# From Use Cases to Post Conditions

*Martin Giese* and Rogardt Heldal

Chalmers Tekniska Högskola,

Göteborg, Sweden

## Motivation

. . . for the KeY crowd:

Until now, the KeY method starts at the design phase:

- add OCL pre/post conditions to methods
- instantiate design patterns
- Constraints like argument ranges, objects being non-null

What is missing is the analysis phase:

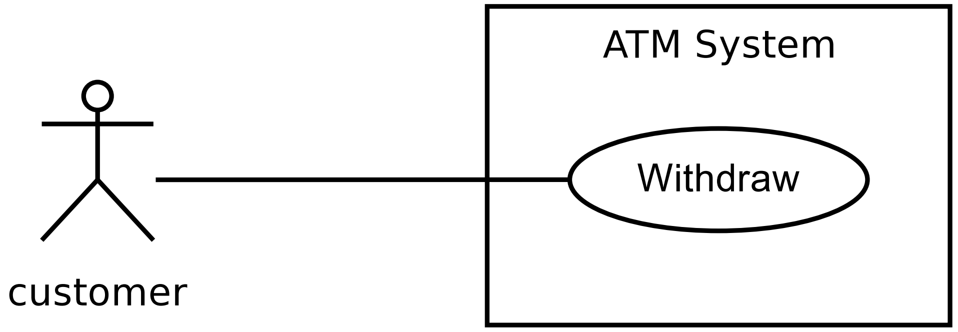
- Specifications that correspond to customer requirements
- Done before the first class diagram
- Textual descriptions

## Informal Specification in UML

- UML *use cases* are a means for *informal* specification
- Describe the system’s behaviour
- Suitable for communication with customers
- Parts of use case:
  - name
  - main flow
  - alternative flows
  - pre and *post* conditions
  - . . .

## Example: Automated Teller Machine

- In this case study, look only at withdrawal:



- Main flow:

  insert card, enter PIN, request cash amount, get money, return card

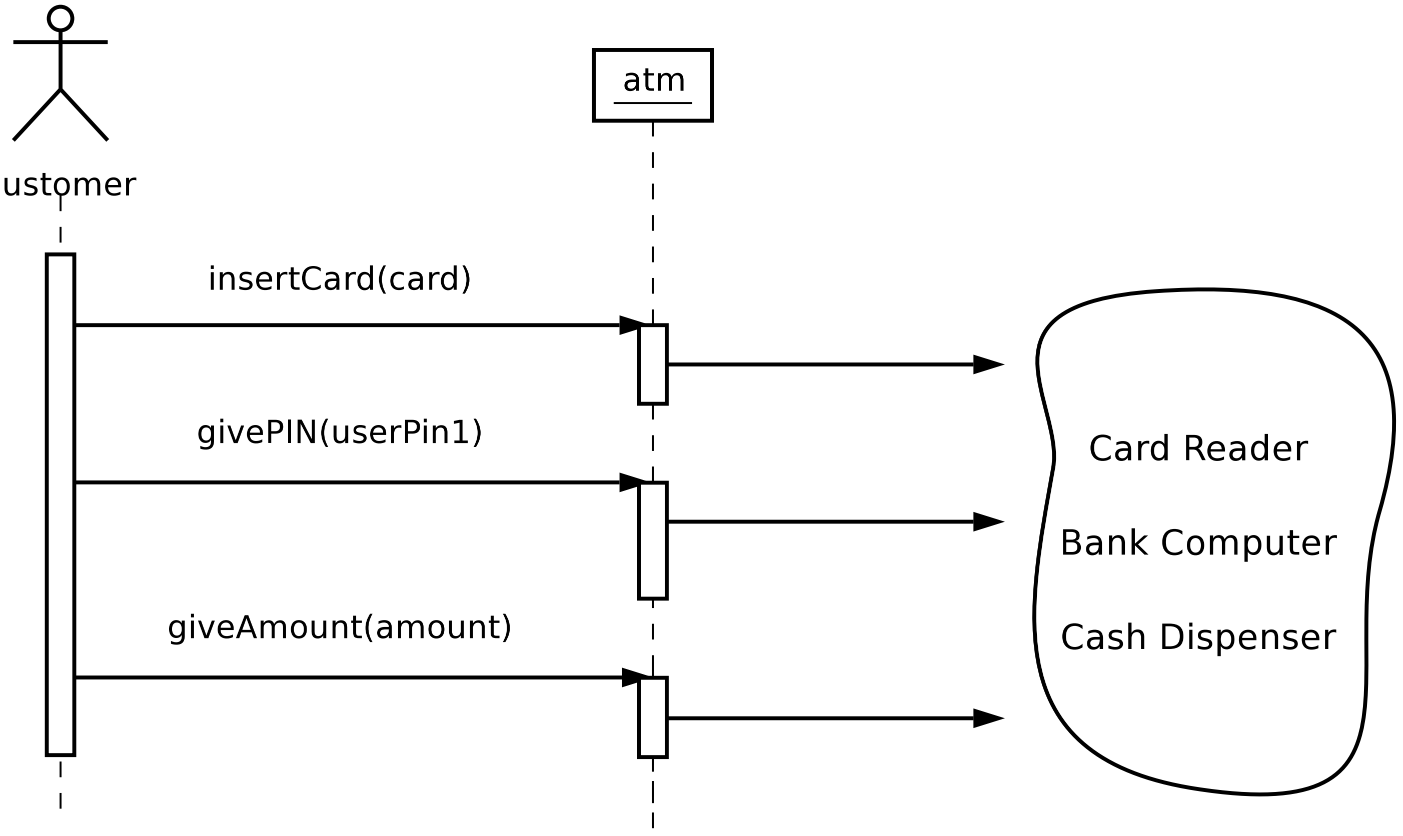
- Alternative flows: wrong PIN, cash amount exceeds balance, etc.

