Modeling the Wind Turbine Benchmark with PWA Hybrid Automata

Nikolaos Kekatos[1], Marcelo Forets[1], and Goran Frehse[1]*

University Grenoble Alpes, Verimag, France
{nikolaos.kekatos, marcelo.forets-irurtia, goran.frehse}@univ-grenoble-alpes.fr

Abstract

The wind turbine benchmark, from the ARCH benchmark repository, originates from an industrial model and entails closed-loop requirements that most verification tools have difficulties evaluating. The benchmark incorporates nonlinear and hybrid dynamics. The modeling is done with MATLAB/Simulink. To provide formal guarantees, we construct a PWA model of the wind turbine. In this format, the model can be used by several reachability tools for further analysis. The model transformation consists of three steps. The first step is to translate the Simulink blocks to equivalent blocks in SX format. The SL2SX translator facilitates with this task. The second step is to use a recently proposed technique for conducting compositional, syntactic hybridization in order to approximate the blocks that cannot be translated exactly into PWA dynamics. The third is a model validation step to check that the individual approximations (base components) are correct and non-blocking. We report some preliminary experiments on the subsystems (network components) of the wind turbine that we conducted with SpaceEx.

1 Introduction

The benchmark was proposed by Simone Schuler, Fabiano Daher Adegas, and Adolfo Anta from GE [7] in the ARCH workshop [2]. The authors proposed a simplified nonlinear model of a wind turbine equipped with switching controllers. The composition of the wind turbine and the controllers results in a hybrid system with nonlinear dynamics. The control synthesis of this system along with the accompanied control objectives form a pertinent benchmark for verification of requirements in hybrid systems [7].

This paper describes the process of constructing a PWA model expressed in SX format from the original Simulink model of the wind turbine benchmark and presents preliminary verification results undertaken with the SpaceEx tool [4]. The model transformation includes three main steps. We first use the experimental tool (SL2SX) [6] to assist with the translation and the process of building the SpaceEx model. Then, we conduct compositional syntactic hybridization [5] to obtain an approximation of the dynamics of the nonlinear blocks and finally we perform model validation.

*The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921 and by the Metro Grenoble through the project NANO2017.

The rest of this paper is organized as follows. Section 2 reviews the existing tools that are utilized. In Section 3, we briefly present the benchmark model. In Section 4, we describe the model transformation process. We report some preliminary results in Section 5. We draw conclusions and describe our future work in Section 6. The model and configuration files for running the case study in SpaceEx are available as an attachment.

2 Background

In this section, we present the three tools that are used for the model transformation, namely, Simulink, SpaceEx and SL2SX, and describe the hybridization technique that we apply.

2.1 Simulink

Simulink [1] is a graphical programming environment for modeling, simulating and analyzing dynamical systems. It includes a set of block libraries and is commonly used in automatic control and digital signal processing for multidomain simulation and Model-Based Design. A Simulink model consists of a set of input and outputs variables, blocks and connections between the blocks.

2.2 SpaceEx

SpaceEx [4] is a verification tool for hybrid systems. The goal is to verify (ensure beyond reasonable doubt) that a given mathematical model of a hybrid system satisfies the desired safety properties. It performs set-based reachability analysis and uses support functions as a set representation. A SpaceEx model consists of components (base or network) and binds. Base components correspond to single hybrid automata, whereas network components correspond to the parallel composition of several hybrid automata. Bind is a relation that associates each network component with a set of components. SpaceEx models respect the semantics of SX grammar; the format is similar to the standard hybrid automata, syntactically extended with hierarchy and templates.

2.3 Simulink to SpaceEx Translator (SL2SX)

SL2SX [6] is a semi-automated tool that undertakes the translation of Simulink models to SpaceEx models. The translator accepts a Simulink model that is saved in XML format and generates as an output a network of hybrid automata in SX format. The translation preserves most of the structural aspects of the Simulink diagram, such as the names, hierarchy and graphical positions.

2.4 Compositional Syntactic Hybridization

Compositional syntactic hybridization, a recently proposed method [5], is suitable for Simulink models and takes advantage of the on-the-fly composition of hybrid systems that is supported by the SpaceEx platform. The objective is to approximate in a compositional manner the original nonlinear model with a hybrid automaton with PWA dynamics. Three main steps are involved: *syntactic decomposition*, replacing the original system by an equivalent one with extra variables; *hybridization*, constructing a PWA approximation for each domain and providing a sound over-approximation of the original system by adding an error term; and finally *HA composition*, where the PWA model is transformed into a hybrid automaton in SX format.

