Modelling Software-based Systems

Lecture 1

The Modelling Language Event-B

Master Informatique

Dominique Méry
Telecom Nancy,Université de Lorraine

23 octobre 2024

dominique.mery@loria.fr

General Summary

Current Summary

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
- **Using the Arche platform of UL and accessing the course MOSOS with password mery2020**

Current Summary

RTE

```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;
}
```

RTE with frama-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 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).

Characterization of a System (II) cont.

What are the main properties to consider for formalization ?

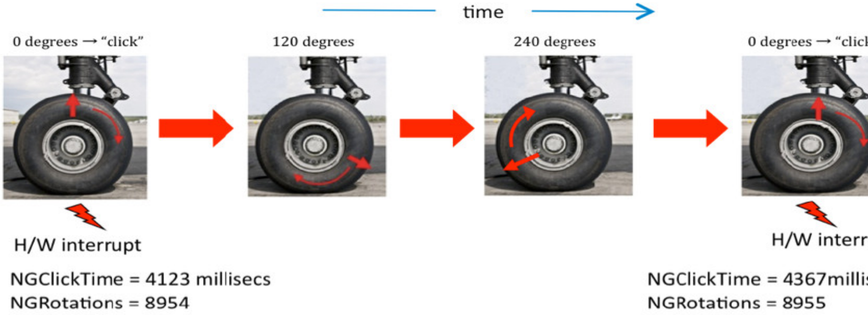
- Two different types of data :
  - counters with modulo semantics
  - non-negative values for time, distance, and velocity
- Two dimensions : *distance* and *time*
- Many units : distance (`inches, kilometers, miles`), time (`milliseconds, hours`), velocity (`kph, mph`)
- And interaction among components

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 $mphTokph \in \mathtt{MPH} \rightarrowtail\!\!\!\!\rightarrow \mathtt{KPH}$

Sample Velocity Estimation



WHEEL_DIAMETER = 22 inches
PI = 3.14

12 inches/foot
5280 feet/mile

estimatedGroundVelocity = distance travel/elapsed time
= ((3.14 * 22)/(12*5280))/((4367-4123)/(1000*3600
= 16 mph

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

```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;
}
```

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

Context and Objectives

- Software Systems assist our dayly lifes
- Questions on dependability and security assurance shoud be addressed
- Questions on certification with resepct to norms and standards
- Improving the life-cycle development for addressing these questions

Problem Definition

Critical System

Critical systems are systems in which defects could have a dramatic impact on human life or the environment.

System failure

Software failure or fault of complex systems is the major cause in the software crisis. For example,

- Therac-25 (1985-1987) : six people overexposed through radiation.
- Cardiac Pacemaker (1990-2002) : 8834 pacemakers were explanted.
- Insulin Infusion Pump (IIP) (2010) : 5000 adverse events.

Critical Systems

- Safety-critical systems : A system whose failure may result in injury, loss of life or serious environmental damage. An example of a safety-critical system is a control system for a chemical manufacturing plant.
- Mission-critical systems : A system whose failure may result in the failure of some goal-directed activity. An example of a mission-critical system is a navigational system for a spacecraft.
- Business-critical systems : A system whose failure may result in very high costs for the business using that system. An example of a business-critical system is the customer accounting system in a bank.

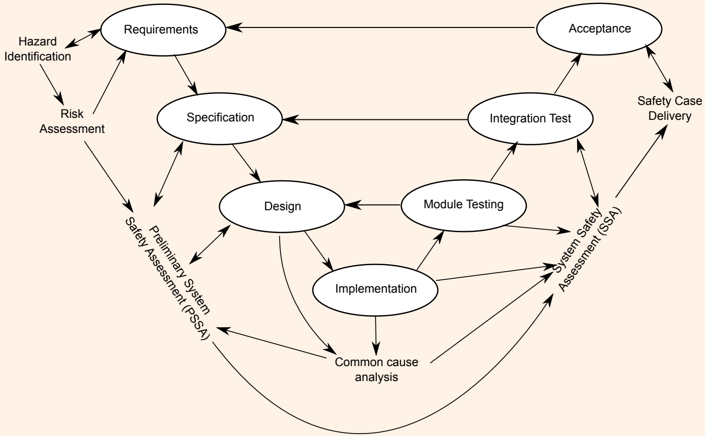
The high costs of failure of critical systems means that trusted methods and techniques must be used for development. C

Legacy systems

- Legacy systems are socio-technical computer-based systems that have been developed in the past, often using older or obsolete technology.
- Legacy systems include not only hardware and software but also legacy processes and procedures ;old ways of doing things that are difficult to change because they rely on legacy software. Changes to one part of the system inevitably involve changes to other components.
- Legacy systems are often business-critical systems. They are maintained because it is too risky to replace them.
- For example, for most banks the customer accounting system was one of their earliest systems.

