EPiC Series in Computer Science
Volume 34, 2015, Pages 73–79
ARCH14-15. 1st and 2nd International Workshop on
Applied veRification for Continuous and Hybrid Systems

Benchmark: Stratified Controllers of Tank Networks *

Stanley Bak[1], Sergiy Bogomolov[2], Marius Greitschus[3], and Taylor T. Johnson[4]

[1] Air Force Research Laboratory, Rome, NY, USA
stanleybak@gmail.com
[2] IST Austria, Austria
sergiy.bogomolov@ist.ac.at
[3] University of Freiburg, Germany
greitsch@informatik.uni-freiburg.de
[4] University of Texas at Arlington, USA taylor.johnson@gmail.com

Abstract

We present a new model of a tank network used to transfer liquid. Tanks are connected by channels. The throughput velocity of every particular channel is governed by the controller. We consider a special class of *stratified* controllers which are organized in several phases. Every phase can be further partitioned into multiple options. This structure makes it easy to generate a variety of benchmark instances ranging in the size, branching factor and generally analysis complexity. We provide a flexible benchmark generator for this class of benchmarks and a sample benchmark suite built by the generator. Finally, we use the Hyst model transformation framework to convert the original model into the formats of several reachability tools.
Category: academic **Difficulty:** medium

1 Context and Origins

The area of hybrid automata [10] has undergone a rapid development in the last decade. In particular, several extensible frameworks [7, 4] for the analysis of hybrid automata are now available. Such frameworks provide an excellent environment to develop new analysis algorithms in a timely manner. However, an extensive testing suite is necessary to ensure the correctness of the developed algorithms and furthermore evaluate their performance against other available algorithms and tools. Unfortunately, only a few benchmark suites are publicly available [6]. Furthermore, the available benchmark instances are hard-coded and cannot be easily adjusted. In this paper, we suggest a new extensible class of benchmarks inspired by the work of Frehse et al. [8].

In addition, we provide a benchmark generator that automatically builds SpaceEx [7] models based on the provided benchmark instance description and a sample benchmark suite built

*v0.1, 2015-03-01, DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA #88ABW-2015-1313, 20 MAR 2015) by the generator[1]. This benchmark class is of particular interest because of its scalability: benchmark instances can be scaled both in discrete and continuous dimensions. We can generate controllers that exhibit multiple branching points and several types of continuous dynamics. Therefore, by increasing the branching factor and varying the continuous dynamics of the controller we can easily adjust the complexity of benchmark instances.

Finally, we leverage the Hyst model transformation tool [2] in order to convert the generated model into a number of formats compatible of recent hybrid systems reachability tools. This gives tool developers the option of extending Hyst to support their tool, rather than manually converting each specific benchmark to their tool's format. Furthermore, Hyst features like automatic automaton flattening can be leveraged in order to ease benchmark conversion.

We base our work on the switched buffer network benchmark [8]. In order to provide benchmark instances that scale both in discrete and continuous dimensions, we consider controllers that exhibit multiple branching points and several types of continuous dynamics. Therefore, by increasing the branching factor and varying the continuous dynamics of the controller we can easily adjust the complexity of benchmark instances. We provide a benchmark suite in the SpaceEx [7] format, and through Hyst, also analyze the benchmark in Flow* [4], HyCreate2 [1], and dReach [9]. Of these, each of Flow*, dReach, and HyCreate2 have challenges when analyzing this benchmark class.

2 Benchmark Description

Our system consists of a network of tanks connected by channels. The liquid flows into the network through the *initial* tank. The controller adjusts the throughput rates of the channels in order to ensure the liquid delivery to the *sink* tank. We consider properties that reason over the fill level of the sink tank. The system can be seen as a composition of a plant (consisting of a number of components modeling tanks and channels) and a controller which governs the liquid flow throughout the network.

