Introduction

Requirements Specification
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques

Documents
  - Dictionary, Specification

Specification Languages
  - Natural Language
  - Decision Tables
    - Syntax, Semantics
    - Completeness, Consistency, …
  - Scenarios
    - User Stories, Use Cases
    - Live Sequence Charts
      - Syntax, Semantics

Definition: Software & SW Specification

Wrap-Up
**Risks Implied by Bad Requirements Specifications**

- **design and implementation**, without specification, programmers may just “ask around” when in doubt, possibly yielding different interpretations → **difficult integration**
- **negotiation** (with customer, marketing department, or …)
- **documentation**, e.g., the user’s manual,
  - without specification, the user’s manual author can only describe what the system does, not what it should do ("every observation is a feature")
  - later re-implementations.
    - the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → **additional effort**
- preparation of **tests**, without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → **systematic testing hardly possible**
- **acceptance** by customer, resolving later objections or regress claims,
  - without specification, it is unclear at delivery time whether behaviour is an error (developer needs to fix) or correct (customer needs to accept and pay) → **nasty disputes, additional effort**
  - the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → **additional effort**
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.
**Example:** Requirements Engineering

- **Vocabulary**
  - e.g. consistent, complete, tacit, etc.

- **Techniques**
  - informal
  - semi-formal
  - formal

  - e.g. “Whenever a crash…”
  - e.g. “Always, if \(\text{crash}\) at \(t\)…”
  - e.g. “\(\forall t, t' \in \text{Time} \bullet\)…”

- **In the course:**
  - Use Cases
  - Pattern Language
  - Decision Tables
  - Live Sequence Charts
Content

- (Basic) Decision Tables
  - Syntax, Semantics
- …for Requirements Specification
- …for Requirements Analysis
  - Completeness,
  - Useless Rules,
  - Determinism
- Domain Modelling
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation
- Collecting Semantics
- Discussion
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

<table>
<thead>
<tr>
<th>$T$: decision table</th>
<th>$r_1$</th>
<th>$\cdots$</th>
<th>$r_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>description of condition $c_1$</td>
<td>$v_{1,1}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$c_m$</td>
<td>description of condition $c_m$</td>
<td>$v_{m,1}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>description of action $a_1$</td>
<td>$w_{1,1}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$a_k$</td>
<td>description of action $a_k$</td>
<td>$w_{k,1}$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>

- where
  - $c_1, \ldots, c_m \in C$,
  - $a_1, \ldots, a_k \in A$,
  - $v_{1,1}, \ldots, v_{m,n} \in \{-, \times, \ast\}$ 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 $r_i$. 
Each rule \( r \in \{r_1, \ldots, r_n\} \) of table \( T \)

is assigned to a propositional logical formula \( F(r) \) over signature \( C \cup A \) as follows:

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

\[
F(r) := F(v_1, c_1) \land \cdots \land F(v_m, c_m) \land F(w_1, a_1) \land \cdots \land F(w_k, a_k)
\]

where

\[
F(v, x) = \begin{cases} 
  x, & \text{if } v = \times \\
  \neg x, & \text{if } v = - \\
  \text{true}, & \text{if } v = * 
\end{cases}
\]
**Decision Table Semantics: Example**

\[ F(r) := F(v_1, c_1) \land \cdots \land F(v_m, c_m) \land F(v_1, a_1) \land \cdots \land F(v_k, a_k) \]

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

<table>
<thead>
<tr>
<th></th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>-</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( \times )</td>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>-</td>
<td>( \times )</td>
<td>*</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( \times )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>-</td>
<td>( \times )</td>
<td>-</td>
</tr>
</tbody>
</table>

- \( F(r_1) = F(\times, c_1) \land F(\times, c_2) \land \neg c_3 \land F(\times, a_1) \land \neg a_2 \)
- \( F(r_2) = F(\times, c_1) \land \neg c_2 \land c_3 \land \neg a_1 \land a_2 \)
- \( F(r_3) = \neg c_1 \land \text{true} \land \text{true} \land \neg a_1 \land \neg a_2 \)
Decision Tables as Requirements Specification
We can use decision tables to model (describe or prescribe) the behaviour of software!

Example:
Ventilation system of lecture hall 101-0-026.

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

- We can observe whether button is pressed and whether room ventilation is on or off, and whether (we intend to) start ventilation or stop ventilation.

- We can model our observation by a boolean valuation $\sigma : C \cup A \rightarrow \mathbb{B}$, e.g., set

  $\sigma(b) := true$, if button pressed now and $\sigma(b) := false$, if button not pressed now.

  $\sigma(go) := true$, we plan to start ventilation and $\sigma(go) := false$, we plan to stop ventilation.

