Softwaretechnik / Software-Engineering

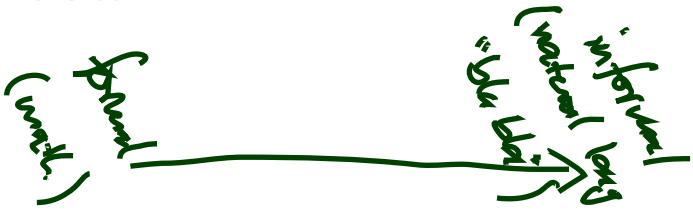
Lecture 07: Formal Methods for Requirements Engineering

2015-05-21

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

Albert-Ludwigs-Universität Freiburg, Germany

You Are Here



Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
Requirements Engineering	L 5:	11.5., Mo
	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
	-	28.5., Do
	T 3:	1.6., Mo
Design Modelling & Analysis	-	4.6., Do
	L 8:	8.6., Mo
	L 9:	11.6., Do
	L 10:	15.6., Mo
	T 4:	18.6., Do
Implementation, Testing	L 11:	22.6., Mo
	L 12:	25.6., Do
	L 13:	29.6., Mo
	T 5:	2.7., Do
Formal Verification	L 14:	6.7., Mo
	L 15:	9.7., Do
	L 16:	13.7., Mo
	T 6:	16.7., Do
The Rest	L 17:	20.7., Mo
	L 18:	23.7., Do

Contents & Goals

Last Lecture:

- requirements engineering basics, "the natural language case"

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a rule, a decision table?
  - What is the interleaving, collecting, update semantics?
  - Analyse this rule for: vacuous, redundant, complete, consistent
  - In what sense can decision tables serve as requirements specification language?
  - Formalise this requirement with a decision table.
  - What does this have to do with the previous lecture?
- **Content:**
  - definition decision table syntax and interleaving semantics,
  - definition tables as requirements specification,
  - interesting/useful properties of decision tables

Recall: Formal Methods

Formal Methods (in the Software Development Domain)

... back to "'technological paradise' where 'no acts of God can be permitted' and everything happens according to the blueprints".

(Kopetz, 2011; Lovins and Lovins, 2001)

Definition. [Bjørner and Havelund (2014)]
A method is called **formal method** if and only if its techniques and tools can be explained in mathematics.

Example: If a method includes, as a tool, a specification language, then that language has

- a **formal syntax**,
- a **formal semantics**, and
- a **formal proof system**.

– 1 – 2015-04-20 – main –

Formal, Rigorous, or Systematic Development

"The **techniques** of a formal method help

- **construct** a specification, and/or
- **analyse** a specification, and/or
- **transform** (**refine**) one (or more) specification(s) into a **program**.

The **techniques** of a formal method, (besides the specification languages) are typically software packages that help developers use the techniques and other tools.

The aim of developing software, either

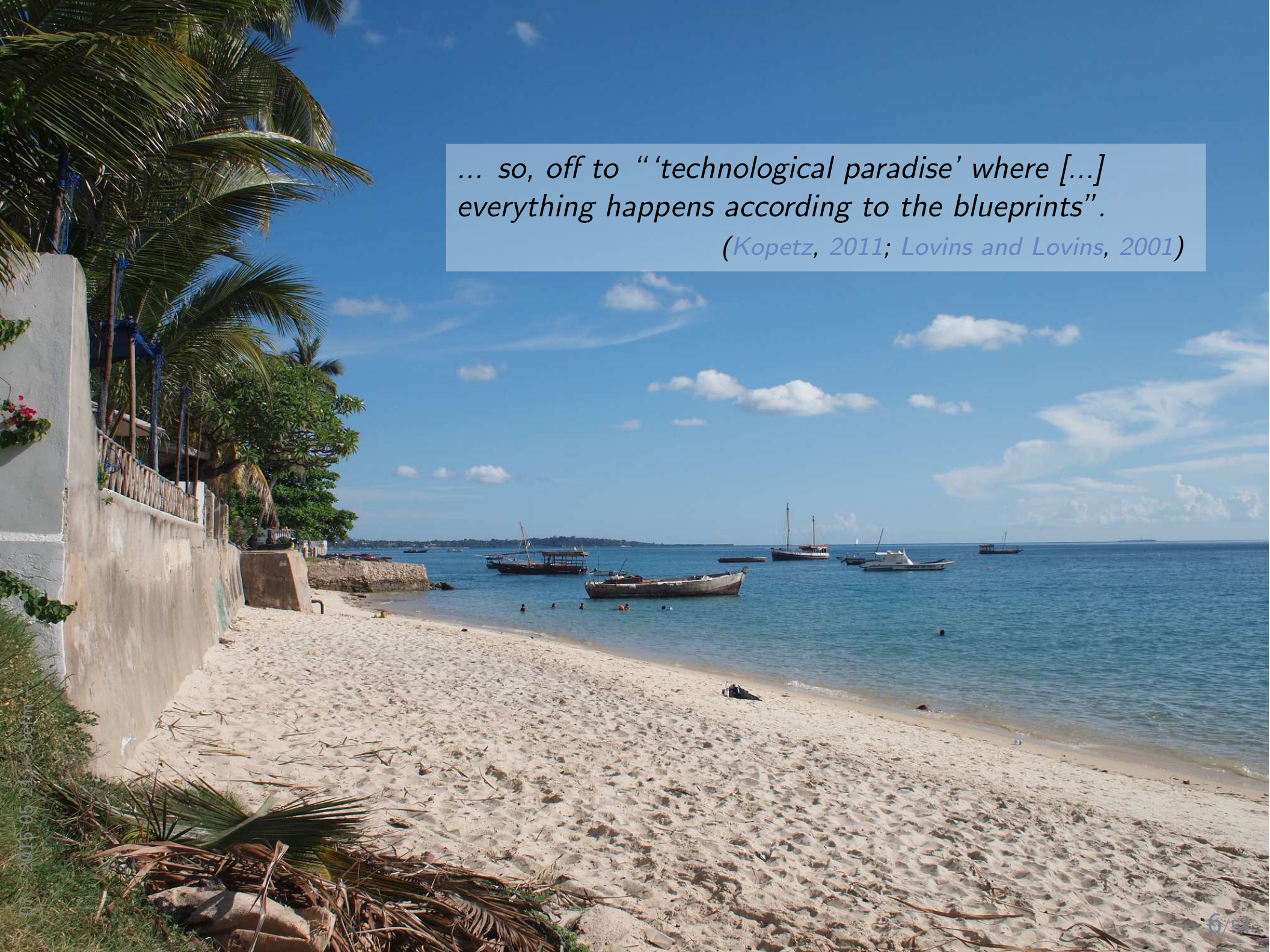
- **formally** (all arguments are formal) or
- **rigorously** (some arguments are made and they are formal) or
- **systematically** (some arguments are made on a form that can be made formal)

