Data Quantification in Temporal Specification Languages

Domenico Bianculli, Giles Reger, Dmitriy Traytel

December 5, 2017

In this working group, we tried to collect and characterize the different kinds of quantification or parametrization that can be encountered in temporal specification languages used in the runtime verification community. Our selection, albeit far from being comprehensive, shows that the main semantic difference is the domain of quantification. We have identified five different groups.

Standard First-Order Quantification An early approach taken by Emerson's first-order linear temporal logic (FOLTL) [13] is to add standard first-order logic quantifiers to LTL. Thereby, the quantifiers range over a fixed domain, which is independent of the trace (sequence of structures). Chomicki's real-time extension of FOLTL, called metric first-order temporal logic (MFOTL) [9] follows this approach. The MonPoly monitoring tool [4] demonstrates how such quantifiers can be handled algorithmically.

Quantification over the Active Domain A different approach, inspired by the database community, is to quantify over the active domain, i.e., values that occur in the trace, rather than a fixed, separately given domain. For certain classes of properties, e.g., where all quantifiers are bounded by atomic propositions as in $\forall x.p(x) \rightarrow \phi$ or $\exists x.p(x) \wedge \phi$, active domain quantification coincides with the standard first-order quantification.

Several specification languages favor the active domain quantification, as it appears to be algorithmically more tractable. They differ in their definition of what the *active domain* in a trace is. The traditional database definition as *all values contained in the trace*, used in LTL-FO [12], is hard to realize in the online setting, where the trace is an infinite stream of events. An adaptation of the active domain to *all previously seen values* would fit this setting better. However, the most widespread interpretation is to restrict the quantification to *values seen at the current time-point*. For example, the languages LTL^{FO} [7], LTL-FO^{+} [15], and Parametrized LTL [19] use this semantics.

Freeze Quantification Freeze quantification is a further refinement of the quantification over the current time-point approach. The usage of registers restricts the quantification to be a singleton: the only value that populates the

register at a given time-point. Timed propositional temporal logic (TPTL) [1] uses such quantifiers to extend LTL with real-time constraints. Here, we are interested in quantification over the data dimension rather than the time dimension, as used in Freeze LTL [11] and its extensions [10]. A recent extension of MTL with freeze quantification over data $\text{MTL}^{\downarrow}$ [5] was used as the specification language when online monitoring our-of-order traces.

Templates and Parametric Trace Slicing Some approaches avoid explicit quantification in their formalisms. Yet, they allow parametric specifications, which are handled by decomposing traces containing data into propositional ones. This approach is known as parametric trace slicing [8, 18], which is at the core of the JavaMOP system [17] and in quantified event automata QEA [2].

More recently, the stream-based specification language LOLA [14] introduced parametrization in terms of template specifications. Semantically, templates behave similarly to parametric trace slicing, but the precise connections are yet to be explored.

Quantitative Quantifiers Finally, some data quantifiers in addition to binding a variable also perform an arithmetic operation (be it filtering, grouping, or aggregation) on the quantified values (be them data or the number of satisfied instances). Example languages in this space are LTL^{FO} extended with counting quantifiers [6], LTL_4-C [16] with its probabilistic quantifiers, and the extension of MFOTL with aggregations [3].

References

[1] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[2] H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In D. Giannakopoulou and D. Méry, editors, *FM 2012*, volume 7436 of *LNCS*, pages 68–84. Springer, 2012.

[3] D. A. Basin, F. Klaedtke, S. Marinovic, and E. Zalinescu. Monitoring of temporal first-order properties with aggregations. *Formal Methods in System Design*, 46(3):262–285, 2015.

[4] D. A. Basin, F. Klaedtke, S. Müller, and E. Zalinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, 2015.

[5] D. A. Basin, F. Klaedtke, and E. Zalinescu. Runtime verification of temporal properties over out-of-order data streams. In R. Majumdar and V. Kuncak, editors, *CAV 2017*, volume 10426 of *LNCS*, pages 356–376. Springer, 2017.

[6] A. Bauer, R. Goré, and A. Tiu. A first-order policy language for history-based transaction monitoring. In M. Leucker and C. Morgan, editors, *ICTAC 2009*, volume 5684 of *LNCS*, pages 96–111. Springer, 2009.

[7] A. Bauer, J. Küster, and G. Vegliach. The ins and outs of first-order runtime verification. *Formal Methods in System Design*, 46(3):286–316, 2015.

[8] F. Chen and G. Rosu. Parametric trace slicing and monitoring. In S. Kowalewski and A. Philippou, editors, *TACAS 2009*, volume 5505 of *LNCS*, pages 246–261. Springer, 2009.

[9] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.

[10] N. Decker and D. Thoma. On freeze LTL with ordered attributes. In B. Jacobs and C. Löding, editors, *FoSSaCS 2016*, volume 9634 of *LNCS*, pages 269–284. Springer, 2016.

[11] S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

[12] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In S. Vansummeren, editor, *PODS 2006*, pages 90–99. ACM, 2006.

[13] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. 1990.

[14] P. Faymonville, B. Finkbeiner, S. Schirmer, and H. Torfah. A stream-based specification language for network monitoring. In Y. Falcone and C. Sánchez, editors, *RV 2016*, volume 10012 of *LNCS*, pages 152–168. Springer, 2016.

[15] S. Hallé and R. Villemaire. Runtime monitoring of message-based workflows with data. In *EDOC 2008*, pages 63–72. IEEE Computer Society, 2008.

[16] R. Medhat, B. Bonakdarpour, S. Fischmeister, and Y. Joshi. Accelerated runtime verification of LTL specifications with counting semantics. In Y. Falcone and C. Sánchez, editors, *RV 2016*, volume 10012 of *LNCS*, pages 251–267. Springer, 2016.

[17] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Rosu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.

[18] G. Reger and D. E. Rydeheard. From first-order temporal logic to parametric trace slicing. In E. Bartocci and R. Majumdar, editors, *RV 2015*, volume 9333 of *LNCS*, pages 216–232. Springer, 2015.

[19] V. Stolz. Temporal assertions with parametrized propositions. *J. Log. Comput.*, 20(3):743–757, 2010.