Egon Börger (Pisa) & Alexander Raschke (Ulm)

Modeling an Automatic Teller Machine

Illustrating Componentwise and Stepwise ASM Definition

Università di Pisa, Dipartimento di Informatica boerger@di.unipi.it
Universität Ulm, Abteilung Informatik alexander.raschke@uni-ulm.de

See Ch. 2.4 of Modeling Companion

¹ Unless stated otherwise, some figures are ©2015 ACM and are used with licence 4271320674209.
PlantReq. There are many tills which can access a central resource containing the detailed records of customers’ bank accounts.

TillAccessReq. A till is used by inserting a card and typing in a PIN which is encoded by the till and compared with a code stored on the card.

FunctionalReq. After successfully identifying themselves to the system, customers may try to:

1. view the balance of their accounts,
2. make a withdrawal of cash,
3. ask for a statement of their account to be sent by post.

Information on accounts is held in a central database and may be unavailable. If the database is available, any amount up to the total in the account may be withdrawn, subject to a fixed daily limit on withdrawals.
ATM requirements (2)

CardReq. The fixed daily limit on withdrawals means that the amount withdrawn within the day must be stored on the card. "Illegal" cards are kept by the till.

InterruptReq. A till or the Central Resource can be interrupted and the connection between them can fail.

CustomerInterruptReq. Customers can change or cancel their request any time, e.g. stop the usage, change amount they want to withdraw.

ConcurrencyReq. Concurrent access to the database from two or more different tills is allowed, in particular concurrent attempts from two card holders who are authorised to use the same account.

TransactionalReq. Once a user has initiated a transaction, the transaction is completed at least eventually, and preferably within some real time constraint.

ReliabilityReq. Minimise the possibility of the use of stolen cards to gain access to an account.
ATM interface structure

Figure © 2018 Springer-Verlag Germany, reused with permission.
**ATM controller architecture (Component Structure View)**

*TillAccessReq* and *FunctionalReq* request a sequence of actions reflected by **sequential composition** of ground model out of action components:

```
Idle
→ PROCESSCardInsertion
→ StartPinRequest
→ PROCESSPinRequest
→ ValidPin
→ PROCESSOpRequest
→ Ready(ContactCR)
→ PROCESSCRContact
→ WaitFor(ContactCR)
→ PROCESSCRResponse
→ Terminate
→ HANDLEFailure
```

Component failure triggers **exit to HANDLEFailure** component.
In parallel to the ‘normal’ session execution by \texttt{Atm}, at any moment \texttt{InterruptTriggers} may occur and must be handled

- by \texttt{CustomerInterruptReq} and \texttt{InterruptReq}

\begin{verbatim}
\texttt{GROUNDAtm} =
\begin{align*}
\text{if } & \text{ThereAreInterrupts} \\
\rightarrow & \text{HANDLEINTERRUPT} \\
\text{else} & \\
\rightarrow & \text{HANDLEFAILURE} \\
\text{INTERRUPTTRIGGER}
\end{align*}
\end{verbatim}

We now procedurally refine each component of \texttt{GROUNDAtm}

- to capture/complete one by one the corresponding \texttt{AtmRequirements}
Procedural ASM refinement follows Knuth’s advice (1974):

\[
\ldots \text{we rapidly lose our ability to understand larger and larger flowcharts; some intermediate levels of abstraction are necessary.}
\]

\[
\ldots \text{we should give meaningful names for the larger constructs in our program that correspond to meaningful levels of abstraction, and we should define those levels of abstraction in one place, and merely use their names (instead of including the detailed code) when they are used to build larger concepts.}
\]

In general multiple entries/exits and arbitrary—even run-time determined—step relations \((m, n)\) are allowed.

\[
\text{Diagram: } i \xrightarrow{\text{rule}} j \text{ by } i \xrightarrow{} k_1 \ldots k_n \xrightarrow{} j
\]
Card insertion: questions about requirements

CardInserted: monitored location
- becoming true/false upon physical card insertion/removal

A number of questions about the requirements:
- ATM presumably assumed to be used any time only by one user
  – i.e. new session can be started only when till is in mode = idle
- card validity check presumably assumed
  – InitializeSession and StartPinRequest only if can ReadCard
  - upon Fail(InvalidCard) move to HANDLEFAILURE
- meaning of ValidCard?
  
