Fault-based Test Case Generation for Component Connectors

Sun Meng

CWI, Amsterdam, The Netherlands
http://www.cwi.nl/∼sun

joint work with
Bernhard K. Aichernig (TU Graz)
Farhad Arbab, Lacramioara Astefanoaei, Frank S. de Boer, Jan Rutten (CWI)

CIC workshop, Braga, Protugal, May 7-8, 2009

Outline

Introduction

Connectors as Designs

Fault-based Test Case Generation

Conclusion

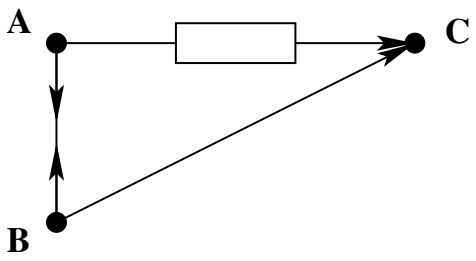
Problem

- The complexity and importance of coordination models in service-oriented applications necessarily lead to a higher relevance of testing issues for connectors during development of systems.
- Aim of testing: to show *conformance* or *non-conformance* of the final software system with some requirements or specifications.
- The behavior of connectors generally describes the manifold interactions among components / services rather than simple input-output behavior.
- we can use not simply sequences of input and output, but relations on different input / output sequences as test cases for connectors.

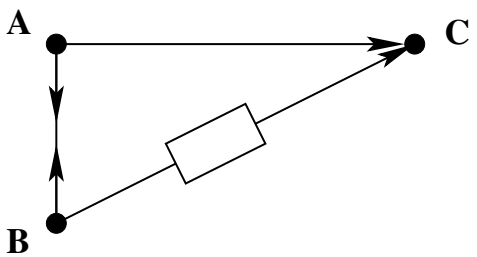
An Example

- For connectors, faults are caused by possible errors during the development process, such as wrongly used channels, missing or redundant subcircuits, or circuits with wrongly constructed topology, etc.

The intended connector:



The faulty connector:



- In the faulty connector, the topology is changed and the sequence of values that appear through **C** consist of zero or more repetitions of the pairs of values written to **B** and **A**, in a different order, comparing with the intended one.

What and How?

- The idea: to automatically generate a test case that can detect such errors and exclude faulty implementations of connectors.
- The specification and implementation of connectors are given by pairs of pre- and post-condition.
- Every connector is given by a pair of predicates $P \vdash Q$ where
  - the assumption P is what the designer can rely on when the communicating operation is initiated by inputs to the connectors, and
  - the commitment Q must be true for the outputs when the communicating operation terminates.
- Test cases will detect certain faults in connectors by using refinement calculus.

Timed Data Sequences

- For an arbitrary connector **R**, the relevant observations come in pairs, with one observation on the source nodes of **R**, and one observation on the sink nodes of **R**. For every node N, the corresponding observation on N is given by a timed data sequence.
- Let $TS = \{a \in \mathbb{R}_+^* \mid \forall 0 \leq n < |a|.a(n) < a(n+1)\}$ and $DS = D^*$ be the set of time sequences and data sequences, the set of timed data sequences is defined by $TDS \subseteq DS \times TS$ that contains pairs $\langle \alpha, a \rangle$ consisting of $\alpha \in DS$ and $a \in TS$ with $|\alpha| = |a|$.
- Timed data sequences can be alternatively and equivalently defined as (a subset of) $(D \times \mathbb{R}_+)^*$ because of the existence of isomorphism

$$\langle \alpha, a \rangle \mapsto (\langle \alpha(0), a(0) \rangle, \langle \alpha(1), a(1) \rangle, \langle \alpha(2), a(2) \rangle, \cdots)$$

Design

- For connectors, a design $P \vdash Q$ has the following meaning:

$$P \vdash Q =_{df} (ok \wedge P \Rightarrow ok' \wedge Q)$$

  where ok and ok' are two variables being used to analyze explicitly the phenomena of communication initialization and termination.
- The variable ok stands for a successful initialization and the start of a communication. When ok is **false**, the communication has not started, so no observation can be made.
- The variable ok' denotes the observation that the communication has terminated. The communication is divergent when ok' is **false**.