Traditional System Engineering Approach

Spiral Model, Waterfall Model, V-Shaped Model, etc.



The Cleanroom Model

- The Cleanroom method, developed by the late Harlan Mills and his colleagues at IBM and elsewhere, attempts to do for software what cleanroom fabrication does for semiconductors : to achieve quality by keeping defects out during fabrication.
- In semiconductors, dirt or dust that is allowed to contaminate a chip as it is being made cannot possibly be removed later.
- But we try to do the equivalent when we write programs that are full of bugs, and then attempt to remove them all using debugging.

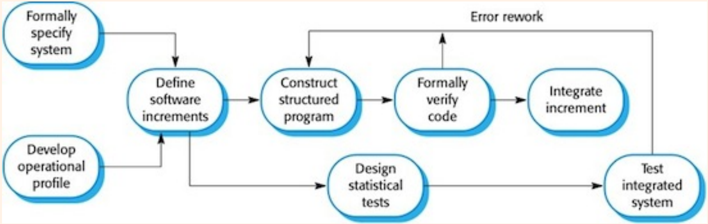
The Cleanroom Method

The Cleanroom method, then, uses a number of techniques to develop software carefully, in a well-controlled way, so as to avoid or eliminate as many defects as possible before the software is ever executed. Elements of the method are :

- specification of all components of the software at all levels ;
- stepwise refinement using constructs called "box structures" ;
- verification of all components by the development team ;
- statistical quality control by independent certification testing ;
- no unit testing, no execution at all prior to certification testing.

The Cleanroom approach to software development is based on five key strategies :

- Formal specification : The software to be developed is formally specified. A state-transition model which shows system responses to stimuli is used to express the specification.
- Incremental development : The software is partitioned into increments which are developed and validated separately using the Cleanroom process. These increments are specified, with customer input, at an early stage in the process. ¡/li¿
- Structured programming : Only a limited number of control and data abstraction constructs are used. The program development process is a process of stepwise refinement of the specification. A limited number of constructs are used and the aim is to apply correctness-preserving transformations to the specification to create the program code.
- Static verification : The developed software is statically verified using rigorous software inspections. There is no unit or module testing process for code components.
- Statistical testing of the system : The integrated software increment is tested statistically (see Chapter XX), to determine its reliability. These statistical tests are based on an operational profile which is developed in parallel with the system specification.


Formally specify system
Define software increments
Develop operational profile
Construct structured program
Formally verify code
Integrate increment
Error rework
Design statistical tests
Test integrated system

Modelling systems

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

- c defines the static environment $\Gamma(m)$ for the proofs related to m : sets, constants, axioms, theorems.
- $\Gamma(m) \vdash \forall x, x' \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$
- $\forall e$:
  $\Gamma(m) \vdash \forall x, x' \in Values$:
  $\mathrm{I}(x) \wedge G(x,u) \wedge R(u,x,x') \Rightarrow \mathrm{I}(x')$
- $\Gamma(m) \vdash \forall x, x' \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$
- e
  ANY
    u
  WHERE
    $G(x,u)$
  THEN
    $x :| (R(u,x,x')$
  END

  or $x \xrightarrow{e} x'$

Refinement of a model by another one (I)

$$
\begin{tikzcd}
I(x) & x \arrow[r, "ae(x{,}x')"] \arrow[d, leftrightarrow] & x' \arrow[d, leftrightarrow] & I(x') \\
J(x,y) & y \arrow[r, "ce(y{,}y')"] & y' & J(x',y')
\end{tikzcd}
$$

$$
\begin{tikzcd}
I(x) & AM \arrow[r, "\text{SEES}"] & AC & \mathcal{T}h_i \\
J(x,y) & CM \arrow[u, "\text{REFINES}"] \arrow[r, "\text{SEES}"] & CC \arrow[u, "\text{EXTENDS}"] & \mathcal{T}h_{i+1}
\end{tikzcd}
$$