In the following, we describe the way the tanks and channels are modeled. The rate of change of the fill level f_T of a tank T, depends on the rates of inflow $v_{in\,i}$ and the rates of outflow $v_{out\,j}$ of the liquid, where $v_{in\,i}$ is the velocity at which the liquid flows into the tank of the i-th input channel, and $v_{out\,j}$ is the velocity at which the liquid flows out of the tank for the j-th output channel. Thus, the evolution of the fill level of the tank T is described by the differential equation $\dot{f_T} = \sum_i v_{in\,i} - \sum_j v_{out\,j}$, where i and j range over incoming and outgoing channels of T, respectively.

Here, we mainly focus on the controller structure. In particular, we consider a class of *stratified*, i.e., layered, controllers. In this setting, the controller iterates over a number of *phases*. The controller can open, close or generally modify the throughput values of network channels in every phase. Every phase has a particular duration. Furthermore, a phase can contain a number of *options* which allow for the fine-granular throughput adjustment in every phase. All the options in one phase agree on the channel throughput they modify, i.e., if a particular channel is to be opened in one option, this will also be the case in all other options. However, the resulting throughput velocity may vary among the options. In other words, the throughput adjustments in every phase are organized in *strata*. Those ideas are illustrated in Fig. 1.

[1] The benchmark generator and suite can be downloaded from http://swt.informatik.uni-freiburg.de/tool/spaceex/benchgen.

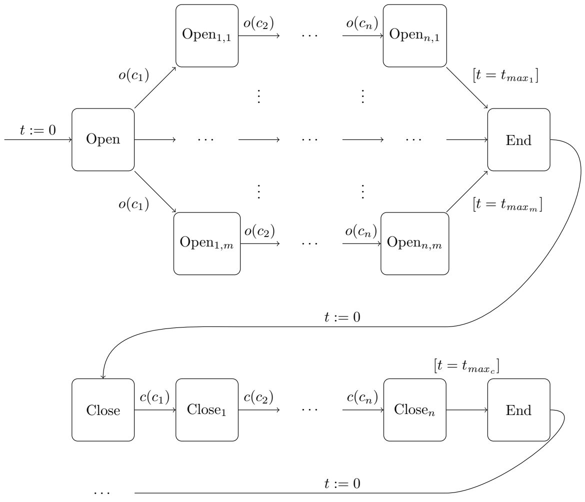


Figure 1: Controller consisting of "open" and "close" phases. The "open" phase has m options. Every option refers to n channels. The controller and the plant communicate through the shared labels $o(c_i)$ and $c(c_i)$ which correspond to opening and closing channel c_i, respectively. The "close" phase always have at most one option. Variable t measures the time spent in a phase.

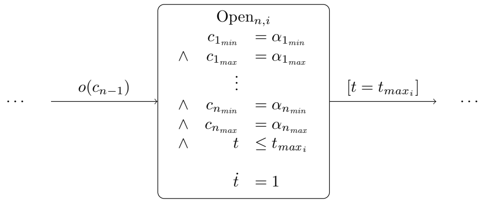

…
$o(c_{n-1})$
$\text{Open}_{n,i}$
$c_{1_{min}} = \alpha_{1_{min}}$
$\wedge\ c_{1_{max}} = \alpha_{1_{max}}$
⋮
$\wedge\ c_{n_{min}} = \alpha_{n_{min}}$
$\wedge\ c_{n_{max}} = \alpha_{n_{max}}$
$\wedge\ t \leq t_{max_i}$
$\dot{t} = 1$
$[t = t_{max_i}]$
…

Figure 2: Location $\text{Open}_{n,i}$ of the controller in the mode "No dynamics". This location finalizes the impact of the option i. In particular, the values of shared variables $c_{j_{min}}$ and $c_{j_{max}}$ are updated with the constant values $\alpha_{j_{min}}$ and $\alpha_{j_{max}}$, respectively. We enforce this update by encoding it as a part of the location invariant. This in turn impacts the channels which also refer to $c_{j_{min}}$ and $c_{j_{max}}$. Variable t measures the time spent in the phase which is reflected by the differential equation $\dot{t} = 1$.

