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.

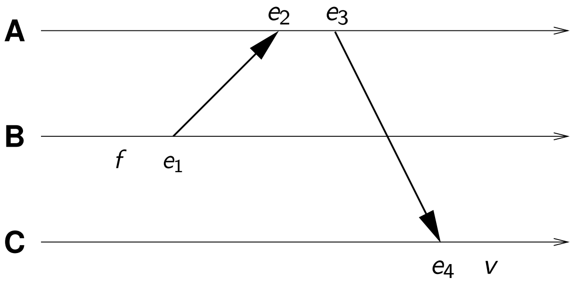
Context

- A component-based system
  $\rightsquigarrow$ components are provided by different vendors
- Each component C_i is equipped with a **contract** $(\mathcal{A}_i, \mathcal{G}_i)$:
  used according to $\mathcal{A}_i$, C_i promises to behave like $\mathcal{G}_i$.
- Components are **black boxes**: only the contracts are known, not the implementation
  $\rightsquigarrow$ 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

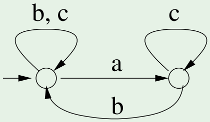
Contract $\mathcal{C}$ = pair of automata $(\mathcal{A}, \mathcal{G})$.

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

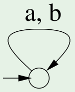
$\Rightarrow$ clean specification and limitation of the responsibilities of components.

Example (Contract satisfaction)

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

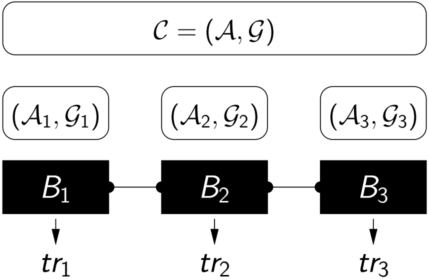


$\mathcal{G}$: c never occurs

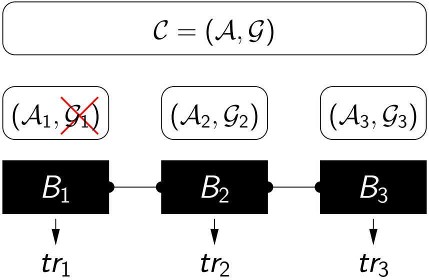


tr:	a	b	a	a	c	c	$\not\models \mathcal{A}$ but $\models \mathcal{C} = (\mathcal{A}, \mathcal{G})$
tr':	a	b	c	a			$\models \mathcal{A}$ and $\not\models \mathcal{G}$ thus $\not\models \mathcal{C}$

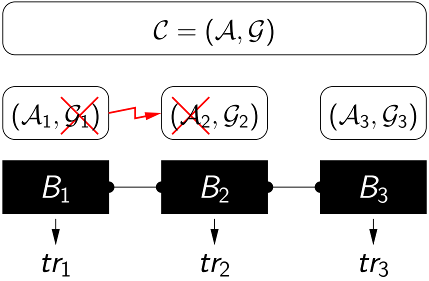
Causality in Contract Violation: Overview



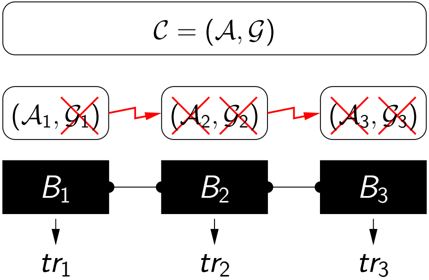
Causality in Contract Violation: Overview



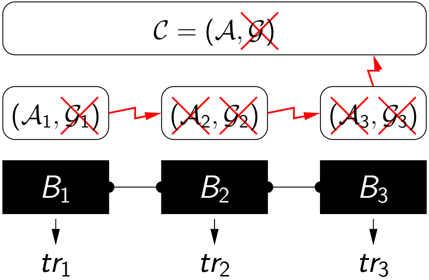
Causality in Contract Violation: Overview



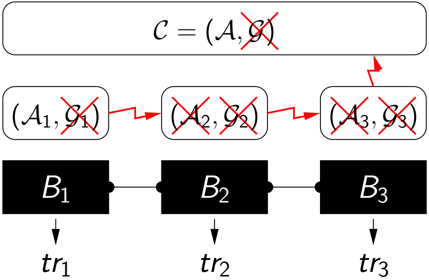
Causality in Contract Violation: Overview



