Demo: DryVR 2.0 - A tool for verification and controller synthesis of black-box cyber-physical systems

Bolun Qi, Chuchu Fan, Minghao Jiang and Sayan Mitra
Coordinated Science Laboratory, University of Illinois at Urbana Champaign
{bolunqi2,cfan10,mjiang24,mitras}@illinois.edu

ABSTRACT

We present a demo of DryVR 2.0, a framework for verification and controller synthesis of cyber-physical systems composed of black-box simulators and white-box automata. For verification, DryVR 2.0 takes as input a black-box simulator, a white-box transition graph, a time bound and a safety specification. As output it generates over-approximations of the reachable states and returns "Safe" if the system meets the given bounded safety specification, or it returns "Unsafe" with a counter-example. For controller synthesis, DryVR 2.0 takes as input black-box simulator(s) and a reach-avoid specification, and uses RRTs to find a transition graph such that the combined system satisfies the given specification.

ACM Reference Format:
Bolun Qi, Chuchu Fan, Minghao Jiang and Sayan Mitra. 2018. Demo: DryVR 2.0 - A tool for verification and controller synthesis of black-box cyber-physical systems. In *HSCC '18: 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), April 11–13, 2018, Porto, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3178126.3187008

1 INTRODUCTION

For many cyber-physical systems (CPS), closed form mathematical models are either not available or are outside the reach of existing formal analysis tools such as [2, 3, 5]. DryVR [4] aims to perform rigorous analysis of such systems by combining formal reasoning and probabilistic sensitivity analysis. The system is viewed as a combination of a black-box simulator for generating trajectories and a white-box transition graph specifying the mode switches. The DryVR verification algorithm combines learning the sensitivity of the black-box by exercising the simulator from different states, and the formal reasoning and reachability analysis on the transition graph. The tool has been successfully used to analyze a realistic models of autonomous spacecraft [1] and engine control benchmarks [6].

Additionally, DryVR 2.0 supports GraphSearch() – a functionality that automatically finds a transition graph for meeting a given reach-avoid specification. That is, given the inputs as a black-box simulator, an initial set, unsafe regions, target region, and a time bound T, GraphSearch() returns a transition graph which defines a sequence of mode switches such that all executions of the resulting system reach the target within time T, while maintaining a safe distance from the unsafe regions. The synthesis algorithm is based on the classical rapidly-exploring random trees (RRT) [7] and searches the space of transition graphs effectively. RRT-type algorithms have been successfully applied to solve planning problems for dynamical systems. The GraphSearch() function makes this approach now available to systems with black-box models for planning without complete model information. This function also frees-up users from tediously creating transition graph models manually. Results on existing controller synthesis benchmark models show that this new functionality is a promising starting point for DryVR to become a comprehensive verification and synthesis toolbox for practical CPS.

This work is supported by National Science Foundation's research grants NSF CAREER 1054247 and NSF CSR 1422798.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HSCC '18, April 11–13, 2018, Porto, Portugal
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5642-8/18/04.
https://doi.org/10.1145/3178126.3187008

2 DRYVR 2.0 MODELS AND VERIFICATION ALGORITHM

Model and semantics. The input to DryVR 2.0 is a hybrid system described as a combination of black-box simulators and white-box transition graphs. That is, a tuple $\mathcal{H} = \langle \mathcal{L}, \Theta, G, \mathcal{TL} \rangle$, where (a) $\mathcal{L}$ is a finite set of names for the discrete modes; (b) $\Theta \subseteq \mathbb{R}^n$ is a compact set of initial states; (c) $G = \langle \mathcal{L}, \mathcal{V}, \mathcal{E}, elab \rangle$ is a transition graph— each vertex $v \in \mathcal{V}$ corresponds to a mode $l \in \mathcal{L}$, with edges connecting different vertices and *elab* defining the guards and reset functions; (d) Finally, $\mathcal{TL}$ is a set of deterministic trajectories as generated by the black-box simulators. For each mode $l \in \mathcal{L}$ and an initial state $x(0)$, the simulator generates a simulation of the continuous trajectory $x_l(t)$ in mode l.

A *state* of $\mathcal{H}$ is a point in $\mathbb{R}^n \times \mathcal{L}$. Given an initial state $(x(0), l_{\text{init}})$, an *execution* of the system is the sequence of trajectories $\text{exec}(x(0), l_{\text{init}}) = \langle x_{l_1}(t), l_1 \rangle \cdots \langle x_{l_k}(t), l_k \rangle$ such that (1) $l_1 = l_{\text{init}}$, $x_{l_1}(0) = x(0)$, (2) $l_1, \cdots, l_k$ follow the transition graph, and (3) for each $i > 1$, there is an edge $e \in \mathcal{E} : v_{i-1} \to v_i$ with the edge label *elab*, such that v_{i-1} defines the mode l_{i-1} and v_i defines the mode l_i, $x_{l_{i-1}}(t)$ stops at a state where a guard on *elab* is met, and $x_{l_i}(t)$ starts with a state defined by the corresponding reset function of *elab*.

For verification, in addition to $\mathcal{H}$, we also supply a time bound T and safety specification (unsafe region $\mathcal{U}$), and DryVR 2.0 either returns "Safe" and an over-approximation of the reach set, or it returns "Unsafe" and a counter-example execution. It is also possible for DryVR to return "Unknown" when the refinement limit specified by the user is reached.

