Verification of embedded control programs

Thao Dang[1], Bertrand Jeannet[2], and Romain Testylier[1]

Abstract— **In this paper we are concerned with the problem of verifying embedded control programs. The approach we use combines the logico-numerical techniques developped for the verification of Lustre programs and the set-based image computation for continuous systems. The practical interest of this approach lies in the fact that there exists a tool for generating Lustre code for controllers described in Simulink. We also illustrate the approach with some experimental results obtained for a robotic controller for LEGO Mindstorm.**

I. Introduction

Model-based design has emerged as an important paradigm in the development of cyber-physical systems. Its essence is the use of models in all the steps of the development process, from design to implementation. Using models is key to keeping the development costs manageable, since correcting bugs on models is much cheaper than on real protypes. This paradigm can however be effective only when it is accompanied by powerful design tools for simulation, verification, synthesis, code generation, etc. In a cyber-physical system, the control and computation part is often realized as computer programs that will be executed to control the physical part. The design of the control and computation part can be assisted by a variety of tools, among which Simulink has become a de facto standard in industry and is extensively used in various application domains. Simulink supports the description and analysis of models in a graphical block-diagram language. The Simulink environment provides extensive capabilities for simulation and control design based on advanced algorithms. Embedded code generation from Simulink models is supported by tools, such as Real-Time Workshop Embedded Coder from Mathworks, TargetLink from dSpace. Although these code generators translate models into production-quality C code that can be deployed onto embedded systems, their major problem with respect to safety critical applications is the preservation of semantics. The question of fidelity between the generated code and the simulated models, that is "what is simulated is what is executed", is not addressed by these tools. For this reason, an alternative approach is to translate Simulink models to an intermediate formal language, which permits formal reasoning on the relation between the models of each step in the design process. This work is developed within the framework of model-based development of embedded control design and we use Lustre as intermediate formal language and a tool for translating Simulink models to Lustre. The choice of Lustre in this approach is motivated by its formal semantics and a number of associated formal validation tool[1]. Lustre is also supported by a code generator from SCADE[2] which is compliant the DO-178B Level A standard[3].

This work contributes a method for the verification of Lustre programs generated from controller models specified in Simulink. Such Lustre programs contain both Boolean and numerical variables. Their verification often requires an exhaustive enumeration of modes corresponding to all possible values of the Boolean variables, which often leads to a state-space explosion. In this paper we employ a technique for verification of logico-numerical programs without resorting to the enumeration of the Boolean state space. On the other hand, to extend the technique to programs with non-linear computation, we combine it with a method for set-based image computation for polynomial functions.

II. From Simulink controller models to Lustre programs: a robotic example

A controller designed in Simulink can be translated to a Lustre program using the tool S2L [14]. The tool translates only the discrete-time blocks of Simulink, since implemented controllers are in discrete time. This means that to be translated, a controller that is designed in continuous time needs to be discretized using a sampling method. In addition, only a "safe" subset of Simulink blocks can be translated. This restriction is needed to guarantee semantics preservation, that is the "informal" semantics of Simulink specified by the behavior of the simulator for a given set of simulation parameters and options.

Throughout the paper we use a case study of robotic planning to explain our method. This case study is simple (compared to advanced control and planification methods) but sufficiently complex to illustrate the interest of our method. The goal of this case study is to verify a controller for a LEGO-Mindstorm robot. The robot is in form of a two-wheeled vehicle and should follow a black line on the ground while avoiding obstacles. The trajectory of the robot is defined by 3 variables: its position (x, y) on the plane and its orientation θ (compared to the x axis). These state variables are controlled by two control inputs: the speeds v_l and v_r of the left and right wheels. The dynamics of the

[1]See http://www-verimag.imag.fr/Tools,36.html for the available tools for Lustre

[2]SCADE is the graphical version of Lustre commercialized by Esterel Technologies, which is used in highest criticality applications, such as European avionic projects (Airbus A340-600, A380, Eurocopter).