is to (be able to) **reason in a precise manner about properties** of what is being developed." (Bjørner and Havelund, 2014)

– 1 – 2015-04-20 – main –

Decision tables (DT) are **one example** for a formal requirements specification language:

- we give a formal syntax and semantics,
- **requirements quality criteria**, e.g. completeness, can be **formally defined**,
- **thus** for a DT we can **formally argue** whether it is complete or not,
- (some) formal arguments can be done **automatically** ($\rightarrow$ tool support).



... so, off to "'technological paradise' where [...] everything happens according to the blueprints".

(Kopetz, 2011; Lovins and Lovins, 2001)

Formal Specification and Analysis of Requirements: Decision Tables for Example

Decision Tables

Definition. [*Decision Table*] Let C be a set of (atomic) **conditions** and A a set of **actions**.

(i) The set $\Phi(C)$ of **premises** over C consists of the terms defined by the following grammar: $\varphi ::= \mathit{true} \mid c \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2$, $c \in C$.

(ii) A **rule** r (over C and A) is a pair (φ, α), written $\varphi \to \alpha$, which comprises

- a premise $\varphi \in \Phi(C)$ and
- a finite set $\alpha \subseteq A$ of actions (the **effect**).

(iii) Any finite set T of rules (over C and A) is called **decision table** (over C and A).

Decision Tables: Example

This might've been the specification of lecture hall 101-0-026's **ventilation system**:

- **Conditions**:

  $C = \{button\_pressed, ventilation\_on, ventilation\_off\}$ **shorthands**: $\{b, on, off\}$.

- **Actions**:

  $A = \{do\_ventilate, stop\_ventilate\}$ **shorthands**: $\{go, stop\}$.

- **Rules**:
  - $r_1 = \underbrace{b \wedge off}_{\varphi} \rightarrow \underbrace{\{go\}}_{\alpha}$
  - $r_2 = b \wedge on \rightarrow \{stop\}$

- **Decision table**:

  $T = \{r_1, r_2\}$.

And Where's The Table?

Decision tables can be written in tabular form:

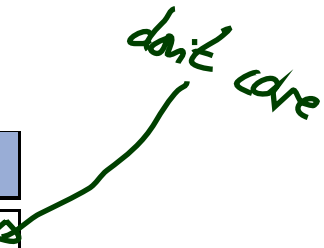
T: room ventilation		r_1	r_2	
b	button pressed?	×	×	φ
off	ventilation off?	×	-	
on	ventilation on?	-	×	
go	start ventilation	×	-	α
$stop$	stop ventilation	-	×	

From the table to the rules:

- $r_1 = b \wedge on \wedge \neg off \rightarrow \{go\}$
- $r_2 = b \wedge \neg on \wedge off \rightarrow \{stop\}$

And Where's The Table?

Decision tables can be written in tabular form:



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	-	$\times$	-

From the table to the rules:

- $r_1 = b \wedge on \wedge \neg off \rightarrow \{go\}$
- $r_2 = b \wedge \neg on \wedge off \rightarrow \{stop\}$
- $r_3 = \neg on \wedge \neg off \rightarrow \emptyset$

Decision Tables vs. Rules In General

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

$v_{i,j} \in \{-, \times, *\}$, $w_{i,j} \in \{-, \times\}$

- $C = \{c_1, \ldots, c_m\}$, $A = \{a_1, \ldots, a_k\}$
- $r_i = F(v_{1,i}, c_i) \wedge \cdots \wedge F(v_{m,i}, c_m) \rightarrow \{a_j \mid w_{j,i} = \times\}$, $F(v, c) = \begin{cases} c & \text{, if } v = \times \\ \neg c & \text{, if } v = - \\ \mathit{true} & \text{, if } v = * \end{cases}$

  Recall: $\bigwedge_{1 \leq j \leq m} F(v_{j,i}, c_i) = \mathit{true}$ by definition.
- $T = \{r_1, \ldots, r_n\}$ (multiple tables $T_1, \ldots, T_n$ denote **one set** of rules)
- From rules to table: use disjunctive normal form of φ.

