SL2SX Translator: From Simulink to SpaceEx Verification Tool.

Stefano Minopoli and Goran Frehse

***Abstract*— The tool *Matlab/Simulink* is numerical simulation environment that is widely used in industry to validate systems in a model-based design methodology. Numerical simulation scales well and can be applied to systems with highly complex dynamics, but it is also inherently incomplete in the sense that critical events or behavior may be overlooked. The application of formal verification techniques to Simulink models could help to overcome this limitation. Existing verification tools such as *SpaceEx* typically use as underlying formalism hybrid automata, which are semantically and structurally different from Simulink models. To address this issue, we are building the tool *SL2SX* for transforming a subset of the Simulink modeling language into a corresponding SpaceEx model. Our method is designed to preserve the syntactic aspects of a given Simulink diagram: the resulting SpaceEx model shows the same hierarchical structure and preserves the names of components and variables. Placeholders with the correct interface are provided for unsupported Simulink blocks, which can then be translated manually. We illustrate the tool *SL2SX* and the verification of the transformed models in SpaceEx via two examples provided by the Mathworks example library.**

I. Introduction

Simulink [17] is a widespread standard tool used to model and simulate physical systems. For modeling, it provides a set of blocks to create a diagram, possibly hierarchical, of the system. For simulating, it provides an extensive library of solvers, each of which determines the time of the next simulation step and applies a numerical method to solve the set of ordinary differential equations arising from the model.

Different from simulation, formal verification does not suffer for intrinsically incomplete coverage, and hence it can be used to mathematically ensure correctness of designs, by showing whether or not a model meets a requirement. Hence formal verification, for example via reachability algorithms on *Hybrid Automata (HA)* [8] and related tools like HyTech [9], PHAVer [7] or SpaceEx [6], could be used to try to overcome the incomplete coverage of simulation. In particular, the SpaceEx verification language is able to preserve the structure and the hierarchy of a Simulink diagram, by the meaning of *basic* and *network* components (a single hybrid automaton and a network of them, respectively). This, coupled with the SpaceEx reachability algorithms highly optimized for piecewise affine dynamics, makes this tool a good candidate to be used for this task. We want to remark that preserving the structure of the simulation model is not just a secondary aspect because, as well described in [24], the structure has a profound impact on several aspects of safety-critical model development.

S. Minopoli and G. Frehse are with VERIMAG, Centre Équation - 2, avenue de Vignate, 38610 GIÉRES [minopoli,freshe]@imag.com

To achieve the goals above, it is necessary to switch from a *simulation model* to a *verification model*. Such a transition needs to take into account the inherent differences between the two classes of models. Indeed, a simulation model contains details that need to be abstracted for the verification model. Instead, a verification model needs to be enriched with non determinism to fully exploit the state space exploration of verification engines. In addition, Simulink lacks an actual formal semantics description and it uses *must semantic*, meaning that a transition must be taken as soon as possible its guard is satisfied, while usually verification models (like HA) are defined with *may semantics*. This may potentially affect the quality of the translation, resulting in overapproximations, and the complexity of the model, involving state space explosion. For all these reasons, and because of the wide variety of Simulink blocks, the process of translating a Simulink diagram into a verification model is not an easy task, and a full automatic approach seems to be not plausible.

There have been a range of works with the aim of applying formal verification to a Simulink model, showing that the main issues to solve are due to (i) lack of a formal definition of Simulink semantics, (ii) difficulties in modeling the structure and the hierarchy of a Simulink model, (iii) complexity of the involved dynamics, (iv) discrepancy between must and may semantics, and (v) limited scalability in the verification task. For example, the tools *HyLink* [13] and *GreAT* [1] translate a Simulink diagram into hybrid automaton expressed by intermediate formats (i.e. *hybrid input/output automata (HIOA)* [20] and *hybrid system interchange format (HSIF)* [22], respectively). Both the formalisms are not able to model hierarchy and must semantics. The tool *Checkmate* [21] provides a Simulink toolbox containing additional blocks that the designer is allowed to use. A Simulink model thus obtained is then translated into the special class of *Polyhedra Invariant Hybrid Automata (PIHA)* [5]. PIHA formalism featuring ordinary differential equations (ODE) to express the dynamics, hyperplane for guard transitions and only identity as update functions. Hierarchy can not be explicitly handle, and moreover the PIHA may semantics causes overapproximation on modeling must transitions. Other similar attempts are given by not using hybrid automata. For example, the technique introduced in [23] consists of translating the discrete part of a Simulink model into a pushdown automaton defined by the *SAL transition system language* [3], while the differential equations arising from the Simulink component are converted into difference equations. The resulting model is a discretization of the original. The tool *S2H* [26] gives a translation of a Simulink model into the *Hybrid CSP (HCSP)*

formalism [11], based on the separation of variable definitions, process definitions, assertions definitions and goals to be proved. This causes loss of compositional properties and the obtained model may be hard to be compared with the model in input. Another approach is introduced by the tool *HySon* [4], which performs simulation with "imprecise" or "uncertain" inputs directly on a Simulink model. The aim is to compute a good approximation of the set of all possible Simulink executions. Being based on numerical simulation methods, HySon is able to handle systems with both non-linear dynamics and guards, and zero-crossing events are properly treated. About performance, we expect that a tool like SpaceEx, based on algorithms that are highly optimized for piecewise affine dynamics, could outperform the more general algorithm of HySon. Unfortunately, HySon is not publicly available and hence a comparison is not feasible.

