybank/EQL on action

moneybank := moneybank + amount

cannot be discharged by the Rodin auto-prover.

This **EQL** proof obligation requires a proof that ***moneybank*** is not changed, but of course,

moneybank := moneybank + amount

must change the value of the variable moneybank, unless amount is 0.

Violation of refinement rules

CONTRIBUTE appears in the refinement as a new event, but here it is changing the value of the variable ***moneybank***, which is part of the state of **CoffeeClub**, the machine being refined.

The event **FEEDBANK** of **CoffeeClub** changes the value of the variable moneybank in a similar way to contribute, thus **CONTRIBUTE** must be seen as a refinement of **FEEDBANK** and **CONTRIBUTE** should be defined as follows.

CONTRIBUTE event refines FEEDBANK

CONTRIBUTE: refines FEEDBANK ≙

any *amount*

m

where

***grd1*:** *amount* ∈ N1

***grd2*:** *m* ∈ members

then

***act1*:** accounts(m) := accounts(m) + amount

***act2*:** moneybank := moneybank + amount

end

BUYCOFFEE event

Similar reasoning on the event **BUYCOFFEE:**

BUYCOFFEE is a new event in the refinement, but it is changing the value of the variable ***moneybank***, which is part of the state of **CoffeeClub.**

The event **ROBBANK** of **CoffeeClub** changes the value of the variable moneybank in a similar way, thus **BUYCOFFEE** must be seen as a refinement of **ROBBANK.**

Structure of the resulting **MemberShip** model

MACHINE **MemberShip** **REFINES** **CoffeeClub**
SEES **Members**
VARIABLES
moneybank members accounts coffeeprice
INVARIANTS
inv1: $moneybank \in \mathbb{N}$
inv2: $members \subseteq MEMBERS$
inv3: $accounts \in members \rightarrow \mathbb{N}$
inv4: $coffeeprice \in \mathbb{N}$
EVENTS
INITIALISATION: extended ≙ …
SETPRICE ≙ …
NEWMEMBER ≙ …
CONTRIBUTE **refines** **FEEDBANK** ≙ …
BUYCOFFEE **refines** **ROBBANK** ≙ …
END

On requirements document

The main goal of formal specification is to identify ambiguous, missing or contradicting requirements

There should be a correspondance between the description of the requirements and specification.

Start from transforming a general system description into a set of labeled requirements and assumptions.

For example: ”a libraty system keeps track of the books borrowed by each reader and ensures that no more than 20 books can be borrowed”

asm1. There is a finitie non empty set of readers

asm2. There is a finite non empty set of books

req1. Each user has a set of borrowed books associated with her. The cardinality of the set is from zero to 20.

On requirements document

While specifying and refining system, keep track of which assumptions and requirements have been modelled

For example:

asm1. There is a finite non empty set of readers Context0/axm1

asm2. There is a finite non empty set of books Context0/axm2

req1. Each user has a set of borrowed books associated with her. The cardinality of the set is from zero to 20. Variable loans Machine M1

req2. A user can borrow a book which is on shelf and not reserved if the maximum amount of books in her loan is not exceeded. Event Borrow Machine M1

Sequent calculus

- Sequent calculus is used to perform the proofs of the PO:s for the Event-B models
- A sequent is something we want to prove
  - It usually is of the form :

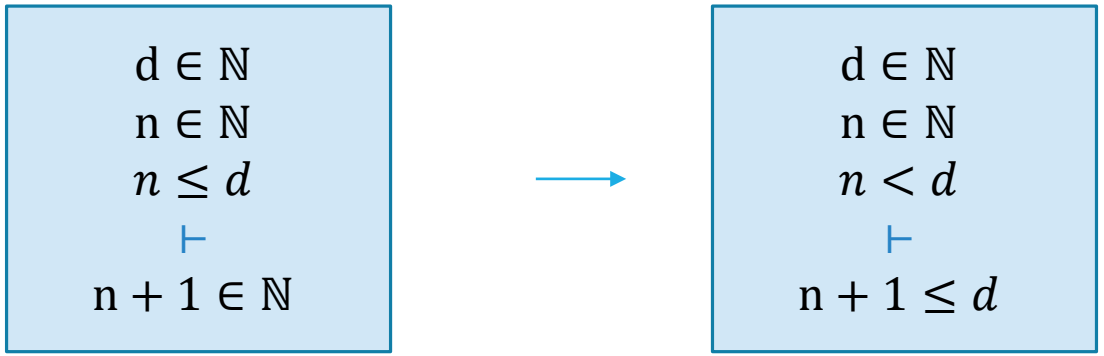
Hypotheses (H)

$\vdash$

Goal (G)

  - The symbol $\vdash$ is called turnstile
  - H is a (possibly empty) set of predicates that are called hyphothesis or assumptions
  - G is predicate that is called the goal or conclusion
- The intuitive semantics is that G can be proved under the assumptions H. That is, the assumptions H yield conclusion G.

