Softwaretechnik / Software-Engineering

Lecture 7: Formal Methods for Requirements Engineering

2016-05-30

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
- Working Definition: Software
- Live Sequence Charts
  - Syntax, Semantics
- Discussion
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>$\ldots$</th>
<th>$r_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$ description of condition $c_1$</td>
<td>$v_{1,1}$</td>
<td>$\ldots$</td>
<td>$v_{1,n}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ldots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$c_m$ description of condition $c_m$</td>
<td>$v_{m,1}$</td>
<td>$\ldots$</td>
<td>$v_{m,n}$</td>
</tr>
<tr>
<td>$a_1$ description of action $a_1$</td>
<td>$w_{1,1}$</td>
<td>$\ldots$</td>
<td>$w_{1,n}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ldots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$a_k$ description of action $a_k$</td>
<td>$w_{k,1}$</td>
<td>$\ldots$</td>
<td>$w_{k,n}$</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**.
- $r_1, \ldots, r_m$ 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>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$ description of condition $c_1$</td>
<td>$v_1,1$</td>
<td>$v_2,2$</td>
<td>$v_3,3$</td>
</tr>
<tr>
<td>$c_2$ description of condition $c_2$</td>
<td>$v_1,4$</td>
<td>$v_2,5$</td>
<td>$v_3,6$</td>
</tr>
<tr>
<td>$c_3$ description of condition $c_3$</td>
<td>$v_1,7$</td>
<td>$v_2,8$</td>
<td>$v_3,9$</td>
</tr>
<tr>
<td>$a_1$ description of action $a_1$</td>
<td>$w_1,1$</td>
<td>$w_2,2$</td>
<td>$w_3,3$</td>
</tr>
<tr>
<td>$a_2$ description of action $a_2$</td>
<td>$w_1,4$</td>
<td>$w_2,5$</td>
<td>$w_3,6$</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

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

Decision Table Semantics: Example

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

<table>
<thead>
<tr>
<th>$F$</th>
<th>$F_1$</th>
<th>$F_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$x$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$\ast$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$x$</td>
<td>$\ast$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\ast$</td>
<td>$\ast$</td>
</tr>
</tbody>
</table>

- $F(r_1) = F(x, c_1) \land F(x, c_2) \land F(\neg c_3, a_1) \land F(\neg c_3, a_2)$
- $F(r_2) = c_1 \land c_2 \land c_3 \land a_1 \land \neg a_2$
- $F(r_3) = \neg c_1 \land \text{true} \land \text{true} \land \text{true} \land \neg a_2$
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>Condition</th>
<th>Action</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>ventilation off?</td>
<td>off</td>
<td>ventilation on?</td>
<td>go</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>start ventilation</td>
<td>x</td>
<td>stop ventilation</td>
<td>x</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 \( \sigma : C \cup A \to B \), e.g., set
  \[
  \sigma(b) := \text{true}, \text{ if button pressed now and } \sigma(b) := \text{false}, \text{ if button not pressed now.}
  \]
  \[
  \sigma(go) := \text{true}, \text{ we plan to start ventilation and } \sigma(go) := \text{false}, \text{ we plan to stop ventilation.}
  \]
- A valuation \( \sigma : C \cup A \to B \) 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 \( F(r) \) are propositional formulae over \( C \cup A \) thus, given \( \sigma \), we have either \( \sigma \models F(r) \) or \( \sigma \not\models F(r) \).
- 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>r1</th>
<th>r2</th>
<th>r3</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</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>

\[
F(r_1) = b \land \neg c_1 \land \neg c_2 \land \neg c_3 \land \neg \text{go} \land \neg \text{stop}
\]
\[
F(r_2) = b \land \neg c_1 \land \neg c_2 \land \neg c_3 \land \text{a_1} \land \text{a_2}
\]
\[
F(r_3) = \neg b \land \text{true} \land \text{true} \land \neg \text{a_1} \land \neg \text{a_2}
\]