- In this work: put a more precise informal description in the *post condition* of the use case. . .

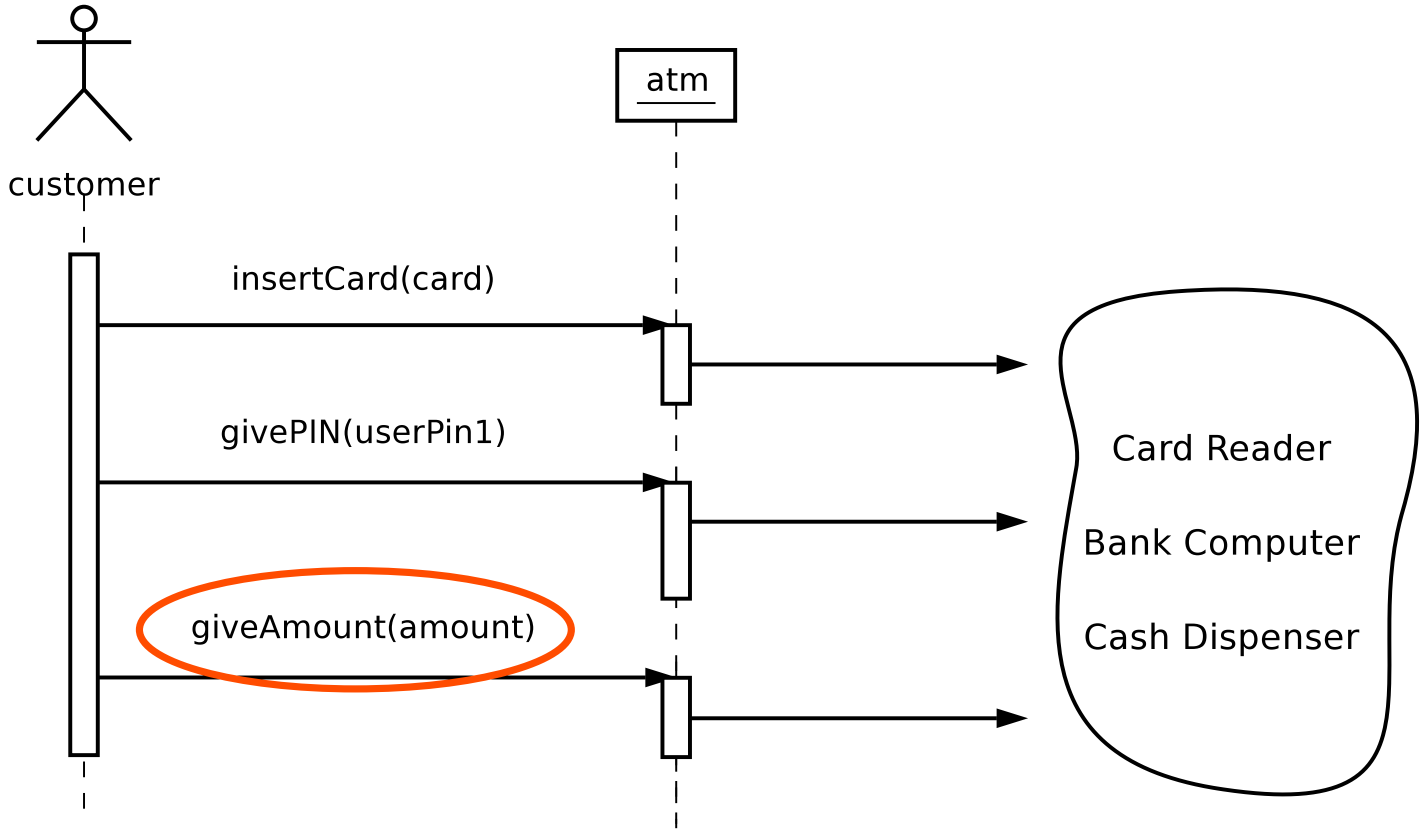
## Post Condition of Withdraw

- If the customer entered the PIN on the Card, and the customer's balance was greater or equal to the requested amount, then the customer got the requested amount and the amount was deducted from the balance.
