Last Lecture:
• requirements engineering basics, "the natural language case"

This Lecture:
• Educational Objectives:
  • What is a rule, a decision table?
  • What is the interleaving, collecting, update semantics?
  • Analyse this rule for: vacuous, redundant, complete, consistent
  • In what sense can decision tables serve as requirements specification language?
  • Formalise this requirement with a decision table.
  • What does this have to do with the previous lecture?

Content:
• definition decision table syntax and interleaving semantics,
• definition tables as requirements specification,
• interesting/useful properties of decision tables

Recall: Formal Methods

Formal Methods (in the Software Development Domain)

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

(Kopetz, 2011; Lovins and Lovins, 2001)

Definition.

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

Example: If a method includes, as a tool, a specification language, then that language has
• a formal syntax,
• a formal semantics,
• a formal proof system.

Formal, Rigorous, or Systematic Development

"The techniques of a formal method help
• construct a specification, and/or
• analyse a specification, and/or
• transform (refine) one (or more) specification(s) into a program.

The techniques of a formal method, (besides the specification languages) are typically software packages that help developers use the techniques and other tools. The aim of developing software, either
• formally (all arguments are formal)
• rigorously (some arguments are made and they are formal)
• systematically (some arguments are made on a form that can be made formal)
is to (be able to) reason in a precise manner about properties of what is being developed." (Bjørner and Havelund, 2014)

Decision tables (DT) are one example for a formal requirements specification language:
• we give a formal syntax and semantics,
• requirements quality criteria, e.g. completeness, can be formally defined,
• thus for a DT we can formally argue whether it is complete or not,
• (some) formal arguments can be done automatically (→ tool support).
A set of conditions: $\{ \pi_1, \ldots, \pi_r \}$, written $\phi_1 \land \ldots \land \phi_r$:

$$\begin{align*}
\phi_1 & := \neg C_1 \\
\phi_2 & := A \land \neg C_2 \\
\phi_3 & := A \land \neg \gamma_1, \ldots, \gamma_n
\end{align*}$$

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Start Ventilation</th>
<th>Stop Ventilation</th>
<th>Button Pressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>$\neg \pi_1$</td>
<td>$\pi_1$</td>
<td>$\gamma_1$</td>
</tr>
<tr>
<td>Start</td>
<td>$\pi_1$</td>
<td>$\neg \pi_1$</td>
<td>$\neg \gamma_1$</td>
</tr>
</tbody>
</table>

From the table to the rules:

$$\begin{align*}
\{ \pi_1, \pi_2, \ldots, \pi_r \} & \Rightarrow \{ \gamma_1, \gamma_2, \ldots, \gamma_n \} \\
\{ \pi_1, \pi_2, \ldots, \pi_r \} & \Rightarrow \{ \neg \gamma_1, \neg \gamma_2, \ldots, \neg \gamma_n \}
\end{align*}$$

Decision Tables: Example

And Where’s The Table?
T satisfies specification \( [\llbracket T \rrbracket] \) of a software.

We want a software, don't we?...?

Isn't there a bell ringing...?
Page 30

Quality Criteria for DTs: Uselessness and Vacuity

Uselessness: Example

Decision Table as Requirements Specification: Examples

But formal requirements specification language tends to support abstract specifications; specifying technical details is tedious. Additional quality criteria are required in the sense of relative completeness: we can at least define a kind of requirements that are comprehensible, traceable, testable.

A rule is given in form of a table then $r'$ is another (different) rule whose effect is the same as $r$ is subsumed by $r$. A rule whose premise is the same as $r$ is not subsumed is useless. A rule $r$ is useless if and only if there is no state $\sigma$ in the sense of states $\Sigma$ if and only if $\exists r \in B \cup \{\} \rightarrow \sigma$.

In other words, a requirements specification does not constrain the realisation more than necessary, things which are not relevant to the project should not be constrained. Relevant requirements specification is correct if and only if it correctly represents the wishes/needs of the customer. Things which are not relevant to the project should not be constrained, the final product can be made of everything the customer asks for. But formal requirements specification language tends to support abstract specifications; specifying technical details is tedious. Additional quality criteria are required in the sense of relative completeness: we can at least define a kind of requirements that are comprehensible, traceable, testable.

A requirement is free of contradiction if and only if there is no state $\sigma$ in the sense of states $\Sigma$ if and only if $\exists r \in B \cup \{\} \rightarrow \sigma$.

In other words, a requirements specification does not constrain the realisation more than necessary, things which are not relevant to the project should not be constrained. Relevant requirements specification is correct if and only if it correctly represents the wishes/needs of the customer. Things which are not relevant to the project should not be constrained, the final product can be made of everything the customer asks for. But formal requirements specification language tends to support abstract specifications; specifying technical details is tedious. Additional quality criteria are required in the sense of relative completeness: we can at least define a kind of requirements that are comprehensible, traceable, testable.

