Event-B and Refinement

Seminar Formal Methods II

Johann Gschnaller

June 27, 2018

Overview

1. Recap
2. Mathematical foundation
3. Event-B modelling
4. Refinement
5. References

Event-B and Rodin

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

- 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 (typed) set theory
- Use of refinement to represent the system at different abstraction levels
- Mathematical proofs to verify invariants and consistency between different refinement levels

Event-B and the B-method have been used in several safety-critical systems. Some industrial applications can be found at the website http://wiki.event-b.org/index.php/Industrial_Projects.

The IDE Rodin[1] can be used to develop such Event-B models.

[1] Rodin can be downloaded at the website https://sourceforge.net/projects/rodin-b-sharp/files/Core_Rodin_Platform/.

Refinement (basic principle)

Refinement is a hierarchical modelling approach.

- Start at an abstraction level where reasoning is simple
- Gradually add complexity to abstract models such that they get closer to the reality
- Nice analogy: view through a microscope. The object of interest does not change, but the more one zooms into a specific part, the more details are revealed (refinements represent the different zoom levels)
- Transform models such that they are easier to implement[2]

[2] It is possible to generate program code from Event-B models that is correct by construction. For more details consult the paper by Fürst et al. [FHB+14].

1. Recap
2. Mathematical foundation
3. Event-B modelling
4. Refinement
5. References

First-order predicate logic

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

Event-B supports the usual first-order logic predicates:

- Primitives: $\top$ and $\bot$
- Operators: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Equality: $=$ and $\neq$
- Quantifiers: $\forall x \cdot P(x)$ and $\exists y \cdot Q(y)$

Set theory

Apart from predicate logic, Event-B uses set theory as its basic modelling language.

Many set operations are available: set comprehension ($\{s \in S \mid P(s)\}$), union ($\cup$), intersection ($\cap$), Cartesian product ($\times$), power set ($\mathbb{P}$), cardinality ($card(S)$), set membership ($\in$), subset ($\subset$), set partitions, ...

Relations and functions

Event-B offers a variety of different types of relations and functions on sets, each of them with different additional properties.

A whole zoo of function and relation constructions is available: total/partial injections/surjections, bijective relations, range, domain, composition, ...

Arithmetic and types

The standard arithmetic operations are defined for the built-in sets $\mathbb{N}, \mathbb{N}_1$, and $\mathbb{Z}$.

Variables in Event-B are strongly typed. A type can be either

- a build-in type ($BOOL, \mathbb{N}, \mathbb{N}_1, \mathbb{Z}$) or
- an user-defined type (e.g. an enumerated set).

In contrast to most strongly typed programming languages, the variable type is not given at the declaration, but inferred from constraining properties such as axioms and invariants.

Well-definedness

Every formula in Event-B must be well-defined. The well-definedness predicate for a formula f, denoted by $\mathcal{L}(f)$, describes the condition when the formula can be safely evaluated.

Formula	Well-definedness condition
x	$\top$
$\neg P$	$\mathcal{L}(P)$
$\forall x \cdot P$	$\forall x \cdot \mathcal{L}(P)$
$E_1 \div E_2$	$\mathcal{L}(E_1) \wedge \mathcal{L}(E_2) \wedge E_2 \neq 0$
$card(S)$	$\mathcal{L}(S) \wedge finite(S)$

Sequent

To guarantee that a formal model fulfils its specified properties, Event-B defines so called *proof obligations*. Proof obligations are sequents of the form

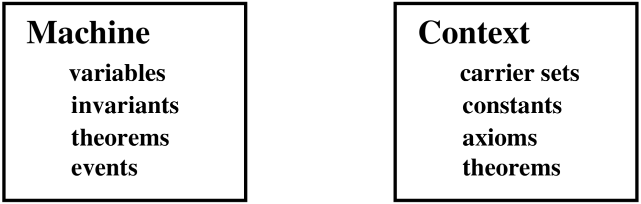
$$H \vdash G,$$

where G is a goal that must hold within the set of hypothesis H. Such proof obligations must be discharged using certain inference rules (not part of the presentation).