Refinement of a model by another one (II)

$$
\begin{tikzcd}
I(x) & x \arrow[r, "x=x'"] \arrow[d, leftrightarrow] & x' \arrow[d, leftrightarrow] & I(x') \\
J(x,y) & y \arrow[r, "ce(y{,}y')"] & y' & J(x',y')
\end{tikzcd}
$$

$$
\begin{tikzcd}
I(x) & & x & & I(x) \\
J(x,y) & y \arrow[ur] \arrow[rr, "ce(y{,}y')"] & & y' \arrow[ul] & J(x,y')
\end{tikzcd}
$$

Context and Objectives

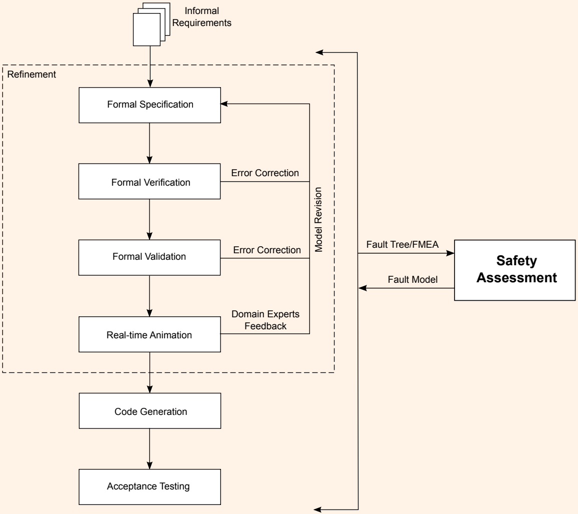
Context

Developing a life-cycle methodology combining the refinement approach with various tools including verification tool, model checker tool, real-time animator and finally, producing the source code into many languages using automatic code generation tools.

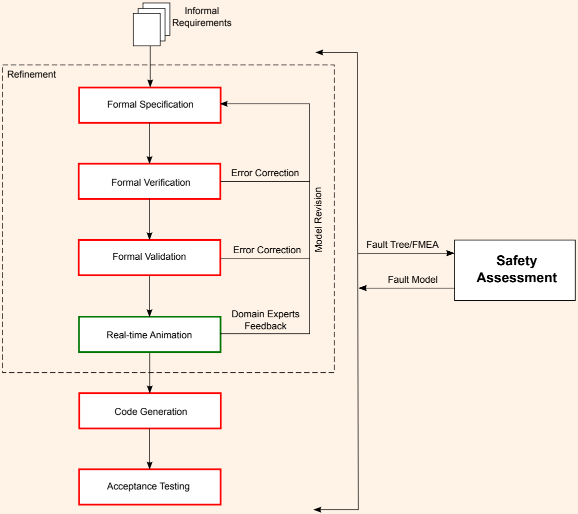
Objectives

- To establish a unified theory for the critical system development.
- To build a set of tools for supporting new development life-cycle methodology.
- To develop a closed-loop system for verification purpose.
- Graphical based refinement technique to handle the complexity of the system.
- To satisfy requirements and metrics for certifiable assurance and safety.
- To support evidence-based certification.

Critical System Development Life-Cycle Methodology



Critical System Development Life-Cycle Methodology



Overview of Methodology

Methodology

- Informal Requirements

  (Restricted form of natural language)
- Formal Specification

  (Modeling language like Event-B , Z, ASM, VDM, TLA+ etc.)
- Formal Verification

  (Theorem Prover Tools like PVS, Z3, SAT, SMT Solver etc.)
- Formal Validation

  (Model Checker Tools like ProB, UPPAAL , SPIN, SMV etc.)
- Real-time Animation

  (Our proposed approach...Real-Time Animator)
- Code Generation

  (Our proposed approach...EB2ALL : EB2C, EB2C++, EB2J, EB2C#)
- Acceptance Tesing

  (Failure Mode, Effects and Critically analysis(FMEA and FMEA), System Hazard Analyses(SHA))

Real-Time Animator

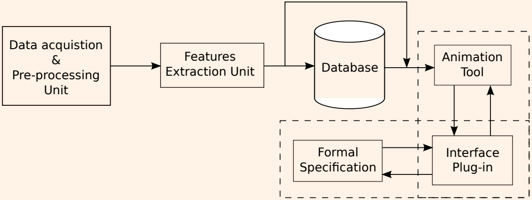
What is Real-time Animator ?

- Visual representation of formal model using real time data set.

Why should we use Formal Model Animator ?

- To validate system behavior according to the stakeholders
- To express formal models for non-mathematical domain experts
- To discover the error in the early stage of system development (Traceability)

Proposed Architecture



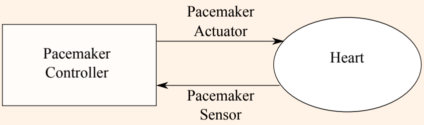
Case Study : Cardiac Pacemaker

Cardiac Pacemaker

A cardiac pacemaker is an electronic device implanted in the body to regulate the heart beat.

