A Logical Approach to Systems Engineering Artifacts and Traceability: From Requirements to Functional and Architectural Views

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Motivation

- Presentation of key artifacts in systems engineering in logic
  - Assertions about the system

- System models and their representation in logic
  - Interfaces
  - Architectures

- Key artifacts in systems engineering
  - System level requirements
  - Functional specification
  - Architecture

- Concepts for relating assertions: logical relations

- Concepts for relating artifacts
  - Understanding the logical dependencies between artifacts
  - Traceability: Intra- and inter-artifact links and traces
Traceability Use Case: ISO 26262 – Functional Safety

Management of safety requirements
- Hierarchical structure
- Traceability
- Completeness
- External consistency
- ...

Safety requirement 1
- unambiguous
- comprehensible
- atomic
- internally consistent
- feasible
- verifiable
- ...

Safety requirement 2
- unambiguous
- comprehensible
- atomic
- internally consistent
- feasible
- verifiable
- ...

Figure 3 — Relationship between management of safety requirements and particular safety requirements

6.3 Inputs to this clause
6.3.1 Prerequisites
The following information shall be available:

Safety plan (see ISO 26262-2, 6.5.1)

6.3.2 Further supporting information
None

6.4 Requirements and recommendations
6.4.1 Specification of safety requirements
6.4.1.1 To achieve the characteristics of safety requirements listed in 6.4.2.4, safety requirements shall be specified by an appropriate combination of

a) natural language and
b) methods listed in Table 1
Traceability Use Case: ISO 26262 – Functional Safety

• The management of safety requirements includes
  ◊ managing requirements,
  ◊ obtaining agreement on the requirements,
  ◊ obtaining commitments with those implementing the requirements, and
  ◊ maintaining traceability

• During the development of the software architectural design the following shall be considered:
  ◊ a) the verifiability of the software architectural design;
    NOTE This implies bi-directional traceability.

• The software unit design and implementation shall be verified in accordance with ISO 26262-8:
  ◊ b) the completeness regarding the software safety requirements and the software architecture through traceability;
Assertions
**Assertions**

- A logical predicate $p$ over a universe $D$ is a mapping
  
  $$p: D \rightarrow \text{IB}$$

  where $D$ is a mathematical set also called the *universe of discourse*.

- Often the elements $d \in D$ can be characterized by a set of *attributes*
  
  $$x_i: D \rightarrow T_i \quad \text{for} \quad 1 \leq i \leq n$$

  where $T_i$ are the (data) types for these attributes and $n$ is the number of attributes.
Example: Assertions

• For a simple universe of discourse Car representing cars, consider attributes such as
  
  length: Car → IN
  number_of_seats: Car → IN
  speed: Car → IN
  situation: Car → \{city, country, high\_way\}

• Based on the attributes, given \( d \in \text{Car} \), we write logical expressions such as
  
  \[ \text{speed}(d) \geq 50 \land \text{situation}(d) = \text{city} \]

• This notation can be simplified for a fixed car \( d \):
  
  \[ \text{speed} \geq 50 \land \text{situation} = \text{city} \]

• Such a logical expression referring to the attributes of the elements of the considered universe is called assertion.
Language of assertions

- Given a signature $\Sigma$ of attributes by $\text{LA}(\Sigma)$, we denote the assertion language over signature $\Sigma$ which is the set of assertions that can be formulated over signature $\Sigma$. 
Notation

• For assertions $Q$ the following shorthand notation is used:

$\forall X:Q$ for $\forall x_1, ..., x_n: Q$

$\exists X:Q$ for $\exists x_1, ..., x_n: Q$
where $X = \{x_1, ..., x_n\}$ are free variables in $Q$

$\forall Q$ iff $Q \equiv \text{true}$ e.g. $\forall x_1, ..., x_n: Q$
where $x_1, ..., x_n$ are all the free variables in $Q$

$\exists Q$ iff $\neg \forall \neg Q$
Artifacts - Structure and Content

• An artifact is a development document
• An artifact has structure and content
• An artifact contains content that is structured into
  ◊ (finite) sets of content chunks as well as
  ◊ finite sets of finite sets of content chunks and so on.
• This way we get nested sets of content chunks forming content hierarchies.
Illustrating Examples

• System level requirements (functional requirements)
  “the car must not increase its speed without user’s control”

• System level functional specification
  ◊ “the function acc (adaptive cruise control) accelerates the car up to
    the speed selected by the user, provided no obstacles are
    recognized in front”

• Architecture specification
  ◊ “the radar signal based sensor measures the distance to the car in
    front and sends it to the acc monitor every 100 ms”
Given two assertions P and Q; what does logical dependency mean?

