# *Softwaretechnik / Software-Engineering*

# *Lecture 7: Decision Tables*

*2019-05-20*

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

# Topic Area Requirements Engineering: Content

VL 5

- **Introduction**
- Definition: **Software** & **SW Specification**
- **Requirements Specification**
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques

VL 6

- **Documents**
  - Dictionary, Specification

VL 7

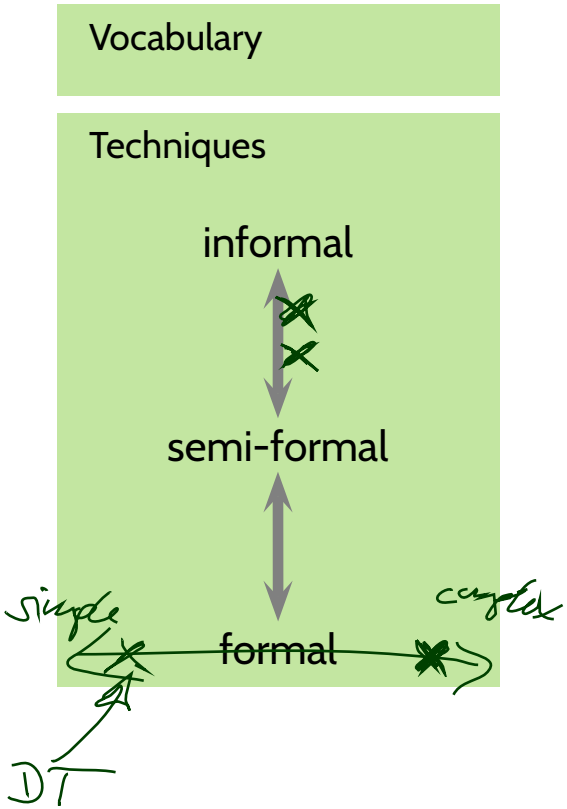
- **Specification Languages**
  - Natural Language
  - Decision Tables
    - Syntax, Semantics
    - Completeness, Consistency, …

VL 8

  - Scenarios
    - User Stories, Use Cases
    - Live Sequence Charts
      - Syntax, Semantics

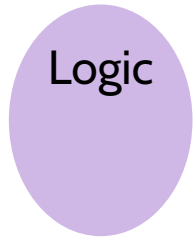
VL 9

- **Wrap-Up**



# Content

- **Decision Tables for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
    - **Detour**: Apropos (Non-)Determinism
- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
    - **Detour**: Apropos Assumptions
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
  - Consistency
- **Discussion**



# *Recall: Decision Tables*

# *Decision Table Syntax*

- Let $C$ be a set of **conditions** and $A$ be a set of **actions** s.t. $C \cap A = \emptyset$.
- A **decision table** $T$ **over** $C$ **and** $A$ is a labelled $(m + k) \times n$ matrix

| $T$: decision table | | $r_1$ | $\cdots$ | $r_n$ |
|---|---|---|---|---|
| $c_1$ | description of condition $c_1$ | $v_{1,1}$ | $\cdots$ | $v_{1,n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $c_m$ | description of condition $c_m$ | $v_{m,1}$ | $\cdots$ | $v_{m,n}$ |
| $a_1$ | description of action $a_1$ | $w_{1,1}$ | $\cdots$ | $w_{1,n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $a_k$ | description of action $a_k$ | $w_{k,1}$ | $\cdots$ | $w_{k,n}$ |

- where
  - $c_1, \ldots, c_m \in C$,
  - $a_1, \ldots, a_k \in A$,
  - $v_{1,1}, \ldots, v_{m,n} \in \{-, \times, *\}$ and
  - $w_{1,1}, \ldots, w_{k,n} \in \{-, \times\}$.

- Columns $(v_{1,i}, \ldots, v_{m,i}, w_{1,i}, \ldots, w_{k,i})$, $1 \leq i \leq n$, are called **rules**,
- $r_1, \ldots, r_n$ are **rule names**.
- $(v_{1,i}, \ldots, v_{m,i})$ is called **premise** of rule $r_i$,
  $(w_{1,i}, \ldots, w_{k,i})$ is called **effect** of $r_i$.

# *Decision Table Semantics*

Each rule $r \in \{r_1, \ldots, r_n\}$ of table $T$

| $T$: decision table | | $r_1$ | $\cdots$ | $r_n$ |
|---|---|---|---|---|
| $c_1$ | description of condition $c_1$ | $v_{1,1}$ | $\cdots$ | $v_{1,n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $c_m$ | description of condition $c_m$ | $v_{m,1}$ | $\cdots$ | $v_{m,n}$ |
| $a_1$ | description of action $a_1$ | $w_{1,1}$ | $\cdots$ | $w_{1,n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $a_k$ | description of action $a_k$ | $w_{k,1}$ | $\cdots$ | $w_{k,n}$ |

is assigned to a **propositional logical formula** $\mathcal{F}(r)$ over signature $C \dot\cup A$ as follows:

- Let $(v_1, \ldots, v_m)$ and $(w_1, \ldots, w_k)$ be premise and effect of $r$.
- Then

$$\mathcal{F}(r) := \underbrace{F(v_1, c_1) \wedge \cdots \wedge F(v_m, c_m)}_{=:\mathcal{F}_{pre}(r)} \wedge \underbrace{F(w_1, a_1) \wedge \cdots \wedge F(w_k, a_k)}_{=:\mathcal{F}_{eff}(r)}$$

where

$$F(v, x) = \begin{cases} x & \text{, if } v = \times \\ \neg x & \text{, if } v = - \\ \textit{true} & \text{, if } v = * \end{cases}$$

