Modelling Software-based Systems

The Modelling Language Event-B

Telecom Nancy

Dominique Méry
Telecom Nancy,Université de Lorraine

25 mars 2025

dominique.mery@loria.fr

General Summary

1. Documentation
2. Introduction by Problems
   - Safety Properties of C Programs
   - Importance of Domain
   - Tracking bugs in C codes
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

Tools

- Event B : http ://www.event-b.org/
- Atelier B : http ://www.atelierb.eu/
- RODIN Platform : http ://www.event-b.org/platform.html
- EB2ALL Toolset : http ://eb2all.loria.fr
- RIMEL project : http ://rimel.loria.fr
- The Modelling Language Event-B and related topics as lectures notes, tutorials, models. https ://mery54.github.io/teaching/mosos/
- The Modelling Language Event-B and related topics as lectures notes, tutorials, models. https ://mery54.github.io/teaching/mosos/

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

RTE

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int x, y;
    // Seed the random number generator with the current time
    srand(time(NULL));
    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
    // Perform some calculations
    y = x / (100 - x);
    printf("Result: %d\n", y);
    return 0;
}
```

RTE with frama-c

```c
int main(void)
{
  int __retres;
  int x;
  int y;
  time_t tmp;
  int tmp_0;
  tmp = time((time_t *)0);
  srand((unsigned int)tmp);
  { /* sequence */
    tmp_0 = rand();
    /*@ assert rte: signed_overflow: (int)(tmp_0 % 100) + 1 <= 214
    x = tmp_0 % 100 + 1;
  }
  /*@ assert rte: signed_overflow: 100 - x <= 2147483647; */
  /*@ assert rte: division_by_zero: (int)(100 - x) != 0; */
  /*@ assert rte: signed_overflow: x / (int)(100 - x) <=  21474836
  y = x / (100 - x);
  printf("Result: %d\n",y); /* printf_va_1 */
  __retres = 0;
  return __retres;
}
```

RTE with frama-c

```c
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y,i=0;

    for (i = 0; i <= 100000; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL));

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
        printf("Result: x= %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d %d\n",i, y);
    }

    return 0;
}
```

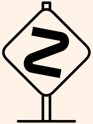
RTE with frama-c but a modification

```
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y,i=0;

    for (i = 0; i <= 100; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL)+i);

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
        printf("Result: x= %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d  %d\n",i, y);
    }

    return 0;
}
```

Implicit and explicit in formal modelling

Our aim is to analyze what is implicit and what is explicit in formal modelling...

- **Semantics in modelling** :
  - Semantics expressed by a *theory* (e.g. Event-B) used to formalize hardware and/or software systems
  - Same theory is used for wide variety of heterogeneous systems
- **Semantics in domain** :
  - Environment within which system evolve : application domain/context
  - Information provided by domain is often associated while in operation
  - Either assumed and omitted while formalising systems or hardcoded in formal models
  - Same context is used for wide variety of heterogeneous systems



A case study for studying these properties

Nose Gear Velocity

- Estimated ground velocity of the aircraft should be available only if it is within 3 $\mathtt{km/hr}$ of the true velocity at some moment within past 3 seconds

Characterization of a System (I)

- NG velocity system :
  - **Hardware** :
    - *Electro-mechanical sensor* : detects rotations
    - *Two 16-bit counters* : Rotation counter, Milliseconds counter
    - *Interrupt service routine* : updates rotation counter and stores current time.
  - **Software** :
    - *Real-time operating system* : invokes update function every `500 ms`
    - *16-bit global variable* : for recording rotation counter update time
    - *An update function* : estimates ground velocity of the aircraft.
- Input data available to the system :
  - *time* : in `milliseconds`
  - *distance* : in `inches`
  - *rotation angle* : in `degrees`
- Specified system performs velocity estimations in *imperial* unit system
- Note : expressed functional requirement is in *SI* unit system (km/hr).

What are the main properties to consider for formalization ?

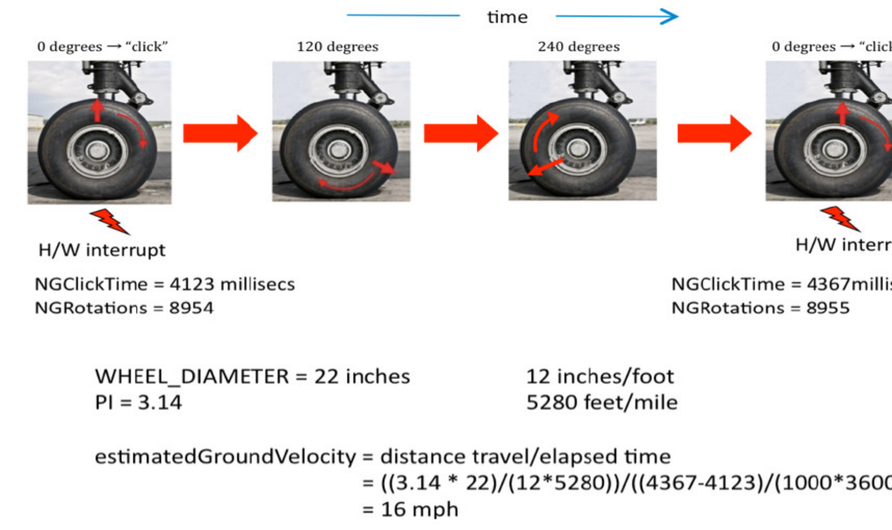
- Two different types of data :
  - counters with modulo semantics
  - non-negative values for time, distance, and velocity
- Two dimensions : *distance* and *time*
- Many units : distance (`inches, kilometers, miles`), time (`milliseconds, hours`), velocity (`kph, mph`)
- And interaction among components

How should we model ?

- Designer needs to consider units and conversions between them to manipulate the model
- One approach : Model units as *sets*, and conversions as constructed types – *projections*.
- Example :

  1 $estimateVelocity \in \mathtt{MILES} \times \mathtt{HOURS} \rightarrow \mathtt{MPH}$
  2 $mphTokph \in \mathtt{MPH} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathtt{KPH}$

Sample Velocity Estimation



bug0.c

Listing 1 – Bug bug0

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int x, y;
    // Seed the random number generator with the current time
    srand(time(NULL));
    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
    // Perform some calculations
    y = x / (100 - x);
    printf("Result: %d\n", y);
    return 0;
}
```