- If the customer entered the wrong PIN three times, the card was retained.
- If the customer requested too much money, the card was returned to the customer.

# Main Scenario: Sequence Diagram



## Main Scenario: Sequence Diagram



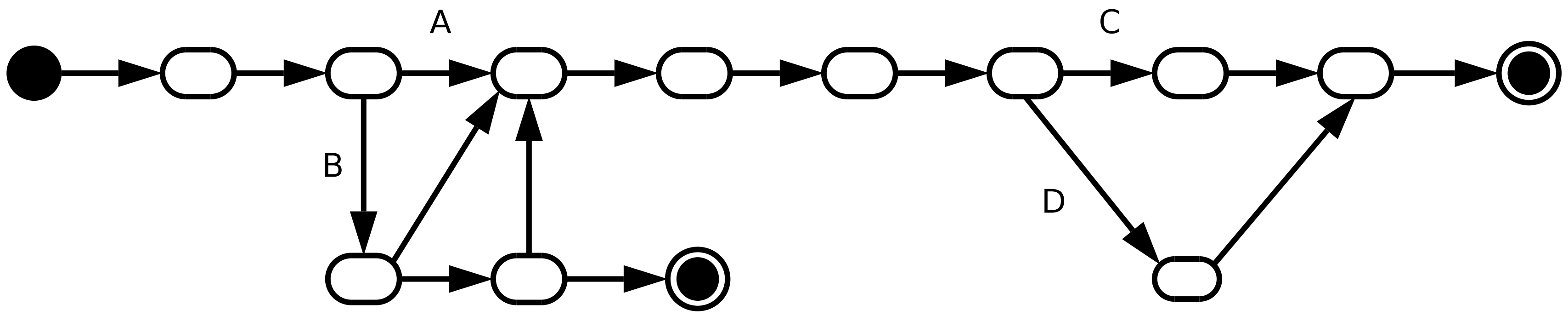
➠ Last Operation (giveAmount) must ensure post condition of use case