# Decision Table Semantics: Example

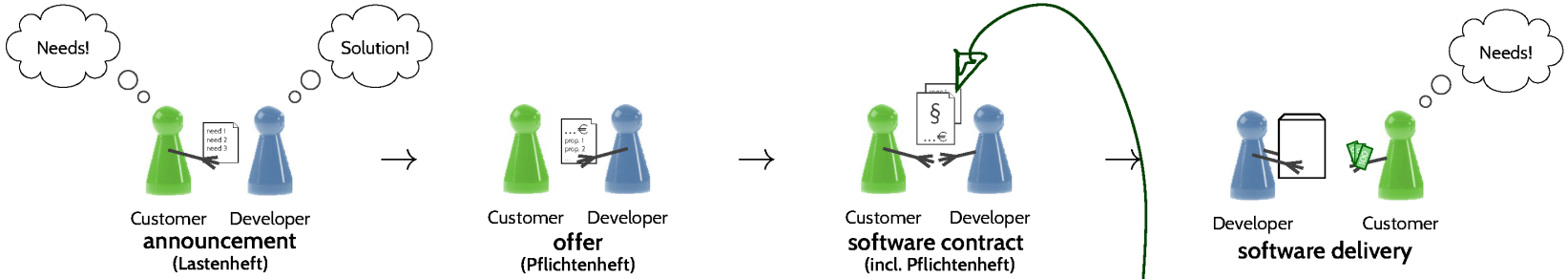
$$\mathcal{F}(r) := F(v_1, c_1) \wedge \cdots \wedge F(v_m, c_m) \wedge F(v_1, a_1) \wedge \cdots \wedge F(v_k, a_k)$$

$$F(v, x) = \begin{cases} x & \text{, if } v = \times \\ \neg x & \text{, if } v = - \\ \mathit{true} & \text{, if } v = * \end{cases}$$

| $T$ | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $c_1$ | | $\times$ | $\times$ | $-$ |
| $c_2$ | | $\times$ | $-$ | $*$ |
| $c_3$ | | $-$ | $\times$ | $*$ |
| $a_1$ | | $\times$ | $-$ | $-$ |
| $a_2$ | | $-$ | $\times$ | $-$ |

- $\mathcal{F}(r_1) = F(\times, c_1) \wedge F(\times, c_2) \wedge F(-, c_3) \wedge F(\times, a_1) \wedge F(-, a_2)$
  $= c_1 \wedge c_2 \wedge \neg c_3 \wedge a_1 \wedge \neg a_2$
- $\mathcal{F}(r_2) = c_1 \wedge \neg c_2 \wedge c_3 \wedge \neg a_1 \wedge a_2$
- $\mathcal{F}(r_3) = \neg c_1 \wedge \mathit{true} \wedge \mathit{true} \wedge \neg a_1 \wedge \neg a_2$

# Decision Tables as Specification Language




- Decision Tables can be used to **objectively** describe desired software behaviour.

- **Example**: Dear developer, please provide a program such that
  - in each situation (button pressed, ventilation on/off),
  - whatever the software does (action start/stop)
  - is **allowed** by decision table $T$.

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | × | × | – |
| $off$ | ventilation off? | × | – | * |
| $on$ | ventilation on? | – | × | * |
| $go$ | start ventilation | × | – | – |
| $stop$ | stop ventilation | – | × | – |

# *Decision Tables for Requirements Analysis*

# Once Again...

## Requirements on Requirements Specifications

A **requirements specification** should be

- **correct**
  – it correctly represents the wishes/needs of the customer,
- **complete**
  – all requirements (existing in somebody's head, or a document, or ...) should be present,
- **relevant**
  – things which are not relevant to the project should not be constrained,
- **consistent**, **free of contradictions**
  – each requirement is compatible with all other requirements; otherwise the requirements are **not realisable**,
- **neutral**, **abstract**
  – a requirements specification does not constrain the realisation more than necessary,
- **traceable**, **comprehensible**
  – the sources of requirements are documented, requirements are uniquely identifiable,
- **testable**, **objective**
  – the final product can **objectively** be checked for satisfying a requirement.

- **Correctness** and **completeness** are defined **relative** to something which is usually only **in the customer's head**.
  → is is **difficult** to **be sure of correctness** and **completeness**.
- **"Dear customer, please tell me what is in your head!"** is in almost all cases **not a solution**!
  It's not unusual that even the customer does not precisely know...!
  For example, the customer may not be aware of contradictions due to technical limitations.

# *Completeness*

**Definition.** [*Completeness*] A decision table $T$ is called **complete** if and only if the disjunction of all rules' premises is a tautology, i.e. if

$$\models \bigvee_{r \in T} \mathcal{F}_{pre}(r).$$

# *Completeness: Example*

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ |

- Is $T$ **complete**?

NO! $b \mapsto$ true, $off \mapsto$ true, $on \mapsto$ true

# Completeness: Example

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ |

- Is $T$ **complete**?

  **No.** (Because there is no rule for, e.g., the case $\sigma(b) = \mathit{true}$, $\sigma(on) = \mathit{false}$, $\sigma(off) = \mathit{false}$).