Connectors as Designs

- To specify input and output explicitly, for a connector **R**, we use $in_{\mathbf{R}}$ and $out_{\mathbf{R}}$ for the lists of timed data sequences on the input ends and output ends of **R** respectively.
- Every connector **R** can be represented as

$$\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$

  - $P(in_{\mathbf{R}})$ is the precondition that should be satisfied by inputs $in_{\mathbf{R}}$ on the source nodes of **R**, and
  - $Q(in_{\mathbf{R}}, out_{\mathbf{R}})$ is the postcondition that should be satisfied by outputs $out_{\mathbf{R}}$ on the sink nodes of **R**.
- A predicate $\mathcal{D}$ is used to denote well-defined timed data sequence types. In other words, we define the behavior only for valid sequences expressed via a predicate $\mathcal{D}$.

Some Examples

- Synchronous channel $\longrightarrow$:

$$\begin{aligned} \textbf{con}: &\quad \textbf{Sync}(in : (\langle \alpha, a \rangle); out : (\langle \beta, b \rangle)) \\ P: &\quad \mathcal{D}\langle \alpha, a \rangle \\ Q: &\quad \mathcal{D}\langle \beta, b \rangle \wedge \beta = \alpha \wedge b = a \end{aligned}$$

- FIFO1 channel $-\square\!\!\rightarrow$:

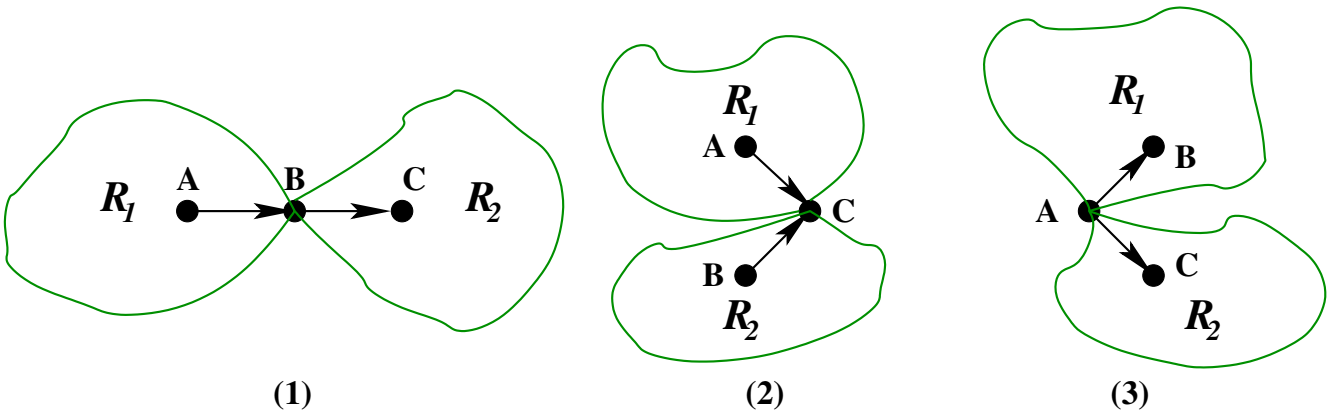
$$\begin{aligned} \textbf{con}: &\quad \textbf{FIFO1}(in : (\langle \alpha, a \rangle); out : (\langle \beta, b \rangle)) \\ P: &\quad \mathcal{D}\langle \alpha, a \rangle \\ Q: &\quad \mathcal{D}\langle \beta, b \rangle \wedge \beta = \alpha \wedge \\ &\quad\quad a < b \wedge (tail(b^R))^R < tail(a) \end{aligned}$$

- Synchronous drain $\rightarrow\!\!\leftarrow$:

$$\begin{aligned} \textbf{con}: &\quad \textbf{SyncDrain}(in : (\langle \alpha, a \rangle, \langle \beta, b \rangle); out : (\,)) \\ P: &\quad \mathcal{D}\langle \alpha, a \rangle \wedge \mathcal{D}\langle \beta, b \rangle \wedge a = b \\ Q: &\quad \textbf{true} \end{aligned}$$

Connector Composition

- Three Composition Patterns:



- We use

$$\mathbf{R}_i(in : in_{\mathbf{R}_i}; out : out_{\mathbf{R}_i})$$
$$P_i(in_{\mathbf{R}_i}) \vdash Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i})$$

where $i = 1, 2$ to denote the two connectors being composed.

