

**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/>

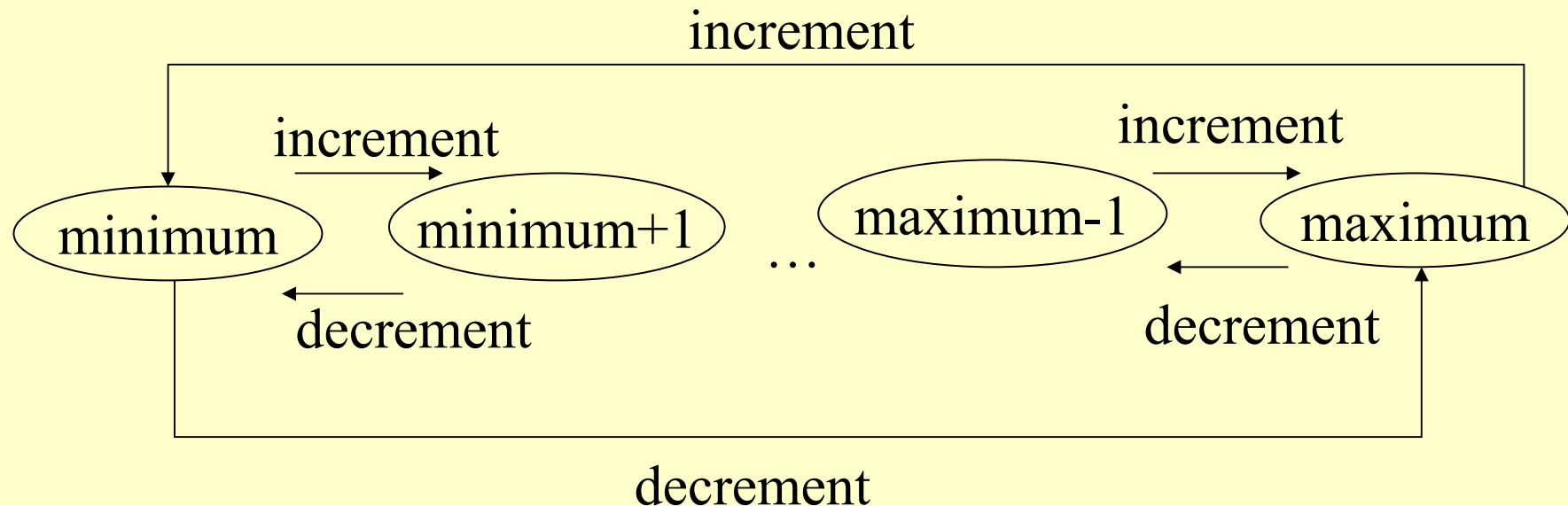
## **A Round Counter – Alternative Specifications**

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

# The RoundCounter Problem

In a finite state system where one needs a timer/counter then one can choose to implement a counter with a finite range of values:

One possible design is a form of wraparound:

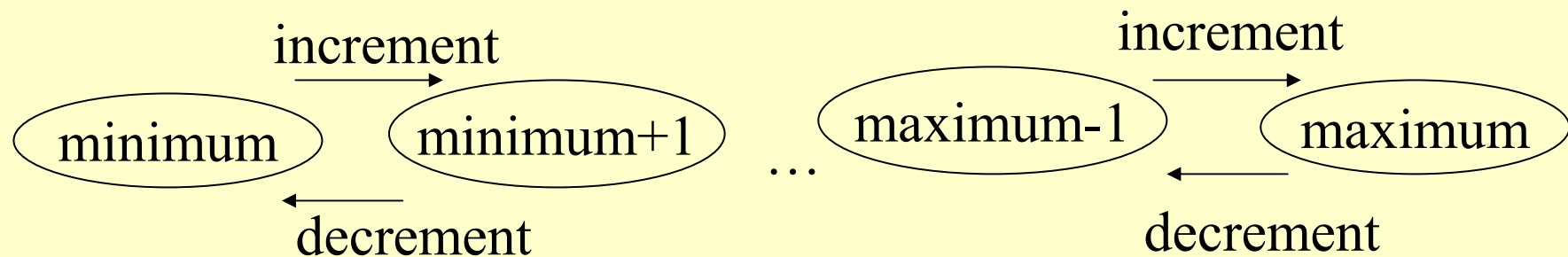


This behaviour can be modelled as a **RoundCounter**

# The RoundCounter Problem

An alternative design does not allow:

- Incrementing the maximum
- Decrementing the minimum



This behaviour can be modelled as a **LimitedCounter**

## The RoundCounter Problem

These counters must guarantee to never be in a state where the count value is less than the minimum, or the count value is more than the maximum; otherwise it becomes unsafe!

