

**CSC 4504 : *Langages formels et applications***

## **(Event-B)**

**J Paul Gibson, A207**

`paul.gibson@it-sudparis.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC4504/>

## **A Simple Machine - A Pair of Data Values**

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC4504/PairMachine.pdf>

## Even the simplest behaviour can prove problematic to implement correctly.

Consider a University that has a database of student records.

Each student has a subject data field which corresponds to their field of study, eg Computing, Maths, French, ...

At registration, the student is allocated a subject and this is fixed for the duration of their study at the University.

Two « extensions » are proposed to improve the system:

1. Allow students to study 2 subjects in order to get a double degree
2. Allow students to change their chosen subjects after registration

The following scenario could be problematic:

1. A student registers for Maths and Computing
2. They then change Computing for Maths (after registration)

## Even the simplest behaviour can prove problematic to implement correctly.

This could pose a problem because the student is now studying a double degree in Maths and Maths

Imagine what would happen if the rules for a double degree stated that the student must achieve  $\frac{2}{3}$  of the honours programme in each subject to get a double honours degree.

Now, in order to get an honours degree, does the student need:

$\frac{2}{3}$  or  $\frac{4}{3}$  or ??? of the maths honours programme

Could this sort of thing really happen ?

Unfortunately, the answer is yes.

Good programmers would/could not make the mistake?

Using formal methods we reduce the risk of poor programmers from making the mistake

# The Pair of Values Problem in Java

```
/**
 *
 * This class provides the behaviour of a Pair (2-tuple) of data values<br>
 * The 2 data are chosen from a simple enumeration of 3 values - EL1, EL2, EL3 <br>
 * These values <u>must never be the same</u>
 * @author J Paul Gibson
 * @version 1
 */
public class PairOfData {

    /**
     * The Data values for the Pair elements
     * @author gibson
     */
    enum Data {EL1, EL2, EL3};

    /**
     * The first element of the Pair of Data
     */
    private Data first;

    /**
     * The second element of the Pair of Data
     */
    private Data second;
```

# The Pair of Values Problem in Java

```
/**
 *
 * @param d1 First data element
 * @param d2 Second data element
 */
PairOfData (Data d1, Data d2){
  if (d1!= d2) {first = d1; second = d2;}
}

/**
 *
 * @return First data element
 */
public Data get_first(){return first;}

/**
 *
 * @param new_first New value for First data element
 */
public void set_first (Data new_first){first = new_first;}
```

# The Pair of Values Problem in Java

```
/**
 *
 * @return Second data element
 */
public Data get_second(){return second;}

/**
 *
 * @param new_first New value for First data element
 */
public void set_second (Data new_first){second = new_first;}

/**
 * Generate string representation as (<em>first</em>, <em>second</em>)
 */
public String toString(){ return "("+ first+", "+second+"");}
```

# The Pair of Values Problem in Java

```
/**
 * Simple self-test - for students - with results output to screen
 * @param args expected to be empty
 */
public static void main (String args []){
  PairOfData p = new PairOfData(Data.EL1, Data.EL1);
  System.out.println("Created the pair "+ p);
}

} // ENDCLASS PairOfData
```

## QUESTIONS:

- What is output by the simple test in the main methods?
- What problem(s) do you see with this code?
- Can you fix them? ... (open up Eclipse) and have a go – download [code](#) and [documentation](#) from module web site

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC4504/Downloads/PairOfData.java>

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC4504/Downloads/PairOfData-JavaDoc/>

## The Pair of Values More Formally

We need to be more precise about how the code is supposed to manage/enforce the error situation where the 2 data values are the same.