3 Benchmark Model

The wind turbine modeling is done with MATLAB and Simulink [7]. Figure 1 shows the top-level model of the Simulink block diagram and Figure 2 depicts the physical part (plant) of the wind turbine. It should be noted that the wind turbine dynamics are highly nonlinear functions of the operating point defined by the rotor speed, wind speed and blade pitch angle.

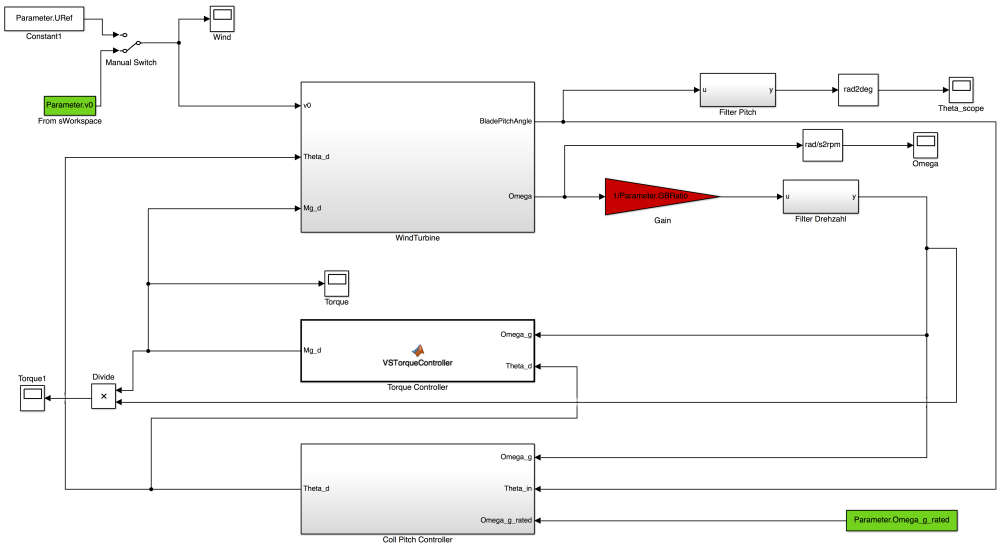


Figure 1: Wind turbine - Simulink model - Top Level [7].

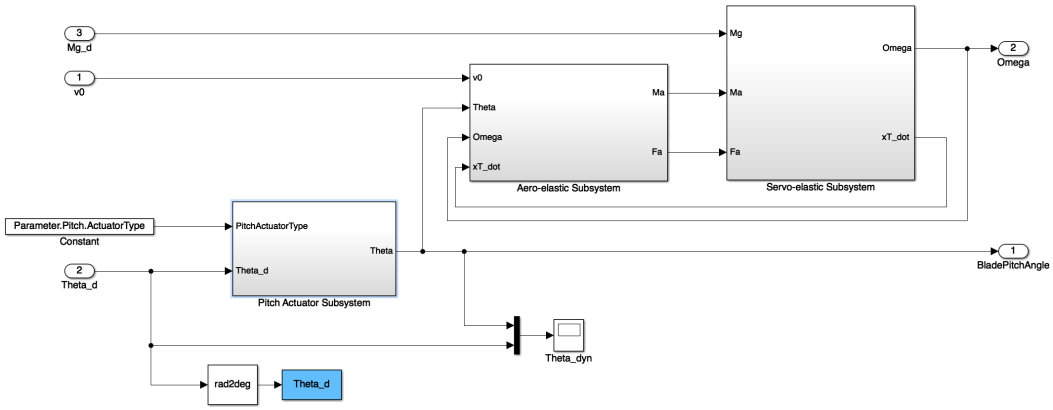


Figure 2: Wind turbine - Simulink model - Plant [7].

The plant consists of three subsystems: servo-elastic, aeroelastic and pitch-actuator. The servo-elastic subsystem describes the tower fore-aft dynamics and the rotor dynamics. It is interconnected with the aeroelastic subsystem through signals that correspond to the aerodynamic torque and thrust. While the servo-elastic subsystem consists of linear operators, the aeroelastic subsystem has a large number of nonlinear blocks. There are two `Square` functions,

two `Products`, two `Divisions` and two fourth-order `Polynomials`. The polynomials correspond to the `cP` and `cT` coefficients, and they are computing through a regression model. The pitch actuator dynamics can be expressed by a second-order lag, a first-order lag or time-delay. The pitch actuator subsystem has two input signals, the demanded pitch angle and the pitch actuator type and outputs the actual pitch angle to the aeroelastic subsystem. Apart from the linear operators, it contains a `Multiport Switch`, three `Compare to constant` blocks, three `Enable` blocks and three `Enabled Subsystems`.

