Event-B and Rodin

Seminar Formal Methods II

Johann Gschnaller

April 25, 2018

Overview

Motivation

"Faultless systems – yes we can!"

Prologue in "Modeling in Event-B" by Jean-Raymond Abrial [Abr10]

In a nutshell: Based on the idea of refinement, gradually construct formal models and enable a systematic reasoning by means of proofs.

A few key points:

- Modelling (not programming) a problem by analysing requirements.
- Show that the model has certain properties (invariants) by performing mathematical proofs.
- Gradually refine models: Start with an abstract model and add complexity step-by-step or transform the model such that it can be implemented more easily.

Such models can be used to eventually construct:

- Sequential programs
- Concurrent programs
- Distributed programs
- Electronic circuits
- ...

Detailed examples for each of these points can be found in the book of Abrial [Abr10].

Event-B

Event-B is a formal method for system modelling and analysis.

- Evolution of the B-method
- A notation used for developing mathematical models of discrete transition systems, i.e. a state based modelling approach where the transitions are described by events
- Basic language is predicate logic
- Problem modelling using set theory
- Use of refinement to represent the system at different abstraction levels
- Mathematical proofs to verify consistency between refinement levels and to guarantee system invariants

A great summary of the mathematical toolkit supported by Event-B may be found in Robinson [Rob10].

Rodin (Rigorous Open Development Environment for Complex Systems)

An development environment for creating Event-B models.

- Extension of the Eclipse IDE
- Type-checker and well-formedness
- Generates proof obligations (more on this later)
- Proof manager: (semi)automatic discharge of the proof obligations
- Allows refinement of the created models

Rodin is open source and can be downloaded at:
`https://sourceforge.net/projects/rodin-b-sharp/files/Core_Rodin_Platform/`.

Rodin plug-ins

Rodin is a modular software with many extensions. Some useful plug-ins:

Atelier B Provers
For conducting mathematical proofs

ProB
For interactive animation of Event-B models and model checking

Industrial applications

Event-B (and the B-method) have been used in several safety-critical systems. Some examples are shown on the following slides. For more applications consult the website:

`http://wiki.event-b.org/index.php/Industrial_Projects`.

Paris, Metro line 14

First real success (using the B-method) [LSP07]:

- A fully automatic driverless Metro 14 line was launched in Paris, October 1998.
- Over 110.000 lines of B models were written, generating 86.000 lines of Ada.
- After the development, no bugs were ever detected at the different testing, validation and operational phases.
- The safety-critical software is still in version 1.0 in 2007.

Other projects using Event-B

Siemens Transportation
Train control and signalling systems

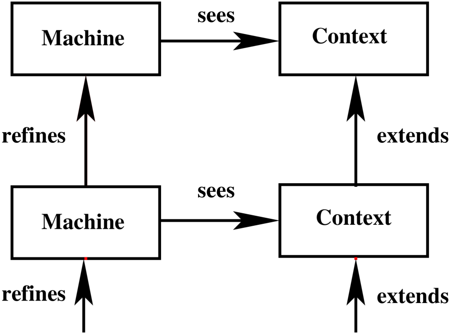
Bosch
Development of a cruise control system and a start-stop system

SAP
Analysis of business choreography models

Event-B components

An Event-B model consists of several components. A component can be either:

1. **Context**: Contains the static structure of the discrete system
2. **Machine**: Describes the dynamic part of the discrete system



Context

Contexts are grouped into the following sections:

- **Sets** User-defined data types. The identifier of a set implicitly creates a new constant.
- **Constants** Declared constants. The type must be declared in the axiom section.
- **Axioms** A list of predicates (called axioms). Axioms are statements that are assumed to be true in the model. They can be used as hypotheses in the proofs.

Other relevant modifiers:

- **Theorems** Axioms may be marked as theorems. Once proven, they can be used like regular axioms.
- **Extends** A context can extend other contexts to inherit their sets/constants/axioms.

Example context:

```
CONTEXT
    Array  ›
SETS
∘   V     ›values stored in the array
CONSTANTS
∘   f     ›f(i) gives the value of the array 'V' at index 'i'
∘   n     ›maximum array index
∘   t     ›target value
AXIOMS
∘   type_of_n:  n ∈ ℕ not theorem ›
∘   type_of_f:  f ∈ 1‥n → V not theorem ›
∘   target_in_array:    t ∈ ran(f) not theorem ›
∘   min_size_array: n ∈ ℕ1 theorem ›
END
```

Machine

Machines are grouped into the following sections:

- **Variables** Variables that change their values over time (state of the machine). Initialised in a special event.
- **Invariants** Predicates that must be true in every reachable state.
- **Variants** Used to guarantee termination. Termination means that a chosen set of events are enabled only a finite number of times.
- **Events** Assigns new values to a subset of the variables. Only active when its guard is true.

Other relevant modifiers:

- **Theorems** Can be applied to certain predicates.
- **Refines** A machine can be an refinement of another machine (a more concrete version).
- **Sees** The contexts that this machine has access to.

