GR(1) Synthesis for LTL Specification Patterns

Shahar Maoz[1] and Jan Oliver Ringert[1]

This abstract reports on [MR15a], where we investigated support for linear temporal logic (LTL) specification patterns in General Reactivity of Rank 1 (GR(1)) synthesis.

Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification [PR89]. Rather than manually constructing a system and using model checking to verify its compliance with its specification, synthesis offers an approach where a correct implementation of the system is automatically obtained for a given specification, if such an implementation exists. In the case of reactive synthesis, an implementation is typically given as an automaton that accepts input from the environment (e.g., from sensors) and produces the system's output (e.g., on actuators).

Two main challenges in bringing reactive synthesis to software engineering practice are its very high worst-case complexity – for LTL it is double exponential in the length of the formula, and the difficulty of writing declarative specifications using basic LTL operators.

To address the first challenge, Piterman et al. [PPS06, Bl12] have suggested the GR(1) fragment of LTL, which has an efficient polynomial time symbolic synthesis algorithm. GR(1) is a strict assume/guarantee subset of LTL, comprised of constraints for initial states, safety propositions over the current and successor state, and justice constraints (i.e., assertions about what should hold infinitely often). GR(1) synthesis has been used in various application domains and contexts, including robotics [KFP09, MR15b], and scenario-based specifications [MS12], to name a few.

To address the second challenge, Dwyer et al. [DAC99] have identified 55 LTL specification patterns, which are common in industrial specifications and make writing specifications easier. An example pattern is *p* `occurs between` *q* `and` *r* where *p*, *q*, and *r* are parameters of the pattern, which can be instantiated with non-temporal propositions. This pattern is numbered P09 and its semantics expressed in LTL according to [DAC99] is `G (q & !r -> (!r W (p & !r)))`.

In [MR15a] we show that almost all LTL specification patterns identified by Dwyer et al. can be used as assumptions and guarantees in the GR(1) fragment of LTL. Specifically, we present an automated, sound and complete translation of the patterns to the GR(1) form, which effectively results in an efficient reactive synthesis procedure for any specification that is written using the patterns.

Technically, the translation starts from the LTL formula of the pattern, translates it to a minimal deterministic Büchi automaton (DBW), if one exists, and then translates the

[1] School of Computer Science, Tel Aviv University, Israel

automaton to a GR(1) assumption or guarantee formula, while possibly adding auxiliary variables to the GR(1) synthesis problem. If no DBW exists, the pattern is not supported.

Critical to the usefulness of our approach is that the costly translation of LTL to DBW is done only once for every pattern. In fact, we have already done it and saved the result as a set of templates inside our synthesis tool. This works because patterns are instantiated only with propositions (not with nested temporal operators). We further show that patterns can even be instantiated with past LTL formulas, but not with nested future temporal operators.

To summarize our contribution, [MR15a] answers the following three questions: (1) is GR(1) expressive enough to support the Dwyer et al. patterns, which are well-recognized as common in industrial specifications?, (2) can the translation be done automatically (and correctly)?, and (3) what's the extra cost of doing it (e.g., in number of auxiliary variables)?

To answer the first two questions, we have implemented and automated the translation, and our findings show that 52 of the 55 patterns from the original work of Dwyer et al. [DAC99] can be expressed as assumptions and guarantees in the GR(1) fragment. Moreover, our work shows that the remaining 3 patterns are indeed not expressible as assumptions or guarantees in the GR(1) fragment by our approach.

To answer the third question, our pattern representation in GR(1) requires at most 3 auxiliary variables per pattern instance. This gives an upper bound for the complexity of a GR(1) synthesis problem where patterns are used as assumptions or guarantees. Note that this is a very satisfying result, since based on the translation via a DBW, one could expect in the worst case an exponential number of auxiliary variables per pattern.

References

[Bl12] Bloem, Roderick; Jobstmann, Barbara; Piterman, Nir; Pnueli, Amir; Sa'ar, Yaniv: Synthesis of Reactive(1) Designs. J. Comput. Syst. Sci., 78(3):911–938, 2012.

[DAC99] Dwyer, Matthew B.; Avrunin, George S.; Corbett, James C.: Patterns in Property Specifications for Finite-State Verification. In: ICSE. ACM, pp. 411–420, 1999.

[KFP09] Kress-Gazit, Hadas; Fainekos, Georgios E.; Pappas, George J.: Temporal-Logic-Based Reactive Mission and Motion Planning. IEEE Trans. Robotics, 25(6):1370–1381, 2009.

[MR15a] Maoz, Shahar; Ringert, Jan Oliver: GR(1) Synthesis for LTL Specification Patterns. In: ESEC/FSE. ACM, pp. 96–106, 2015. Supporting materials: `http://smlab.cs.tau.ac.il/syntech/patterns/`.

[MR15b] Maoz, Shahar; Ringert, Jan Oliver: Synthesizing a Lego Forklift Controller in GR(1): A Case Study. In: Proc. 4th Workshop on Synthesis (SYNT), co-located with CAV. 2015.

[MS12] Maoz, Shahar; Sa'ar, Yaniv: Assume-Guarantee Scenarios: Semantics and Synthesis. In: MODELS. volume 7590 of LNCS. Springer, pp. 335–351, 2012.

[PPS06] Piterman, Nir; Pnueli, Amir; Sa'ar, Yaniv: Synthesis of Reactive(1) Designs. In: VMCAI. volume 3855 of LNCS. Springer, pp. 364–380, 2006.

[PR89] Pnueli, Amir; Rosner, Roni: On the Synthesis of a Reactive Module. In: POPL. ACM Press, pp. 179–190, 1989.