Timed Trace Expressions

Luca Ciccone[1] , Angelo Ferrando[2] ,
Davide Ancona[1] , and Viviana Mascardi[1]

[1] University of Genova, Genova, Italy
[2] Liverpool University, Liverpool, United Kingdom

CILC 2019, 34th Italian Conference on Computational Logic
19-21 June 2019, Trieste, Italy

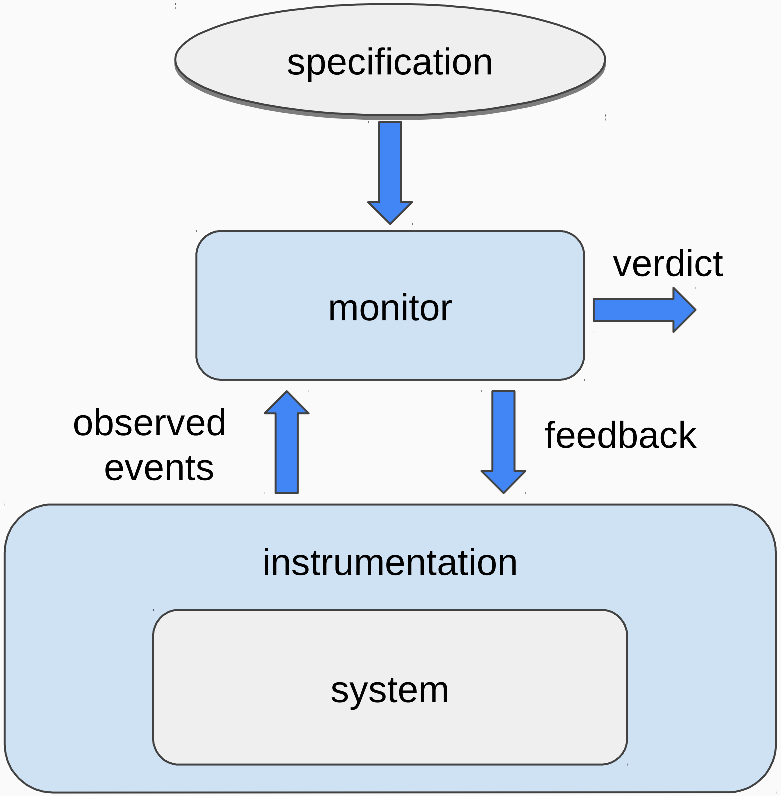
The context

Runtime Verification

in a nutshell

- Method based on dynamic analysis to ensure correctness of the execution of the “system under scrutiny”
- Systematic study and use is recent (first conference in 2001)

Runtime Verification Process



Trace Expressions

Trace Expressions

Trace Expressions (TEs in the sequel) are a compact and expressive formalism, which can be employed to model “expected behaviors of the system” (choreographies, complex interaction protocols, properties on data types) based on a set of operators to denote finite and infinite traces of events

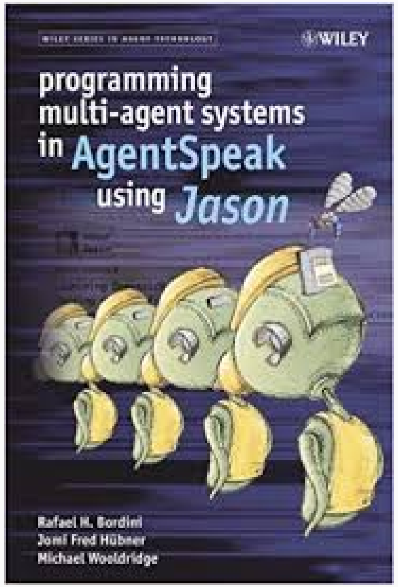
$$\text{(prefix)}\ \frac{}{\vartheta{:}\tau \xrightarrow{e} \tau}\ e\in\vartheta \qquad \text{(or-l)}\ \frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1\vee\tau_2 \xrightarrow{e} \tau_1'} \qquad \text{(or-r)}\ \frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1\vee\tau_2 \xrightarrow{e} \tau_2'} \qquad \text{(and)}\ \frac{\tau_1 \xrightarrow{e} \tau_1' \quad \tau_2 \xrightarrow{e} \tau_2'}{\tau_1\wedge\tau_2 \xrightarrow{e} \tau_1'\wedge\tau_2'} \qquad \text{(shuffle-l)}\ \frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1|\tau_2 \xrightarrow{e} \tau_1'|\tau_2}$$