A requirement is free of contradiction if and only if there is no state $\sigma$ in the sense of states $\Sigma$ if and only if $\exists r \in B \cup \{\} \rightarrow \sigma$.
• Doesn't hurt wrt. the final product: The decision table $T$ with useless rules has the same computation paths as the one with useless rules removed, thus specifies the same set of software.

• But decision tables with useless rules are unnecessarily hard to work with (read, maintain, ...).

• May make communication (with customer) harder!

**Quality Criteria for DTs: Completeness**

**Definition.**

\[ \text{Completeness} \]

A decision table $T$ is called complete if and only if the disjunction of all rules' premises is a tautology, i.e., if $|= \bigvee \phi \rightarrow \alpha \in T \phi$.

**Completeness: Example**

$T$: room ventilation

<table>
<thead>
<tr>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$b$</th>
<th>button pressed?</th>
<th>$\times$</th>
<th>$\times$</th>
<th>$\rightarrow$</th>
<th>$\ast$</th>
<th>$\ast$</th>
<th>$\rightarrow$</th>
<th>$\ast$</th>
<th>$\ast$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>ventilation off?</td>
<td>$\times$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\ast$</td>
<td>ventilation on?</td>
<td>$\times$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>go start ventilation</td>
<td>$\times$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>stop stop ventilation</td>
<td>$\times$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
</tbody>
</table>

• is not complete: there is no rule, e.g., for the case $\neg b \land \text{on} \land \neg \text{off}$.

**Incompleteness: Consequences**

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

• This very incomplete decision table:

$T$: room ventilation

<table>
<thead>
<tr>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
<th>$r_5$</th>
<th>$b$</th>
<th>button pressed?</th>
<th>$\times$</th>
<th>$\times$</th>
<th>$\rightarrow$</th>
<th>$\ast$</th>
<th>$\ast$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td>$\rightarrow$</td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
</tbody>
</table>

• forbids all actions in case $b \land \neg \text{on} \land \neg \text{off}$.

• May not be the intention of the customer!
In the abstract setting, two different, independent sensors are outputs of \( \sigma \): if

\[
\text{if off \& on = | } \sigma \text{ then effect } ( ) ;
\]

execute,

\[
\text{if off \& on = | } \sigma \text{ then effect ( ) = void ( ) ;}
\]

\[
\text{if b \& on = | } \sigma \text{ then effect ( ) = go ( ) ;}
\]

\[
\text{if b \& off = | } \sigma \text{ then effect ( ) = stop ( ) ;}
\]

In the real-world, maybe it should never happen:

\[
\text{if off \& on = | } \sigma \text{ then effect ( ) = 0 ;}
\]

\[
\text{if ( b \& on ) effect = stop ;}
\]

\[
\text{if ( b \& off ) effect = go ;}
\]

\[
\text{if ( b \& off ) effect = void ;}
\]

\[
\text{Execute,}
\]

\[
\text{Read off ( ) ;}
\]

\[
\text{Write effect ( ) ;}
\]

...and only if the premises of all rules are pairwise disjoint.

### Domain Modelling for DTs

#### Quality Criteria for DTs: Determinism

**Determinism**

1. Deterministic final product: one of the most powerful
2. Deterministic final product: A decision table which is deterministic leaves no more choices to the developer.
3. Deterministic final product: The number of possible states is known.
4. Deterministic final product: The execution time is known.
5. Deterministic final product: The system is predictable.
6. Deterministic final product: The system is easy to understand.
7. Deterministic final product: The system is easy to test.
8. Deterministic final product: The system is easy to maintain.
9. Deterministic final product: The system is easy to debug.

**Non-determinism**

1. Non-deterministic final product: A non-deterministic decision table leaves more freedom to the developer.
2. Non-deterministic final product: A non-deterministic decision table allows for more choices to the developer.
3. Non-deterministic final product: A non-deterministic decision table is more flexible.
4. Non-deterministic final product: A non-deterministic decision table is more robust.
5. Non-deterministic final product: A non-deterministic decision table is more scalable.
6. Non-deterministic final product: A non-deterministic decision table is more adaptable.
7. Non-deterministic final product: A non-deterministic decision table is more resilient.
8. Non-deterministic final product: A non-deterministic decision table is more sustainable.

**Examples**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>stop</td>
</tr>
</tbody>
</table>

**Deterministic Example**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>stop</td>
</tr>
</tbody>
</table>

**Non-deterministic Example**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>go</td>
</tr>
<tr>
<td>off &amp; on</td>
<td>stop</td>
</tr>
<tr>
<td>off &amp; off</td>
<td>stop</td>
</tr>
</tbody>
</table>
wheels didn't turn fast due to hydroplaning.