1. Informal Requirements are available at McMaster University (SQRL).
2. One and Two-electrode pacemaker development using refinement-based incremental development.
3. Cover possible operating modes of pacemaker (i.e. Sensing threshold value, Hysteresis mode (ON and OFF) and Rate modulation)
4. Refinements relation among modes with different parameters.
5. Model checker helps to analyze behavior of the formal specification according to the medical experts.

System : Heart $\oplus$ Pacemaker

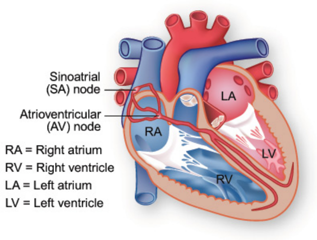


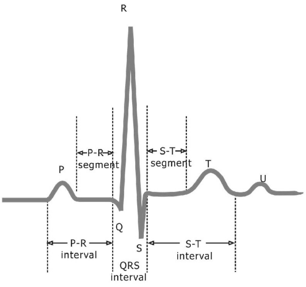
Bradycardia Operating Modes

Operating Modes

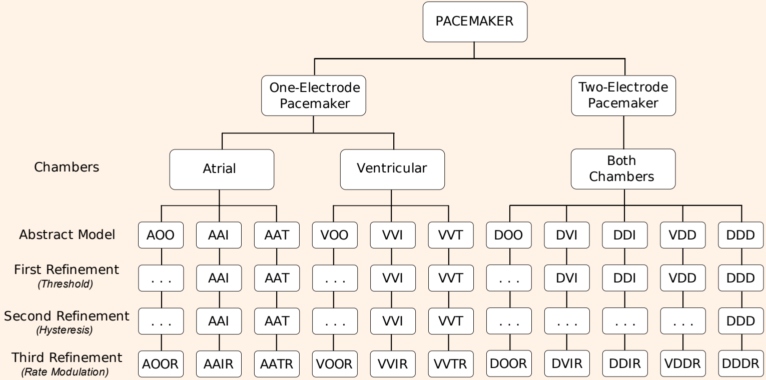
Category	Chambers Paced	Chambers Sensed	Response to Sensing	Rate Modulation
Letters	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **T**-Triggered **I**-Inhibited **D**-Dual(T+I)	**R**-Rate Modulation

i.e. AOO, VOO, AAI, AAT, VVI, VVT, AATR,VVTR, AOOR etc...

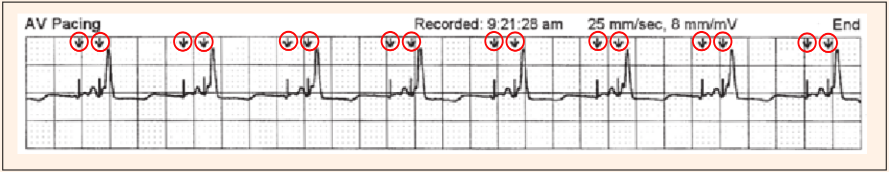




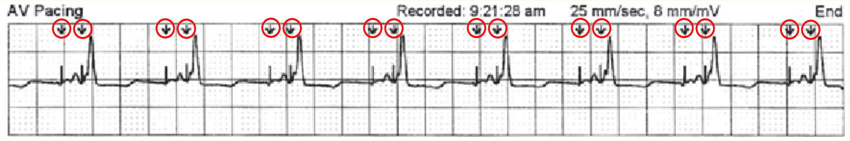
One and Two-Electrode Pacemaker



DDD Pacing Modes

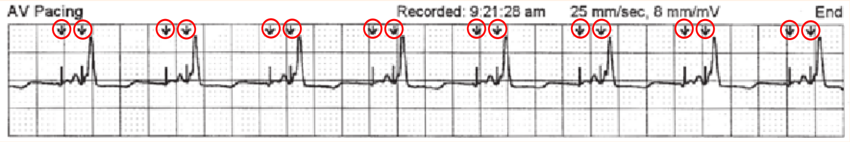


DDD Pacing Modes



$$
\begin{array}{l}
axm1 : LRL \in 30 \,..\, 175 \\
axm2 : URL \in 50 \,..\, 175 \\
axm3 : URI \in \mathbb{N}_1 \ \land URI = 60000/URL \\
axm4 : LRI \in \mathbb{N}_1 \ \land LRI = 60000/LRL \\
axm5 : status = \{ON, OFF\} \\
axm6 : FixedAV \in 70 \,..\, 300 \\
axm7 : ARP \in 150 \,..\, 500 \\
axm8 : VRP \in 150 \,..\, 500 \\
axm9 : PVARP \in 150 \,..\, 500 \\
axm10 : V\_Blank \in 30 \,..\, 60 \\
\ldots
\end{array}
$$