In this work we describe our approach in addressing the five points described above, and present the tool *Simulink to SpaceEx Translator (SL2SX)* that takes a Simulink model (in xml format) as input, and generates a network of hybrid automata in a format compatible with *SpaceEx* verification tool. About issue (i), the proposed translation is based on the ideal interpretation of the Simulink semantics [12]. The choice of SpaceEx as target tool allows us to achieve several goals. About point (ii), as we said above, the SpaceEx modeling language supports hierarchy, via basic and network components. Our tool is hence able to preserve structural aspects of the Simulink diagram in input. Moreover, the translator preserves the names of blocks and variables, and components of the resulting SpaceEx model shows the same graphical positions and dimensions of the corresponding Simulink blocks. SpaceEx also provides features useful to handle issue (iv), because of the recent introduction of an exact reachability algorithm for Piecewise-Constant Derivatives, also known as *Linear Hybrid Automata (LHA)*, with *urgent conditions* [19]. Indeed, urgency allows one to model must semantics without the introduction of additional locations and variables, by preventing both to relapse in the issues (ii) again, and to cause state space explosion. For the case of LHA, the exact algorithm prevents overapproximations. Moreover, we are currently working on urgency for affine dynamics. About issue (v) (performance), experimental results of full fixed-point computations hybrid systems with more than 100 variables illustrate for the scalability of SpaceEx algorithms [6].

We think that *SL2SX* can help in the process of translating a Simulink simulation model into a SpaceEx verification model. The tool speeds up the process, limits the errors and builds a model that can be easy compared with the Simulink diagram in input, due to the preservation of structure and names of blocks and variables. The translation is not fully automatic, meaning that the the SpaceEx model carried out by *SL2SX* needs to be manually completed. In particular by defining parameters for the reachability analysis, and by designing hybrid automata to model blocks for which no translation is given. For this last task, the tool help the user by arranging in advance corresponding placeholders with proper interface and warns the user to complete the model.

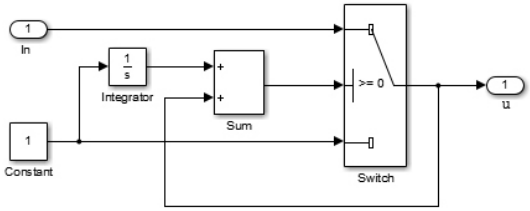


Fig. 1. An example of Simulink modelling the dynamics $\dot{x} = c + u$, when $x \geq 0$, and $\dot{x} = c$ otherwise

SL2SX together with models used as cases study are available in [18].

II. Simulink, Hybrid Automata and SpaceEx

In this Section we introduce the modeling language of Simulink, the formalism of Hybrid Automata and the SpaceEx verification tool. We highlight those features relevant to this work.

A. Simulink Models

Simulink is an environment for simulation and model-based design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of blocks that let one design, simulate, implement, and test a variety of time-varying systems. The modeling language lacks a formal and rigorous definition of its semantics, usually estimated by either the ideal or the numerical simulation interpretation [12]. A Simulink design is represented graphically as a diagram consisting of inter-connected Simulink blocks. It represents the time-dependent mathematical relationship between the inputs, states, and outputs of the design. Figure 1 depicts an example of Simulink diagram, whose signal that drives the switch blocks is modeled by the hybrid dynamics $\dot{x} = c + u$, when $x \geq 0$, and $\dot{x} = c$ otherwise (where x is the integrator output).

The follows definition derives from [2]. A Simulink model $SL = \langle D, B, C \rangle$ consists of the following components:

- A finite set D of *variables*, partitioned into the set of input variables D_I and the set of the output variables D_O.
- A finite set B of Simulink *blocks*. Each block $b \in B$ has input, output and parameters. The input and output variables are associated with input and output block ports. A Simulink block can be itself a Simulink Diagram by allowing hierarchy. Such a block is called a *subsystem*.
- An ordered relation $C \subseteq B \times B$ that represents *connections* between the blocks. A connection $c = \langle b, b' \rangle \in C$ connects an outport of b with an inport of b' and represent the flow of the data between the corresponding variables of b and b'.

Simulink features a *must semantics*, meaning that discrete events happen as soon as possible a given condition (guard) is satisfied (also referred as *urgent* or *as-soon-as-possible (ASAP)* semantics). We consider Simulink semantics under

the ideal interpretation, and exploit the numerical interpretation to deeper understand how to treat strict guards (see Section III-B).