- A valuation $\sigma : C \cup A \rightarrow \mathbb{B}$ can be used to assign a truth value to a propositional formula $\varphi$ over $C \cup A$. As usual, we write $\sigma \models \varphi$ iff $\varphi$ evaluates to true under $\sigma$ (and $\sigma \not\models \varphi$ otherwise).

- Rule formulae $\mathcal{F}(r)$ are propositional formulae over $C \cup A$ thus, given $\sigma$, we have either $\sigma \models \mathcal{F}(r)$ or $\sigma \not\models \mathcal{F}(r)$.

- Let $\sigma$ be a model of an observation of $C$ and $A$.

  We say, $\sigma$ is allowed by decision table $T$ if and only if there exists a rule $r$ in $T$ such that $\sigma \models \mathcal{F}(r)$. 
### Example

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

\[
F(r_1) = b \land \text{off} \land \neg \text{on} \land \text{go} \land \neg \text{stop}
\]
\[
F(r_2) = b \land \neg \text{off} \land \text{on} \land \neg \text{go} \land \text{stop}
\]
\[
F(r_3) = \neg b \land \text{true} \land \text{true} \land \neg \text{go} \land \neg \text{stop}
\]

(i) **Assume**: button pressed, ventilation off, we (only) plan to start the ventilation.
- Corresponding valuation: $\sigma_1 = \{b \mapsto \text{true}, \text{off} \mapsto \text{true}, \text{on} \mapsto \text{false}, \text{start} \mapsto \text{true}, \text{stop} \mapsto \text{false}\}$.
- Is our intention (to start the ventilation now) allowed by $T$? **Yes!** (Because $\sigma_1 \models F(r_1)$)

(ii) **Assume**: button pressed, ventilation on, we (only) plan to stop the ventilation.
- Corresponding valuation: $\sigma_2 = \{b \mapsto \text{true}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{true}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true}\}$.
- Is our intention (to stop the ventilation now) allowed by $T$? **Yes.** (Because $\sigma_2 \models F(r_2)$)

(iii) **Assume**: button not pressed, ventilation on, we (only) plan to stop the ventilation.
- Corresponding valuation: $\sigma_3 = \{b \mapsto \text{false}, \text{on} \mapsto \text{true}, \text{off} \mapsto \text{false}, \text{start} \mapsto \text{true}, \text{stop} \mapsto \text{false}\}$.
- Is our intention (to stop the ventilation now) allowed by $T$?
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 $T$.

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Decision Tables as Specification Language

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

- **Another Example**: Customer session at the bank:

  
<table>
<thead>
<tr>
<th>$T1$: cash a cheque</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$ credit limit exceeded?</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>$c_2$ payment history ok?</td>
<td>×</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>$c_3$ overdraft &lt; 500 €?</td>
<td>−</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>$a_1$ cash cheque</td>
<td>×</td>
<td>−</td>
<td>×</td>
</tr>
<tr>
<td>$a_2$ do not cash cheque</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>$a_3$ offer new conditions</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

  
  (Balsert, 2009)

- clerk checks database state (yields $\sigma$ for $c_1, \ldots, c_3$),
- database says: credit limit exceeded over 500 €, but payment history ok,
- clerk cashes cheque but offers new conditions (according to $T1$).
Decision Tables for Requirements Analysis
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

$$\vdash \bigvee_{r \in T} \mathcal{F}_{pre}(r).$$
Completeness: Example

<table>
<thead>
<tr>
<th>$r$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>stop</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

- Is $T$ complete?

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

**Recall:**

$$\mathcal{F}(r_1) = c_1 \land c_2 \land \neg c_3 \land a_1 \land \neg a_2$$

$$\mathcal{F}(r_2) = c_1 \land \neg c_2 \land c_3 \land \neg a_1 \land a_2$$

$$\mathcal{F}(r_3) = \neg c_1 \land true \land true \land \neg a_1 \land \neg a_2$$

$$\mathcal{F}_{pre}(r_1) \lor \mathcal{F}_{pre}(r_2) \lor \mathcal{F}_{pre}(r_3)$$

$$= (c_1 \land c_2 \land \neg c_3) \lor (c_1 \land \neg c_2 \land c_3) \lor (\neg c_1 \land true \land true)$$

is not a tautology.
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:

<table>
<thead>
<tr>
<th>T: decision table</th>
<th>r₁</th>
<th>⋯</th>
<th>rₙ</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁ description of condition c₁</td>
<td>v₁,₁</td>
<td>⋯</td>
<td>v₁,n</td>
<td></td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td></td>
</tr>
<tr>
<td>cₘ description of condition cₘ</td>
<td>vₘ,₁</td>
<td>⋯</td>
<td>vₘ,n</td>
<td></td>
</tr>
<tr>
<td>a₁ description of action a₁</td>
<td>w₁,₁</td>
<td>⋯</td>
<td>w₁,n</td>
<td>w₁,e</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td></td>
</tr>
<tr>
<td>aₖ description of action aₖ</td>
<td>wₖ,₁</td>
<td>⋯</td>
<td>wₖ,n</td>
<td>wₖ,e</td>
</tr>
</tbody>
</table>

• Semantics:

\[
\mathcal{F}(\text{else}) := \neg \left( \bigvee_{r \in T \setminus \{\text{else}\}} \mathcal{F}_{\text{pre}}(r) \right) \land F(w₁,e,a₁) \land \cdots \land F(wₖ,e,aₖ)
\]

Proposition. If decision table T has an ‘else’-rule, then T is complete.
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

$$\exists r' \neq r \in T \quad \models (\mathcal{F}_{pre}(r) \implies \mathcal{F}_{pre}(r')) \land (\mathcal{F}_{eff}(r) \iff \mathcal{F}_{eff}(r')).$$

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

- Again: uselessness is decidable; reduces to SAT.
# Uselessness: Example

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>×</td>
<td>−</td>
<td>∗</td>
<td>−</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>−</td>
<td>×</td>
<td>∗</td>
<td>×</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

- 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 (→ 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.

- Rule \( r_4 \) is subsumed by \( r_3 \).
- Rule \( r_3 \) is not subsumed by \( r_4 \).

- A very precise **objective** requirements specification may not be easily understandable by every affected person.
  → provide redundant explanations.

- It is not trivial to have both, low maintenance effort and low access effort.
  → **value low access effort higher**, a requirements specification document is much more often **read** than **changed** or **written** (and most changes require reading beforehand).

- Useless rules “do not hurt” as such.
- Yet useless rules should be removed to make the table more readable, yielding an easier usable specification.
**Definition.** [Determinism]
A decision table $T$ is called **deterministic** if and only if the premises of all rules are pairwise disjoint, i.e. if

$$\forall r_1 \neq r_2 \in T \bullet \models \neg (F_{pre}(r_1) \land F_{pre}(r_2)),$$

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

• And again: determinism is **decidable**; reduces to SAT.
Determinism: Example

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

- Is $T$ deterministic?
## Determinism: Example

### Table

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

- Is $T$ deterministic? **Yes.**
Determinism: Another Example

\[ T_{\text{abstr}}: \text{room ventilation} \]

<table>
<thead>
<tr>
<th></th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>( go )</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( stop )</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

- Is \( T_{\text{abstr}} \) deterministic? **No.**

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 \( T_{\text{abstr}} \) 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

- (Basic) Decision Tables
  - Syntax, Semantics

- …for Requirements Specification

- …for Requirements Analysis
  - Completeness,
  - Useless Rules,
  - Determinism

- Domain Modelling
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation

- Collecting Semantics

- Discussion
Domain Modelling for Decision Tables
Domain Modelling

Example:

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

- If $on$ and $off$ model opposite output values of **one and the same sensor** for “room ventilation on/off”, then $\sigma \models on \land off$ and $\sigma \models \neg on \land \neg off$ **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 $\varphi_{\text{conf}}$ 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 $\varphi_{\text{conf}} = (\text{on} \land \text{off}) \lor (\neg \text{on} \land \neg \text{off})$.
  “on models an opposite of off, neither can both be satisfied nor both non-satisfied at a time”

- Notation:

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

$\neg ((\text{on} \land \text{off}) \lor (\neg \text{on} \land \neg \text{off}))$
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

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

- **Intuition**: a relative complete decision table explicitly cares for all cases which ‘may happen’.
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

$$\models \varphi_{confl} \lor \bigvee_{r \in T} F_{pre}(r).$$

- **Intuition:** a relative complete decision table explicitly cares for all cases which ‘may happen’.

- **Note:** with $\varphi_{confl} = false$, we obtain the previous definitions as a special case.
  
