# On B and Event-B: Principles, Success and Chalenges

Jean-Raymond Abrial

jrabrial@neuf.fr

ABZ 2018 (Southampton)

# To Begin with: a Simple Preamble

- Let us have a little example.

- I used this example in an undergraduate course recently.

- I know you are not undergraduate students any more!

- But it‘s good sometimes to have some refreshments!

- It’s not an example in: ASM, ALLOY, B, Event-B, TLA, VDM, Z, ...

- It’s an example in: C. I suppose everyone knows C

# A small C Program

```
#define black 0
#define white 1
#define m 5
int color[m] = {black, black, white, black, white}
int print_black_color_index() {
    int k;
    printf("black = ");
    for(k=0; k<=m; k++)
        if (color[k]==white) printf("%d ",k);
    return(0);
}
```

Is this C program correct? Let us execute it

# Executing this program

```
#define black 0
#define white 1
#define m 5
int color[m] = {black, black, white, black, white}
int print_black_color_index() {
    int k;
    printf("black = ");
    for(k=0; k<=m; k++)
        if (color[k]==white) printf("%d ",k);
    return(0);
}
```

- We obtain the wrong result: black = 2 4 (although correct for white)

## Some Comments



- The name of the program is: "print_black_color_index".

- However, the program prints white color index instead of black.

- Does it means that the program is not correct?

- Does the name of the program say what the program should do?

## Some Comments (cont'd)



- How to ensure that the program is correct?

- We need a statement outside the program saying what it should do.

- Now, let us modify this program and execute it again

# Modifying the program: printing black now

```
#define black 0
#define white 1
#define m 5
int color[m] = {black, black, white, black, white}
int print_white_color_index() {
    int k;
    printf("black = ");
    for(k=0; k<=m; k++)
        if (color[k]==black) printf("%d ",k);
    return(0);
}
```

- We obtain the strange result: black = 0 1 3 5

## What has happened?



- We just changed white to black

- How come that we get 5?

- We did not get any issue with white

- But now the program acts in a strange manner with black

- Let us look again at the program

- Can you guess what has happened?

# Modifying the program: printing black now

```
#define black 0
#define white 1
#define m 5
int color[m] = {black, black, white, black, white}
int print_black_color_index() {
    int k;
    printf("black = ");
    for(k=0; k<=m; k++)
        if (color[k]==black) printf("%d ",k);
    return(0);
}
```

- We obtain the strange result: black = 0 1 3 5

# Explanation

-The test in the loop is "k<=m" although it should be "k<m"

- This had the effect to have the loop going off the array

- In fact, the program continues to read the memory after the array

- Notice that C does not say anything in this case

## Explanation (cont'd)

- It happens that black is 0

- It happens that the memory after the end of the array contains a 0

- As white is 1, this error was not discovered with white

- If the memory would have been 1, white would have been wrong

- If the memory would have not been 0 or 1, the program would have been "correct" in both cases (although wrong in fact)

- So, this error discovery is very arbitrary

# Some Concluding Remarks



- The C language is misleading

- Array indices in C start at 0

- So, when we write "int color[m]", the last index value is m-1

- This is the reason why having "k<=m" instead of "k<m" led to an error

# Some Concluding Remarks (cont'd)

- But, again, this error was not discovered with white, only with black

- The same program was "correct" with white and wrong with black

- So, testing is not a reliable approach to programming

- Does this happen with other programming languages (i.e. Java)?

- Yes and no: Java raises an exception in this case (and many more)

- Such an exception can be treated by an exception handler

## Exception Handlers

- Exception handlers are terrible features

- What to do if an exception occurs while the aircraft is about to land

- But people like them because they are useful in the testing phase

- However, still dangerous (e.g. Ariane 5 crash)

- Still a treatment performed on the final program (like testing)

- How about abstract interpretation?

# "Abstract" Interpretation



- The exception can be raised (systematically?) in the laboratory

- But still a treatment performed on the final program (like testing)

- It does not say that the program is correct

- It just detects that the program does not contain bad things:
    - array bound overflow
    - null pointers
    - division by zero
    - ...