  $ ValidCard = \text{Readable}(\text{insertedCard}) \text{ and } \text{circuit}(\text{currCard}) \in \text{Circuit} $
Card insertion: meaning of \texttt{ReadCard}

Domain experts must decide which are the attributes the reader can retrieve from a card, so that the till can manage the session:

- \texttt{circuit(card)} describing the card type, \texttt{pinCode(card)}, \texttt{account(card)}
- \texttt{centralResource(card)} where the \texttt{account(card)} is managed
- \texttt{dailyLimit(card)}
- \texttt{alreadyWithdrawn(day, card)} indicating the total amount of money withdrawn this \texttt{day} (as a date) in previous sessions at some tills using \texttt{card}
- \texttt{dayOfLastWithdrawal(card)}

Abstract from how \texttt{ReadCard} records card attributes:

- \texttt{currCard := insertedCard with derived function attribute(currCard) = attributeValReadFrom(insertedCard)}
Due to \textit{CardReq}, the location \textit{alreadyWithdrawn}($today$, $card$) must be updated, say in a component \texttt{InitializeSession}:

\begin{verbatim}
if dayOfLastWithdrawal($card$) < $today$ then
    alreadyWithdrawn($today$, $card$) := 0
\end{verbatim}

Therefore $today$, which is monitored for \texttt{ProcessCardInsertion}, must be assumed to be updated at midnight by a \texttt{Calendar} component of the ATM.
Formally this procedural refinement is of type \((1, 1)\)

- because reading the guard goes together with writing the updates, whatever is performed in passing from one to the next control state counts as one step
‘Typing in a PIN’ in TillAccessReq describes one kind of user/machine interaction:
– till will AskFor(Pin) and check whether input is a ValidPin
We make AskFor reusable as parameterized component
– parameter specifies the type of interaction object

We separate a detailed component definition— involving processing keywise provided input streams—from its interface behavior spec
– elaborate and store the monitored userInput value asked for in a location, say valFor(param), and enter mode Ready(param)

\[
\text{AskFor}(\text{param}) = \\
\text{valFor}(\text{param}) := \text{userInput} \\
\text{mode} := \text{Ready}(\text{param}) \\
\text{ResetTimer}(\text{AskFor}(\text{param}))
\]

Copyright CC BY–NC-SA 4.0
Definition of **ProcessPin**

Consider *additional req*: user **HasMoreAttempts** to input pin until **ValidPin** or **Fail(InvalidPin)** or timeout interrupt.

\[ \text{ValidPin} = (\text{pinCode}(\text{currCard}) = \text{encode}_{Pin}(\text{valFor}(\text{Pin}))) \]

where by **TillAccessReq** \(\text{pinCode}\) extracts the ‘code stored on the card’ and \(\text{encode}_{Pin}\) performs the ‘encoding by the till’
Parameterized ASMs are called by name

Declaration of parameterized ASM $N(x_1, \ldots, x_n) = M$
- where $M$ is an ASM whose free variables occur in $x_1, \ldots, x_n$

permits to call $N(exp_1, \ldots, exp_n)$ (by name) whereby
- body $M$ of the machine declaration is executed with the variables $x_i$
  substituted by the call parameters $exp_i$ (not by their values)
  - call parameters are evaluated only in the state in which the body is executed
- executing a submachine call is treated as one atomic step
  - $M$ may contain recursive calls of $N$
  which yields a defined result only if the execution of the machine body yields a defined result

NB. call by value is definable by

$$N(exp_1, \ldots, exp_n) = \text{let } (x_1 = exp_1, \ldots, x_n = exp_n) \text{ in } M$$
Component **PROCESSOPREQUEST**

By FunctionalReq users have *OpChoice* to ask for their account balance or an account statement or a cash withdrawal

- **AskFor**(*OpChoice*) captures this choice
- if *op = Withdrawal* the till acquires further *RequiredData* via **AskFor**(*Amount*)

FunctionalReq also requests to **CHECKLOCALAVAILABILITY** of the requested money

- whether *AmountExceedsDailyLimit*
- to which (for the sake of illustration) we add an *AmountATMUnavailability* check
**ProcessOpRequest** component of **Atm**