Causality in Contract Violation: Overview



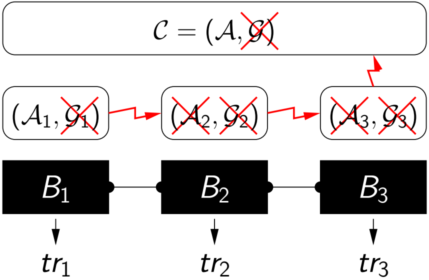
Causality in Contract Violation: Overview



Hypothesis

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

Causality in Contract Violation: Overview



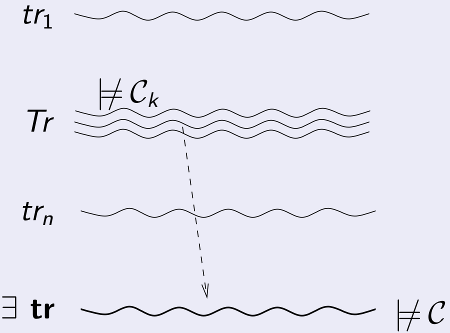
Hypothesis

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

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

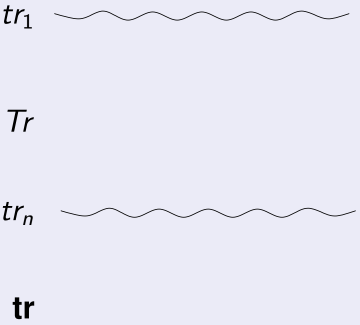
Definition (**Necessary** causality)

$Tr \nearrow^{\mathbf{n}} \mathcal{C}$ if



Definition (**Necessary** causality)

$Tr \nearrow^{\mathbf{n}} \mathcal{C}$ if

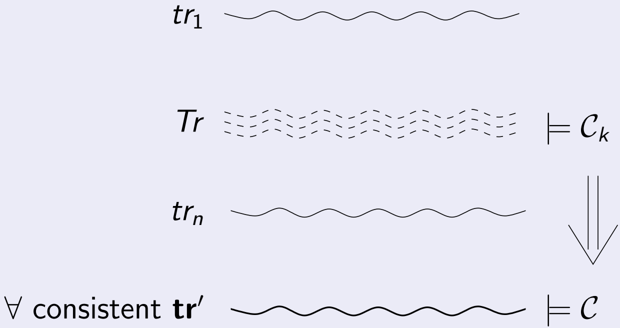


Logical Causality from Component Trace to Failure

Necessary Causality

Definition (Necessary causality)

$Tr \nearrow^{\mathbf{n}} \mathcal{C}$ if



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)

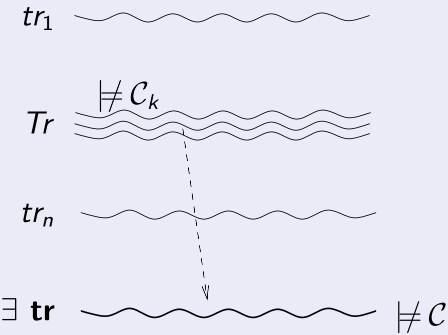
Tr is a necessary cause of the violation of $\mathcal{C}$ if $\exists tr \in Tr$: $tr \nearrow \mathcal{C}$ and $\forall \mathbf{tr}'$:

$$\Big(\forall j \in \{1, ..., n\} \setminus \mathcal{I} : \pi_j(\mathbf{tr}') = tr_j \wedge \forall k \in \mathcal{I} : \pi_k(\mathbf{tr}') \models \mathcal{C}_k\Big) \implies \mathbf{tr}' \models \mathcal{C}$$

where $\mathcal{I} = \{i \mid tr_i \in Tr \wedge tr_i \not\models \mathcal{C}_i\}$.