## Informal to Formal Specification

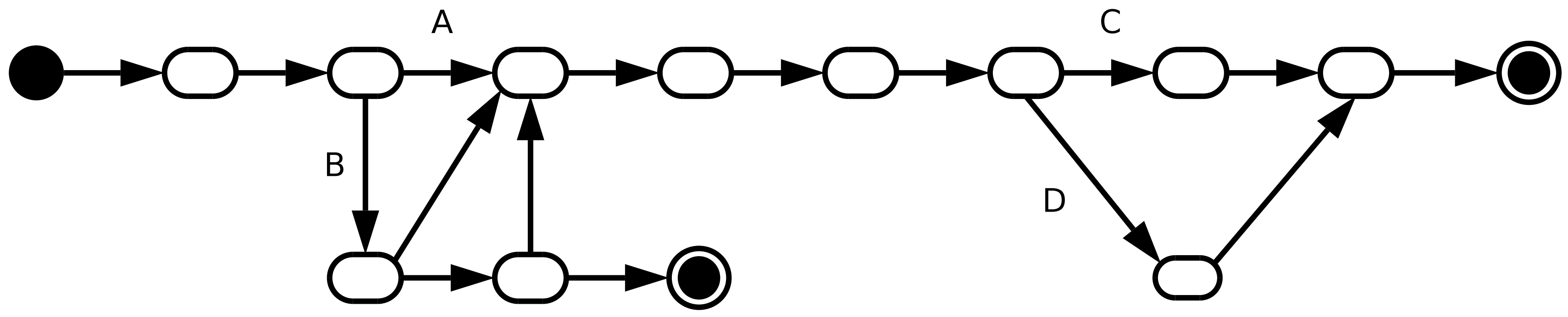
- Want to give the formal post-condition for the final operation in the scenario, for example operation giveAmount
- Guarantee that the formal post-condition of the operation satisfy the informal post-condition of the use case
- Problem:
  - There might be several scenarios of the use case
  - The final operation can vary for different scenarios