bug00.c

Listing 2 – Bug bug00

```
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y,i=0;

    for (i = 0; i <= 100000; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL));

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
        printf("Result: x=  %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d  %d\n",i, y);
    }

    return 0;
}
```

bug000.c

Listing 3 – Bug bug000

```c
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y, i=0;

    for (i = 0; i <= 100; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL)+i);

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
        printf("Result: x= %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d %d\n",i, y);
    }

    return 0;
}
```

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

Modelling Systems

- Distributed systems : web services, information systems, distributed algorithms . . .
- Safety critical systems : medical devices, embedded systems, cyber-physical systems, . . .
- Fault-tolerant systems : networks, communication infrastructure, . . .
- Environments : heart, the glucose-insulin regulatory system, . . .

Modelling in action

- Abstraction and refinement of features, 2000 *with D. Cansell*
- Incremental Proof of the Producer/Consumer Property for the PCI Protocol, 2002 *with D. Cansell, G. Gopalkrishnan, S. Jones.*
- A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol, 2003, *with J.-R. Abrial and D. Cansell.*
- The challenge of QoS for digital television services-. *EBU Technical Review* (avril 2005) *with D. Abraham, D. Cansell, C. Proch.*
- -Formal and Incremental Construction of Distributed Algorithms : On the Distributed Reference Counting Algorithm, 2006 *with D. Cansell.*

Modelling in action

- Refinement : A Constructive Approach to Formal Software Design for a Secure e-voting Interface-, 2007 *with D. Cansell and P. Gibson.*
- Incremental Parametric Development of Greedy Algorithms, 2007, *with D. Cansell.*
- System-on-Chip Design by Proof-based Refinement, 2009 *with D. Cansell and C. Proch*
- -A simple refinement-based method for constructing algorithms, 2009. *Alone.*
- -Refinement-based guidelines for algorithmic systems-. *Alone. International Journal of Software and Informatics* (2009),

Modelling in action

- Cryptologic algorithms : Event B development, combining cryptologic properties, modeling attacks.
- Access control systems : relating policy models and Event B models like in RBAC, TMAC, ORBAC
- Distributed algorithms : integration of local computation models into Event B, tool B2VISIDIA, algorithms of naming, election etc
- Medical devices : modelling the pacemaker, interacting with cardiologists, ...
- Modelling self-$\star$ systems
- Modelling medical devices item Modelling environments for medical devices : closed-loop modelling

Next modelling

- Modelling human-in-the -loop systems
- Modelling cyber-physical systems

General Approach

- Constructing a model of the system
- Elements for defining a formal or semi-formal model : syntax, semantics, verification, validation, documentation
- Mathematical structures : transition systems, temporal/modal/deontic/... logics,
- Validation of a model : tests, proofs, animation,...
- Modelling Techniques : state-based techniques
- Structure of a model : module, object, class,
- Design Patterns

Mathematical tools for modelling systems

- set theory : sets, relations, functions ...
- transition systems
- predicate calculus
- decision procedures
- interactive theorem prover

Examples of modelling languages

- Z : set theory, predicate calculus, schemas.
- VDM : types, pre/post specification, invariant, operations
- B : set theory, predicate calculus, generalized susbtitution, abstract machines, refinement, implementation.
- RAISE : abstract data types, functions,
- TLA$^+$: set theory, modules, temporal logic of actions.
- UNITY : temporal logic,actions systems, superposition.
- UML
- JML and Spec$\#$: programming by contract

Objectives of the modelling

- To get a better understanding of the current system : requirements, properties, cost, maintenance . . .
- To document the the system
- To systematize operations of modelling : reuse, parametrization
- To ensure the quality of the final product : safety, security issues
- To elaborate a contract between the customer and the designer

The Triptych Approach

$$\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R} \tag{1}$$

- $\mathcal{R}$ requirements or system properties
- $\mathcal{D}$ domain of the problem
- $\mathcal{S}$ model of the system
- $\longrightarrow$ relation of satisfaction

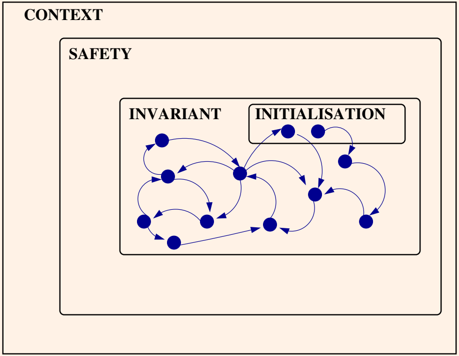
Formal modelling

- Mathematical foundations of Models : syntax, semantics, pragmatics, theory, soundness.
- Mathematical reasoning is based on sound proof rules
- Common language for fac ilitating the communication.