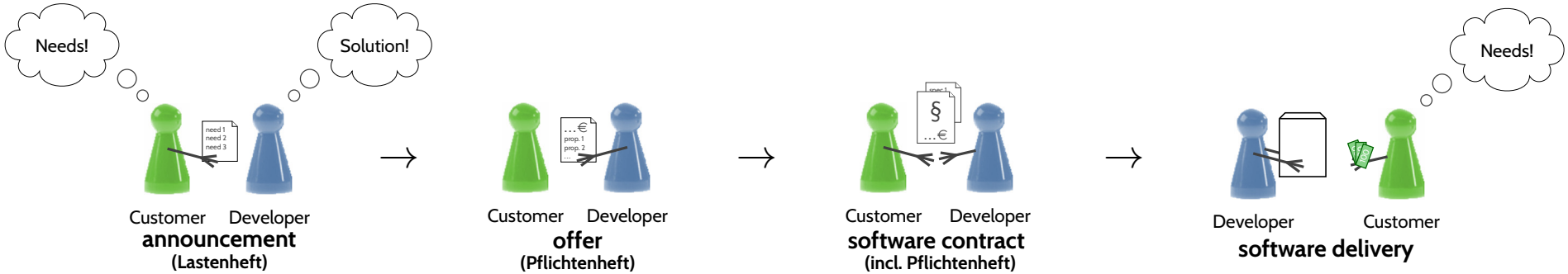
  **Recall**:

$$\mathcal{F}(r_1) = c_1 \wedge c_2 \wedge \neg c_3 \wedge a_1 \wedge \neg a_2$$
$$\mathcal{F}(r_2) = c_1 \wedge \neg c_2 \wedge c_3 \wedge \neg a_1 \wedge a_2$$
$$\mathcal{F}(r_3) = \neg c_1 \wedge \mathit{true} \wedge \mathit{true} \wedge \neg a_1 \wedge \neg a_2$$

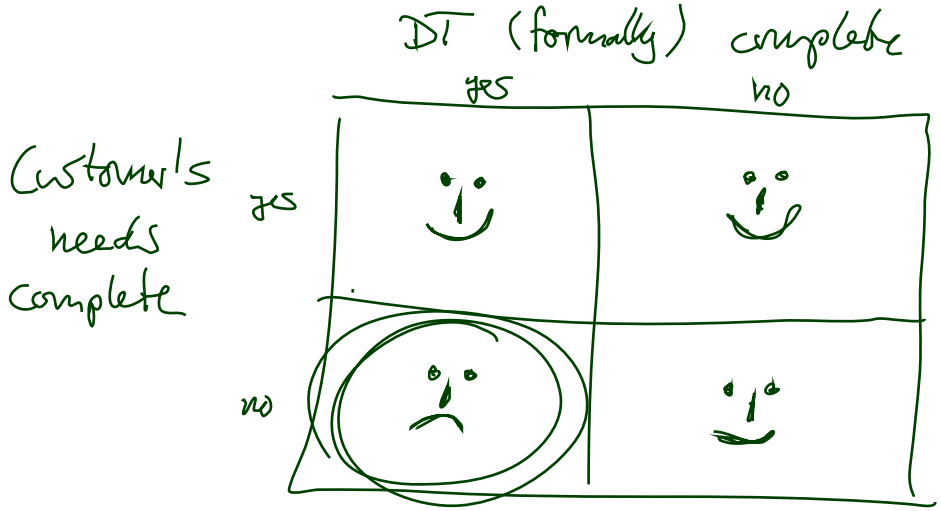
$$\mathcal{F}_{pre}(r_1) \vee \mathcal{F}_{pre}(r_2) \vee \mathcal{F}_{pre}(r_3)$$
$$= (c_1 \wedge c_2 \wedge \neg c_3) \vee (c_1 \wedge \neg c_2 \wedge c_3) \vee (\neg c_1 \wedge \mathit{true} \wedge \mathit{true})$$

  is **not a tautology**.

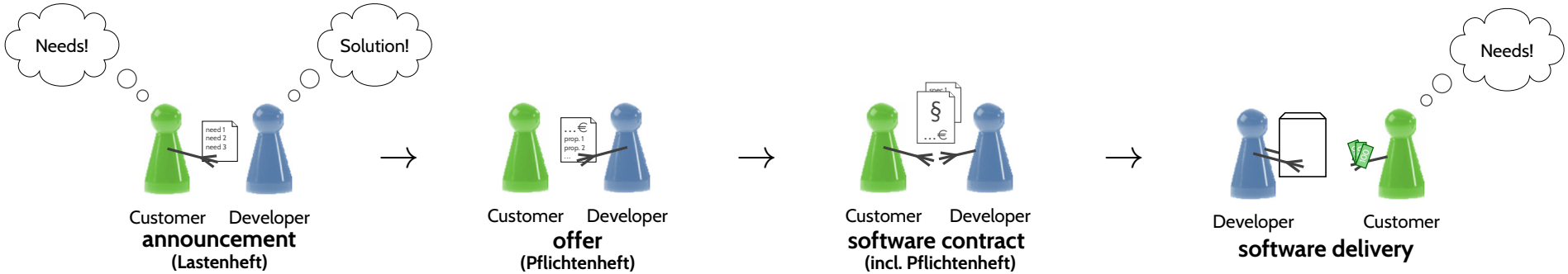
# *Requirements Analysis with Decision Tables*




- Assume we have formalised requirements as decision table $T$.