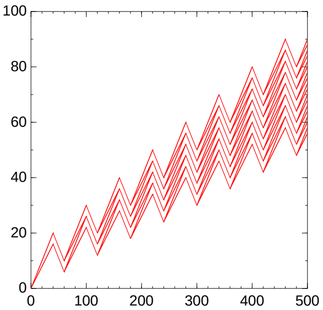
The system behavior crucially depends on the way the controller adjusts the channel throughput. We distinguish three modes:

(1) No dynamics: As soon as the channel is opened, its throughput v is given by the inequality $v_{min} \leq v \leq v_{max}$.

(2) Constant dynamics: The throughput is governed by the differential equation $\dot{v} = c$ for some constant c.

(3) Affine dynamics: The channel opens gradually with the opening speed decaying towards the target velocity v_{target}, where the throughput evolution is provided by the differential equation $\dot{v} = c(v_{target} - v)$ for a constant c.

Note that in case of "No dynamics" the throughput is changed instantaneously when the channel is opened, whereas the channel is opened gradually in the other cases. In Fig. 2,



```
# TANKS:
define tank "MainTank" capacity=10
define tank "SubTank1" capacity=10
define tank "LastTank" capacity=100 sink="true" output=0.5

# CHANNELS:
define channel "RootChannel" root="true" gen_flow=2
define channel "channel1"
define channel "channel2"

# CONNECTIONS:
define connection "RootChannel" target="MainTank"
define connection "channel1" source="MainTank" target="SubTank1"
define connection "channel2" source="SubTank1" target="LastTank"

# PHASES:
define phase "openphase"
define option "option1" duration=40
define option "option1" "RootChannel" action="open"
define option "option1" "channel1" action="open"
                lowest_throughput=1 highest_throughput=1
define option "option1" "channel2" action="open"
                lowest_throughput=1 highest_throughput=1

define option "option2" duration=40
define option "option2" "RootChannel" action="open"
define option "option2" "channel1" action="open"
                lowest_throughput=0.9 highest_throughput=0.9
define option "option2" "channel2" action="open"
                lowest_throughput=0.9 highest_throughput=0.9

define phase "close"
define option "option1" duration=20
define option "option1" "RootChannel" action="close"
define option "option1" "channel1" action="close"
define option "option1" "channel2" action="close"
```

Figure 3: Fill level evolution of the sink tank for the network consisting of four tanks with linear topology, i.e., initial tank, sink tank and two further tanks in between vs. time on the left hand side. On the right hand side, the definition file used by the generator to produce a SpaceEx model for this model is shown. The controller works in a cyclic manner, i.e., it proceeds with the "open" phase as soon as the "close" phase is over.

we explain the structure of the controller with no dynamics in more detail. The modes with constant and affine dynamics are built similarly by incorporating the differential equations reflecting the appropriate throughput dynamics. The treatment of continuous dynamics poses a major challenge for hybrid model checkers. By introducing those three modes, we provide benchmark instances with gradually increasing continuous dynamics complexity which in turn leads to increased verification effort.

Fig. 3 exemplifies the behavior of a sample tank network consisting of four tanks with linear topology, i.e., initial tank, sink tank and two further tanks in between assuming a controller with no dynamics.

3 Conversion

In order to make use of the proposed benchmark it is essential to be able to create a working model in the tool of choice. Since tools have different input formats, we extend and make use of the Hyst [2] tool. Hyst is a model transformation and translation tool, which takes input in the SpaceEx format and generates models in the formats of Flow* [4], HyCreate2 [1], and dReach [9]. For this work, the tool was extended by adding support for automaton flattening, as well as nondeterministic flows and guards, and urgent transitions (which are all used in the benchmark).

Additionally, in order to validate model import, we create a printer back to the SpaceEx format so the flattened and modified automaton could be re-run with SpaceEx to validate that the reachable set of states was unchanged during flattening. Using the sample network system previously used is given in Figure 3, the reachable set of states was confirmed to match, increasing confidence in the correctness of the flattening process. The file size during this process grew from 29 kB (original) to 445 kB (flattened).