- **ValidPin**
  - SetTimer(AskFor(OpChoice))
    - AskFor(OpChoice)
    - AskFor(OpChoice)
    - Ready(OpChoice)
- RequiredData(valFor(OpChoice))
  - yes: SetTimer(AskFor(Amount))
    - AskFor(Amount)
  - no: CheckLocalAvail
- Ready(ContactCR)
- Fail(AmountAtmUnavail)
- Fail(AmountExceedsDailyLimit)
- Ready(Amount)
For \texttt{CheckLocalAvail} we show how to \textit{specify an interface behavior}: to which next control state the component may proceed depending on the underlying data: normal exit \texttt{Ready(ContactCR)} or a \texttt{Failure} exit

\begin{verbatim}
\texttt{CheckLocalAvail =}
\texttt{ choose m \in NxtCtlState // abstract from data determining m}
\texttt{ mode := m}
\texttt{ if m = Ready(ContactCR) then}
\texttt{ amount := valFor(Amount)}
\end{verbatim}

\begin{verbatim}
\texttt{where NxtCtlState =}
\texttt{ \{ Ready(ContactCR),}
\texttt{ Fail(AmountAtmUnavail),}
\texttt{ Fail(AmountExceedsDailyLimit)\}}
\end{verbatim}
Component \textit{ProcessCRContact}

- triggers a request that is sent to Central Resource (CR)
- makes the till \texttt{WaitFor(ContactCR)} until a response is received
  – by \textit{InterruptReq} unless a timeout or contact failure happen
\textbf{SEND}(\textit{encode}_\textit{till}(Atm, CR, RequestData))
\textbf{DISPLAY}(\textit{WaitingForCentralResourceContact})

where

\begin{align*}
Atm &= \text{address}(\textit{till}(\textit{self})) \\
CR &= \text{address}(\text{centralResource}(\textit{currCard})) \\
RequestData &= \text{opChoiceData}(\textit{currCard}, \text{valFor}(\textit{OpChoice})) \\
opChoiceData(c, o) &=
\begin{cases}
(c, o) & \text{if } o \in \{\text{Balance, Statement}\} \\
(c, o, \text{amount}) & \text{if } o = \text{Withdrawal}
\end{cases}
\end{align*}

Interface to \textbf{PROCESSCRRESPONSE} component:

- monitored location \textit{CRresp} where response messages from the Central Resource (CR) are received
ProcessCRResponse

- by *InterruptReq* a *ConnectionRefused* response may arrive
- other *CRresponses* lead the till to normally *PROCESSRESPONSE*
**ProcessResponse**($r$)

- **type($r$) = IllegalCard**
  - **TerminateOp**($r$, \{Keep(currCard)\})

- **type($r$) \in \{InfoUnavailable, UnknownCard, Balance, Statement\}**
  - **TerminateOp**($r$, \{Eject(currCard)\})
  - **TerminateOp**($r$, \{Eject(currCard), Eject(amount)\})

- **type($r$) = Withdrawal**
  - **answer($r$) = notOk**
  - **answer($r$) = Ok**

**TerminateOp**($reason$, $actions$) = **Display**($reason$)  
--- explain action to the user

TerminationActions := $actions$  
--- executed by **Terminate**

Copyright CC BY–NC-SA 4.0
Component TERMINATE

- **RecordMoneyWithdrawalOnCard**
- **Keep(currCard) ∈ TerminationActions**
  - yes → **Retract(currCard)** → Idle
  - no → **Eject(currCard)** → Idle
  - no → **SetTimer(Removal)** → Idle
  - no → **WaitFor(Removal)** → Idle
  - no → **Removed(currCard) → Retract(currCard)** → Idle

- **Eject(amount) ∈ TerminationActions**
  - yes → **Eject(amount)** → Idle
  - yes → **Display(amount)** → Idle
  - yes → **SetTimer(Removal)** → Idle
  - yes → **WaitFor(Removal)** → Idle
  - yes → **Removed(amount) → Retract(money)** → Idle

- **RecordMoneyWithdrawalAtATM(amount)**

- Idle
**Terminate macros**

\[ \text{Retract}(o) = \]
\[ \text{Remove}(o) \quad -- \text{physically remove card or money from slot} \]
\[ \text{LogMissedWithdrawal}(o) \quad -- \text{if applicable} \]

\[ \text{RecordMoneyWithdrawalOnCard} = \]
\[ \text{if} \ \text{MoneyWithdrawalToRecord} \ \text{then} \]
\[ \text{alreadyWithdrawn}(\text{today, currCard}) := \]
\[ \text{amount} + \text{alreadyWithdrawn}(\text{today, currCard}) \]
\[ \text{dayOfLastWithdrawal}(\text{currCard}) := \text{today} \]
\[ \text{MoneyWithdrawalToRecord} := \text{false} \]

\[ \text{RecordMoneyWithdrawalAtAtm}(o) = \]
\[ \text{money}(\text{Atm}) := \text{money}(\text{Atm}) - o \]
**HandleFailure**: an example