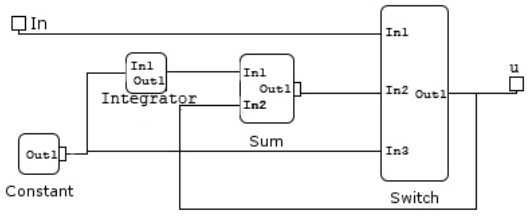
B. Hybrid Automata

Hybrid automata [8] are a verification modeling formalism that combines discrete states (modeled by locations) with continuously evolving, real-valued variables. The discrete states and the possible transitions from one state to another are described with a finite state-transition system. A change in discrete state can update the continuous variables and modify the set of differential equations that describes how variables evolve with time. Hybrid automata are nondeterministic, which means that different futures may be available from any given state. Rates of change or variable updates can be described by providing bounds instead of fixed numbers. Incomplete knowledge about initial conditions, perturbations, parameters, etc. can easily be captured in this way. A specific source of non-determinism is due to the *may semantics*. This means that in hybrid automata, a transition may happen at every time that the associated guard is satisfied. This can be a major limitation in modeling the Simulink must semantics. A way to conciliate may and must semantics is provided by hybrid automata with urgent conditions [19], that allow the definition of a urgency condition for each location. A must transition can be easily encoded by adding its guard as urgent condition to the source location. Notice that, the allowed urgent conditions are topological closed polyhedra, that may be problematic in modeling a urgent transition with a strict guard (see Section III-B). The approach by using urgent conditions is effective for the class of LHA, meaning that an exact reachability algorithm is available in SpaceEx [19]. We are currently work on a corresponding algorithm for affine dynamics.

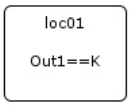
C. SpaceEx

SpaceEx is a verication platform for hybrid systems. The basic functionality is to compute the sets of reachable states of a system, modeled by the SpaceEx definition language. The SpaceEx verification engine provides specific reachability algorithms, and each of them may come with its own set representation, apply to its own class of models, and produce a different kind of output. This bundle is also referred as a *scenario*. Currently four scenarios are implemented:

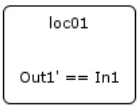
1) PHAVer: implements the exact reachability algorithm for Linear Hybrid Automata, also available for the LHA with urgent conditions (SpaceEx allows the declaration of urgent transitions and automatically converts to the corresponding urgent conditions).
2) LGG support function: implements a variant of the Le Guernic-Girard (LGG) algorithm from [10]. It applies to hybrid systems with piecewise affine dynamics with nondeterministic inputs, and computes overapproximation of the result.
3) STC: a refinement of LGG, that produces fewer convex sets for a given accuracy and computes more precise images of discrete transitions.



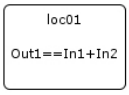
(a) The Network component



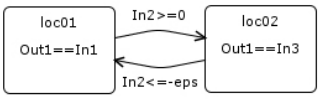
(b) HA for component "constant"



(c) HA for component "integrator"



(d) HA for component "sum"



(e) HA for component "switch"

Fig. 2. An example of SpaceEx model for the dynamics of the previous Simulink example.

4) Simulation: an exhaustive simulation algorithm that mimics reachability analysis by random sampling of the initial states.

The overapproximation of LGG and STC scenarios is efficiently computed by a-priory fixing the facet normals of the result, which leads to so-called *template polyhedra*. Templates currently available are essentially boxes and octagonal shape. The user can increase the accuracy of the approximation by including additional directions.

A SpaceEx model is similar to the hybrid automata known in literature, except that it provides a richer mechanism of hierarchy, templates and instantiations. In addition, it is possible to declare a transition as urgent, allowing must semantics. A SpaceEx model consists of one or more *components*. Each component has a set of *formal parameters*, like *continuous variable* with arbitrary dynamics, *constant* and synchronization labels. A formal parameter is part of the *interface* of a component, unless it is declared as *local* to the component. There are two types of components: *Base Components* correspond to a single hybrid automaton, and *Network Components* that allow the instantiation one or more components (base or other network components), via their variables, and assign values to their constants. Hence, a network is a parallel composition of its subcomponents, and this allows to model hierarchy. When instantiating a component A in network B, one must specify what happens to each of the formal parameters in its interface. This is called a *bind*. Every formal parameter of A must be bound to either a formal parameter of B or to a numeric value. Components inside B can be connected by binding their variables to the same symbols in B. Because of the acausal semantics, the variables binding does not distinguish source and destination (non-oriented connection). Figure 2(a) depicts an example of SpaceEx model, whose components are the hybrid automata

(a) A Simulink subsystem with one input and one output

(b) The corresponding SX network component preserves the interface

Fig. 3. A Simulink subsystem and the corresponding SX network component

shown in Figure 2. The dynamics is the same as the Simulink model in Figure 1.

Formally, a SpaceEx model $SX = (Comp, Bind)$ consists of the following components:

- A finite set $Comp$ of SpaceEx *components*, partitioned into the set $Comp_b$ of basic components and the set $Comp_n$ of the network components. Each component $b \in Comp$ has a set of formal parameters, including a set of variables Var_b. For the aim of this work and a sake of space, we omit discussion about synchronization labels and local variables.
- A relation $Bind \subseteq Comp_n \times Comp$ that associates each network component with a set of components (also networks components, by allowing hierarchy). Each variable of a component associated to a network, is also a variable of the network. For each $(n, c) \in Bind$, a mapping $Map_{n,c} : Var_c \rightarrow Var_n$ associates to each basic variable a network variable.

