Softwaretechnik / Software-Engineering

Lecture 8: Decision Tables

2018-05-28

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Requirements Engineering: Content

- 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

- 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

- Preparation of tests:
  - Without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → systematic testing hardly possible

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 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.
**Structure of Topic Areas**

**Example: Requirements Engineering**

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

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

  - e.g. “Whenever a crash...”
  - e.g. “Always, if (crash) at t...”
  - e.g. “∀ t, t’ ∈ Time • ...”

**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

**Logic**
**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>$C$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$w_1, w_1, v_1, v_1$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$c_m$</td>
<td>$w_m, w_m, v_m, v_m$</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 $\{v_1,1, \ldots, v_m,n, w_1,1, \ldots, w_k,n\}$, $1 \leq i \leq n$, are called **rules**.
- $r_1, \ldots, r_n$ are **rule names**.
- $\{v_1,1, \ldots, v_m,n\}$ is called **premise** of rule $r_i$.
- $\{w_1,1, \ldots, w_k,n\}$ is called **effect** of $r_i$. 

---

*Note: The text continues beyond the visible area.*
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>$\ldots$</th>
<th>$r_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>description of condition $c_1$</td>
<td>$r_{1,1}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_m$</td>
<td>description of condition $c_m$</td>
<td>$r_{m,1}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>description of action $a_1$</td>
<td>$r_{1,1}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_k$</td>
<td>description of action $a_k$</td>
<td>$r_{k,1}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

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

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

**Decision Table Semantics: Example**

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

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

- $F(r_1) = c_1 \land c_2 \land c_3 \land \neg c_4 \land \neg a_2$
- $F(r_2) = c_1 \land \neg c_2 \land c_3 \land \neg c_4 \land \neg a_2 \land a_1$
- $F(r_3) = \neg c_1 \land c_2 \land \neg c_3 \land \neg a_4 \land \neg a_2$
Yes, And?

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>go</th>
<th>stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off</td>
<td>×</td>
<td>∗</td>
</tr>
<tr>
<td>button pressed?</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>start ventilation</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>stop ventilation</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
  \[
  \sigma(b) := \text{true, if button pressed now and } \sigma(b) := \text{false, if button not pressed now.}
  \]
  \[
  \sigma(go) := \text{true, we plan to start ventilation and } \sigma(go) := \text{false, we plan to stop ventilation.}
  \]
- 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 \( \sigma \nmid \varphi \) otherwise).
- Rule formulae \( F(r) \) are propositional formulae over \( C \cup A \) thus, given \( \sigma \), we have either \( \sigma \models F(r) \) or \( \sigma \nmid F(r) \).
- 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 \( \sigma \models F(r) \).
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: $\sigma_1 = \{ b \mapsto \text{true}, \text{off} \mapsto \text{true}, \text{on} \mapsto \text{false}, \text{go} \mapsto \text{true}, \text{stop} \mapsto \text{false} \}$.
- 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: $\sigma_3 = \{ b \mapsto \text{false}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{true}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true} \}$.
- Is our intention (to stop the ventilation now) allowed by $T$?

---

**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>×</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>
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>T'1: cash a cheque</th>
<th>r₁</th>
<th>r₂</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁ credit limit exceeded?</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>c₂ payment history ok?</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>c₃ overdraft &lt; 500 €?</td>
<td>-</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>a₁ cash cheque</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>a₂ do not cash cheque</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>a₃ offer new conditions</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- clerk checks database state (yields σ for c₁, . . . , c₃),
- database says: credit limit exceeded over 500 €, but payment history ok,
- clerk cashes cheque but offers new conditions (according to T'1).
Decision Tables for Requirements Analysis

Recall Once Again

Requirements on Requirements Specifications

A requirements specification should be:

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

**Correctness and completeness** are defined relative to something which’s usually only **in the customer’s head**.

→ it is difficult to be sure of correctness and completeness.

“Dear customer, please tell me what is in your head?!” is in almost all cases not a solution!

It’s not unusual that even the customer does not precisely know…

For example, the customer may not be aware of contradictions due to technical limitations.
Completeness

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

$$\models \bigvee_{r \in T} F_{\text{pre}}(r).$$

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>$-$</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.** (Because there is no rule for, e.g., the case $\sigma(b) = \text{true}, \sigma(on) = \text{false}, \sigma(off) = \text{false}$).