As for the control design part, there is a generator-torque controller and a collective blade-pitch controller. The blade pitch controller is a gain-scheduled PID whose objective is to minimize the speed error between the filtered generator speed and the rated generator speed. It incorporates an anti-windup scheme to prevent integration wind-up when the actuator is saturated. The nonlinear blocks correspond to a `Division` and a `Product`. The generator-torque controller aims to maximize the extracted maximum power from the wind by tracking the optimal tip-speed ratio λ_{opt}. It is a hybrid controller with 5 locations, 2 input signals, namely the filtered generator speed and the pitch angle. It is written as an `Embedded MATLAB Function` and it includes two nonlinear operators, a `Square` and a `Division`.

Note that the benchmark model is designed with an industrial viewpoint and its verification task is considered to be of high difficulty. That results from the existence of several nonlinearities (over 10 distinct ones) and blocks that cannot be expressed by linear or hybrid dynamics. As far as the model dimension is concerned, the controllers introduce two state variables and the plant introduces 3-5 states (depending on the pitch actuation). As such, the total number of state variables of the Simulink (closed-loop) model varies from 5 to 7. This number could be further increased if the rate limiters and memory blocks are considered. The SpaceEx model adds an extra variable to capture the time evolution. Another remark is that the Simulink model has some extra functionalities with respect to the mathematical formulas presented in the benchmark paper [7] (e.g. rate limiters). Finally, there are two minor remarks regarding the benchmark model; some MATLAB paths that need to be modified and there is a call to a toolbox (WAFO) that is not included in the benchmark files.

4 Model Transformation

This section presents the model transformation process, covering the translation, hybridization and validation steps.

4.1 Translation

The first step of the model transformation is to translate the Simulink model to an equivalent[1] SpaceEx model. However, it should be noted that not all Simulink blocks can be translated exactly, as they cannot be expressed as hybrid automata. In this respect, before we start the translation process, it is necessary to pre-process and modify the Simulink model. In fact, all the `Scopes`, `Mux`, `Demux`, `Enabled Subsystems`, `Manual Switches`, `Save to workspace` have to be deleted/replaced as their corresponding actions are defined in a different way with SX format and SpaceEx. In addition, the `Read from the workspace` blocks and the discrete-time user-defined signals are not particularly efficient and should be replaced. Considering the wind disturbance (speed) block, one can observe that there are 33 possible combinations of user-defined wind signals (`aeromaps.mat`). They are used to evaluate the performance of the wind

[1] Equivalent in the sense that the model remains the same while the syntax changes.

turbine under varying initial conditions and input signals. However, even if all these combinations are tested, there is still no guarantee that the closed-loop system operates correctly. Actually, there might be a different wind profile that could yield undesirable behaviors. On the contrary, with SX format, this matter could be simply solved by adding a non-deterministic input signal that covers the minimum and maximum wind speeds. It is also possible to impose bounds on the wind speed derivative. Another modification concerns the three available pitch actuation models that exist in the Simulink model. As for large wind turbines, the pitch actuator dynamics cannot be neglected, we consider the most critical and representative case, namely the second-order lag.

After modifying the Simulink blocks, we have to perform further pre-processing in order to be able to use the SL2SX translator. Special attention should be given to the names and the connections. As of version 1.0.1 of SL2SX, the block names should not extend to multiple lines, as the respective blocks will not be parsed by the translator. Also, there may occur errors with models that are saved in old Simulink versions.

Once the pre-processing is completed, we are ready to use the SL2SX translator. It should be highlighted that having the mechanical aspects of the model translation carried out by a tool significantly reduces errors. The translation takes a few seconds and produces an XML file with 18 Network Components and 89 Base Components. The Simulink blocks that are automatically translated to base components are the following: *Add, Subtract, Divide, Multiply, Constant, Gain, Saturation, Integrator, Subsystem, Inport, Outport.* The binds should be checked if they are connected correctly. Also, for the current SL2SX version, the expressions ;&*gt* and ;&*lt* should be manually replaced by $<$ and $>$ respectively[2].

Finally, it is possible to make the SX model more compact by deleting the base components that are duplicate and adding the appropriate binds. It should be noted that base components that serve the same role, i.e. `Sum`, since the number of inputs or outputs may be different, as well as the orientation of the corresponding ports may not be the same. With this configuration, the total number of base components can be reduced approximately up to 30 distinct ones. Figure 3 and Figure 4 present the top level block (network component) and the plant subsystem of the wind turbine model in SX format, as shown in the Model Editor of SpaceEx.