We would like to be able to verify that our design guarantees that this situation does not arrive when our system is executing

We would like this verification to be static (at compile time) – in the form of a mathematical proof that can be checked (by machine)

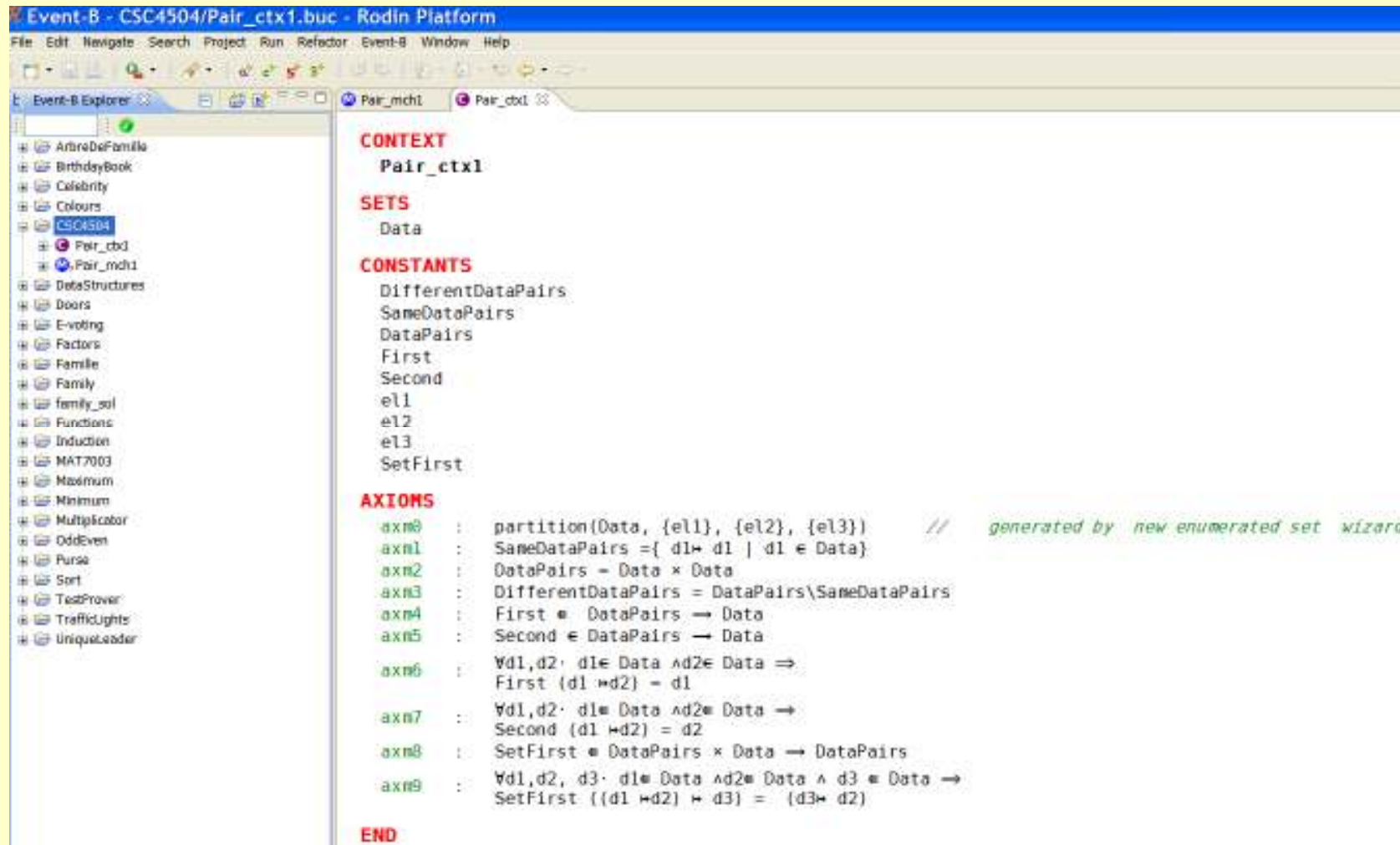
Testing cannot, in general, provide an exhaustive/complete verification

If this was critical software one should probably choose to develop the code using formal methods (based on mathematical models)

We will take a look at this example in Event-B

Precise details of how this works will follow later in the module. We will build experience/knowledge incrementally.