Specify the 2 designs in Event-B:

- RoundCounter (context and machine)
- LimitedCounter (context and machine)

Verify (using the prover) that the system is always in a safe state

# The RoundCounter Context

## CONTEXT

RoundCounter\_ctx

## CONSTANTS

minimum  
maximum  
range  
inc  
dec

## AXIOMS

axm1 : minimum  $\in$  N  
axm2 : maximum  $\in$  N  
axm3 : minimum  $\leq$  maximum  
axm4 : range = minimum .. maximum  
axm5 : inc  $\in$  range  $\rightarrow$  range  
axm9 : dec  $\in$  range  $\rightarrow$  range  
axm6 :  $\forall x. x \in \text{minimum} \dots \text{maximum} - 1 \Rightarrow$   
inc(x) = x+1  
axm7 : inc (maximum) = minimum  
axm8 :  $\forall x. x \in \text{minimum}+1 \dots \text{maximum} \Rightarrow$   
dec(x) = x-1  
axm10 : dec(minimum) = maximum

END

# The RoundCounter Machine

## MACHINE

RoundCounter\_mch

## SEES

RoundCounter\_ctx

## VARIABLES

count

## INVARIANTS

inv1 : count  $\in$  N  
inv2 : count  $\geq$  minimum  
inv3 : count  $\leq$  maximum

## EVENTS

**INITIALISATION**  $\triangleq$

**STATUS**

ordinary

**BEGIN**

act1 : count  $\in$  minimum .. maximum

**END**

**increment**  $\triangleq$

**STATUS**

ordinary

**BEGIN**

act1 : count := inc (count)

**END**

**decrement**  $\triangleq$

**STATUS**

ordinary

**BEGIN**

act1 : count := dec(count)

**END**

**END**

# The LimitedCounter Machine (re-using the RoundCounter context)

```
MACHINE
  LimitedCounter

SEES
  RoundCounter_ctx

VARIABLES
  count

INVARIANTS
  inv1 : count ∈ range

EVENTS

  INITIALISATION ≐
  STATUS
    ordinary
  BEGIN
    act1 : count :∈ range
  END
```

```
  increment ≐
  STATUS
    ordinary
  WHEN
    grd1 : count < maximum
  THEN
    act1 : count = count+1
  END

  decrement ≐
  STATUS
    ordinary
  WHEN
    grd1 : count > minimum
  THEN
    act1 : count = count-1
  END

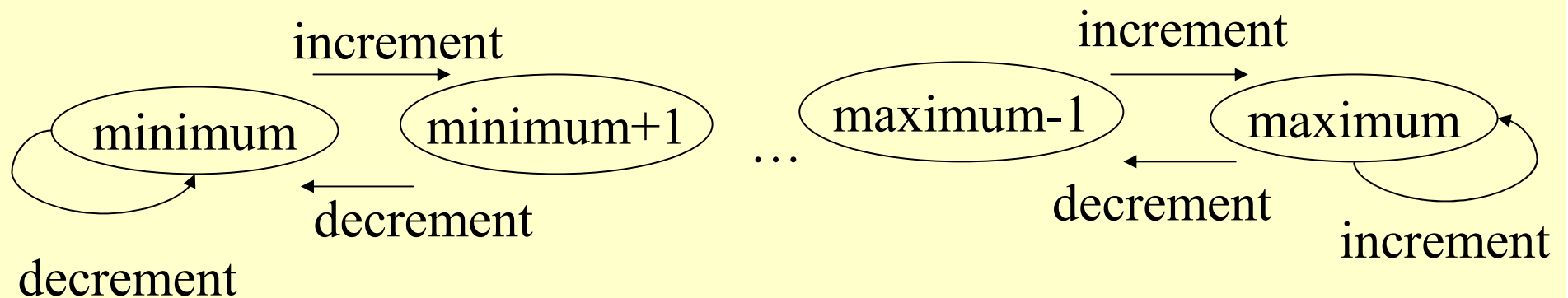
FND
```

NOTE: the same invariant is specified two different (but equivalent) ways in each machine