Verification algorithm. The core analysis idea in DryVR is simulation and sensitivity based over-approximation of

bounded reachable sets. (a) For any vertex v of G, it generates a group of numerical simulations from the initial set Θ_v of v by calling the black-box simulator of the corresponding mode. (b) Using these simulations, it learns the sensitivity (discrepancy function) of this mode. (c) These simulations are bloated by the factor given by the discrepancy function such that the bloated tubes over-approximate the reachable states from Θ_v. The over-approximations are represented using a sequence of hyper-rectangles for small time intervals. The probabilistic algorithm for learning discrepancy in step (b) has a probabilistic guarantee such that for any $\epsilon, \delta > 0$, if the number of simulation points for the trajectories is greater than $[\frac{1}{\epsilon} \ln \frac{1}{\delta}]$ then, with probability $\geq 1 - \delta$, the learned discrepancy function has an error that is $< \epsilon$ (see Proposition 4 of [4]). The overall verification algorithm then uses a depth first search algorithm to cover all the vertices of the transition graph, using the learned discrepancy for each vertex as described above. If the learned discrepancy function is correct, the overall verification algorithm is proven to be sound and relatively complete.

3 SYNTHESIS FOR REACH-AVOID SPECIFICATIONS USING RRT

For controller synthesis of DryVR, the input to the function GraphSearch() is (1) the list of modes $\mathcal{L}$, the black-box simulator $\mathcal{TL}$, and the initial set Θ (that is, $\mathcal{H}$ without G), (2) an initial mode $l_{\text{init}} \in \mathcal{L}$, (3) the unsafe regions $\mathcal{U}$, (4) a target region $\mathcal{T}$, (5) a time bound T, and (6) a minimum dwell time in each mode t_{min}. The output is a transition graph G such that for the resulting hybrid system $\mathcal{H} = \langle \mathcal{L}, \Theta, G, \mathcal{TL} \rangle$, all executions starting from any $x(0) \in \Theta$, will reach $\mathcal{T}$ within bounded time T and will not intersect with $\mathcal{U}$. DryVR uses RRT algorithm [7] to search for transition graph G. It builds a tree that attempts to reach the target without colliding with the obstacles by exploring possible sequences of mode switches. For each tree node, it still uses the learning-based reachability analysis module for the black-box simulator within each mode.

The synthesis algorithm is sound if the learned discrepancy function is correct. That is, any execution of the hybrid system $\mathcal{H}$ with the synthesized graph G reaches the target region $\mathcal{T}$ without colliding with the unsafe regions $\mathcal{U}$. The algorithm will return "Fail" if the random tree has grown to a full k-ary tree with depth $[T/t_{min}]$ before a transition graph is found, where k is the number of candidate time intervals at each node, and such k can be specified by the user. However, if the algorithm returns "Fail", it does not mean that there is no such graph.

Example. Consider a black-box robot model with eight modes "Up", "Down", "Left", "Right", "UpLeft", "UpRight", "DownLeft", "DownRight". Each mode gives the direction of the robot it tries to move towards. The robot is placed in 5x5 arena as shown in Figure 1 (left). Starting from the initial set (the yellow region), the robot should avoid obstacles (the red regions or outside the arena) and reach the target (the green region), within 10 seconds. Figure 1 (right) shows synthesized transition graph. The reach set of the robot running with the synthesized graph is shown as the blue tube.

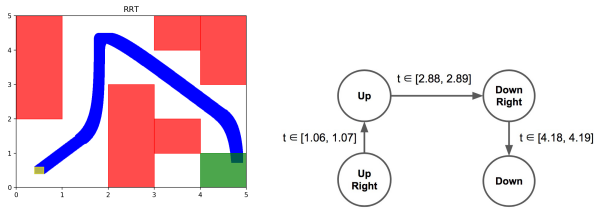

Figure 1: Controller synthesis of a black-box robot in maze with reach-avoid specification.

4 TOOLS DETAILS AND DEMONSTRATION

DryVR 2.0 is an open source project available at https://github.com/qibolun/DryVR_0.2. It comes with 32 new examples illustrating the verification and synthesis capabilities. A user guide is provided at http://dryvr-02.readthedocs.io/en/latest/. It has a command-line user interface to parse user-supplied input configuration files.

To give the audiences a closer look at DryVR 2.0, we would like to demonstrate the following features:

(1) Connecting black-box simulators with DryVR to model a combined hybrid system.

(2) Format of the input files required for DryVR 2.0 verification and controller synthesis.

(3) Plotting functions for visualizing the verification and controller synthesis results, including reachable states of several variables with respect to a given variable or time.

(4) Retrieving the textual representation of reachable states generated by verification, or the white-box transition graph generated by the controller synthesis function GraphSearch().

We introduced DryVR 2.0 with multiple updates to make it a much more powerful framework for verification and controller synthesis of hybrid systems combing black-box simulators and white-box transition graphs. With functionality and usability enhancements, we believe that DryVR 2.0 is now ready for a broader academic community and hence presenting a demo at HSCC would be a great opportunity for introducing it.

REFERENCES

[1] Nicole Chan and Sayan Mitra. 2017. Verified hybrid LQ control for autonomous spacecraft rendezvous. In *CDC*.
[2] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *CAV*. Springer, 258–263.
[3] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. 2015. C2E2: A Verification Tool for Stateflow Models.. In *TACAS*. 68–82.
[4] Chuchu Fan, Bolun Qi, Sayan Mitra, and Mahesh Viswanathan. 2017. DryVR: Data-Driven Verification and Compositional Reasoning for Automotive Systems. In *CAV, Part I*. 441–461. https://doi.org/10.1007/978-3-319-63387-9_22
[5] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *CAV*. Springer, 379–395.
[6] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Powertrain control verification benchmark. In *HSCC*. ACM, 253–262.
[7] Steven M LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. (1998).