Relating Assertions
Relating Assertions to Assertions - Implication

• Two assertions
  \( P, Q \)
  are in an *implication* relation if
  \( \forall (P \Rightarrow Q) \)
  or vice versa
  \( \forall (Q \Rightarrow P) \)

• Related relations are
  \( \forall (\neg Q \Rightarrow P) \)
  or
  \( \forall (P \Rightarrow \neg Q) \)
If one of the following four relations

\[ \forall (P \implies Q) \]
\[ \forall (Q \implies P) \]
\[ \forall (\neg Q \implies P) \]
\[ \forall (P \implies \neg Q) \]

holds then we call assertions P and Q *logically dependent*.
Inconsistency

• Assertions P and Q are called *inconsistent* if

\[ \neg \exists (P \land Q) \]

• If assertions P and Q are inconsistent, then both propositions

\[ \forall (P \Rightarrow \neg Q) \]
\[ \forall (Q \Rightarrow \neg P) \]

Hold, i.e. they are logically dependent.
Logical Overlap

• Two assertions $P$ and $Q$ are called *logically overlapping* iff

$$\neg \forall (P \lor Q)$$

which is equivalent to the statement,

$$\exists (\neg P \land \neg Q)$$

• Then there is a non-trivial property $R$ (nontrivial means that $\neg \forall R$ holds) that is implied both by assertion $P$ and by assertion $Q$; i.e.

$$\forall (P \Rightarrow R) \text{ and } \forall (Q \Rightarrow R)$$

• We choose the strongest assertion $R$ that is implied both by assertion $P$ and by assertion $Q$ as follows:

$$R = P \lor Q$$

• Property $R$ is not trivially true (i.e. $\exists \neg R$) iff assertions $P$ and $Q$ are overlapping.
Logical Overlap

• Not overlapping assertions are logically not independent, since

\[ \forall (P \lor Q) \]

which transforms to

\[ \neg \exists (\neg P \land \neg Q) \]

And to

\[ \forall (\neg P \Rightarrow Q) \]

\[ \forall (\neg Q \Rightarrow P) \]

• In other terms, independent assertions are always overlapping.
System Properties at Different Levels of Abstractions: Relating Views
**Example: Relating Levels of Abstraction**

<table>
<thead>
<tr>
<th>Logical_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>crash ⇒ air_bag</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technical_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>crash_sensor ⇒ activate_air_bag</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Translator</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>crash ⇔ crash_sensor</td>
</tr>
<tr>
<td>air_bag ⇔ activate_air_bag</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Translators between Levels of Abstractions

• A specification given by a set \( S_1 \subseteq LA(\Sigma_1) \) of assertions over some attribute signature \( \Sigma_1 \)
is translated into
• a specification \( S_2 \subseteq LA(\Sigma_2) \) over some attribute signature \( \Sigma_2 \)
  ◊ where signatures \( \Sigma_1 \) and \( \Sigma_2 \) only partially overlap or are disjoint
• by a set \( T \) of assertions formulated over signatures \( \Sigma_1 \) and \( \Sigma_2 \).
Translators between Levels of Abstractions

For a translation we require that for every assertion

\[ a_1 \in LA(\Sigma_1) \]

over signature \( \Sigma_1 \) there exists an assertion

\[ a_2 \in LA(\Sigma_2) \]

over \( \Sigma_2 \) such that the following formula is valid:

\[ (\land T) \Rightarrow (a_1 \leftrightarrow a_2) \]

- Then the set \( T \) is called a **translator from signature** \( \Sigma_1 \) **to**
  **signature** \( \Sigma_2 \).
- A set \( S_1 \) of assertions is called a **refinement** of a set \( S_2 \) of
  assertions according to translator \( T \) if

\[ \land T \land \land S_1 \Rightarrow \land S_2 \]
Translators between Levels of Abstractions

• If $T$ is free of contradictions $T$ is called \textit{consistent translator}.

