Time-Aware Abstractions in HybridSal*

Ashish Tiwari

SRI International, Menlo Park, CA

Abstract. HybridSal is a tool for enabling verification of hybrid systems using infinite bounded model checking and k-induction. The core component of the tool is an abstraction engine that automatically creates a discrete, but infinite, state transition system abstraction of the continuous dynamics in the system. In this paper, we describe HybridSal's new capability to create time-aware relational abstractions, which gives users control over the precision of the abstraction. We also describe a novel approach for abstracting nonlinear expressions that allows us to create time-aware relational abstractions that are more precise than those described previously. We show that the new approach enables automatic verification of systems that could not be verified previously.

1 Introduction

Hybrid automata is a modeling formalism in which discrete transitions and continuous evolution can be intermixed to describe fairly complex cyber-physical systems. HybridSal [15] is a tool for performing verification of hybrid automata models [8,11,9,3,2,6,14,16,5]. It is a tool built over the SAL tool [7] that can be used to model and verify discrete (finite or infinite) state transition systems. The core component of HybridSal is an *abstraction* engine that takes a hybrid model and outputs a discrete (SAL) model that is a sound abstraction of the hybrid model [13]. The abstract SAL model can be analyzed using the usual SAL model checking tools (such as the infinite bounded model checker or the k-induction prover).

In this paper, we revisit the relational abstraction technique [13] and its improvement to time-aware relational abstraction [10], which are both available in the current version of HybridSal [15]. We identify a class of problems for which these techniques yield very coarse abstractions, and present an approach to fix this shortcoming. The approach is based on creating sound approximations of nonlinear functions using sound approximations of the natural logarithm (ln) function. We present examples that can be verified using the new approach that could not be verified previously by HybridSal.

* This work was supported, in part, by the National Science Foundation under grant CNS-1423298, and the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL), under contract FA8750-12-C-0284. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the funding agencies.

2 Relational Abstraction

The HybridSal abstraction engine constructs a so-called *relational abstraction* of the system. A relational abstraction does not abstract the state space of the system, but only over-approximates the transition relation [13]. Concretely, the relational abstraction of a transition system $(S, \rightarrow)$ (with state space S and transition relation $\rightarrow$) is another transition system $(S, \rightarrow^a)$ such that $\rightarrow \subseteq \rightarrow^a$.

The semantics of a hybrid system is given as a state transition system $(S, \rightarrow_d \cup \rightarrow_c)$, where $\rightarrow_d$ are transitions that capture the "discrete" behavior of the system, and $\rightarrow_c$ are transitions that capture the "continuous" behavior. The HybridSal tool constructs relational abstraction of hybrid systems by abstracting only the relation $\rightarrow_c$; that is, the abstract system is $(S, \rightarrow_d \cup \rightarrow_c^a)$. We next briefly describe the relations $\rightarrow_c$ and $\rightarrow_c^a$.

The continuous behavior of a hybrid system is typically specified using ordinary differential equations. Consider a system of differential equations, $d\boldsymbol{x}/dt = f(x)$, whose dynamics are constrained to remain within invariant `Inv`. The concrete semantics of this continuous behavior is defined by the relation $\rightarrow_c$:

$$\begin{aligned} \boldsymbol{x_1} \rightarrow_c \boldsymbol{x_2} \text{ if } \exists F : \exists (t \geq 0) : \boldsymbol{x_1} = F(0), \boldsymbol{x_2} = F(t), dF/dt = f(F(t)), \\ \forall (0 \leq t' \leq t) : F(t') \in \texttt{Inv} \end{aligned} \tag{1}$$

Here, the function F is a solution of the differential equations [1].

It is extremely difficult to reason with the relation in Equation 1. Relational abstraction overcomes this problem by constructing an over-approximation of this relation that is much easier to analyze. Henceforth, let us assume that continuous dynamics are specified using *linear* ordinary differential equations; for simplicity, say $d\boldsymbol{x}/dt = A\boldsymbol{x}$, where A is an $n \times n$ matrix.