Definition (Sufficient causality)

$Tr \nearrow^{\mathbf{s}} \mathcal{C}$ if



Definition (Sufficient causality)

$Tr \nearrow^{\mathbf{s}} \mathcal{C}$ if

tr_1

Tr 

tr_n

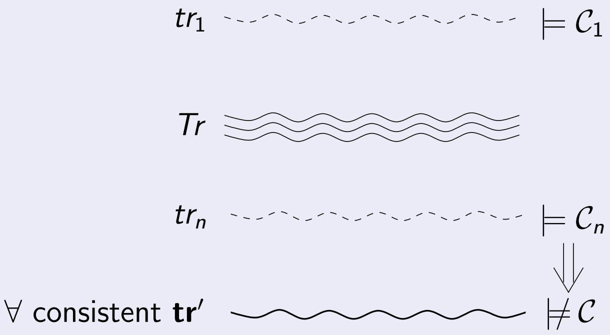
$\mathbf{tr}$

Logical Causality from Component Trace to Failure

Sufficient Causality

Definition (Sufficient causality)

$Tr \nearrow^{\mathbf{s}} \mathcal{C}$ if



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.*

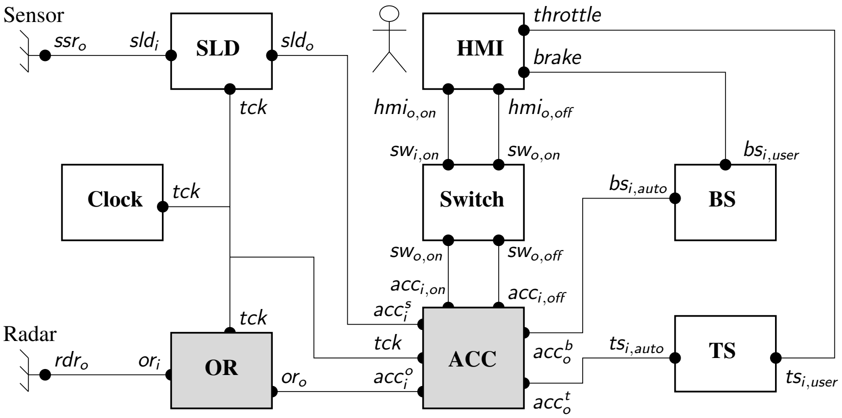
Property (Completeness)

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

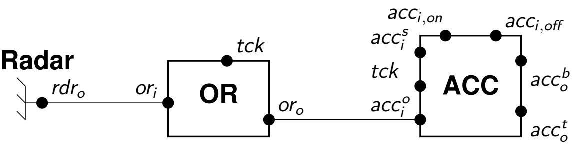
Remark

Causality defined on **contracts** *and* **observed traces**, *not implementations.*

Example 1: Adaptive Cruise Control



Example 1: Adaptive Cruise Control



- Obstacle recognition (OR)
  $\rightsquigarrow$ $\mathcal{G}_{OR}$: *"output 1 time unit after sensing"*
- Adaptive Cruise Control (ACC)
  $\rightsquigarrow$ $\mathcal{G}_{ACC}$: *"output 1 time unit after latest input"*
- Global guarantee
  $\rightsquigarrow$ $\mathcal{G}$: *"ACC output at most 3 time units after data acquisition"*

Example 1: Adaptive Cruise Control

Two necessary causes

Consider the following trace excerpts:

OR: $\ldots$ 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 ($\Delta_{OR} = 2$, $\Delta_{ACC} = 2$)
$\Longrightarrow$ 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: … or_i, **tck**, **tck**, **tck**, or_o, tck, tck, …

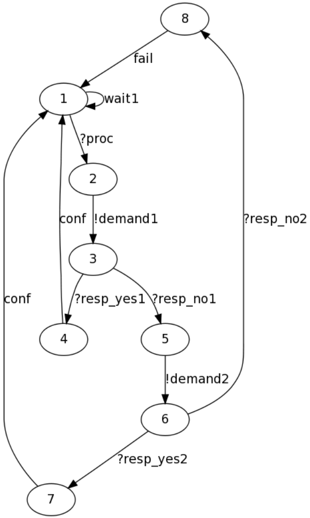
ACC: … tck, tck, tck, acc_i^s, **tck**, **tck**, acc_o^t, …

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

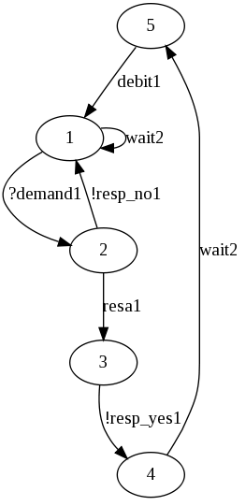
- OR's violation is a **necessary and sufficient** cause for the global failure.
- The violation of ACC is no longer a necessary cause.

