

# Bank Account

Event-B model

Rigorous Software  
Development

Lampros Barmounis  
Stefan Vikoler  
Gautier Candaes

# Introduction

- Develop an Event-B model
- Bank account
  - Different actions :
    - open
    - withdraw
    - deposit
    - transfer

How did we manage ?

# Context definition

```
CONTEXT
    bankaccount_C0 >
SETS
[] CLIENTS >
[] ACCOUNTS >
CONSTANTS
[] k >
AXIOMS
[] axm1: finite(CLIENTS) not theorem >
[] axm2: finite(ACCOUNTS) not theorem >
[] axm3: k > 0 not theorem >
END
```

# Invariants

```
MACHINE
  bankaccount_M0 >
SEES
  bankaccount_C0
VARIABLES
  client >
  account >
  balance >
  ctransf >
INVARIANTS
  invAcc: account  $\subseteq$  ACCOUNTS not theorem >
  invBal: balance  $\in$  account  $\rightarrow$  N not theorem >
  invCli: client  $\in$  account  $\rightarrow$  CLIENTS not theorem >
  invTra: ctransf  $\in$  account  $\rightarrow$  N not theorem >
```

# Initialisation

```
EVENTS
[]  INITIALISATION:    extended ordinary >
    THEN
    []  iniCli: client = Ø >
    []  iniAcc: account = Ø >
    []  iniBal: balance = Ø >
    []  iniTra: ctransf = Ø >
    END
```

# Open event

```
open:    not extended ordinary ›
ANY
[] cli ›
[] acc ›
WHERE
[] isClient: cli ∈ CLIENTS not theorem ›
[] noRepeat: acc ∈ account not theorem ›
THEN
[] setBal: balance(acc) = 0 ›
[] setCli: client(acc) = cli ›
[] setTra: ctransf(acc) = 0 ›
END
```

# Deposit event

```
deposit:      not extended ordinary ›  
ANY  
  acc ›  
  amount ›  
  cli ›  
WHERE  
  isCli: cli ∈ CLIENTS not theorem ›  
  isAcc: acc ∈ account not theorem ›  
  isPos: amount ≥ 0 not theorem ›  
THEN  
  incBal: balance(acc) = balance(acc) + amount ›  
END
```



# Withdraw event

```
withdraw:  not extended ordinary >
ANY
[] acc >
[] amount >
[] cli >
WHERE
[] isAcc:  acc ∈ account not theorem >
[] isCli:  cli ∈ CLIENTS not theorem >
[] valCli: client(acc) = cli not theorem >
[] isPos:  amount ≥ 0 not theorem >
[] isEnough:  balance(acc) ≥ amount not theorem >
[] smallerK:  (ctransf(acc) + amount) ≤ k not theorem >
THEN
[] decBal:  balance(acc) := balance(acc) - amount >
[] setTra:  ctransf(acc) := amount >
END
```

# Transfer event

```
transfer: not extended ordinary ›
```

```
ANY
```

```
  fromAcc ›  
  toAcc ›  
  amount ›  
  cli ›
```

```
WHERE
```

```
  isAcc1: fromAcc ∈ account not theorem ›  
  isAcc2: toAcc ∈ account not theorem ›  
  isPos: amount ≥ 0 not theorem ›  
  isCli: cli ∈ CLIENTS not theorem ›  
  valCli: client(fromAcc) = cli not theorem ›  
  isEnough: balance(fromAcc) ≥ amount not theorem ›  
  smallerK: (ctransf(fromAcc) + amount) ≤ k not theorem ›  
  notSame: fromAcc ≠ toAcc not theorem ›
```

```
THEN
```

```
  setBal: balance = balance ◁ {fromAcc ↦ balance(fromAcc) - amount, toAcc ↦ balance(toAcc) + amount} ›  
  setTra: ctransf(fromAcc) = amount ›
```

```
END
```

# Proof obligations

- ▲ ✓ Proof Obligations
  - ✓<sup>A</sup> INITIALISATION/invBal/INV
  - ✓<sup>A</sup> INITIALISATION/invCli/INV
  - ✓<sup>A</sup> INITIALISATION/invTra/INV
  - ✓<sup>A</sup> open/invBal/INV
  - ✓<sup>A</sup> open/invCli/INV
  - ✓<sup>A</sup> open/invTra/INV
  - ✓<sup>A</sup> deposit/invBal/INV
  - ✓<sup>A</sup> deposit/incBal/WD
  - ✓<sup>A</sup> withdraw/valCli/WD
  - ✓<sup>A</sup> withdraw/isEnough/WD
  - ✓<sup>A</sup> withdraw/smallerK/WD
  - ✓<sup>A</sup> withdraw/invBal/INV
  - ✓<sup>A</sup> withdraw/invTra/INV
  - ✓<sup>A</sup> withdraw/decBal/WD
  - ✓<sup>A</sup> transfer/valCli/WD
  - ✓<sup>A</sup> transfer/isEnough/WD
  - ✓<sup>A</sup> transfer/smallerK/WD
  - ✓<sup>A</sup> transfer/invBal/INV
  - ✓<sup>A</sup> transfer/invTra/INV
  - ✓<sup>A</sup> transfer/setBal/WD

# Conclusions

- Faultless software
- Satisfy requirements
- Sensitive software