[3]This industrial standard specifies objectives across the development cycle to achieve flight software certification.

robot is described by the following differential equations:

$$\begin{aligned}\dot{x} &= (v_l + v_r)cos(\theta)/2\\ \dot{y} &= (v_l + v_r)sin(\theta)/2\\ \dot{\theta} &= (v_r - v_l)/l\end{aligned}$$

where l is the distance between the two wheels.

The robot is equipped with two light sensors and one ultrasonic sensor. The light sensors are used to measure the deviation of the robot from the black line, and the ultrasonic sensor to detect the presence of an obstacle in front of the robot. A controller was designed for this control objective as follows.

The controller consists of two modes: one is for following the line, and the other is for avoiding obstacles. The controller takes as input the data from the sensors and computes the desired speeds v_l and v_r of the left and right wheels which are realized by the robot motor. The control law for the line-following mode is:

$$\begin{aligned}v_d &= 50c_d + 50c_d^2\\ v_g &= 50c_g + 50c_g^2\end{aligned}$$

where c_d and c_g are the values of the right and left light sensors (ranging in the interval $[0, 1]$). The control law for the mode of obstacle avoidance mode is as follows. When an obstacle is detected, the robot should first stop, make 180-degree turn (for example towards the left), and then continue to follow the black line in the other direction. Since there is a delay in the motor's actions, it is important to wait until the robot stops fully before turning around (otherwise, turning aroung while advancing may make the robot lose track of the black line). In addition, the 180-degree turn ends when a sequence of colors White-Black-White is detected on the left light sensor.

The above controller was implemented as a Simulink model. In order to analyze the behavior of the controller in the closed loop, the dynamics of the robot and the outputs c_d and c_g of the light sensors are also emulated by a Simulink model. Since the proposed verification method works for polynomial dynamics, the cosinus and sinus functions are approximated by piecewise linear functions.

The resulting model of the closed loop system contains many switch blocks and Boolean operators (such as to encode the piecewise linear approximation and detection of a White-Black-White sequence). The Simulink model can be translated to a Lustre program using the tool S2L.

Such a Lustre program corresponds to a discrete-time hybrid automaton, which can be defined by exhautively enumerating all the possible Boolean values in the program to construct a control flow graph with only numerical variables. Although the above controller is simple in terms of control laws and planification strategies, by a simple exploration of the control flow graph of a medium-sized Lustre program, the discrete structure of the corresponding hybrid automata can already be very large and this discrete structure may contain a lot of modes sharing the same continuous dynamics. To address this problem, we employ the logico-numerical techniques for Lustre programs.

III. Techniques for verification of logico-numerical programs

A Lustre program defines a set of equations:

$$\begin{aligned}s_1 &= f_1(s_1, \ldots, s_n, u_1, \ldots, u_m)\\ &\ldots\\ s_n &= f_n(s_1, \ldots, s_n, u_1, \ldots, u_m\end{aligned}$$

where S_i are variables and u_i are input variables. The variables in Lustre are also called flows and they can be Boolean and numerical. A flow is a partial function $y : N \rightarrow D_y$ where where N is the set of natural numbers modelling discrete time, and D_y is the domain of y. Besides the usual arithmetic operations and control-flow operators (e.g., if then else), the functions f_i are made up of a number of temporal operators, in particular the pre operator which is used to specify a delay of one time unit, that is $pre(y)$ defines a flow z such that the value of z at instant k is equal to the value of y at instant $(k-1)$.

