Automatic Partitions Extraction to Distribute the Runtime Verification of a Global Specification

Angelo Ferrando*

DIBRIS, University of Genova, Italy
angelo.ferrando@dibris.unige.it

Abstract. Trace expressions are a compact and expressive formalism, initially devised for runtime verification of agent interactions in multiagent systems (MAS), which has been successfully employed to model real protocols, and to generate monitors for mainstream multiagent system platforms, and generalized to support runtime verification of different kinds of properties and systems. In this paper we present the trace expression formalism and its use in the runtime verification context focusing on future works related to the distribution aspects.

Key words: Distributed Runtime Verification, Multiagent Systems, Distributed Monitoring, Trace expressions

1 Introduction and Motivations

Runtime verification (RV) is a software verification technique that complements formal static verification (as model checking) and testing. In RV dynamic checking of the correct behavior of a system can be performed by a monitor which is generated from a formal specification of the properties to be verified. A possible way to specify the monitor behavior is through the set of all correct traces (finite or infinite sequences of events) which can be generated during the system execution. Sets of traces may be represented in many different ways. In this extended abstract we present trace expressions [3, 4], a formalism inspired to session types [8, 12] which can be used to design monitors for the RV of centralized and decentralized software systems. Distributed runtime verification (DRV), as described by S. Rajsbaum in his keynote speech at the SSS 2015 Symposium [10], tackles the problem of building a decentralized runtime monitor for a distributed system and involves designing a distributed algorithm that coordinates the monitors in order for them to correctly verify the dynamic behavior of the whole system. Once the formal specification of the global pattern of events is given, however, distributing the monitoring activity can be resorted to decomposing the global specification into "sub-specifications", involving less events than the global one, which can be monitored in an independent way from each other and such that

* Prof. Viviana Mascardi and Davide Ancona are the author's advisors.

the union of their independent monitoring gives the same guarantees as the monitoring of the whole system w.r.t. the original specification. The trace expression formalism can be used to construct protocols representing sequences on generic events. Since we focus on its distributed aspects, we limit the set of the events to the subset corresponding to the interaction events, where interactions are intended as exchange of messages among agents in a MAS. Trace expressions are already used for centralized runtime verification (no distribution) [4] and for the automatic generation of protocol-driven agents (total distribution) [2] through projection on each single agent involved in the protocol. This work lays the foundations to fill the gap between these two projects obtaining a middle way approach between the centralization (bottleneck) and the total distribution (inefficient) implementations.

2 Trace Expressions

A trace expression τ represents a set of possibly infinite event traces, and is defined on top of the following operators: (1) ϵ (empty trace), denoting the singleton set $\{\epsilon\}$ containing the empty event trace ϵ; (2) $\vartheta{:}\tau$ (*prefix*), denoting the set of all traces whose first event e matches the event type[1] ϑ ($e \in \vartheta$), and the remaining part is a trace of τ; (3) $\tau_1{\cdot}\tau_2$ (*concatenation*), denoting the set of all traces obtained by concatenating the traces of τ_1 with those of τ_2; (4) $\tau_1 \wedge \tau_2$ (*intersection*), denoting the intersection of the traces of τ_1 and τ_2; (5) $\tau_1 \vee \tau_2$ (*union*), denoting the union of the traces of τ_1 and τ_2; (6) $\tau_1 | \tau_2$ (*shuffle*), denoting the set obtained by shuffling the traces of τ_1 with the traces of τ_2; (7) $\vartheta \gg \tau$ (*filter*), a derived operator denoting the set of all traces contained in τ, when deprived of all events that do not match ϑ. Event types represent sets of generic events; in the following when we consider interaction types, which are a special case of event types related to the exchange of messages among agents, we denote them as α and we represent the interaction events contained inside as $e_{s \to r}$ meaning that the sender s sends the message e to the receiver r. To support recursion without introducing an explicit construct, trace expressions are regular (a.k.a. rational or cyclic) terms: they correspond to trees where nodes[2] are either the leaf ϵ, or the node (corresponding to the prefix operator) ϑ with one child, or the nodes $\cdot$, $\wedge$, $\vee$, and $|$ all having two children. According to the standard definition of rational trees, their depth is allowed to be infinite, but the number of their subtrees must be finite. The semantics of trace expressions is specified by the transition relation $\delta \subseteq \mathcal{T} \times \mathcal{E} \times \mathcal{T}$, where $\mathcal{T}$ and $\mathcal{E}$ denote the set of trace expressions and of events, respectively. As it is customary, we write $\tau_1 \xrightarrow{e} \tau_2$ to mean $(\tau_1, e, \tau_2) \in \delta$. If the trace expression τ_1 specifies the current valid state of the system, then an event e is considered valid iff there exists a transition $\tau_1 \xrightarrow{e} \tau_2$; in such a case, τ_2 will specify the next valid state of the system after

[1] To be more general, trace expressions are built on top of event types (chosen from a set $\mathcal{ET}$), rather than of single events; an event type denotes a subset of $\mathcal{E}$.

