Trace Expressions for Parametric Runtime Verification

(Recent Developments)

Davide Ancona Angelo Ferrando Viviana Mascardi

DIBRIS, Università di Genova, Italy

{davide.ancona,viviana.mascardi}@unige.it, angelo.ferrando@dibris.unige.it

Introduction Runtime verification (RV) is a software verification technique that complements formal static verification (as model checking), and testing. In RV the correct behavior of a system is dynamically checked by a monitor generated from a formal specification which defines the set of valid or invalid execution traces of the system.

Parametricity [5] is an important feature of a monitoring system for effective RV, since, typically, a set of property needs to be verified for a statically unknown collection of different entities (objects, threads, processes, or agents) or resources (files, locks, etc.) that may evolve at runtime and may depend on specific data values.

In this work we propose an extension of *trace expressions* [3] to allow parametric specifications for RV; preliminary experimental results show that the extended formalism can be effectively adopted for RV of properties parametric in the involved entities, resources, and data values; interestingly, the formalism can be effectively exploited also when time is one of the monitored resources, to express and dynamically check non functional requirements such as performance and response time.

Trace expressions The formalism of trace expressions evolved from global types [2, 4, 1], which has been initially proposed for RV of agent interactions in multiagent systems. For the purpose of RV, trace expressions are strictly more powerful than Linear Temporal Logic [3]. Their semantics is based on a labeled transition system defined by a simple set of rewriting rules which directly drive the behavior of monitors generated from trace expressions.

Trace expressions are an expressive formalism based on *event types* and a set of operators (including prefixing, concatenation, shuffle, union, and intersection) to denote finite and infinite traces of events; recursion is supported by allowing trace expressions to be regular terms. An event type is a predicate specifying a set of possible events. For instance, the event type $safe(o)$ denotes all safe method invocations for a given object o; more formally, an event e matches type $safe(o)$ (that is, $match(e, safe(o))$ holds) iff $e = invk(o, m)$ (that is, a method named m has been called on an object o), and the method name m belongs to a given set of method names considered harmless.

Parametric trace expressions Parametric behavior in trace expressions can be achieved by allowing event types to contain variables that are instantiated at runtime, when events are matched against event types; to this aim, variables are introduced together with a corresponding new construct $X.\tau$ to limit the scope of variable X within τ.

Let us consider for instance the communication protocol consisting of an infinite sequence of interactions where at each step alice sends an integer value i to bob, which, in turn, is expected to reply with an integer greater than i; this behavior can be specified independently of the exchanged values by the parametric trace expression τ s.t. $\tau = I.alice(I):bob(I):\tau$, where the event types $alice(I)$ and $bob(I)$ match iff alice sends value I to bob, and bob sends a value larger than I to alice, respectively. The trace expression τ is a regular term which, in fact, contains infinite binders for I, hence, at each iteration in the protocol, alice can send a different integer to bob; this is different from the trace expression $I.\tau'$, with $\tau' = alice(I):bob(I):\tau'$, where there exists a unique binder and, hence, alice must always send the same value initially sent to bob; for instance, if $send(s, r, c)$ is the event "s sends to r message content c", then the event trace $send(alice, bob, 0)send(bob, alice, 1)send(alice, bob, -1)send(bob, alice, 5)\ldots$ is compatible with τ, but not with τ'.

Figure 1 defines the semantics of parametric trace expressions. The main transition relation $\tau \xrightarrow{e} \tau'$, where e is an event, is defined in terms of the auxiliary relation $\tau \xrightarrow{e} \tau', \sigma$, where σ is the substitution generated by the transition step.

A substitution $\sigma: Vars \to Values$ is a partial function from variables to values defined over the finite domain $dom(\sigma)$; $\sigma = match(e, \vartheta)$ holds iff event e matches event type ϑ with substitution σ, while $\sigma = \sigma_1 \cup \sigma_2$ is verified iff $dom(\sigma) = dom(\sigma_1) \cup dom(\sigma_2)$, and for all $X \in dom(\sigma)$, $\sigma(X) = \sigma_1(X)$ if $X \in dom(\sigma_1)$, and $\sigma(X) = \sigma_2(X)$, if $X \in dom(\sigma_2)$.

$$(\text{main})\frac{\tau \xrightarrow{e} \tau', \emptyset}{\tau \xrightarrow{e} \tau'} \qquad (\text{prefix})\frac{}{\vartheta{:}\tau \xrightarrow{e} \tau, \sigma}\ \sigma{=}match(e,\vartheta) \qquad (\text{or-l})\frac{\tau_1 \xrightarrow{e} \tau_1', \sigma}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_1', \sigma} \qquad (\text{or-r})\frac{\tau_2 \xrightarrow{e} \tau_2', \sigma}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_2', \sigma}$$

$$(\text{and})\frac{\tau_1 \xrightarrow{e} \tau_1', \sigma_1 \quad \tau_2 \xrightarrow{e} \tau_2', \sigma_2}{\tau_1 \wedge \tau_2 \xrightarrow{e} \tau_1' \wedge \tau_2', \sigma}\ \sigma{=}\sigma_1 \cup \sigma_2 \qquad (\text{shuffle-l})\frac{\tau_1 \xrightarrow{e} \tau_1', \sigma}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1' | \tau_2, \sigma} \qquad (\text{shuffle-r})\frac{\tau_2 \xrightarrow{e} \tau_2', \sigma}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1 | \tau_2', \sigma}$$