By the Way: Decision Tables as Business Rules

$T1$: cash a cheque		r_1	r_2	else
c_1	credit limit exceeded?	×	×	
c_2	payment history ok?	×	-	
c_3	overdraft < 500 €?	-	$*$	
a_1	cash cheque	×	-	×
a_2	do not cash cheque	-	×	-
a_3	offer new conditions	×	-	-

(Balzert, 2009)

- One customer session at the bank:

$$\sigma \xrightarrow{\{a_1, a_3\}} \sigma'$$

if $\sigma = \{c_1 \mapsto 1, c_2 \mapsto 1, c_3 \mapsto 0\}$.

  - clerk checks database state σ,
  - database says: credit limit exceeded over 500 €, but payment history ok,
  - clerk cashes cheque but offers new conditions.

Decision Table Standard Semantics

- Let C be a set of conditions and A a set of actions.
- Let $\Sigma := (C \to \mathbb{B})$ be the set of **valuations** of C, $\mathbb{B} := \{0, 1\}$.
- Let $\varphi \in \Phi(C)$ be a premise and $\sigma \in \Sigma$. Set:
  - $\sigma \models \mathit{true}$, for all $\sigma \in \Sigma$,
  - $\sigma \models c$, if and only if $\sigma(c) = 1$,
  - $\sigma \models \neg\varphi_1$, if and only if $\sigma \not\models \varphi$,
  - $\sigma \models \varphi_1 \vee \varphi_2$, if and only if $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$.

  Note: In the following, we may use $\wedge$, $\Longrightarrow$, $\Longleftrightarrow$ as abbreviations as usual.
- We call a rule $r = \varphi \to \alpha$ over C and A **enabled** in σ if and only if $\sigma \models \varphi$.

- Let T be a decision table over C and A. The set $[\![T]\!]_{interleave}$, the (standard) **interleaving** semantics/interpretation of T, consists of the finite or infinite computation paths

$$\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots, \quad \sigma_i \in \Sigma, i \in \mathbb{N}_0,$$

where $\forall\, i \in \mathbb{N}_0\ \exists\, r = \varphi \to \alpha \in T \bullet \sigma_i \models \varphi \wedge \alpha_{i+1} = \alpha$.

Decision Tables: Example

Back to lecture hall 101-0-026's **ventilation system**:

- $C = \{button\_pressed, ventilation\_on, ventilation\_off\}$ **shorthands**: $\{b, on, off\}$.
- $A = \{do\_ventilate, stop\_ventilate\}$ **shorthands**: $\{go, stop\}$.
- $r_1 = b \wedge ventilation\_off \rightarrow \{go\}, \quad r_2 = b \wedge ventilation\_on \rightarrow \{stop\}, \quad T = \{r_1, r_2\}.$

What's in $[\![T]\!]_{interleaving}$? Naja, for **example**

by r_1 by r_2

- $\pi_1 = \sigma_0 \xrightarrow{\{go\}} \sigma_1 \xrightarrow{\{stop\}} \sigma_2$

  $\sigma_0 = \{b \mapsto 1, off \mapsto 1, on \mapsto 0\},$
  $\sigma_1 = \{b \mapsto 1, off \mapsto 0, on \mapsto 1\}$

- $\pi_2 = \sigma_0$

  $\sigma_0 = \{b \mapsto 0, off \mapsto 1, on \mapsto 0\}$

- and also $\pi_3 = \sigma_0 \xrightarrow{\{go\}} \sigma_1 \xrightarrow{\{go\}} \sigma_2$

  $\sigma_0 = \{b \mapsto 1, off \mapsto 1, on \mapsto 0\},$
  $\sigma_1 = \{b \mapsto 1, off \mapsto 1, on \mapsto 0\}$

- also $\pi_4 = \sigma_0 \xrightarrow{\{go\}} \sigma_1 \xrightarrow{\{go\}} \sigma_2 \ldots$

  $\sigma_i = \{b \mapsto 1, off \mapsto 1, on \mapsto 0\},\ i \in \mathbb{N}_0$

- but not $\sigma_0 \xrightarrow{\{go\}} \sigma_1 \xrightarrow{\{go,stop\}} \sigma_2$

  $\sigma_0 = \{b \mapsto 0, off \mapsto 1, on \mapsto 0\},$

Isn't There a Bell Ringing...?

Definition. **Software** is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form