# *Requirements Analysis with Decision Tables*




- Assume we have formalised requirements as decision table $T$.
- **If $T$ is (formally) incomplete**,
  - then there is probably a case not yet discussed with the customer, or some misunderstandings.
- **If $T$ is (formally) complete**,
  - then there still may be misunderstandings. If there are no misunderstandings, then we did discuss all cases.

---

- **Note**:
  - Whether $T$ is (formally) complete is **decidable**.
  - Deciding whether $T$ is complete reduces to plain SAT.
  - There are efficient tools which decide SAT.
  - In addition, decision tables are often much easier to understand than natural language text.

# *For Convenience: The 'else' Rule*

- **Syntax**:

| $T$: decision table | | $r_1$ | $\cdots$ | $r_n$ | else |
|---|---|---|---|---|---|
| $c_1$ | description of condition $c_1$ | $v_{1,1}$ | $\cdots$ | $v_{1,n}$ | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | |
| $c_m$ | description of condition $c_m$ | $v_{m,1}$ | $\cdots$ | $v_{m,n}$ | |
| $a_1$ | description of action $a_1$ | $w_{1,1}$ | $\cdots$ | $w_{1,n}$ | $w_{1,e}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $a_k$ | description of action $a_k$ | $w_{k,1}$ | $\cdots$ | $w_{k,n}$ | $w_{k,e}$ |

- **Semantics**:

$$\mathcal{F}(\text{else}) := \neg \left( \bigvee_{r \in T \setminus \{\text{else}\}} \mathcal{F}_{pre}(r) \right) \wedge F(w_{1,e}, a_1) \wedge \cdots \wedge F(w_{k,e}, a_k)$$

**Proposition.** If decision table $T$ has an 'else'-rule, then $T$ is complete.

# *Uselessness*

**Definition.** [*Uselessness*] Let $T$ be a decision table.

A rule $r \in T$ is called **useless** (or: **redundant**)
if and only if there is another (different) rule $r' \in T$

- whose premise is implied by the one of $r$ and
- whose effect is the same as $r$'s,

i.e. if

$$\exists\, r' \neq r \in T \bullet \models (\mathcal{F}_{pre}(r) \implies \mathcal{F}_{pre}(r')) \wedge (\mathcal{F}_{\mathit{eff}}(r) \iff \mathcal{F}_{\mathit{eff}}(r')).$$

$r$ is called **subsumed** by $r'$.

- Again: uselessness is **decidable**; reduces to SAT.

# *Uselessness: Example*

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ | $r_4$ |
|---|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ | $-$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ | $\times$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ | $-$ |

- Rule $r_4$ is **subsumed** by $r_3$.
- Rule $r_3$ is **not** subsumed by $r_4$.

- Useless rules "do not hurt" as such.
- Yet useless rules should be removed to make the table more readable, yielding an **easier usable** specification.

## Requirements on Requirements Specification Documents

The **representation** and **form** of a requirements specification should be:

- **easily understandable, not unnecessarily complicated** –
  all affected people should be able to understand the requirements specification,
- **precise** –
  the requirements specification should not introduce new unclarities or rooms for interpretation ($\rightarrow$ testable, objective),
- **easily maintainable** –
  creating and maintaining the requirements specification should be easy and should not need unnecessary effort,
- **easily usable** –
  storage of and access to the requirements specification should not need significant effort.

**Note**: Once again, it's about compromises.

- A very precise **objective** requirements specification may not be easily understandable by every affected person.
  $\rightarrow$ provide redundant explanations.
- It is not trivial to have both, low maintenance effort and low access effort.
  $\rightarrow$ **value low access effort higher**,
  a requirements specification document is much more often **read** than **changed** or **written**
  (and most changes require reading beforehand).



- Rule $r_4$ is subsumed by $r_3$.
- Rule $r_3$ is not subsumed by $r_4$.

- Useless rules "do not hurt" as such.
- Yet useless rules should be removed to make the table more readable, yielding an **easier usable** specification.

# *Determinism*

**Definition.** [*Determinism*]
A decision table $T$ is called **deterministic**
if and only if the premises of all rules are pairwise disjoint, i.e. if

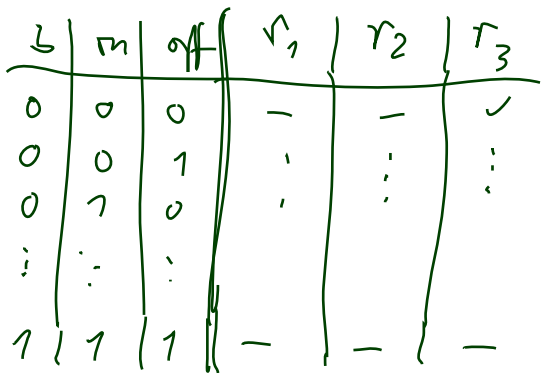
$$\forall\, r_1 \neq r_2 \in T\bullet \models \neg(\mathcal{F}_{pre}(r_1) \wedge \mathcal{F}_{pre}(r_2)).$$

Otherwise, $T$ is called **non-deterministic**.

- And again: determinism is **decidable**; reduces to SAT.

# *Determinism: Example*

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ |

- Is $T$ **deterministic**? **Yes.**
  - $\overline{f_{pre}(r_1)} \wedge \overline{f_{pre}(r_2)}$
  - 

# *Determinism: Another Example*

| $T_{abstr}$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | × | × | – |
| $go$ | start ventilation | × | – | – |
| $stop$ | stop ventilation | – | × | – |

- Is $T_{abstr}$ **deterministic**? **No.** $b \mapsto$ true

By the way…

- Is non-determinism **a bad thing** in general?
  - **Just the opposite**: non-determinism is a very, very powerful **modelling tool**.
  - Read table $T_{abstr}$ as:
    - **the button** may switch the ventilation **on** **under certain conditions** (which I will specify later), and
    - **the button** may switch the ventilation **off** **under certain conditions** (which I will specify later).