The compilation of Lustre programs produces a symbolic transition systems with the following components: $\mathbf{s} = (\mathbf{b}, \mathbf{x})$ is the vector of the state variables composed of Boolean variables $\mathbf{b}$ and numerical variables $\mathbf{x}$, $\mathbf{u} = (\mathbf{u}_b, \mathbf{u}_x)$ is the vector of the input variables composed of Boolean variables $\mathbf{u}_b$ and numerical variables $\mathbf{u}_x$, $\mathcal{C}(\mathbf{x}, \mathbf{u}_x)$ is the set of Boolean constraints over the numerical variables. The transitions are specified as:

$$\mathcal{A}(\mathbf{b}, \mathbf{u}_b, \mathcal{C}) \rightarrow \begin{pmatrix}\mathbf{b}'\\ \mathbf{x}'\end{pmatrix} = \begin{pmatrix}f^b(\mathbf{b}, \mathbf{u}_b, \mathcal{C})\\ f^x(\mathbf{b}, \mathbf{u}_b, \mathcal{C}, \mathbf{x}, \mathbf{u}_x)\end{pmatrix}$$

where $\mathcal{A}(\mathbf{b}, \mathbf{u}_b, \mathcal{C})$ is an assertion constraining the input variables depending on the state variables. The numerical transition functions are written as a disjunction of guard actions: $f^x(\mathbf{b}, \mathbf{u}_b, \mathcal{C}, \mathbf{x}, \mathbf{u}_x) = \bigvee_i g_i(\mathbf{b}, \mathbf{u}_b, \mathcal{C}) \rightarrow a_i(\mathbf{x}, \mathbf{u}_x)$. An execution of such a program is a sequence

$$\mathbf{s}^0 \xrightarrow{\mathbf{u}^0} \mathbf{s}^1 \xrightarrow{\mathbf{u}^1} \mathbf{s}^2 \ldots$$

such that

$$\forall k \geq 0, \mathcal{A}(\mathbf{b}^k, \mathbf{u}_b^k, \mathcal{C}) \rightarrow \mathbf{s}^{k+1} = f(\mathbf{s}^k, \mathbf{u}^k)$$

where $f = (f^b, f^x)$ is the vector of both Boolean and numerical transition functions. In this work, to define the control flow graph, we consider partitions defined by equivalence relations on the Boolean state variables. A fully partitioned control flow graph constructed by enumerating all Boolean states is charaterized by the following equivalence relation:

$$\mathbf{b}_1 \sim \mathbf{b}_2 \Leftrightarrow \mathbf{b}_1 = \mathbf{b}_2.$$

To combine symbolically discrete and continuous behaviors, we adopt the abstract interpretation approach for approximations, with adaptative techniques to improve the precision when necessary. Concerning enumerating continuous modes, based on the observation that, in cyber-physical

systems, many discrete states are likely to share the same continuous dynamics, and that the number of different dynamic behaviours remains small in many cases, we can build an automaton with one location per continuous mode, and use techniques for discrete systems to handle with the complex transitions generated by the controller program.

On the other hand, separating discrete modes only when they do not share the same continuous mode may lead to a quite abstract view of the system w.r.t. the qualitative behaviors. We adapted the techniques developped for purely discrete systems [8] to refine the control structure w.r.t. precision requirements. In order to choose automatically a suitable partition for the verification of a safety property, that is precise enough to conclude if the property holds, and also the coarsest possible to limit the complexity of analysis. the adopted solution consists in starting from a very coarse partition (basically, distinguishing only between initial states, bad states and others), and in refining it dynamically according to heuristics. The above mentioned techniques for verification of logico-numerical programs were implemented in the tool NBAC [9], [8], which can also be used for hybrid systems with piecewise-constant dynamics. In this work, with view to verification of embedded control programs with nonlinear numerical transition functions, we combine the tool with a method for computing the image of polynomial transition functions.

Before continuing, we briefly discuss related work regarding symbolic treatment of discrete dynamics. The standard approach to combine continuous and (finite) discrete dynamics consists in encoding the values of the finite-state discrete variables in the control structure of the hybrid systems [7], [5]. However, this easily results in a combinatorial explosion of the number of control locations. More recently, [3] proposed a fully symbolic technique based on backward (greatest) fixpoint computation, in which sets of states are represented with a variant of Boolean circuits mixing Boolean variables and linear constraints. The technique relies on a semi-canonical representation for sets of states.

