Topic Area Requirements Engineering: Content

- Introduction
- Definition: Software & SW Specification
- 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
- Wrap-Up
• 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

<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>$\vdots$</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>$\vdots$</td>
<td>$\vdots$</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 $r_i$. 


Decision Table Semantics

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

<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>( \vdots )</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>( \vdots )</td>
<td>( \vdots )</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>

is assigned to a propositional logical formula \( \mathcal{F}(r) \) 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

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

where

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

\( =: \mathcal{F}_{\text{pre}}(r) \)

\( =: \mathcal{F}_{\text{eff}}(r) \)
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>( T )</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>

- \( F(r_1) = F(x, c_1) \land F(x, c_2) \land F(\neg c_3) \land F(x, a_1) \land F(\neg a_2) \)
  \[ = c_1 \land c_2 \land \neg c_3 \land a_1 \land \neg a_2 \]

- \( F(r_2) = c_1 \land \neg c_2 \land c_3 \land \neg a_1 \land a_2 \)

- \( F(r_3) = \neg c_1 \land \text{true} \land \text{true} \land \neg a_1 \land \neg a_2 \)
**Decision Tables as 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 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.
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

$$\models \bigvee_{r \in T} \mathcal{F}_{\text{pre}}(r).$$
Completeness: 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>$\times$</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 $T$ complete?

**NO**: $b \rightarrow \text{true}$, $off \rightarrow \text{true}$, $on \rightarrow \text{true}$
Completeness: 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 $T$ 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 $T$. 

$$
\begin{array}{|c|c|}
\hline
\text{Customer's needs complete} & \text{(formally) complete} \\
\hline
\text{yes} & \text{yes} \\
\hline
\text{no} & \text{no} \\
\hline
\end{array}
$$
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>$r_1$</th>
<th>$\cdots$</th>
<th>$r_n$</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$v_{1,1}$</td>
<td>$\cdots$</td>
<td>$v_{1,n}$</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_m$</td>
<td>$v_{m,1}$</td>
<td>$\cdots$</td>
<td>$v_{m,n}$</td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>$w_{1,1}$</td>
<td>$\cdots$</td>
<td>$w_{1,n}$</td>
<td>$w_{1,e}$</td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_k$</td>
<td>$w_{k,1}$</td>
<td>$\cdots$</td>
<td>$w_{k,n}$</td>
<td>$w_{k,e}$</td>
</tr>
</tbody>
</table>

- Semantics:

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

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

$$\exists r' \neq r \in T \mid \models (F_{\text{pre}}(r) \implies F_{\text{pre}}(r')) \land (F_{\text{eff}}(r) \iff F_{\text{eff}}(r')).$$

$r$ 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.
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 $r_1$: A very precise objective requirements specification may not be easily understandable by every affected person.
  → provide redundant explanations.

- Rule $r_2$: 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

$$\forall r_1 \neq r_2 \in T \iff \neg (F_{pre}(r_1) \land F_{pre}(r_2)).$$

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>button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>start ventilation</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

- Is $T$ **deterministic**? Yes.

- $\overline{\text{pre}(b)} \land \overline{\text{pre}(r_2)}$

- $\begin{array}{c|c|c|c|c|c}
   b & m & \text{off} & r_1 & r_2 & r_3 \\
   \hline
   0 & 0 & 0 & - & - & - \\
   0 & 0 & 1 & : & : & : \\
   0 & 0 & 0 & : & : & : \\
   : & : & : & : & : \\
   1 & 1 & 1 & - & - & - \\
\end{array}$
**Determinism: Another Example**

<table>
<thead>
<tr>
<th>$T_{abstr}$: 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>$go$ 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 $T_{abstr}$ 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 $T_{abstr}$ 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
Domain Modelling for Decision Tables
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>$o$ ventilation off?</td>
<td>×</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>$o$ ventilation on?</td>
<td></td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>$g$ start ventilation</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s$ stop ventilation</td>
<td></td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

- If $o$ and $o$ model opposite output values of **one and the same sensor** for “room ventilation on/off”, then $\sigma \models o \land o$ and $\sigma \models \neg o \land \neg o$ **never happen** in reality for any observation $\sigma$.