2.1 Time-Oblivious Relational Abstraction

A relational abstraction that does not mention the *time* variable, t, explicitly is called a *time-oblivious relational abstraction*. By default, HybridSal constructs time-oblivious abstractions [15]. If $\boldsymbol{c}^T$ is a left-eigenvector of A corresponding to eigenvalue λ, then it is easily proved that if $\boldsymbol{x} \rightarrow_c \boldsymbol{x'}$, then

$$\boldsymbol{c}^T \boldsymbol{x'} = \boldsymbol{c}^T \boldsymbol{x} \mathrm{e}^{\lambda(t'-t)} \tag{2}$$

When $\lambda > 0$, we can over-approximate the relationship between $\boldsymbol{x}$ and $\boldsymbol{x'}$ in the form of the following time-oblivious abstraction:

$$(\boldsymbol{c}^T \boldsymbol{x'} = \boldsymbol{c}^T \boldsymbol{x} = 0) \vee (\boldsymbol{c}^T \boldsymbol{x'} \geq \boldsymbol{c}^T \boldsymbol{x} > 0) \vee (\boldsymbol{c}^T \boldsymbol{x'} \leq \boldsymbol{c}^T \boldsymbol{x} < 0) \tag{3}$$

Note there is no mention of the time elapsed $(t'-t)$ in the above expression. Furthermore, all expressions above are *linear*. Each left eigenvector of A corresponding to a real eigenvalue will generate one constraint of the form in Equation 3. If the eigenvalue λ has a nonzero imaginary part, we can still get a *piecewise linear time-oblivious relational abstraction* on the real and imaginary parts of the corresponding eigenvector [15].

There are two shortcomings in the time-oblivious abstraction computed above. First, it is too coarse. It loses all time-related information. For example, if we have a 2-d system $dx/dt = -x, dy/dt = -y$, the time-oblivious abstraction forgets that the (exponential) decay rates of x and y are the same. Second, if matrix A is *defective*, that is, it has fewer eigenvectors corresponding to eigenvalue λ than the algebraic multiplicity of λ, then we do not even know how to construct a reasonably good time-oblivious abstraction.

As an example, consider the 2-d system $dx/dt = x + y, dy/dt = y$. The corresponding matrix $A = [1, 1; 0, 1]$ is defective – it has eigenvalue 1 with multiplicity 2, but there is only one associated left eigenvector $[0, 1]$. The solution of the ODE is given by the equations:

$$y(t) = y(0)\mathrm{e}^t, \quad x(t) = x(0)\mathrm{e}^t + y(0)t\mathrm{e}^t \tag{4}$$

The default time-oblivious relational abstraction constructed by HybridSal would just have inequalities (as in Equation 3) for $y(t)$, but $x(t)$ would be unconstrained. This is because HybridSal lacked a general technique for handling defective A matrices.

2.2 Time-Aware Relational Abstraction

The first shortcoming of the time-oblivious abstraction computed by HybridSal was recently recognized and it resulted in the introduction of *time-aware* relational abstractions [10].

A *time-aware* relational abstraction relates the change in the value of an expression to the time elapsed. It can be more precise than a time-oblivious abstraction. Consider the exponentially increasing/decaying expression in Equation 2 constructed from the left eigenvector $\boldsymbol{c}^T$ corresponding to a real eigenvalue λ. Taking the natural logarithm on both sides of Equation 2, we get

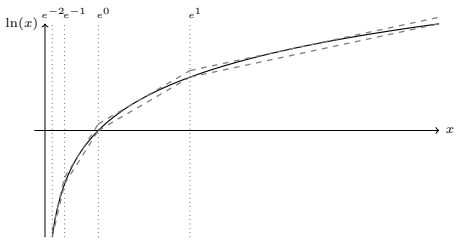
$$\ln(\boldsymbol{c}^T\boldsymbol{x'}) = \ln(\boldsymbol{c}^T\boldsymbol{x}) + \lambda(t' - t) \tag{5}$$

The expressions $\boldsymbol{c}^T\boldsymbol{x'}$, $\boldsymbol{c}^T\boldsymbol{x}$, and $\lambda(t' - t)$ (for a fixed value of λ) are *linear* in the state variables $\boldsymbol{x}$ and the time variable t (and their *next* values). So, we just need a *piecewise-affine* (lower and upper) approximation of the natural logarithm function ln to construct a *time-aware* relational abstraction. Such an approximation exists and is shown in Figure 1: intuitively, the upper bound $\ln_{\mathtt{ub}}$ is defined by first-order Taylor approximations of ln at points $e^{i+1} - e^i$ [4].