DDD Pacing Modes



$axm1 : LRL \in 30 \,..\, 175$
$axm2 : URL \in 50 \,..\, 175$
$axm3 : URI \in \mathbb{N}_1 \;\; \land URI = 60000/URL$
$axm4 : LRI \in \mathbb{N}_1 \;\; \land LRI = 60000/LRL$
$axm5 : status = \{ON, OFF\}$
$axm6 : FixedAV \in 70 \,..\, 300$
$axm7 : ARP \in 150 \,..\, 500$
$axm8 : VRP \in 150 \,..\, 500$
$axm9 : PVARP \in 150 \,..\, 500$
$axm10 : V\_Blank \in 30 \,..\, 60$
$\ldots$

$inv1 : PM\_Actuator\_A \in status$
$inv2 : PM\_Sensor\_A \in status$
$inv5 : Pace\_Int \in URI \,..\, LRI$
$inv6 : sp \in 1 \,..\, Pace\_Int$
$inv7 : last\_sp \geq PVARP \land last\_sp \leq Pace\_Int$

$inv11 : sp < VRP \land AV\_Count\_STATE = FALSE \Rightarrow$
$PM\_Actuator\_V = OFF \land PM\_Sensor\_A = OFF$
$PM\_Sensor\_V = OFF \land PM\_Actuator\_A = OFF$

$inv12 : Pace\_Int\_flag = FALSE \land PM\_Actuator\_V =$
$sp = Pace\_Int \lor (sp < Pace\_Int \land$
$AV\_Count > V\_Blank \land AV\_Count \geq FixedAV)$

$inv13 : Pace\_Int\_flag = FALSE \land PM\_Actuator\_A =$
$(sp \geq Pace\_Int - FixedAV)$
$\ldots$

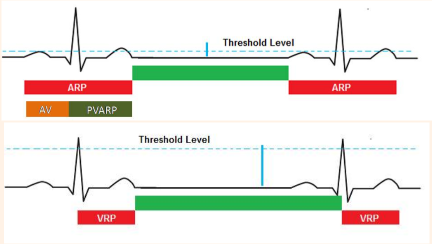
DDD Pacing Modes

EVENT Actuator_ON_V
WHEN
grd1 : $PM\_Actuator\_V = OFF$
grd2 : $(sp = Pace\_Int)$
$\vee$
$(sp < Pace\_Int \wedge AV\_Count > V\_Blank \wedge$
$AV\_Count \geq FixedAV)$
grd3 : $sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM\_Actuator\_V := ON$
act2 : $last\_sp := sp$
END

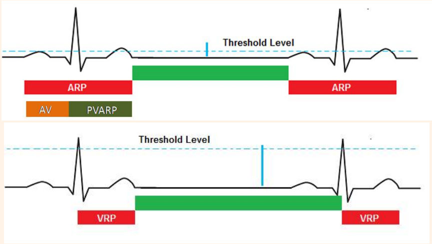
EVENT Actuator_OFF_V
WHEN
grd1 : $PM\_Actuator\_V = ON$
grd2 : $(sp = Pace\_Int)$
$\vee$
$(sp < Pace\_Int \wedge AV\_Count > V\_Blank$
$AV\_Count \geq FixedAV$
grd3 : $AV\_Count\_STATE = TRUE$
grd4 : $PM\_Actuator\_A = OFF$
grd5 : $PM\_Sensor\_A = OFF$
THEN
act1 : $PM\_Actuator\_V := OFF$
act2 : $AV\_Count := 0$
act3 : $AV\_Count\_STATE := FALSE$
act4 : $PM\_Sensor\_V := OFF$
act5 : $sp := 1$
END

EVENT tic
WHEN
grd1 : $(sp < VRP)$
$\vee$
$(sp \geq VRP \wedge sp < Pace\_Int - FixedAV \wedge$
$PM\_Sensor\_A = ON \wedge PM\_Sensor\_V = ON$
THEN
act1 : $sp := sp + 1$
END