$$(\text{cat-l})\frac{\tau_1 \xrightarrow{e} \tau_1', \sigma}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_1' \cdot \tau_2, \sigma} \qquad (\text{cat-r})\frac{\tau_2 \xrightarrow{e} \tau_2', \sigma}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_2', \sigma}\ \epsilon(\tau_1)$$

$$(\text{var-t})\frac{\tau \xrightarrow{e} \tau', \sigma}{X.\tau \xrightarrow{e} \sigma\tau', \sigma_{\setminus X}}\ X \in dom(\sigma) \qquad (\text{var-f})\frac{\tau \xrightarrow{e} \tau', \sigma}{X.\tau \xrightarrow{e} X.\tau', \sigma}\ X \notin dom(\sigma)$$

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

$$(\epsilon\text{-cat})\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1 \cdot \tau_2)} \qquad (\epsilon\text{-and})\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1 \wedge \tau_2)} \qquad (\epsilon\text{-var})\frac{\epsilon(\tau)}{\epsilon(X.\tau)}$$

Figure 1: Semantics of parametric trace expressions and empty trace containment

The notation $\sigma\tau$ denotes the term obtained from τ by substituting all free occurrences of $X \in dom(\sigma)$ with $\sigma(X)$: $\sigma(\vartheta{:}\tau) = (\sigma\vartheta){:}(\sigma\tau)$, $\sigma(\tau_1\, op\, \tau_2) = (\sigma\tau_1)\, op\, (\sigma\tau_2)$ for $op \in \{\vee, \wedge, |, \cdot\}$, $\sigma(X.\tau) = X.(\sigma_{\setminus X}\tau)$; $\sigma\vartheta$ is defined as the homomorphic extension s.t. $\sigma X = \sigma(X)$, if $X \in dom(\sigma)$, and $\sigma X = X$, if $X \notin dom(\sigma)$.

Finally, $\sigma_{\setminus X}$ is the substitution where X is removed from the domain: $\sigma_{\setminus X} = \sigma'$ iff $dom(\sigma') = dom(\sigma) \setminus \{X\}$ and for all $X \in dom(\sigma')$ $\sigma'(X) = \sigma(X)$.

Examples of transition steps For the trace expression $\tau = I.alice(I){:}bob(I){:}\tau$ the transition steps $\tau \xrightarrow{e_1} \tau_1 \xrightarrow{e_2} \tau$ can be derived, where $e_1 = send(alice, bob, 0)$, $\tau_1 = bob(0){:}\tau$, and $e_2 = send(bob, alice, 1)$:

$$(\text{main})\frac{(\text{var-t})\dfrac{(\text{prefix})\dfrac{}{alice(I){:}bob(I){:}\tau \xrightarrow{e_1} bob(I){:}\tau, \{I \mapsto 0\}}}{I.alice(I){:}bob(I){:}\tau \xrightarrow{e_1} \{I \mapsto 0\}(bob(I){:}\tau), \emptyset}}{I.alice(I){:}bob(I){:}\tau \xrightarrow{e_1} bob(0){:}\tau} \qquad (\text{main})\frac{(\text{prefix})\dfrac{}{bob(0){:}\tau \xrightarrow{e_1} \tau, \emptyset}}{bob(0){:}\tau \xrightarrow{e_1} \tau}$$

For the trace expression $\tau = O.new(O){:}(M.invk(O, M){:}\epsilon|\tau)$, the transition steps $\tau \xrightarrow{e_1} \tau_1 \xrightarrow{e_2} \tau_2 \xrightarrow{e_3} \tau_3 \xrightarrow{e_4} \tau_4$ can be derived, where $e_1 = new(o_1)$, $\tau_1 = M.invk(o_1, M){:}\epsilon|\tau$, $e_2 = new(o_2)$, $\tau_2 = M.invk(o_1, M){:}\epsilon|M.invk(o_2, M){:}\epsilon|\tau$, $e_3 = invk(o_2, m_1)$, $\tau_3 = M.invk(o_1, M){:}\epsilon|\epsilon|\tau$, $e_4 = invk(o_1, m_2)$, and $\tau_4 = \epsilon|\epsilon|\tau$, and the event type $new(O)$ corresponds to "new object O has been created".

References

[1] D. Ancona, V. Bono, M. Bravetti, J. Campos, P. Castagna, G.and Deniélou, S.J. Gay, N. Gesbert, E. Giachino, R. Hu, E.B. Johnsen, F. Martins, V. Mascardi, F. Montesi, R. Neykova, N. Ng, L. Padovani, V. T. Vasconcelos, and N. Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.

[2] 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, 2012.

[3] D. Ancona, A. Ferrando, and V. Mascardi. Comparing trace expressions and linear temporal logic for runtime verification. In *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, pages 47–64, 2016.

[4] G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.

[5] Qingzhou Luo, Yi Zhang, Choonghwan Lee, Dongyun Jin, Patrick O'Neil Meredith, Traian-Florin Serbanuta, and Grigore Rosu. Rv-monitor: Efficient parametric runtime verification with simultaneous properties. In *Runtime Verification, RV 2014*, pages 285–300, 2014.