Using the piecewise-linear approximation functions $\ln_{\mathtt{lb}}$ and $\ln_{\mathtt{ub}}$ defined in Figure 1, we abstract Equation 5 using the following linear arithmetic formula:

$$\ln_{\mathtt{lb}}(\boldsymbol{c}^T\boldsymbol{x'}) \leq \ln_{\mathtt{ub}}(\boldsymbol{c}^T\boldsymbol{x}) + \lambda(t' - t) \ \wedge \ \ln_{\mathtt{ub}}(\boldsymbol{c}^T\boldsymbol{x'}) \geq \ln_{\mathtt{lb}}(\boldsymbol{c}^T\boldsymbol{x}) + \lambda(t' - t) \tag{6}$$

Since the number of intervals that define the piecewise linear approximation of the ln function is unbounded (Figure 1), HybridSal creates an approximation that uses only *finitely* many intervals: $(-\infty, \mathrm{e}^{-n}]$, $[\mathrm{e}^m, +\infty)$, and $[\mathrm{e}^i, \mathrm{e}^{i+1}]$, $i = -n, \ldots, -1, 0, 1, \ldots, m - 1$. When creating time-aware relational abstrac-



The $\ln(x)$ function in Figure 1 is approximated by dividing the x axis into the (infinitely many) intervals $[\mathrm{e}^i, \mathrm{e}^{i+1}]$, $i \in \mathbb{Z}$. In the interval $[e^i, e^{i+1}]$, a lower-bound, $\ln_{\mathtt{lb}}(x)$, for $\ln(x)$ is given by the line joining the two end-points; that is, $\ln_{\mathtt{lb}}(x) := \frac{(x-e^i)}{(e^{i+1}-e^i)} + i$; whereas an upper-bound is given by $\ln_{\mathtt{ub}}(x) := \ln_{\mathtt{lb}}(x) + 0.12$. The tangents at the end-points also provide an upper-bound; hence, a better upper-bound, $\ln^{(2)}_{\mathtt{ub}}(x)$, is $\min(\ln_{\mathtt{ub}}(x), \frac{(x-e^i)}{e^i} + i, \frac{(x-e^{i+1})}{e^{i+1}} + i + 1)$.

Fig. 1. Piecewise-linear lower and upper approximation for natural logarithm function.

tions in HybridSal, the *precision parameters* n, m are chosen by the user (via a commandline argument): picking a higher value increases precision.

Using $\ln_{\mathtt{lb}}$ and $\ln_{\mathtt{ub}}$, we know how to create time-aware abstractions for exponentially changing expressions (Equation 2), as well as, periodically changing expressions ($x(t) = \frac{x(0)}{\cos(\theta_0)}\mathrm{e}^{\lambda t}\cos(\omega t + \theta_0)$); see [10].

2.3 Defective Matrices

If the dynamics are specified by an A matrix that is not defective, then, for an n-dimensional system, we can always find n linearly independent vectors, such that for each vector $\boldsymbol{c}$, the value of the linear expression $\boldsymbol{c}^T\boldsymbol{x}$ is either exponentially changing or periodically changing. Hence, for such systems, we can get relatively good linear, time-aware relational abstractions.

If A is defective, then we can still find n linearly independent vectors, but now their dynamics can *additionally* contain terms that are "products" of exponential/trigonometric function and t^k, where t is the time variable and k is some natural number. See the dynamics of $x(t)$ in Equation 4 for an example.

The main observation we make in this paper is that we can abstract a product of two terms by using the approximations for ln function described above. In particular, the equation $z = xy\mathrm{e}^u$ can be abstracted by the expression:

$$\begin{aligned}&(z = 0 \leftrightarrow (x = 0 \vee y = 0)) \wedge (z > 0 \leftrightarrow ((x > 0 \wedge y > 0) \vee (x < 0 \wedge y < 0))) \quad \wedge \\ &\ln_{\mathtt{lb}}(|x|) + \ln_{\mathtt{lb}}(|y|) + u \leq \ln_{\mathtt{ub}}(|z|) \wedge \ln_{\mathtt{ub}}(|x|) + \ln_{\mathtt{ub}}(|y|) + u \geq \ln_{\mathtt{lb}}(|z|) \qquad (7)\end{aligned}$$

Let $\phi(u, x, y, z)$ denote the above formula. The formula ϕ contains only *linear* expressions, and hence we can use a linear theory solver to reason about it.