III. From Simulink to SpaceEx

This section describes the technique used for implementing the tool *SimuLink To SpaceEX Translator (SL2SX)*. The core of the process is a parser that analyzes a Simulink model $SL = (D, B, C)$ expressed in XML format, and produces the corresponding SpaceEx model $SX = (Comp, Bind)$. The basic idea consists of converting each Simulink proper block (i.e. not a subsystem) in a SpaceEx basic component, while each subsystem is modeled by a network component. Instead, Simulink connections are expressed by opportunely mapping related variables. Because of Simulink connections are oriented, while SpaceEx mappings are not, this task requires some additional considerations as explained in Section III-D.

The technique handles also Simulink blocks for which no translation is given (we refer to these as *non-supported* blocks). When the parser meets one of them, the process does not stop and a basic component, together with related variables and mapping, is added as a placeholder. At the end of the process, the tool warn the user to complete the obtained model (by building hybrid automata to model the non-supported blocks).

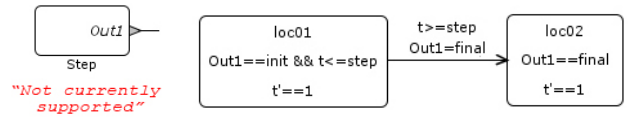
A. Translating Blocks

For each block $b_i \in B$, such that b_i is not a subsystem, an inport or an outport, the translator adds to the set $Comp_b$ the corresponding SpaceEx basic component (an hybrid automaton) with same name of b_i. For each input and output of b_i a variable is added to Var_b. According to the class of the block, namely continuous or discrete, the translator adds to the hybrid automaton a certain number of locations. In particular, when b_i is a continuous block only a single location is required. Otherwise, the translator adds one location for each different control mode of b_i. Invariants and flows of locations are set according to the behavior of b_i. Obviously transitions are required only if b is discrete (hybrid automaton with more than one location). Each transition is associated to a guard that is the condition that drives b_i from a control mode to another. Moreover, according to the must semantics, transitions are declared as urgent. If b_i is an inport or an outport, then the translator adds to the network component containing b_i a corresponding variable with the same name of b_i. Example 1 will clarifies the translation of Simulink basic blocks.

B. Translating must semantics

The Simulink must semantics is modeled by SpaceEx urgent transitions. Notice that, SpaceEx allows only non-strict guards for urgent transitions. This can be a limitation in modeling strict guards. To handle this issue, we consider strict guards under the numerical interpretation of the Simulink semantics, which corresponds to the set of traces generated by the simulation engine through numerical interpretation. Under this interpretation, similarly to the *guard enlargement* considered by the *Almost ASAP* semantics [25], a strict guard can be relaxed and scaled according to the minimum difference d between the values that variables may assume before and after an integration step. Clearly, d depends on the time step δ. Theorem 1 in [12] guarantees that as δ approaches to zero, the numerical interpretation converges to the ideal interpretation. Moreover, d converges to the machine epsilon eps. Hence if we relax each strict guard and scale it by the parameter eps, then the ideal interpretation is preserved. This means that a strict guard on the form $x > 0$ (resp., $x < 0$) is translated into a non-strict guard in the form $x \geq eps$ (resp., $x \leq -eps$). The translation of the Switch block in Example 1 shows a practical case for the described approach.

Example 1 (Translation supported blocks). In this example we consider the Simulink model depicted in Figure 1. The set of blocks B consists of constant, integrator, sum, switch, inport and outport. For the inport In_1 and the outport Out_1, the translator adds the continuous variables In_1 and Out_1. All the other blocks are translated into corresponding and homonyms SpaceEx basic components (see Figure 2(a)). Each basic component consists of a hybrid automaton that models the block behavior (see Figure 2). For example, the SX basic component "integrator" (hybrid automaton shown by Figure 2(c)) consists of a single location and two variables, In_1 and Out_1, to model input and output of the integrator block. The behavior of the integrator is modeled by adding the flow defined by $Out_1' == In_1$. The Switch is, instead an example of a discrete block. It consists of three input and a single output. The translator adds the corresponding variables In_1, In_2, In_3 and Out_1. The two



(a) Placeholder (b) HA manually built

Fig. 4. Example with the non-supported Step block type.

different control modes are modeled by adding the location loc_{01} and loc_{02}. The first control mode c_1 is activated by the condition $G_1 \equiv In_2 \geq 0$ and leads to $Out_1 = In_1$. The second control c_2 mode is activated by the strict condition $G_2 \equiv In_2, < 0$ and leads to $Out_1 = In_3$. Switches between c_1 and c_2 are modeled by urgent transitions, from loc_{01} to loc_{02} and viceversa. The guard of the first transition is trivially G_1. Instead, being G_2 a strict condition, it can not be directly used as guard for the second transition. In this case, according to the description above, the translator adds the guard $In_2 \leq -eps$. Finally, the output is modeled by associating the invariants $Out_1 == In_1$ and $Out_1 == In_3$ to loc_{01} and loc_{02}, respectively.

