X-by-Construction
Correctness Meets Probability

Maurice H. ter Beek[1(✉)], Loek Cleophas[2,5(✉)], Axel Legay[3], Ina Schaefer[4], and Bruce W. Watson[5,6]

[1] ISTI–CNR, Pisa, Italy
maurice.terbeek@isti.cnr.it
[2] TU Eindhoven, Eindhoven, The Netherlands
l.g.w.a.cleophas@tue.nl
[3] UC Louvain, Louvain-la-Neuve, Belgium
axel.legay@uclouvain.be
[4] TU Braunschweig, Braunschweig, Germany
i.schaefer@tu-braunschweig.de
[5] Stellenbosch University, Stellenbosch, South Africa
bwwatson@sun.ac.za
[6] CAIR, Stellenbosch, South Africa

Abstract. In recent years, researchers have started to investigate X-by-Construction (XbC) as a refinement approach to engineer systems that by-construction satisfy certain non-functional properties, beyond correctness as considered by the more traditional Correctness-by-Construction (CbC). In line with increasing attention for fault-tolerance and the use of machine-learning techniques in modern software systems, in which even correctness is hard to establish, this track brings together researchers and practitioners that are interested in XbC in particular in the setting of probabilistic properties.

1 Motivation and Aim

Correctness-by-Construction (CbC) approaches the development of software (systems) as a true form of Engineering, with a capital 'E'. CbC advertises a step-wise refinement process from specification to code, ideally by CbC design tools that automatically generate error-free software (system) implementations from rigorous and unambiguous specifications of requirements. Afterwards, testing only serves to validate the CbC process rather than to find bugs. (Of course, bugs might still be present outside the boundaries of the verified system: in libraries, compilers, hardware, the CbC design tools themselves, etc.)

A lot of progress has been made in this domain, implying it is time to look further than correctness and investigate a move from CbC to *XbC*, i.e., by considering also *non-functional properties*. XbC is thus concerned with a step-wise refinement process from specification to code that automatically generates software (system) implementations that *by-construction* satisfy specific non-functional

T. Margaria and B. Steffen (Eds.): ISoLA 2020, LNCS 12476, pp. 211–215, 2020.
https://doi.org/10.1007/978-3-030-61362-4_11

properties concerning security, dependability, reliability or resource/energy consumption, etc. A track on XbC was organised at ISoLA 2018 [2], in turn building on an ISoLA 2016 track focusing on the combination of post-hoc verification with CbC [3]. The 2018 XbC track brought together researchers and practitioners interested in CbC and the promise of XbC, with a particular emphasis on security-by-construction.

Building on the highly successful ISoLA 2018 track, the aim of this track is to once again bring together researchers and practitioners that are interested in CbC and XbC. In line with the growing attention to fault-tolerance (thanks to increasingly common failures in hardware and software) and the increasing use of machine-learning techniques in modern software systems—in both of these contexts, *guaranteed properties* are hard to establish—we particularly emphasise XbC in the setting of probabilistic properties.

We therefore invited researchers and practitioners working in the following communities to share their views on (moving from CbC to) XbC:

- People working on system-of-systems, who address modelling and verification (correctness, but also non-functional properties concerning security, reliability, resilience, energy consumption, performance, and sustainability) of networks of interacting legacy and new software systems, and who are interested in applying XbC techniques in this domain in order to prove—potentially probabilistic—non-functional properties of systems-of-systems by construction (from their constituent systems satisfying these properties).
- People working on quantitative modelling and analysis, e.g., through probabilistic or real-time systems and probabilistic or statistical model checking, in particular in the specific setting of dynamic, adaptive or (runtime) reconfigurable systems with variability. These people work on lifting successful formal methods and verification tools from single systems to families of systems, i.e., modelling and analysis techniques that need to cope with the complexity of systems stemming from behaviour, variability, and randomness—and which focus not only on correctness but also on non-functional properties concerning safety, security, performance, or dependability properties. As such, they may be interested in applying XbC techniques in this domain to prove non-functional properties of families of systems by construction (from their individual family members satisfying these properties).
- People working on systems involving components that employ machine-learning (ML) or other artificial-intelligence (AI) approaches. In these settings, models and behaviour are typically dependent on what is learned from large data sets, and may change dynamically based on yet more data being processed. As a result, guaranteeing properties (whether functional or non-functional ones) becomes hard, and probabilistic reasoning needs to be applied instead with respect to such properties for the components employing ML or AI approaches, and as a consequence, for systems involving such components as well.
- People working on generative software development, who are concerned with the automatic generation of software from specifications given in general for-

mal languages or domain-specific languages, leading to families of related software (systems). Also in this setting, the emphasis so far has typically been on functional correctness, but the restricted scope of the specifications—especially for domain-specific languages—may offer a suitable ground for reasoning about non-functional properties, and for using XbC techniques to guarantee such properties.