$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$$

where

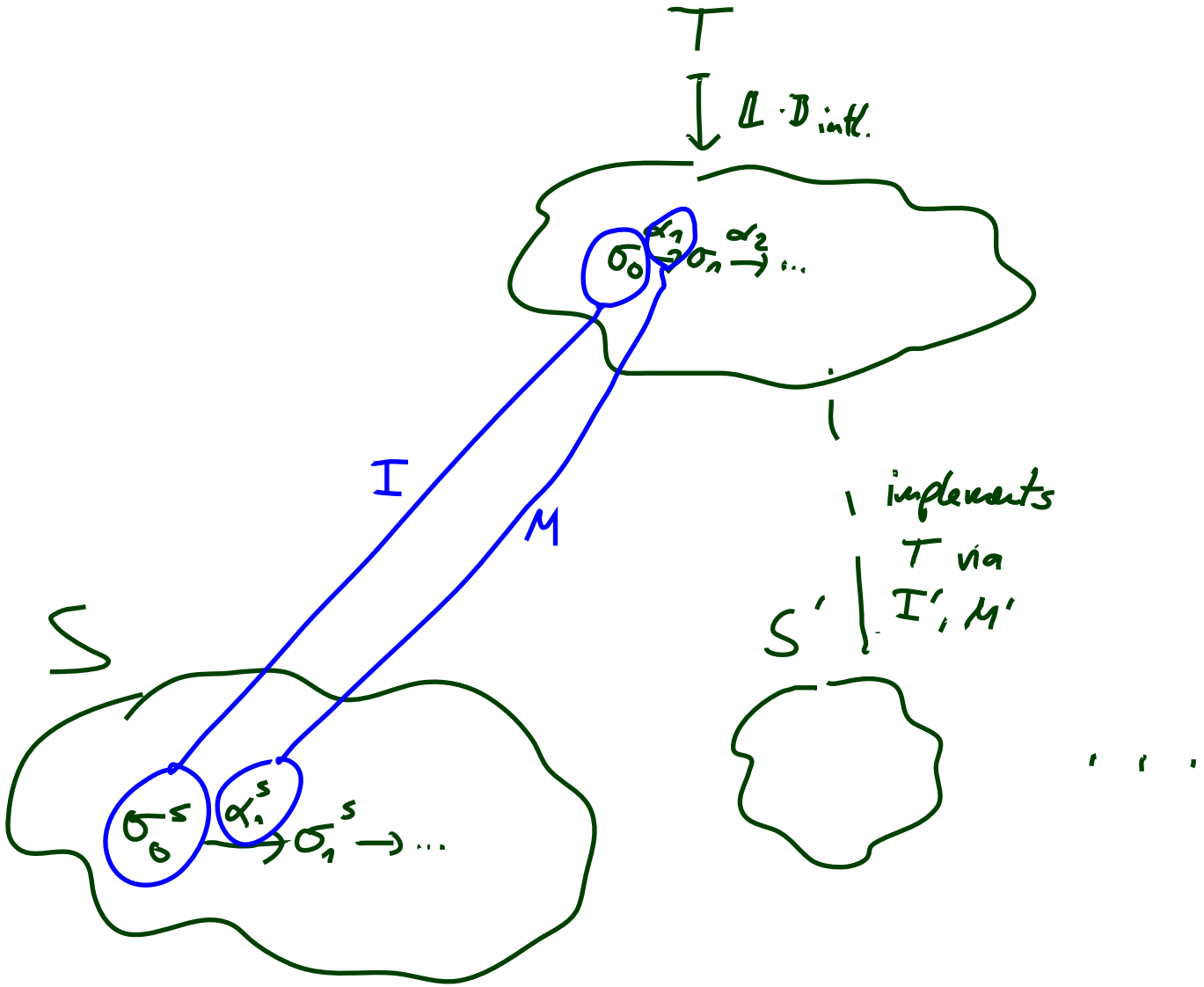
- $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called **state** (or **configuration**), and
- $\alpha_i \in \mathcal{A}$, $i \in \mathbb{N}_0$, is called **action** (or **event**).

The (possibly partial) function $[\![\cdot]\!] : S \mapsto [\![S]\!]$ is called **interpretation** of S.

$\rightarrow$ a decision table T is **software** in the sense of the definition.

(**surprise, surprise**!?)

*But We Want A Software **Specification**, Don't We...?*



*But We Want A Software **Specification**, Don't We...?*

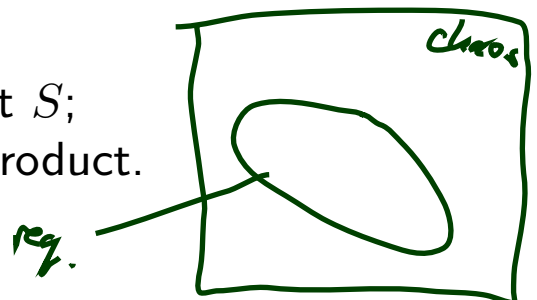
- Let T be a decision table over C and A.
- Let S be a software with $[\![S]\!] = \{\sigma_0^s \xrightarrow{\alpha_1^s} \sigma_1^s \xrightarrow{\alpha_2^s} \sigma_2^s \cdots, \ldots\}$, $\sigma_i^s \in \Sigma^s$.
- Let $I : \Sigma^s \to (C \to \mathbb{B})$ be an interpretation of conditions C in states Σ^s, and $M : \mathcal{A}^s \to 2^A$ a mapping of events to sets of actions.
- We say S **implements** T wrt. I and M if and only if

$$\forall\, \sigma_0^s \xrightarrow{\alpha_1^s} \sigma_1^s \xrightarrow{\alpha_2^s} \sigma_2^s \cdots \in [\![S]\!] \bullet I(\sigma_0^s) \xrightarrow{M(\alpha_1)} I(\sigma_1^s) \cdots \in [\![T]\!]_{interleaving}$$

- T can be seen as a **software specification** by setting

$$[\![T]\!]_{spec} := \{S \mid \exists\, I, M \bullet S \text{ implements } T \text{ wrt. } I \text{ and } M\}.$$

  - Any software S, **whose behaviour** viewed through I and M **is a subset** (!) of T's behaviour, satisfies the specification T.
  - The computation paths of T are **all allowed** for the final product S; what is not a computation path of T is **forbidden** for the final product.



- **The refinement view**:
  A software S' which **is a refinement** of a software $S \in [\![T]\!]_{spec}$ satisfies specification T.

Decision Table as Requirements Specification: Examples

How can these I and M look like?