IV. Image computation for parametric polynomial dynamical systems

In this section, we present a technique for computing the image of a polynomial function, which is used to handle transition functions of Lustre programs. We also augment the framework with uncertain parameters, since they are very useful in practice to model imprecision in modelling. Compared to the technique presented in [11], this technique is more efficient in term of accuracy since it handles parameters in a more symbolic way.

We consider the following numerical transition function:

$$\mathbf{x}(k+1) = \pi(\mathbf{x}(k), \mathbf{p}) \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state variables, $\mathbf{p} \in P \subseteq \mathbb{R}^m$ is the vector of uncertain parameters. The set P is called the parameter set. We assume that P is a convex polyhedron. The initial state $\mathbf{x}(0)$ is inside some set $X_0 \subset \mathbb{R}^n$. Note that the above function π plays the role of a guard action in the symbolic transition system associated with a Lustre program.

One way to approximate the image $\pi(X)$ is to use special convex polyhedra with fixed geometric form, called template polyhedra [13], [2]. A template is a set of linear functions defined by an $l \times n$ matrix H and a real-valued vector $\mathbf{c} \in \mathbb{R}^l$. Let H^i be the i^{th} row of the matrix H and H^i_k its k^{th} element, the corresponding template polyhedron is defined by

$$\langle H, \mathbf{c} \rangle = \{\mathbf{x} \mid \bigwedge_{i=1,\ldots,l} H^i\mathbf{x} \leq c_i\}.$$

The vector $\mathbf{c}$ is called a polyhedral coefficient vector. By choosing the templates it is possible to control both the precision and the geometric complexity of the approximations. However this problem needs a deep investigation and is not a topic of this paper.

The template matrix H is assumed to be given (which can be composed of the normal directions of regular polyhedra); to over-approximate the image $\pi(X)$, the polyhedral coefficient vector $\mathbf{c} = (c_1, \ldots, c_n)^T$ can be determined as the solution the following optimization problems:

$$\forall i \in \{1, \ldots, l\}, c_i = \max(\Sigma_{k=1}^n H^i_k \pi_k(\mathbf{x}, \mathbf{p})) \tag{2}$$

$$\text{subj. to } \mathbf{x} \in X, \mathbf{p} \in P. \tag{3}$$

This polynomial optimization problem is computationally difficult, we propose to use a linear relaxation in order to take advantage of well-developed linear programming techniques [12]. To this end, the Bernstein expansion can be used to compute affine bound functions of polynomials.

The essence of this method, proposed in [6], for computing an affine bound function is to find a hyperplane that is close to *all* the control points, using linear least squares approximation. For our problem, we need to extend this method to parametric polynomial functions.

A multi-index $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n)$ is a vector of n non-negative integers. Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$, we write $\mathbf{i} \leq \mathbf{d}$ if for all $j \in \{1, \ldots, n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. The parametric polynomial $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ can be represented using the power base as

$$\pi(\mathbf{x}, \mathbf{p}) = \sum_{\mathbf{i} \in I_\mathbf{d}} \mathbf{a_i}(\mathbf{p})\mathbf{x}^\mathbf{i}$$

where $\mathbf{a_i}$ is a function $\mathbb{R}^m \to \mathbb{R}$; $I_\mathbf{d}$ is the set of *all* multi-indices $\mathbf{i} \leq \mathbf{d}$, that is

$$I_\mathbf{d} = \{\mathbf{i} \mid \mathbf{i} \leq \mathbf{d}\}.$$

In this work, we consider only a linear dependence of the coefficients $\mathbf{a_i}(\mathbf{p})$ on the parameters $\mathbf{p}$. The multi-index $\mathbf{d}$ is called the *degree* of π. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the $\mathbf{i}^{th}$ Bernstein polynomial of degree $\mathbf{d}$ is defined as follows:

$$\mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(x_1) \ldots \beta_{\mathbf{d}_n,\mathbf{i}_n}(x_n)$$

where for a real number y,

$$\beta_{\mathbf{d}_j,\mathbf{i}_j}(y) = \binom{\mathbf{d}_j}{\mathbf{i}_j} y^{\mathbf{i}_j}(1 - y^{\mathbf{d}_j - \mathbf{i}_j}),$$

and $\binom{\mathbf{d}}{\mathbf{v}}$ denotes $\binom{\mathbf{d}_1}{\mathbf{i}_1}\binom{\mathbf{d}_2}{\mathbf{i}_2}\dots\binom{\mathbf{d}_n}{\mathbf{i}_n}$. Then, for all $\mathbf{x}$ inside the unit box $\mathcal{B} = [0,1]^n$, the polynomial π can be written using the Bernstein expansion as follows:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i}\in I_{\mathbf{d}}} \mathbf{g_i}(\mathbf{p})\mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x})$$

where for each $\mathbf{i} \in I_{\mathbf{d}}$ the Bernstein coefficient $\mathbf{g_i}(\mathbf{p})$ is defined as:

$$\mathbf{g_i}(\mathbf{p}) = \sum_{\mathbf{j}\le\mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}}\mathbf{a_j}(\mathbf{p}). \tag{4}$$

Note if all the coefficients $\mathbf{a_j}$ depend linearly on $\mathbf{p}$, so do the Beinstein coefficients. The Bernstein coefficients allow to derive useful geometric properties, and we now show that they can be used to efficiently compute an affine approximation of the polynomial π. Since the Bernstein coefficients reflect the form of the image set, we will derive the lower affine function that fits "best" these Bernstein coefficients using linear least squares. Note that in the method proposed in [] the direction of the bound function is determined for some midpoint in the parameter set and then is shifted down to remain a lower bound for all other parameters $\mathbf{p}$. In this work, we improve this method by treating the parametes in a symbolic way.

Let the set of multi-indices be written as $I_{\mathbf{d}} = \{\mathbf{i}^j \mid 1 \le j \le n_b\}$, and the set of all control points are denoted similarly. Let A be a matrix of size $n_b \times (n+1)$ (n is the number of state variables of the dynamical systems in question) such that its elements are defined as follows. For all $1 \le j \le n_b$ and $1 \le k \le n$, $A^j_k = \frac{\mathbf{i}^j_k}{\mathbf{d}_k}$ and $A^j_{n+1} = 1$. Let $\zeta(\mathbf{p})$ be the solution of the following linear least squares approximation problem:

$$A^T A\zeta(\mathbf{p}) = A^T\mathbf{g}(\mathbf{p}).$$

Then, we can write $\zeta(\mathbf{p}) = (A^TA)^{-1}A^T\mathbf{g}(\mathbf{p})$ where $(A^TA)^{-1}$ is the MoorePenrose pseudoinverse[4] of A. From $\zeta(\mathbf{p})$, we can define an affine function

$$\tilde{l}(\mathbf{x}) = \sum_{k=1}^{n} \zeta_k(\mathbf{p})\mathbf{x}_k + \zeta_{n+1}(\mathbf{p})$$

which corresponds to the "median" axis of the convex hull of all the control points. Its distance to a control point $\mathbf{g}^j(\mathbf{p})$ is

$$\tilde{l}(\frac{\mathbf{i}^j}{\mathbf{d}}) - \mathbf{g}^j(\mathbf{p}) = \sum_{k=1}^{n} \zeta_k(\mathbf{p})\mathbf{x}_k + \zeta_{n+1}(\mathbf{p}) - \mathbf{g}^j(\mathbf{p}).$$