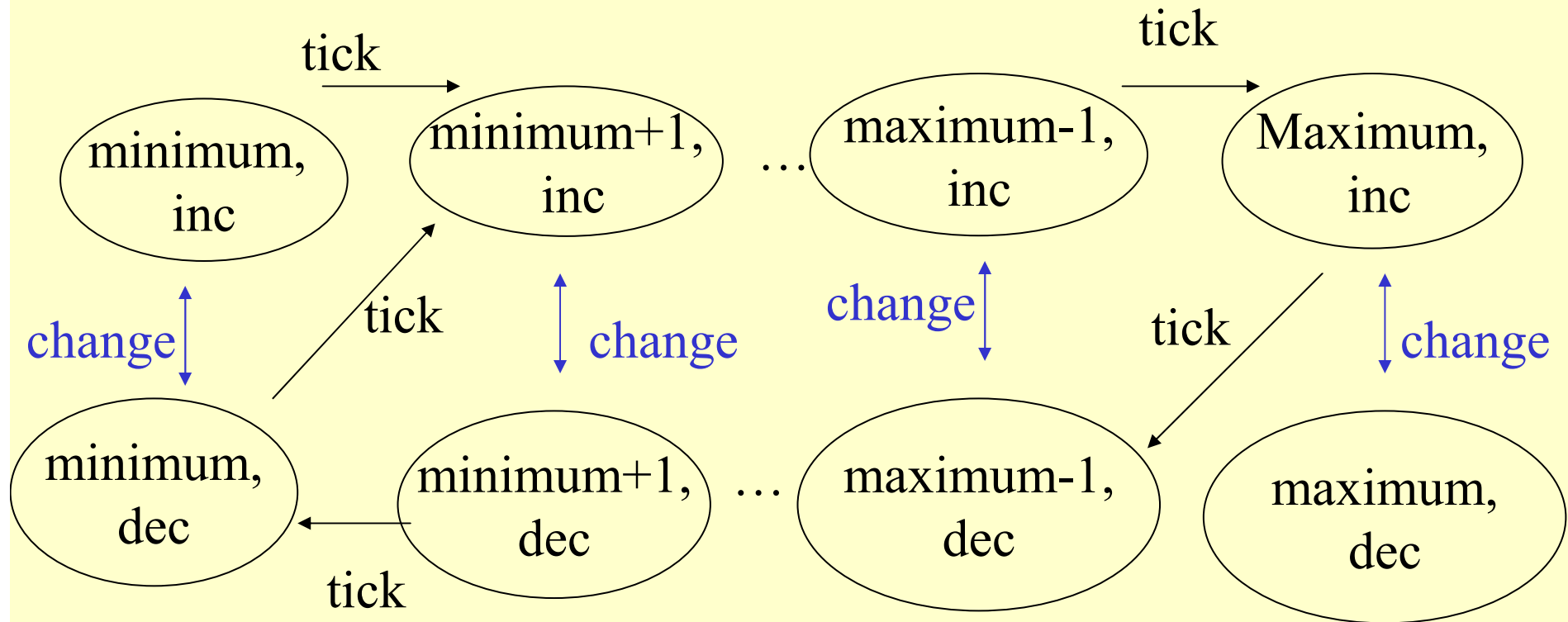
# Hybrid Counter

Model the following *hybrid counter* in Event-B

You should use a *mix* of the RoundCounter and LimitedCounter



## A New Type Of Counter – that ticks (up and down)



The machine ticks up and down the range of allowed values.

It automatically changes direction when it reaches the minimum or maximum

There is also a change event which changes the direction of the next tick

# A New Type Of RoundCounter

## TO DO:

Specify the new behavioural requirements in Event-B

Try to structure the specification between the context and the machine

Verify that the machine stays in a safe state (using an invariant)

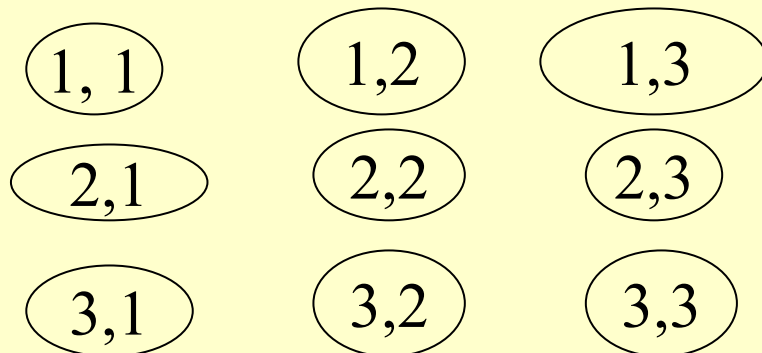
Once you have validated/verified your specification, implement the machine as a Java class

## Composition in Event-B

We wish to compose 2 limited counters in order to provide a bigger (limited) counter.

For example,

We wish to count to 9 by using 2 counters that can each count to 3



TO DO: Try to do specify this composition in an Event-B **context**, then try to build the single counter **machine**