C. Translating Subsystems

For each block $b \in B$ such that b is a Simulink subsystem, the tool adds to the set $Comp_n$ a network component with the same name of b. In this case, the translator keeps track of the Simulink blocks b_i that belong to b. Then, for each b_i that is not an inport or an outport, the bind (b, b_i) is added to the set $Bind$. At the end of this step, by considering the Simulink diagram depicted by Figure 1, the resulting network will appear like Figure 2(a) (without connections among components).

D. Translating Block Connections (Input/Output among blocks)

Once components are added to the network, it is necessary to decode the set of blocks connections C. The main issue for this task is due to the acasual semantics of SpaceEx, where connections among variables are not oriented. Let $c_{ij} = (b_i, b_j) \in C$ be a Simulink connection that links the output of the *source* block b_i to the input of the *destination* block b_j, inside subsystem b_s and let b'_i, b'_j and b_n the SpaceEx components used to model b_i, b_j and b_s, respectively, and Out_i and In_j be the variables used to model source and destination of c_{ij}. By definition of SpaceEx model, $Map_{n,i}$ contains the mapping between Out_i of b'_i and Out_i of b_n, while $Map_{n,j}$ contains the mapping between In_j of b'_j and In_j of b_n. To model c_{ij} it is necessary to map Out_i with In_j. This is done by mapping the destination variable to the source variable, that is by replacing the mapping of In_j in $Map_{n,j}$ by the mapping between In_j and Out_i. Notice that, if the source Out_i is connected to many destinations In_z (i.e. there exists at least another $c_{iz} = (b_i, b_z) \in C$), the translator replaces each mapping in Map_{nz} that involves variable In_z, by the mapping between In_z and Out_i. This technique does not preserve the names of variables that model outports, that is one of our goal. Indeed, an outport can be only a destination, and hence the corresponding variable will be replaced by the variable that model the source. In order to fix this issue, it is necessary a further step after all mappings are derived. The additional step replaces, for each mapping $Map_{n,i}(Out_i) = Out_j$ such that Out_i is an outport, all the mappings $Map_{n,z}(x) = Out_j$ by $Map_{n,z}(In_z) = Out_i$, where x is a variable that model a source. At the end of this task, all variables Out_i modeling outports, will appear as network interface We show the connections task by Example 2.

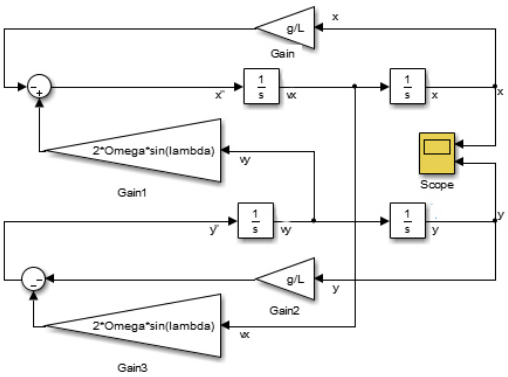
Example 2 (Translating connections). Consider the SpaceEx model in Figure 2(a), and the connection between the switch output (modeled by the SX variable Out_1) and the outport u (modeled by the SX variable u). This connection is modeled by replacing the existing mapping between the network variable u and it self, by the mapping between u and Out_1. Such a mappings configuration affects the preservation of the interface. Indeed the resulting network does not show the variable u (in favor of Out_1) in its interface, by loosing the reference to the variable that model the outport. The additional step described above, regains this preservation requirement, by replacing each occurrence of Out_1 by u. The mapping thus obtained is such that the variable u correctly belongs to the network interface. The SpaceEx model after this task is depicted by Figure 2(a), and it was obtained as output of *SL2SX* on the Simulink diagram of Figure 1 as input.

E. Translating non-supported Blocks

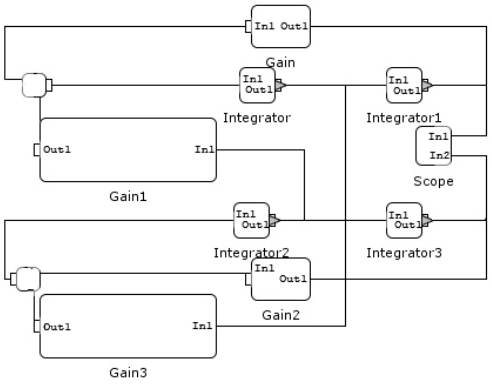
The tool *SL2SX* does not provide support for all the (wide range) of Simulink blocks. When the parser meets a non-supported block $b_i \in B$, the translator arranges in advance a placeholder with proper interface. That is a basic component, with all the necessary variables and mappings, consisting of an empty hybrid automaton. The tool assigns a text note to the component, that can be viewed by the SpaceEx model editor, to warn the user to manually build a HA for modeling the block. Consider for example the Simulink Step block type, that is a discrete block not currently supported by *SL2SX*. Its behavior is defined by a time threshold, an initial and a final value. For all the time before the threshold, it provides the initial value as output. As soon as possible the threshold is met, the final value is given as output. In correspondence of such a block, the tool places a corresponding basic component with proper interface and connections (shown in Figure 4(a)). Then, to model the block, the user could build a HA similar to that shown in Figure 4(b), whose transition is clearly defined as urgent.