• If for every assertion $a_1 \in \text{LA}(\Sigma_1)$ and every set $S_1$ of assertions formulated over signature $\Sigma_1$ and for every assertion $a_2 \in \text{LA}(\Sigma_2)$ and every set $S_2$ of assertions formulated over signature $\Sigma_2$
  
  \[
  [(\land T) \land (\land S_1) \Rightarrow a_1] \iff [(\land S_1) \Rightarrow a_1]
  \]
  \[
  [(\land T) \land (\land S_2) \Rightarrow a_2] \iff [(\land S_2) \Rightarrow a_2]
  \]

$T$ is called \textit{unbiased translator} between signatures $\Sigma_1$ and $\Sigma_2$. 
Logical Basis: Specifying Systems by Assertions
System and its context
Basic System Notion: What is a discrete system (model)

A system has

• a system **boundary** that determines
  ◦ what is part of the systems and
  ◦ what lies outside (called its context)

• an **interface** (determined by the system boundary), which determines,
  ◦ what ways of interaction (actions) between the system and its context are possible (static or **syntactic interface**)  
  ◦ which behavior the system shows from view of the context (**interface behavior**, dynamic interface, interaction view)

• a structure and distribution with an internal structure, given
  ◦ by its structuring in sub-systems (**sub-system architecture**)  
  ◦ by its states and state transitions (**state view**, state machines)

• **quality** profile

• the views use a **data model**

• the views may be documented by adequate models
Interfaces
Systems: the model

Sets of typed channels

\[ I = \{x_1 : T_1, x_2 : T_2, \ldots \} \]

\[ O = \{y_1 : T'_1, y_2 : T'_2, \ldots \} \]

syntactic interface

\[ (I \triangleright O) \]

data stream of type \( T \)

\[ \text{STREAM}[T] = \{\text{IN} \rightarrow T^*\} \]

valuation of channel set \( C \)

\[ \text{IH}[C] = \{C \rightarrow \text{STREAM}[T]\} \]

interface behaviour for syn. interface \((I \triangleright O)\)

\[ [I \triangleright O] = \{\text{IH}[I] \rightarrow \emptyset (\text{IH}[O])\} \]

interface specification

\[ p : I \cup O \rightarrow IB \]

represented as interface assertion \( S \)

logical formula with channel names as variables for streams
Interface Assertion

- Given a syntactic interface (I \(\rightarrow\) O) with
  - a set I of typed input channels and
  - a set O of typed output channels,
  The channels form attributes in assertions.

- an interface assertion is a logical formula with the channel identifiers in I and O as free logical variables denoting streams of the respective types.
Example: Component interface specification

A transmission component TMC

Spec name

TMC

Input channel

\[
\text{in} \quad x : T \\
\text{out} \quad y : T \\
x \sim y
\]

Output channel

\[
x \sim y \equiv (\forall m \in T : \{m\} \odot x = \{m\} \odot y)
\]

Interface assertion

x : T \quad y : T

TMC \quad x \sim y
Can we give a purely logical specification of architecture?

Architectures
Composition and Decomposition of Systems

\[ F_1 \in [I_1 \rightarrow O_1] \]
\[ F_2 \in [I_2 \rightarrow O_2] \]
\[ C_1 = O_1 \cap I_2 \]
\[ C_2 = O_2 \cap I_1 \]
\[ I = I_1 \setminus C_2 \cup I_2 \setminus C_1 \]
\[ O = O_1 \setminus C_1 \cup O_2 \setminus C_2 \]

\[ F_1 \otimes F_2 \in [I \rightarrow O], \]

\[ (F_1 \otimes F_2).x = \{ z | O : x = z | I \land z | O_1 \in F_1(z | I_1) \land z | O_2 \in F_2(z | I_2) \} \]
Interface specification composition rule

F1
\[ \text{in } x_1, z_{21}: T \]
\[ \text{out } y_1, z_{12}: T \]
\[ P_1 \]

F2
\[ \text{in } x_2, z_{12}: T \]
\[ \text{out } y_2, z_{21}: T \]
\[ P_2 \]

F_1 \otimes F_2
\[ \text{in } x_1, x_2: T \]
\[ \text{out } y_1, y_2: T \]
\[ \exists z_{12}, z_{21}: P_1 \land P_2 \]
Specifying Architectures

Given composable systems $k \in K$ with specifying interface assertions $C_k$ the specification of the architecture reads

$$\wedge \{C_k : k \in K\}$$

and the interface assertion of the composed is given by hiding the internal channels in $Z$

$$\exists Z: \wedge \{C_k : k \in K\}$$

Syntactic Architecture
Traceability in Software and System Development
Linking, Tracing, and Relating Artifacts

• A link $t$ is a directed relation between two content chunks $e$ and $e'$ of artifacts $E$ and $E'$.
  ◊ $e$ is called the source of $t$ and
  ◊ $e'$ is called the target of $t$.