Example machine:

```
MACHINE
    Search ›
REFINES
∘   SearchAbstr
SEES
∘   Array
VARIABLES
∘   t_ind     ›target index
∘   j     ›current search index
INVARIANTS
∘   type_j: j ∈ ℕ not theorem ›
∘   searched_indices:   t ∉ f[1‥j] not theorem ›
VARIANT
∘   n − j ›
EVENTS
∘   INITIALISATION:     extended ordinary ›
    THEN
    ∘   init_t_ind: t_ind ≔ 1 ›
    ∘   init_j: j ≔ 0 ›
    END

∘   SEARCH: not extended ordinary ›
    REFINES
    ∘   SEARCH
    WHERE
    ∘   curr_index_is_target:   f(j + 1) = t not theorem ›
    WITH
    ∘   i:   j + 1 = i ›
    THEN
    ∘   find_target_index:   t_ind ≔ j + 1 ›
    END

∘   PROGRESS:   not extended convergent ›
    WHERE
    ∘   curr_index_not_target:  f(j + 1) ≠ t not theorem ›
    THEN
    ∘   increment_j:    j ≔ j + 1 ›
    END

END
```

Event

Events consist of the following concepts:

- **Parameters** A number of parameters for this event.
- **Guards** A number of predicates that specify when the event is enabled.
- **Witnesses** Used in refinements of an abstract machine.
- **Actions** Assignment of new values to a subset of the variables. Assignments can be deterministic or non-deterministic.

Other relevant modifiers:

- **Status** One of the values: ordinary, convergent, anticipated.
- **Refines** Designates the event(s) of the abstract machine that this event refines (special *SKIP* event for genuinely new events).

Example events:

```
∘ SEARCH: not extended ordinary ›
  REFINES
    ∘ SEARCH
  WHERE
    ∘ curr_index_is_target:  f(j + 1) = t not theorem ›
  WITH
    ∘ i:  j + 1 = i ›
  THEN
    ∘ find_target_index:  t_ind ≔ j + 1 ›
  END

∘ PROGRESS:  not extended convergent ›
  WHERE
    ∘ curr_index_not_target:  f(j + 1) ≠ t not theorem ›
  THEN
    ∘ increment_j:  j ≔ j + 1 ›
  END
```

Refinement

A central aspect in Event-B/Rodin. Used to gradually introduce details and add complexity. Refinement is relevant only for machines (contexts can be extended only). Two important aspects of machine refinement:

1. Ensure that the state of the refined machine is somehow connected to the abstract machine.
2. Each event of the abstract machine is refined by one in the more concrete machine.

Refinement variants:

Horizontal refinement Adds complexity to the model (superposition refinement)

Vertical refinement Introduce details to the data structures (data refinement)

Important concepts with respect to the mentioned refinement aspects (examples in live demo):

Gluing invariant
: Invariant that connects variables (state) in the concrete machine to variables in the abstract machine.

Witnesses
: When an abstract event has a parameter that is no longer used in the concrete event, a witness for the abstract parameter is needed. **Note:** In Rodin, witnesses have special labels.

Verification outline (details in second part of the seminar)

To guarantee the correctness of the model (e.g. invariants are never violated), certain conditions must be mathematically proven. In Event-B, the goals that need to be proven to verify this are called *proof obligations*.

Proof obligations

For contexts:
Necessary for theorems and to ensure well-formedness

For machines:
More involved, basically it must be guaranteed that

- the machine must be consistent, i.e., it should never reach a state which violates invariants
- behaviour of refined machines corresponds to that of the abstract machine.

Rodin comes with a proof manager that generates the necessary proof obligations automatically.

Many proof obligations are discharged automatically by Rodin. In other cases, human intervention is necessary. For the latter case, Rodin supplies a Proving Perspective (details in live demo).

List of generated proof obligations:

generated in contexts	
well-definedness of an axiom	label/WD
axiom as theorem	label/THM
generated for machine consistency	
well-definedness of an invariant	label/WD
invariant as theorem	label/THM
well-definedness of a guard	event/guardlabel/WD
guard as theorem	event/guardlabel/THM
well-definedness of an action	event/actionlabel/WD
feasibility of a non-det. action	event/actionlabel/FIS
invariant preservation	event/invariantlabel/INV
generated for refinements	
guard strengthening	event/abstract_grd_label/GRD
action simulation	event/abstract_act_label/SIM
equality of a preserved variable	event/variable/EQL
guard strengthening (merge)	event/MRG
well definedness of a witness	event/identifier/WWD
feasibility of a witness	event/identifier/WFIS
generated for termination proofs	
well definedness of a variant	VWD
finiteness for a set variant	FIN
natural number for a numeric variant	event/NAT
decreasing of variant	event/VAR

References I

[Abr10] Jean-Raymond Abrial, *Modeling in event-b: system and software engineering*, Cambridge University Press, 2010.

[LSP07] Thierry Lecomte, Thierry Servat, and Guilhem Pouzancre, *Formal methods in safety-critical railway systems*.

[Rob10] Ken Robinson, *A concise summary of the event b mathematical toolkit*.