Example:

- $\Sigma^1 = C \to \{0,1\}$ — a state $\sigma \in \Sigma_1$ is a (boolean) valuation of the conditions;

  $I : \sigma^1 \mapsto \sigma^1$, the identity

- $\Sigma^2 = \{b, V\} \to \mathbb{B} \cup \mathbb{R}_0^+$; $\sigma^2(b) \in \mathbb{B}$ (**button state**), $\sigma^2(V) \in \mathbb{R}_0^+$ (**voltage at ventilator**)

  I is defined by:

  - $I(\sigma^2)(b) = 1$ if and only if $\sigma(b) = 1$,
  - $I(\sigma^2)(on) = 1$ if and only if $\sigma(V) \geq 0.7$ (ventilator rotates),
  - $I(\sigma^2)(off) = 1$ if and only if $\sigma(V) < 0.7$ (voltage too low for rotation).

- Σ^3 : internal state of a Java VM running a Java program with global variables `b`, `on`, `off`.

  $I(\sigma^3)(b) = 1$ if and only if value of `b` in σ is Java's `true`.

- **In other words**: Σ^s can be **everything**, as long as you explain/define how to read out from $\sigma^s \in \Sigma^s$ whether the conditions (here: b, on, off) should be considered 0 or 1 in σ.

Basic Requirements Quality Criteria for DTs

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.

Requirements on Specifications and Formal Methods

- **correctness** is relative to "in the head of the customer" $\rightarrow$ still difficult;
- **complete**: we can at least define a kind of **relative completeness** in the sense of "did we cover all cases?";
- **relevant** also not analyseable **within** decision tables;
- **consistency** can formally be analysed!
- **neutral**/**abstract** is relative to the realisation $\rightarrow$ still difficult;
  But formal requirements specification language tend to support abstract specifications; specifying technical details is tedious.
- **traceable**/**comprehensible** are meta-properties, need to be established separately;
- a formal requirements specification, e.g. using decision tables, is immediately **objective**/**testable**.

We can formally define **additional quality criteria**:

- rules should not be **useless** or **vacuous**,
- a **deterministic** decision table may be desired,
- rules should be **consistent** wrt. a domain model,

Quality Criteria for DTs: Uselessness and Vacuity

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

- A rule $r = \varphi \to \alpha$ is called **vacuous** (wrt. states Σ) if and only if there is no state $\sigma \in \Sigma$ such that

$$\sigma \models \varphi.$$

- A rule $r = \varphi \to \alpha$ is called **useless** (or: **redundant**) if and only if there is another (different) rule r' whose premise φ' is subsumed by φ and whose effect is the same as r's, i.e. if

$$\exists\, r' = \varphi' \to \alpha', r' \neq r \bullet \models \varphi' \Longrightarrow \varphi \wedge \alpha = \alpha'.$$

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

Uselessness: Example

Example:

- $(c \wedge \neg c) \rightarrow \alpha$ is **vacuous**.

  Proposition: any rule with insatisfiable premise is vacuous.

-

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

  - r_4 is **useless** — it is subsumed by r_3.
  - r_3 is **not subsumed** by r_4!

- **Proposition**: if rule r is given in form of a table then r is not vacuous (yet it may be subsumed by another rule).

Uselessness: Consequences

- **Doesn't hurt** wrt. the final product:

  The decision table T with useless rules has the same computation paths as the one with useless rules removed, thus specifies the same set of software.

- **But**
  - decision tables with useless rules are unnecessarily hard to work with (read, maintain, ...).
  - **May make communication** (with customer) **harder!**

Quality Criteria for DTs: 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

$$\models \bigvee_{\varphi \to \alpha \in T} \varphi.$$

Completeness: Example

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

$\models (b \wedge off \wedge \neg on) \vee (b \wedge \neg off \wedge on) \; [\leftrightarrow true]$

NO:

$b=0, off=0, on=0$

Completeness: Example

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

- is **not complete**: there is no rule, e.g., for the case $\neg b \wedge on \wedge \neg off$.

T: room ventilation		r_1	r_2	r_3	r_4	r_5
b	button pressed?	$\times$	$\times$	-	$*$	$*$
off	ventilation off?	$\times$	-	$*$	$\times$	-
on	ventilation on?	-	$\times$	$*$	$\times$	-
go	start ventilation	$\times$	-	-	-	-
$stop$	stop ventilation	-	$\times$	-	-	-

- is complete.

Completeness: Example

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

- is **not complete**: there is no rule, e.g., for the case $\neg b \wedge on \wedge \neg off$.

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

- is complete.

Incompleteness: Consequences

- An incomplete decision table may allow too little behaviour (it forbids too much)!

- This very incomplete decision table:

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

- forbids all actions in case $b \wedge \neg on \wedge off$ is satisfied.
- **May not be the intention of the customer!**

Quality Criteria for DTs: 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

$$\forall\,(\varphi_i \to \alpha_i), (\varphi_j \to \alpha_j) \in T, i \neq j \bullet \models \neg(\varphi_i \wedge \varphi_j).$$

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

Determinism: Example

T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	-
go	start ventilation	×	-	-
$stop$	stop ventilation	-	×	-

- is **non-determistic**: In a state σ with $\sigma \models b$, rules r_1 and r_2 are both enabled.
- Is non-determinism **a bad thing** in general?
  - **Just the opposite**: one of the most powerful **modelling tools** we have.
  - Read table T 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).
  - This is quite some less chaos than full chaos!
  - We can already analyse the specification, e.g., we state that we do not (under any condition) want to see on and off executed together.

Non-determism: Consequences

- **Good**:
  - A decision table which is **intentionally** non-deterministic leaves **more choices** (more freedom) to the developer.