(i) **Assume**: button pressed, ventilation off, we (only) plan to start the ventilation.

\[\sigma = \{ b \mapsto \text{true}, \text{off} \mapsto \text{true}, \text{on} \mapsto \text{false}, \text{go} \mapsto \text{false}, \text{stop} \mapsto \text{false}\}\]

✓ allowed by \(r_1\) of \(T\)

(ii) **Assume**: button pressed, ventilation on, we (only) plan to stop the ventilation.

\[\sigma = \{ b \mapsto \text{true}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{true}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true}\}\]

✓ allowed by \(r_2\) of \(T\)

(ii) **Assume**: button not pressed, ventilation on, we (only) plan to stop the ventilation.

\[\sigma = \{ b \mapsto \text{false}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{false}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true}, \text{go} \mapsto \text{false}\}\]

✗ not allowed by \(r_3\) of \(T\)
**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>$\text{off}$ ventilation off?</td>
<td>×</td>
<td>—</td>
<td>×</td>
</tr>
<tr>
<td>$\text{on}$ ventilation on?</td>
<td>—</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>$\text{go}$ start ventilation</td>
<td>×</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>$\text{stop}$ 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>$T$: cash a cheque</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_3$ credit limit exceeded?</td>
<td>×</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>$c_2$ payment history ok?</td>
<td>x</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>$c_1$ overdraft &lt; 500 €?</td>
<td>—</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>$a_3$ cash cheque</td>
<td>×</td>
<td>—</td>
<td>x</td>
</tr>
<tr>
<td>$a_2$ do not cash cheque</td>
<td>—</td>
<td>x</td>
<td>—</td>
</tr>
<tr>
<td>$a_1$ offer new conditions</td>
<td>x</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

- clerk checks database state (yields $c_1, \ldots, c_3$),
- database says: credit limit exceeded, but below 500 € and payment history ok,
- clerk cashes cheque but offers new conditions (according to $T'$).
**Decision Tables as Specification Language**

**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.

---

... so, off to “technological paradise where […] everything happens according to the blueprints”.

(Kapetan, 2015; Lovins and Lovins, 2001)
## 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, in 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.

"*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. A decision table $T$ is called \textbf{complete} if and only if the disjunction of all rules' premises is a \textbf{tautology}, i.e. if

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

Completeness: Example

$T$: 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>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>off</td>
<td>$\times$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>on</td>
<td>$-$</td>
<td>$\times$</td>
<td>$+$</td>
</tr>
<tr>
<td>go</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?

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

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 \textbf{not a tautology}. 
For Convenience: The ‘else’ Rule

- Syntax:

\[
\begin{array}{|c|c|c|c|}
\hline
\text{condition} & \text{true} & \text{false} & \text{else} \\
\hline
\text{condition}^1 & v_{1,1} & v_{1,2} & v_{1,n} \\
\hline
\text{condition}^m & v_{m,1} & v_{m,2} & v_{m,n} \\
\hline
\end{array}
\]

- Semantics:

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

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

$$\exists r' \neq r \in T \land (\phi_r(\text{pre}) \implies \phi_{r'}(\text{pre})) \land (\phi_r(\text{eff}) \iff \phi_{r'}(\text{eff})).$$

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

- Again: uselessness is decidable; reduces to SAT.

Uselessness: Example

$T$: 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>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>

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

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

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

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

Otherwise, $T$ is called non-deterministic.

• And again: uselessness 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>$T_{\text{abstr}}$: 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>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 $T_{\text{abstr}}$ 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.
Domain Modelling for Decision Tables

Domain Modelling

Example:

<table>
<thead>
<tr>
<th>Condition</th>
<th>r1</th>
<th>r2</th>
<th>r3</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>

- 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 $\sigma \not\models \neg on \land \neg off$ **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 $\varphi_{\text{conf}} = (\text{on} \land \neg \text{off}) \lor (\neg \text{on} \land \text{off})$.