    We in particular state that we do not (under any condition) want to see $on$ and $off$ executed together, and that we do not (under any condition) see $go$ or $stop$ without button pressed.
- On the other hand: non-determinism may not be intended by the customer.

# Content

- **Decision Tables for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
    - **Detour**: Apropos (Non-)Determinism

- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
    - **Detour**: Apropos Assumptions
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
  - Consistency
- **Discussion**

Logic

# *Domain Modelling for Decision Tables*

# *Domain Modelling*

**Example**:

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ |

- If $on$ and $off$ model opposite output values of **one and the same sensor** for "room ventilation on/off", then $\sigma \models on \wedge off$ and $\sigma \models \neg on \wedge \neg off$ **never happen** in reality for any observation $\sigma$.
- Decision table $T$ is incomplete for exactly these cases.
  ($T$ "does not know" that $on$ and $off$ can be opposites in the real-world).
- We should be able to "tell" $T$ that $on$ and $off$ are opposites (if they are).
  Then $T$ would be **relative complete** (relative to the domain knowledge that $on$/$off$ are opposites).

**Bottom-line**:

- Conditions and actions are **abstract entities** without inherent connection to the **real world**.
- When modelling **real-world aspects** by conditions and actions,
  we may also want to represent **relations between actions/conditions** in the real-world
  ($\rightarrow$ **domain model** (Bjørner, 2006)).

# *Conflict Axioms for Domain Modelling*

- A **conflict axiom** over conditions $C$ is a propositional formula $\varphi_{confl}$ over $C$.

  **Intuition:** a conflict axiom characterises all those cases,
  i.e. all those combinations of condition values which 'cannot happen'
  — **according to our understanding of the domain**.

- **Note**: the decision table semantics remains unchanged!

**Example**:

- Let $\varphi_{confl} = (on \wedge off) \vee (\neg on \wedge \neg off)$.

  "$on$ models an opposite of $off$, neither can both be satisfied nor both non-satisfied at a time"

- **Notation**:

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | × | × | – |
| $off$ | ventilation off? | × | – | * |
| $on$ | ventilation on? | – | × | * |
| $go$ | start ventilation | × | – | – |
| $stop$ | stop ventilation | – | × | – |
| $\neg[(on \wedge off) \vee (\neg on \wedge \neg off)]$ | | | | |

# *Relative Completeness*

**Definition.** [*Completeness wrt. Conflict Axiom*]
A decision table $T$ is called **complete wrt. conflict axiom** $\varphi_{confl}$ if and only if the disjunction of all rules' premises and the conflict axiom is a tautology, i.e. if

$$\models \varphi_{confl} \vee \bigvee_{r \in T} \mathcal{F}_{pre}(r).$$

- **Intuition**: a relative complete decision table explicitly cares for all cases which 'may happen'.
- **Note**: with $\varphi_{confl} = \mathit{false}$, we obtain the previous definitions as a special case.

  **Fits intuition**: $\varphi_{confl} = \mathit{false}$ means we don't exclude any states from consideration.

# *Example*

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ |
| $\neg[(on \wedge off) \vee (\neg on \wedge \neg off)]$ | | | | |

- $T$ is complete wrt. its conflict axiom.

- **Pitfall**: if $on$ and $off$ are outputs of **two different, independent sensors**, then $\sigma \models on \wedge off$ **is possible in reality** (e.g. due to sensor failures).

  Decision table $T$ does not tell us what to do in that case!

# *Pitfalls in Domain Modelling (Wikipedia, 2015)*

**"Airbus A320-200 overran runway at Warsaw Okecie Intl. Airport on 14 Sep. 1993."**

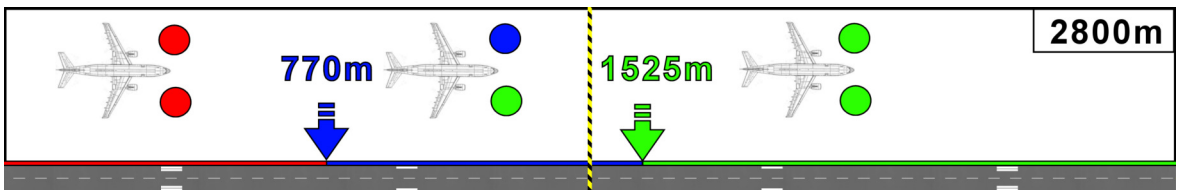
- To stop a plane after touchdown, there are **spoilers** and **thrust-reverse systems**.
- Enabling one of those while in the air, can have **fatal consequences**.
- **Design decision**: the **software should block** activation of spoilers or thrust-revers while in the air.
- Simplified decision table of **blocking** procedure:

| $T$ | | $r_1$ | $r_2$ | $r_3$ | else |
|---|---|---|---|---|---|
| $splq$ | spoilers requested | × | × | – | |
| $thrq$ | thrust-reverse requested | – | – | × | |
| $lgsw$ | at least 6.3 tons weight on each landing gear strut | × | * | × | |
| $spd$ | wheels turning faster than 133 km/h | * | × | * | |
| $spl$ | enable spoilers | × | × | – | – |
| $thr$ | enable thrust-reverse | – | – | × | – |

**Idea**: if conditions $lgsw$ and $spd$ **not satisfied**, then aircraft is in the air.

## 14 Sep. 1993:

- wind conditions not as announced from tower, tail- and crosswinds,
- anti-crosswind manoeuvre puts **too little weight** on landing gear
- wheels didn't turn fast due to **hydroplaning**.