**Modular (case-by-case) definition via parameterization** of $\text{Fail}(\text{param})$ values of $\text{mode}$:

\[
\text{HandleFailure} = \quad \text{-- called when } \text{mode} = \text{Fail}(\text{param})
\]

\[
\begin{align*}
\text{if} & \quad \text{mode} = \text{Fail}(\text{InvalidPin}) \quad \text{then} \\
& \quad \text{TerminatEOP}(\text{InvalidPin}, \{\text{Keep}(\text{currCard})\}) \\
\text{else} & \quad \text{TerminatEOP}(\text{mode}, \{\text{Eject}(\text{currCard})\}) \\
\text{CloseConnectionToCentralResource}
\end{align*}
\]

- **Eject** unreadable cards, cards of not accepted circuits, etc.
- **NB.** cards the CR declares as $\text{IllegalCard}$ are kept by the corresponding $\text{TerminatEOP}$eration

Copyright CC BY–NC-SA 4.0
Interrupts with interrupt region

**Modular (case-by-case) definition** via

- parameterization
- separately definable concept of *interrupt region* where interrupt events should have an effect

Exl: interrupt is triggered (inserted into `InterruptEvent`):

- when user has *Pressed* the `CancelKey` and `Atm IsInCancelRegion`
- automatically upon a `Timeout(timedOp)` event when the `Atm IsInTimerRegion` for the `timedOp`

\[
\text{INTERRUPT_TRIGGER} = \\
\text{INTERRUPT_BY} (\text{Cancel}) \hspace{1cm} \text{INTERRUPT_BY} (\text{Time})
\]

\[
\ldots
\]
Cancel and timeout interrupts

\textbf{InterruptBy}(\textit{Cancel}) =

\textbf{if} \textit{Pressed}(\textit{CancelKey}) \textbf{and} \textit{IsInCancelRegion}(\textit{ATM})

\textbf{then} \textbf{INSERT}(\textit{Cancel}, \textit{InterruptEvent})

\textbf{InterruptBy}(\textit{Time}) =

\textbf{forall} \textit{timedOp} \in \{\textit{AskFor}(\textit{param}), \textit{ContactCR}, \textit{Removal}\}

\textbf{if} \textit{Timeout}(\textit{timedOp}) \textbf{and} \textit{IsInTimerRegion}(\textit{timedOp}) \textbf{then}

\textbf{INSERT}(\textit{timer}(\textit{timedOp}), \textit{InterruptEvent})

\textbf{RESETTIMER}(\textit{timedOp})
**HandleInterrupt with priority scheme**

\[
\text{HandleInterrupt} = \begin{align*}
\text{let } e &= \text{highPriority}(\text{InterruptEvent}) \\
\text{Handle}(e) &\quad \text{DELETE}(e, \text{InterruptEvent})
\end{align*}
\]

where

\[
\text{Handle}(\text{Cancel}) =
\begin{align*}
\text{if } \text{IsInCancelRegion}(\text{Atm}) \text{ then } &\quad \text{TerminateSession}(\text{Cancel}) \\
\text{Handle}(\text{timer}(\text{timedOperation})) &=
\begin{align*}
\text{if } \text{IsInTimerRegion}(\text{timedOperation}) \text{ then } &\quad \text{TerminateSession}(\text{Timeout}(\text{timedOperation}))
\end{align*}
\]

\[
\text{TerminateSession}(p) =
\begin{align*}
\text{DisconnectAtmFromCR} \\
\text{TerminateOp}(p, \text{Eject}(\text{currCard}))
\end{align*}
\]

mode := Terminate
Defining interrupt regions

In control state ASMs interrupt regions are definable by \textit{mode} intervals.

- Exl: no \textit{Cancel} command has any effect outside a user session (when \textit{mode} = \textit{idle}) or when the ATM is performing automatically its final stage to \textbf{TERMINATE} the session

\[
\text{IsInCancelRegion}(\text{ATM}) = \text{mode} \notin \{\text{Idle, Terminate}\}
\]

Analogously for timer regions:

\[
\text{IsInTimerRegion}(\text{AskFor}(\text{param})) = \\
\text{mode} \in \{\text{AskFor}(\text{param}), \text{WaitFor}(\text{param})\}
\]

\[
\text{IsInTimerRegion}(\text{ContactCR}) = \\
\text{mode} = \text{WaitFor}(\text{ContactCR})
\]

\[
\text{IsInTimerRegion}(\text{RemovalCard}) = \\
(\text{mode} \in \\
\{\text{WaitFor}(\text{RemovalCard}), \text{WaitFor}(\text{RemovalMoney})\})
\]
works asynchronously together with multiple ATMs
to satisfy the ConcurrencyReq, our spec permits any processing order for independent requests
– separate priority and scheduling concerns from per-account-exclusive access guarantee in FunctionalReq