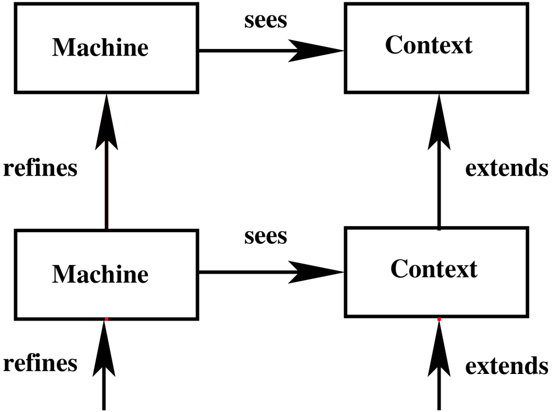
Event-B model

An Event-B *model* is a complete mathematical description of the discrete transition system. A model consists of several *components*, each of which can be either:

- Context: Describes the static part of a model
- Machine: Specifies the dynamic behaviour of a model



In order to facilitate a stepwise modelling approach, the following relations are defined for machines and contexts.



Context

Complete formal description of a context ($\star$ denotes an optional part):

context
    $< context\_identifier >$
extends $\star$
    $< context\_identifier >$
    $\ldots$
sets $\star$
    $< set\_identifier >$
    $\ldots$
constants $\star$
    $< constant\_identifier >$
    $\ldots$
axioms $\star$
    $< label >$: $< predicate >$
    $\ldots$
theorems $\star$
    $< label >$: $< predicate >$
    $\ldots$
end

Extends A context can extend other contexts to inherit their sets/constants/axioms.

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 axioms 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 proofs.

Theorems Once proven, theorems can be used like axioms.

```
CONTEXT
    TrafficLightContext ›
SETS
∘   Colour ›
CONSTANTS
∘   red ›
∘   orange ›
∘   green ›
AXIOMS
∘   axm1:   partition(Colour, {red}, {orange}, {green}) not theorem ›
END
```

Machine

Complete formal description of a machine ($\star$ denotes an optional part):

machine
  $< machine\_identifier >$
refines $\star$
  $< machine\_identifier >$
sees $\star$
  $< context\_identifier >$
  $\ldots$
variables
  $< variable\_identifier >$
  $\ldots$
invariants
  $< label >$: $< predicate >$
  $\ldots$
theorems $\star$
  $< label >$: $< predicate >$
  $\ldots$
events
  $< event >$
  $\ldots$
variant $\star$
  $< variant >$
end

Refines A machine can be a refinement of another machine (a more concrete version).

Sees The contexts that this machine has access to.

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.

Theorems Same as in the case of contexts.

Events Assigns new values to a subset of the variables. Only active when its guards are true.

Variants Used to guarantee termination. Termination means that a chosen set of events are enabled only a finite number of times.

```
MACHINE
    AbstractTrafficLight ›
VARIABLES
∘   cars_go ›
∘   peds_go ›
INVARIANTS
∘   cars_go_type:   cars_go ∈ BOOL not theorem ›
∘   peds_go_type:   peds_go ∈ BOOL not theorem ›
∘   lights_not_both:    ¬(cars_go = TRUE ∧ peds_go = TRUE) not theorem ›
EVENTS
∘   INITIALISATION:    not extended ordinary ›
    THEN
    ∘   cars_go_init:   cars_go ≔ FALSE ›
    ∘   peds_go_init:   peds_go ≔ FALSE ›
    END

∘   start_peds_go: not extended ordinary ›
    WHERE
    ∘   grd1:   cars_go = FALSE not theorem ›
    THEN
    ∘   act1:   peds_go ≔ TRUE ›
    END

∘   stop_peds_go:   not extended ordinary ›
    THEN
    ∘   act1:   peds_go ≔ FALSE ›
    END

∘   set_cars_go:   not extended ordinary ›
    ANY
    ∘   cars_go_param ›
    WHERE
    ∘   grd1:   cars_go_param ∈ BOOL not theorem ›
    ∘   grd2:   cars_go_param = TRUE ⇒ peds_go = FALSE not theorem ›
    THEN
    ∘   act1:   cars_go ≔ cars_go_param ›
    END

END
```

Event

Complete formal description of an event ($\star$ denotes an optional part):

$< event\_identifier > \;\widehat{=}$

status

$\{normal, convergent, anticipated\}$

refines $\star$

$< event\_identifier >$

$\ldots$

any $\star$

$< parameter\_identifier >$

$\ldots$

where $\star$

$< label >: \; < predicate >$

$\ldots$

with $\star$

$< label >: \; < witness >$

$\ldots$

then $\star$

$< label >: \; < action >$

$\ldots$

end

Status One of the values *ordinary*, *convergent*, *anticipated*. The latter two are useful in the presence of variants.

Refines Designates the event(s) of the abstract machine that this event refines (special *SKIP* event for genuinely new events).

Any A number of parameters for this event.

Where A number of predicates (called guards) that specify when the event is enabled.

With In case an event refines a more abstract event, the abstract parameters must receive a value in the refined event. Such assignments are called witnesses. The label of witnesses have a special form.