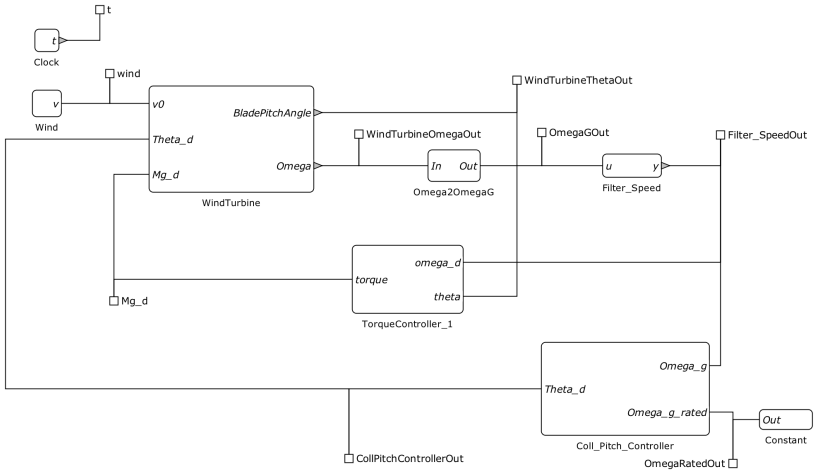


Figure 3: Wind turbine - SpaceEx model - Top Level.

[2]This a known problem with the SAX parser that the SL2SX translator calls.

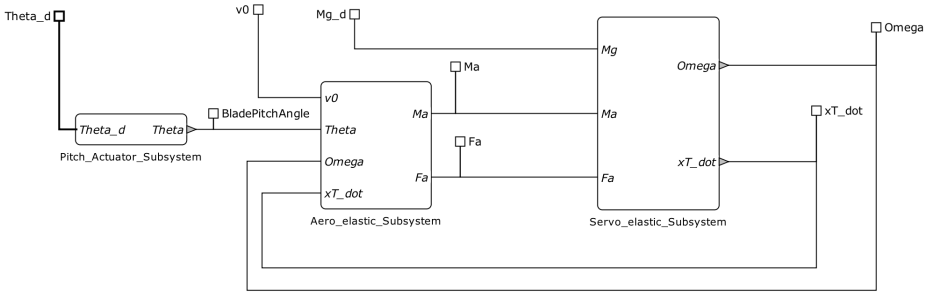


Figure 4: Wind turbine - SpaceEx model - Plant.

4.2 Hybridization

The second step is to generate PWA approximations and describe them in SX format for the Simulink blocks that cannot be handled automatically, either because no exact translation is available (e.g. nonlinearities) or because translation cannot be applied (e.g. Embedded MATLAB Function).

To obtain PWA approximations of nonlinear dynamics, we use an abstraction method known as syntactic hybridization. This method essentially partitions the state-space into a set of domains, and for each domain, it approximates the nonlinear dynamics by simpler ones with added nondeterministic inputs to account for the abstraction error. However, instead of operating on the composed (flattened) system, we decompose the original dynamics and carry out the state-space partitioning and PWA approximation on the components. In this way, we can avoid having intractably large models and the explosion in the number of partitions can be mitigated. In practice, we break down the nonlinear blocks into base components that have a small number of input variables and interconnect them together.

A practical intermediate step is to compute an estimation of the range of the signals of the blocks that shall be approximated, as it could result in more tractable approximation models. That is the case as the smaller the range of the signals, the smaller the number of locations that is required, given a desired error bound. The corresponding range is computed through Simulink simulations. In fact, we observe the values of the block signals for different scenarios and initial conditions. The resulting range is enlarged by a relative percentage. In case the range is shown to be insufficient during reachability computations, it is further refined (enlarged). The degree of enlargement is identified through visual inspection.

For the wind turbine benchmark, a number of simplifying assumptions has been considered. In particular, the constructed base components are designed to have only one or two input signals and one output signal, the state space is partitioned into a set of non-overlapping domains, described by hyper-rectangles (boxes), the approximations are linearized (first-order Taylor) around the center of each domain, and the abstraction (linearization) error is computed by the evaluation of the Lagrange remainder. All the computations (linearization domains, quantization parameters, operating points, PWA approximations, errors) are conducted with MATLAB. We also have a script to facilitate the integration of the constructed PWA approximations, stored as MATLAB structures, with the main SpaxeEx XML file. More specifically, we can automatically transform the MATLAB approximations into equivalent base components models, described in SX format, and integrate all these components with the original XML file.