- Even no loop termination checking (I think)

## What we Need

- What we need is to prove that our program does not contain bad things like this

- And many more: the program meets its specification

- And many more: the program is correct within an entire system

- The answer is in B and Event-B

- And, clearly, in other formalisms presented here at ABZ

- End of preamble

# Outline of this Presentation



- Basic principles of B and Event-B

- Differences and similarities between B and Event-B

- B and Event-B spreading

- Issues and challenges

# Basic Principles: Being Correct by Construction (CxC)



- "CxC" is the main purpose of B and Event-B

- So, B and Event-B are not programming languages

- They are rather intellectual modelling tool

- They correspond to a practice used in other engineering disciplines

## Basic Principles: Refinement



- Refinement is fundamental in engineering practice

- Because a model cannot be defined in a single step

- Models are thus constructed gradually

- From an initial very abstract view to a final concrete one

- Abstraction is usually very difficult to master by informaticians

- So, practitioners have to be seriously educated on this matter

# Basic Principles: Mathematical Notation

- Refinement is not sufficient

- At each step of the development some statements have to be proved

- Such statements intend to ensure that each step is valid

- For such statements, no "new language" is developed

- The most classical mathematical notation is used

- Such a notation is that of predicate logic and typed set theory

## Basic Principles: Tools for Developing Models

- Model cannot be developed with pen and papers

- Because they are too big and too complex

- Tools are thus absolutely necessary to help users writing models

- The tool for B is Atelier B

- The tool for Event-B is Rodin

- Both are free

- Among the tools, one is very important

- It analyses models provided by users

- And generate "proof obligations" necessary to validate models

- This cannot be done manually by users as far too much error prone

- Such a "proof obligation generator" was inspired by that for VDM

## Basic Principles: Tools for Proving

- Once proof obligations are generated, they must be proved

- For this, some proving tools have been developed or imported

- Such tools work either automatically or interactively

## Basic Principles: More Tools



- Other tools are developed

- By Universities (Southampton, Düsseldorf, Turku, ...)

- By industries (Siemens Transport, Clearsy, Systerel, ...)

- These tools are the following (among others):
    - Model checking
    - Animation
    - Automatic refinement
    - Model decomposition and structuring
    - Link with UML
    - Data validation
    - ...

## Comparing B and Event-B

- Book on B published in 1996

- Book on Event-B published in 2010

- So, Event-B took advantage of this time difference

- Event-B strongly influenced by Action System
  (R-J. Back and R. Kurki-Suonio)

- Event-B is used for the modelling of entire systems

# Differences (1)



- Main difference: operations (in B) and events (in Event-B)

- Operations are pre-conditioned, whereas events are guarded

- Pre-conditions determine when an operation can be called

- Guards determine when an event can occur

- Both are assumed in proofs

- But pre-conditions are weakened in refinements

- Whereas guards are strengthened in refinements

# Differences (2)

- Differences in refinements allows events to be developed gradually

- Event parameters can be modified, added or instantiated in refinements

- This is not possible for operations (fixed structure from abstraction)

- New events can be added in refinements

- Constants defined in separate components in Event-B

- This allows for more flexibility in Event-B than in B

## Differences (3)

- Event-B has no programming constructs as B does (conditionals, loops, sequencing, ...)

- Proof obligations are thus simpler in Event-B than in B

- This simplification is important because of the absence of sequencing in Event-B

- However, code generation is simpler for B than for Event-B

## Differences (4)

- In Event-B, we worked a lot on welldefinedness proof obligations

- Examples: f(x), card(S), min, max, a/b

## Similarities

- Usage of the math notation: predicate logic and typed set theory

- So, proof obligations can be handled by similar provers in both

- Consequence: some provers of Atelier B are used in Rodin.

- Also both use similar external provers (SMT provers)

- Event-B can be simulated in B (adding specific proof obligations)

## Spreading in Industries



- B is extensively used in Industries (by Clearsy and others)

- Clearsy claims to make 30% of its business with B

- The main industrial activity is with train systems

- Alstom and Siemens Transport actively participate in these activities

- Train systems with B in Europe, North and South America, Asia

## Spreading in Academia

- Event-B widely spread in universities around the world

- In France, United Kingdom, Germany, Spain, Finland

- Also in North and South America (Canada, Brazil, Columbia) and Asia (China, Japan)

# Challenges



- Poor spreading in industries except in train industries

- Other industries could use B (or other formal methods) but do not

- Examples: energy, automotive, aeronautics, space, ...

- People there claim it is too expensive

- They also claim that introducing this technology is difficult

- Event-B and B not used yet in the same project

  (system with Event-B and then software with B)

## Some References

- For B: www. clearsy.com/en

- For Event-B: www.Event-B.org

- R.-J. Back and R. Kurki-Suonio *Decentralisation of Process Net with Centralized Control*. Distributed Computing (1989)

- T. Lecomte and al. *Applying a Formal Method in Industry: a 25 years Trajectory*. Formal Methods: Foundations and Applications. Springer (2017)