• We write

$$\text{src}(t) = e \text{ and } \text{trg}(t) = e'$$
Linking, Tracing: Relating Artifacts and their Content Chunks

- A trace is a nonempty finite sequence of links $t_0, t_1, t_2, \ldots, t_n$ where the source of $t_{i+1}$ is the target of $t_i$: 
  $$\text{trg}(t_i) = \text{src}(t_{i+1}) \quad \text{for } i = 0, 1, \ldots, n-1$$

We distinguish between links and traces that
- relate the content elements of an artifact, called *intra-artifact links*, and links that
- relate the content elements $e$ of one artifact $E$ to those of a different artifact $E'$, called *inter-artifact links*.
Meaning of Links and Traces

• A link has a meaning.
• A link states a proposition about the relationship between its source and its target.
• A link can be valid or invalid.
  ◇ It is called valid, if the proposition associated with the link is true. Otherwise it is called invalid.
Example: Linking

List of stakeholders

... Product manager
   Architect
   ...

List of requirements

R1: ...
...
Rk: High usability
...

IFIP W.G. 2.3 Seattle July 2012
Example: Linking

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_i ): ...</td>
<td>( S_i ): ...</td>
</tr>
<tr>
<td>( R_k ): Thrust reversal can only be activated, if airplane is on the ground</td>
<td>( S_j ): Sensor AoG yields true if and only if airplane is on the ground</td>
</tr>
<tr>
<td>...</td>
<td>( S_{j+1} ): ( \neg \text{AoG} \Rightarrow \text{Thrust reversal cannot be activated} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Syntax and Meaning of Links

• Syntactically a link is
  ◊ a relationship between content chunks of artifacts.

• Semantically a link expresses that
  ◊ there is a particular relation between two content chunks.

Example: link t with

\[
\begin{align*}
\text{trg}(t) &= \text{"Product\_Manager: Stakeholder"} \\
\text{src}(t) &= \text{"High\_Usability: Quality\_Attribute"}.
\end{align*}
\]

• Link is to express that the stakeholder Product\_Manager is the source of the quality requirement High\_Usability.

• In other terms, the link has the meaning
  ◊ \((\text{Product\_Manager: Stakeholder, High\_Usability: Quality\_Attribute})\in \text{Source\_of\_Requirement}\)
  ◊ where Source\_of\_Requirement is a relation between stakeholders and requirements.
Formalizing Links and Tracing

We distinguish the following concepts of links

• *supplemental* links: link \( t \) relating \( a \) and \( z \) documents relationships between content chunks \( a \) and \( z \) providing additional information not explicitly contained in artifacts \( E_k \) and \( E_{k'} \);

  ◊ Example: link between a stakeholder \( a \) and a requirement \( z \) that originates from that stakeholder.

• *derived* links: link \( t \) relating \( a \) and \( z \) documents relationships between chunks \( a \) and \( z \) that can be derived from its logical meaning and justified logically (or even proved) from the assertions in artifacts \( E_k \) and \( E_{k'} \);

  ◊ Example: specification of a functional property by assertion \( a \) and its implementation or refinement by assertion \( z \) such that \( z \Rightarrow a \).
Representing Artifacts by Logic: Requirements
System level functional requirements

- The system interface behaviour $F$ is specified by the system requirements specification

$$A = \{A_i: 1 \leq i \leq n\}$$

where the $A_i$ are interface assertions

<table>
<thead>
<tr>
<th></th>
<th>Functional</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>...</td>
<td>Yes</td>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_2$</td>
<td>...</td>
<td>No</td>
<td>medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_n$</td>
<td>...</td>
<td>no</td>
<td>low</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analyzing System Level Requirements: Intra-Artifact Links