Note that the approximations can be conservative or non-conservative. The error is com-

puted for each location of the base components and is added as a non-deterministic input in the corresponding invariants. However, it is possible to disregard the errors either for debugging purposes or if the goal is to obtain a deterministic model. Table 1 presents all the Simulink blocks that need to be approximated along with the approximation method, the maximum induced errors and the number of locations of each approximation. Let us underscore that the error corresponds to the maximum error that appears in one of the locations of the PWA approximation. That means that the individual error in the other locations can be significantly smaller.

No	**Block (Name)**	**Approximation Method**	**Number of Locations**	**Error Bounds**	**Remarks**
1	Product *(Anti windup)*	syntactic	2x2	2	2 variables
2	Division (C_p/λ)	syntactic	10	0.027613	2 var: (x/y)
3	Division (*GS factor*)	syntactic	2	0.020111	1 var: $\frac{1}{1+x}$
4	Division (λ)	syntactic	4	0.6983	2 var: x/y $(\omega R/v_{ref})$
5	Product ($C_T \cdot v_{ref}^2$)	syntactic	4	25.5781	$x^2 * y$
6	Product ($C_p/\lambda \cdot v_{ref}^2$)	syntactic	4	3.23437	$x^2 * y$
7	Polynomial (*aeromaps*)	syntactic	6 and 6	3.39472, 12.2809	two 4th-order approx.
8	Embedded MATLAB	syntactic	28	10.48584	x^2 and $1/x$.
9	Saturation	substituted	3-4	0	exact translation (add initially false)
10	Read from workspace	substituted	1	-	non-deterministic input
11	Mux, Scope	unnecessary	–		
12	Save to workspace	unnecessary	–		
13	Enabled Subsystem	unnecessary	–		
14	Multiport Switch	unnecessary	–		
15	Compare to Constant	unnecessary	–		
16	Manual Switch	unnecessary	–		
17	Rate Limiter	ignored	–		DAE
18	Memory	ignored	–		continuous delay
Total			$\mathbf{15,482,880}$		upper bound (worst case)

Table 1: SpaceEx base components - Approximations

To perform reachability analysis, SpaceEx performs on-the-fly composition of the hybrid automata and instantiates only the reachable parts of the state-space. The upper bound of the number of locations of the composed model is 16 millions[3].

[3]Performing nonlinear simulation with SpaceEx could be beneficial, as we can arrive at a better estimate of the numbers of locations that are actually reachable.

4.3 Model Validation

The third step of the model transformation concerns the model validation. We propose an empirical method to evaluate whether the generated base components are correct, yield satisfactory behaviors and do not introduce any deadlocks. In this step, we are essentially testing the implementation in this step. In this vein, we have created an XML signal library with predefined components that describe the *trigonometric (sine* & cosine), step and *ramp* functions. We have added a monitor to visualize the output[4]. Once we create a new approximation, we can integrate it with the XML library and obtain a tester SpaceEx module (consisting of base and network components).

Our objective is twofold: on the one hand, to check that the approximation is non-blocking and on the other hand to check that is indeed an over-approximation, in case the errors are added. For the first objective, we select an input signal, typically a ramp function, which spans the entire operating range. In practice, the input signal starts from the minimum allowed value and reaches the maximum. Accordingly, the automaton should visit all of its locations and then find a fixed point. For the second objective, we perform reachability analysis on the tester module and then compare the results with random simulations. In this way, we check whether the simulations are included in the reachable sets.

As an illustration, let us consider a Simulink block of the aero-elastic subsystem of the wind turbine (plant). Figure 5 shows the entire aero-elastic subsystem. The block that we focus on is the `Divide` block and it is highlighted in red.

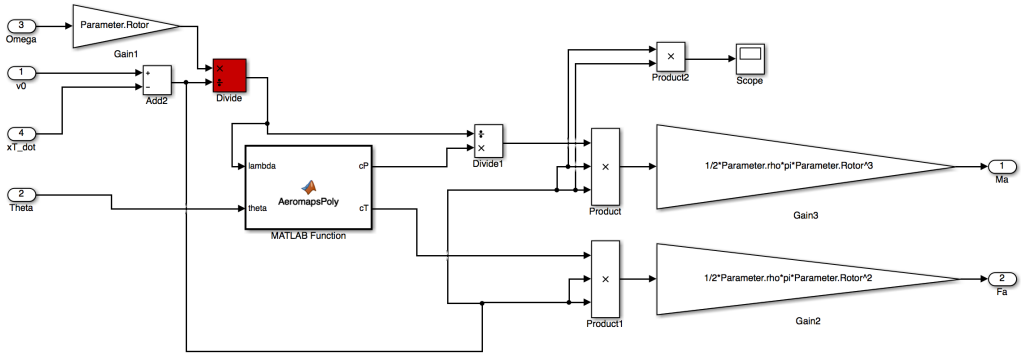


Figure 5: Aero-elastic subsystem - Simulink - Nonlinear `Divide` block shown in red.