Current Summary

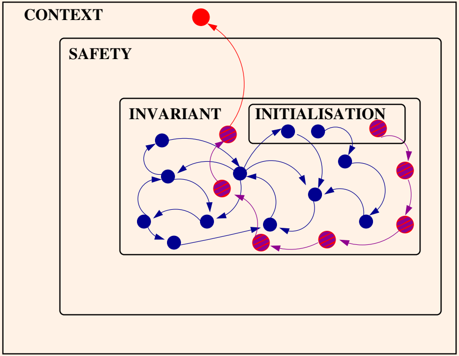
1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

Observing the safe system



- The context defines the possible values
- Safety requirement means that *something bad will never happen*.
- Invariant defines the set of effective possible values
- Transitions modify state variables and maintains the invariant.

Observing the unsafe system



- Transitions modify state variables and may not maintain the invariant.
- . . . and may not guaranteesafety properties.

Tools

- Event B : http ://www.event-b.org/
- Atelier B : http ://www.atelierb.eu/
- RODIN Platform : http://www.event-b.org/platform.html

The Event B Method

- The Event B Method is invented by J.-R. Abrial from 1988 : abstract system, events, refinement, invariant.
- Atelier B and RODIN are supporting the Event B method
- An event is observed and triggered, when a guard is true
- Proof obligations are generated using the weakest-precondition semantics.
- A Event B model intends to model a reactive system.

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

A Simple Example

Managing teachers, students, lectures and class rooms

- Modelling the access control of students for lectures given by teachers
- When a student is attending a lecture, he/she can not attend another lecture
- When a teacher is lecturing, he/she is not lecturing another session.
- A student can not be lecturing without a teacher and when he is not attending a lecture, he is outside the classroom.
- When a teacher is ending a lecture, every student which is attending, is leaving the class room.
- When a student is not attending a lecture, he is free.

First step : identification of sets, constants, properties

- Sets : students, teachers
- **Property 1 :** When a student is attending a lecture, he/she can not attend another lecture
- **Property 2 :** When a teacher is lecturing, he/she is not lecturing another session.
- **Property 3 :**A student can not be lecturing without a teacher and when he is not attending a lecture, he is outside the classroom.
- **Property 4 :** When a teacher is ending a lecture, every student which is attending, is leaving the class room.
- **Property 5 :** When a student is not attending a lecture, he is free.

Second step : definition of state variables

- **The system model should be able to record the lecturing teachers and the attending students.**
- **The system model should be enough expressive to state when a given student is attending a lecture given by whom.**
- **Variable attending records students which atteding some lecture with a given teacher.**
- **Variable islecturing records teachers who are lecturing.**
- **Variable pause records sudents are not attending a lecture but are somewhere not in a lecture.**

Third step : properties of state variables

Expression of the invariant

$$
\begin{array}{l}
inv1 : attending \in STUDENTS \rightarrowtail\!\!\!\!\!+ TEACHERS \\
inv2 : islecturing \subseteq TEACHERS \\
inv3 : \forall e \cdot e \in STUDENTS \wedge e \in dom(attending) \\
\qquad\qquad\qquad\qquad \Rightarrow \quad attending(e) \in islecturing \\
inv4 : pause \subseteq STUDENTS \\
inv5 : pause \cap dom(attending) = \varnothing \\
inv6 : pause \cup dom(attending) = STUDENTS
\end{array}
$$

Checking proof obligations !

Fourth step : Updating state variables

UseCases

- **EVENT** INITIALISATION : initializing state variables
- **EVENT** startingattending : a group of students is moving from *pause* to *lecture*
- **EVENT** teachergivinglecture : a teacher is starting a new lecture
- **EVENT** teacherendinglecture : a teacher is halting the lecture
- **EVENT** studentleavinglecture : a group of students is moving from *lecture* to *pause*

Fourth step : Updating state variables

EVENT INITIALISATION
BEGIN
$act1: \ attending := \varnothing$
$act2: \ islecturing := \varnothing$
$act3: \ pause := STUDENTS$
END

Fourth step : Updating state variables

EVENT startingattending
ANY
e e is a student
p p is a teacher
WHERE
$grd1:\ e \in STUDENTS$
$grd3:\ p \in TEACHERS$
$grd4:\ p \in islecturing$
$grd2:\ e \notin dom(attending)$
THEN
$act1:\ attending(e) := p$
$act2:\ pause := pause \setminus \{e\}$
END

Fourth step : Updating state variables

EVENT teachergivinglecture
  ANY
    p
  WHERE
    $grd2: \ p \in TEACHERS$
    $grd1: \ p \notin islecturing$
  THEN
    $act1 \ islecturing := islecturing \cup \{p\}$
END