Then Assignments of new values to a subset of the variables (called actions). Assignments can be deterministic or non-deterministic.

```
set_cars_light:    not extended ordinary ›
REFINES
∘   set_cars_go
ANY
∘   cars_light_param      ›
WHERE
∘   grd1:   cars_light_param ⊂ Colour not theorem ›
∘   grd2:   green ∈ cars_light_param ⇒ peds_light = red not theorem ›
∘   grd3:   cars_light = {orange} ⇒ cars_light_param = {red} not theorem ›
∘   grd4:   cars_light = {red} ⇒ cars_light_param = {red, orange} not theorem ›
∘   grd5:   cars_light = {red, orange} ⇒ cars_light_param = {green} not theorem ›
∘   grd6:   cars_light = {green} ⇒ cars_light_param = {orange} not theorem ›
WITH
∘   cars_go_param:  cars_go_param = TRUE ⇔ green ∈ cars_light_param ›
THEN
∘   act1:   cars_light ≔ cars_light_param ›
END
```

Action

Formal description of a deterministic action:

$$< variable\_identifier\_list > \; := \; < expression\_list >$$

Actions can also be non-deterministic. In this case the formal description has the following form:

$$< variable\_identifier\_list > \; :| \; < before\_after\_predicate >$$

or alternatively (for a single variable):

$$< variable\_identifier > \; :\in \; < set\_expression >$$

Non-deterministic example: $i, j \;\; :| \;\; i' > j \;\wedge\; j' > i' + k$

Proof obligation

Proof obligations are conditions that must be proven to ensure that the model is consistent and has certain properties.
A selection of important kinds of proof obligations:

1. Well-definedness conditions
2. Invariant establishment/preservation (initial model)
3. Feasibility (initial model)
4. Guard strengthening (refinement)
5. Invariant preservation (refinement)
6. Simulation (refinement)
7. ... (consult the following slide for the proof obligations generated by Rodin)

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

Invariant establishment and preservation

Invariants are an essential concept of machines. One proves that an invariant holds in every state of the discrete transition system by induction:

Establishment: Invariants hold after the initialisation event.

Preservation: Each state transition preserves the invariant.

Invariant establishment: $init \mathrel{\widehat{=}} \textbf{then } v :\mid AP(s, c, v') \textbf{ end}$

$$\begin{aligned} s &: \text{seen sets} \\ c &: \text{seen constants} \\ v &: \text{machine variables} \\ A(s, c) &: \text{seen axioms} \\ AP(s, c, v') &: \text{initialisation after predicate} \end{aligned}$$

$$A(s, c), AP(s, c, v') \;\vdash\; I(s, c, v')$$

Invariant preservation:

$e \mathrel{\widehat{=}}$ **any** x **where** $G(x, s, c, v)$ **then** $v :\mid BAP(x, s, c, v, v')$ **end**

$$
\begin{aligned}
s &: \text{seen sets} \\
c &: \text{seen constants} \\
v &: \text{machine variables} \\
x &: \text{event parameters} \\
A(s, c) &: \text{seen axioms} \\
I(s, c, v) &: \text{invariants} \\
G(x, s, c, v) &: \text{event guards} \\
BAP(x, s, c, v, v') &: \text{event before-after predicate}
\end{aligned}
$$

$$A(s, c), I(s, c, v), G(x, s, c, v), BAP(x, s, c, v, v') \vdash I(s, c, v')$$

Feasibility

Feasibility ensures that an action is always feasible when the event guards are true, i.e., there always exists a value satisfying the before-after predicate (trivially true in the deterministic case).

$e \mathrel{\widehat{=}}$ **any** x **where** $G(x, s, c, v)$ **then** $v :\mid BAP(x, s, c, v, v')$ **end**

$$
\begin{aligned}
s &: \text{seen sets} \\
c &: \text{seen constants} \\
v &: \text{machine variables} \\
x &: \text{event parameters} \\
A(s, c) &: \text{seen axioms} \\
I(s, c, v) &: \text{invariants} \\
G(x, s, c, v) &: \text{event guards} \\
BAP(x, s, c, v, v') &: \text{event before-after predicate}
\end{aligned}
$$

$$A(s, c), I(s, c, v), G(x, s, c, v) \vdash \exists v' \cdot BAP(x, s, c, v, v')$$

Refinement

A central part of Event-B is the concept of refinement. *Refinement* is a mechanism for introducing details to the dynamic part of a model.

Note: It is possible to introduce more details to the static part of a model by context extensions.

Principle of substitutivity:
If a machine can be substituted by another in such a way that the users can not tell a substitution has taken place, the latter is called a refinement of the former.