Examples of Sequents



Inference rules

- An inference rule is used to construct proofs of sequents
- It consists of two parts:
  - Antecedent – a finite number of sequents
  - Consequent – a sequent

$$\mathbf{r1}\,\frac{A}{C}$$

Inference rule r1 yields a proof of C when we have a proof of each sequent in A!

A is the antecedent, C the consequent and r1 gives the name of the inference rule

Axioms

- An inference rule can have an empty antecedent

$$\mathbf{r2}\,\frac{}{C}$$

Inference rule r2 yields a proof of C!

- The inference rule is then called an axiom (schema)
- A sequent that can be proved from the axioms is called a theorem

Inference rule examples

$$\frac{\mathbf{H1} \vdash \mathbf{G}}{\mathbf{H1}, \mathbf{H2} \vdash \mathbf{G}}\ \text{MON}$$

$$\frac{}{\mathbf{H},\ \mathbf{n} \in \mathbb{N} \ \vdash\ \mathbf{n}+1 \in \mathbb{N}}\ \text{P2}$$

$$\frac{}{\mathbf{n} \leq \mathbf{m} \ \vdash\ \mathbf{n}-1 < \mathbf{m}}\ \text{DEC}$$

$$\frac{\mathbf{H} \vdash \mathbf{P}}{\mathbf{H} \vdash \mathbf{P} \vee \mathbf{Q}}\ \text{OR\_R1}$$

$$\frac{\mathbf{H} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \vee \mathbf{Q}}\ \text{OR\_R2}$$

Theory

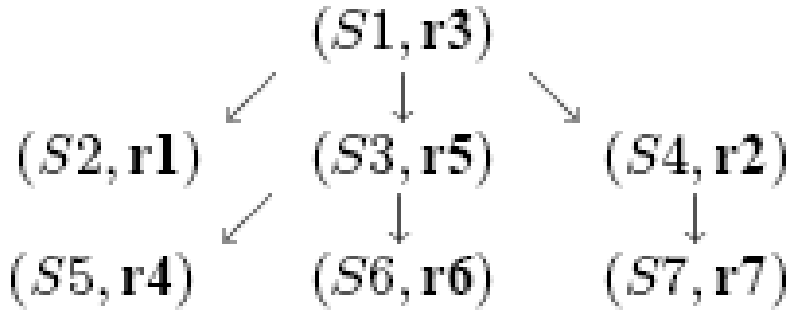
- A theory is a set of inference rules
- Enables proof of a sequent within a certain theory T
- A proof is a finite tree of applications of inference rules
  - Every node in the tree consists of a pair (r,s) where r is the inference rule and s is the consequent for that rule
  - The children of a node are all the sequents that appear in the antecedent of the rule
  - All leaf-nodes have inference rules without antecedents

Theory and proof - example

- A theory T

$$\mathbf{r1}\ \frac{}{S2} \qquad \mathbf{r2}\ \frac{S7}{S4} \qquad \mathbf{r3}\ \frac{S2\ \ S3\ \ S4}{S1} \qquad \mathbf{r4}\ \frac{}{S5} \qquad \mathbf{r5}\ \frac{S5\ \ S6}{S3} \qquad \mathbf{r6}\ \frac{}{S6} \qquad \mathbf{r7}\ \frac{}{S7}$$

- A proof of S1



$$\begin{array}{ll} S1 & \mathbf{r3} \\ \quad S2 & \mathbf{r1} \\ \quad S3 & \mathbf{r5} \\ \quad\quad S5 & \mathbf{r4} \\ \quad\quad S6 & \mathbf{r6} \\ \quad S4 & \mathbf{r2} \\ \quad\quad S7 & \mathbf{r7} \end{array}$$

A practical example