Fourth step : Updating state variables

EVENT teacherendinglecture
  ANY
    p
  WHERE
    $grd1: p \in TEACHERS$
    $grd2: p \in islecturing$
  THEN
    $act1: islecturing := islecturing \setminus \{p\}$

$$act3: \; attending := attending \setminus \{f \mapsto q | \left(\begin{array}{l} f \in STUDENTS \\ \land q \in TEACHERS \\ \land f \mapsto q \in attending \\ \land q = p \end{array} \right) \}$$

    $act2: pause := pause \cup \{f | f \in STUDENTS \land f \in attending^{-1}[\{p\}]\}$
  END

Fourth step : Updating state variables

EVENT studentleavinglecture
ANY
ge
WHERE
$grd1 : ge \subseteq dom(attending)$
$grd2 : ge \neq \varnothing$
THEN
$act1 : attending := ge \lhd attending$
$act2 : pause := pause \cup ge$
END

Mathematical tools for modelling systems

- **set theory : sets, relations, functions . . .**
- **transition systems**
- **predicate calculus**
- **decision procedures**
- **interactive theorem prover**

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. **Modelling state-based systems**
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

Modelling systems

- A **system** is **observed**
- Observation of things which are
  - either changing over the **time** (*variable*)
  - or stuttering over the **time** (*constant*)
- A system is characterized by a **state**
- A state is made up of contextual **constant informations** over the problem theory and of **modifiable flexible informations** over the system.

Changing state of system

A **flexible variable** x is observed at different instants :

$x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} x_2 \xrightarrow{\tau} x_3 \xrightarrow{\tau} \ldots \xrightarrow{\tau} x_i \xrightarrow{\tau} x_{i+1} \xrightarrow{\tau} \ldots$

τ hides effectives changes of state or actions or events

$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$

Occurences of e τ can be added between two instants ie stuttering steps :

$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\tau} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\tau} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$

Changing state of system

A **flexible variable** x is observed at different instants :
$x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} x_2 \xrightarrow{\tau} x_3 \xrightarrow{\tau} \ldots \xrightarrow{\tau} x_i \xrightarrow{\tau} x_{i+1} \xrightarrow{\tau} \ldots$
τ hides effectives changes of state or actions or events
$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$
Occurences of e τ can be added between two instants ie stuttering steps :
$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\tau} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\tau} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$

Changing state of system

A **flexible variable** x is observed at different instants :
$x_0 \stackrel{\tau}{\rightarrow} x_1 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_3 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_{i+1} \stackrel{\tau}{\rightarrow} \ldots$
τ hides effectives changes of state or actions or events
$x_0 \stackrel{\alpha_1}{\rightarrow} x_1 \stackrel{\alpha_2}{\rightarrow} x_2 \stackrel{\alpha_3}{\rightarrow} x_3 \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} x_i \stackrel{\alpha_{i+1}}{\rightarrow} x_{i+1} \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$
Occurences of e τ can be added between two instants ie stuttering steps :
$x_0 \stackrel{\alpha_1}{\rightarrow} x_1 \stackrel{\alpha_2}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\alpha_3}{\rightarrow} x_3 \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_i \stackrel{\alpha_{i+1}}{\rightarrow} x_{i+1} \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$

Changing state of system

A **flexible variable** x is observed at different instants :

$x_0 \stackrel{\tau}{\rightarrow} x_1 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_3 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_{i+1} \stackrel{\tau}{\rightarrow} \ldots$

τ hides effectives changes of state or actions or events

$x_0 \stackrel{\alpha_1}{\rightarrow} x_1 \stackrel{\alpha_2}{\rightarrow} x_2 \stackrel{\alpha_3}{\rightarrow} x_3 \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} x_i \stackrel{\alpha_{i+1}}{\rightarrow} x_{i+1} \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$

Occurences of e τ can be added between two instants ie **stuttering steps** :

$x_0 \stackrel{\alpha_1}{\rightarrow} x_1 \stackrel{\alpha_2}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\alpha_3}{\rightarrow} x_3 \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_i \stackrel{\alpha_{i+1}}{\rightarrow} x_{i+1} \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$

Changing state of system

A **flexible variable** x is observed at different instants :
$x_0 \xrightarrow{\tau} x_1 \xrightarrow{\tau} x_2 \xrightarrow{\tau} x_3 \xrightarrow{\tau} \ldots \xrightarrow{\tau} x_i \xrightarrow{\tau} x_{i+1} \xrightarrow{\tau} \ldots$
τ hides effectives changes of state or actions or events
$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$
Occurences of e τ can be added between two instants ie **stuttering steps** :
$x_0 \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} x_2 \xrightarrow{\tau} x_2 \xrightarrow{\alpha_3} x_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_i} x_i \xrightarrow{\tau} x_i \xrightarrow{\alpha_{i+1}} x_{i+1} \xrightarrow{\alpha_{i+2}} \ldots$

Properties of system

A **safety** property S over x states that something will not happen : S(x) means that S holds for x

An **invariant** property I over x states a strong safety property

$x_0 \stackrel{\alpha_1}{\rightarrow} x_1 \stackrel{\alpha_2}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\alpha_3}{\rightarrow} x_3 \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_i \stackrel{\alpha_{i+1}}{\rightarrow} x_{i+1} \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$