$$\text{(shuffle-r)}\ \frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1|\tau_2 \xrightarrow{e} \tau_1|\tau_2'} \qquad \text{(cat-l)}\ \frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1\cdot\tau_2 \xrightarrow{e} \tau_1'\cdot\tau_2} \qquad \text{(cat-r)}\ \frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1\cdot\tau_2 \xrightarrow{e} \tau_2'}\ \epsilon(\tau_1) \qquad \text{(cond-t)}\ \frac{\tau \xrightarrow{e} \tau'}{\vartheta\gg\tau \xrightarrow{e} \vartheta\gg\tau'}\ e\in\vartheta \qquad \text{(cond-f)}\ \frac{}{\vartheta\gg\tau \xrightarrow{e} \vartheta\gg\tau}\ e\notin\vartheta$$

$$(\epsilon\text{-empty})\ \frac{}{\epsilon(\epsilon)} \qquad (\epsilon\text{-or-l})\ \frac{\epsilon(\tau_1)}{\epsilon(\tau_1\vee\tau_2)} \qquad (\epsilon\text{-or-r})\ \frac{\epsilon(\tau_2)}{\epsilon(\tau_1\vee\tau_2)} \qquad (\epsilon\text{-shuffle})\ \frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1|\tau_2)}$$

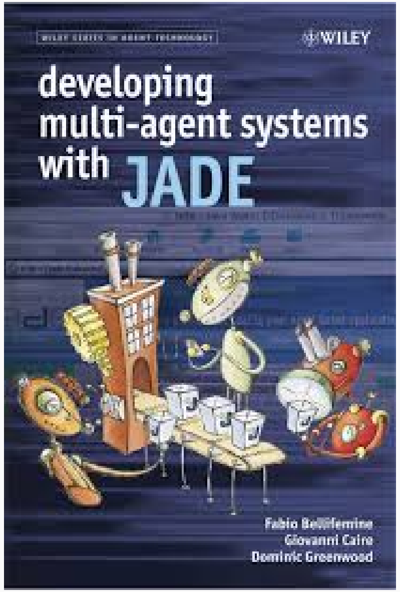
$$(\epsilon\text{-cat})\ \frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1\cdot\tau_2)} \qquad (\epsilon\text{-and})\ \frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1\wedge\tau_2)} \qquad (\epsilon\text{-cond})\ \frac{\epsilon(\tau)}{\epsilon(\vartheta\gg\tau)}$$