This block corresponds to the division operator and computes the division between two input signals over time. From a physical point of view, it computes the tip-speed ratio, which is a dimensionless variable denoted by `lambda`(λ). This block, being a nonlinear one, is replaced by a PWA over-approximation with 4 locations. Figure 6 shows the approximation as a SpaceEx base component.

The variables `In1` and `In2` relate to the inputs, `Out1` is the PWA approximation of the division operator `In1/In2` and corresponds to λ. The approximation error `w1` is added in the form of a non-deterministic disturbance. Note that all the variables should be uncontrollable, the error term is included in the invariance and it is described by a local variable.

[4]Note that STC and LGG scenarios in SpaceEx cannot output algebraic variables, so visualizing is possible by mapping the algebraic variable to the solution of an ordinary differential equation.

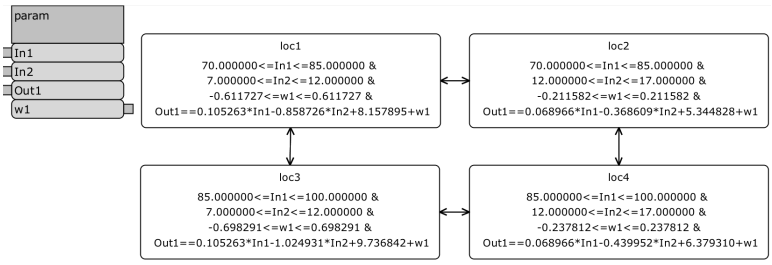
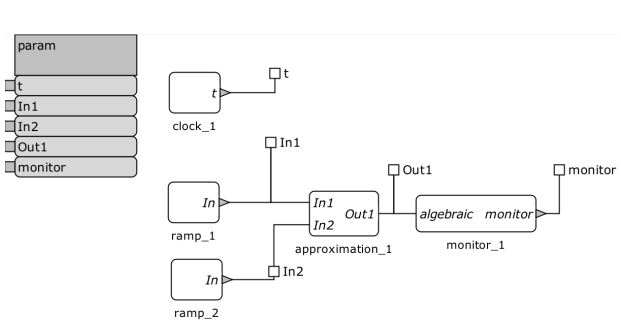


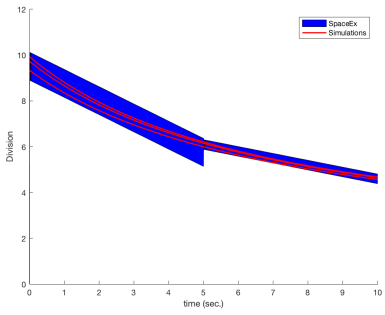
Figure 6: `Divide` block - PWA approximation in SpaceEx.

After integrating the base component of the `Divide` block with our input library, we want to check the previously mentioned objectives (no deadlocks and over-approximation). For this reason, we consider that the inputs are ramp signals with varying initial conditions. The first input is described by $\dot{\texttt{In1}}$`==1` with initial conditions in `69 < In1 < 71` and the second input by $\dot{\texttt{In2}}$`==1` with initial conditions in `6.9 < In2 < 7.1`. It is necessary to add a monitor in order to visualize the output, as we intend to use the STC algorithm. Then, we perform infinite-time reachability analysis with SpaceEx, a flowpipe tolerance of 0.1 and box directions. SpaceEx finds a fixed point after 10 seconds and 4 iterations. It terminates after 10 seconds, as `In2` exceeds the maximum allowed value it can take. Finding a fixed point, after visiting its locations, indicates that there is no deadlock.

Then, we compare the results with random simulations of the original nonlinear function and we observe that for the considered scenario SpaceEx yields an over-approximation. The tester module is shown in Figure 7a, the reachable sets and the simulation runs are shown in 7b. With blue the reachable sets of the approximate SpaceEx block are indicated, while with red we specify the simulation runs of the nonlinear function. By visual inspection, it is evident that the simulation runs are contained in the reachable sets.



(a) Tester - Division λ.