$(S(x_0) \stackrel{\alpha_1}{\rightarrow} S(x_1) \stackrel{\alpha_2}{\rightarrow} S(x_2) \stackrel{\tau}{\rightarrow} S(x_2) \stackrel{\alpha_3}{\rightarrow} S(x_3) \stackrel{\alpha_4}{\rightarrow} \ldots \stackrel{\alpha_i}{\rightarrow} S(x_i) \stackrel{\tau}{\rightarrow} S(x_i) \stackrel{\alpha_{i+1}}{\rightarrow} S(x_{i+1}) \stackrel{\alpha_{i+2}}{\rightarrow} \ldots$

or equivalently $\forall i \in \mathbb{N} : S(x_i)$

Checking the relation

- You can check for every i in $\mathbb{N}$ that $S(x_i)$ is true but it can be long if states are different
- You can compute an abstraction of the set of states
- You can try to prove and for instance the induction principle may be usefull
- So be carefull and improve your modelling before to run the checker
- Use the induction

State properties of a system

- A state property namely $P(x)$ is a first order predicate with free variables x, where x is a flexible variable.
- A flexible variable x has a current value x, a next value x', an initial value x_0 and possibly a final value x_f.
- A predicate $P(x)$ is considered as a set of values v such that $P(v)$ holds : set-theoretical interpretation

Safety Property

A safety property states that nothing bad can happen.

Example

Safety Properties

- Partial correctness a component is correct wit respect to a precondition and a postcondition.
- No Run Time Error any software action or event does not produce a run-time error as overflow, division by zero ...
- Mutual exclusion a set of processes share common ressources, a printer is shared by users, ...
- Deadlock freedom the system is never blocked, there is always at least one next state, ...

Relation/action over states

- An action α over states is a relation between values of state variables **before** and values of variables **after**

$$\alpha(x, x') \text{ or } x \xrightarrow{\alpha} x'$$

- Flexible variable x has two values x and x'.
- Priming flexible variables is borrowed from TLA
- **Hypothesis 1** : **Values of x belongs to a set of values called** $\textsc{Values}$ and defines the context of the system.
- **Hypothesis 2** : **Relations over x and x' belong to a set of relations $\{r_0, \ldots, r_n\}$**

Operational model of a system

- A system $\mathcal{S}$ is observed with respect to flexible variables x.
- Flexible variables x of $\mathcal{S}$ are modified according to a finite set of relations over the set of values $\textsc{Values} : \{r_0, \ldots, r_n\}$
- $\textsc{Init}(x)$ denotes the set of possible intial values for x.

$$\mathcal{OMS} = (x, \textbf{Values}, \textbf{Init}(x), \{r_0, \ldots, r_n\})$$

Safety and invariance of system

- **Hypothesis 3** : $\mathcal{OMS} = (x, \textsc{Values}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$
- **Hypothesis 4** : $x \longrightarrow x' \triangleq (x\ r_0\ x') \vee \ldots \vee (x\ r_n\ x')$
- $\mathrm{I(x)}$ is inductively invariant for a system called $\mathcal{S}$, if
  $\begin{cases} \forall x \in \textsc{Values} : \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ \forall x, x' \in \textsc{Values} : \mathrm{I}(x) \wedge x \longrightarrow x' \Rightarrow \mathrm{I}(x') \end{cases}$
  I(x) is called an invariant in B
- $\mathrm{Q(x)}$ is a safety property for a system called $\mathcal{S}$, if
  $\forall x, y \in \textsc{Values} : \textsc{Init}(x) \wedge x \xrightarrow{\star} y \Rightarrow \mathrm{Q}(y)$
  Q(x) is called a theorem in B

Modelling systems : first attempt

```
MODEL
   m
...
...
...
VARIABLES
   x
INVARIANT
   I(x)
THEOREMS
   Q(x)
INITIALISATION
   Init(x)
EVENTS
   {r_0, ..., r_n}
END
```

- A model has a name m
- Flexibles variables x are declared
- $I(x)$ provides informations over x
- $Q(x)$ provides informations over x

Checking safety properties of the model

- $\forall x_0, x \in \textsc{Values} : \textsc{Init}(x_0) \wedge x_0 \xrightarrow{\star} x \Rightarrow \mathrm{Q}(x)$
- **Solution 1** Writing a procedure checking $\textsc{Init}(x_0) \wedge x_0 \xrightarrow{\star} x \Rightarrow \mathrm{Q}(x)$ for each pair $x_0, x \in \textsc{Values}$, when $\textsc{Values}$ is finite and small.
- **Solution 2** Writing a procedure checking $\textsc{Init}(x_0) \wedge x_0 \xrightarrow{\star} x \Rightarrow \mathrm{Q}(x)$ for each pair $x_0, x \in \textsc{Values}$, by constructing an abstraction of $\textsc{Values}$.
- **Solution 3** Writing a proof for $\forall x_0, x \in \textsc{Values} : \textsc{Init}(x_0) \wedge x_0 \xrightarrow{\star} x \Rightarrow \mathrm{Q}(x)$.