- ☑ People working on systems security, privacy, and algorithmic transparency and accountability, who care about more than correctness. The application of XbC techniques could provide certain guarantees from the outset when designing critical systems. It could also enforce transparency when developing algorithms for automated decision-making, in particular those based on data analytics—thus reducing algorithmic bias by avoiding opaque algorithms.

2 Contributions

In their keynote contribution, McIver and Morgan [8] describe a correct-by-construction proof method for probabilistic programming. It is based on their probabilistic extension pGCL of Dijkstra's Guarded-Command Language, which allows to describe program correctness by a generalisation of Hoare logic that includes quantitative analysis, and to develop programs by refinement such that both functional and probabilistic properties are preserved. They demonstrate how to apply their method by deriving a fair-coin implementation of any given discrete probability distribution in a systematic, layered way such that the reasoning in each layer does not depend on earlier layers nor affect later ones. Moreover, in the special case of simulating a fair die, the authors show how one final correctness-preserving step allows them to obtain Knuth and Yao's optimal die-roll algorithm.

In the context of probabilistic component-based systems that interact via synchronised execution of shared actions, Baier et al. [1] present different notions of (component) suitability, results on their decidability for restricted classed, and associated algorithmic analysis procedures. The basic notion of suitability is provided through threshold suitability that determines whether each one of the given quantitative properties exceeds a given threshold, while weighting quantitative properties leads to a quantitative measure of degrees of suitability. The applicability of the resulting notions of quantitative suitability analysis is illustrated with a case study of vehicle components with features for different road conditions, which is particularly appealing due to its feature-oriented nature.

Fahrenberg and Legay [4] consider the notion of behavioural specification theories, giving an overview of the underlying theoretical concepts, as well as various kinds of behavioural specification theories, with examples for most of them. To make the connection to this specific XbC track, they include three different behavioural specification theories for modelling real-time and probabilistic systems. The survey emphasises commonalities and differences between the various specification and modelling formalisms in such theories, leading to a taxonomy in the form of a table showing the different behavioural specification theories and their properties.

Jaeger et al. [6] consider the resolution of control problems under uncertainty and continuous domains. For doing so, they rely on finite-state imprecise Markov decision processes that can be used to approximate the behaviour of these infinite models. The authors address two questions. First, they investigate what kind of approximation guarantees are obtained when the process is approximated by finite-state approximations induced by increasingly fine partitions of the continuous state space. They show that for cost functions over finite time horizons the approximations become arbitrarily precise. Second, they use imprecise Markov decision process approximations as a tool to analyse and validate cost functions and strategies obtained by reinforcement learning. Finally, the authors compare this constructive process with classical learning-based solutions.

Könighofer et al. [7] present an overview of shield synthesis approaches in order to ensure that policies learned using reinforcement learning do not make incorrect or unsafe actions. They discuss the approaches of pre-shielding where the set of possible actions an agent can take are filtered before decision making, to only include correct actions and post-shielding where the selected action after decision making is corrected in case it is incorrect. The paper reviews existing work for shield synthesis for temporal, probabilistic and timed properties and presents examples and evaluation results for each of the reviewed approaches.

Performance is an important non-functional property that, being related to metrics like response time and throughput, directly affects end-user perception of the quality of a software system. Therefore, controlling a software system's performance is an important endeavour in today's engineering practice. Incerto et al. [5] argue that the performance-by-construction development paradigm by which executable code carries some kind of performance guarantees, as opposed to the current practice in software engineering where performance concerns are left to later stages of the development process by means of profiling or testing, needs to support techniques that are probabilistic in nature, leveraging accurate models for the analysis. To this aim, they present a literature review and a classification of methods that can form the basis of such performance-by-construction development approaches, focussing on methods where performance information is extracted directly from the code. This is a line of research that has apparently been less explored by the software performance engineering community. They conclude by discussing limitations of the state of the art.

References

1. Baier, C., et al.: Components in probabilistic systems: suitable by construction. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 240–261. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_13
2. ter Beek, M.H., Cleophas, L., Schaefer, I., Watson, B.W.: X-by-construction. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11244, pp. 359–364. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03418-4_21
3. ter Beek, M.H., Hähnle, R., Schaefer, I.: Correctness-by-construction and post-hoc verification: friends or foes? In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 723–729. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_51

4. Fahrenberg, U., Legay, A.: Behavioral specification theories: an algebraic taxonomy. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 262–274. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_14
5. Incerto, E., Napolitano, A., Tribastone, M.: Inferring performance from code: a review. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 307–322. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_17
6. Jaeger, M., Bacci, G., Bacci, G., Larsen, K.G., Jensen, P.G.: Approximating euclidean by imprecise Markov decision processes. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 275–289. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_15
7. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 209–306. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_16
8. McIver, A., Morgan, C.: Correctness by construction for probabilistic programs. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020, Part I. LNCS, vol. 12476, pp. 216–239. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_12