- Decision table $T$ is incomplete for exactly these cases.
  ($T$ “does not know” that $o$ and $o$ can be opposites in the real-world).

- We should be able to “tell” $T$ that $o$ and $o$ are opposites (if they are).
  Then $T$ would be **relative complete** (relative to the domain knowledge that $o/o$ 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 \( \varphi_{confl} = (on \land off) \lor (\neg on \land \neg off) \).

  “\( on \) 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>( 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>

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

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

- **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>$b$</td>
<td>button pressed?</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$\mathtt{off}$</td>
<td>ventilation off?</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>$\mathtt{on}$</td>
<td>ventilation on?</td>
<td>−</td>
<td>×</td>
</tr>
<tr>
<td>$\mathtt{go}$</td>
<td>start ventilation</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>$\mathtt{stop}$</td>
<td>stop ventilation</td>
<td>−</td>
<td>×</td>
</tr>
</tbody>
</table>

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

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

- **Pitfall**: if $\mathtt{on}$ and $\mathtt{off}$ are outputs of two different, independent sensors, then $\sigma \models \mathtt{on} \land \mathtt{off}$ 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></th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>( splq )</td>
<td>spoilers requested</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( thr )</td>
<td>thrust-reverse requested</td>
<td>( _ )</td>
<td>( _ )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( lgsw )</td>
<td>at least 6.3 tons weight on each landing gear strut</td>
<td>( \times )</td>
<td>( _ )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( spd )</td>
<td>wheels turning faster than 133 km/h</td>
<td>( _ )</td>
<td>( _ )</td>
<td>( _ )</td>
</tr>
<tr>
<td>( spl )</td>
<td>enable spoilers</td>
<td>( \times )</td>
<td>( _ )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( thr )</td>
<td>enable thrust-reverse</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>$b$ 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>$\times$</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$\ast$</td>
<td>$\times$</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>$\times$</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

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

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

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

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

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

- 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 $\not\!
ot\!
ot\!$ 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 $\not\!
ot\!
ot\!$.

- 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

Logic
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:

\[ T: \text{room ventilation} \]

<table>
<thead>
<tr>
<th></th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>×</td>
<td>×</td>
<td>−</td>
<td>×</td>
</tr>
<tr>
<td>$\text{off}$</td>
<td>×</td>
<td>−</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{on}$</td>
<td>−</td>
<td>×</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{go}$</td>
<td>×</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$\text{stop}$</td>
<td>−</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$\text{blink}$</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>×</td>
</tr>
</tbody>
</table>

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

- “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 $\not\in$ in the collecting semantics (under conflict axiom $\varphi_{\text{conf}}$) if and only if there are no conflicting actions in the effect of jointly enabled transitions, i.e. if

$$\models \mathcal{F}_{\text{coll}}(T) \land \neg \varphi_{\text{conf}} \rightarrow \bigwedge_{(a_1,a_2) \in \not\in} \neg (a_1 \land a_2).$$
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.
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.

### Table: room ventilation

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>else</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>

$\sigma$ valuation:

- $\sigma$ is a valuation of the DT.
- $\sigma$ is used to determine whether a scenario is valid or not.
- $\sigma$ is the result of translating the system scenario $S$.

F: room ventilation

<table>
<thead>
<tr>
<th>$r_1$</th>
<th>$r_2$</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>–</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>×</td>
<td>–</td>
</tr>
</tbody>
</table>

$\sigma$: valuation

- $\sigma$ is a valuation of the DT.
- $\sigma$ is used to determine whether a scenario is valid or not.
- $\sigma$ is the result of translating the system scenario $S$.

$\sigma$ valuation:

- $\sigma$ is a valuation of the DT.
- $\sigma$ is used to determine whether a scenario is valid or not.
- $\sigma$ is the result of translating the system scenario $S$.
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