➠ Gather possible scenarios in state chart

## State Chart for ATM Example

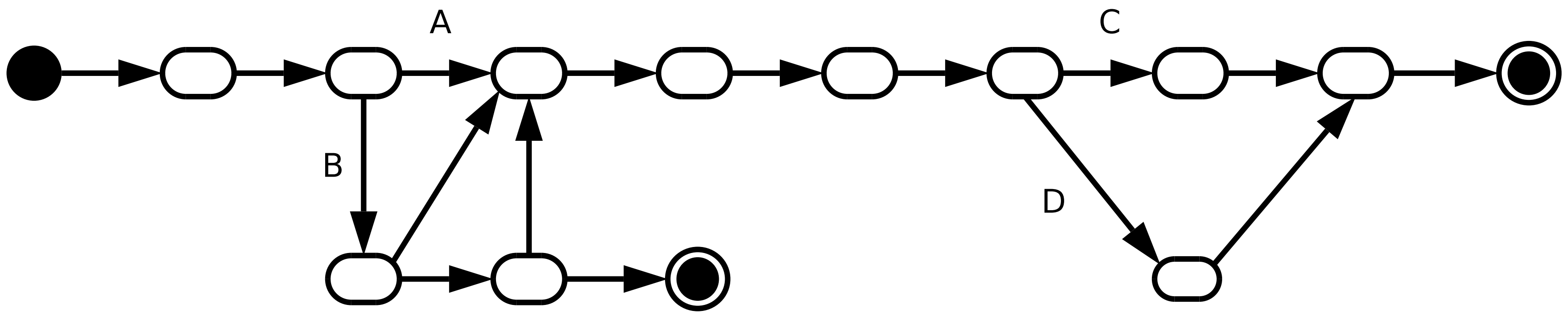


# State Chart for ATM Example



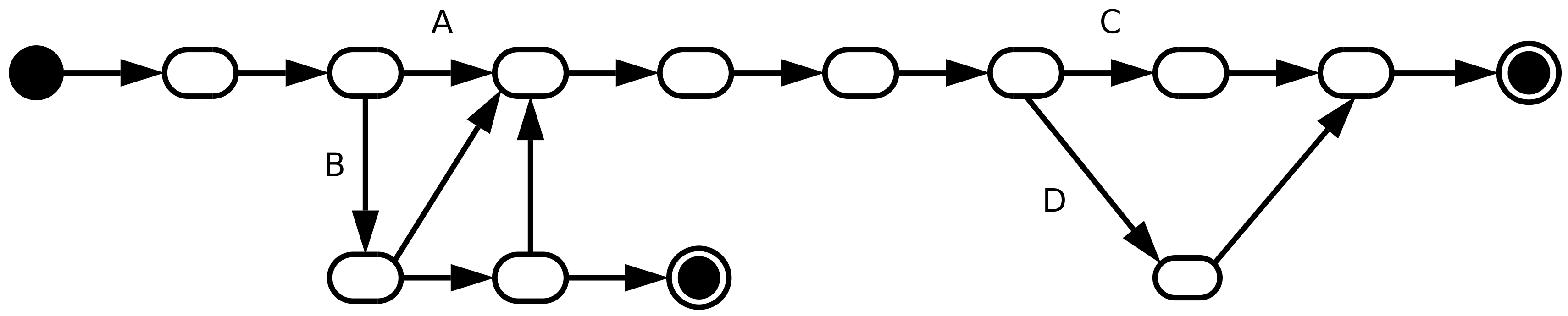
- A: `givePin(userPin1) [userPin1 = cardPin1]`

# State Chart for ATM Example



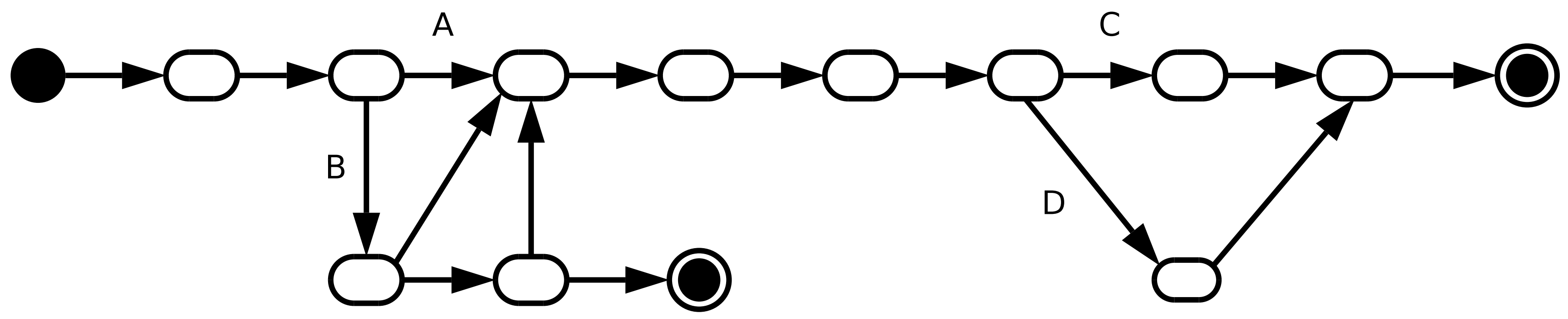
- A: `givePin(userPin1)[userPin1 = cardPin1]`
- B: `givePin(userPin1)[userPin1 <> cardPin1]`

# State Chart for ATM Example



- A: `givePin(userPin1)[userPin1 = cardPin1]`
- B: `givePin(userPin1)[userPin1 <> cardPin1]`
- C: `[balance >= amount]`

## State Chart for ATM Example



- A: `givePin(userPin1)[userPin1 = cardPin1]`
- B: `givePin(userPin1)[userPin1 <> cardPin1]`
- C: `[balance >= amount]`
- D: `[balance < amount]`