**CentralResource** =

one of (AcceptRequests, HandleRequests)

where AcceptRequests =

if Mailbox_{CR} \neq \emptyset then

choose R \subseteq Mailbox_{CR} with R \neq \emptyset

forall msg \in R -- move them from mailbox into internal record

\text{INSERT}(\text{decode}_{CR}(msg), \text{Request})

\text{DELETE}(msg, Mailbox_{CR})
Let $select_{CR}$ be any policy for selecting a *Consistent* set of requests for a parallel handling.

**HandleRequests** =

- if $Request \neq \emptyset$ then
  - let $R = select_{CR}(Request)$
  - forall $r \in R$
    - Handle$(r)$
    - Delete$(r, Request)$

*Consistent$(R)$ iff* -- no two withdrawals from one account

- there is no $r, r' \in R$ with $r \neq r'$ and
  - account$(r) = account(r')$ and $op(r) = op(r') = \text{Withdrawal}$
HANDLE\( (req) \) component of \( \text{Atm} \)

How CR elaborates a correct \( CR\text{response} \) of type \( op(\text{req}) \):

\[
\text{let } \text{atm} = \text{sender}(\text{req}), \text{card} = \text{card}(\text{req}), \text{account} = \text{account}(\text{card}), \text{op} = \text{op}(\text{req}), \text{amount} = \text{amount}(\text{req})
\]

- **Known(card)**
  - yes: \( \text{SEND(atm, UnknownCard)} \)
  - no: \( \text{SEND(atm, IllegalCard)} \)

- **Blocked(card)**
  - yes: \( \text{SEND(atm, IllegalCard)} \)
  - no:
    - **op = Statement**
      - yes: \( \text{SEND(atm, InfoUnavailable)} \)
      - no: \( \text{SEND(atm, InfoUnavailable)} \)
    - **op = Balance**
      - yes: \( \text{SEND(atm, info(account))} \)
      - no: \( \text{SEND(atm, InfoUnavailable)} \)
    - **op = WithDrawal**
      - yes: \( \text{SEND(atm, Ok, amount)} \)
      - no: \( \text{SEND(atm, notOk)} \)

- **CanBeGranted(\text{amount, account})**
  - yes:
    - **op = WithDrawal**
      - yes: \( \text{SEND(atm, Ok, amount)} \)
      - no: \( \text{SEND(atm, notOk)} \)
  - no:
    - **op = Statement**
      - yes: \( \text{SEND(atm, InfoUnavailable)} \)
      - no:
        - **op = Balance**
          - yes: \( \text{SEND(atm, bal(account))} \)
          - no: \( \text{SEND(atm, InfoUnavailable)} \)
        - **op = WithDrawal**
          - yes: \( \text{SEND(atm, Ok, amount)} \)
          - no: \( \text{SEND(atm, notOk)} \)

- **TrIGGERMAIL(\text{req})**

Copyright CC BY–NC-SA 4.0
Integrate data into ctl flow: by data refinement (type $(1, 1)$)

**CheckLocalAvail** = choose $m \in NxtCtlState$

$mode := m$

if $m = \text{Ready}(\text{ContactCR})$ then $amount := \text{valFor}(\text{Amount})$

Refinement computing how $mode$ update depends on data:

```
Ready(Amount)
   \[\rightarrow\]
WithinDailyLimit
   \[\rightarrow\]
AtmAvail
   \[\rightarrow\]
Fail(AmountExceedsDailyLimit)
   \[\rightarrow\]
Fail(AmountAtmUnavail)
```

Copyright CC BY–NC-SA 4.0
Integrate data into ctl flow: by procedural refinement

Ex: combined data and operation refinement of \texttt{AskedFor} =

\[
\text{valFor}(\text{param}) := \text{userInput} \quad \text{mode} := \text{Ready}(\text{param})
\]

\texttt{ResetTimer(AskedFor(param))}

Idea: implement successive reading and processing of single input key values inserted by the user as follows:

- start to \texttt{InitializeInputElaboration}
  - \texttt{Display} request to user
  - guarantee robustness: keys pressed before start of \texttt{WaitFor(param)} should yield no input
- iterate \texttt{ReadInputStream} and \texttt{ProcessInputStream}
- upon a \texttt{Confirm} key input move to \texttt{Ready(param)}
**Procedural refinement of** \texttt{AskedFor} **component**

\begin{align*}
\text{AskedFor} &= \text{valFor}(\text{param}) := \text{userInput}, \\
\text{mode} &:= \text{Ready}(\text{param}), \text{RESETTIMER}(\text{AskedFor}(\text{param}))
\end{align*}
\textbf{Initialization component of refined \texttt{AskedFor}}

\begin{verbatim}
\texttt{InitializeInputElaboration(param)} =
\texttt{Initialize(inputStream)} \quad \text{-- Start listening to user input}
\texttt{Initialize(userInput)} \quad \text{-- Start processing user input}
\texttt{Display(AskFor(param))} \quad \text{-- Ask user for param}
\texttt{if param = Pin then CountDown(attemptsFor(Pin))}
\end{verbatim}

The auxiliary macros are defined as follows:

\begin{verbatim}
\texttt{Initialize(Stream)} = (Stream := [])
\texttt{Initialize(userInput)} = (userInput := [])
\texttt{CountDown(attemptsFor(Pin))} =
\texttt{attemptsFor(Pin) := attemptsFor(Pin) - 1}
\end{verbatim}

NB. An initialization of \texttt{attemptsFor(Pin)} belongs to (for example) \texttt{InitializeSession}.
what if a user hits simultaneously a set of multiple keys?
- hardware transforms the set into a randomly ordered `inputStream`
- before applying `randomOrder` to a `set`, the hardware will `truncate(set)` in a device dependent manner to a subset
- `inputVal` yields input value sequence for key sequence

**Read/ProcessInputStream component**

```
Read/ProcessInputStream =

let PressedKeys = {key | Pressed(key)}
let Newinput =
    inputval(randomOrder(truncate(PressedKeys)))
    AddAtTheLeft(Newinput, inputStream)

ProcessInputStream =

if inputStream ≠ [] then
    let val = fstOut(inputStream) -- say rightmost element
    RemoveAtTheRight(val, inputStream)
```
UpdateInputBy component

- writes the inputStream values that are LegalFor param into userInput
- since user can change the input any time (CustomerInterruptReq), Delete key is LegalFor every param

\[
\text{UpdateInputBy}(val, \text{param}) =
\begin{align*}
\text{if } val \neq \text{Delete } & \text{ then } \text{ AddToInput}(val, \text{param}) \\
\text{if } val = \text{Delete } & \text{ then } \text{ RemoveFromInput}(\text{param})
\end{align*}
\]

\[
\text{AddToInput}(val, \text{param}) =
\begin{align*}
\text{userInput} & := \text{concatenateAtTheRight}(\text{userInput}, val) \\
\text{Display}(\text{concatenateAtTheRight}(\text{userInput}, val), \text{param})
\end{align*}
\]

\[
\text{RemoveFromInput}(\text{param}) =
\begin{align*}
\text{userInput} & := \text{removeLast}(\text{userInput}) \\
\text{Display}(\text{removeLast}(\text{userInput}), \text{param})
\end{align*}
\]
the input is recorded in interface location $valFor(param)$

due to in-time termination the timer is reset

mode switches to $Read(param)$

$$Confirming(val, param) \text{ if and only if }$$

$$\begin{cases} 
\text{param} \in \{\text{Pin, Amount}\} \text{ and } val = \text{Confirm} \\
\text{param} = \text{OpChoice} \text{ and } val \in \{\text{Balance, Statement, Withdrawal}\}
\end{cases}$$

$$\text{Record}(input, param) =$$

$$valFor(param) := input \text{ if } param \in \{\text{Pin, Amount}\}$$

$$valFor(param) := param \text{ if }$$

$$\text{param} \in \{\text{Balance, Statement, Withdrawal}\}$$
References

- Requirements from: Automatic Teller Machine or Till: Case Study
  - *The FM’99 ATM modelling challenge*
- S. Zenzaro: A CoreASM refinement implementing the ATM ground model.
  - available at http://modelingbook.informatik.uni-ulm.de (October 2014)
- E. Börger and A. Raschke: Modeling Companion for Software Practitioners. Springer 2018
  http://modelingbook.informatik.uni-ulm.de
It is permitted to (re-) use these slides under the CC-BY-NC-SA licence

https://creativecommons.org/licenses/by-nc-sa/4.0/

i.e. in particular under the condition that

- the two original authors are mentioned
- modified slides are made available under the same licence
- the (re-) use is not commercial