Defining an induction principle for an operational model

$$\textbf{(I)}\ \forall x_0, x \in \textbf{Values} : \textbf{Init}(x_0) \wedge x_0 \xrightarrow{\star} x \Rightarrow \textbf{Q}(x)$$

if, and only if,

(II) there exists a state property $\mathrm{I}(x)$ such that :

$$\forall x_0, x, x' \in \textbf{Values} : \left\{ \begin{array}{ll} (1) & \textbf{Init}(x_0) \Rightarrow \textbf{I}(x_0) \\ (2) & \textbf{I}(x) \Rightarrow \textbf{Q}(x) \\ (3) & \textbf{I}(x) \wedge x \longrightarrow x' \Rightarrow \textbf{I}(x') \end{array} \right.$$

if, and only if,

(III) there exists a state property $\mathrm{I}(x)$ such that :

$$\forall x_0, x, x' \in \textbf{Values} : \left\{ \begin{array}{ll} (1) & \textbf{Init}(x_0) \Rightarrow \textbf{I}(x_0) \\ (2) & \textbf{I}(x) \Rightarrow \textbf{Q}(x) \\ (3) & \forall i \in \{0, \ldots, n\} : \textbf{I}(x) \wedge x \ r_i \ x' \Rightarrow \textbf{I}(x') \end{array} \right.$$

Modelling systems : second attempt

MODEL
m
...
...
...
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\{r_0, \ldots, r_n\}$
END

- $\forall x_0 \in \textsc{Values} : \textsc{Init}(x_0) \Rightarrow \mathrm{I}(x_0)$
- $\forall x, x' \in \textsc{Values} : \forall i \in \{0, \ldots, n\} :$
  $\mathrm{I}(x) \wedge x \; r_i \; x' \Rightarrow \mathrm{I}(x')$
- $\forall x \in \textsc{Values} : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

Modelling systems : last attempt ?

MODEL
  m
?
?
?
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\{r_0, \ldots, r_n\}$
END

- What are the environment of the proof for properties ?
- What are theories ?
- How are defining the static objects ?

Modelling systems : last attempt !

MODEL
m
$\Gamma(m)$
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\{r_0, \ldots, r_n\}$
END

- $\Gamma(m)$ defines the static environment for the proofs related to m.
- $\Gamma(m) \vdash \forall x \in \textsc{Values} : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$
- $\forall i \in \{0, \ldots, n\}$:
  $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \mathrm{I}(x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(x')$
- $\Gamma(m) \vdash \forall x \in \textsc{Values} : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

Events System Models

An **event system model** is made of

- State **constants** and state **variables** constrained by a state **invariant**
- A finite set of **events**

Proofs ensures the consistency between the invariant and the events

An event system model can be **refined**

Proofs must ensure the correctness of refinement

MODEL *FACTORIAL_EVENTS*

`Static Part` *context*

CONSTANTS

$factorial, m$

AXIOMS

$m \in \mathbb{N} \ \wedge \ factorial \in \ \mathbb{N} \ \leftrightarrow \ \mathbb{N} \ \wedge \ 0 \mapsto 1 \in factorial \ \wedge$
$\forall (n, fn).(n \mapsto fn \in factorial \Rightarrow n+1 \mapsto (n+1) * fn \in factorial) \ \wedge$

$$\forall f \cdot \left(\begin{array}{l} f \in \mathbb{N} \leftrightarrow \mathbb{N} \wedge \\ 0 \mapsto 1 \in f \wedge \\ \forall (n, fn).(n \mapsto fn \in f \Rightarrow n+1 \mapsto (n+1) \times fn \in f) \\ \Longrightarrow \\ factorial \subseteq f \end{array} \right)$$

`Dynamic Part` *machine*

VARIABLES

$result, ok$

INVARIANT

$result \in \mathbb{N}$
$ok \in \mathbb{B}$
$ok = TRUE \Rightarrow result = factorial(n)$

THEOREMS

$factorial \ \in \ \mathbb{N} \ \longrightarrow \ \mathbb{N}$;
$factorial(0) = 1$;
$\forall n.(n \ \in \ \mathbb{N} \ \Rightarrow \ factorial(n+1) = (n+1) \times factorial(n))$

INITIALISATION

$result :\in \mathbb{N}$
$ok := FALSE$

EVENTS

$computation =$ **ANY** $ok = FALSE$ **THEN** $result, ok := factorial(m), TRUE$ **END**

END

Modelling systems relations as events

MODEL
  m
`Static Part` *context*

> **SETS**
> s
> **CONSTANTS**
> c
> **AXIOMS**
> $P(s, c)$
> **THEOREMS**
> $Q(s, c)$

`Dynamic Part` *machine*
VARIABLES
  x
INVARIANT
  $I(s, c, x)$
THEOREMS
  $S(s, c, x)$
INITIALISATION
  $Init(s, c, x)$
EVENTS
  $\{r_1, \ldots, r_n\}$
END

- $\Gamma(m)$ defines the static environment for the proofs related to m from s, c and $P(s, c)$ and $\Gamma(m)$ is defined from the static part.
- $\Gamma(m) \vdash Q(s, c)$
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \textsc{Init}(s, c, x) \Rightarrow \mathrm{I}(s, c, x)$
- $\forall i \in \{1, \ldots, n\}$:
  $\Gamma(m) \vdash \forall x, x' \in \textsc{Values}$:
  $\mathrm{I}(s, c, x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(s, c, x')$
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \mathrm{I}(s, c, x) \Rightarrow \mathrm{S}(s, c, x)$

Modelling systems relations as events

CONTEXT
$context\_name$
SETS
s
CONSTANTS
c
AXIOMS
$P(s,c)$
THEOREMS
$Q(s,c)$

MACHINE
m
SEES
$context\_name$
VARIABLES
x
INVARIANT
$I(s,c,x)$
THEOREMS
$S(s,c,x)$
INITIALISATION
$Init(s,c,x)$
EVENTS
$\{e_1,\ldots,e_n\}$
END