## Condition to Satisfy: First Attempt

Given a path $\pi$:

$$\begin{tikzcd}
\boxed{s_0} \arrow[r, "op_1(args_1)"] & \boxed{s_1} \arrow[r, "op_2(args_2)"] & \boxed{s_2} \arrow[r] & \ldots \arrow[r, "op_k(args_k)"] & \boxed{s_k} \arrow[r] & \odot
\end{tikzcd}$$

- Last operation for path: $\mathit{final}(\pi) := op_k$
- Post cond. of last method should ensure post cond. of use case:

$$Post_{\mathit{final}(\pi)} \rightarrow Post_{UC}$$

## Condition to Satisfy: First Attempt

Given a path $\pi$:

$$\begin{tikzcd}
\boxed{s_0} \arrow[r, "op_1(args_1)"] & \boxed{s_1} \arrow[r, "op_2(args_2)"] & \boxed{s_2} \arrow[r] & \cdots \arrow[r, "op_k(args_k)"] & \boxed{s_k} \arrow[r] & \odot
\end{tikzcd}$$

- Last operation for path: $\mathit{final}(\pi) := op_k$
- Post cond. of last method should ensure post cond. of use case:

$$Post_{\mathit{final}(\pi)} \rightarrow Post_{UC} \qquad \textit{wrong!}$$

- Problem:

  Holds only if we add information about the path taken

## Information about a Path

Let $Cond(\pi)$ gather information about the path taken.

$$\begin{tikzcd}
s_0 \arrow[r, "op_{1a}(args_{1a})[cond_{1a}]"] \arrow[dr, "op_{1b}(args_{1b})[cond_{1b}]"'] & s_{1a} \arrow[r, "op_2(args_2)[cond_2]"] & s_2 \arrow[r] & \cdots \\
 & s_{1b} \arrow[r] & \cdots &
\end{tikzcd}$$

# Information about a Path

Let $Cond(\pi)$ gather information about the path taken.

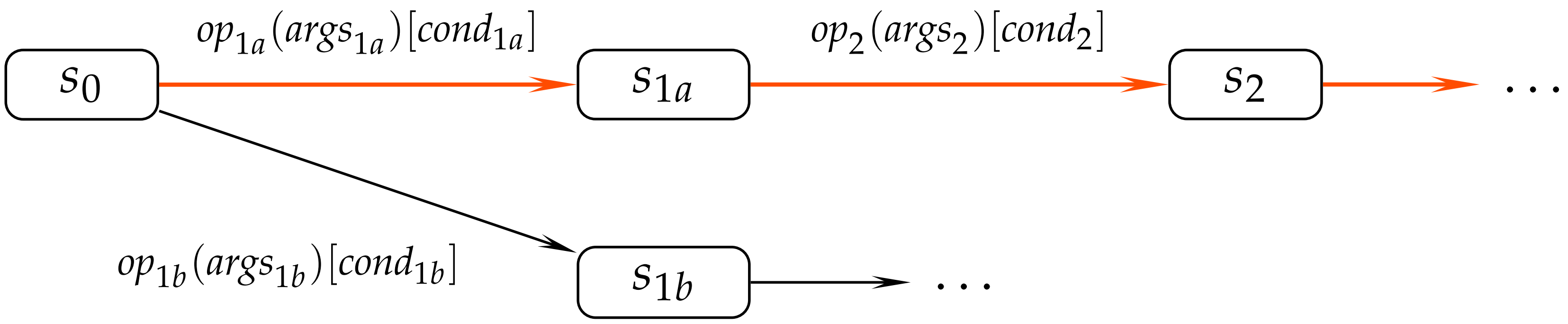
$$\begin{tikzcd}
\boxed{s_0} \arrow[r, "op_{1a}(args_{1a})[cond_{1a}]"] \arrow[dr, "op_{1b}(args_{1b})[cond_{1b}]"'] & \boxed{s_{1a}} \arrow[r, "op_2(args_2)[cond_2]"] & \boxed{s_2} \arrow[r] & \ldots \\
 & \boxed{s_{1b}} \arrow[r] & \ldots &
\end{tikzcd}$$