Flow-through Composition

- For $\mathbf{R}_1$ and $\mathbf{R}_2$, suppose one sink node of $\mathbf{R}_1$ and one source node of $\mathbf{R}_2$ are joined together into a mixed node B. Such mixed nodes are hidden (encapsulated) by using existential quantification.
- If one sink node of $\mathbf{R}_1$ and one source node of $\mathbf{R}_2$ are joined together into a mixed node B, $\langle \beta_1, b_1 \rangle \in out_{\mathbf{R}_1}$ and $\langle \beta_2, b_2 \rangle \in in_{\mathbf{R}_2}$ are the timed data sequences on B in $\mathbf{R}_1$ and $\mathbf{R}_2$ respectively, for two predicates P and Q,

$$P(\langle \beta_1, b_1 \rangle); Q(\langle \beta_2, b_2 \rangle) = \exists \langle \beta, b \rangle . P(\langle \beta, b \rangle) \wedge Q(\langle \beta, b \rangle)$$

Flow-through Composition

- Let $\langle \beta_1, b_1 \rangle \in out_{\mathbf{R}_1}$ and $\langle \beta_2, b_2 \rangle \in in_{\mathbf{R}_2}$ be the timed data sequences on the node B in $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively. Then the resulting connector is

$$\mathbf{R}(in : (\bigcup_{i=1,2} in_{\mathbf{R}_i}) \setminus \{\langle \beta_2, b_2 \rangle\}; out : (\bigcup_{i=1,2} out_{\mathbf{R}_i}) \setminus \{\langle \beta_1, b_1 \rangle\})$$

$$P \vdash Q$$

where

$$P = P_1 \wedge \neg(Q_1(\langle \beta_1, b_1 \rangle); \neg P_2(\langle \beta_2, b_2 \rangle))$$
$$Q = Q_1(\langle \beta_1, b_1 \rangle); Q_2(\langle \beta_2, b_2 \rangle)$$

Merging Sink Nodes

- In (2), Let $\langle \gamma_i, c_i \rangle \in out_{\mathbf{R}_i}$, $i = 1, 2$, be the timed data sequences on the node C in $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively. Then the resulting connector is

$$\mathbf{R}(in : \bigcup_{i=1,2} in_{\mathbf{R}_i}; out : (\bigcup_{i=1,2} (out_{\mathbf{R}_i} \setminus \{\langle \gamma_i, c_i \rangle\})) \cup \{\langle \gamma, c \rangle\})$$

$$P \vdash Q$$

where

$$P = \bigwedge_{i=1,2} P_i(in_{\mathbf{R}_i})$$

$$Q : \quad \mathcal{D}\langle \gamma, c \rangle \wedge \exists \langle \gamma_1, c_1 \rangle, \langle \gamma_2, c_2 \rangle . (\bigwedge_{i=1,2} Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i})) \wedge$$

$$M(\langle \gamma_1, c_1 \rangle, \langle \gamma_2, c_2 \rangle, \langle \gamma, c \rangle)$$

Merging Sink Nodes

In the definition, the ternary relation M is defined as

$$
\begin{aligned}
& M(\langle \gamma_1, c_1 \rangle, \langle \gamma_2, c_2 \rangle, \langle \gamma, c \rangle) \\
= & \begin{cases}
\langle \gamma, c \rangle = \langle \gamma_1, c_1 \rangle & \text{if } |\langle \gamma_2, c_2 \rangle| = 0 \\
\langle \gamma, c \rangle = \langle \gamma_2, c_2 \rangle & \text{if } |\langle \gamma_1, c_1 \rangle| = 0 \\
c_1(0) \neq c_2(0) \wedge & \\
\quad \begin{cases}
\gamma(0) = \gamma_1(0) \wedge c(0) = c_1(0) \wedge & \\
M(\langle \gamma_1', c_1' \rangle, \langle \gamma_2, c_2 \rangle, \langle \gamma', c' \rangle) \text{ if } c_1(0) < c_2(0) & \\
\gamma(0) = \gamma_2(0) \wedge c(0) = c_2(0) \wedge & \\
M(\langle \gamma_1, c_1 \rangle, \langle \gamma_2', c_2' \rangle, \langle \gamma', c' \rangle) \text{ if } c_2(0) < c_1(0) &
\end{cases} \\
\text{otherwise} &
\end{cases}
\end{aligned}
$$