- anti-crosswind manoeuvre puts too little weight on landing gear
- actually tailwind,

http://commons.wikimedia.org/wiki/File:Flight_29041129.png

“Lufthansa Flight 2904 crash site Siecinski” by Mariusz Siecinski -

http://www.airliners.net/photo/Lufthansa/Airbus-A320-211/0265541/L/.
• In certain domains, we may not want to execute certain actions together. Let's declare them to be conflicting and define consistency also wrt. conflicting actions.

• A conflict relation on actions $\mathcal{A}$ is a transitive and symmetric relation $\mathcal{C} \subseteq (\mathcal{A} \times \mathcal{A})$.

Definition.

[Consistency] Let $r = \phi \rightarrow \alpha \in T$ be a rule.

(i) $r$ is called consistent with conflict relation $\mathcal{C}$ if and only if there are no conflicting actions in $\alpha$, i.e. if $\forall a_1, a_2 \in \alpha \cdot a_1 \mathcal{C} a_2$.

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

Example: Conflicting Actions

• Let $\mathcal{C}$ be the transitive, symmetric closure of $\text{stop} \mathcal{C} \text{go}$.

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

$T$:

- room ventilation
  - $r_1$ button pressed?
    - on ventilation on?
    - off ventilation off?
  - go start ventilation
  - stop stop ventilation

• Rule $r_1$ is inconsistent with $\mathcal{C}$.

Conflicting Actions: Consequences

• Consistency: A decision table with inconsistent rules may do harm in operation!

Detecting an inconsistency only late during a project can incur significant cost!

Inconsistencies (in particular in (multiple) decision tables, created and edited by multiple people, as well as in requirements in general) are not always as obvious as in the toy examples given here! (would be too easy...) And is more difficult to handle with the collecting semantics.

Other Semantics for Decision Tables

A Collecting Semantics for Decision Tables

• Let $T$ be a decision table and $\pi = \sigma_0 \alpha_1 \rightarrow \sigma_1 \alpha_2 \rightarrow \sigma_2 \cdot \cdot \cdot$ a state/event sequence.

• Recall: $\pi$ is a computation path of $T$ (in the interleaving semantics) if $\forall i \in \mathbb{N}_0 \exists r = \phi \rightarrow \alpha \in T \cdot \sigma_i | = \phi \land \alpha_{i+1} = \alpha$.

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

• $\pi$ is a computation path of $T$ (in the collecting semantics) if and only if $\forall i \in \mathbb{N}_0 \exists r = \phi \rightarrow \alpha \in T \cdot \sigma_i | = \phi \land \bigcup \phi' \rightarrow \alpha' \in T, \sigma_i | = \phi'$.

• That is, all rules which are enabled in $\sigma_i$ "fire" simultaneously, the joint effect is the union of the effects.

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

• Disadvantages: conflicts much less obvious.
We want the specification to be able to deal with all possible sequences of inputs. One more quality criterion, then, is to consider the set of inputs that the specification cannot deal with:

- Controlled conditions
- Uncontrolled conditions

Controlled conditions are those circumstances that can be influenced by the specification. For example, the button is not controlled by the software, but by the user external to the computer system. There is no rule enabled if the button is pressed when the ventilation is off.

In terms of decision tables:

<table>
<thead>
<tr>
<th>input sequence complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>input sequence not complete</td>
</tr>
</tbody>
</table>

Example: with work stoppage.

Computation paths of decision tables are then required to have:

- Consistency in the Collecting Semantics
- Actuality of action and state

A set of conditions and another notion of completeness is required. We require:

- There is no rule enabled if the button is pressed when the ventilation is off.
- The software switches the ventilation on or off.

Recall: a notion of vacuity is another notion of completeness.

Consistency in the Collecting Semantics:

Definition. For some systems, we can distinguish between controlled and uncontrolled conditions.

Distinguishing Controlled and Uncontrolled Conditions: Example

<table>
<thead>
<tr>
<th>button pressed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ventilation on?</td>
</tr>
<tr>
<td>ventilation off?</td>
</tr>
</tbody>
</table>

In terms of decision tables:

<table>
<thead>
<tr>
<th>button pressed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ventilation on?</td>
</tr>
<tr>
<td>ventilation off?</td>
</tr>
</tbody>
</table>

Decision Tables: Discussion

An Update Semantics for Decision Tables Cont’d

The button is not controlled by the software, but by the environment, by a user external to the computer system.
Decision Tables Summary

- Decision tables are one (very simple) example for a formal requirements specification language with
- formal syntax (there are even two: the formula and the table notation)
- formal semantics (also multiple: interleaving, collecting, update)
- a formal proof system — naja, not a dedicated one
- a requirements specification in form of decision tables allows us to formally reason about properties of what is being developed. We can, e.g., prove that a decision table is complete. (Automatically?)
- Whether a decision table is useful in a particular software development project (of course) depends on the project.
- Like many formal specification language, a decision table may not be the right tool for all problems, it may be tedious to specify all requirements using decision tables — don't do that then.
- One particular drawback of decision tables: they don't scale so well in the number of conditions.

References