[2] Since the filter is a derived operator it can be rewritten and there is not a correspondence on the tree.

event e. Otherwise, the event e is not considered to be valid in the current state represented by τ_1.
Example 1. Let $\mathcal{E} = \{e_1, \ldots, e_7\}$, and ϑ_i, $i = 1, \ldots, 7$, be the event types such that $e \in \vartheta_i$ iff $e = e_i$ (that is, $[\![\vartheta_i]\!] = \{e_i\}$); then the trace expression $\tau_1 = ((\vartheta_1{:}\epsilon|\vartheta_2{:}\epsilon)\vee(\vartheta_3{:}\epsilon|\vartheta_4{:}\epsilon))\cdot(\vartheta_5{:}\vartheta_6{:}\epsilon|\vartheta_7{:}\epsilon)$ denotes the following set of event traces:

$$\left\{\begin{array}{l} e_1e_2e_5e_6e_7, e_1e_2e_5e_7e_6, e_1e_2e_7e_5e_6, e_2e_1e_5e_6e_7, e_2e_1e_5e_7e_6, e_2e_1e_7e_5e_6, \\ e_3e_4e_5e_6e_7, e_3e_4e_5e_7e_6, e_3e_4e_7e_5e_6, e_4e_3e_5e_6e_7, e_4e_3e_5e_7e_6, e_4e_3e_7e_5e_6 \end{array}\right\}$$

Example 2. Let the set of interaction types $\mathcal{ET} = \{\alpha_{ping}, \alpha_{pong}\}$ and $\mathcal{E} = \{ping_{A\to B}, pong_{B\to A}\}$, $[\![\alpha_{ping}]\!] = \{ping_{A\to B}\}$ and $[\![\alpha_{pong}]\!] = \{pong_{B\to A}\}$; then the trace expression $PingPong = (\alpha_{ping}{:}PingPong \cdot \alpha_{pong}{:}\epsilon)\vee\epsilon$ denotes the following set of interaction events:
$\{ping_{A\to B}pong_{B\to A}, ping_{A\to B}ping_{A\to B}pong_{B\to A}pong_{B\to A}, ..., ping^n_{A\to B}pong^n_{B\to A}\}$.

2.1 Projection

The projection function was first introduced in [1]. Given the finite set AGS of all the agents that could play a role in the MAS and an interaction type α (an event type containing only interaction events), $senders(\alpha)$ is the set of all the agents in AGS that could play the role of sender in actual interactions having type α, and $receivers(\alpha)$ is the set of all the agents in AGS that could play the role of receiver in interactions of type α. The *involves* predicate holds on one interaction type α and one set of agents Ags, $involves(\alpha, Ags)$, iff $(senders(\alpha) \cap Ags \neq \emptyset) \vee (receivers(\alpha) \cap Ags \neq \emptyset)$.

We define two auxiliary functions $first : \mathcal{T} \to \mathcal{P}(\mathcal{ET})$ ($last : \mathcal{T} \to \mathcal{P}(\mathcal{ET})$), $first(\tau) = \{\vartheta_1, \vartheta_2, ..., \vartheta_n\}$ ($last(\tau) = \{\vartheta_1, \vartheta_2, ..., \vartheta_n\}$) iff $\forall_{\vartheta_i}.\exists_{e\in\vartheta_i}$ s.t. e is one of the first (last) events which can be consumed by the δ transition relation called on τ. In this way we can introduce the derived sets $involved(\alpha) = Ags \iff senders(\alpha)\cup receivers(\alpha) = Ags$ and $involved(\epsilon) = \emptyset$, $involved(\tau) = involved(\alpha_1)\cup...\cup involved(\alpha_n)\cup involved(\tau_1)\cup...\cup involved(\tau_m)$, with $first(\tau) = \{\alpha_1, ..., \alpha_n\}$ and $\forall_{1\leq i\leq m}.\tau \xrightarrow{e} \tau_i$, for some e. Projection can be described as a function $\Pi : \mathcal{T} \times \mathcal{P}(AGS) \to \mathcal{T}$. Π is driven by the syntax of the trace expression to project; since Π is defined on cyclic terms, the simplest way to define it is by coinduction as follows: (i) $\Pi(\epsilon, Ags) = \epsilon$, (ii) $\Pi(\alpha : \tau, Ags) = \alpha : \Pi(\tau, Ags)$ if $involves(\alpha, Ags)$, (iii) $\Pi(\alpha : \tau, Ags) = \Pi(\tau, Ags)$ if $\neg involves(\alpha, Ags)$, (iv) $\Pi(\tau' \ op \ \tau'', Ags) = \Pi(\tau', Ags) \ op \ \Pi(\tau'', Ags)$, where $op \in \{\cdot, \wedge, \vee, |\}$.

