Towards Explainable Controller Synthesis and Reinforcement Learning for Cyber-Physical Systems

Joel Greenyer[1]

1 Future CPS and Understanding their Behavior

In domains like transportation and automation, Cyber-Physical Systems (CPS) control more and more complex and critical tasks. The complexity of the controlled processes, user interactions, the distributed nature of the systems, and the uncertainty in the environments are just some of the factors that make CPS development increasingly challenging. This complexity must be met with more high-level development techniques that raise the abstraction level from imperative programming or state-based modeling towards modeling specifications or goals on a level that corresponds more closely to how humans conceive requirements; this could be scenarios [Gr17], temporal logic specifications, or controlled natural languages; controller synthesis and reinforcement learning techniques will then construct or learn the executable software that exhibits the desired behavior. However, when the system behavior is constructed algorithmically, and even changes at run-time, this creates new challenges for comprehending the system behavior—at design-time and at run-time.

From our experience with game-based controller synthesis (CS) from scenario-based specifications[Co15, BGS15, Gr17], we found that it can be very difficult to understand specification inconsistencies when CS is unable to construct an implementation (valid control strategy). It can also be that issues remain in the specification, but CS is still able to construct an implementation (typically due to over-optimistic assumptions).

In reinforcement learning (RL), there is a similar challenge: Very often one does not know whether the system can enter states where it will struggle or fail to reach its goals, or whether it may choose a strategy (called *policy* in RL) that is only locally optimal (for example a robot that always cleans some dirt in the kitchen instead of cleaning the whole apartment).

[1] Leibniz Universität Hannover, Software Engineering Group, Welfengarten 1, 30167 Hannover, Germany greenyer@inf.uni-hannover.de

cc by nc

2 Explainability at Design-Time and Run-Time

We ultimately desire systems that are able to explain their behavior, misbehavior, struggles, or other aspects of interest—at design-time and run-time. But how could such (self-)explanation capabilities look like? First of all, the question is what kinds of questions human stakeholders need answers to? From our experience, some examples are "Can you (the system) reach a certain state from another / the current state?" (Reachability), "What action(s) will you perform under certain conditions?", "Which goals/requirements cause you to choose these actions in a certain state?", "What is your goal in a certain state?", "Under what conditions do you perform certain actions?", "When are you struggling to achieve a goal?", or "How can the environment/user help the system to achieve a certain goal?"

Such questions must be expressible in a structured query language, and we require an engine that computes answers to these queries. On what basis should the responses be computed? They should be computed, first of all, based on a synthesized or learned strategy, but also based on the specification or goal models, or other associated design artifacts. At run-time, also a log of past behavior can be considered. The answers must be as concise as possible. For example, queries that result in sets of states should summarize the defining features of the states in the set. Questions that result in (sets of) traces, cycles, or (sub-)strategies should also summarize their defining features, highlight relevant events, commonalities, or key decision points. Eventually, human-friendly front-ends are required that allow users to refine queries, or interactively simulate certain behaviors of the system.

3 Research Directions

Achieving the above requires research in multiple directions: (1) on algorithms for finding and filtering certain aspects from traces or (counter-)strategies. Some work already exist in this direction, e.g., [KMR17]; (2) on techniques for rendering such behavioral aspects as higher-level models, possibly scenarios; (3) on program/specification repair [SGH17].

Extending these ideas in the direction of RL is interesting. Usually learned policies in RL are represented as state-action matrices, or various abstraction techniques are employed, including neural-networks that classify states. One question is whether higher-level, explanation models can be learned, or initial design models be refined, in the learning process, so that answers to queries as described above can be computed. Where states are classified using neural networks, techniques would be necessary to extract their defining features in a comprehensible form. Reinforcement learning is successfully applied where no comprehensive model of the system's behavior or environment exist (model-free learning), but there exist approaches that combine model-free learning and model-based learning or planning (e.g. the dyna principle, [Su91]) for increased efficiency—we could similarly combine model-free learning with the learning of explanation models, which could then serve to answer queries as above at run-time.

References

[BGS15] Brenner, Christian; Greenyer, Joel; Schäfer, Wilhelm: On-the-fly Synthesis of Scarcely Synchronizing Distributed Controllers from Scenario-Based Specifications. In (Egyed, Alexander; Schaefer, Ina, eds): Fundamental Approaches to Software Engineering (FASE 2015), volume 9033 of Lecture Notes in Computer Science, pp. 51–65. Springer Berlin Heidelberg, 2015.

[Co15] Cordy, Maxime; Davril, Jean-Marc; Greenyer, Joel; Gressi, Erika; Heymans, Patrick: All-at-once-synthesis of Controllers from Scenario-based Product Line Specifications. In: Proceedings of the 19th International Conference on Software Product Line. SPLC '15, ACM, New York, NY, USA, pp. 26–35, 2015.

[Gr17] Greenyer, Joel; Gritzner, Daniel; Gutjahr, Timo; König, Florian; Glade, Nils; Marron, Assaf; Katz, Guy: ScenarioTools – A tool suite for the scenario-based modeling and analysis of reactive systems. Science of Computer Programming, 149(Supplement C):15 – 27, 2017. Special Issue on MODELS'16.

[KMR17] Kuvent, Aviv; Maoz, Shahar; Ringert, Jan Oliver: A Symbolic Justice Violations Transition System for Unrealizable GR(1) Specifications. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2017, ACM, New York, NY, USA, pp. 362–372, 2017.

[SGH17] Schmelter, D.; Greenyer, J.; Holtmann, J.: Toward Learning Realizable Scenario-Based, Formal Requirements Specifications. In: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), 4th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). pp. 372–378, Sept 2017.

[Su91] Sutton, Richard S.: Dyna, an Integrated Architecture for Learning, Planning, and Reacting. SIGART Bull., 2(4):160–163, July 1991.