- **Bad**:
  - A non-deterministic decision table leaves **more choices** to the developer; even the choice to create a **non-deterministic final product**.

    (Input-)**Deterministic final products**, i.e. "same data in, same data out", are easier to deal with and **are usually desired**.
  - Postponing decisions too long may lead to hasty, bad decisions.

- **Another benefit**:
  deterministic decision tables can be **implemented** with deterministic programming languages.

  For deterministic decision tables, we can easily devise **code generation** patterns.

Implementing Decision Tables

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

```
1  int b, on, off;
2
3  extern void go();   extern void stop();
4
5  void (*effect)() = 0;
6
7  void dt() {
8
9    read_b_on_off(); // read
10
11   // compute
12   //
13   if (b && off) effect = go;
14
15   if (b && on) effect = stop;
16
17   execute_effect(); // write
18 }
```

read out of here (from b, on, off)

T

read out of

Domain Modelling for DTs

Domain Modelling

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

- Conditions and actions are **abstract entities** without inherent connection to the **real world**.
- Yet we **want to use** decision tables to model/represent requirements on the behaviour of software systems, which are **used** in the real world.
- When modelling **real-world aspects** by conditions and actions, we should also model **relations between actions/conditions** in the real-world
  ($\rightarrow$ **domain model** (Bjørner, 2006)).

Example:

- if on and off model opposite output values
  of **one and the same sensor** for "room ventilation on/off",
- then $\sigma \models on \wedge off$ **never happens** in reality,
- in the abstract setting, $\sigma \models on \wedge off$ is still possible.

  T "doesn't know" that on and off are opposites in the real-world; **maybe it should**.

- **Note**: 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).

"Poor Man's Domain Modelling"

- Add an action **cannot happen**.

T: room ventilation		r_1	r_2	$r_{\lightning}$	else
b	button pressed?	$\times$	$\times$	$*$	
off	ventilation off?	$\times$	-	$\times$	
on	ventilation on?	-	$\times$	$\times$	
go	start ventilation	$\times$	-		-
$stop$	stop ventilation	-	$\times$		-
ch	cannot happen			$\times$	

- **Pro**:
  - old definition of completeness applies
- **Con**:
  - all actions are equal, the action 'ch' is more equal
  - well-formedness property: no other actions are used with 'ch'
  - 'ch' is in the computation paths of T

Conflict Axioms for Domain Modelling

- A **conflict axiom** for conditions C is a formula $\varphi_{confl} \in \Phi(C)$.
- **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**.

Standard semantics wrt. conflict axiom:

- Let T be a decision table over C and A. The set $[\![T, \varphi_{confl}]\!]_{interleave}$, the (standard) interleaving semantics/interpretation of T **under conflict axiom** φ_{confl}, consists of the finite or infinite computation paths

$$\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots, \quad \sigma_i \in \{\sigma \in \Sigma \mid \sigma \not\models \varphi_{confl}\}, i \in \mathbb{N}_0,$$

  where $\forall\, i \in \mathbb{N}_0\ \exists\, r = \varphi \to \alpha \in T \bullet \sigma_i \models \varphi \wedge \alpha_{i+1} = \alpha$.

- **Note**: states satisfying φ_{confl} do not occur in π.

Vacuity, Completeness, Etc. With Conflict Axiom

Definition. [*Vacuitiy wrt. Conflict Axiom*] A rule $r = \varphi \to \alpha \in T$ over C and A is called **vacuous wrt. conflict axiom** $\varphi_{confl} \in \Phi(C)$ if and only if the premise of r implies the conflict axiom, i.e. if $\models (\varphi \implies \varphi_{confl})$.

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

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

$$\models \varphi_{confl} \vee \bigvee_{\varphi \to \alpha \in T} \varphi.$$

- **Intuition**: a complete decision table 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: Conflict Axioms

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

  "on models an opposite of off, neither can both be satisfied nor bot non-satisfied"

- Then

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	-	$\times$	-
$(on \wedge off) \vee (\neg on \wedge \neg off)$				

  is **complete** wrt. φ_{confl}.

- **Advantage**: no conditions 'hidden' in the else-rule.

Conflict Axiom: Consequences

- **Vacuity** wrt. φ_{confl}:

  Same as with uselessness and general vacuity, **doesn't hurt** but

  May make communication with customer harder!

  Implementing vacuous rules is a waste of effort!

- **Incompleteness**:

  An incomplete decision table may allow too little behaviour (it forbids too much)!
  May not be the intention of the customer!

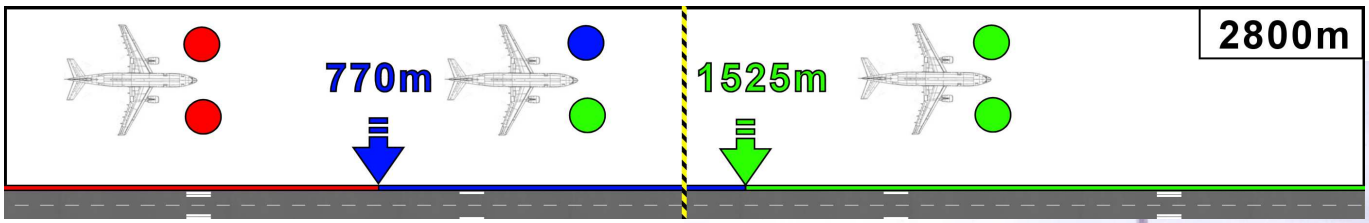
Domain Modelling: Consequences (Wikipedia, 2015)

"Airbus A320-200 overran runway at 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 in this case,
- **spoilers**: at least one of (i) and (ii) must be true; **thrust-reverse**: only if (i) true.
  - (i) at least 6.3 tons **weight on** each main **landing gear strut**,
  - (ii) the **wheels** of the plane must be **turning faster than 133 km/h**.
- **domain model**: "it cannot happen, that (i) and (ii) are not satisfied on ground".

