BAS Benchmark Models

Nathalie Cauchi, Alessandro Abate
Department of Computer Science, University of Oxford

Overview

The BASBenchmark script repository contains the main files needed to (i) construct and perform simulation a model for a required BAS setup based on the components making it up, (ii) perform simulation of the whole BAS setup described in [2] and (iii) simulate the models described within each case study. One can easily add more components to the benchmark and compositionally simulate different BAS structures. The core file to build the underlying component models is called `CreateModel.m`. This file allows you to build symbolic models based on their type. Models can be either (i) linear time invariant or (ii) bilinear time invariant and either (i) deterministic or (ii) stochastic and all can have additive disturbances. These can all be defined using the class `CreateModel` where one inputs the continuous time system matrices of the underlying model. The class automatically performs discretisation of the specified models.

A high-level description of the BASBenchmark code structure is given in Figure 1. The benchmarks follow the form:

1. *Load parameters*: The used parameters are given within the file `BASParameters.m`. Unknown parameters where identified using data collected within the BAS Set-up at the Department of Computer Science. The user may opt to update/add parameters as needed. We can also provide data from our set-up for performing system identification of the parameters.

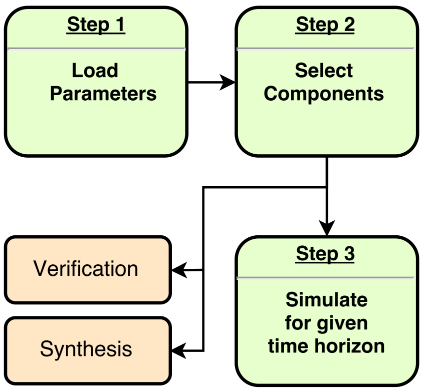

Figure 1: High-level structure of benchmark models structure. Green boxes are tasks which can be directly performed using the provided scripts. The orange boxes can be performed by performing adaptation to the current scripts.

2. *Select components*: Within the script repository, one will find scripts named in the form of `XXModel.m`, where `XX` corresponds to the specific component. These files are used to instantiate models representing the components. Based on the BAS setup one wishes to construct one should define a model for each of the BAS component (cf. Lines 17 - 26 in `WholeBAS_setup.m`). This builds a symbolic model for each of the components.

   Note, more components can be added by defining a new `XXModel.m` file and use the `CreateModel.m` file to compose the symbolic model for that specific component.

3. *Simulate for given time horizon*: Once the models for the individual components are defined, one should define the input signals and the control strategy to be used. The BAS setup is then simulated by connecting the different components based on input-output to get the whole BAS setup and (1) for static models simply provide the inputs to the corresponding component model or (2) for dynamical models call the `runModel` function found within the `createModel` class. This function simulates the dynamic models given the current state, inputs, disturbances and time horizon. Finally, once all the simulation is performed the trajectories are plotted by calling the `plotFigures.m` function.

Once, the components are selected and the whole model is constructed by connecting the input-outputs, one can also perform verification tasks such as reachability analysis or policy synthesis. These tasks are currently performed external to the provided benchmarks. In the next subsections, we'll give a brief overview of how one can use these models to perform verification or synthesis tasks.

Verification & Synthesis

We consider reachability analysis as the main verification task to be performed. For deterministic models this task can be performed using either Axelerator [1] or SpaceEx [3], while for stochastic models probabilistic reachability analysis can be performed using FAUST^2 [4] (can also be used to perform synthesis of control policies). To interface with Axelerator one needs to convert the matrices of the whole model into polyhedral sets where constrains on the states of the model correspond to actual physical feasible values the states can undertake. Once the dynamics are defined, the model can be directly fed into Axelerator. One should note that Axelerator only works with discrete-time linear time invariant models. On the other hand, to interface with SpaceEx one needs to convert the model in sX format (similar to XML) and write the original continuous time models within the `flow` field. An example of a model in sX format is given in `CaseStudy3_Hybrid.m`. SpaceEx can cater for hybrid deterministic models where the underlying dynamics are described in continuous time.

In the case of stochastic models, the generated BAS model needs to first be converted into an equivalent general Markov decision process. This results in defining a stochastic transition kernel for each time step, which is then fed into FAUST^2 such that either synthesis or probabilistic reachability can be performed.

References

[1] Dario Cattaruzza, Alessandro Abate, Peter Schrammel, and Daniel Kroening. Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration. In *International On Static Analysis*, pages 312–331. Springer, 2015.

[2] Nathalie Cauchi and Alessandro Abate. Benchmarks for cyber-physical systems: A modular model library for buildings automation. In *International Conference on Cyber-Physical Systems, 2018.* Under review.

[3] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[4] Sadegh Esmaeil Zadeh Soudjani, Caspar Gevaerts, and Alessandro Abate. Faust 2: Formal Abstractions of Uncountable-STate STochastic processes. In *TACAS*, volume 15, pages 272–286, 2015.