

Verification and tools in Event-B modelling

Mike Poppleton

users.ecs.soton.ac.uk/mrp

Slides adapted from Prof. Michael Butler,
Marktoberdorf Summer School 2012

Overview

- Abstraction & refinement
validation & verification
- Proof obligations in Event-B
- Rodin tool features

Problem Abstraction

- Abstraction can be viewed as a process of **simplifying** our understanding of a system.
- The simplification should
 - **focus** on the **intended purpose** of the system
 - **ignore** details of **how** that purpose is achieved.
- The modeller/analyst should make **judgements** about what they believe to be the **key features** of the system.

Abstraction (continued)

- If the purpose is to provide some **service**, then
 - model **what** a system does from the perspective of the service users
 - ‘users’ might be computing agents as well as humans.
- If the purpose is to **control**, **monitor** or **protect** some **phenomenon**, then
 - the abstraction should **focus** on those phenomenon
 - in **what** way should they be controlled, monitored or protected?

Refinement

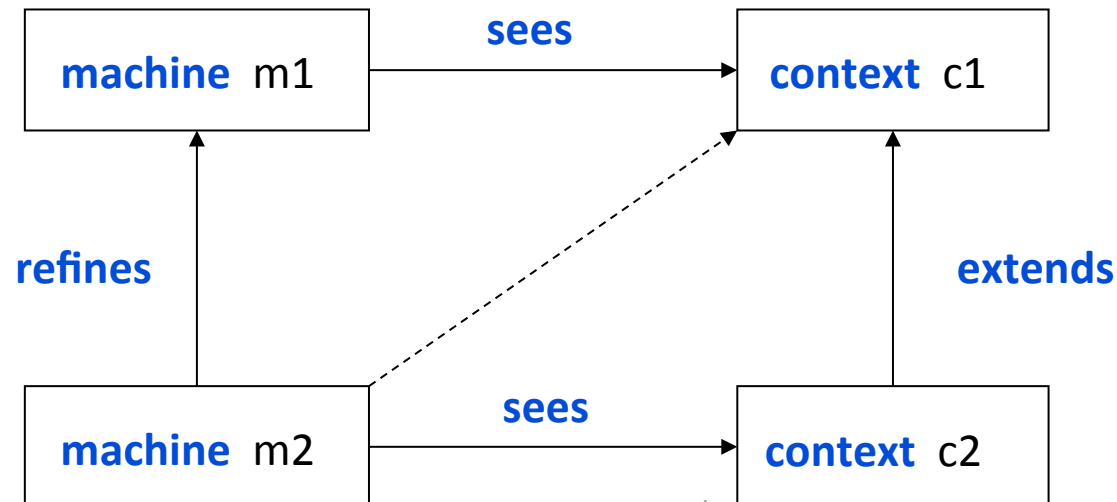
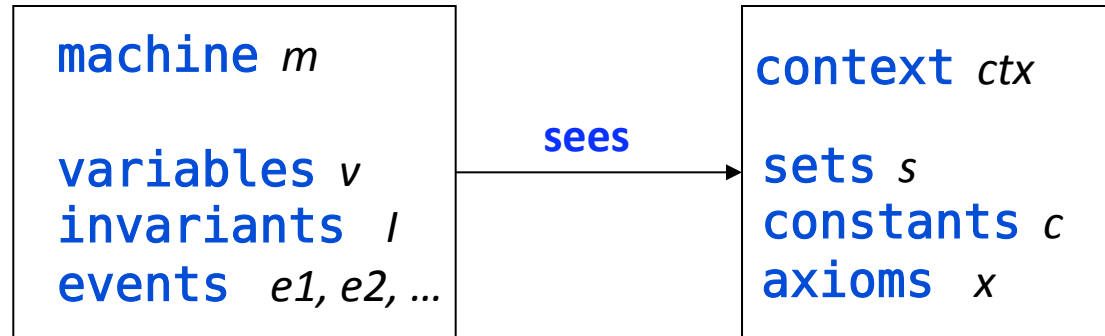
- Refinement is a process of **enriching** or **modifying** a model in order to
 - **augment** the functionality being modelled, **or**
 - **explain** how some purpose is achieved
- Facilitates abstraction: we can **postpone** treatment of some system features **to later** refinement steps
- Event-B provides a notion of **consistency** of a refinement:
 - Use proof to **verify the consistency** of a refinement step
 - **Failing proof** can help us identify **inconsistencies**