First Refinement (Threshold) : Sensor Activity in DDD



First Refinement (Threshold) : Sensor Activity in DDD



$inv1 : Thr\_A \in \mathbb{N}_1 \land Thr\_V \in \mathbb{N}_1$

$inv2 : Pace\_Int\_flag = FALSE \land sp > VRP \land sp < Pace\_Int - FixedAV \Rightarrow PM\_Sensor\_V$

$inv3 : Pace\_Int\_flag = FALSE \land sp > Pace\_Int - FixedAV \land sp < Pace\_Int \land AV\_Count\_$
$PM\_Sensor\_A = OFF \land PM\_Sensor\_V = ON \land PM\_Actuator\_A = OFF$

Second Refinement : Hysteresis

EVENT **Hyt_Pace_Updating Refines Change_Pace_Int**

ANY

Hyt_Pace_Int

WHERE

grd1 : $Pace\_Int\_flag = TRUE$
grd2 : $Hyt\_Pace\_Int\_flag = TRUE$
grd3 : $Hyt\_Pace\_Int \in Pace\_Int \,..\, LRI$

THEN

act1 : $Pace\_Int := Hyt\_Pace\_Int$
act2 : $Hyt\_Pace\_Int\_flag := FALSE$
act3 : $HYT\_State := TRUE$

END

Second and Third Refinements

Second Refinement : Hysteresis

EVENT Hyt_Pace_Updating Refines Change_Pace_Int
ANY
Hyt_Pace_Int
WHERE
grd1 : $Pace\_Int\_flag = TRUE$
grd2 : $Hyt\_Pace\_Int\_flag = TRUE$
grd3 : $Hyt\_Pace\_Int \in Pace\_Int \,..\, LRI$
THEN
act1 : $Pace\_Int := Hyt\_Pace\_Int$
act2 : $Hyt\_Pace\_Int\_flag := FALSE$
act3 : $HYT\_State := TRUE$
END

Third Refinement : Rate Modulation

EVENT Increase_Interval Refines Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
grd1 : $acler\_sensed \geq threshold$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $Pace\_Int := 60000/MSR$
act1 : $acler\_sensed\_flag := TRUE$
END

inv3 : $HYT\_State = FALSE \wedge acler\_sensed < acc\_thr \wedge acler\_sensed\_flag = TR$
inv4 : $HYT\_State = FALSE \wedge acler\_sensed \geq acc\_thr \wedge acler\_sensed\_flag = TR$

Validation & Proof Statistics

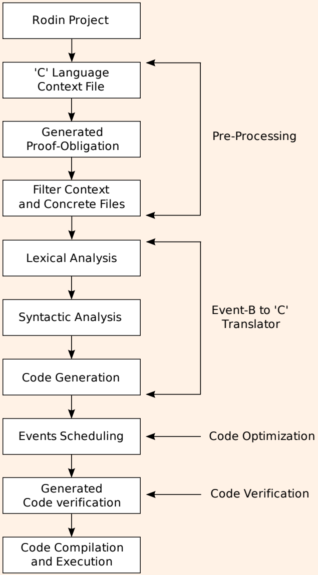
ProB

Model Checker is used to verify the Event-B model and correctness of operating modes.

Proof Statistics

Model	Total number of POs	Automatic Proof	Interactive Proof
One-electrode pacemaker			
Abstract Model	203	199(98%)	4(2%)
First Refinement	48	44(91%)	4(9%)
Second Refinement	12	8(66%)	4(34%)
Third Refinement	105	99(94%)	6(6%)
Two-electrode pacemaker			
Abstract Model	204	195(95%)	9(5%)
First Refinement	234	223(95%)	11(5%)
Second Refinement	3	3(100%)	0(0%)
Third Refinement	83	74(89%)	9(11%)
Total	892	845(94%)	47(6%)