3 Trace Expression Distribution

As already anticipated, we consider trace expressions to model interaction protocols inside a MAS. With respect to the centralized runtime verification approach proposed in [4], we want to reason about the trace expression distribution in order to obtain a set of trace expressions where each one represents the protocol subset related to a fixed set of agents. The main problem of this approach is to understand which agents can be monitored separately and which must

be monitored together (*unsplittable*) in order to guarantee that the distributed monitoring gives the same results as the centralized one. This notion is strongly related to the correctness definition in the *Choreographies* research field [9] where a choreography can be projected on each single entity (agent for us) only if it satisfies three conditions (for trace expressions the first two are enough). We report these two conditions adapted for the trace expression formalism.

Definition 1. *The two conditions which a trace expression τ must satisfy in order to be correctly projected on each single agent involved in its event types are:*

1. Connectedness for sequence.
   For each sub-expression $\alpha{:}\tau_1$, with $first(\tau_1) = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, we have $\forall_{\alpha_i \in first(\tau_1)}.involved(\alpha) \cap involved(\alpha_i) \neq \emptyset$. For each sub-expression $\tau_1 \cdot \tau_2$, with $last(\tau_1) = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ and $first(\tau_2) = \{\alpha'_1, \alpha'_2, ..., \alpha'_n\}$, we have $\forall_{\alpha \in last(\tau_1)}.\forall_{\alpha' \in first(\tau_2)}.involved(\alpha) \cap involved(\alpha') \neq \emptyset$.
2. Unique point of choice.
   For each sub-expression $\tau_1 \vee \tau_2$, with $first(\tau_1) = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ and $first(\tau_2) = \{\alpha'_1, \alpha'_2, ..., \alpha'_n\}$, we have $\forall_{\alpha \in first(\tau_1)}.\forall_{\alpha' \in first(\tau_2)}.involved(\alpha) \cap involved(\alpha') \neq \emptyset$ and $involved(\tau_1) = involved(\tau_2)$.

Without loss of generality we consider only trace expressions where all interaction types represent finite sets of events. Thus, we can have interaction types like $\alpha = \{i1_{A \to B}, i2_{B \to A}, i3_{A \to B}, ...\}$ and not interaction types like $\alpha = \{i1_{A \to B}, i2_{C \to D}, ...\}$ or $\alpha = \{i1_{A \to B}, i2_{A \to D}, ...\}$.

We can observe that the two conditions are too restrictive when we want to distribute our protocols on sets of agents. In particular, we can easily find protocols that cannot be projected on a single agent but can be projected on a set of agents. Considering for instance the trace expression $\tau = ...(\alpha_{ping}{:}\alpha_{pong}{:}\epsilon)...$ where ... means that τ contains also other terms and $[\![\alpha_{ping}]\!] = \{ping_{A \to B}\}$, $[\![\alpha_{pong}]\!] = \{pong_{C \to D}\}$, can immediately note that it does not satisfy the first condition (the connectedness for sequence). Since τ can be a very large trace expression we would like to distribute it anyway. Even if we cannot distribute it on each single agent $\{\{A\}, \{B\}, \{C\}, \{D\}\}$, in this particular case a possible choice for the distribution could be $\{\{A\}, \{B, C\}, \{D\}\}$ obtaining three different projections of the global protocol in three separated parts where B and C are projected and monitored together.

Example 3. Given the trace expression $\tau = (\alpha_{m1}{:}\epsilon) \vee (\alpha_{m2}{:}\epsilon)$ where $AGS = \{A, B, C, D\}$, $[\![\alpha_{m1}]\!] = \{msg1_{A \to B}\}$ and $[\![\alpha_{m2}]\!] = \{msg2_{C \to D}\}$. $\Pi(\tau, \{A\}) = \Pi(\tau, \{B\}) = \Pi((\alpha_{m1}{:}\epsilon) \vee (\alpha_{m2}{:}\epsilon)) = (\alpha_{m1}{:}\Pi(\epsilon)) \vee (\Pi(\epsilon)) = (\alpha_{m1}{:}\epsilon) \vee (\epsilon) = \alpha_{m1}{:}\epsilon$ and $\Pi(\tau, \{C\}) = \Pi(\tau, \{D\}) = \Pi((\alpha_{m1}{:}\epsilon) \vee (\alpha_{m2}{:}\epsilon)) = (\Pi(\epsilon)) \vee (\alpha_{m2}{:}\Pi(\epsilon)) = (\epsilon) \vee (\alpha_{m2}{:}\epsilon) = \alpha_{m2}{:}\epsilon$ The two local versions of the protocol have lost the unique point of choice information.