Validation and verification

- Requirements validation:
 - The extent to which (informal) requirements satisfy the needs of the stakeholders
- Model validation:
 - The extent to which (formal) model accurately captures the (informal) requirements
- Model verification:
 - The extent to which a model correctly maintains invariants or refines another (more abstract) model
 - Measured, e.g., by degree of validity of proof obligations

Event-B verification and tools

Event-B modelling components



Event structure

E =	// event name
any	
x1, x2, ...	// event parameters
where	
G1	// event guards (predicates)
G2	
...	
then	
v1 := exp1	// event actions
v2 := exp2	
...	
end	

Role of Event Parameters

- Most generally, parameters represent nondeterministically chosen values, e.g.,

NonDetInc =

any d **where** $v+d \leq \text{MAX}$ **then** $v:=v+d$ **end**

- Event parameters can also be used to model **input** and **output** values of an event
- Can also have nondeterministic actions:

when $v < \text{MAX}$ **then** $v :| v < v' \leq \text{MAX}$ **end**

Refinement for events

- A refined machine has two kinds of events:
 - **Refined** events that refine some event of the abstract machine
 - **New** events that refine *skip*
- Verification of event refinement uses
 - **gluing** invariants linking abstract and concrete variables
 - **witnesses** for abstract parameters

Rodin: Event-B perspective: model text, explorer, statemachine views:

The screenshot displays the Rodin Platform interface for an Event-B model. The main window is titled "Event-B - Pacemaker_VDD_3/machine1.bum - Rodin Platform - /Users/mrp/Working/rodinWorkspace/workspace 2.4".

Event-B Explorer: The left sidebar shows a tree view of the model. The selected item is "machine1", which contains "Variables", "Statemachine Atrium_SM", "Statemachine Ventricle_SM", "Invariants", "Events", and "Proof Obligations".

machine1: The central pane shows the Event-B text for "machine1". The code defines variables for Atrium and Ventricle, and includes events for Refractory, Start Monitoring, and Sensing.

```
where
  @isin_SensedIntSignal_A Atrium_SM_1 = SensedIntSignal_A
then
  @act1 intrinsic_signal_a = FALSE
  @enter_Atrium_SM_Refractory_A Atrium_SM_1 = Refractory_A
end

event Refractory2_A refines Refractory_A
where
  @grd2 a2v_signal = FALSE
  @grd1 v2a_signal = TRUE
  @isin_Refractory_A_or_isin_Start_Monitoring_A Atrium_SM_1 = Refractory_A
then
  @act1 v2a_signal = FALSE
  @enter_Atrium_SM_Refractory_A Atrium_SM_1 = Refractory_A
  @act2 intrinsic_signal_a = FALSE
end

event Start_Monitoring_A refines Monitor_A
where
  @grd2 v2a_signal = FALSE
  @isin_Refractory_A Atrium_SM_1 = Refractory_A
  @grd1 intrinsic_signal_v = FALSE
  @grd3 a2v_signal = FALSE
  @grd4 pace_signal_v = FALSE
then
  @enter_Monitoring_A_SM_Start_Monitoring_A Atrium_SM_1 = Start_Monitoring_A
end

event Sensing_A
```

***machine1.Atrium_SM.smd#0:** The top right pane shows a state machine diagram for the Atrium state machine. It includes states like "Refractory_A", "Start_Monitoring_A", and "Sensing_A", with transitions labeled "INITIALISATION", "Refractory2_A", and "Start_Monitoring_A".

machine1.Ventricle_SM.smd#0: The bottom right pane shows a state machine diagram for the Ventricle state machine. It includes states like "Refractory_V", "Start_Pacing_V", "End_Pacing_V", "Pace_V", "Monitor_V", "Start_Monitoring_V", and "Sensing_V", with transitions labeled "Refractory1_V", "Monitor_V", "Pace_V", "INITIALISATION", "Start_Monitoring2_V", and "Sensing_V".

Rodin Problems: The bottom left pane shows a table of problems. It indicates 997 errors, 40 warnings, and 0 infos. The table has columns for Description, Resource, and Path.

Description	Resource	Path
Errors (100 of 997 items)		
Warnings (40 items)		

Symbols: The bottom right pane shows a list of symbols used in the model, including logical operators and set notation.