It now remains to shift it downward by the amount:

$$\begin{aligned}\delta \quad = max\{&\textstyle\sum_{k=1}^{n} \zeta_k(\mathbf{p})\frac{\mathbf{i}^j}{\mathbf{d}} + \zeta_{n+1}(\mathbf{p}) - \mathbf{g}^j(\mathbf{p}) \mid \\ &0 \le j \le n_b \ \wedge \ \mathbf{p} \in P\}\end{aligned}$$

[4]Note that $\zeta(\mathbf{p})$ can be computed using the Cholesky decomposition or using two steps: first solving $R^T\mathbf{z} = A^{\mathrm{T}}\mathbf{g}(\mathbf{p})$ for $\mathbf{z}$, and then solving $R\zeta(\mathbf{p}) = \mathbf{z}$ for $\zeta(\mathbf{p})$.

where $\frac{\mathbf{i}^j}{\mathbf{d}} = (\mathbf{i}^j_1/\mathbf{d}_1, \dots, \mathbf{i}^j_n/\mathbf{d}_n)$. Due to linear dependence of the Beinstein coefficients on $\mathbf{p}$, the above optimization problem is a set of linear programs and can thus be solved efficiently. This results in a parametric lower bound function

$$l(\mathbf{x},\mathbf{p}) = \tilde{l}(\mathbf{x},\mathbf{p}) - \delta, \text{ for all } \mathbf{x} \in \mathcal{B}.$$

As mentioned earlier, the methods to compute affine bound functions for polynomials can be applied only when the set P is inside the unit box $\mathcal{B}$ anchored at the origin. To extend it to polyhedral domains, we transform the polyhedra to the unit box by two methods: (1) via an (oriented) box approximation, and (2) by rewriting the polynomials using a change of variables. In this paper, we describe only the second method.

The initial polyhedron X can be mapped to the unit box $\mathcal{B}$ by a change of variables. We assume that the polyhedron X is bounded and let $V = \{\mathbf{v}_1, \dots, \mathbf{v}_l\}$ be the set of its vertices. We first express the coordinates of a point $x \in X$ as a linear combination of the vertices of X, that is $\mathbf{x} = \sum_{j=1}^{l} \alpha_j\mathbf{v}_j = \nu(\alpha_1, \dots, \alpha_l)$ such that

$$\forall j \in \{1, \dots, l\} \ \alpha_j \ge 0 \tag{5}$$

$$\sum_{j=1}^{l} \alpha_j = 1. \tag{6}$$

We then substitute $\mathbf{x}$ in π with $\nu(\alpha_1, \dots, \alpha_l)$ to yield a new polynomial in $\alpha_1, \dots, \alpha_l$. It is not hard to see that a bound function computed for the polynomial $\xi(\tilde{\alpha})$ on this unit box is also a bound function for the original polynomial π on the polyhedron P

The above image computation is similar to a number of existing numerical techniques in the use of linear approximation. Its novelty resides in the efficient way of computing linear approximations. Indeed, a common method to approximate a non-linear function by a piecewise linear one, as in the hybridization approach [1], [4] for hybrid systems, requires non-linear optimization. In the hybrid systems verification, polynomial optimization is used to compute barrier certificates [10].

V. Experimentation

We have integrated the above image computation method with the tool NBAC and tested the resulting prototype on the controller for LEGO robot trajectory planning.

To prove that the controller assures that the robot follows the line, we verify for a family of black lines that the distance from the centroid of the robot to the line never exceeds some threshold (which is 0.1 for this experimentation). The Lustre program, shown in Appendix, contains three nodes corresponding to Simulink subsystems: *controller* (that computes the wheel speeds), *position* (that emulates the evolution of the position of the robot), *track* (that emulates the output of the light sensors). The assertions in the program specify the constraints of the initial conditions. The execution of the prototype tool allowed to detect a violation of the desired property, as shown by a divergence of the distance of the distance from the centroid of the robot to the black line in Figure 1.

