Definitions of Logical Causality for Log Analysis

Gregor Gössler¹
Joint work with Daniel Le Métayer¹ and Jean-Baptiste Raclet²

¹ INRIA Grenoble – Rhône-Alpes, France
² IRIT - CNRS, Toulouse, France

Syncron 2011
Objectives

General objective of the LISE project:

Provide a set of methods and tools (both legal and technical) to

- **Define liability** in a precise and unambiguous way
- **Establish liability** in case of failure

Scope:

- **Contractual** framework (not tort law)
- Liability for **software defects** (not intellectual property infringements)

Priority: settle liability issues in an amicable way.
A component-based system
⇝ components are provided by different vendors
Each component \( C_i \) is equipped with a contract \((A_i, G_i)\):
used according to \( A_i \), \( C_i \) promises to behave like \( G_i \).
Components are black boxes: only the contracts are known, not the implementation
⇝ implementations may violate their contract
Interactions between components are logged, logs may be distributed

**Problem:**
Define notions of causality between contract violations that can be used to establish liability of the component vendors.
Causality in distributed systems

Lamport causality \( \prec \) too weak for our needs:

\[ f \prec v \] does not mean that failure \( f \) causes the violation \( v \) of the specification of \( C \).

Lamport causality is a necessary but not sufficient condition for causality between contract violations.
Contracts

Contract $\mathcal{C} = \text{pair of automata } (\mathcal{A}, \mathcal{G})$.

$\mathcal{C}$ specifies under which **assumption** $\mathcal{A}$ the component provides **guarantee** $\mathcal{G}$.

$\Rightarrow$ clean specification and limitation of the responsibilities of components.

**Example (Contract satisfaction)**

$\mathcal{A}$: $a$ cannot reoccur before $b$

$\mathcal{G}$: $c$ never occurs

$tr$: $a$ $b$ $a$ $a$ $c$ $c$ $\not\models \mathcal{A}$ but $\models \mathcal{C} = (\mathcal{A}, \mathcal{G})$

$tr'$: $a$ $b$ $c$ $a$ $\models \mathcal{A}$ and $\not\models \mathcal{G}$ thus $\not\models \mathcal{C}$
Causality in Contract Violation: Overview

$C = (\mathcal{A}, \mathcal{G})$

$(\mathcal{A}_1, \mathcal{G}_1)$ $(\mathcal{A}_2, \mathcal{G}_2)$ $(\mathcal{A}_3, \mathcal{G}_3)$

$B_1 \xrightarrow{tr_1} B_2 \xrightarrow{tr_2} B_3 \xrightarrow{tr_3}$

Hypothesis

If the implementations $B_i$ of all components are correct, then $C$ is respected.

$\Rightarrow$ Any contract violation is due to some faulty implementation $B_i$. 
Causality in Contract Violation: Overview

\[ C = (A, G) \]

\( (A_1, G_1) \) \( (A_2, G_2) \) \( (A_3, G_3) \)

\[ B_1 \rightarrow B_2 \rightarrow B_3 \]

\( tr_1 \) \( tr_2 \) \( tr_3 \)

Hypothesis

If the implementations \( B_i \) of all components are correct, then \( C \) is respected. \( \Rightarrow \) Any contract violation is due to some faulty implementation \( B_i \).
Causality in Contract Violation: Overview

\[ C = (A, G) \]

\((A_1, G_1)\) \(\rightarrow\) \((A_2, G_2)\) \(\rightarrow\) \((A_3, G_3)\)

\(B_1\) \(\rightarrow\) \(B_2\) \(\rightarrow\) \(B_3\)

\(tr_1\) \(\rightarrow\) \(tr_2\) \(\rightarrow\) \(tr_3\)

Hypothesis

If the implementations \(B_i\) of all components are correct, then \(C\) is respected. \(\Rightarrow\) Any contract violation is due to some faulty implementation \(B_i\).
Causality in Contract Violation: Overview

\[ C = (\mathcal{A}, \mathcal{G}) \]

If the implementations \( B_i \) of all components are correct, then \( C \) is respected. \( \Rightarrow \) Any contract violation is due to some faulty implementation \( B_i \).

\[ (A_1, G_1) \quad (A_2, G_2) \quad (A_3, G_3) \]

\[ B_1 \quad B_2 \quad B_3 \]

\[ tr_1 \quad tr_2 \quad tr_3 \]
Causality in Contract Violation: Overview

$$C = (A, G)$$

$$\neg (A_1, G_1) \quad \neg (A_2, G_2) \quad \neg (A_3, G_3)$$

If the implementations $$B_i$$ of all components are correct, then $$C$$ is respected. ⇒ Any contract violation is due to some faulty implementation $$B_i$$. 