Example: Trace Expressions for Agent Interaction Protocols

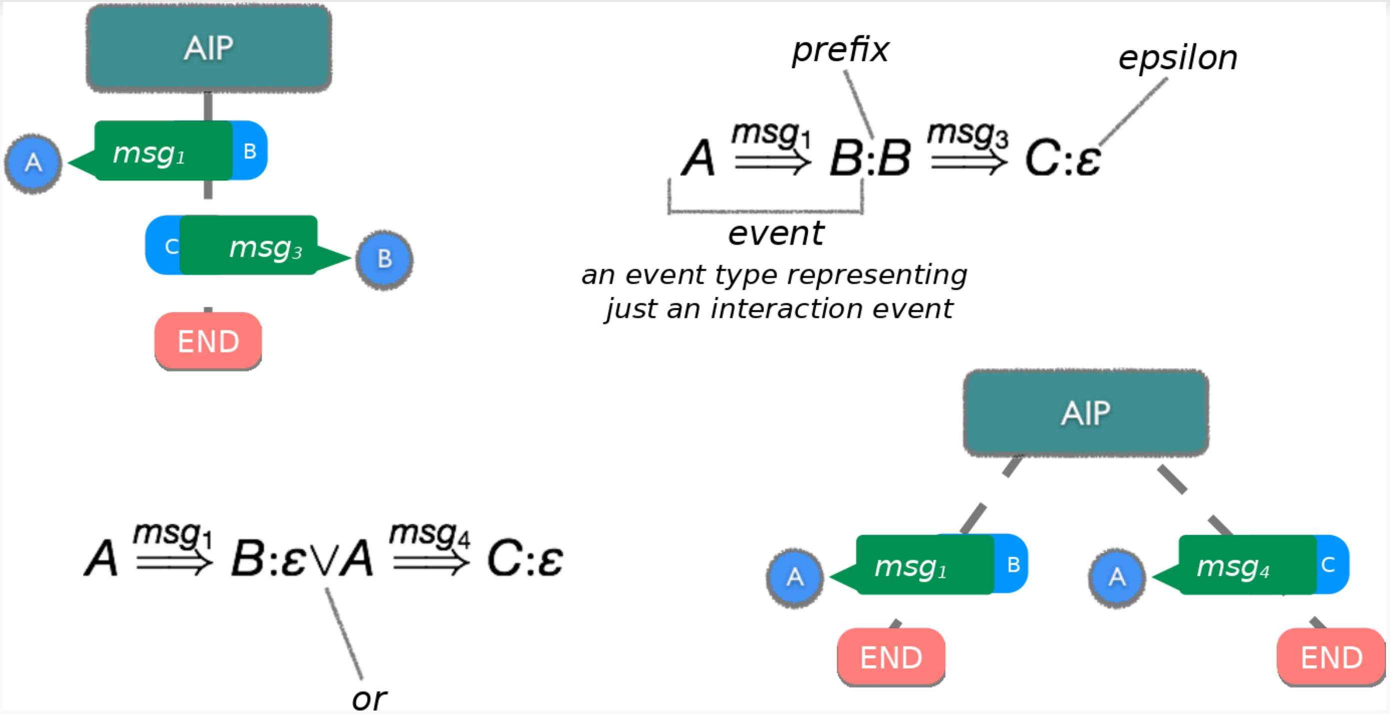


Monitors automatically generated from trace expressions, to verify at runtime interactions among agents in Jason and JADE

Relevant events:
sent messages



Example: Trace Expressions for Agent Interaction Protocols



Example: Trace Expressions for Agent Interaction Protocols

Also

- ✗ shuffle (interlaving)
- ✗ concatenation (extends the expressive power to non context free languages: we can model traces like $a^n b^n c^n$, which cannot be modeled in LTL)
- ✗ intersection
- ✗ recursion

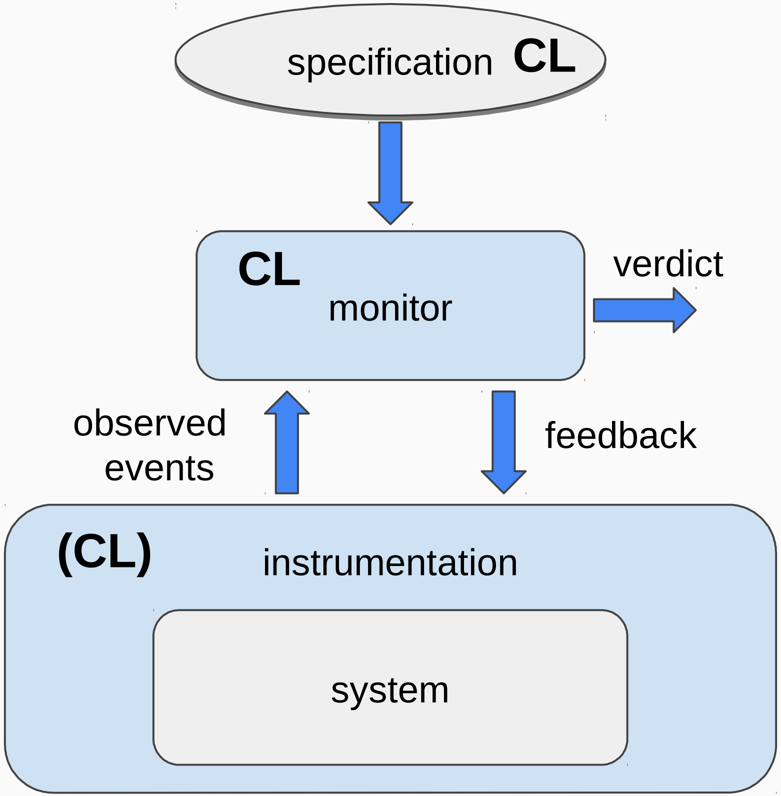
Example: Trace Expressions for other domains

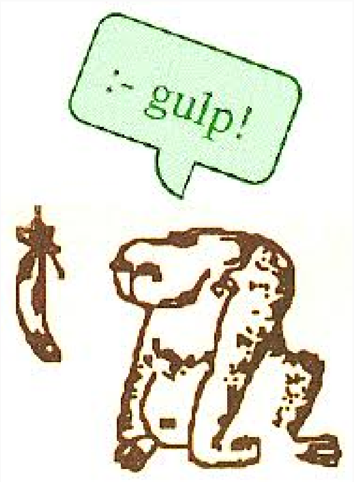
- RV of Node.js applications:
  Relevant events: function/method calls and execution of their callbacks