- $\Gamma(m)$ defines the static environment for the proofs related to m from s, c and $P(s,c)$ and $\Gamma(m)$ is defined from the static part.
- $\Gamma(m) \vdash Q(s,c)$
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \textsc{Init}(s,c,x) \Rightarrow \mathrm{I}(s,c,x)$
- $\forall i \in \{0,\ldots,n\} : r_i(x,x') \triangleq BA(e_i)(s,c,x,x')$
- $\forall i \in \{0,\ldots,n\}$:
  $\Gamma(m) \vdash \forall x, x' \in \textsc{Values}$:
  $\mathrm{I}(x) \wedge BA(e_i)(s,c,x,x') \Rightarrow \mathrm{I}(s,c,x')$
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \mathrm{I}(x) \Rightarrow \mathrm{S}(s,c,x)$

Modelling systems

step 1 : Understanding the **problem** to solve
step 2 : **Organizing** requirements and extracting properties
step 3 : Writing a first very **abstract** system model
step 4 : Consulting the requirements and **adding** a new detail in the current model by **refinement**
step 5 : Either the model is enough detailed and the process stops, or the model is not yet enough concrete and the step 4 is repeated.

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. **The Event B modelling language**
8. Examples of Event B models
9. Summary on Events

Expressing models in the event B notation

- Models are defined in two ways :
  - an abstract machine
  - a refinement of an existing model
- Models use **constants** which are defined in structures called **contexts**
- B structures are related by the three possible relations :
  - the **sees** relationship for expressing the use of constants, sets satisfying axioms and theorems.
  - the **extends** relationship for expressing the extension of contexts by adding new constants and new sets
  - the **refines** relationship stating that a B model is refined by another one.

Machines and contexts

Machines

- **REFINES**
- **SEES** a context
- **VARIABLES** of the model
- **INVARIANTS** satisfied by the variables
- **THEOREMS** satisfied by the variables
- **EVENTS** modifying the variables
- **VARIANT**

Contexts

- **EXTENDS** another context
- **SETS** declares new sets
- **CONSTANTS** define a list of constants
- **AXIOMS** define the properties of constants and sets
- **THEOREMS** list the theorems which should be derived from axioms

Machines en Event B

MACHINE
m
REFINES
am
SEES
c
VARIABLES
u
INVARIANTS
$I(u)$
THEOREMS
$Q(u)$
VARIANT
$< variant >$
EVENTS
$< event >$
END

- $\Gamma(m)$: environment for the machine m defined by the context c
- $\Gamma(m) \vdash \forall u \in \textsc{Values} : \textsc{Init}(u) \Rightarrow \mathrm{I}(u)$
- For each event e in E :
  $\Gamma(m) \vdash \forall u, u' \in \textsc{Values} : \mathrm{I}(u) \wedge BA(e)(u, u') \Rightarrow \mathrm{I}(u')$
- $\Gamma(m) \vdash \forall u \in \textsc{Values} : \mathrm{I}(u) \Rightarrow \mathrm{Q}(u)$

Contexts in Event B

CONTEXTS
  c
EXTENDS
  ac
SETS
  s
CONSTANTS
  c
AXIOMS
  $ax1 : \ldots$
THEOREMS
  $th1 : \ldots$
END

- ac : c is extending ac and add new features
- s : sets are defined either by intension or by extension
- c : constants are defined and
- axioms characterize constants and sets
- theorems are derived from axioms in the current context

Events

Event : E	Before-After Predicate
BEGIN $x :\| P(x,x')$ **END**	$P(x,x')$
WHEN $G(x)$ **THEN** $x :\| P(x,x')$ **END**	$G(x) \quad \wedge \quad P(x,x')$
ANY t **WHERE** $G(t,x)$ **THEN** $x :\| P(x,x',t)$ **END**	$\exists t \cdot (\, G(t,x) \quad \wedge \quad P(x,x',t)\,)$

Guards of event

Event : E	Guard : grd(E)
BEGIN S **END**	$TRUE$
WHEN $G(x)$ **THEN** T **END**	$G(x)$
ANY t **WHERE** $G(t,x)$ **THEN** T **END**	$\exists t \cdot G(t,x)$

Proof obligations for a B model

	Proof obligation
(INV1)	$\Gamma(s,c) \vdash Init(x) \Rightarrow I(x)$
(INV2)	$\Gamma(s,c) \vdash I(x) \wedge BA(e)(x,x') \Rightarrow I(x')$
(DEAD)	$\Gamma(s,c) \vdash I(x) \Rightarrow (\mathsf{grd}(e_1) \vee \ldots \mathsf{grd}(e_n))$
(SAFE)	$\Gamma(s,c) \vdash I(x) \Rightarrow A(x)$
(FIS)	$\Gamma(s,c) \vdash I(x) \wedge \mathsf{grd}\,(E) \Rightarrow \exists x' \cdot P(x,x')$