14 Sep. 1993:

- tower announced crosswind,
- actually tailwind,
- 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

Cross-Checking DTs Against Conflicting Actions

Decision Tables Vs. Conflicting Conditions and Actions

- In certain domains, we may not want to execute certain actions together.
- Let's declare them to be conflicting and define **consistency** also wrt. conflicting actions.

- A **conflict relation** on actions A is a **transitive** and **symmetric** relation $\lightning \subseteq (A \times A)$.

Definition. [*Consistency*] Let $r = \varphi \to \alpha \in T$ be a rule.

(i) r is called **consistent with conflict relation** $\lightning$ if and only if there are no conflicting actions in α, i.e. if $\nexists\, a_1, a_2 \in \alpha \bullet a_1 \lightning a_2$.

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

Example: Conflicting Actions

- Let $\lightning$ be the transitive, symmetric closure of $stop \lightning go$.

  "actions $stop$ and go are not supposed to be executed at the same time"

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

- Rule r_1 is inconsistent with $\lightning$.

Conflicting Actions: Consequences

- **Consistency**:

  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 more difficult to handle with the **collecting semantics**.

Other Semantics for Decision Tables

A Collecting Semantics for Decision Tables

- Let T be a decision table and $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$ a state/event sequence.
- **Recall**: π is a **computation path** of T (in the interleaving semantics) if

$$\forall\, i \in \mathbb{N}_0\ \exists\, r = \varphi \to \alpha \in T \bullet \sigma_i \models \varphi \wedge \alpha_{i+1} = \alpha.$$

- That is, at each point in time **exactly one** rule fires, even if T is non-deterministic.

- π is a **computation path** of T (in the **collecting semantics**) if and only if

$$\forall\, i \in \mathbb{N}_0\ \exists\, r = \varphi \to \alpha \in T \bullet \sigma_i \models \varphi \wedge \alpha_{i+1} = \bigcup_{\substack{\varphi' \to \alpha' \in T,\\ \sigma_i \models \varphi'}} \alpha'.$$

That is, **all** rules which are **enabled** in σ_i "fire" simultaneously,
the joint effect is the union of the effects.

- **Advantage**:
  - separation of concerns, multiple (smaller) tables may contribute to a transition,
  - no non-determinism between rules: all enabled ones "fire".
- **Disadvantages**: conflicts much less obvious.

Consistency in The Collecting Semantics

Definition. [*Consistency in the Collecting Semantics*] Let T be a decision table.
T is called **consistent with conflict relation $\lightning$ in the collecting semantics** if and only if there are no conflicting actions in the effect of jointly enabled transitions, i.e. if

$$\nexists\, \varphi_1 \to \alpha_1, \varphi_2 \to \alpha_2, \sigma \in \Sigma \bullet$$
$$\sigma \models \varphi_1 \wedge \varphi_2 \wedge \exists\, a_1, a_2 \in \alpha_1 \cup \alpha_2 \bullet a_1 \lightning a_2.$$

An Update Semantics for Decision Tables

- By now, we didn't talk about the **effect of actions** from A on states.
  Actions are **uninterpreted**.
- **Recall** the **"cash cheque"** example:

  Here it makes sense, because the next state seen by the bank clerk may be the result of many database updates by other bank clerks, not only hers/his.

We can also define a semantics with action effects:

- Let C a set of conditions and A a set of actions, $\Sigma = C \to \mathbb{B}$ the standard states.
- Let $[\![\cdot]\!]_{acteff} : A \times \Sigma \to \Sigma$ which assigns to each pair of (a, σ) of action and state a new state σ'. σ' is called **the result** of applying a to σ.
- **Example**: on $\Sigma = \{b, on, off\}$, we could define

$$[\![go]\!]_{acteff}(\sigma) = \{b \mapsto \sigma(b), on \mapsto 1, off \mapsto 0\}$$

$$[\![stop]\!]_{acteff}(\sigma) = \{b \mapsto \sigma(b), on \mapsto 0, off \mapsto 1\}$$

- The interleaving semantics with action effects then requires

$$\forall\, i \in \mathbb{N}_0\ \exists\, r = \varphi \to \alpha \in T \bullet \sigma_i \models \varphi \wedge \alpha_{i+1} = \alpha \wedge \sigma_{i+1} = [\![\alpha]\!]_{acteff}(\sigma).$$

An Update Semantics for Decision Tables Cont'd

- In addition, we may want to constrain initial states, i.e. give a set $\Sigma_{ini} \subseteq \Sigma$.
- Computation paths of T (over Σ) are then required to have $\sigma_0 \in \Sigma_{ini}$.

- **Example**: decision table

T: room ventilation		r_1
off	ventilation off?	×
on	ventilation on?	-
go	start ventilation	×
stop	stop ventilation	-

with $\Sigma_{ini} = \{\{on \mapsto 0, off \mapsto 1\}\}$ has only one computation path, namely

$$\sigma_0 \xrightarrow{off} \sigma_1$$

with $\sigma_1 = \{on \mapsto 0, off \mapsto 1\}$.