In principle, one distinguishes between the following kinds of machine refinement (Event-B, however, does not differentiate between them):

1. Superposition refinement
2. Data refinement

Terminology: If a machine M refines a machine N, one calls N the abstract machine and M the concrete machine, respectively.

Superposition refinement

In *superposition refinement*, the variables of the abstract machine are kept. This refinement can introduce new variables and events.

Most importantly, the following conditions must be fulfilled in the concrete events:

- The concrete guards are stronger than the abstract ones. Thus, when the concrete event is enabled, so must be the corresponding abstract one (*guards strengthening*).
- The concrete actions simulate the abstract actions, i.e., actions do not contradict (*simulation*).
- Concrete invariants are preserved by each pair of concrete and abstract event.

$$
\begin{aligned}
e \;\widehat{=}\;& \textbf{any } x \textbf{ where } G(x, s, c, v) \\
& \textbf{then } v :\mid BAP_e(x, s, c, v, v') \textbf{ end} \\
f \;\widehat{=}\;& \textbf{refines } e \textbf{ any } x \textbf{ where } H(x, s, c, v, w) \\
& \textbf{then } v, w :\mid BAP_f(x, s, c, v, v', w, w') \textbf{ end}
\end{aligned}
$$

For the sake of brevity, the arguments are omitted in the following slides when they are of the following form.

$$
\begin{aligned}
s &: \text{seen sets} \\
c &: \text{seen constants} \\
v &: \text{abstract machine variables} \\
w &: \text{additional concrete machine variables} \\
x &: \text{event parameters} \\
A(s, c) &: \text{seen axioms} \\
I(s, c, v) &: \text{abstract invariants} \\
J(s, c, v, w) &: \text{concrete invariants} \\
G(x, s, c, v) &: \text{abstract event guards} \\
H(x, s, c, v, w) &: \text{concrete event guards} \\
BAP_e(x, s, c, v, v') &: \text{abstract event before-after predicate} \\
BAP_f(x, s, c, v, v', w, w') &: \text{concrete event before-after predicate}
\end{aligned}
$$

Guard strengthening:

$$A, I, J, H \ \vdash \ G$$

Simulation:

$$A, I, J, H, BAP_f \ \vdash \ BAP_e$$

Invariant preservation:

$$A, I, J(s, c, v, w), H, BAP_f \ \vdash \ J(s, c, v', w')$$

Data refinement

Data refinement is the case when abstract variables are removed and replaced by concrete variables. Here, the case when event parameters are replaced is included.

Gluing invariants: Since parts of the abstract variables are no longer available in a concrete machine, one has to establish a mechanism that connects the state of both machines. This is done by so called *gluing invariants*.

Witnesses: Similarly, when event parameters are replaced in a concrete event, a state transition in the abstract machine with a suitable parameter must be simulated. Such a suitable parameter is called a *witness*. Note that witnesses must be feasible.