Merging Source Nodes

- In (3), Let $\langle \alpha_i, a_i \rangle \in in_{\mathbf{R}_i}$ $i = 1, 2$, be the timed data sequences on the node A in $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively. Then the resulting connector is

$$\mathbf{R}(in : (\bigcup_{i=1,2} in_{\mathbf{R}_i} \setminus \{\langle \alpha_i, a_i \rangle\}) \cup \{\langle \alpha, a \rangle\}; out : \bigcup_{i=1,2} out_{\mathbf{R}_i})$$

$$P \vdash Q$$

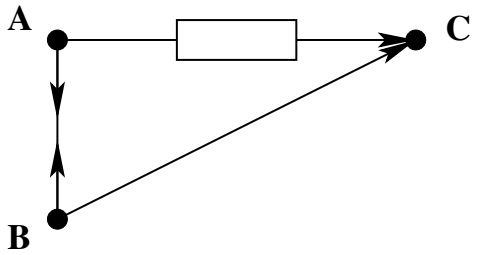
where

$$P = \bigwedge_{i=1,2} P_i(in_{\mathbf{R}_i})[\langle \alpha, a \rangle / \langle \alpha_i, a_i \rangle]$$

$$Q = \bigwedge_{i=1,2} Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i})[\langle \alpha, a \rangle / \langle \alpha_i, a_i \rangle]$$

For a predicate P, if v is a variable in P, $P[u/v]$ is the predicate obtained by replacing all occurrance of v in P by u.

Example



Order$(in : (\langle \alpha, a \rangle, \langle \beta, b \rangle); out : \langle \gamma, c \rangle)$

$P \vdash Q$

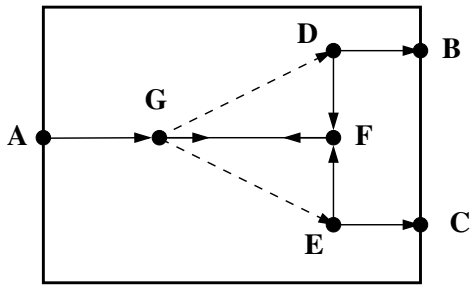
$P = \mathcal{D}\langle \alpha, a \rangle \wedge \mathcal{D}\langle \beta, b \rangle \wedge a = b$

$Q = \mathcal{D}\langle \gamma, c \rangle \wedge \exists \langle \gamma_1, c_1 \rangle, \langle \gamma_2, c_2 \rangle . \mathcal{D}\langle \gamma_1, c_1 \rangle \wedge \mathcal{D}\langle \gamma_2, c_2 \rangle \wedge$

$\gamma_1 = \alpha \wedge a < c_1 \wedge ((c_1^R)')^R < a' \wedge \gamma_2 = \beta \wedge c_2 = b \wedge$

$M(\langle \gamma_1, c_1 \rangle, \langle \gamma_2, c_2 \rangle, \langle \gamma, c \rangle)$

Example



EXRouter($in : (\langle \alpha, a \rangle); out : (\langle \beta, b \rangle, \langle \gamma, c \rangle)$)

$P \vdash Q$

$P = \mathcal{D}\langle \alpha, a \rangle$

$Q = \mathcal{D}\langle \beta, b \rangle \wedge \mathcal{D}\langle \gamma, c \rangle \wedge L(\langle \alpha, a \rangle, \langle \beta, b \rangle) \wedge L(\langle \alpha, a \rangle, \langle \gamma, c \rangle) \wedge$
$M(\langle \beta, b \rangle, \langle \gamma, c \rangle, \langle \alpha, a \rangle)$

where

$L(\langle \alpha, a \rangle, \langle \beta, b \rangle) \equiv \langle \beta, b \rangle = (\,) \vee$

$$\left(a(0) \leq b(0) \wedge \begin{cases} \alpha(0) = \beta(0) \wedge L(\langle \alpha', a' \rangle, \langle \beta', b' \rangle) & \text{if } a(0) = b(0) \\ L(\langle \alpha', a' \rangle, \langle \beta, b \rangle) & \text{if } a(0) < b(0) \end{cases} \right)$$