# The Pair of Values in Event-B



The screenshot shows the Rodin Platform Event-B editor. The left pane displays a project tree with 'CSC4504' selected, containing sub-projects like 'Pair\_ctx1' and 'Pair\_mch1'. The main editor area shows the following Event-B code:

```
CONTEXT
  Pair_ctx1

SETS
  Data

CONSTANTS
  DifferentDataPairs
  SameDataPairs
  DataPairs
  First
  Second
  e1
  e2
  e3
  SetFirst

AXIOMS
  axn0 : partition(Data, {e1}, {e2}, {e3}) // generated by new enumerated set wizard
  axn1 : SameDataPairs = { d1 ↦ d1 | d1 ∈ Data }
  axn2 : DataPairs = Data × Data
  axn3 : DifferentDataPairs = DataPairs \ SameDataPairs
  axn4 : First ∈ DataPairs → Data
  axn5 : Second ∈ DataPairs → Data
  axn6 : ∀ d1, d2. d1 ∈ Data ∧ d2 ∈ Data ⇒
    First (d1 ↦ d2) = d1
  axn7 : ∀ d1, d2. d1 ∈ Data ∧ d2 ∈ Data ⇒
    Second (d1 ↦ d2) = d2
  axn8 : SetFirst ∈ DataPairs × Data → DataPairs
  axn9 : ∀ d1, d2, d3. d1 ∈ Data ∧ d2 ∈ Data ∧ d3 ∈ Data ⇒
    SetFirst ((d1 ↦ d2) ↦ d3) = (d3 ↦ d2)

END
```

We, typically, start by specifying the *context* of the problem

# The Pair of Values in Event-B

The Date enumeration -

« *Data is a SET of elements  $\{el1, el2, el3\}$  which are distinct, i.e.:  
 $el1 \neq el2 \ \& \ el2 \neq el3 \ \& \ el1 \neq el3$  »*

## SETS

Data

## CONSTANTS

el1 el2 el3

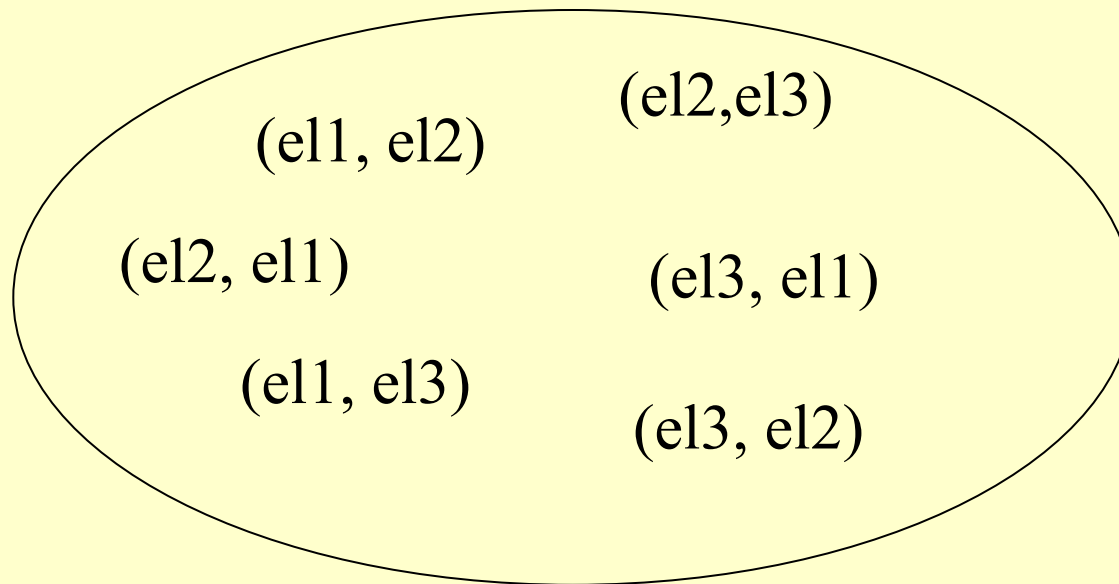
## AXIOMS

axm0 : partition(Data, {el1}, {el2}, {el3})  
// *generated by new enumerated set wizard*