Proof obligations in Event-B

- Well-definedness (WD)
 - e.g, avoid division by zero, partial function application
- Invariant preservation (INV) ***
 - each event maintains invariants
- Guard strengthening (GRD) ***
 - Refined event only possible when abstract event possible
- Simulation (SIM) ***
 - update of abstract variable correctly simulated by update of concrete variable
- Convergence (VAR)
 - Ensure convergence of new events using a variant

Invariant Preservation

- Assume: variables v and invariant $I(v)$
- Deterministic event:
 $Ev = \text{when } P(v) \text{ then } v := \text{exp}(v) \text{ end}$

- To prove Ev preserves $I(v)$:

$$\text{INV:} \quad P(v), I(v) \vdash I(\text{exp}(v))$$

- This is a sequent of the form $\text{Hypotheses} \vdash \text{Goal}$
- The sequent is a **Proof Obligation (PO)** that must be verified

Using Event Parameters

- Event has form:

$E_v = \text{any } x \text{ where } P(x,v) \text{ then } v := \text{exp}(x,v) \text{ end}$

INV: $I(v), P(x,v) \vdash I(E(x,v))$

Example PO from Rodin

Enter/inv3/INV

☐ ☒ ☒ ☐

☐ $\forall u, r \cdot$
 $u \in \text{dom}(\text{location}) \wedge$
 $\text{location}(u) = r$
 \Rightarrow
 $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$

☐ $u \in \text{USER} \setminus \text{dom}(\text{location})$

☐ $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$

☐ $(\text{location} \cup \{u \mapsto r\})(u_0) = r_0$

☐ $u_0 \in \text{dom}(\text{location} \cup \{u \mapsto r\})$

☐ $\text{takeplace} = \text{ROOM} \times \text{ACTIVITY}$

☐ $\text{location} \in \text{USER} \leftrightarrow \text{ROOM}$

Selected Hypotheses

☒ Goal ☒

$\text{takeplace}[\{(\text{location} \cup \{u \mapsto r\})(u_0)\}] \subseteq \text{authorised}[\{u_0\}]$

How do we know what to prove?

- Need for proofs imposes *proof obligations*
 - the user *does not have to state them*
 - they *are automatically generated* by a *tool*
- Proof obligations serve to
 - *verify properties* of a model

Rodin: Proving perspective: proof info, proof & control, PO explorer views:

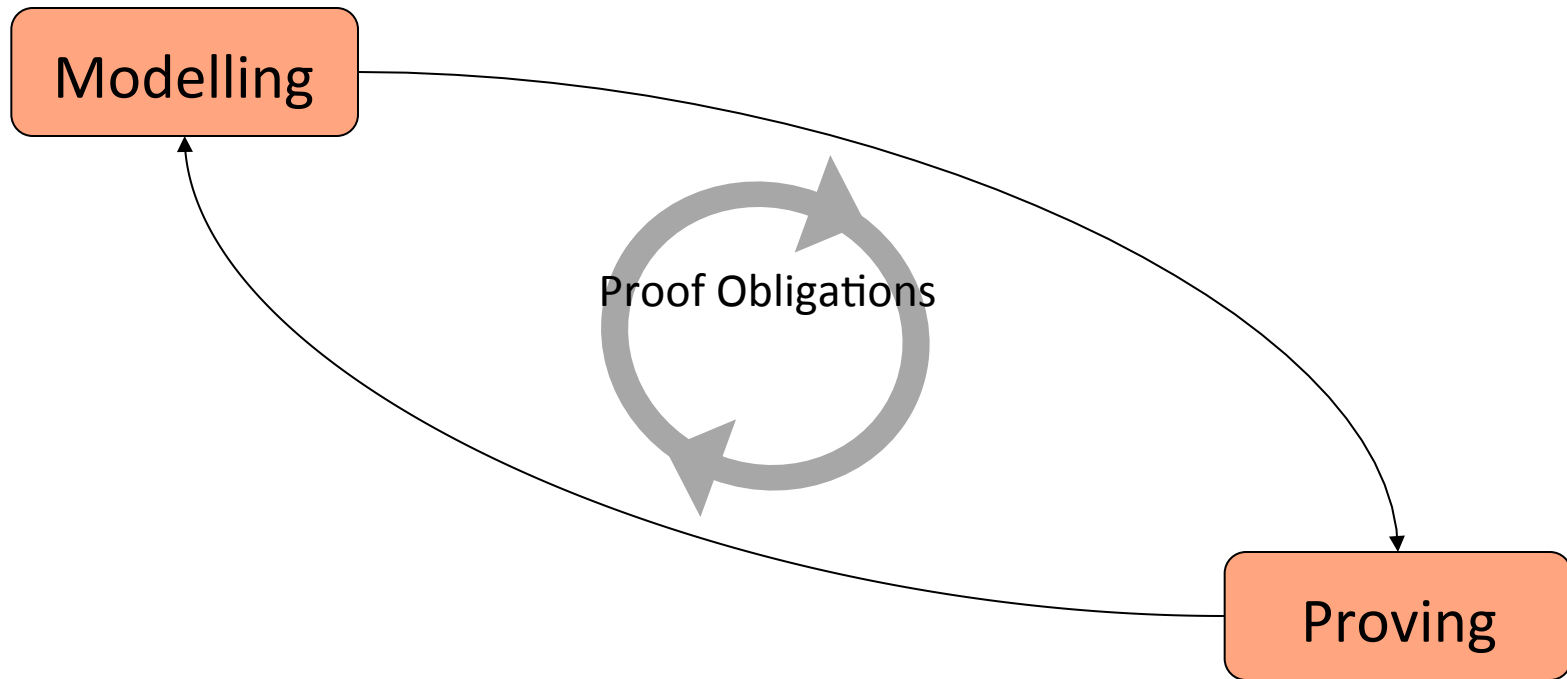
The screenshot displays the Rodin Platform interface with the following components:

- Proof Information:** Shows the event in machine0 and machine1, including the invariant $\text{Refractory_V_Invtriant} : \text{Ventricle_SM_1} = \text{Refractory_V} \Rightarrow \text{Ventricle_SM_0} =$.
- Proof:** Displays the goal $\text{Start_Monitoring_V} = \text{Refractory_V} \Rightarrow \text{Monitoring_V} = \text{Refractory_V}$ and the selected hypotheses.
- Proof Control:** Includes a red sad face icon and a large empty box for proof control.
- Event-B Explorer:** Lists various events and invariants, such as $\text{Refractory1_V/Start_Monitoring_V_Invtriant/INV}$ and $\text{Sensing_V/Start_Pacing_V_Invtriant/INV}$.
- Symbols:** A panel showing various symbols and variables.

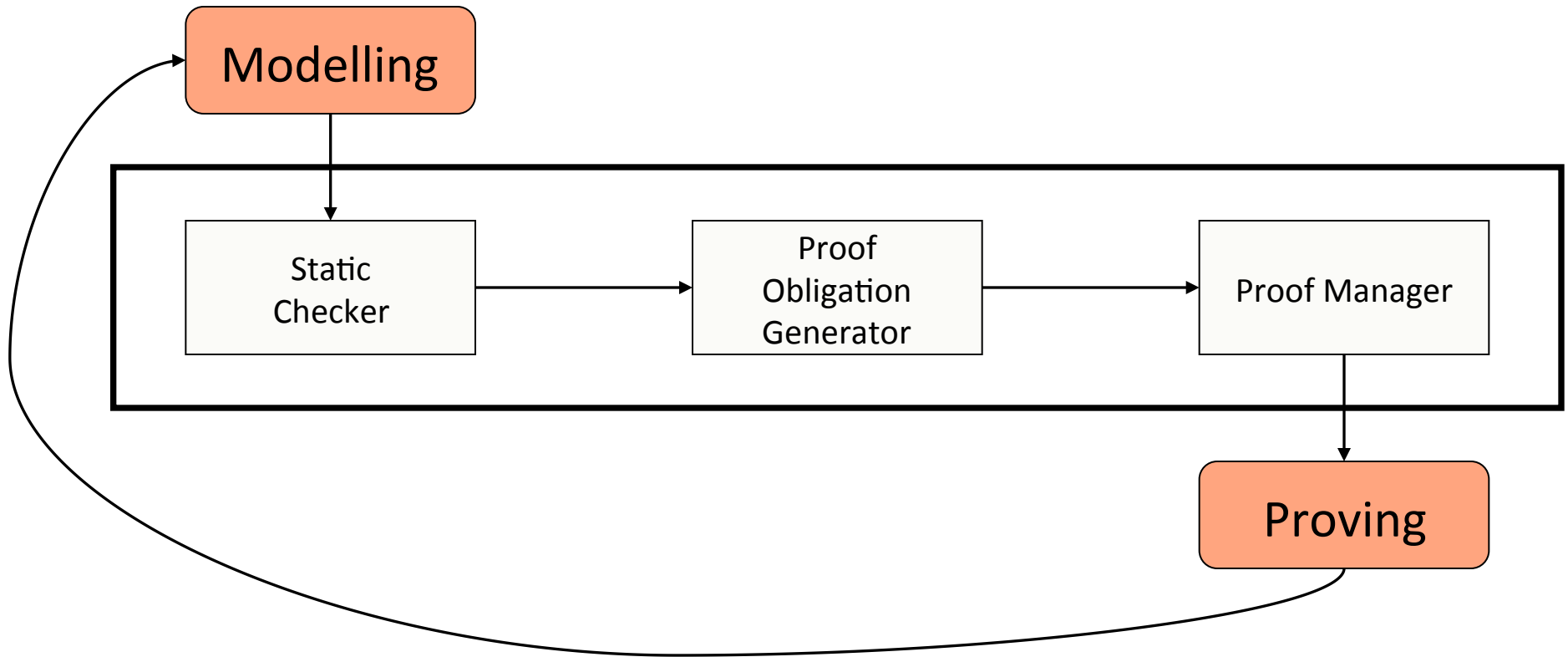
Proof and model checking

- **Model checking:** force the model to be finite state and explore state space looking for invariant violations
 - completely automatic
 - powerful debugging tool (counter-example)
- **(Semi-)automated proof:** based on logical deduction rules
 - no restrictions on state space
 - leads to discovery of invariants that deepen understanding
 - not completely automatic

Models are created and verified iteratively



Rodin architecture



Rodin Architecture

- Extension of Eclipse IDE
- Repository of structured modelling elements
- Rodin Eclipse Builder manages:
 - Well-formedness + type checker
 - Consistency/refinement PO generator
 - Proof manager
 - Propagation of changes
- Extension points to support plug-ins

Differential proving in Rodin

- Models are constantly being changed
- When a model changes, proof impact of changes should be minimised as much as possible:
- Sufficiency comparison of POs
 - In case of success, provers return list of *used hypotheses*
 - Proof valid provided the used hypothesis are in the new version of a PO
- Model refactoring:
 - Identifier renaming applied to models (avoiding name clash)
 - Corresponding POs and proofs automatically renamed

Rodin Proof Manager (PM)

- PM constructs **proof tree** for each PO
- Automatic and interactive modes
- PM manages **used hypotheses**
- PM calls ***reasoners*** to
 - **discharge** goal, or
 - **split** goal into **subgoals**
- Collection of reasoners:
 - **simplifier, rule-based, decision procedures, ...**
- Basic **tactic language** to define PM and reasoners

Statistics from Flash-based file development in Event-B

Machines	Total POs	Automatic	Interactive
MCH0	35	22	13
MCH1	57	49	8
MCH2	33	32	1
MCH3	37	34	3
MCH4	26	26	0
MCH5	27	26	1
MCH6	31	30	1
MCH7	109	97	12
MCH_FL0	8	8	0
MCH_FL1	110	110	0
MCH_FL2	57	57	0
MCH_FL3	9	9	0
Overall	540	501 (93%)	39 (7%)

Range of Automated Provers

- **Built-in:** tactic language, simplifiers, decision procedures
- **AtelierB plug-in** for Rodin (ClearSy, FR)
- **SMT plug-in** (Systerel, FR)
- **Isabelle plug-in** (Schmalz, ETHZ)

Validation/verification offered by ProB

- Animation: show behaviour of model in clear terms
- Model Checking
- Refinement Checking
- Graphical Domain Specific Visualization
- Visualization of State Space



ProB

- Animator and model checker
 - search for **invariant violations**
 - search for **deadlocks**
 - search for **proof obligation violations**
- Implementation uses constraint logic programming
 - makes all types **finite**
 - exploits **symmetries** in B types

Rodin: ProB perspective: model text, ProB views:

ProB - Pacemaker_VDD_3/machine1.bum - Rodin Platform - /Users/mrp/Working/rodinWorkspace/workspace 2.4

Events

Event	Parameter(s)
Refractory1_A	
Refractory2_A	
Start_Monitoring_A	
Sensing_A (x2)	
SenseIntSignal_A	
Refractory1_V	
Refractory2_V	
Start_Monitoring1_V	
Start_Monitoring2_V	
Sensing_V (x2)	
SenseIntSignal_V	
StartPacing_V	
EndPacing_V	