GG, DLM, and JBR (INRIA/IRIT)
Causality in Contract Violation: Overview

Hypothesis

If the implementations $B_i$ of all components are correct, then $C$ is respected.
Causality in Contract Violation: Overview

Hypothesis

If the implementations $B_i$ of all components are correct, then $C$ is respected.

⇒ Any contract violation is due to some faulty implementation $B_i$. 
Definition (Necessary causality)

\[ Tr \not\models^n C \text{ if } \]

\[ \exists \ tr \quad \models C \]
Logical Causality from Component Trace to Failure

Necessary Causality

**Definition (Necessary causality)**

\[ Tr \uparrow^n C \text{ if} \]

\[ tr_1 \quad \quad \quad \quad \quad \quad \quad \]

\[ Tr \]

\[ tr_n \quad \quad \quad \quad \quad \quad \quad \]

\[ tr \]
**Definition (**Necessary** causality)**

\[ Tr \models_n C \text{ if } \]

\[ \forall \text{ consistent } tr' \models C \]

\[ tr_1 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \]

\[ Tr \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \models C_k \]

\[ tr_n \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \]

GG, DLM, and JBR (INRIA/IRIT)
Logical Causality from Component Trace to Failure

Necessary Causality

Given:

- \((tr_1, ..., tr_n)\) vector of observed traces
- \(Tr \subseteq \{tr_1, ..., tr_n\}\) set of traces to be analyzed jointly

**Definition (Necessary causality)**

\(Tr\) is a necessary cause of the violation of \(C\) if \(\exists tr \in Tr: tr \uparrow C\) and \(\forall tr'\):

\[
\left( \forall j \in \{1, ..., n\} \setminus I : \pi_j(tr') = tr_j \land \right.

\[
\left. \forall k \in I : \pi_k(tr') \models C_k \right) \implies \ tr' \models C
\]

where \(I = \{i \mid tr_i \in Tr \land tr_i \not\models C_i\}\).
Definition (Sufficient causality)

\[ \forall r \not\rightarrow s C \quad \text{if} \]

\[ \exists t r_1 \]

\[ T r \not\rightarrow C_k \]

\[ t r_n \]

\[ \exists t r \not\rightarrow C \]
Logical Causality from Component Trace to Failure

Sufficient Causality

Definition (**Sufficient** causality)

$$Tr /\sim^s C \quad \text{if}$$

- $$tr_1$$
- $$Tr \sim \sim \sim \sim$$
- $$tr_n$$
- $$tr$$
Definition (Sufficient causality)

\[ \text{Tr} \not\rightarrow^s \text{C} \quad \text{if} \]

\[ \forall \text{consistent } \text{tr}' \]

\[ \text{tr}_1 \quad \text{tr}_n \quad \text{Tr} \]

\[ \models C_1 \quad \models C_n \quad \nexists C \]

\[ \text{GG, DLM, and JBR (INRIA/IRIT)} \]
Properties

**Property (Soundness)**

*Necessary and sufficient causality are sound:*

1. *Any (necessary or sufficient) cause contains at least one component trace violating its contract.*
2. *Any minimal set of traces forming a cause only contains traces violating the component contracts.*

**Property (Completeness)**

*Every violation of the system-level contract has a necessary and a sufficient cause.*

**Remark**

*Causality defined on contracts and observed traces, not implementations.*
Example 1: Adaptive Cruise Control

Sensor

SLD

HMI

Switch

Radar

Clock

OR

ACC

BS

TS

\[ s_{sr}^o \quad s_{ld}^i \quad s_{ld}^o \quad tck \quad h_{mi}^o,_{on} \quad h_{mi}^o,_{off} \quad s_{wi}^o,_{on} \quad s_{wi}^o,_{off} \quad s_{wo}^o,_{on} \quad s_{wo}^o,_{off} \quad a_{ci}^o,_{on} \quad a_{ci}^o,_{off} \quad a_{co}^t \quad t_{si},_{auto} \quad t_{si},_{user} \quad b_{si},_{user} \]