IV. Case Studies

In this Section we illustrate applications of the tool *SL2SX*, via two Simulink diagrams provided by the Mathworks examples library. We show that the important goal (according to [24]) of preserving structure is achieved. The obtained SpaceEx models are then subjected to simulation, by showing that results are the same as the simulation on the corresponding Simulink diagrams. Finally, we perform

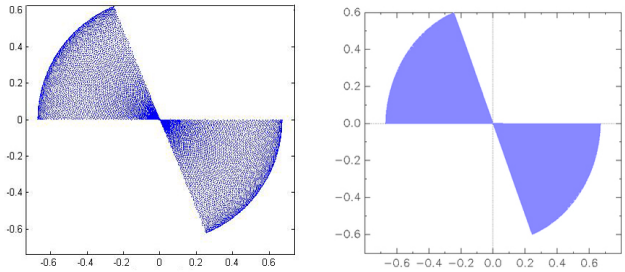


(a) Simulink diagram in input



(b) SX output model generated by *SL2SX*

Fig. 5. Simulink and SX models for the Foucault pendulum.



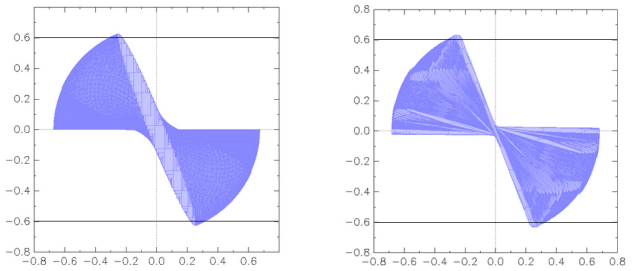
(a) Simulation by Simulink engine. (b) Simulation by SpaceEx engine.

Fig. 6. Positions of the Foucault Pendulum Bob in the xy-Plane over 6 hours (21600 sec).

reachability analysis on the obtained SpaceEx models, by introducing perturbations on the initial set of states.

A. Foucault Pendulum

The Simulink diagram that will be shown in this section is part of the Mathworks examples library, and it is used to model a Foucault pendulum [15]. More details, like system parameters and initial conditions, are described in [15]. Figure 5(a) shows the Simulink diagram used to model the Foucault pendulum, while Figure 5(b) depicts the SpaceEx model produced by the tool *SL2SX*. Notice that the structure is preserved. In order to perform the simulation on the SpaceEx model, it is necessary to correctly establish some parameters of the simulations (like time steps and simulation



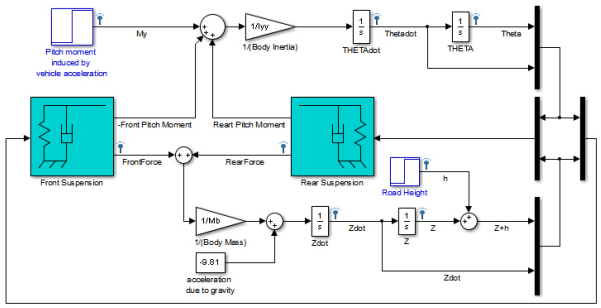
(a) LGG algorithm, with sample time of 0.01 and flowpipe tolerance of 0.01

(b) STC algorithm, with flowpipe tolerance of 0.5

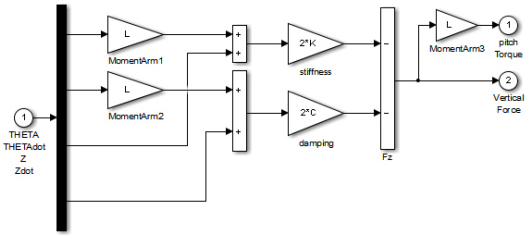
Fig. 7. Reachability analysis (box directions) of the Foucault Pendulum Bob in the xy-Plane over 6 hours, with no-rigid wire.

stop time) and the initial state. The tool *SL2SX* is designed to provide a support for this, by writing a specific configuration file containing some of these information. Currently the tool is able to automatically carry out parameters for the simulation. Initial states, indeed, needs to be manually written (we are currently working on automating this task). The initial state can be easily derived from the initial values of Simulink variables. In particular, the initial state for this example is defined with $x = 0.67$ (length of the pendulum string), $y = 0$ (y coordinate) and $Integrator1Out_1 = Integrator3Out_1 = 0$ (initial velocities in the two coordinates).

Figure 6 shows the comparison between the simulation results obtained by the two tools, with a simulation stop time of 6 hours. Figure 6(a) depicts result of Simulink simulation, plotted on the xy-coordinates assumed by the pendulum, while Figure 6(b) depicts the simulation with same parameters performed by SpaceEx, showing that the behavior of the two models is the same. According to the deterministic semantics of Simulink, the simulation model can only consider the ideal case of anelastic wire (length always fixed at 67 meters). On the other side, the verification model can easily takes into account an elastic wire, whose factors that determine the wire expansions and contractions are modeled as uncertain in the initial condition. In practice, to model a wire whose elasticity coefficient is such that the length can variate between 66.9 and 67.1 meters, the initial condition is extended from the single point $x = 0.67$ to the interval $x \in [0.669, 0.671]$. Figure 7 depicts the reachability analysis on such a model, by showing how the uncertain could affect the results. In particular, Figure 7(a) shows the reachability set computed by the LGG algorithm, with 0.01 sample time and 0.01 flowpipe tolerance, while Figure 7(b) shows reachability set computed by STC algorithm, with 0.5 as flowpipe tolerance. Both sets are obtained by considering box template directions and 6 hours of time horizon. About the performance, LGG scenario requires 8.1 seconds, while STC scenario needs 5.4 seconds. Notice that, because of the potential variation on the wire length, the reachable set contains not only the origin $(0, 0)$, but all the points inside a circle around the origin, with a ray of 0.1 meters. Moreover, the pendulum may overpass the limit of 60 meters, again because of the extension.