Example 2: Travel Agency

Travel agency:



Hotel 1:



Example 2: Travel Agency

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

Spec 2: "each request is ack'ed by either **fail** or $\textbf{resa}_i \,.\, \textbf{!resp\_yes}_i$ for $i \in \{1, 2\}$"

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 $\textbf{resa}_i\,.\,\textbf{!resp\_yes}_i$ for $i \in \{1, 2\}$"

Observed traces:

agency: ?proc . $!\text{demand}_1$. $?\text{resp\_no}_1$. $!\text{demand}_2$. $?\text{resp\_yes}_2$. !conf

Example 2: Travel Agency

Spec 1: "at any time, #(debits) $\leq$ #(confirmations)"

Spec 2: "each request is ack'ed by either **fail** or **resa**$_i$ **. !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, $\#(\text{debits}) \leq \#(\text{confirmations})$"

Spec 2: "each request is ack'ed by either **fail** or $\textbf{resa}_i$ **.** $\textbf{!resp\_yes}_i$ 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
hotel 2: ?demand_2 . !resp_yes_2 . wait_2 . debit_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 $\mathbf{resa}_i$ **. !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$_1$
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

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 $\exists \mathbf{tr}\ \forall i : \pi_i(\mathbf{tr}) = tr_i'$.
- $Tr \subseteq \{tr_1, ..., tr_n\}$ set of traces to be analyzed jointly

Definition (Necessary causality)

Tr is a necessary cause of the violation of $\mathcal{C}$ if $\exists tr \in Tr$: $tr \nearrow \mathcal{C}$ and $\forall \mathbf{tr}'$:

$$\Big(\forall j \in \{1, ..., n\} \setminus \mathcal{I}' : \pi_j(\mathbf{tr}') = tr_j \wedge$$

$$\forall k \in \mathcal{I}' : \pi_k(\mathbf{tr}') \models \mathcal{C}_k\Big) \implies \mathbf{tr}' \models \mathcal{C}$$

where $\mathcal{I} = \{i \mid tr_i \in Tr \wedge tr_i \not\models \mathcal{C}_i\}$.

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 $\exists \mathbf{tr}\ \forall i : \pi_i(\mathbf{tr}) = tr'_i$.
- $Tr \subseteq \{tr_1, \ldots, tr_n\}$ set of traces to be analyzed jointly

Definition (Necessary causality)

Tr is a necessary cause of the violation of $\mathcal{C}$ if $\exists tr \in Tr$: $\boxed{tr \nearrow \mathcal{C}}$ and $\forall \mathbf{tr}'$:

$$\Big(\forall j \in \{1, \ldots, n\} \setminus \mathcal{I} : \boxed{\pi_j(\mathbf{tr}') = tr_j} \quad \wedge \quad \forall k \in \mathcal{I} : \pi_k(\mathbf{tr}') \models \mathcal{C}_k \implies \mathbf{tr}' \models \mathcal{C}$$

where $\mathcal{I} = \{i \mid tr_i \in Tr \wedge \boxed{tr_i \not\models \mathcal{C}_i} \}$.

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 $\exists \mathbf{tr}\ \forall i : \pi_i(\mathbf{tr}) = tr'_i$.
- $Tr \subseteq \{tr_1, ..., tr_n\}$ set of traces to be analyzed jointly

Definition (Necessary causality)

Tr is a necessary cause of the violation of $\mathcal{C}$ if $\exists tr \in Tr$: $\boxed{tr \nearrow \mathcal{C}}^{=}$ and $\forall \mathbf{tr}'$:

$$\Big(\forall j \in \{1, ..., n\} \setminus \mathcal{I} : \boxed{\pi_j(\mathbf{tr}') = tr_j}^{+} \wedge$$
$$\forall k \in \mathcal{I} : \pi_k(\mathbf{tr}') \models \mathcal{C}_k \implies \mathbf{tr}' \models \mathcal{C}$$

where $\mathcal{I} = \{i \mid tr_i \in Tr \wedge \boxed{tr_i \not\models \mathcal{C}_i}^{=}\}$.

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
  $\leadsto$ verify satisfaction of assumption and guarantee;
  no notion of causality, no concurrency.
- **Diagnosis:** determine (unobservable) faults from observations
  $\leadsto$ no notion of logical causality.

Discussion

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?