  **Fits intuition:** $\varphi_{confl} = false$ means we don’t exclude any states from consideration.
# Example

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

\[ \neg[(\text{on} \land \text{off}) \lor (\neg\text{on} \land \neg\text{off})] \]

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

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

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

- 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:

<table>
<thead>
<tr>
<th>T</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>splq</td>
<td>×</td>
<td>×</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>thrq</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lgsw</td>
<td>×</td>
<td>*</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>spd</td>
<td>*</td>
<td>×</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>spl</td>
<td>×</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>thr</td>
<td></td>
<td></td>
<td></td>
<td>−</td>
</tr>
</tbody>
</table>

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.
**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:**

<table>
<thead>
<tr>
<th>( T ): room ventilation</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
<th>( r_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>✗</td>
<td>✗</td>
<td>–</td>
<td>✗</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>✗</td>
<td>–</td>
<td>*</td>
<td>✗</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>–</td>
<td>✗</td>
<td>*</td>
<td>✗</td>
</tr>
<tr>
<td>start ventilation</td>
<td>✗</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>–</td>
<td>✗</td>
<td>–</td>
<td>✗</td>
</tr>
</tbody>
</table>

\[ \neg[(on \land off) \lor (\neg on \land \neg off)] \]

- **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

- **(Basic) Decision Tables**
  - Syntax, Semantics

- …for Requirements Specification

- …for Requirements Analysis
  - Completeness,
  - Useless Rules,
  - Determinism

- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation

- **Collecting Semantics**

- **Discussion**
Conflicting Actions
Definition. [Conflict Relation] A conflict relation on actions $A$ is a transitive and symmetric relation $\not\subseteq (A \times A)$.

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

(i) Rule $r$ is called consistent with conflict relation $\not\subseteq$ if and only if there are no conflicting actions in its effect, i.e. if

$$\models \mathcal{F}_{\text{eff}}(r) \rightarrow \bigwedge_{(a_1, a_2) \in \not\subseteq} \neg (a_1 \land a_2).$$

(ii) $T$ is called consistent with $\not\subseteq$ iff all rules $r \in T$ are consistent with $\not\subseteq$.

- Again: consistency is decidable; reduces to SAT.
Example: Conflicting Actions

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

$\neg[(on \land off) \lor (\neg on \land \neg off)]$

- Let $\mathcal{I}$ be the transitive, symmetric closure of $\{(\text{stop}, \text{go})\}$.
  “actions stop and go are not supposed to be executed at the same time”
- Then rule $r_1$ is inconsistent with $\mathcal{I}$.

- 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 (→ in a minute).
Content

- (Basic) Decision Tables
  - Syntax, Semantics

- …for Requirements Specification

- …for Requirements Analysis
  - Completeness,
  - Useless Rules,
  - Determinism

- Domain Modelling
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation

- Collecting Semantics

- 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

$$F_{\text{coll}}(T) := \bigwedge_{a \in A} a \leftrightarrow \bigvee_{r \in T, r(a) = \times} F_{\text{pre}}(r)$$

is called the **collecting semantics** of $T$.

- We say, $\sigma$ is **allowed by** $T$ in the collecting semantics if and only if $\sigma \models F_{\text{coll}}(T)$.

That is, if exactly **all actions** of **all enabled** rules are planned/executed.

**Example:**

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$\ast$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$\ast$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$blink$ blink button</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

$\neg[(on \land off) \lor (\neg on \land \neg off)]$

- “Whenever the button is pressed, let it blink (in addition to go/stop action.”
Definition. [Consistency in the Collecting Semantics]

Decision table $T$ is called **consistent with conflict relation $\not\vdash$ in the collecting semantics** (under conflict axiom $\varphi_{conf}$) if and only if there are no conflicting actions in the effect of jointly enabled transitions, i.e. if

$$\models F_{coll}(T) \land \neg \varphi_{conf} \rightarrow \bigwedge_{(a_1, a_2) \in \not\vdash} \neg (a_1 \land a_2).$$
Discussion
“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.
Two broad directions:

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

<table>
<thead>
<tr>
<th>T: room ventilation</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
</tr>
<tr>
<td>ventilation off?</td>
<td>( \times )</td>
<td>( - )</td>
<td></td>
</tr>
<tr>
<td>ventilation on?</td>
<td>( - )</td>
<td>( \times )</td>
<td></td>
</tr>
<tr>
<td>start ventilation</td>
<td>( \times )</td>
<td>( - )</td>
<td></td>
</tr>
<tr>
<td>stop ventilation</td>
<td>( - )</td>
<td>( \times )</td>
<td></td>
</tr>
</tbody>
</table>

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.
**Tell Them What You’ve Told Them...**

- **Decision Tables**: one example for a formal requirements specification language with
  - formal syntax,
  - formal semantics.

- Requirements analysts can use **DTs** to
  - formally (objectively, precisely)
  describe their understanding of requirements. Customers may need translations/explanation!

- **DT** properties like
  - (relative) completeness, determinism,
  - uselessness,
  can be used to **analyse** requirements.
  The discussed DT properties are **decidable**, there can be automatic analysis tools.

- **Domain modelling** formalises assumptions on the context of software; for DTs:
  - conflict axioms, conflict relation,
  Note: wrong assumptions can have serious consequences.
References