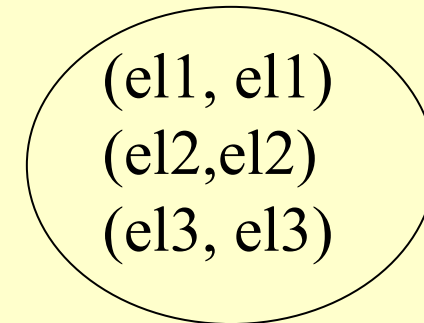
We will see the formal definition of partition later.

# The Pair of Values in Event-B

Allowable Pairs



Forbidden Pairs



## CONSTANTS

DifferentDataPairs  
SameDataPairs  
DataPairs

## AXIOMS

axm1 : SameDataPairs = { d1 ↦ d1 | d1 ∈ Data }

axm2 : DataPairs = Data × Data

axm3 : DifferentDataPairs = DataPairs \ SameDataPairs

## The Pair of Values in Event-B

Now, the accessor functionality/behaviour:

### CONSTANTS

First

Second

`axm4` : `First ∈ DataPairs → Data`

`axm5` : `Second ∈ DataPairs → Data`

## The Pair of Values in Event-B

Now, the setter functionality/behaviour:

### CONSTANTS

SetFirst

### AXIOMS

axm8 : SetFirst  $\in$  DataPairs  $\times$  Data  $\rightarrow$  DataPairs

axm9 :  $\forall d1, d2, d3. d1 \in \text{Data} \wedge d2 \in \text{Data} \wedge d3 \in \text{Data} \Rightarrow$   
SetFirst  $((d1 \mapsto d2) \mapsto d3) = (d3 \mapsto d2)$

QUESTION: Can you specify the SetSecond AXIOMS?

# The Pair of Values in Event-B

Now, lets build our machine in the context

## MACHINE

Pair\_mch1

## SEES

Pair\_ctx1

## VARIABLES

PairOfData

## INVARIANTS

inv1 : PairOfData  $\in$  DifferentDataPairs

## EVENTS

... // INITIALISATION getFirst setFirst getSecond setSecond

## END

## The Pair of Values in Event-B

Now, the event specifications:

**INITIALISATION**  $\triangleq$

**STATUS**

Ordinary

**BEGIN**

**act1** : PairOfData  $\in$  DifferentDataPairs

**END**

## The Pair of Values in Event-B

Now, the event specifications:

```
getFirst    ≐  
STATUS  
Ordinary  
ANY  
First  
WHERE  
grd1      :   first = First(PairOfData)  
THEN  
Skip  
END
```

## The Pair of Values in Event-B

Now, the event specifications:

```
setFirst    ≐  
STATUS  
Ordinary  
ANY  
newFirst  
WHERE  
grd1      :   newFirst ∈ Data ∧ newFirst ≠ Second(PairOfData)  
THEN  
act1      :   PairOfData := SetFirst(PairOfData ↦ newFirst)  
END
```

## The Pair of Values in Event-B

Now, the event specifications:

```
getSecond  ≐  
  STATUS  
  Ordinary  
  ANY  
  Second  
  WHERE  
  grd1    :    second = Second(PairOfData)  
  THEN  
  Skip  
  END
```

## The Pair of Values in Event-B

Now, the event specifications:

setSecond  $\triangleq$

STATUS

Ordinary

ANY

newSecond

WHERE

grd1 : newSecond  $\in$  Data  $\wedge$  newSecond  $\neq$  First(PairOfData)