"Flight 29041129" by Anynobody - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Flight_29041129.png#/media/File:Flight_29041129.png

"Lufthansa Flight 2904 crash site Siecinski" by Mariusz Siecinski - http://www.airliners.net/photo/Lufthansa/Airbus-A320-211/0265541/L/. Licensed under GFDL via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Lufthansa_Flight_2904_crash_site_Siecinski.jpg

# *Vacuity wrt. Conflict Axiom*

**Definition.** [*Vacuity wrt. Conflict Axiom*]
A rule $r \in T$ is called **vacuous wrt. conflict axiom** $\varphi_{confl}$ if and only if the premise of $r$ implies the conflict axiom, i.e. if $\models \mathcal{F}_{pre}(r) \rightarrow \varphi_{confl}$.

- **Intuition**: a vacuous rule would only be enabled in states which 'cannot happen'.

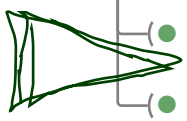
  **Example**:

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ | $r_4$ |
|---|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | $-$ | $\times$ |
| $off$ | ventilation off? | $\times$ | $-$ | $*$ | $\times$ |
| $on$ | ventilation on? | $-$ | $\times$ | $*$ | $\times$ |
| $go$ | start ventilation | $\times$ | $-$ | $-$ | $-$ |
| $stop$ | stop ventilation | $-$ | $\times$ | $-$ | $\times$ |
| $\neg[(on \wedge off) \vee (\neg on \wedge \neg off)]$ | | | | | |

- **Vacuity** wrt. $\varphi_{confl}$: Like uselessness, vacuity **doesn't hurt as such** but
  - **May hint on inconsistencies on customer's side.** (Misunderstandings with conflict axiom?)
  - **Makes using the table less easy!** (Due to more rules.)
  - **Implementing vacuous rules is a waste of effort!**

# Content

- **Decision Tables for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
    - **Detour**: Apropos (Non-)Determinism
- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
    - **Detour**: Apropos Assumptions
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
  - Consistency
- **Discussion**



Logic

# *Conflicting Actions*

# *Conflicting Actions*

**Definition.** [*Conflict Relation*] A **conflict relation** on actions $A$ is a **transitive** and **symmetric** relation $\lightning \subseteq (A \times A)$.

**Definition.** [*Consistency*] Let $r$ be a rule of decision table $T$ over $C$ and $A$.

(i) Rule $r$ is called **consistent with conflict relation** $\lightning$ if and only if there are no conflicting actions in its effect, i.e. if

$$\models \mathcal{F}_{\mathit{eff}}(r) \rightarrow \bigwedge_{(a_1,a_2)\in \lightning} \neg(a_1 \wedge a_2).$$

(ii) $T$ is called **consistent** with $\lightning$ iff all rules $r \in T$ are consistent with $\lightning$.

- Again: consistency is **decidable**; reduces to SAT.

# *Example: Conflicting Actions*

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|---|
| $b$ | button pressed? | × | × | – |
| $off$ | ventilation off? | × | – | * |
| $on$ | ventilation on? | – | × | * |
| $go$ | start ventilation | × | – | – |
| $stop$ | stop ventilation | × | × | – |
| $\neg[(on \wedge off) \vee (\neg on \wedge \neg off)]$ | | | | |

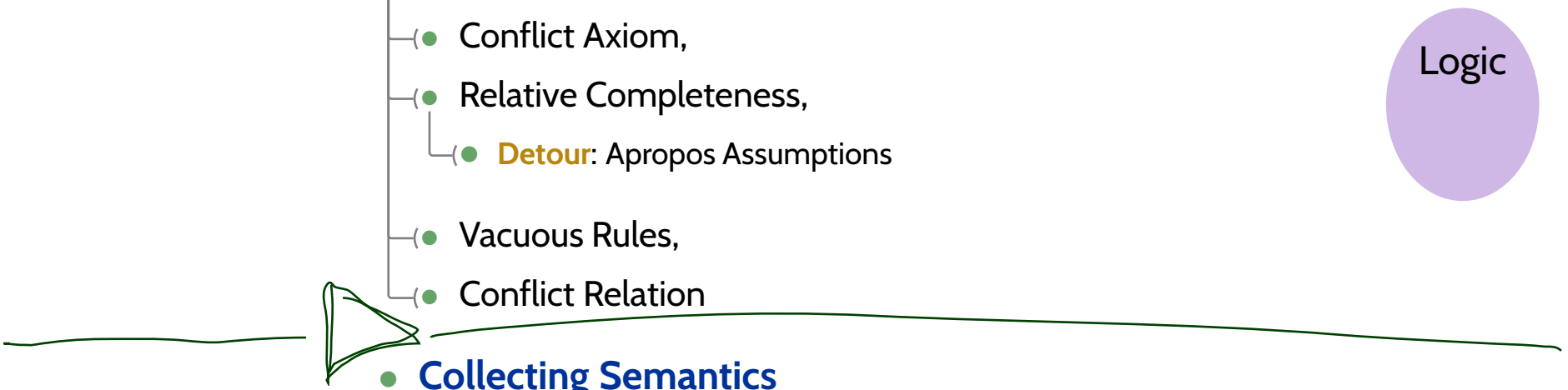
- Let $\lightning$ be the transitive, symmetric closure of $\{(stop, go)\}$.