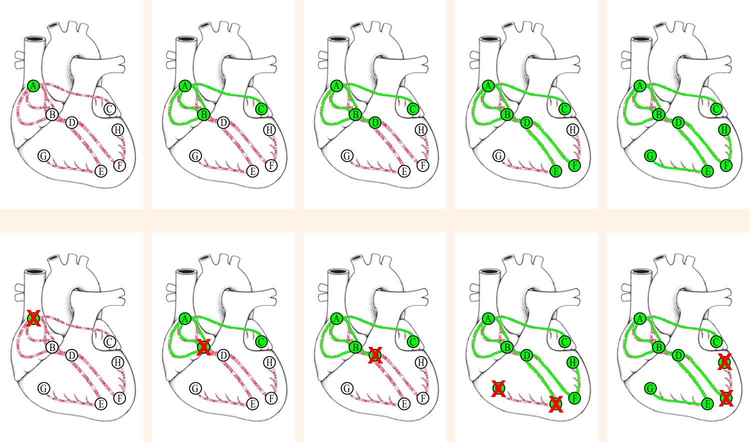
Code Generation



EVENT Actuator_ON_V
WHEN
Actuator_ON_V.Guard1 : $PM\_Actuator$
Actuator_ON_V.Guard2 : $(sp = Pace\_In$
$\vee$
$(sp < Pace\_Int \wedge$
$AV\_Count > V\_Blank \wedge$
$AV\_Count \geq FixedAV)$
Actuator_ON_V.Guard3 : $sp \geq VRP \wedge s$
THEN
Actuator_ON_V.Action1 : $PM\_Actuator$
Actuator_ON_V.Action2 : $last\_sp := sp$
END

```
...
BOOL Actuator\_ON\_V(void)
{
  /* Guards No. 1*/
  if(PM_Actuator_V == OFF){
  /* Guards No. 2*/
  if((sp == Pace_Int) || ((sp < Pace_Int) \&\&
  (AV_Count > V_Blank) && (AV_Count >= FixedAV))){
  /* Guards No. 3*/
  if((sp >= VRP) && (sp >= PVARP) && (sp >= URI)){
  /* Actions */
  PM_Actuator_V = ON;
  last_sp = sp;
  return TRUE;
}}}
  return FALSE;
}
...
```

Electrical Conduction Model



Evaluation of the proposed approach

The development life-cycle is relatively simple and straightforward :

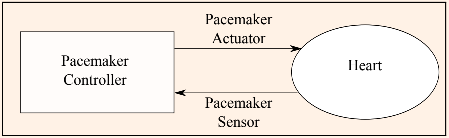
- To formalize the system specification using stepwise development in Event-B.
- Formal verification using the Rodin proof assistant helps for verifying system behavior and safety properties.
- Model checker helps to validate system specification according to the domain experts.
- Real-time animator helps to identify hidden requirements using simulation.
- Automatic code generation generates a reliable code.
- French-Italian Based pacemaker development company is satisfied with this approach.

Conclusion

- Formal methods based development life-cycle methodology to develop the critical system.
- This methodology encourages a view separate from the main 'development' lifecycle for critical systems.
- The Cardiac pacemaker case study indicates successful development from modeling to code generation.
- Help to meet requirements of regulatory agencies like FDA, ISO/IEC and IEEE standards.
- Emphasis on certification from requirements to code implementation within the life-cycle.
- Closed-loop model combining a heart model and the pacemaker model (to appear in 2013 postproceedings FHIES 2012)

Next...

Closed-loop Model



- Applying the complete cycle for a real pacemaker or a new challenge...
- System engineering : developing a pump, managing insulin, ...
- Questions on dependability
- Questions on proving and testing : relationship with physicians.
- Questions on modelling biological environment

Current Summary

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

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

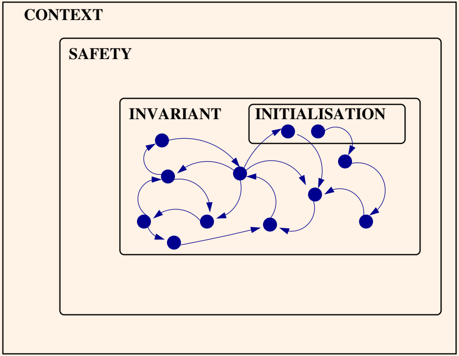
- $\mathcal{R}$ requirements or system properties
- $\mathcal{D}$ domain of the problem
- $\mathcal{S}$ model of the system
- $\longrightarrow$ relation of satisfaction

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.

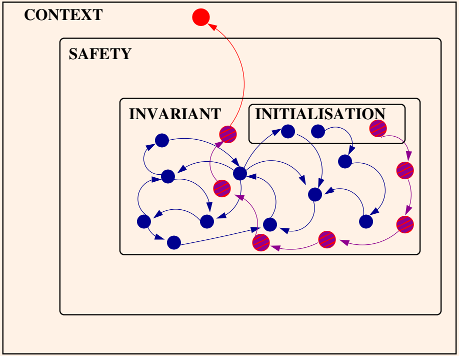
Current Summary

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

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.**

Expression of the invariant

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

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

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

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
    $act1 : islecturing := islecturing \setminus \{p\}$