THEN

act1 : PairOfData  $\equiv$  First(PairOfData)  $\mapsto$  newSecond

END

# The Pair of Values in Event-B

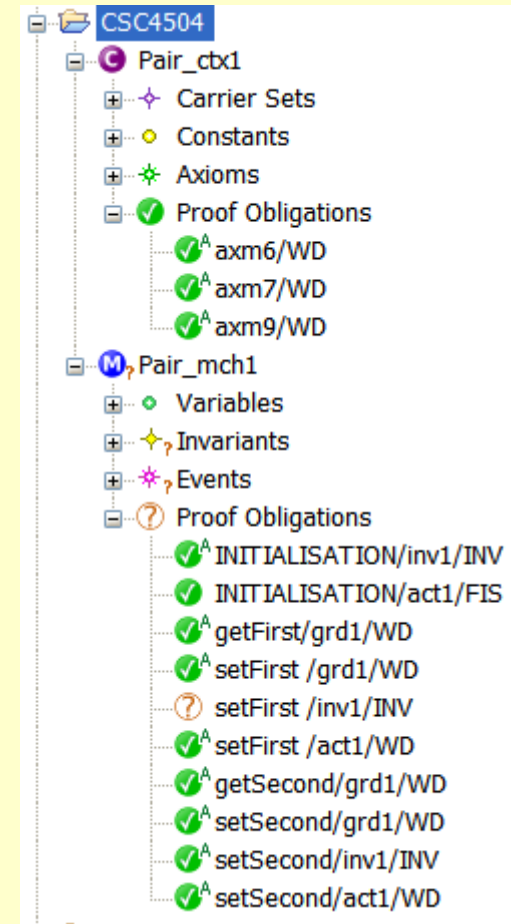
Some things to note:

The 2 setter functions are specified differently

The nondeterministic initialisation

The proof of the invariant (safety property):

All Proof Obligations are discharged automatically except the proof of the invariant for setFirst



So, to discharge the obligation we have to use the prover interactively:

The screenshot displays the Rodin Platform interface for proving a property. The main window is titled "Proving - CSC4504/Pair\_mch1.bps - Rodin Platform". It features several panes:

- Proof Tree:** Shows a hierarchical structure of proof steps, including "simplification rewrites", "type rewrites", "simplification rewrites", "sl/ds", "eh (DifferentDataPairs=)", and "SetFirst(PairOfData)".
- setFirst /inv1/INV:** A list of hypotheses and a goal. The hypotheses are:
  - $\rightarrow \text{newFirst} = \text{Second}(\text{PairOfData})$
  - $\text{DifferentDataPairs} = \text{DataPairs} \setminus \text{SameDataPairs}$
  - $\text{Second} = \text{DataPairs} \rightarrow \text{Data}$
  - $\text{SetFirst} = \text{DataPairs} \times \text{Data} \rightarrow \text{DataPairs}$
  - $\text{PairOfData} \mapsto \text{newFirst} = \text{dom}(\text{SetFirst})$
  - $\text{SetFirst} = \text{Data} \times \text{Data} \times \text{Data} \leftrightarrow \text{Data} \times \text{Data}$
  - $\forall d1 \text{ [redacted]}, d2 \text{ [redacted]}. \text{Second}(d1 \mapsto d2) = d2$The goal is:  $\text{SetFirst}(\text{PairOfData} \mapsto \text{newFirst}) = \text{DataPairs} \setminus \text{SameDataPairs}$
- Event-B Explorer:** A tree view showing the project structure, including "ArbreDeFamille", "BirthdayBook", "Celebrity", "Colours", "CSC4504", "Pair\_cb1", "Carrier Sets", "Constants", "Axioms", "Proof Obligations" (axm6/WD, axm7/WD, axm9/WD), "Pair\_mch1", "Variables", "Invariants", "Events", and "Proof Obligations" (INITIALISATION/inv1/INV, INITIALISATION/act1/FIS, netFirst/ord1/WD).
- Proof Control:** A toolbar with various proof tactics and a "Rodin Problems" panel.
- Symbols:** A keyboard layout for the Rodin prover, including symbols for variables, constants, and logical operators.

We will learn about this in following lecture(s)