Computing reachability for this benchmark is particularly hard for the other tools to which

Hyst can convert, as will be elaborated on shortly. To get some output, a minimal version of the benchmark was generated (2 tanks) and the reach set was computed in SpaceEx. The model was then flattened and converted to the format of Flow*, and the number of discrete transitions was restricted to 20. The resultant output with SpaceEx and Flow* is shown in Figure 4, and appears to overlap for both tools.

The generated benchmarks stress the supported tools in several ways. The largest issue with most of the tools was a lack of fixpoint detection when processing discrete transitions. In the benchmark, there is a large number of urgent modes in the flattened automaton which describe the discrete logic. These urgent modes contain cycles which lead to infinite discrete-jump loops (Zeno behavior) for tools which do not support fixpoint detection. In the HyCreate2 simulator, these loops would be entered and then never exit until the discrete jump limit was reached, all before any time elapsed in the simulation. This was also the reason why we needed to restrict the number of discrete jumps to get an output from Flow* in Figure 4. Using a larger jump bound in Flow* caused the number of possible paths to be so large that the computation ran into memory issues before an output was produced. Several regularization [11] model transformation passes were added to Hyst which attempted to eliminate this problem. One of the passes would enforce a maximum number of jumps per time interval. This did not solve the problem, as if the number of jumps was too low, then a time-progressing state would never be reached, and if the number was too high, the number of paths would remain intractable. Enforcing dwell times in non-urgent modes did not solve the problem, since it is the networks of urgent modes which caused problems. Enforcing dwell times on urgent modes changes the semantics, which led to models where all executions would end.

Another limitation discovered in the tools was that, since the generated mode names are created from a concatenation of the mode names in the subsystems, the names of the modes in the flattened automaton could get quite long. In Flow*, mode names longer than 100 characters were not accounted for, which required recompilation of the tool from source to fix. In HyCreate2, each mode and transition has a file generated which is compiled within the reachability engine. The filenames were too long for the filesystem, causing the OS to raise an error. A model transformation pass was added to Hyst which shortened mode names to overcome these limitations.

Since the number of variables (dimensions) was quite large, this led to poor performance for the mixed-face lifting approach employed by HyCreate2. In dReach, the number of discrete jumps must be specified *exactly* when checking for property violations. This is problematic because the model contains many urgent modes where no time elapses, although a discrete jump occurs. After flattening, it is difficult to manually reason about the exact number of jumps which should be considered. Finally, even the original model in SpaceEx needs to work around limitations of the SpaceEx input format. Specifically, the format only supports a single synchronization label per transition (which mandates using multiple successive urgent transitions if multiple labels are desired, leading to the networks of urgent modes). All of these limitations can be recast as desired enhancements to the corresponding tools, and their improvement can be demonstrated with the proposed benchmark.

The benchmark also led to enhancements in the Hyst model transformation tool. Enhancements were made to the tool to permit automated flattening of models, which was necessary since SpaceEx was the only tool that supported a networked automaton input format. Flattening has the effect of quickly growing the state space of the system, so model transformation passes were added to Hyst in order to trim the number of states generated. Two model transformation passes were added to eliminate unreachable modes. First, a pass was added which, based on the discrete transitions and initial set of states, would remove any states that are

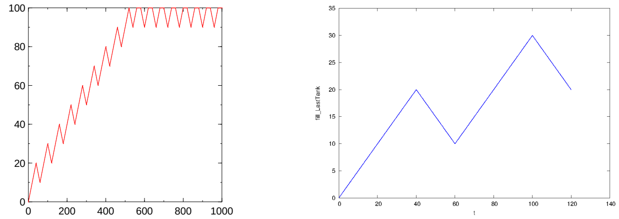


Figure 4: The reachable set of states for the minimal model in SpaceEx (left), and Flow* (right) appears similar, increasing confidence in the translation process. Notice that the Flow* result is over a significantly shorter time period, due to the explosion in runtime due to lack of fixpoint detection.