Refinement

- Implication of predicates establishes a refinement order over connectors. Thus, more concrete implementations imply more abstract specifications. For two connectors

$$\mathbf{R}_i(in : in_{\mathbf{R}_i}; out : out_{\mathbf{R}_i})$$
$$P_i(in_{\mathbf{R}_i}) \vdash Q_i(in_{\mathbf{R}_i}, out_{\mathbf{R}_i})$$

  where $i = 1, 2$, if $in_{\mathbf{R}_1} = in_{\mathbf{R}_2}$ and $out_{\mathbf{R}_1} = out_{\mathbf{R}_2}$, then

$$\mathbf{R_1} \sqsubseteq \mathbf{R_2} \ =_{df} \ (P_1 \Rightarrow P_2) \wedge (P_1 \wedge Q_2 \Rightarrow Q_1)$$

- In other words, preconditions on inputs of connectors are weakened under refinement, and postconditions on outputs of connectors are strengthened.

Faulty Connectors

Definition

Given an intended connector specification

$$\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$

and a connector implementation

$$\mathbf{R}'(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P'(in_{\mathbf{R}}) \vdash Q'(in_{\mathbf{R}}, out_{\mathbf{R}})$$

with the same input and output nodes as in $\mathbf{R}$, which may contain an error. $\mathbf{R}'$ is called a faulty connector if and only if $\mathbf{R} \not\sqsubseteq \mathbf{R}'$.

Test Cases

- For connectors, we consider test cases as specifications that define the expected list of timed data sequences on the output nodes for a given list of timed data sequences on the input nodes.

Definition (deterministic test case)

For a connector $\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$, let i be the input vector and o be the output vector, both lists of timed data sequences with the same lengths as $in_{\mathbf{R}}$ and $out_{\mathbf{R}}$ respectively. A deterministic test case for $\mathbf{R}$ is defined as

$$t_d(in_{\mathbf{R}}, out_{\mathbf{R}}) = in_{\mathbf{R}} = i \vdash out_{\mathbf{R}} = o$$

Test Cases

- Sometimes the behavior of a connector can be non-deterministic. In this case, we can generalize the notion of test case as follows:

Definition (test case)

For a connector $\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$, let i be the input vector and O be a possibly infinite set containing the expected output vector(s). Both i and any $o \in O$ are lists of timed data sequences with the same lengths as $in_{\mathbf{R}}$ and $out_{\mathbf{R}}$ respectively. A test case for $\mathbf{R}$ is defined as

$$t(in_{\mathbf{R}}, out_{\mathbf{R}}) = in_{\mathbf{R}} = i \vdash out_{\mathbf{R}} \in O$$

Test Cases, Specifications and Implementations

- Test cases, as well as connector specifications and implementations, can be specified by designs.
- An implementation that is correct with respect to its specification should refine its test cases.
- Test cases are abstractions of an implementation if and only if the implementation passes the test cases.
- Taking specifications into consideration, test cases should also be abstractions of a specification if they are properly derived from the specification.

Correctness of Test Cases

Definition

For a connector specification **S**, its implementation **R** and a test case t, which satisfy

$$t \sqsubseteq \mathbf{S} \sqsubseteq \mathbf{R}$$

- t is called a correct test case with respect to **S**.
- **R** passes the test case t and conforms to the specification **S**.

Fault-Adequate Test Cases

Definition

Let $t(in_{\mathbf{R}}, out_{\mathbf{R}})$ be a test case (which can be either deterministic or non-deterministic), $\mathbf{R}$ an expected connector, and $\mathbf{R}'$ its faulty implementation. Then t is a fault-adequate test case if and only if

$$t \sqsubseteq \mathbf{R} \wedge t \not\sqsubseteq \mathbf{R}'$$

A fault-adequate test case detects a fault in $\mathbf{R}'$. Alternatively we can say that the test case distinguishes $\mathbf{R}$ and $\mathbf{R}'$. All test cases that detect a certain fault form a fault-adequate equivalence class.

Test Case Generation

- A test case for finding errors in a faulty connector has to,
  - first, be a correct test case of the intended connector;
  - second, it must not be an abstraction of the faulty connector.