Recall:

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

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

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

$$F_{\text{pre}}(r_1) \lor F_{\text{pre}}(r_2) \lor F_{\text{pre}}(r_3)$$

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

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>$r_1$</th>
<th>$r_2$</th>
<th>$\ldots$</th>
<th>$r_n$</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$ : description of condition $c_1$</td>
<td>$\varphi_1$, $\varphi_1, v_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_m$ : description of condition $c_m$</td>
<td>$\varphi_m$, $\varphi_m, v_m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$ : description of action $a_1$</td>
<td>$\psi_1$, $\psi_1$, $w_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_k$ : description of action $a_k$</td>
<td>$\psi_k$, $\psi_k$, $w_k$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Semantics:

$$
\mathcal{F}(\text{else}) := \neg \left( \bigvee_{r \in T \setminus \{\text{else}\}} \mathcal{F}_{\text{pre}}(r) \right) \land F(w_1, e, a_1) \land \cdots \land F(w_k, e, a_k)
$$

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

**Definition. [Uselessness]** Let $T$ be a decision table. A rule $r \in T$ is called **useless** (or: redundant) if and only if there is another (different) rule $r' \in T$
- whose premise is implied by the one of $r$ and
- whose effect is the same as $r$'s,

i.e. if

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

$r$ 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>$x$</td>
<td>$x$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>$x$</td>
<td>$-$</td>
<td>$x$</td>
<td>$-$</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>$-$</td>
<td>$x$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>$x$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>$-$</td>
<td>$x$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

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

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

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

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

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>×</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>

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

<table>
<thead>
<tr>
<th>$T_{abstr}$: room ventilation</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</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>

- 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

- **(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

Logic
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>$\times$</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</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>$\times$</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]).
A conflict axiom over conditions \( C \) is a propositional formula \( \varphi_{conf} \) 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 \( \varphi_{conf} = (\text{on} \land \neg\text{off}) \lor (\neg\text{on} \land \neg\text{off}) \).
  
  "on models an opposite of off, neither can both be satisfied nor both non-satisfied at a time"

**Notation:**

\[
\begin{array}{|c|c|c|c|}
\hline
T & \text{room ventilation} & r_1 & r_2 \\
\hline
& \text{button pressed?} & \times & \times \\
\text{off} & \text{ventilation off?} & \times & \times \\
\text{on} & \text{ventilation on?} & \times & \times \\
\text{go} & \text{start ventilation} & \times & \times \\
\text{stop} & \text{stop ventilation} & \times & \times \\
\hline
\end{array}
\]

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

### Relative Completeness

**Definition.** [Completeness wrt. Conflict Axiom]

A decision table \( T \) is called complete wrt. conflict axiom \( \varphi_{conf} \) if and only if the disjunction of all rules' premises and the conflict axiom is a tautology, i.e. if

\[
\models \varphi_{conf} \lor \bigvee_{r \in T} \mathcal{F}_{pre}(r).
\]

**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 $\varphi_{\text{confi}}$ if and only if the disjunction of all rules' premises and the conflict axiom is a tautology, i.e. if

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

- **Intuition:** a relative complete decision table explicitly cares for all cases which 'may happen'.
- **Note:** with $\varphi_{\text{confi}} = \text{false}$, we obtain the previous definitions as a special case.
  
  Fits intuition: $\varphi_{\text{confi}} = \text{false}$ means we don't exclude any states from consideration.

---

**Example**

<table>
<thead>
<tr>
<th>room ventilation</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>on button pressed?</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>x</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>
<tr>
<td>$\neg \left[\left(\text{on} \land \text{off}\right) \lor \neg \left(\text{on} \land \neg \text{off}\right)\right]$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $T$ is complete wrt. its conflict axiom.
- **Pitfall:** if on and 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>splrs</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>thrrs</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 \( \varphi_{conf} \) if and only if the premise of \( r \) implies the conflict axiom, i.e. if \( \vdash \varphi_{pre}(r) \rightarrow \varphi_{conf} \).

• Intuition: a vacuous rule would only be enabled in states which ‘cannot happen’.
• Example:

<table>
<thead>
<tr>
<th>T: room ventilation</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</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>

~(on ∧ off) \lor (¬on ∧ ¬off)

• Vacuity wrt. \( \varphi_{conf} \): 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 \( \mathcal{I} \subseteq (A \times A) \).

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

(i) Rule \( r \) is called consistent with conflict relation \( \mathcal{I} \) if and only if there are no conflicting actions in its effect, i.e. if

\[
\models F_{\text{eff}}(r) \rightarrow \bigwedge_{(a_1, a_2) \in \mathcal{I}} \neg(a_1 \land a_2).
\]

(ii) \( T \) is called consistent with \( \mathcal{I} \) iff all rules \( r \in T \) are consistent with \( \mathcal{I} \).

• 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>( \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>


• 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 (\( \rightarrow \) in a minute).
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
  \[ F_{\text{coll}}(T) := \bigwedge_{a \in A} a \leftrightarrow \bigvee_{r \in T, r(a) = x} F_{\text{pre}}(r) \]
  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>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$o$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$s$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$g$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$s$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

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

Consistency in the Collecting Semantics

Definition. [Consistency in the Collecting Semantics]

Decision table $T$ is called consistent with conflict relation $\mathcal{C}$ 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

\[ \models F_{\text{coll}}(T) \land \lnot \varphi_{\text{conf}} \land \bigwedge_{(a_1, a_2) \in \mathcal{C}} \lnot (a_1 \land a_2). \]
Speaking of Formal Methods

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

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.
• **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**