  "actions $stop$ and $go$ are not supposed to be executed at the same time"
- Then rule $r_1$ is inconsistent with $\lightning$.

- A decision table with **inconsistent** rules **may do harm in operation!**
- **Detecting an inconsistency** only late during a project can incur significant cost!
- **Inconsistencies** — in particular in (multiple) decision tables, created and edited by multiple people, as well as in requirements in general — are **not always as obvious** as in the toy examples given here! (would be too easy...)
- And is even less obvious with the **collecting semantics** ($\rightarrow$ in a minute).

# Content

- **Decision Tables for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
    - **Detour**: Apropos (Non-)Determinism
- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
    - **Detour**: Apropos Assumptions
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
  - Consistency
- **Discussion**



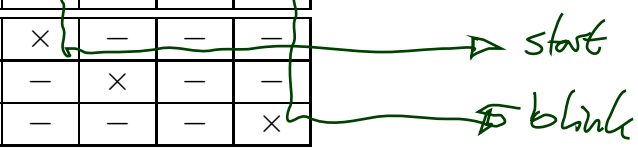
# *A Collecting Semantics for Decision Tables*

# *Collecting Semantics*

- Let $T$ be a decision table over $C$ and $A$
  and $\sigma$ be a model of an observation of $C$ and $A$.
  Then
  $$\mathcal{F}_{coll}(T) := \bigwedge_{a \in A} \Big( a \leftrightarrow \bigvee_{r \in T, r(a) = \times} \mathcal{F}_{pre}(r) \Big)$$
  is called **the collecting semantics** of $T$.
- We say, $\sigma$ is **allowed** by $T$ **in the collecting semantics** if and only if $\sigma \models \mathcal{F}_{coll}(T)$.
  That is, if exactly **all actions** of **all enabled** rules are planned/executed.

**Example**:

| $T$: room ventilation | | $r_1$ | $r_2$ | $r_3$ | $r_4$ |
|---|---|---|---|---|---|
| $b$ | button pressed? | $\times$ | $\times$ | – | $\times$ |
| $off$ | ventilation off? | $\times$ | – | $*$ | $*$ |
| $on$ | ventilation on? | – | $\times$ | $*$ | $*$ |
| $go$ | start ventilation | $\times$ | – | – | – |
| $stop$ | stop ventilation | – | $\times$ | – | – |
| $blnk$ | blink button | – | – | – | $\times$ |
| $\neg[(on \wedge off) \vee (\neg on \wedge \neg off)]$ | | | | | |




- "Whenever the button is pressed, let it blink (in addition to go/stop action."

# *Consistency in the Collecting Semantics*

**Definition.** [*Consistency in the Collecting Semantics*]
Decision table $T$ is called **consistent with conflict relation $\lightning$ in the collecting semantics** (under conflict axiom $\varphi_{confl}$) if and only if there are no conflicting actions in the effect of jointly enabled transitions, i.e. if

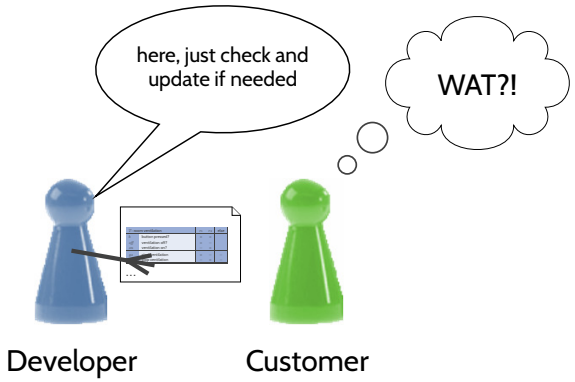
$$\models \mathcal{F}_{coll}(T) \wedge \neg\varphi_{confl} \rightarrow \bigwedge_{(a_1,a_2)\in\lightning} \neg(a_1 \wedge a_2).$$

# *Discussion*

# *Speaking of Formal Methods*

"Es ist aussichtslos, den Klienten mit formalen Darstellungen zu kommen; [...]"

("It is futile to approach clients with formal representations") (Ludewig and Lichter, 2013)