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

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

Modelling systems

- A **system** is **observed**
- Observation of things which are changing over the **time**
- 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 :
$x_0 \stackrel{\tau}{\rightarrow} x_1 \stackrel{\tau}{\rightarrow} x_2 \stackrel{\tau}{\rightarrow} x_3 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} x_i \stackrel{\tau}{\rightarrow} x_{i+1} \stackrel{\tau}{\rightarrow} \ldots$
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** :

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

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

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

Examples of state properties

- 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, ...
- Partial correctness : a component is correct wit respect to a precondition and a postcondition.
- Safety properties : nothing bad can happen

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** : $x \longrightarrow x' \triangleq (x\ r_0\ x') \vee \ldots \vee (x\ r_n\ x')$
- $\mathrm{I}(\mathrm{x})$ 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
- $\mathrm{Q}(\mathrm{x})$ 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
$\{r_0, \ldots, r_n\}$
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

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

Defining an induction principle for an operational model

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

if, and only if,

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

$$\forall x, x' \in \textbf{Values} : \begin{cases} (1) & \textbf{Init}(x) \Rightarrow \textbf{I}(x) \\ (2) & \textbf{I}(x) \Rightarrow \textbf{Q}(x) \\ (3) & \textbf{I}(x) \wedge x \longrightarrow x' \Rightarrow \textbf{I}(x') \end{cases}$$

if, and only if,

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

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

Modelling systems : second attempt

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

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

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

Modelling systems : Hello world !

stop

MODEL
FACTORIAL_EVENTS
CONSTANTS $factorial, m$
AXIOMS

$$m \in \mathbb{N} \ \wedge \ factorial \in \mathbb{N} \ \leftrightarrow \ \mathbb{N} \ \wedge \ 0 \mapsto 1 \in factorial \ \wedge$$

$$\forall (n, fn).(n \mapsto fn \in factorial \Rightarrow n+1 \mapsto (n+1) * fn \in factorial) \ \wedge$$

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

VARIABLES
$result$
INVARIANT
$result \in \mathbb{N}$
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
$result :\in \mathbb{N}$
EVENTS
$computation =$ **BEGIN** $result := factorial(m)$ **END**
END

Modelling systems : relations to events

MODEL
  m
SETS
  s
CONSTANTS
  c
AXIOMS
  $P(s,c)$
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 from s, c and $P(s,c)$.
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$
- $\forall i \in \{0, \ldots, n\}$:
  $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \mathrm{I}(x) \wedge x \; r_i \; x' \Rightarrow \mathrm{I}(x')$
- $\Gamma(m) \vdash \forall x, x' \in \textsc{Values} : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

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

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

$\boxed{\begin{array}{l} \text{MACHINE} \\ \quad m \\ \text{REFINES} \\ \quad am \\ \text{SEES} \\ \quad c \\ \text{VARIABLES} \\ \quad u \\ \text{INVARIANTS} \\ \quad I(u) \\ \text{THEOREMS} \\ \quad Q(u) \\ \text{VARIANT} \\ \quad < variant > \\ \text{EVENTS} \\ \quad < event > \\ \text{END} \end{array}}$

- $\Gamma(m)$: 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 : $\Gamma(m) \vdash \forall u, u' \in \textsc{Values} : \mathrm{I}(x) \wedge BA(e)(u, u') \Rightarrow \mathrm{I}(u')$
- $\Gamma(m) \vdash \forall u \in \textsc{Values} : \mathrm{I}(u) \Rightarrow \mathrm{Q}(u)$

Contexts in Event B

CONTEXTS
  c
EXTENDS
  ac
SETS
  s
CONSTANTS
  k
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
- k : 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	$G(x) \ \wedge \ P(x,x')$
ANY t WHERE $G(t,x)$ THEN $x :\| P(x,x',t)$ END	$\exists t \cdot (\, G(t,x) \ \wedge \ P(x,x',t)\,)$

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)	$\Gamma(s,c) \;\vdash\; I(x) \wedge \mathsf{grd}\,(E) \;\Rightarrow\; \exists x' \cdot P(x,x')$

The factorial model

CONTEXT
$fonctions$
CONSTANTS
$factorial, n$
AXIOMS

$ax1 : n \in \mathbb{N}$

$ax2 : factorial \in \mathbb{N} \leftrightarrow \mathbb{N}$

$ax3 : 0 \mapsto 1 \in factorial$

$ax4 : \forall (i, fn).(i \mapsto fn \in factorial \Rightarrow i+1 \mapsto (i+1) * fi \in factorial) \land$

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

END

Current Summary

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

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