Modelling and verification with B Method

with Event B

J. Christian Attiogbé

Nantes, November 2018

Plan

1. A First Example: GCD
2. A motivating Case Study

The GCD Example

Formal development

mathematical model → programming model

Illustration: From an abstract machine to its refinement into code.

gcd(x,y) is d | x mod d = 0 ∧ y mod d = 0
∧ ∀ other divisors dx d > dx
∧ ∀ other divisors dy d > dy

Refinement = Development method = design

Constructing the GCD: abstract machine

```
MACHINE
    pgcd1 /* the GCD of two naturals */
            /* gcd(x,y) is d | x mod d = 0 ∧ y mod d = 0
            ∧ ∀ other divisors dx d > dx
            ∧ ∀ other divisors dy d > dy */
OPERATIONS
        rr <-- pgcd(xx,yy) = /* OUTPUT : rr ; INPUT xx, yy */
            ...
END
```

Constructing the GCD: abstract machine

```
OPERATIONS
rr <-- pgcd(xx,yy) = /* specification of gcd */
PRE
     xx : INT & xx >= 1 & xx < MAXINT
& yy : INT & yy >= 1 & yy < MAXINT
THEN
     ANY dd WHERE
     dd : INT
     & (xx - (xx/dd)*dd) = 0 /* d is a divisor of x */
     & (yy - (yy/dd)*dd) = 0 /* d is a divisor of y */
          /* and the other common divisors are < d */
     & !dx.((dx : INT & dx < MAXINT
          & (xx- (xx/dx)*dx) = 0 & (yy-(yy/dx)*dx)=0) => dx < dd)
     THEN rr := dd
     END
END
```


Constructing the GCD: refinement

```
REFINEMENT /* refinement of ...*/
     pgcd1_R1
REFINES pgcd1 /* the former machine */
OPERATIONS
rr <-- pgcd (xx, yy) = /* the interface is not changed */
     BEGIN
          ... Body of the refined operation
     END
END
```

Constructing the GCD: refinement

```
rr <-- pgcd (xx, yy) = /* the refined operation */
        VAR cd, rx, ry, cr IN
            cd := 1
            ; WHILE ( cd < xx & cd < yy) DO
                ; rx := xx - (xx/cd)*cd ; ry := yy - (yy/cd)*cd
                IF (rx = 0 & ry = 0)
                THEN /* cd divids x and y; possible GCD */
                    cr := cd /* possible rr */
                END
                ; cd := cd + 1 ; /* searching a greater one */
            INVARIANT
                xx : INT & yy : INT & rx : INT & rx < MAXINT
                & ry : INT & ry < MAXINT & cd < MAXINT
                & xx = cr*(xx/cr) + rx & yy = cr*(y/cr) + ry
            VARIANT
                xx - cd
            END             ; rr := cr
        END
```


Case study: inter-process interactions manag. system

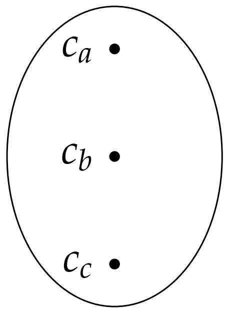
Figure: Interaction between processus

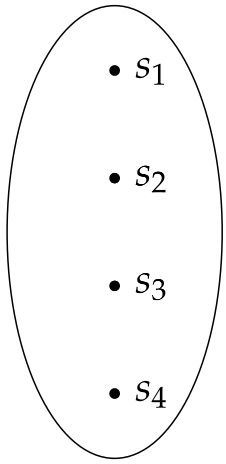
- Read the requirements document
- Analysis of the requirements document
- Modelling of the system
- Development of the system