- RV of IoT systems:
  Relevant events: messages passed through Node-RED nodes

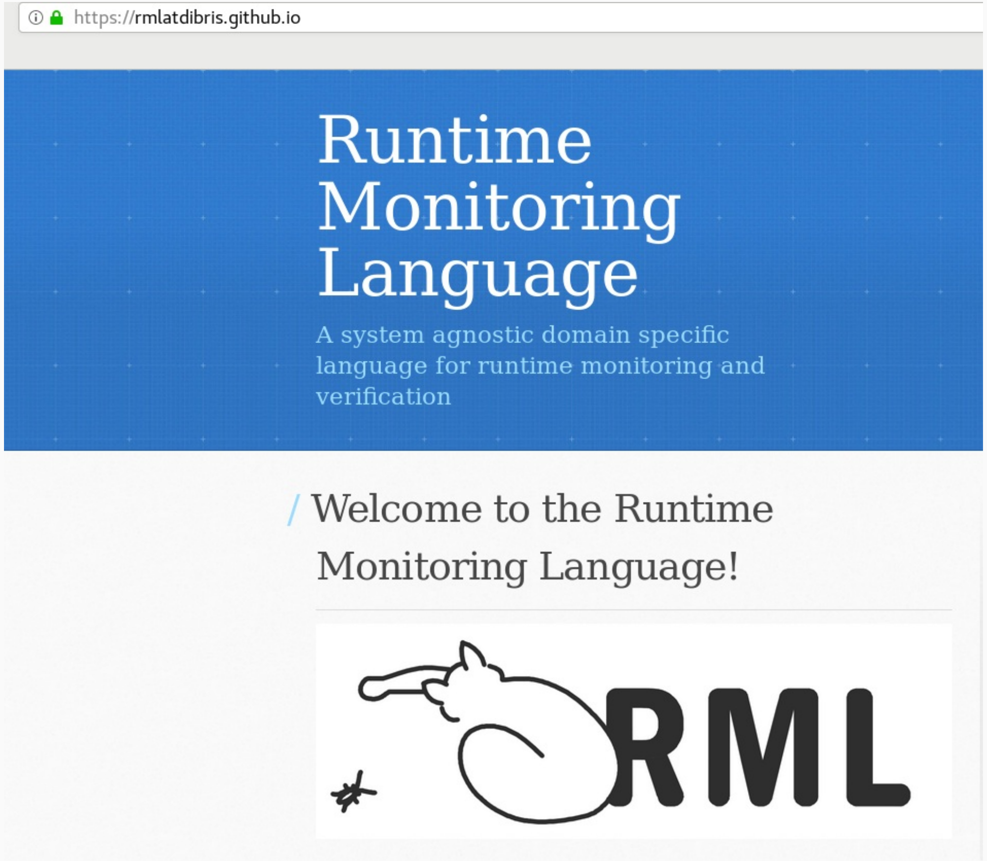
…where is Computational Logic…?