- For simplicity, take conjunction of guards:

$$Cond(\pi) = cond_{1a} \wedge cond_2$$

# Information about a Path

Let $Cond(\pi)$ gather information about the path taken.


$$\begin{tikzcd}[column sep=huge] s_0 \arrow[r, "op_{1a}(args_{1a})[cond_{1a}]"] \arrow[dr, "op_{1b}(args_{1b})[cond_{1b}]"'] & s_{1a} \arrow[r, "op_2(args_2)[cond_2]"] & s_2 \arrow[r] & \ldots \\ & s_{1b} \arrow[r] & \ldots & \end{tikzcd}$$

- For simplicity, take conjunction of guards:

$$Cond(\pi) = cond_{1a} \wedge cond_2$$

➠ Restriction:

- Guards in state chart do not refer to attribute values
- Distinct names for event arguments

## Condition to Satisfy

- *If* we are on this path, the post cond. of the final method ensures that of the use case:

$$Cond(\pi) \rightarrow (Post_{final(\pi)} \rightarrow Post_{UC})$$

## Condition to Satisfy

- *If* we are on this path, the post cond. of the final method ensures that of the use case:

$$Cond(\pi) \rightarrow (Post_{final(\pi)} \rightarrow Post_{UC})$$

- or equivalently:

$$(Cond(\pi) \wedge Post_{final(\pi)}) \rightarrow Post_{UC}$$

## Condition to Satisfy

- *If* we are on this path, the post cond. of the final method ensures that of the use case:

$$Cond(\pi) \rightarrow (Post_{final(\pi)} \rightarrow Post_{UC})$$

- or equivalently:

$$(Cond(\pi) \wedge Post_{final(\pi)}) \rightarrow Post_{UC}$$

- Finally, this should be the case for all possible paths $\pi$:

$$\bigwedge_{\pi \in \Sigma} \Big( (Cond(\pi) \wedge Post_{final(\pi)}) \rightarrow Post_{UC} \Big)$$

## OCL Post-Condition for giveAmount

```
Context ATMController::giveAmount(amount:long) post:
  if ( amount <= bank.getBalance(card.getID()) ) then
        cashDispenser^giveOutCash(amount)
    and   bank.getBalance(card.getID())
        = bank.getBalance@pre(card.getID()) - amount
    and card^returnCard()
  else
        not cashDispenser^giveOutCash(?)
    and   bank.getBalance(card.getID())
        = bank.getBalance@pre(card.getID())
    and card^returnCard()
```

## Case Study: Normal Flow

- Consider: The “normal flow” with no wrong PIN entered and a sufficient balance
- Call this path $\pi_1$.
- Combined conditions from guards:

$$Cond(\pi_1) \quad = \quad (\texttt{userPin1 = cardPin} \wedge \texttt{balance >= amount})$$

- Final operation on $\pi_1$:

$$final(\pi_1) = \texttt{giveAmount}$$

## Case Study: Relationship between specifications

Need to show:

$$(Cond(\pi_1) \land Post_{final(\pi_1)}) \rightarrow Post_{UC}$$

## Case Study: Relationship between specifications

Need to show:

```
    userPin1 = cardPin
and balance >= amount
and if ( amount <= bank.getBalance(card.getID()) ) then
          cashDispenser^giveOutCash(amount)
      and   bank.getBalance(card.getID())
          = bank.getBalance@pre(card.getID()) - amount
      and card^returnCard()
    else
          not cashDispenser^giveOutCash(?)
      and   bank.getBalance(card.getID())
          = bank.getBalance@pre(card.getID())
      and card^returnCard()
```

implies $Post_{UC}$.

## Case Study: Relationship between specifications

Need to show:

```
    userPin1 = cardPin
and balance >= amount
and cashDispenser^giveOutCash(amount)
and   bank.getBalance(card.getID())
    = bank.getBalance@pre(card.getID()) - amount
and card^returnCard()
```

implies $Post_{UC}$.