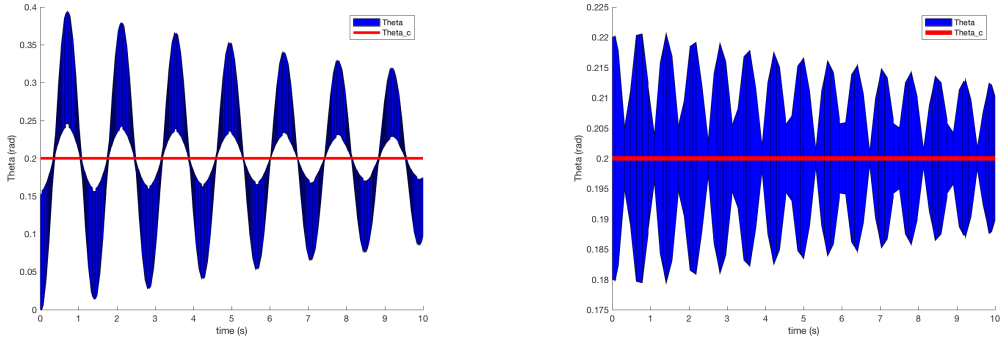
(b) Reachable sets and Simulations.

Figure 7: SpaceEx Base Component - Division λ - Model Validation

The same approach (in a hierarchical way) is utilized in order to check that compositions of multiple base components (over-approximations), network components or larger subsystems operate correctly.

5 Preliminary Results

In this section, we present validation runs for some of the subsystems (network components). We start with the **pitch-actuator** subsystem of the wind turbine plant. The input of this block is the commanded pitch angle and the output is the actual pitch angle. Considering the STC scenario, a global time horizon of 10s, and a constant input signal, we get a fixed point. Figure 8 depicts the reachable sets for different initial conditions. The input signals are shown in red and the output signals in blue.

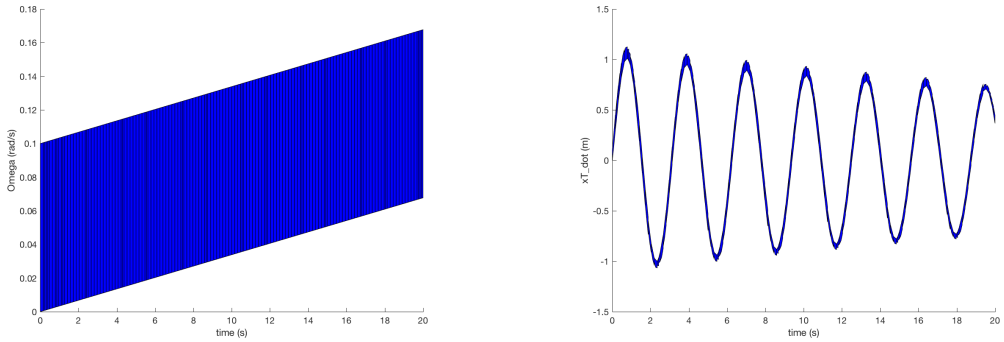


(a) Initial Conditions - $0 \leq \theta \leq 0.15$.

(b) Initial Conditions - $0.18 \leq \theta \leq 0.22$.

Figure 8: Pitch actuator - SpaceEx Network Component - Reachable Sets (blue: Output, red: Input)

Next, we consider the **servo-elastic** subsystem of the wind turbine plant. Given constant inputs, a global time horizon of 20s and the STC scenario, SpaceEx finds a fixed point. Figure 9 depicts the reachable sets for initial conditions: $0 \leq$ Omega ≤ 0.1 and $0 \leq$ xT_dot ≤ 0.1.



(a) Output - Omega

(b) Output - Displacement (xT_dot).

Figure 9: Servo-elastic - SpaceEx Network Component - Reachable Sets

Finally, we analyze the **torque controller**. The approximation consists of 28 locations, has two input signals (omega_d, theta) and one output (torque). We consider the input signals of Figure 11, as they cover the entire operating range (from minimum to maximum allowed values). Indeed, with this configuration, SpaceEx visits all the controller locations and finds a fixed point after 16 seconds. The reachable sets are computed with the PHAVer scenario. PHAVer does not eliminate the algebraic constraints and it can be more suitable than STC and LGG scenarios for visualizing algebraic outputs, such as the controller output. On the downside, it leads to coarser over-approximations. The reachable sets of the torque controller are displayed in Figure 10.