discretely unreachable. Second, a model transformation pass was added in Hyst which removes modes and associated transitions where the invariants are clearly unsatisfiable. This would come up in cases in the tank system, for example, where a tank was in the starved state (which means the tank is empty and its input flow is less than its maximum output flow), whereas the previous tank had an output flow that was greater than the first tank's maximum output flow. The composed invariant for such cases would have constraints like $x \in [0, 3]$ && $x = 5$, where x is the output of the tank, 3 is its maximum output rate, and 5 is the input rate from the previous tank. Such invariants can never be true, and these modes would be removed from the model. Further passes could be imagined which make use of a more powerful procedure to check if invariants are satisfiable, such as an SMT solver [5]. Additional features were added to Hyst to support nondeterministic flows, reset assignments, and the detection and conversion of urgent transitions, which are all present even in the simplest tank benchmark system. Passes were also created to perform regularization in order to try to overcome the problems with Zeno behaviors, which may be reused on other models in the future.

Even with the enhancements, the larger tank benchmarks stress the reachability tools and further improvements may be required before they can successfully be used to compute reachability for longer time periods. The flattened models contained thousands of states and transitions. Users of verification tools who wanted to evaluate the benchmark would be unlikely to accurately perform such a conversion by hand.

4 Outlook

We have presented a challenging benchmark study and a generation tool to create instances of the benchmark with different structure. We have also provided a conversion tool that can take the benchmark, flatten it, and convert it to the input format of various tools for analysis. Key challenges with running the different tools on the converted benchmark were then explored, providing direction for potential tool enhancements. Our benchmark suite can be used to test hybrid model checking algorithms with respect to multiple criteria:

(a) **Accuracy** - generally, it is more challenging to treat affine dynamics compared to piece-wise constant ones. By considering both options in our benchmarks, we enable the user to test and compare algorithms in different settings. Nonlinear models of dynamics arising from the physical tank could also be explored, but would require extensions to the benchmark generator.

(b) **Termination of reachability algorithms** - if no global time horizon is provided in the controller, a fix-point may not be reached. However, in our setting, we assume a given upper-bound on time that leads to the existence of a fix-point. Therefore, the benchmarks provide the ability to test different algorithms' termination conditions.

(c) **Guidance sensitivity** - the rich discrete structure makes our benchmark suite particularly appropriate to test algorithms for search guidance in the system state space, such as guided exploration methods [3].

(d) **Scalability** - the benchmark structure can easily be adjusted to a given structural complexity by introducing additional phases/options and varying controller dynamics.

5 Acknowledgments

This work was partly supported by the Air Force Office of Scientific Research (AFOSR) Systems and Software grant "Perpetual Model Validation", by the European Research Council (ERC) under grant 267989 (QUAREM), by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), and by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/).

References

[1] S. Bak. HyCreate: A tool for overapproximating reachability of hybrid automata. In *http://stanleybak.com/projects/hycreate/hycreate.html.*

[2] S. Bak, S. Bogomolov, and T. T. Johnson. Hyst: A source transformation and translation tool for hybrid automaton models (tool paper). In *Hybrid Systems: Computation and Control (HSCC)*, 2015.

[3] S. Bogomolov, A. Donze, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Abstraction-based guided search for hybrid systems. In E. Bartocci and C. Ramakrishnan, editors, *Model Checking Software*, volume 7976 of *Lecture Notes in Computer Science*, pages 117–134. Springer Berlin Heidelberg, 2013.

[4] X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification*, 2013.

[5] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '08/ETAPS '08, pages 337–340. Springer-Verlag, 2008.

[6] A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, pages 381–397, 2004.

[7] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395, 2011.

[8] G. Frehse and O. Maler. Reachability analysis of a switched buffer network. In *Hybrid Systems: Computation and Control (HSCC)*, pages 698–701, 2007.

[9] S. Gao, S. Kong, W. Chen, and E. M. Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *CoRR*, abs/1404.7171, 2014.

[10] T. A. Henzinger. The theory of hybrid automata. In *Logic in Computer Science*, pages 278–292, 1996.

[11] K. H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry. On the regularization of zeno hybrid automata. *Systems & Control Letters*, 38(3):141–150, 1999.