Example:

```
CONTEXT
    TrafficLightContext ›
SETS
∘   Colour ›
CONSTANTS
∘   red ›
∘   orange ›
∘   green ›
AXIOMS
∘   axm1:   partition(Colour, {red}, {orange}, {green}) not theorem ›
END
```

```
MACHINE
    AbstractTrafficLight
VARIABLES
  ∘ cars_go
  ∘ peds_go
INVARIANTS
  ∘ cars_go_type:  cars_go ∈ BOOL not theorem
  ∘ peds_go_type:  peds_go ∈ BOOL not theorem
  ∘ lights_not_both:  ¬(cars_go = TRUE ∧ peds_go = TRUE) not theorem
EVENTS
  ∘ INITIALISATION:  not extended ordinary
    THEN
      ∘ cars_go_init:  cars_go ≔ FALSE
      ∘ peds_go_init:  peds_go ≔ FALSE
    END

  ∘ start_peds_go:  not extended ordinary
    WHERE
      ∘ grd1:  cars_go = FALSE not theorem
    THEN
      ∘ act1:  peds_go ≔ TRUE
    END

  ∘ stop_peds_go:  not extended ordinary
    THEN
      ∘ act1:  peds_go ≔ FALSE
    END

  ∘ set_cars_go:  not extended ordinary
    ANY
      ∘ cars_go_param
    WHERE
      ∘ grd1:  cars_go_param ∈ BOOL not theorem
      ∘ grd2:  cars_go_param = TRUE ⟹ peds_go = FALSE not theorem
    THEN
      ∘ act1:  cars_go ≔ cars_go_param
    END
```

```
MACHINE
    TrafficLight
REFINES
  ∘ AbstractTrafficLight
SEES
  ∘ TrafficLightContext
VARIABLES
  ∘ peds_light
  ∘ cars_light
INVARIANTS
  ∘ inv1:  peds_light ∈ {red, green} not theorem
  ∘ inv2:  cars_light ⊂ Colour not theorem
  ∘ gluing_peds:  peds_light = green ⇔ peds_go = TRUE not theorem
  ∘ gluing_cars:  green ∈ cars_light ⇔ cars_go = TRUE not theorem
EVENTS
  ∘ INITIALISATION:  not extended ordinary
    THEN
      ∘ peds_light_init:  peds_light ≔ red
      ∘ cars_light_init:  cars_light ≔ {red}
    END

  ∘ set_peds_light_green:  not extended ordinary
    REFINES
      ∘ start_peds_go
    WHERE
      ∘ grd1:  green ∉ cars_light not theorem
    THEN
      ∘ act1:  peds_light ≔ green
    END

  ∘ set_peds_light_red:  not extended ordinary
    REFINES
      ∘ stop_peds_go
    THEN
      ∘ act1:  peds_light ≔ red
    END

  ∘ set_cars_light:  not extended ordinary
    REFINES
      ∘ set_cars_go
    ANY
      ∘ cars_light_param
    WHERE
      ∘ grd1:  cars_light_param ⊂ Colour not theorem
      ∘ grd2:  green ∈ cars_light_param ⇒ peds_light = red not theorem
      ∘ grd3:  cars_light = {orange} ⇒ cars_light_param = {red} not theorem
      ∘ grd4:  cars_light = {red} ⇒ cars_light_param = {red, orange} not theorem
      ∘ grd5:  cars_light = {red, orange} ⇒ cars_light_param = {green} not theorem
      ∘ grd6:  cars_light = {green} ⇒ cars_light_param = {orange} not theorem
    WITH
      ∘ cars_go_param:  cars_go_param = TRUE ⇔ green ∈ cars_light_param
    THEN
      ∘ act1:  cars_light ≔ cars_light_param
    END
```

$$
\begin{aligned}
e \;\widehat{=}\;& \textbf{any } x \textbf{ where } G(x, s, c, v) \\
& \textbf{then } v :\mid BAP_e(x, s, c, v, v') \textbf{ end} \\
f \;\widehat{=}\;& \textbf{refines } e \textbf{ any } y \textbf{ where } H(y, s, c, w) \\
& \textbf{with } x : W_x(x, y, s, c, w), v' : W_{v'}(y, s, c, v', w) \\
& \textbf{then } w :\mid BAP_f(y, s, c, w, w') \textbf{ end}
\end{aligned}
$$

For the sake of brevity, the arguments are omitted in the following slides when they are of the following form.

$$
\begin{aligned}
s &: \text{seen sets} \\
c &: \text{seen constants} \\
v/w &: \text{abstract/concrete machine variables, resp.} \\
x/y &: \text{abstract/concrete event parameters, resp.} \\
A(s,c) &: \text{seen axioms} \\
I(s,c,v) &: \text{abstract invariants} \\
J(s,c,v,w) &: \text{concrete invariants and gluing invariants} \\
G(x,s,c,v) &: \text{abstract event guards} \\
H(y,s,c,w) &: \text{concrete event guards} \\
W_x(x,y,s,c,w) &: \text{witness for abstract parameters} \\
W_{v'}(y,s,c,v',w) &: \text{witness for abstract actions} \\
BAP_e(x,s,c,v,v') &: \text{abstract event before-after predicate} \\
BAP_f(y,s,c,w,w') &: \text{concrete event before-after predicate}
\end{aligned}
$$

Guard strengthening:

$$A, I, J, H, W_x \ \vdash \ G$$

Simulation:

$$A, I, J, H, W_x, W_{v'}, BAP_f \ \vdash \ BAP_e$$

Invariant preservation:

$$A, I, J(s, c, v, w), H, W_x, W_{v'}, BAP_f \ \vdash \ J(s, c, v', w')$$

References I

[Abr07] JR Abrial, *The event-b modelling notation*, wiki. event-b. org (2007).

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

[FHB+14] Andreas Fürst, Thai Son Hoang, David Basin, Krishnaji Desai, Naoto Sato, and Kunihiko Miyazaki, *Code generation for event-b*, International Conference on Integrated Formal Methods, Springer, 2014, pp. 323–338.

[Hoa13] Thai Son Hoang, *An introduction to the event-b modelling method*, Industrial Deployment of System Engineering Methods (2013), 211–236.

References II

[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*.