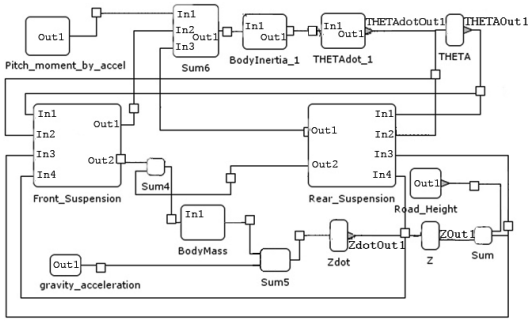


(a) Simulink Main system.

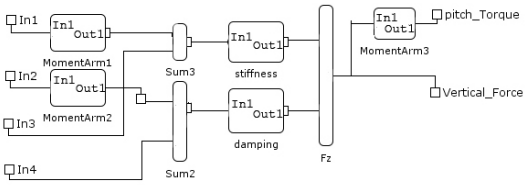


(b) Subsystem for front and rear suspensions.

Fig. 8. Simulink diagram for automotive suspension.

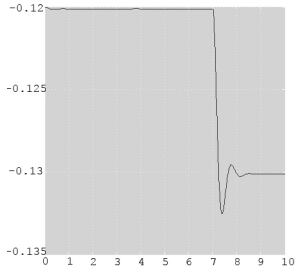


(a) SX network for the main automotive suspension.

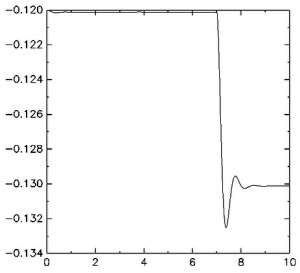


(b) SX network for front suspension subsystem.

Fig. 9. SX suspensions model obtained after user completion.



(a) Simulink simulation.



(b) SpaceEx simulation.

Fig. 10. Vertical bounce over the 10 sec. for the suspension models.

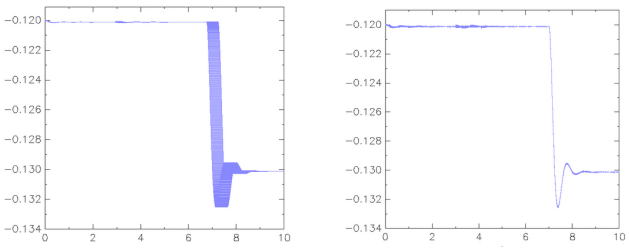
B. Automotive Suspension

In this Section, we consider a Simulink diagram used to design a simplified half car model that includes an independent front and rear vertical suspensions. The model also includes body pitch and bounce degrees of freedom. It can be used to investigate ride characteristics and longitudinal shuffle resulting from changes in throttle setting. More details, as well as system parameters and initial states, are available in [14]. The suspension model has two inputs. The first one is the road height and is modeled by a step block that corresponds to the vehicle driving over a road surface with a step change in height. The second one is the horizontal horizontal force acting through the center of the wheels that results from braking or acceleration maneuvers. This input appears only as a moment about the pitch axis because the longitudinal body motion is not modeled. Notice that, due to the presence of discrete components (i.e. the two steps for modelling road height and horizontal force), the system is purely hybrid. Figure 8 show the main Simulink diagram, while Figure 8(b) depicts the Simulink subsystem that models both front and rear suspensions.

After converting the Simulink diagram by using $SL2SF$, we obtain a SpaceEx model that needs to be completed by the user, because of the presence of non supported blocks (i.e. step, mux and demux). The completion task consists of removing the components for mux and demux, and directly mapping the involved variables. This requires, for each input of the mux, the introduction of an input variable to all the components connected to the mux input. Hence, the variables In_1, In_2, In_3 and In_4 are added to both the networks for the front and the rear suspension. Then these variables are directly mapped to the variables that models the source connection of the mux, that are $ThetaOut_1$, $ThetadotOut_1$, $ZOut_1$, and $ZdotOut_1$, respectively.

Figure 9(a) shows as the SX main system appears after modeling (removing and variables mapping) mux/demux blocks. The completion task also consists of building a hybrid automaton to model the non-supported step block type. A possible solution may consist in the hybrid automaton depicted by Figure 4(b). The automaton also needs to be parametrized, according to the related parameters (i.e. step time, init and final values).

