Timed Trace Expressions

Luca Ciccone¹, Angelo Ferrando², Davide Ancona¹, and Viviana Mascardi¹

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)
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.
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

\[ A \xrightarrow{\text{msg}_1} B:\varepsilon \lor A \xrightarrow{\text{msg}_4} C:\varepsilon \]

or

\[ A \xrightarrow{\text{msg}_1} B:B \xrightarrow{\text{msg}_3} C:\varepsilon \]

Prefix

Epsilon

Event

A event type representing just an interaction event
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

- Specification
- Monitor
- Instrumentation
- System
- Observed events
- Feedback
- Verdict

(CL)
Runtime Verification Process

specification **CL**

**CL** monitor

observed events

feedback

**CL** instrumentation

system

---

RML

:: gulp!
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

- 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
- 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

\[
Agreement = (bob\_on\_time : \varepsilon) \mid \\
(Alice\_on\_time \lor Alice\_standard\_delay \lor Alice\_except\_delay)
\]

where

\[
Alice\_on\_time = (alice\_on\_time : alice\_and\_bob\_enter\_together : \varepsilon)
\]

\[
Alice\_standard\_delay = (alice\_late : ((bob\_enters : \varepsilon) \mid (alice\_enters : \varepsilon)))
\]

\[
Alice\_except\_delay = (alice\_too\_late : ((bob\_enters : \varepsilon) \mid (alice\_gives\_up : \varepsilon)))
\]
Example

bob_on_time = < \{“bob in front of CILC venue”\},
\[9.00 \text{ AM}, \ 9.20 \text{ AM}] >
alice_on_time = < \{“alice in front of CILC venue”\},
\[9.00 \text{ AM}, \ 9.20 \text{ AM}] >
alice_late = < \{“alice in front of CILC venue”\},
\ (9.20 \text{ AM}, \ 11.00 \text{ AM}] >
alice_too_late = < \{“alice in front of CILC venue”\},
\ (11.00 \text{ AM}, \ 12.00 \text{ 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 semantics 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!