Obstacle recognition (OR) \( \Rightarrow \) \( G \): “output 1 time unit after sensing”

Adaptive Cruise Control (ACC) \( \Rightarrow \) \( G \): “output 1 time unit after latest input”

Global guarantee \( \Rightarrow \) \( G \): “ACC output at most 3 time units after data acquisition”
Example 1: Adaptive Cruise Control

- Obstacle recognition (OR)
  \( \leadsto \mathcal{G}_{OR} \): “output 1 time unit after sensing”

- Adaptive Cruise Control (ACC)
  \( \leadsto \mathcal{G}_{ACC} \): “output 1 time unit after latest input”

- Global guarantee
  \( \leadsto \mathcal{G} \): “ACC output at most 3 time units after data acquisition”
Example 1: Adaptive Cruise Control

Two necessary causes

Consider the following trace excerpts:

OR: \( \ldots \) \text{or}_i, \text{tck}, \text{tck}, \text{or}_o, \text{tck}, \text{tck}, \ldots \)

ACC: \( \ldots \) \text{tck}, \text{tck}, \text{acc}^s, \text{tck}, \text{tck}, \text{acc}^b, \ldots \)

Both OR and ACC violate their contracts (\( \Delta_{OR} = 2, \Delta_{ACC} = 2 \))
\[ \rightarrow \] violation of the global timing constraint (\( \Delta = 4 > 3 \)).

- Each of the OR and ACC failures is a \textbf{necessary} cause for the global failure.
- Taken together they are a \textbf{sufficient} cause.
Example 1: Adaptive Cruise Control

One necessary and sufficient cause

Consider the following trace excerpts:

OR: ... \( or_i, \ tck, \ tck, \ tck, \ or_o, \ tck, \ tck, \ ... \)

ACC: ... \( tck, \ tck, \ tck, \ acc^s, \ tck, \ tck, \ acc^t, \ ... \)

Both OR and ACC violate their contracts but OR’s violation is more serious (\( \Delta_{OR} = 3, \Delta_{ACC} = 2 \)).

- OR’s violation is a \textbf{necessary and sufficient} cause for the global failure.
- The violation of ACC is no longer a necessary cause.
Example 2: Travel Agency

Travel agency:

Hotel 1:
Example 2: Travel Agency

Spec 1: “at any time, #(debits) ≤ #(confirmations)”

Spec 2: “each request is ack’ed by either fail or resa; . !resp_yes; for \( i \in \{1, 2\} \)”
Example 2: Travel Agency

Spec 1: “at any time, #(debits) ≤ #(confirmations)”

Spec 2: “each request is ack’ed by either fail or resa; . !resp_yes; for \( i \in \{1, 2\} \)”

Observed traces:

agency: ?proc . !demand_1 . ?resp_no_1 . !demand_2 . ?resp_yes_2 . !conf
Example 2: Travel Agency

Spec 1: “at any time, #(debits) ≤ #(confirmations)”

Spec 2: “each request is ack’ed by either fail or resa_i . !resp_yes_i for \( i \in \{1, 2\} \)”

Observed traces:

agency: ?proc . !demand_1 . ?resp_no_1 . !demand_2 . ?resp_yes_2 . !conf

hotel 1: ?demand_1 . resa_1 . !resp_no_1 . wait_1 . debit_1
Example 2: Travel Agency

Spec 1: “at any time, \#(debits) \leq \#(confirmations)”

Spec 2: “each request is ack’ed by either fail or resa\_i \cdot !resp\_yes\_i; for \(i \in \{1, 2\}\)”

Observed traces:

hotel 1: ?demand\_1 . resa\_1 . !resp\_no\_1 . wait\_1 . debit\_1
hotel 2: ?demand\_2 . !resp\_yes\_2 . wait\_2 . debit\_2

Results of causality analysis:

spec 1  spec 2  travel agency  –  –  hotel 1  N, S  S  hotel 2  –  S

GG, DLM, and JBR (INRIA/IRIT)
Example 2: Travel Agency

Spec 1: “at any time, #(debits) ≤ #(confirmations)”

Spec 2: “each request is ack’ed by either fail or resa; . !resp_yes; for i ∈ {1, 2}”