We can now use the abstraction ϕ to construct abstractions of linear systems whose dynamics are specified by defective matrices. Concretely, we construct a relational abstraction of the dynamics for $x(t)$ given in Equation 4, namely $x' = x\mathrm{e}^{t'-t} + y(t'-t)\mathrm{e}^{t'-t}$, as follows:

$$x' = z_1 + z_2 \wedge \phi(t' - t, x, 1, z_1) \wedge \phi(t' - t, y, t' - t, z_2) \qquad (8)$$

where z_1, z_2 are new variables. For an arbitrary A matrix, defective or not, we can now compute time-aware relational abstractions by first transforming A into Jordan normal form J; say $A = U^{-1}JU$. The value of each expression $\boldsymbol{c}^T\boldsymbol{x}$, where $\boldsymbol{c}^T$ is a row of U, is a linear combination of terms of the form $t^k \mathrm{e}^{\lambda t}$ (or $t^k \cos(bt)\mathrm{e}^{at}$ or $t^k \sin(bt)\mathrm{e}^{at}$). Thus, using ϕ, we can get piecewise linear abstractions for each expression by straight-forwardly extending the ideas used to construct the abstraction in Equation 8 (and combine the ideas with [10] for periodic dynamics).

3 Experiments

Dynamics where the A matrix is defective are quite common. Some of the simplest examples turn out to have defective matrices. For example, a linear motion with constant velocity is described by $[\dot{x}; \dot{v}] = A[x; v] + \boldsymbol{b}$, where $A = [0, 1; 0, 0]$ is defective. Similarly, linear motion with constant acceleration also gives rise to defective A matrices. Even real-world examples appear to more often have defective A matrices than not. So, it was important for us to improve the quality of abstraction HybridSal generates on these dynamics.

One good verification challenge benchmark is the adaptive cruise controller from [12]. The controller sets the acceleration of the *f*ollowing car, a_f, as $\dot{a_f} = -3a_f - 3(v_f - v_l) + gap - (v_f + 10)$, where v_l, a_l denote the velocity and acceleration of the *l*eading car, v_f, a_f are those of the *f*ollowing car, and gap is the distance between the cars. We assume that the controller is engaged whenever $gap \geq 5, 0 \leq v_l, v_f \leq 30$ and $gap - 0.1(v_f^2 - v_l^2) - 10 - (v_f - v_i) \geq 0$. We assume that a_l is an input and is constrained to be within $-5m/s^2$ and $2m/s^2$, and the velocities v_l, v_f are always non-negative. The goal is to prove that after it is engaged, the controller guarantees that $gap \geq 0$ always.

The currently released version of HybridSal, which includes an option to create time-aware relational abstraction, fails to verify the above example since it fails to use the equation $v_l' - v_l = a_l(t' - t)$ (because the A matrix is defective). However, if we add a constraint that abstracts this equation (using the ϕ formula), then HybridSal can prove safety. Even though a_l can vary arbitrarily, this equation still holds (for some a_l due to mean value theorem). Note that the quadratic term in the initial set had to be approximated by linear terms in the HybridSal model.

To further evaluate the precision of the proposed abstraction function, we also created several simple examples of linear systems whose A matrices were defective, but whose explicit solutions could be worked out by hand. For the safety property, we had an upper-bound on the value of the variable whose solution expression had the highest degree in the time variable t. Using knowledge of the explicit solution, we created initial sets and unsafe regions. We report the results in Table 1. The current version of HybridSal, of course, fails on all these examples. The new approach presented in this paper allowed us to prove conservative bounds in each case. In Table 1, Column `truebnd.` contains the true bound (computed by hand using the analytical solution), Column `proved/CE` is

name	#vars	λ	alg. mult.	#evecs	true bnd.	New Approach (default and refined) proved/CE	time	proved/CE	time
real_j2	2	-1	2	1	2.2	2.8/2.7	0.5/0.5	2.6/2.5	1.7/2.3
real_j3	3	-1	3	1	1.6	2.0/1.9	0.8/1.2	1.9/1.8	5.3/9.1
real_j4	4	-1	4	1	3.8	4.6/4.5	2.1/4.5	4.3/4.2	41/50
real_j5	5	-1	5	1	1.4	2.1/2.0	3.1/7.0	1.8/1.7	4.4/7.3
comp_j4	4	-1$\pm$i	4	2	1.8	2.8/2.7	1.8/2.8	2.7/2.6	2.3/7.6
comp_j6	6	-1$\pm$i	6	2	2.1	3.6/3.5	27/37	3.0/2.9	52/92