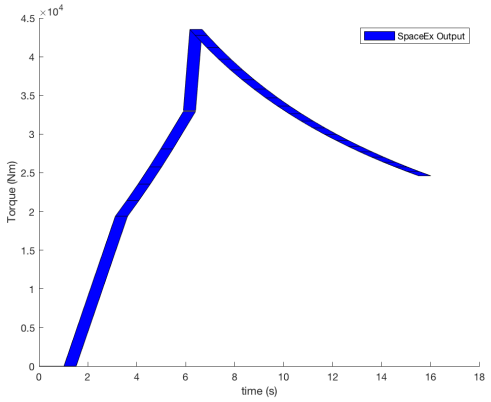
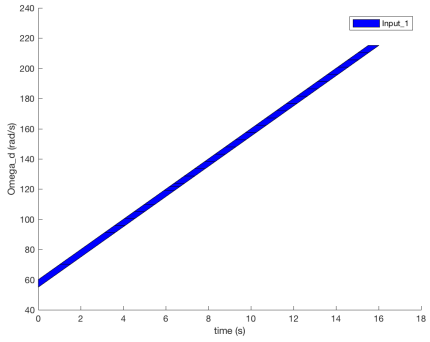
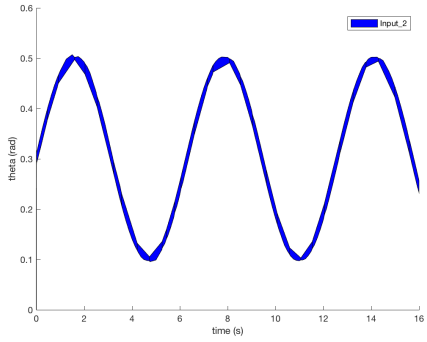


Figure 10: Torque Controller - SpaceEx Network Component - Output.



(a) Input 1 - Omega_d.



(b) Input 2 - Theta.

Figure 11: Torque Controller - SpaceEx Network Component - Inputs.

6 Conclusions

Model transformation plays an important role in bridging the gap between industrially relevant models and verification tools. This work aims to assist the application of hybrid system reachability tools to industrial-sized models described by MATLAB/Simulink. The existence of a translator from Simulink to SpaceEx already facilitates the use of Simulink models within SpaceEx environment. However, the large number of Simulink blocks and their diverse features have posed difficulties mapping Simulink subsystems to SpaceEx components. That is the case since there are several Simulink blocks that cannot be described by hybrid automata with PWA dynamics and therefore cannot be processed by SpaceEx. In practice, most of these blocks correspond to nonlinear functions and operations.

The wind turbine benchmark constitutes a relevant example, as it is an industrial case study and includes more than 10 nonlinear blocks. Through syntactic hybridization, it is possible to over-approximate the dynamical behavior of these blocks and seamlessly integrate them within SpaceEx components. Note that the resulting model of hybrid automata is expressed in the general SX format. As such, it can be fed directly into SpaceEx platform, or translated into formats for other verification tools using the HyST [3] tool.

Our future work is targeted towards verifying the requirements of the wind turbine benchmark. As the proposed approximation contains around 15 million locations, there is a shortcoming concerning the computational costs and execution time. Currently, SpaceEx identifies the initial locations of the approximation through enumeration, which does not scale. A potential way to tackle this issue is to use compositional reasoning to identify the initial locations and instantiate as few locations as possible during the analysis.

Acknowledgments

The authors would like to thank Simone Schuler for valuable discussions on wind turbine models.

References

[1] MATLAB 9.0 and Simulink 8.7. The MathWorks, Inc., Natick, Massachusetts, United States.

[2] Matthias Althoff and Goran Frehse, editors. ARCH16. Proceedings of 3rd International Workshop on Applied Verification of Continuous and Hybrid Systems, EPiC Series in Computing, 2017.

[3] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. HYST: a source transformation and translation tool for hybrid automaton models. In Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pages 128-133. ACM, 2015.

[4] Goran Frehse, Colas Le Guernic, Scott Cotton Alexandre Donzé, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In CAV'11, pages 379-395, 2011.

[5] Nikolaos Kekatos, Marcelo Forets, and Goran Frehse. Formal verification of nonlinear simulink models via syntactic hybridization. Submitted, https://hal.archives-ouvertes.fr/hal-01487658, 2016.

[6] Stefano Minopoli and Goran Frehse. SL2SX translator: From Simulink to SpaceEx models. In Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, pages 93-98. ACM, 2016.

[7] Simone Schuler, Fabiano Daher Adegas, and Adolfo Anta. Hybrid modelling of a wind turbine (benchmark proposal). Applied Verification for Continuous and Hybrid Systems (ARCH), 2016.