The Simulink and the SpaceEx models are then compared via simulations. Figure 10 depicts the vertical bounce variations during the first 10 seconds of simulation, with step time set to 0.01. The comparison shows that results carried out by Simulink and SX engines (see Figure 10(a) and 10(b), respectively) are absolutely comparable. Analogously to the Foucault Pendulum, it is possible to perturb the system by setting the initial front pitch into the interval $[0, 0.3]$ (instead of the single initial value 0). Figure 11 shows the reachable sets for the vertical bounce over the first 10 seconds of activity, computed by the LGG and the STC algorithms (requiring 0.55 and 0.41 sec, respectively).



(a) LGG scenario with sample time of 0.01 and flowpipe tolerance of 0.001

(b) STC scenario with flowpipe tolerance of 0.001

Fig. 11. Reachability analysis for verticals bounce over 10 seconds, considering perturbation on the front pitch.

V. Conclusion and Future Work

In this work we described a technique to translate a subset of Simulink blocks into verification model compatible with SpaceEx. The technique is implemented by the tool *Simulink To SpaceEx (SL2SX)*, that takes a Simulink diagram as input, and generates a verification model with the aim of perform reachability analysis by one of the algorithms provided by SpaceEx. The tool is designed to try to overcome some classical limitations in the transition from simulation to verification models, by building a network of HA that exhibits the same hierarchical structure of the Simulink model in input, and whose components and variables have the same names, graphical dimensions and positions, of the corresponding Simulink elements.

We think that our tool is able to speedup and limit the errors of the translation process, and to help the user in the completion task.

Some results are presented, showing that the approach could be promising and effective. Moreover, SpaceEx will be able to handle urgency on affine HA, by allowing to perform reachability analysis on the obtained verification models even with more complex dynamics (i.e. affine).

We are planning to extend the tool by automating the identification of initial states, and by adding support for more blocks. In particular, we are currently investigating a translation for Stateflow components [16].

References

[1] A. Agrawal, G. Simon, and G. Karsai. Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. *Electron. Notes Theor. Comput. Sci.*, 109:43–56, December 2004.

[2] R. Alur, A. Kanade, S. Ramesh, and K.C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 89–98, New York, NY, USA, 2008. ACM.

[3] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196. NASA Langley Research Center, jun 2000.

[4] O. Bouissou, S. Mimram, and A. Chapoutot. Hyson: Set-based simulation of hybrid systems. In *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, pages 79–85, Oct 2012.

[5] A. Chutinan and B. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In FritsW. Vaandrager and JanH. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 76–90. Springer Berlin Heidelberg, 1999.

[6] 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 *CAV 11: Proc. of 23rd Conf. on Computer Aided Verification*, pages 379–395, 2011.

[7] Goran Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.

[8] T.A. Henzinger. The theory of hybrid automata. In *11th IEEE Symp. Logic in Comp. Sci.*, pages 278–292, 1996.

[9] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.

[10] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250 – 262, 2010.

[11] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid csp. In Kazunori Ueda, editor, *Programming Languages and Systems*, volume 6461 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2010.

[12] K. Manamcheri. *Translation of Simulink/Stateflow models to hybrid automata*. PhD thesis, Graduate College of the University of Illinois, 2011.

[13] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from simulink/stateflow models. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, HSCC '11, pages 317–318, New York, NY, USA, 2011. ACM.

[14] Mathwork. SL model for automative suspension. `http://www.mathworks.com/help/simulink/examples/automotive-suspension.html`.

[15] Mathwork. SL model for Foucault pendulum. `http://www.mathworks.com/help/simulink/examples/modeling-a-foucault-pendulum.html`.

[16] MathWorks. Mathworks stateflow: Design and simulate state machines, September 2012. `http://www.mathworks.fr/products/stateflow/`.

[17] MathWorks. Mathworks simulink: Simulation et model-based design, March 2014. `www.mathworks.fr/products/simulink`.

[18] S. Minopoli and G. Frehse. SL2SX tool and case study. `http://www-verimag.imag.fr/~minopoli/SL2SXdemo.zip`.

[19] S. Minopoli and G. Frehse. Non-convex invariants and urgency conditions on linear hybrid automata. In *Formal Modeling and Analysis of Timed Systems*, pages 176–190, 2014.

[20] S. Mitra. *A verification framework for hybrid systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, September 2007.

[21] B.I. Silva, K. Richeson, B. H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using checkmate. In *ADPM*, 2000.

[22] MoBIES team. Hsif semantics. Technical report, University of Pennsylvania, 2002.

[23] A. Tiwari. Formal semantics and analysis methods for Simulink Stateflow models. Technical report, SRI International, 2002.

[24] M. W. Whalen, A. Murugesan, S. Rayadurgam, and M. P. E. Heimdahl. Structuring simulink models for verification and reuse. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, MiSE 2014, pages 19–24, New York, NY, USA, 2014. ACM.

[25] Martin Wulf, Laurent Doyen, and Jean-Franois Raskin. Almost asap semantics: From timed models to timed implementations. In *HSCC'04*, volume 2993 of *LNCS*, pages 296–310. Springer, 2004.

[26] Liang Zou, N. Zhany, Shuling Wang, M. Franzle, and Shengchao Qin. Verifying simulink diagrams via a hybrid hoare logic prover. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.