Table 1. Experimental results: All six examples have 1 jordan block in the A matrix. For each example, Column #vars denotes the number of state variables, λ is the eigenvalue(s), `alg.mult.` is the (sum of) algebraic multiplicity of the eigenvalue(s), #evecs is the number of eigenvectors, `truebnd.` is the true upper bound (analytically calculated) for the "top" variable, `proved/CE` is the bound proved by the tool followed by the bound that generated a (spurious) counter-example, and `time` is the time (in seconds) taken by Yices to prove/generate a counter-example. The last two columns report the same results, but using a refined upper bound for ln function.

the bound that our approach was able to prove, followed by a bound that yielded a (spurious) counter-example. Column `time` reports the time it took the SMT solver (Yices) to prove the bound in Column `proved/CE`, followed by the time it took Yices to find a counter-example for the second bound in Column `proved/CE`.

To further validate the claim that the sound approximations for ln are the key, we used a slightly better (refined) upper-bound, $\ln_{\mathtt{ub}}^{(2)}$ defined in Figure 1, for ln function and re-ran the experiments, and in each case, the tool proved a tighter safety property than before (last two columns in Table 1). As expected, using the refined approximation caused Yices to take more time. The tool and examples are all available from the HybridSal webpage [15].

We note that on examples that contain only non-defective matrices, there is no overhead added by our new extension: the piecewise-linear approximation of product terms is not triggered and the new approach becomes identical to the old [10]. We also note that for matrices that have large eigenvalues, we may need to create a more precise abstraction by choosing large values for the precision parameters. Consequently, the cost of analysis (model checking) increases for such examples. In future work, we plan to address this issue.

4 Conclusion

We presented an approach for improving the time-aware relational abstraction that is currently computed by the HybridSal tool. In particular, we improved the precision of the abstraction for linear systems whose A matrices are defective. We showed that the new approach enables HybridSal to prove correctness of systems that could not be proved correct using an approach that performed coarse abstraction for defective A matrices. This extension is significant since defective matrices occur frequently in models of real systems.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(3):3–34, 1995.
2. R. Alur, T. Dang, and F. Ivancic. Counter-example guided predicate abstraction of hybrid systems. In *TACAS*, volume 2619 of *LNCS*, pages 208–223, 2003.
3. R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions for hybrid systems. *Proc. IEEE*, 88(2):971–984, 2000.
4. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. 33rd IEEE Real-Time Systems Symposium, RTSS*, pages 183–192. IEEE Computer Society, 2012.
5. X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of the 25th Int. Conf. on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 258–263. Springer-Verlag, 2013.
6. E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *TACAS*, volume 2619 of *LNCS*, pages 192–207, 2003.
7. L. de Moura, S. Owre, H. Ruess, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.
8. G. Frehse, C. L. 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 *Proc. 23rd Intl. Conf. on Computer Aided Verification, CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.
9. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997. `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.
10. S. Mover, A. Cimatti, A. Tiwari, and S. Tonetta. Time-aware relational abstraction for hybrid systems. In *EMSOFT*, 2013.
11. A. Platzer and J.-D. Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In *Proc. Intl. Joint Conf. on Automated Reasoning, IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008.
12. A. Puri and P. Varaiya. Driving safely in smart cars. In *Proceedings of the 1995 American Control Conference*, 1995.
13. S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *Proc. CAV*, volume 6806 of *LNCS*, pages 686–702, 2011.
14. B. I. Silva and B. H. Krogh. Formal verification of hybrid system using CheckMate: A case study. In *American Control Conference*, 2000.
15. A. Tiwari. Hybridsal relational abstracter. In *Proc. CAV*, volume 7358 of *LNCS*, 2012. `http://www.csl.sri.com/~tiwari/relational-abstraction/`.
16. T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. Lamperski. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. *ACM Tans. Embedded Computing Systems (ACM TECS)*, 11, 2012.