3.1 Automatic Partitions Extraction

An important feature in order to distribute a trace expression is the automatic generation of the set of all possible partitions of agents that preserve the semantics during the projection phase. Since we know that a trace expression must

satisfy the two conditions (*Definition 1*) to be correctly projected on each single agents involved in the protocol, we can use this information to guide our analysis. From an high level point of view, given a trace expression τ, for each subterm of τ we can check if the two conditions are satisfied; if a condition is not satisfied on a singular agent, we try to evaluate it on a set of agents containing the agent. Considering for instance the first condition (connectedness for sequence), if we find a subterm $\alpha_1 : \alpha_2 : ...$, with $involved(\alpha_1) \cap involved(\alpha_2) = \emptyset$, it does not satisfy the condition, but we can generate the set of all possible correct partitions $\{\{a, b\} | a \in involved(\alpha_1), b \in involved(\alpha_2)\}$ forcing in this way the connectedness property. A similar thing should be done for the other condition. Once we have obtained the set with all correct partitions we can choose the one that best meets our distribution requirements.

We are working on the implementation of the corresponding algorithm focusing on the correctness and completeness proofs. The intuition is that forcing the preservation of the two conditions (*Definition 1*) we preserve all critical points of the trace expression, where the critical points are all points of the protocol where it could be necessary monitor some agents together.

4 Related and Future Work

In this extended abstract we have presented a possible approach for the distribution of a trace expression on a set of agents. Following the conditions deriving from Choreographies (*Definition 1*) we can extract the partitions of agents which allow us to obtain the semantic preservation during the projection phase. Once we have the set of all possible valid partitions we can use them to generate the monitors to perform the distributed runtime verification of our MAS.

We can find works on distributed runtime monitoring like [7, 6, 11] but w.r.t. DRV of MASs we were not able to find any related work, except for the one which spun off from [1], namely [5]. Another relevant work is [13] where an interesting distributed approach to process mining is presented. Considering the conformance checking, that work is extremely related to ours and it can be useful to study the possible connections. For instance using trace expressions instead of Petri nets for the conformance checking of event logs.

Future work will be to implement the algorithm corresponding to this intuition using the programming language Prolog whereby we have already implemented the semantics of trace expressions and the projection function. Once we will have implemented the distribution algorithm we will be able to project the global protocol on each single partition in order to automatically generate distributed monitors. A possible implementation could be using a MASs framework based on logic programming, in this way we will be able to generate monitors directly by the projected trace expressions. We will try our approach first on some well known communicative protocols (as Contract Net Protocol, Alternating Bit Protocol, and so on) and after on a real case study concerning the distributed runtime verification of a MAS.

References

1. D. Ancona, D. Briola, A. El Fallah Seghrouchni, V. Mascardi, and P. Taillibert. Efficient verification of MASs with projections. In *Engineering Multi-Agent Systems - Second International Workshop, EMAS 2014, Revised Selected Papers*, Lecture Notes in Computer Science, 2014.
2. D. Ancona, D. Briola, A. Ferrando, and V. Mascardi. Global protocols as first class entities for self-adaptive agents. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, pages 1019–1029, 2015.
3. D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In *DALT 2012*, volume 7784 of *LNAI*, pages 76–95. Springer International Publishing, 2012.
4. D. Ancona, A. Ferrando, and V. Mascardi. *Theory and Practice of Formal Methods: Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, chapter Comparing Trace Expressions and Linear Temporal Logic for Runtime Verification, pages 47–64. Springer International Publishing, Cham, 2016.
5. D. Briola, V. Mascardi, and D. Ancona. Distributed runtime verification of JADE multiagent systems. In D. Camacho, L. Braubach, S. Venticinque, and C. Badica, editors, *IDC 2014*, volume 570 of *Studies in Computational Intelligence*, pages 81–91. Springer International Publishing, 2014.
6. T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida. *Asynchronous Distributed Monitoring for Multiparty Session Enforcement*, pages 25–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
7. A. Francalanza, A. Gauci, and G. Pace. Runtime monitoring of distributed systems (extended abstract). Technical report, University of Malta, 2010. WICT.
8. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *Proceedings of the 7th European Symposium on Programming: Programming Languages and Systems*, ESOP '98, pages 122–138, London, UK, UK, 1998. Springer-Verlag.
9. I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction-and process-oriented choreographies. In *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 323–332. IEEE, 2008.
10. S. Rajsbaum. Distributed runtime verification – where combinatorics, fault-tolerance and formal methods meet. Keynote Talk at the SSS 2015, August 2015, 2015.
11. T. Scheffel and M. Schmitz. Three-valued asynchronous distributed runtime verification. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014*, pages 52–61, 2014.
12. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *In PARLE?94, volume 817 of LNCS*, pages 398–413. Springer-Verlag, 1994.
13. W. M. P. van der Aalst. *Distributed Process Discovery and Conformance Checking*, pages 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.