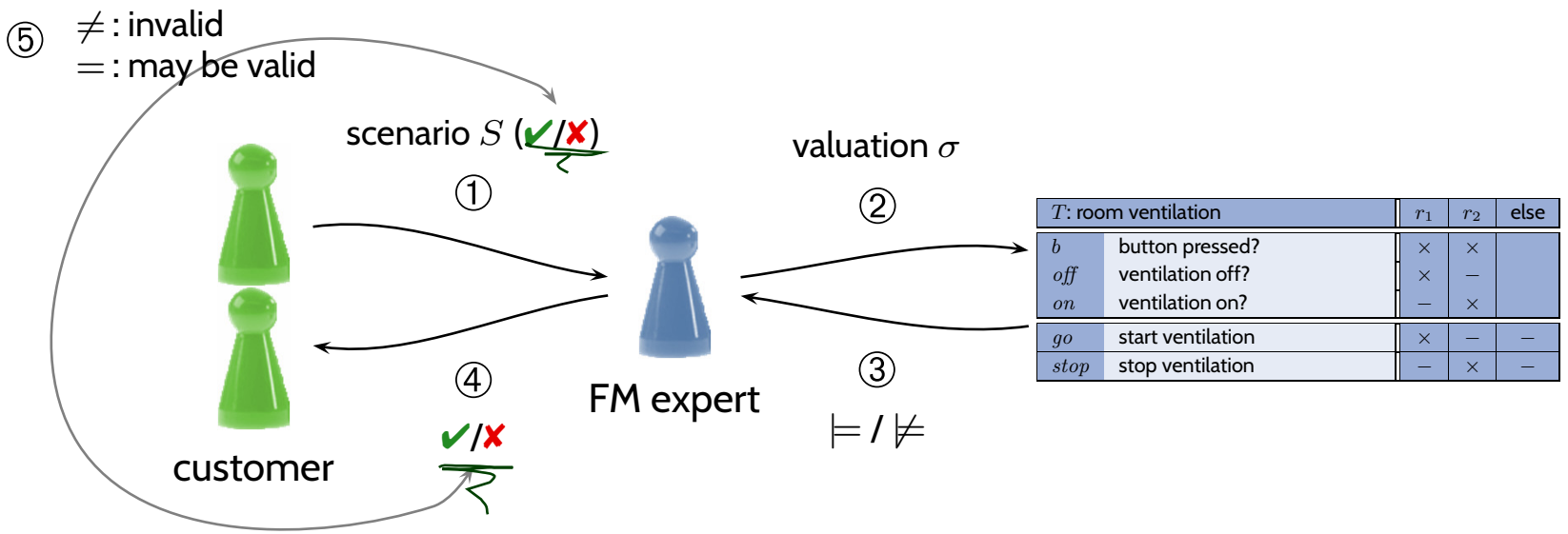

- ...**of course it is** — the vast majority of customers is not trained in formal methods.
- A formalisation is (first of all) for developers — **analysts have to translate** for customers.
- A **formalisation** is the description of **the analyst's understanding**, in a most precise form.

  **Precise/objective**: whoever reads it whenever to whomever, the meaning will not change.

# *Formalisation Validation*

**Two broad directions**:

- **Option 1**: teach formalism (usually not economic).
- **Option 2**: serve as translator / mediator.




| $T$: room ventilation | | $r_1$ | $r_2$ | else |
|---|---|---|---|---|
| $b$ | button pressed? | × | × | |
| $off$ | ventilation off? | × | – | |
| $on$ | ventilation on? | – | × | |
| $go$ | start ventilation | × | – | – |
| $stop$ | stop ventilation | – | × | – |

① domain experts **tell** system scenario $S$ (maybe keep back, whether allowed / forbidden),

② FM expert **translates** system scenario to valuation $\sigma$,

③ FM expert **evaluates** DT on $\sigma$,

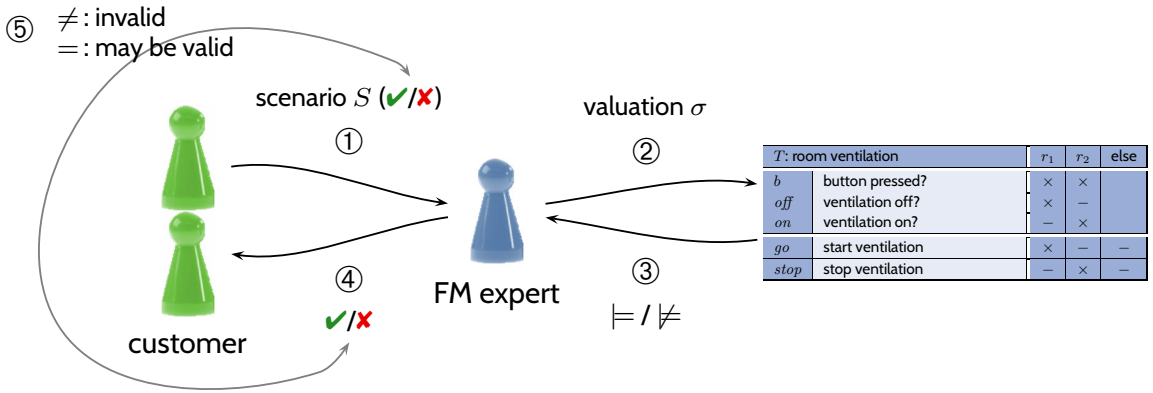
④ FM expert **translates** outcome to “allowed / forbidden by DT”,

⑤ compare expected outcome and real outcome.

# *Formalisation Validation*

**Two broad directions**:

- **Option 1**: teach formalism (usually not economic).
- **Option 2**: serve as translator / mediator.




① domain experts **tell** system scenario $S$ (maybe keep back, whether allowed / forbidden),

② FM expert **translates** system scenario to valuation $\sigma$,

③ FM expert **evaluates** DT on $\sigma$,

④ FM expert **translates** outcome to "allowed / forbidden by DT",

⑤ compare expected outcome and real outcome.

- **Recommendation**: (Course's Manifesto?)
  - use formal methods for the **most important/intricate requirements** (formalising **all requirements** is in most cases **not possible**),
  - use the **most appropriate formalism** for a given task,
  - use formalisms that **you know (really) well**.

# *References*

# *References*


Bjørner, D. (2006). *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Springer-Verlag.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Wikipedia (2015). Lufthansa flight 2904. id 646105486, Feb., 7th, 2015.