Event-B E Rodin Pro

- AccessControl_per_mjb
- ATMBalTransferN
- ATMDepositN
- bms_PaceMaker_Basic_03
- bms_PaceMaker_DDD2
- bms_PaceMaker_DVI2
- bms_PaceMaker_VDD2
- bms_RezaPaceMaker_VDD2_value
- Counter
- de.prob.playback
- de.prob.random
- de.prob.record
- Email
- emails
- List example
- MathExtensions
- MikePaceMaker_Basic_03
- MikeRezaPaceMaker_VDD2

machine1

```

where
  @isin_SensedIntSignal_A Atrium_SM_1 = SensedIntSignal_A
then
  @act1 intrinsic_signal_a = FALSE
  @enter_Atrium_SM_Refractory_A Atrium_SM_1 = Refractory_A
end

event Refractory2_A refines Refractory_A
where
  @grd2 a2v_signal = FALSE
  @grd1 v2a_signal = TRUE
  @isin_Refractory_A_or_isin_Start_Monitoring_A Atrium_SM_1 = Refractory_A v Atrium_SM_1 = Start_Monitoring_A
then
  @act1 v2a_signal = FALSE
  @enter_Atrium_SM_Refractory_A Atrium_SM_1 = Refractory_A
  @act2 intrinsic_signal_a = FALSE
end

event Start_Monitoring_A refines Monitor_A
where
  @grd2 v2a_signal = FALSE
  @isin_Refractory_A Atrium_SM_1 = Refractory_A
  @grd1 intrinsic_signal_v = FALSE
  @grd3 a2v_signal = FALSE
  @grd4 pace_signal_v = FALSE
then
  @enter_Monitoring_A_SM_Start_Monitoring_A Atrium_SM_1 = Start_Monitoring_A
end

event Sensing_A
where
  @isin_Start_Monitoring_A Atrium_SM_1 = Start_Monitoring_A
  @grd1 intrinsic_signal_a = FALSE
then
  @act1 intrinsic_signal_a :c BOOL
  @enter_Monitoring_A_SM_Start_Monitoring_A Atrium_SM_1 = Start_Monitoring_A
end

event SenseIntSignal_A

```

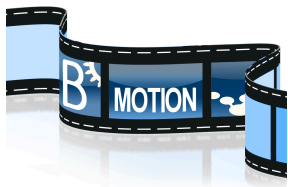
State

Name	Value
machine0_implicitContext	
Monitoring_A	{vSISA, vStMonA}
Monitoring_V	{vSISV, vStMonV}
Pacing_V	{vEndPacV, vStPacV}
Refractory_A	{vRefA}
Refractory_V	{vRefV}
machine1_implicitContext	
End_Pacing_V	{vEndPacV}
SensedIntSignal_A	{vSISA}
SensedIntSignal_V	{vSISV}
Start_Monitoring_A	{vStMonA}
Start_Monitoring_V	{vStMonV}
Start_Pacing_V	{vStPacV}
machine0	
Atrium_SM_0	{vSISA, vStMonA}
Ventricle_SM_0	{vSISV, vStMonV}
machine1	
Atrium_SM_1	{vStMonA}
Ventricle_SM_1	{vStMonV}
a2v_signal	FALSE
intrinsic_signal_a	FALSE
intrinsic_signal_v	FALSE
pace_signal_v	FALSE
v2a_signal	FALSE
Formulas	
invariants	T
axioms	T
guards	