Modelling the data

Given the sets SUBSCRIBER, CONNECTION

$Connections \subseteq CONNECTION$ $\qquad$ $subscribers \subseteq SUBSCRIBER$

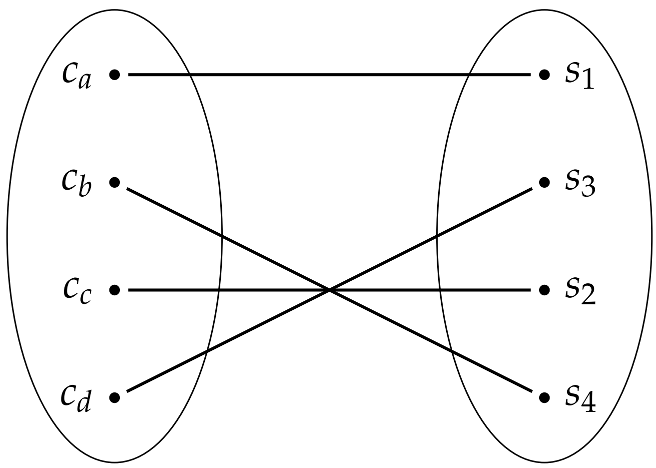




Modelling the state space

Req	Each connection has one caller, which has only one connection

$Conx$ $\qquad$ $Subsc$



$callerConx$

We need a total injective function to specify that.

Modelling

Each element in the domain of *callerConx* has one image:

$$callerConx : Conx \rightarrowtail Subsc$$

$$(c_a, s_1) \in callerConx\ ;\ (c_b, s_4) \in callerConx;\ \cdots$$

$$callerConx(c_b) = s_4 \quad callerConx(c_a) = s_1$$

$$\text{the reverse is defined} \quad callerConx^{-1}(s_4) = c_b$$

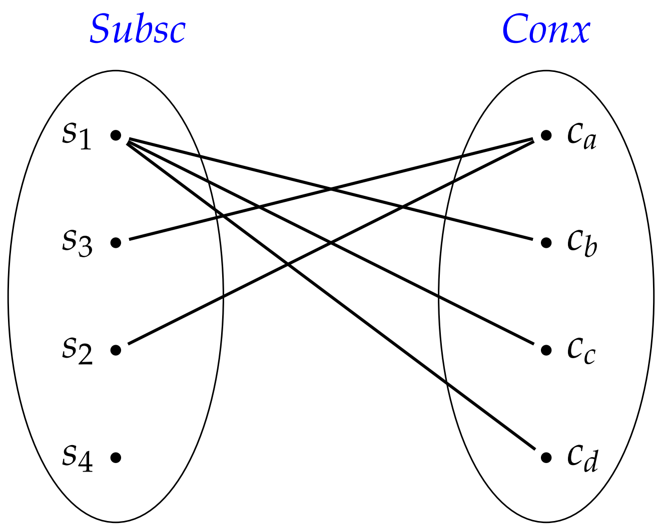
☛ the function is not defined for values not in its domain

$$callerConx(c_8) = ???$$

Before applying a function, check if the arg is in its domains

Modelling the state space

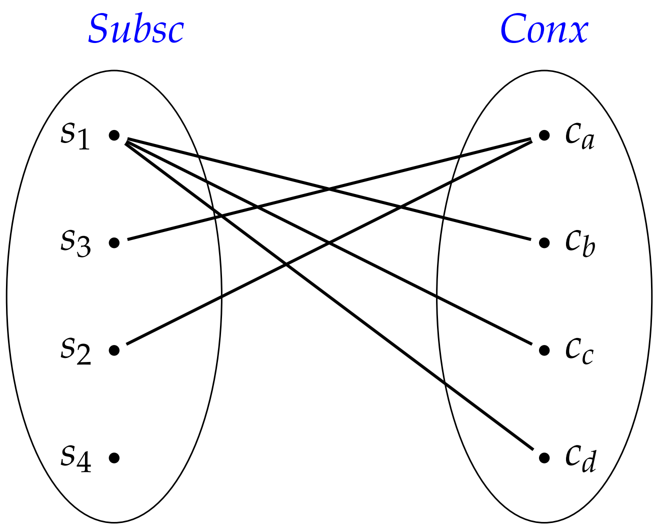
Req	Each connection involves one/several subscribers (called)