Observed traces:

agency: ?proc . !demand_1 . ?resp_no_1 . !demand_2 . ?resp_yes_2 . !conf

hotel 1: ?demand_1 . resa_1 . !resp_no_1 . wait_1 . debit_1
hotel 2: ?demand_2 . !resp_yes_2 . wait_2 . debit_2

Results of causality analysis:

<table>
<thead>
<tr>
<th></th>
<th>spec 1</th>
<th>spec 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>travel agency</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>hotel 1</td>
<td>N, S</td>
<td>S</td>
</tr>
<tr>
<td>hotel 2</td>
<td>–</td>
<td>S</td>
</tr>
</tbody>
</table>
Causality Analysis with Bounded Past

Given:
- \((tr_1, ..., tr_n)\) vector of observed traces
- \(tr'_i\) a suffix of \(tr_i\), \(i = 1, ..., n\), such that \(\exists tr \forall i: \pi_i(tr) = tr'_i\).
- \(Tr \subseteq \{tr_1, ..., tr_n\}\) set of traces to be analyzed jointly

**Definition (Necessary causality)**

\(Tr\) is a necessary cause of the violation of \(C\) if \(\exists tr \in Tr: tr \not\nRightarrow C\) and \(\forall tr':\)

\[
\left(\forall j \in \{1, ..., n\} \setminus I': \pi_j(tr') = tr_j \land \right. \\
\left. \forall k \in I': \pi_k(tr') \models C_k \right) \implies tr' \models C
\]

where \(I = \{i \mid tr_i \in Tr \land tr_i \not\models C_i\}\).
Causality Analysis with Bounded Past

Given:

- \((tr_1, \ldots, tr_n)\) vector of observed traces
- \(tr'_i\) a suffix of \(tr_i\), \(i = 1, \ldots, n\), such that \(\exists tr \forall i : \pi_i(tr) = tr'_i\).
- \(Tr \subseteq \{tr_1, \ldots, tr_n\}\) set of traces to be analyzed jointly

**Definition (Necessary causality)**

\(Tr\) is a necessary cause of the violation of \(C\) if \(\exists tr \in Tr: \quad tr \not\rightarrow C\) and \(\forall tr'\):

\[
\left( \forall j \in \{1, \ldots, n\} \setminus I : \pi_j(tr') = tr_j \right) \land \\
\forall k \in I : \pi_k(tr') \models C_k \implies tr' \models C
\]

where \(I = \{i \mid tr_i \in Tr \land tr_i \not\models C_i\}\).
Causality Analysis with Bounded Past

Given:
- \((tr_1, ..., tr_n)\) vector of observed traces
- \(tr'_i\) a suffix of \(tr_i\), \(i = 1, ..., n\), such that \(\exists tr \forall i : \pi_i(tr) = tr'_i\).
- \(Tr \subseteq \{tr_1, ..., tr_n\}\) set of traces to be analyzed jointly

**Definition (Necessary causality)**

\(Tr\) is a necessary cause of the violation of \(C\) if \(\exists tr \in Tr: \quad \overrightarrow{tr} \uparrow C = \) and \(\forall tr'\):

\[
\left( \forall j \in \{1, ..., n\} \setminus I : \pi_j(tr') = tr_j \right) \quad + \quad \land \\
\forall k \in I : \pi_k(tr') \models C_k \quad \Rightarrow \quad tr' \models C
\]

where \(I = \{i \mid tr_i \in Tr \land \quad tr_i \not\models C_i \} = \).
Related Work

- **Actual causality** (Halpern & Pearl)
  - for Boolean expressions, no “native” support for sequential behavior
  - weak notion of logical causality

- **Dependability:**
  - fault trees: from failure to potential causes
  - FME(C)A: from cause to potential failures

- **Blaming** in contract languages
  - verify satisfaction of assumption and guarantee;
    - no notion of causality, no concurrency.

- **Diagnosis:** determine (unobservable) faults from observations
  - no notion of logical causality.
Contributions:

- General definitions for logical causality, supporting **group causality**
  - **(vertical) causality**: a component causes the violation of a system-level contract.
  - **horizontal causality**: a component causes the violation of the guarantee provided by another component.

- Effective **decision procedure**.

- Causality analysis on **bounded past**.

- Implementation in analysis tool **Loca**.
Future Work

- Generalize framework, instantiate with existing models of computation and communication: synchronous, timed automata, ...
- Allow for uncertainty, e.g., partial observability of events.
- Generalize to a quantitative notion of causality.
- Constructiveness?