- There are many papers and even standards on the quality of requirements.
- The IEEE standard Std 830-1998 (see [IEEE 98]) requires the following quality attributes for system and software requirements:
  ◊ completeness
  ◊ consistency
  ◊ unambiguousness/precision
  ◊ correctness (more precisely validity)
  ◊ understandability/clarity
  ◊ traceability
  ◊ changeability
A set $R$ of system assertions is called

- **consistent**, if the following proposition holds
  \[ \exists (\land R) \]

- **non-overlapping**, if (there is a relationship to case distinctions)
  \[ \forall (\lor R) \]

- **weakly independent**, if for every pair of non-empty subsets $R', R'' \subseteq R$ of disjoint non-empty sets of assertions with $Q = \land R', P = \land R''$
  \[ \exists (P \land Q) \] $P$ and $Q$ are consistent
  \[ \exists (\neg P \land Q) \] $Q$ does not imply $P$
  \[ \exists (P \land \neg Q) \] $P$ does not imply $Q$
Representing Artifacts by Logic: Functional Specification
Combining Functions

Given two functions $F_1$ and $F_2$ in isolation

We want to combine them into a function $F_1 \oplus F_2$
Combining Functions

Their isolated combination

F₁ ⊕ F₂

F₁

F₂

I₁

O₁

I₂

O₂
Combining Functions

If services $F_1$ and $F_2$ have feature interaction we get:

We explain the functional combination $F_1 \otimes F_2$ as a refinement step.
The steps of function combination

Given the isolated function $F_1$

We construct a refinement $F'_1$

And combine $F'_1$ with a refinement $F'_2$ of $F_2$
Functional view: functional decomposition

- The system interface behaviour $F$ as specified by the system requirements specification $A = \{A_i: 1 \leq i \leq n\}$ is structured
  - into a set of sub-interfaces for sub-functions $F_1, \ldots, F_k$
  - that are specified independently by introducing a number of mode channels to capture their feature interactions
  - each $F_i$ sub-function is described by
    - a syntactic interface and
    - an interface assertion $B_i$ such that
      \[ \land \{B_i: 1 \leq i \leq k\} \Rightarrow A \]
Three levels of Specification

• Requirements - system level
  ◊ List of requirements - functional system property
  ◊ Example: “The activation of safety relevant functions by the pilot is always double checked for plausibility by the system."

• Functional specification - system level
  ◊ decomposition of System functionality in hierarchy of (sub-)functions
  ◊ Specification of (sub-)functions
  ◊ Specification of dependencies (feature interactions) between (sub-)functions based on a mode concept
  ◊ Example: “Thrust reversal may only be activated, if the plane is on the ground."

• Architecture specification - component level
  ◊ decomposition a systems in sub-systems (component)
  ◊ relationship to data flow diagram
  ◊ interface specification of component
  ◊ Example: “The weight sensor indicates that the plane is on the ground."

Three levels of system description in logic

- system level requirements

\[ A = \land \{A_i: 1 \leq i \leq r\} \]

- functional specification at system level - functionality

\[ B = \land \{B_i: 1 \leq i \leq n\} \]

- architecture specification (given by component interface specifications)

\[ C = \land \{C_k: 1 \leq k \leq m\} \]

- correctness

  functional Specification correct in relation to requirements:

  \[ B \Rightarrow A \]

  architecture (let \( m_1, \ldots, m_n \) be mode channels):

  \[ C \Rightarrow \exists m_1, \ldots, m_j: B \]
Function Hierarchy
Representing Artifacts by Logic: Architecture
Architectural view: decomposition into components

- architecture specification (given by component interface specifications $C_i$)

$$C = \wedge \{C_i: 1 \leq i \leq k\}$$
Architecture

• Composition $C_1 \land C_2 \land C_3$
What is a function?

What is a sub-system?