- We can say T **terminates**.
- This gives rise to **another notion of vacuity**: $r = (on \wedge off) \rightarrow \alpha$ is never enabled, because no state satisfying the premise is ever reached (even with conflict axiom $\varphi_{confl} = \mathit{false}$).

Distinguishing Controlled and Uncontrolled Conditions

- For some systems, we can distinguish **inputs** and **outputs**.
- **In terms of decision tables**:
  - C is partitioned into **controlled conditions** C_c and **uncontrolled conditions** C_u, i.e. $C = C_c \dot\cup C_u$.
  - actions only affect controlled conditions.
- **Example**:
  - $C_c = \{on, off\}$ (only the software switches the ventilation on or off),
  - $C_u = \{b\}$ (the button is not controlled by the software, but by the environment, by a user external to the computer system)
- One more quality criterion, **another notion of completeness**:

  We want the specification to be able to deal with all possible sequences of inputs, i.e. we require

$$[\![T]\!]|_{C_u} = (\Sigma|_{C_u})^\omega \quad (\text{ or } (\Sigma|_{C_u})^*).$$

Controlled and Uncontrolled Conditions: Example

- **Example**:

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

  - is not **input sequence complete**:

    There is no rule enabled if the button is pressed when the ventilation is off.

  - **Note**: it's not that pressing the button such a state **has no effect**, but the system **stops to work**; it "gets stuck" in that state.

    (Because in order to take a transition, we need to have at least one enabled rule.)

Decision Tables: Discussion

Decision Tables Summary

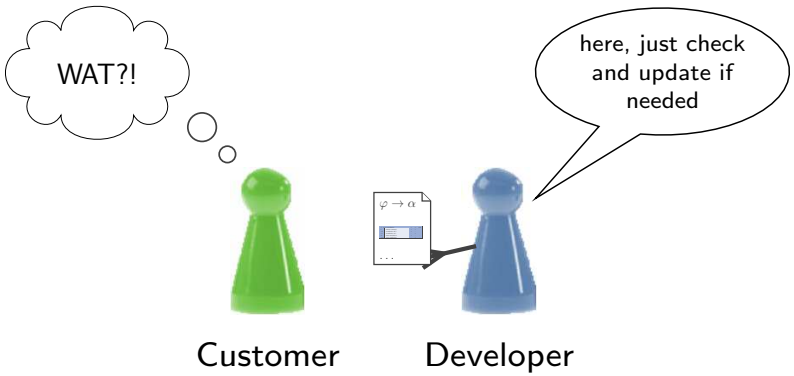
- **decision tables** are **one (very simple) example** for a **formal** requirements specification language with
  - **formal syntax** (there are even two: the formula and the table notation)
  - **formal semantics** (also multiple: interleaving, collecting, update)
  - **a formal proof system** — naja, not a dedicated one
- a requirements specification in form of **decision tables** allows us to **formally** reason about properties of what is being developed.

  We can, e.g., **prove** that a decision table is complete. (Automatically?)

- Whether a **decision table** is useful in a particular software development project (of course) **depends** on the project.
- Like many **formal specification language**,
  - a **decision table** may not be the right tool for all problems,
  - it may be tedious to specify **all** requirements using **decision tables** — don't do that then.
- One particular **drawback** of **decision tables**:
  they don't scale so well in the number of conditions.

Speaking of Formal Methods

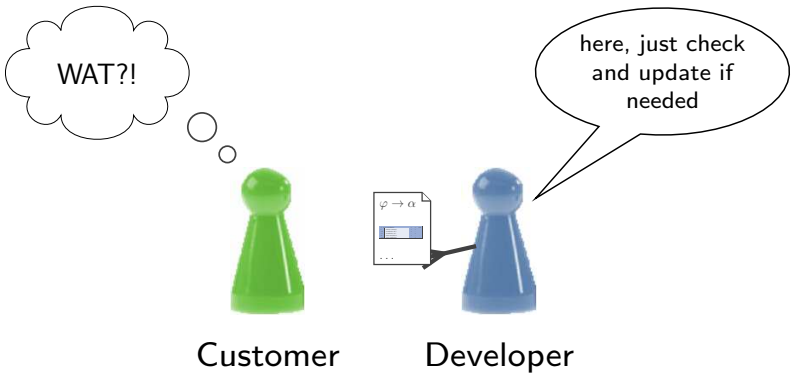
"Es ist aussichtslos, den Klienten mit formalen Darstellungen zu kommen; [...]"
("It is futile to approach clients with formal representations")
(Ludewig and Lichter, 2013)



- . . . **of course it is** — vast majority of customers is not trained in formal methods.

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** — vast majority of customers is not trained in formal methods.
- formalisation is (firstly) for developers — **analysts have to translate** for customers.
- **formalisation** is the description of **the analyst's understanding**, in a most precise form.
- **precision**: whoever reads it whenever to whomever, the meaning will not change.
- **Recommendation**: (Course's Manifesto?)
  - use formal methods for **the most important/intricate requirements**,
  - use formalisms that **you know (really) well**,
  - you may use **different formalisms** for different requirements (if you know what you're doing!)
  - trying to formalise **all requirements** is in most cases futile.

References

References

Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum, 3rd edition.

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

Kopetz, H. (2011). What I learned from Brian. In Jones, C. B. et al., editors, *Dependable and Historic Computing*, volume 6875 of *LNCS*. Springer.

Lovins, A. B. and Lovins, L. H. (2001). *Brittle Power - Energy Strategy for National Security*. Rocky Mountain Institute.

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

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