Definitions of Logical Causality for Log Analysis

Gregor Gössler\(^1\)

Joint work with Daniel Le Métayer\(^1\) and Jean-Baptiste Raclet\(^2\)

\(^1\)INRIA Grenoble – Rhône-Alpes, France
\(^2\)IRIT - CNRS, Toulouse, France

Synchron 2011
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
\( \Rightarrow \) 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
\( \Rightarrow \) 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.
Lamport causality $< \text{too weak}$ for our needs:

$f < v$ 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 $C = \text{pair of automata } (A, G)$. $C$ specifies under which assumption $A$ the component provides guarantee $G$.  

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

Example (Contract satisfaction)

$A$: $a$ cannot reoccur before $b$  

$G$: $c$ never occurs

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

$tr'$: $a \ b \ c \ a \ \models A$ and $\not\models G$ thus $\not\models C$
Hypothesis

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

$$C = (A, G)$$

Any contract violation is due to some faulty implementation $B_i$. 

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

Hypothesis

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

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

\[ C = (A, G) \]

Logical Causality

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

Hypothesis

If the implementations \(B_i\) of all components are correct, then \(C\) is respected.

⇒ Any contract violation is due to some faulty implementation \(B_i\).
Hypothesis

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

⇒ Any contract violation is due to some faulty implementation $B_i$. 

$C = (A, G)$

$(A_1, G_1)$ $(A_2, G_2)$ $(A_3, G_3)$
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.

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

\( Tr \%^n C \) if

\[ tr_1 \quad \cdots \quad tr_n \]

\[ Tr \not\models C_k \]

\[ \exists tr \quad \cdots \quad \not\models C \]
Definition (**Necessary** causality)

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

\[ tr_1 \rightarrow \cdots \rightarrow tr_n \]

**Tr**

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

\[ T r \%^{n} C \text{ if} \]

\[ \forall \text{ consistent } t r' \]

\[ t r_1 \]

\[ \underbrace{} \]

\[ T r \]

\[ \underbrace{} \]

\[ \vdash C_k \]

\[ t r_n \]

\[ \underbrace{} \]

\[ \vdash C \]
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 \(\exists tr \in Tr: tr \% C\) and \(\forall tr':\)

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

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

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

\[ Tr \%^s C \quad \text{if} \]

\[ \exists \ tr \quad \text{tr}_1 \quad \text{tr}_n \quad Tr \quad \not\models C_k \quad \not\models C \]
Definition (**Sufficient** causality)

\[ Tr \%^s C \quad \text{if} \]

\[ tr_1 \]

\[ Tr \quad \]

\[ tr_n \]

\[ tr \]
Definition (Sufficient causality)

$Tr \%^s C$ if

$tr_1 \vdash C_1$

$Tr$

$tr_n \vdash C_n$

$\forall$ consistent $tr'$ $\not\vdash C$
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

Obstacle recognition (OR) ⇝ G
OR: "output 1 time unit after sensing"

Adaptive Cruise Control (ACC) ⇝ G
ACC: "output 1 time unit after latest input"

Global guarantee ⇝ G:
"ACC output at most 3 time units after data acquisition"
Example 1: Adaptive Cruise Control

- Obstacle recognition (OR)
  \[ \Rightarrow G_{OR}: \text{“output 1 time unit after sensing”} \]

- Adaptive Cruise Control (ACC)
  \[ \Rightarrow G_{ACC}: \text{“output 1 time unit after latest input”} \]

- Global guarantee
  \[ \Rightarrow G: \text{“ACC output at most 3 time units after data acquisition”} \]
Example 1: Adaptive Cruise Control

Two necessary causes

Consider the following trace excerpts:

**OR:** \( \ldots or_i, \ tck, \ tck, \ or_o, \ tck, \ tck, \ \ldots \)

**ACC:** \( \ldots \ tck, \ tck, \ acc_i^s, \ tck, \ tck, \ acc_o^b, \ \ldots \)

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

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

One necessary and sufficient cause

Consider the following trace excerpts:

**OR:** \[\ldots \; or_i, \; tck, \; tck, \; tck, \; or_o, \; tck, \; tck, \; \ldots\]

**ACC:** \[\ldots \; tck, \; tck, \; tck, \; acc_i, \; tck, \; tck, \; acc_o, \; \ldots\]

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 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_i . !resp_no; . !resp_yes; for i ∈ {1, 2}”
Example 2: Travel Agency

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

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

Observed traces:

agency: ?proc . !demand\(_1\) . ?resp\(_\text{no}\)\(_1\) . !demand\(_2\) . ?resp\(_\text{yes}\)\(_2\) . !conf
Example 2: Travel Agency

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

Spec 2: “each request is ack’ed by either fail or resa; . !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
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 ∈ {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
Example 2: Travel Agency

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

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

Observed traces:

agency: ?proc . !demand₁ . ?resp_no₁ . !demand₂ . ?resp_yes₂ . !conf

hotel 1: ?demand₁ . resa₁ . !resp_no₁ . wait₁ . debit₁

hotel 2: ?demand₂ . !resp_yes₂ . wait₂ . debit₂

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 \(\exists tr \in Tr: tr \not\% C\) 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 \(I' = \{i \mid tr_i \in Tr \land tr_i \not\models C_i\}\).
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 \(\exists tr \in Tr: tr \not\models C\) and \(\forall tr':\)

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

where \(I = \{i \mid tr_i \in Tr \land tr_i \not\models C_i\}\).
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 \(\exists tr \in Tr: tr \% C = \) 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 \(I = \{i \mid tr_i \in Tr \land \neg tr_i \models C_i\}\).
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?