Sub-function ≠ Sub-system
Relating Content Chunks of Artifacts by Logic: Semantics of Tracing
Guarantors and Guarantor Sets

• Let $P$ be an assertion and $R$ be a set of assertions.
• A subset $R' \subseteq R$ is called *guarantor set for* assertion $P$ in set $R$ if
  \[
  \forall((\land R') \Rightarrow P)
  \]
• In this case the assertions in set $R'$ guarantee logically assertion $P$.
• A guarantor set $R'$ for assertion $P$ is called *minimal*, if every strict subset of set $R'$ is not a guarantor set for assertion $P$.
• A minimal guarantor set $R' \subseteq R$ is called *unique* in set $R$ if there does not exist a different minimal guarantor set in $R$. 
Guarantors and Guarantor Sets

• A assertion $Q$ is called *weak guarantor* for assertion $P \in R$ if it occurs in some minimal guarantor set for assertion $P$ in $R$.
• A assertion $Q$ is called *strong guarantor* for $P$ in $R$ if assertion $Q$ occurs in every guarantor set of assertion $P$ in $R$.
• Note that there is some relationship between guarantors and the so-called Primimplikanten *a la* Quine
# Relational view: Tracing

<table>
<thead>
<tr>
<th>Function</th>
<th>Safety</th>
<th>Priority</th>
<th>Component</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>Yes</td>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₂</td>
<td>No</td>
<td>medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aₙ</td>
<td>no</td>
<td>low</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image_url)
### Relationship: req spec to function spec - tracing

<table>
<thead>
<tr>
<th>sub-function reqs</th>
<th>system level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A_1$</td>
</tr>
<tr>
<td>$B_1$</td>
<td></td>
</tr>
<tr>
<td>$B_2$</td>
<td></td>
</tr>
<tr>
<td>$B_3$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$B_n$</td>
<td></td>
</tr>
</tbody>
</table>

- **Red**: $B_i$ is strong guarantor of $A_j$
- **Yellow**: $B_i$ is weak guarantor of $A_j$
- **Green**: $B_i$ is not a weak guarantor of $A_j
**Relationship: architecture to requirements**

<table>
<thead>
<tr>
<th>sub-system reqs</th>
<th>system level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A₁</td>
</tr>
<tr>
<td>C₁</td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Cₙ</td>
<td></td>
</tr>
</tbody>
</table>

**Colors:**
- **Red:** $C_i$ is strong guarantor of $A_j$
- **Yellow:** $C_i$ is weak guarantor of $A_j$
- **Green:** $C_i$ is not a weak guarantor of $A_j$
Tracing at a logical level

Specification: \( A = A_1 \land A_2 \land A_3 \land \ldots \)

Architecture: \( C = C_1 \land C_2 \land C_3 \land \ldots \)

Correctness architecture: \( C \Rightarrow A \)

Tracing requirement \( k \): \( C \Rightarrow A_1 \land A_2 \land A_3 \land \ldots \land A_k \land \ldots \)

Expanding \( C \): \( C_1 \land C_2 \land C_3 \land \ldots \Rightarrow A_k \)

Weakening \( C \): \( (C_1 \Rightarrow C'_1) \land (C_2 \Rightarrow C'_2) \land (C_3 \Rightarrow C'_3) \land \ldots \)

Such that: \( C'_1 \land C'_2 \land C'_3 \land \ldots \Rightarrow A_k \)

Conclusion:

If the architecture spec \( C \) is correct with respect to a particular requirement \( A_k \) then there exists assertions \( C'_i \) contained in the specifications \( C_i \) the sub-systems of the architectures that guarantee
Analysis

• For every requirement $A_k$ its “guarantors” $C'_i$ are different, in general
  ◊ Syntactic tracing does not work
• For given requirement $A_k$ there are weakest “guarantors” $C'_i$
• For given requirement $A_k$ its weakest “guarantors” $C'_i$ are not necessarily unique
• For given requirement $A_k$ many of its “guarantors” $C'_i$ are not necessarily trivial ("true")
  ◊ Many links
• Generalisation:
  ◊ From $C_1 \land C_2 \land C_3 \land \ldots \Rightarrow A_1 \land A_2 \land A_3 \land \ldots$
  ◊ By weakening to $C'_1 \land C'_2 \land C'_3 \land \ldots \Rightarrow A'_1 \land A'_2 \land A'_3 \land \ldots$
Concluding Remarks

• Artifacts represented by logic
  ◊ Logical representation of the content by assertions

• Dependencies based on logic
  ◊ Logical representation of dependencies

• Formalization of traceability
  ◊ Intra- and inter-artifact links

• Relating different levels of abstraction

• Further questions
  ◊ How many dependencies in systems today
  ◊ What is the complexity of relations between
    • Requirements and functional specification
    • Functional specification and architecture
    • Requirements and architecture