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

• Notation:

Relative Completeness

**Definition.** (Completeness wrt. Conflict Axiom)
A decision table $T$ is called complete wrt. conflict axiom $\varphi_{\text{conf}}$ if and only if the disjunction of all rules’ premises and the conflict axiom is a tautology, i.e. if

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

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

• Note: with $\varphi_{\text{conf}} = \text{false}$, we obtain the previous definitions as a special case.

Fits intuition: $\varphi_{\text{conf}} = \text{false}$ means we don’t exclude any states from consideration.
Example

<table>
<thead>
<tr>
<th>T: room ventilation</th>
<th>r₁</th>
<th>r₂</th>
<th>r₃</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>go</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>¬(on ∧ off) ∨ (¬on ∧ ¬off)</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 \( \sigma \models on \land off \) is possible in reality (e.g. due to sensor failures).
  
Decision table T does not tell us what to do in that case!

More Pitfalls in Domain Modelling ([Wikipedia, 2015](#))


- 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>
</tr>
</thead>
<tbody>
<tr>
<td>spoilers requested</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thrust-reverse</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>at least 6.3 tons</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight on landing</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gear strut</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wheels turning</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>faster than 133 km/h</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enable spoilers</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enable thrust-reverse</td>
<td>×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Idea: if conditions \( lgw \) 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.

![Image of airplane at airport](http://commons.wikimedia.org/wiki/File:Flight_29041129.png#/media/File:Flight_29041129.png)
**Vacuity wrt. Conflict Axiom**

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

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

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>( \varphi_{\text{conf}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>on &amp; ventilation pressed?</td>
<td>×</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>×</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>×</td>
</tr>
<tr>
<td>on</td>
<td>×</td>
</tr>
<tr>
<td>start ventilation</td>
<td>×</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>×</td>
</tr>
</tbody>
</table>

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

- **Vacuity wrt. \( \varphi_{\text{conf}} \):** 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!

---

**Conflicting Actions**
**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

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

(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>( )</th>
<th>room ventilation</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>button pressed?</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
</tr>
<tr>
<td>( a )</td>
<td>ventilation off?</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
</tr>
<tr>
<td>( c )</td>
<td>ventilation on?</td>
<td>( x )</td>
<td>( \star )</td>
<td>( \star )</td>
</tr>
<tr>
<td>( s )</td>
<td>start ventilation</td>
<td>( \star )</td>
<td>( \star )</td>
<td>( \star )</td>
</tr>
<tr>
<td>( s )</td>
<td>stop ventilation</td>
<td>( \star )</td>
<td>( \star )</td>
<td>( \star )</td>
</tr>
<tr>
<td>( \neg ((\text{on} \land \text{off}) \lor (\neg \text{on} \land \neg \text{off})) )</td>
<td></td>
<td></td>
<td></td>
<td></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).
Collecting Semantics

- Let \( T \) be a decision table over \( C \) and \( A \) and \( \sigma \) be a model of an observation of \( C \) and \( A \). Then
  
  \[ \mathcal{F}_{\text{coll}}(T) := \bigwedge_{a \in A} a \leftrightarrow \bigvee_{r \in T, r(a) = \times} \mathcal{F}_{\text{pre}}(r) \]

  is called the collecting semantics of \( T \).

- We say, \( \sigma \) is allowed by \( T \) in the collecting semantics if and only if \( \sigma \models \mathcal{F}_{\text{coll}}(T) \).
  That is, if exactly all actions of all enabled rules are planned/executed.

Example:

- "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

$$\models \mathcal{F}_{\text{coll}}(T) \land \varphi_{\text{conf}} \rightarrow \bigwedge_{(a_1, a_2) \in \mathcal{C}} \neg(a_1 \land a_2).$$

Discussion
Speaking of Formal Methods

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

• Decision Tables: an example for a formal requirements specification language with
  • formal syntax,
  • formal semantics.

• Analysts can use DTs to
  • formally (objectively, precisely)
    describe their understanding of requirements.

• 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