Test Case Generation Algorithm

Algorithm

Consider an intended connector

$$\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$

and its faulty implementation

$$\mathbf{R}'(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P'(in_{\mathbf{R}}) \vdash Q'(in_{\mathbf{R}}, out_{\mathbf{R}})$$

as inputs. A test case t is generated by the following steps:

Test Case Generation Algorithm

1. A test case t is searched by
   1. find a pair $\langle \widehat{i}, \widehat{o} \rangle$ as a solution of

$$P(\widehat{i}) \wedge Q'(\widehat{i}, \widehat{o}) \wedge \neg Q(\widehat{i}, \widehat{o})$$

   2. if it exists, then the test case $t(i, O)$ is generated by finding the maximal set O of output vectors, such that for all $o \in O$, $i = \widehat{i} \wedge P(i) \wedge Q(i, o)$ is satisfied.
2. If the previous step does not succeed, then look for a test case $t(i, O)$ with O the maximal set of output vectors, such that for all $o \in O$, $\neg P'(i) \wedge P(i) \wedge Q(i, o)$ is satisfied.

Correctness

Theorem (Correctness)

Given an intended connector

$$\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$

its faulty implementation

$$\mathbf{R}'(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P'(in_{\mathbf{R}}) \vdash Q'(in_{\mathbf{R}}, out_{\mathbf{R}})$$

and a test case $t(i, O)$ generated by the algorithm,

$$t(i, O) \sqsubseteq \mathbf{R}$$

Correctness

Proof: The proof is divided into two cases, corresponding to steps 1 and 2 of the algorithm, respectively.

1. Since O is the maximal set such that for all $o \in O$, $P(i) \wedge Q(i, o)$ is satisfied, we have

$$
\begin{aligned}
& t(i, O) \sqsubseteq \mathbf{R} \\
\equiv & \{ \text{ Definition of refinement and non-deterministic test cases } \} \\
& (in_{\mathbf{R}} = i \Rightarrow P(in_{\mathbf{R}})) \wedge (Q(in_{\mathbf{R}}, out_{\mathbf{R}}) \wedge in_{\mathbf{R}} = i \Rightarrow out_{\mathbf{R}} \in O) \\
\equiv & \{ O \text{ is the maximal set such that } P(i) \wedge Q(i, o) \text{ is satisfied} \\
& \text{for all } o \in O \} \\
& \mathbf{true} \wedge \mathbf{true} \\
\equiv & \mathbf{true}
\end{aligned}
$$

Correctness

2. Since O is the maximal set of output vectors o each of which satisfies $\neg P'(i) \wedge P(i) \wedge Q(i, o)$, it follows that $P(i) \wedge Q(i, o)$ is also satisfied for all $o \in O$ and O is still maximal. By the same style of reasoning as in the first case, we can derive that

$$t(i, O) \sqsubseteq \mathbf{R} \equiv \mathbf{true}$$

Fault Coverage

Theorem (Fault Coverage)

Given an intended connector

$$\mathbf{R}(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P(in_{\mathbf{R}}) \vdash Q(in_{\mathbf{R}}, out_{\mathbf{R}})$$

its faulty implementation

$$\mathbf{R}'(in : in_{\mathbf{R}}; out : out_{\mathbf{R}})$$
$$P'(in_{\mathbf{R}}) \vdash Q'(in_{\mathbf{R}}, out_{\mathbf{R}})$$

and a test case $t(i, O)$ generated by the algorithm,

$$t(i, O) \not\sqsubseteq \mathbf{R}'$$

Fault Coverage

Proof: This proof is also divided into the two cases of the algorithm.

1. When O is the maximal set such that for all $o \in O$, $i = \widehat{i} \wedge P(i) \wedge Q(i, o)$ is satisfied. The test case is generated only if there exists $(\widehat{i}, \widehat{o})$, such that $P(\widehat{i}) \wedge Q'(\widehat{i}, \widehat{o}) \wedge \neg Q(\widehat{i}, \widehat{o})$ is satisfied. Thus we have $i = \widehat{i}$, $\widehat{o} \notin O$ and