## Informal Proof 1

```
    userPin1 = cardPin
and balance >= amount
and cashDispenser^giveOutCash(amount)
and   bank.getBalance(card.getID())
    = bank.getBalance@pre(card.getID()) - amount
and card^returnCard()
```

$$\Downarrow$$

If the customer entered the PIN on the Card, and the customer’s balance was greater or equal to the requested amount, then the customer got the requested amount and the amount was deducted from the balance.

## Informal Proof 2

```
    userPin1 = cardPin
and balance >= amount
and cashDispenser^giveOutCash(amount)
and   bank.getBalance(card.getID())
    = bank.getBalance@pre(card.getID()) - amount
and card^returnCard()
```

$$\Downarrow$$

If the customer entered the wrong PIN three times, the card was retained.

## Informal Proof 3

```
    userPin1 = cardPin
and balance >= amount
and cashDispenser^giveOutCash(amount)
and   bank.getBalance(card.getID())
    = bank.getBalance@pre(card.getID()) - amount
and card^returnCard()
```

$$\Downarrow$$

If the customer requested too much money, the card was returned to the customer.

## Reflection

We have considered whether

- every behavior allowed by the formal specification

  is also allowed according to the informal one

We have not yet considered whether

- every behavior the informal specification allows

  is still allowed by the formal one, stated as:

$$\bigwedge_{\pi \in \Sigma} \Big( (Cond(\pi) \land Post_{UC}) \longrightarrow Post_{final(\pi)} \Big)$$

## Too Strong Statement

- In general it is not the case that:

$$\bigwedge_{\pi\in\Sigma} \Big( (Cond(\pi) \wedge Post_{UC}) \rightarrow Post_{final(\pi)} \Big)$$

- Information is added when the informal specification is formalized
- Changing $Post_{UC}$ might make it unnecessarily verbose
- Might be important to keep track of the extra information,

  so we state instead:

$$\bigwedge_{\pi\in\Sigma} \Big( (Cond(\pi) \wedge Post_{UC} \wedge Extra(\pi)) \rightarrow Post_{final(\pi)} \Big)$$

## Case Study: Extra Information

Extra information in the post condition of `giveAmount`:

```
if ( amount <= bank.getBalance(card.getID()) ) then
      cashDispenser^giveOutCash(amount)
  and   bank.getBalance(card.getID())
      = bank.getBalance@pre(card.getID()) - amount
  and card^returnCard()
else
      not cashDispenser^giveOutCash(?)
  and   bank.getBalance(card.getID())
      = bank.getBalance@pre(card.getID())
  and card^returnCard()
```

➠ In this case, about half of the specification!

## Future Work

- Handle state charts with (infinitely) many paths efficiently

  ➠ handle groups of similar paths together

- Remove restrictions on state chart guards
- Tool support within the KeY system
- Automatically translate OCL constraints back to natural language

  ➠ Round-trip engineering

- More clearly separate the steps

Informal $\Leftrightarrow$ Formal

Analysis $\Leftrightarrow$ Design

## Grouping Similar Paths

The post condition $Post_{UC}$ is often composed of a number of clauses:

$$Post_{UC} = (C_1 \rightarrow P_1) \land (C_2 \rightarrow P_2) \land (C_3 \rightarrow P_3)$$

Sufficient to show

$$\bigwedge_{\pi \in \Sigma} \Big( (Cond(\pi) \land Post_{final(\pi)} \land C_i) \rightarrow P_i \Big)$$

for $i = 1, 2, 3$.

➠ many paths are immediately excluded by $C_i$

## Case Study: Simplification

Consider:

- $C_3 =$ "If the customer requested too much money"

Here we can conclude that

- only fulfilled on the three paths that go through the state 'insufficientBalance' in the state chart
- $\mathit{final}(\pi) = \texttt{giveAmount}$, on all these paths
- condition `balance < amount` is on all three paths

In this case it is sufficient to show:

- $(\texttt{balance < amount} \wedge \mathit{Post}_{\texttt{giveAmount}} \wedge C_3) \rightarrow P_3$

## End of Talk

Questions?