Runtime Verification Process

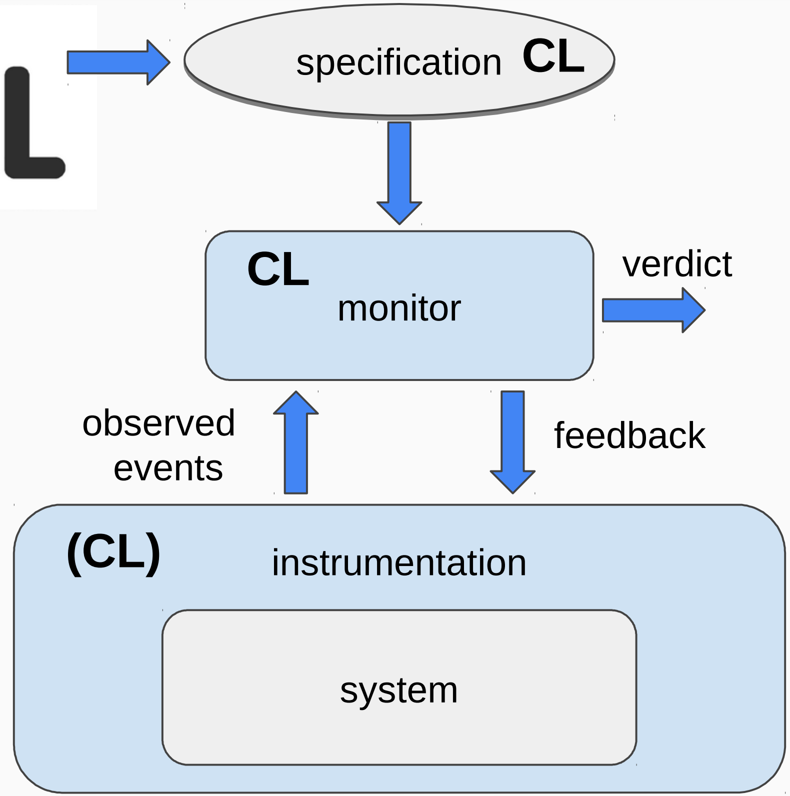


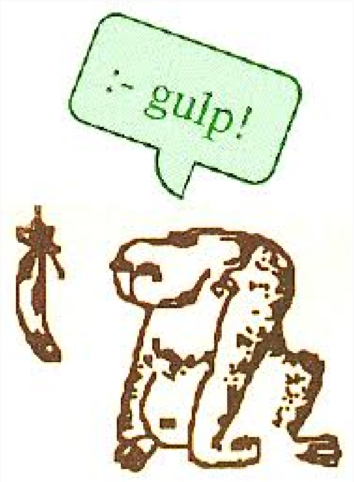


Runtime Verification Process



Runtime Verification Process





Prolog implementation of TE semantics

- One-to-one natural translation of transition rules into clauses

- Using the native Prolog support in Jason
- Using SWI Prolog and the JPL bidirectional Java-SWI Prolog interface for JADE, Node.js, IoT systems

Prolog implementation of...

All the algorithms driven by the TE structure, such as

- Projecting TEs representing AIPs on subsets of agents
- Transforming TEs into TEs with probabilities
- Checking the possibility to verify them in a distributed way
- ....

- Sophisticated management of cycles thanks to the “assoc” SWI-Prolog library (also experimented the coinduction library)

Many years of TEs, always using Prolog

- ✔ Davide Ancona, Angelo Ferrando, Viviana Mascardi: Improving flexibility and dependability of remote patient monitoring with agent-oriented approaches. IJAOSE 6(3/4): 402-442 (2018)
- ✔ Maurizio Leotta, Davide Ancona, Luca Franceschini, Dario Olianas, Marina Ribaudo, Filippo Ricca: Towards a Runtime Verification Approach for Internet of Things Systems. ICWE Workshops 2018: 83-96
- ✔ Davide Ancona, Angelo Ferrando, Viviana Mascardi: Agents Interoperability via Conformance Modulo Mapping. WOA 2018: 109-115
- ✔ Angelo Ferrando, Davide Ancona, Viviana Mascardi: Decentralizing MAS Monitoring with DecAMon. AAMAS 2017: 239-248
- ✔ Davide Ancona, Angelo Ferrando, Viviana Mascardi: Parametric Runtime Verification of Multiagent Systems. AAMAS 2017: 1457-1459
- ✔ Davide Ancona, Angelo Ferrando, Luca Franceschini, Viviana Mascardi: Parametric Trace Expressions for Runtime Verification of Java-Like Programs. FTfJP@ECOOP 2017: 10:1-10:6
- ✔ ....

Timed Trace Expressions

Addition of time intervals to events

Extension to the basic setting inspired by existing logics for dealing with time intervals

- ✗ Minor syntactic extension: intervals associated with event types
- ✗ No need to change the transition rules: only the “match” predicate has been modified
- ✗ Implemented in SWI Prolog

Example

$$Agreement = (\text{bob\_on\_time} : \epsilon) \mid$$
$$(Alice\_on\_time \lor Alice\_standard\_delay \lor Alice\_except\_delay)$$

where

$$Alice\_on\_time = (\text{alice\_on\_time} : \text{alice\_and\_bob\_enter\_together} : \epsilon)$$

$$Alice\_standard\_delay = (\text{alice\_late} : ((\text{bob\_enters} : \epsilon) \mid (\text{alice\_enters} : \epsilon)))$$

$$Alice\_except\_delay = (\text{alice\_too\_late} : ((\text{bob\_enters} : \epsilon) \mid (\text{alice\_gives\_up} : \epsilon)))$$

Example

bob_on_time = < {“bob in front of CILC venue”},
[9.00 AM, 9.20 AM] >

alice_on_time = < {“alice in front of CILC venue”},
[9.00 AM, 9.20 AM] >

alice_late = < {“alice in front of CILC venue”},
(9.20 AM, 11.00 AM] >

alice_too_late = < {“alice in front of CILC venue”},
(11.00 AM, 12.00 PM] >

Issues & future work

- Intervals associated with event types have a global scope, but in some cases it might be more convenient to have intervals with a local scope, associating them with sub-traces, rather than with events.
- This extension, however, would require major changes to the syntax and semantcs by introducing an explicit notion of “scope of an interval”
- Design of static check in CLP under way
- Can we translate Metric Temporal Logic (MTL) and Metric Interval Temporal Logic (MITL) into Timed Trace Expressions?

Thank you for your attention!