$$
\begin{aligned}
& t(i, O) \not\sqsubseteq \mathbf{R}' \\
\equiv & \{\text{Definition of refinement and non-deterministic test cases}\} \\
& \neg(in_{\mathbf{R}} = i \Rightarrow P'(in_{\mathbf{R}})) \vee \neg(in_{\mathbf{R}} = i \wedge Q'(in_{\mathbf{R}}, out_{\mathbf{R}}) \Rightarrow out_{\mathbf{R}} \in O) \\
\equiv & (in_{\mathbf{R}} = i \wedge \neg P'(in_{\mathbf{R}})) \vee \\
& \exists in_{\mathbf{R}}, out_{\mathbf{R}}.(in_{\mathbf{R}} = i \wedge Q'(in_{\mathbf{R}}, out_{\mathbf{R}}) \wedge out_{\mathbf{R}} \notin O) \\
\equiv & \{\text{let } in_{\mathbf{R}} = \widehat{i}, out_{\mathbf{R}} = \widehat{o}\} \\
& (in_{\mathbf{R}} = i \wedge \neg P'(in_{\mathbf{R}})) \vee \mathbf{true} \\
\equiv & \mathbf{true}
\end{aligned}
$$

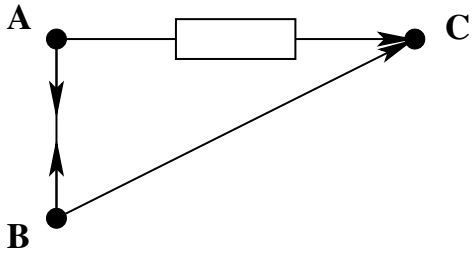
Fault Coverage

2. If O is the maximal set of output vectors o each of which satisfies $\neg P'(i) \wedge P(i) \wedge Q(i, o)$, then

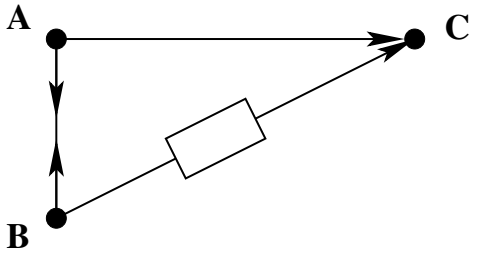
$$
\begin{aligned}
& t(i, O) \not\sqsubseteq \mathbf{R}' \\
\equiv & \{\text{Definition of refinement and non-deterministic test cases}\} \\
& \neg(in_{\mathbf{R}} = i \Rightarrow P'(in_{\mathbf{R}})) \vee \\
& \neg(in_{\mathbf{R}} = i \wedge Q'(in_{\mathbf{R}}, out_{\mathbf{R}}) \Rightarrow out_{\mathbf{R}} \in O) \\
\equiv & \exists in_{\mathbf{R}}.(in_{\mathbf{R}} = i \wedge \neg P'(in_{\mathbf{R}})) \vee \\
& \neg(in_{\mathbf{R}} = i \wedge Q'(in_{\mathbf{R}}, out_{\mathbf{R}}) \Rightarrow out_{\mathbf{R}} \in O) \\
\equiv & \{\text{let } in_{\mathbf{R}} = i\} \\
& \mathbf{true} \vee \neg(in_{\mathbf{R}} = i \wedge Q'(in_{\mathbf{R}}, out_{\mathbf{R}}) \Rightarrow out_{\mathbf{R}} \in O) \\
\equiv & \mathbf{true}
\end{aligned}
$$

Example

The intended connector:



The faulty connector:



- Generated test case: $((\langle \alpha, a \rangle, \langle \beta, b \rangle), \langle \gamma, c \rangle)$
- $\langle \alpha, a \rangle$ and $\langle \beta, b \rangle$ are two timed data sequences with length 1, while $\langle \gamma, c \rangle$ with length 2, and the constraint $\alpha \neq \beta$ should be satisfied.

Conclusion

- A semantic model of Reo connectors, which makes it possible to specify both finite and infinite behavior.
- Faults are modeled on the same level as the specification and implementation of connectors by using the notion of design.
- Test cases are generated based on a general theory of refinement for pre- and post-condition specifications, and has been instantiated in a prototype developed in Maude.

- more complex case studies,
- testing data selection based on constraint solving and its integration into the generated symbolic test cases, and
- specification-based testing of black-box connectors.

Conclusion

- A semantic model of Reo connectors, which makes it possible to specify both finite and infinite behavior.
- Faults are modeled on the same level as the specification and implementation of connectors by using the notion of design.
- Test cases are generated based on a general theory of refinement for pre- and post-condition specifications, and has been instantiated in a prototype developed in Maude.

- more complex case studies,
- testing data selection based on constraint solving and its integration into the generated symbolic test cases, and
- specification-based testing of black-box connectors.

Thank you!

Questions?