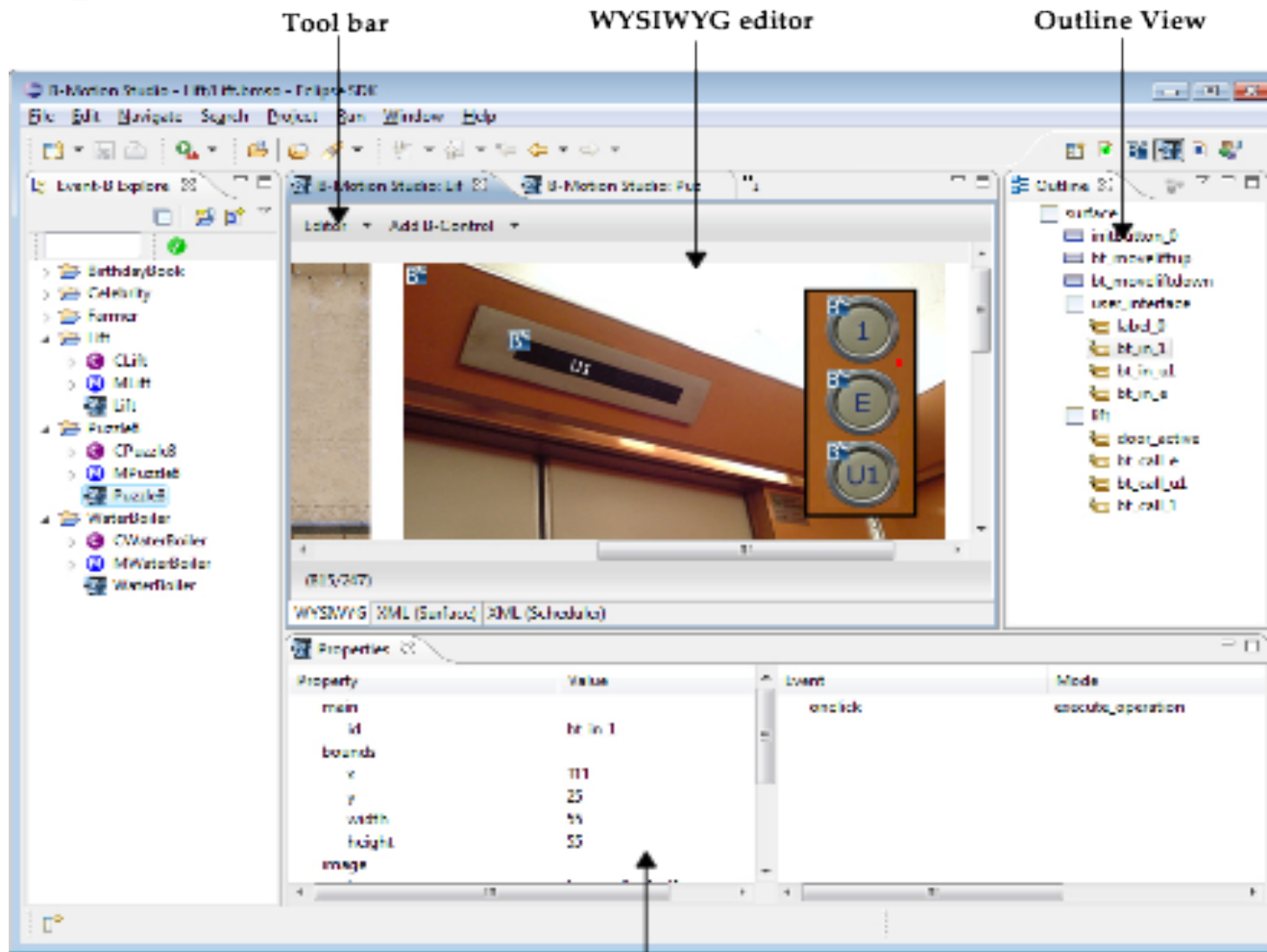
machine1

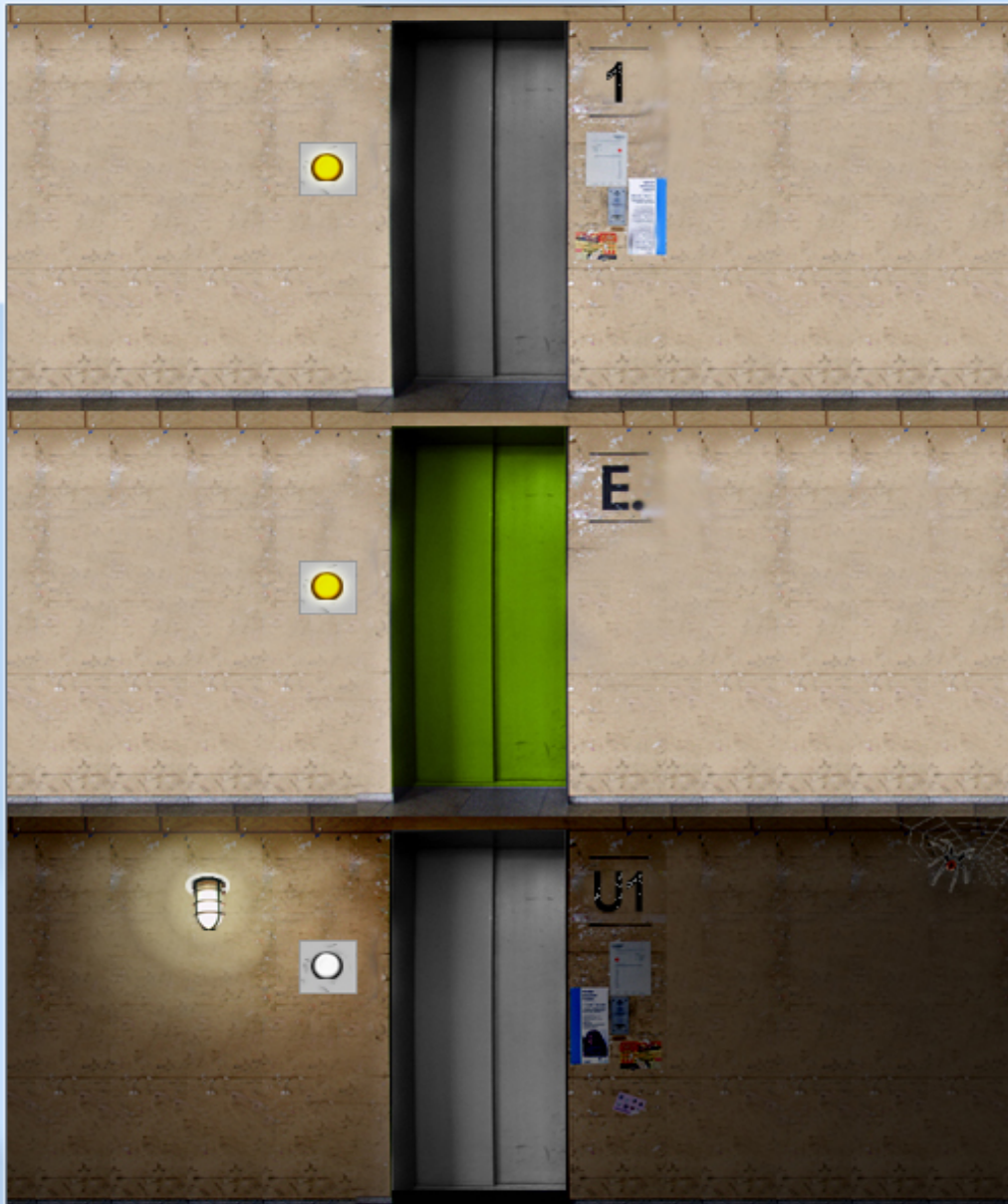
- Start_Monito...
- Start_Monito...
- Refractory2_A
- Refractory1_V
- SenseIntSign...
- Sensing_V
- INITIALISATION
- SETUP_CONT...
- (uninitialised...

Invariant ok no event errors detected



BMotionStudio





Move Lift UP

Move Lift DOWN

Initialize machine

Proof and model checking

- **Model checking:** force the model to be finite state and explore state space looking for invariant violations
 - ☺ completely automatic
 - ☺ powerful debugging tool (counter-examples)
 - ☹ state-space explosion
- **(Semi-)automated proof:** based on deduction rules
 - ☹ not completely automatic
 - ☺ leads to discovery of invariants - deepen understanding
 - ☺ no restrictions on state space

Some references

- Full introduction to modelling and verification in Event-B, to advanced level (including definition of proof obligations):
 - Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press 2010
- Abrial, Butler, Hallerstedde, Hoang, Mehta and Voisin
 - *Rodin: An Open Toolset for Modelling and Reasoning in Event-B*.
 - International Journal on Software Tools for Technology Transfer (STTT), 12 (6), 2010.
- Leuschel and Butler
 - *ProB: An Automated Analysis Toolset for the B Method*. *International Journal on Software Tools for Technology Transfer*, 10, (2), 185-203, 2008.

Rodin and its plug-ins:

read about and install via www.event-b.org

- ProB model checker:
 - consistency and refinement checking
- External provers:
 - AtelierB plug-in for Rodin (ClearSy, FR)
 - SMT plug-in (Systerel, FR)
 - Isabelle plug-in (Schmalz, ETHZ)
- Theory plug-in – user-defined mathematical theories
- UML-B: Linking UML and Event-B
- Graphical model animation
 - ProB, AnimB, B-Motion Studio
- Requirements management (ProR)
- Team-based development
- Decomposition
- Code generation
- ...

END