Current Summary

1 Documentation

2 Introduction by Problems

3 Overview of formal techniques and formal methods

4 Modelling Language

5 A Simple Example : Management of Students and Teachers

6 Modelling state-based systems

7 The Event B modelling language

8 Examples of Event B models

9 Summary on Events

The factorial model

CONTEXT
$fonctions$
CONSTANTS
$factorial, n$
AXIOMS
$ax1 : n \in \mathbb{N}$
$ax2 : factorial \in \mathbb{N} \leftrightarrow \mathbb{N}$
$ax3 : 0 \mapsto 1 \in factorial$
$ax4 : \forall (i, fn).(i \mapsto fn \in factorial \Rightarrow i+1 \mapsto (i+1) * fi \in factorial) \wedge$

$$\forall f \cdot \left(\begin{array}{l} f \in \mathbb{N} \leftrightarrow \mathbb{N} \wedge \\ 0 \mapsto 1 \in f \wedge \\ \forall (n, fn).(n \mapsto fn \in f \Rightarrow n+1 \mapsto (n+1) \times fn \in f) \\ \Longrightarrow \\ factorial \subseteq f \end{array} \right)$$

END

The factorial model

MACHINE
  $specification$
SEES $fonctions$
VARIABLES
  $resultat$
INVARIANT
  $resultat \in \mathbb{N}$
THEOREMS
  $th1 : factorial \in \mathbb{N} \longrightarrow \mathbb{N}$;
  $th2 : factorial(0) = 1$;
  $th3 : \forall n.(n \in \mathbb{N} \Rightarrow factorial(n+1) = (n+1) \times factorial(n))$
INITIALISATION
  $resultat :\in \mathbb{N}$
EVENTS
  $computing1 =$ **BEGIN** $resultat := factorial(n)$ **END**
END

Communications between agents

MACHINE *agents*
SEES *data*
VARIABLES

$sent$
got
$lost$

INVARIANTS

$inv1 : sent \subseteq AGENTS \times AGENTS$
$inv2 : got \subseteq AGENTS \times AGENTS$
$inv4 : (got \cup lost) \subseteq sent$
$inv6 : lost \subseteq AGENTS \times AGENTS$
$inv7 : got \cap lost = \varnothing$

INITIALISATION
BEGIN

$act1 : sent := \varnothing$
$act2 : got := \varnothing$
$act4 : lost := \varnothing$

END

Communications between agents

EVENT sending a message
ANY
  a, b
WHERE
  $grd11 : a \in AGENTS$
  $grd12 : b \in AGENTS$
  $grd1 : a \mapsto b \notin sent$
THEN
  $act11 : sent := sent \cup \{a \mapsto b\}$
END

EVENT getting a message
ANY
  a, b
WHERE
  $grd11 : a \in AGENTS$
  $grd12 : b \in AGENTS$
  $grd13 : a \mapsto b \in sent \setminus (got \cup lost)$
THEN
  $act11 : got := got \cup \{a \mapsto b\}$
END

Communications between agents

EVENT loosing a messge
ANY
a
b
WHERE $grd1 : a \in AGENTS$
$grd2 : b \in AGENTS$
$grd3 : a \mapsto b \in sent \setminus (got \cup lost)$
THEN
$act1 : lost := lost \cup \{a \mapsto b\}$
END

CONTEXTS
$data$
SETS
$MESSAGES$
$AGENTS$
$DATA$
CONSTANTS
n
$infile$
AXIOMS
$axm1 : n \in \mathbb{N}$
$axm2 : n \neq 0$
$axm3 : infile \in 1 \,..\, n \rightarrow DATA$
END

Current Summary

1. Documentation
2. Introduction by Problems
3. Overview of formal techniques and formal methods
4. Modelling Language
5. A Simple Example : Management of Students and Teachers
6. Modelling state-based systems
7. The Event B modelling language
8. Examples of Event B models
9. Summary on Events

General form of an event

```
EVENT e
  ANY  t
  WHERE
    G(c, s, t, x)
  THEN
    x :| (P(c, s, t, x, x'))
  END
```

- c et s are constantes and visible sets by e
- x is a state variable or a list of variabless
- $G(c, s, t, x)$ is the condition for observing e.
- $P(c, s, t, x, x')$ is the assertion for the relation over x and x'.
- $BA(e)(c, s, x, x')$ is the *before-after* relationship for e and is defined by $\exists t.G(c, s, t, x) \ \wedge \ P(c, s, t, x, x')$.

General form of proof obligations for an event e

Proofs obligations are simplified when they are generated by the module called POG and goals in sequents as $\Gamma \vdash G$:

1. $\Gamma \vdash G_1 \wedge G_2$ is decomposed into the two sequents $\begin{array}{l}(1)\Gamma \vdash G_1 \\ (2)\Gamma \vdash G_2\end{array}$
2. $\Gamma \vdash G_1 \Rightarrow G_2$ is transformed into the sequent $\Gamma, G_1 \vdash G_2$

Proof obligations in Rodin

- *INIT/I/INV* : $C(s,c), INIT(c,s,x) \vdash I(c,s,x)$
- *e/I/INV* : $C(s,c), I(c,s,x), G(c,s,t,x), P(c,s,t,x,x') \vdash I(c,s,x')$
- *e/act/FIS* : $C(s,c), I(c,s,x), G(c,s,t,x) \vdash \exists x'.P(c,s,t,x,x')$

notation

- Chapter Event B
- The Event B Modelling Notation Version 1.4
- The Event-B Mathematical Language 2006
- User Manual of the RODIN PLatform