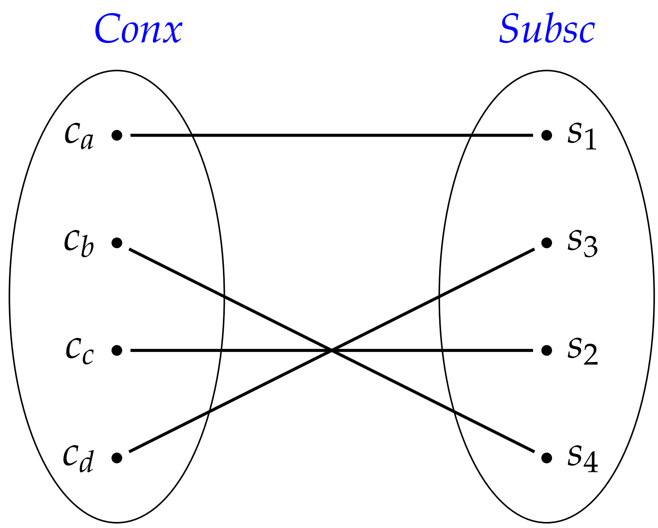
calledConx

We need a partial surjective function to specify

Modelling the state space

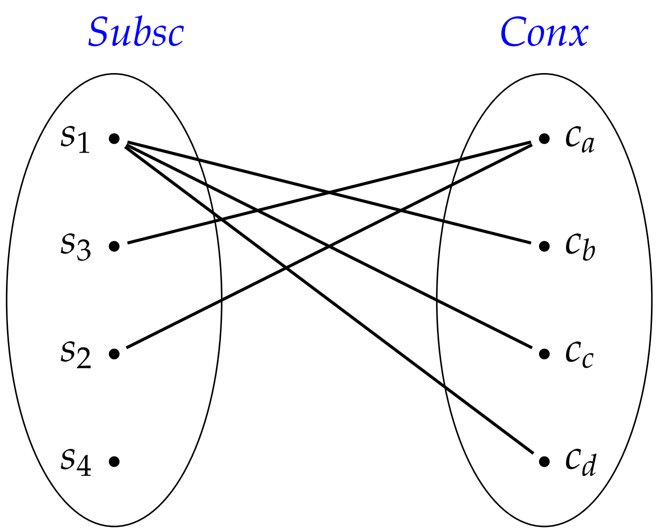


calledConx



callerConx

Modelling the state space



calledConx

The called subscribers in a connection:

$$calledConx \in Subsc \leftrightarrow Conx$$

But, **every connection should have callee**

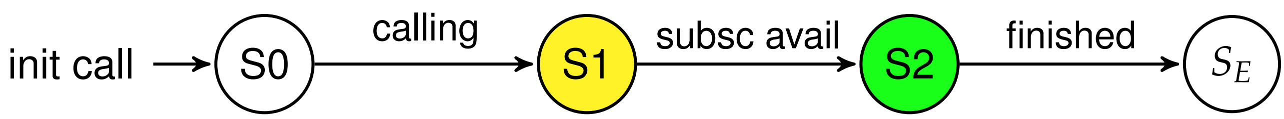
$$\mathrm{ran}(calledConx) = conx$$

How to get the called:

$$calledConx^{-1}[\{c_a\}] = \{s_3, s_2\}$$

if $c_a \in Conx$

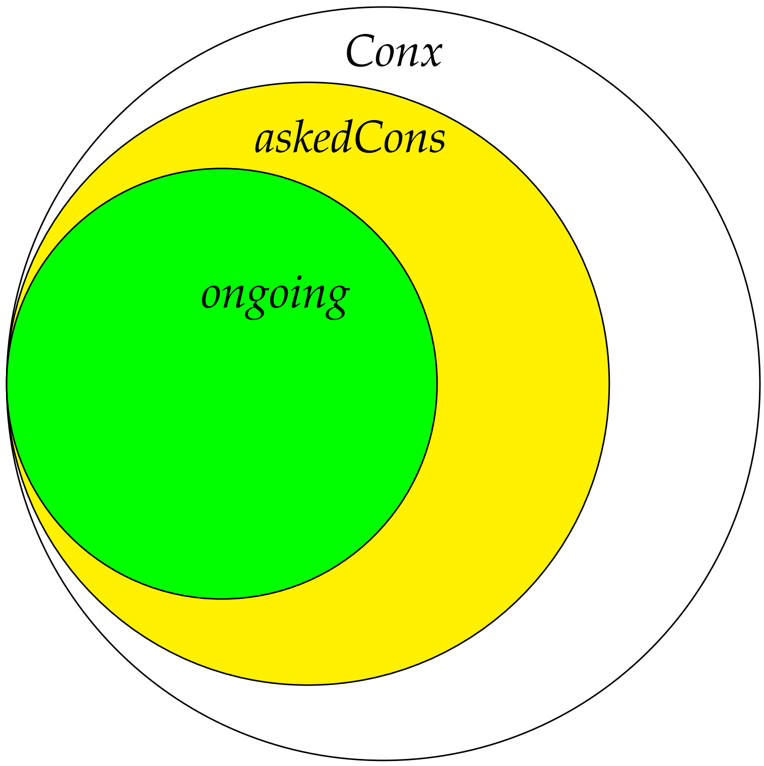
Modelling: analysis of the evolution of the system



- In each state, we have a set of connections/subscribers.
- At the beginning, a connection is created by a subscriber.
- Then the connection becomes asked (ie waiting for resources = involved subscribers)
- Then the connections move from asked to ongoing when the subscribers are availalabe.

From the set modeling point of view, we use subset relations.

Modelling: analysis of the evolution of the system



$Conx$ all the connections

↓

$askedConx$ asked connections

↓

$ongoingConx$ ongogoing.

↓

termination

Modelling the system properties

REQ. A subscriber should not be involved in more than one ongoing connection.

The subscribers called/involved in a connection ce:

$$calledConx^{-1}(ce)$$

The subscribers involved in a set of connection $startedConx$:

$$calledConx^{-1}[startedConx]$$

Hence, if we have some connections in $ongoingConx$ then

$$\bigcap_{ce \in ongoingCnx} calledConx^{-1}(ce) = \{\}$$

A motivating Case Study

Modelling the system properties

Safety	The ongoing connections do not share called subscribers

The subscribers called in a connection is : $calledConx^{-1}(ce)$

Hence, if we have some connections in $ongoingConx$ then

$$(ongoingConx / = \{\}) \Rightarrow (\bigcap_{ce \in ongoingCnx} calledConx^{-1}(ce) = \{\})$$

Modelling the system properties

REQ. A connection cannot be in the waiting state, if any of its called are not involved in an already ongoing connection.

$$waitingConx = askedConx - ongoingConx$$

Safety	active subscribers set contains some of the waiting subscribers

$$calledConx^{-1}[startedConx] \cap (calledConx^{-1}[askedConx - ongoingConx]) \neq \{\}$$

☞ Consequence: an asked connection is moved to ongoing if only all of its called subscribers are available (not involved in other ongoing connections): a guard of an event.

Structuring in Event B (with AtelierB 4.2)

```
SYSTEM
      ConnectMgr
SETS
      CONNECTION ; SUBSCRIBER /* the needed sets */
ABSTRACT VARIABLES
      ...
INVARIANT
      ... /* properties of variables */
* ---- The properties of the system ------*/
/* Safety SAF1, SAF2, ... */

INITIALISATION
...

END
```

Structuring in Event B

```
. . .(continued)
EVENTS
  newSubscriber = ... /* add a new subscriber ns */
;    initiateConnection = ...
  /* the initiation of a connection by sa, which calls some ss*/
;   res ← participantsConx = ...
  /* to get the participants(called) to a connection: caller
+ called */
;   startConnection = ... /* start one of the waiting connection,
  which does not have a subscriber already involved elsewhere
*/
;   endConnection = ... /* end one of the ongoing connections
*/
END
```


Further study

- Study of liveness properties using ProB
- Simulation of the system
- Refinement into code
- ...

Conclusion

We have seen

- a simple example of algorithmic development (GCD)
- a more complex example of analysis and modelling with Event B

This gives a quick overview of the B method.

We will then focus on the study and the practice of B.