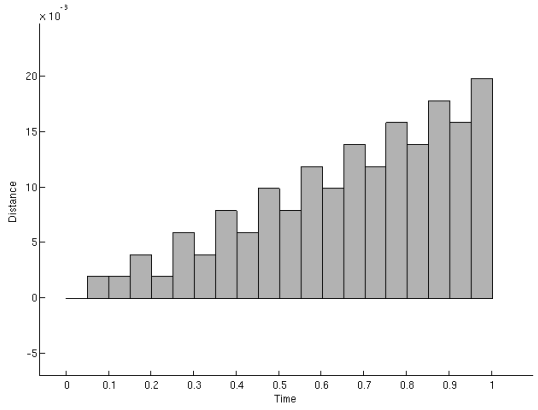

Fig. 1. The evolution of the distance from the centroid of the robot to the black line (for the first 20 iterations).

VI. Conclusion

In this paper we described an approach to verification of embedded control programs with nonlinear transition functions. An important advantage of the approach is that it can handle practical control systems described in Simulink. We intend to explore more case studies in order to evaluate the prototype tool. As future work, we plan to investigate spectific treatments for common operations in a program (such as resets and equalities) that may make the resulting optimization problems become flat.

References

[1] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica.*, 43(7):451–476, 2007.

[2] Liqian Chen, Antoine Miné, Ji Wang, and Patrick Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *SAS*, volume 5673 of *Lecture Notes in Computer Science*, pages 309–325. Springer, 2009.

[3] Werner Damm, Stefan Disch, Hardi Hungar, Swen Jacobs, Jun Pang, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In *Automated Technology for Verification and Analysis, ATVA'07*, volume 4762 of *LNCS*, 2007.

[4] Thao Dang and Romain Testylier. Hybridization domain construction using curvature estimation. In *Proceedings HSCC 2011*. ACM, 2011.

[5] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10:263–279, 2008.

[6] J. Garloff and A.P. Smith. Rigorous affine lower bound functions for multivariate polynomials and their use in global optimisation. In *Proceedings of the 1st International Conference on Applied Operational Research, Tadbir Institute for Operational Research, Systems Design and Financial Services*, volume 1 of *Lecture Notes in Management Science*, pages 199–211, 2008.

[7] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Int. J. on Software Tools for Technology Transfer*, 1:110–122, 1997.

[8] B. Jeannet. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. *Formal Methods in System Design*, 23(1):5–37, July 2003.

[9] B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In *Static Analysis Symposium, SAS'99*, volume 1694 of *LNCS*, Venezia (Italy), September 1999.

[10] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004.

[11] Romain Testylier and Thao Dang. Analysis of Parametric Biological Models. In *Workshop on Hybrid Systems and Biology HSB 2012*. Springer, 2012.

[12] S. Boyd and S. Vandenberghe. *Convex optimization*. Cambridge Uni. Press, 2004.

[13] S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005)*, LNCS 3385. Springer, 2005.

[14] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic. Translating discrete-time simulink to lustre. *ACM Trans. Embed. Comput. Syst.*, 4(4):779–818, November 2005.

Appendix

```
node system(orientation0:real)
returns (ok:bool;
 posx,posy,orientation,
 cd,cg,dist,d,
 ud,ug:real)
let
assert(orientation0>-5.0*pi/180.0
      and orientation0<5.0*pi/180.0);

assert(y0>=-0.1 and y0<=-0.1);

(ud,ug) = controller(cd,cg,dist);

(posx,posy,orientation) = position(0.0,
                  0.0,
                  orientation0,
                  0.0 -> (pre ud),
                  (0.0 -> (pre ug)));

(cd,cg,dist,d) = track(0.0 -> pre posx,
                 0.0 -> pre posy,
                 0.0 -> pre orientation);

ok = true -> (pre ok) and
             (d)>=-0.004 and
             (d)<=0.004;
tel
```