HybridSAL Relational Abstracter

Ashish Tiwari

SRI International, Menlo Park, CA. ashish.tiwari@sri.com

Abstract. In this paper, we present the HybridSAL relational abstracter – a tool for verifying continuous and hybrid dynamical systems. The input to the tool is a model of a hybrid dynamical system and a safety property. The output of the tool is a discrete state transition system and a safety property. The correctness guarantee provided by the tool is that if the output property holds for the output discrete system, then the input property holds for the input hybrid system. The input is in HybridSal input language and the output is in SAL syntax. The SAL model can be verified using the SAL tool suite. This paper describes the HybridSAL relational abstracter – the algorithms it implements, its input, its strength and weaknesses, and its use for verification using the SAL infinite bounded model checker and k-induction prover.

1 Introduction

A dynamical system $(\mathbb{X}, \xrightarrow{a})$ with state space $\mathbb{X}$ and transition relation $\xrightarrow{a} \subseteq \mathbb{X} \times \mathbb{X}$ is a *relational abstraction* of another dynamical system $(\mathbb{X}, \xrightarrow{c})$ if the two systems have the same state space and $\xrightarrow{c} \subseteq \xrightarrow{a}$. Since a relational abstraction contains all the behaviors of the concrete system, it can be used to perform safety verification.

HybridSAL relational abstracter is a tool that computes a relational abstraction of a hybrid system as described by Sankaranarayanan and Tiwari [8]. A hybrid system $(\mathbb{X}, \rightarrow)$ is a dynamical system with
(a) state space $\mathbb{X} := \mathbb{Q} \times \mathbb{Y}$, where $\mathbb{Q}$ is a finite set and $\mathbb{Y} := \mathbb{R}^n$ is the n-dimensional real space, and
(b) transition relation $\rightarrow := \rightarrow_{cont} \cup \rightarrow_{disc}$, where $\rightarrow_{disc}$ is defined in the usual way using guards and assignments, but $\rightarrow_{cont}$ is defined by a system of *ordinary differential equation* and a *mode invariant*. One of the key steps in defining the (concrete) semantics of hybrid systems is relating a system of differential equation $\frac{d\boldsymbol{y}}{dt} = f(\boldsymbol{y})$ with mode invariant $\phi(\boldsymbol{y})$ to a binary relation over $\mathbb{R}^n$, where $\boldsymbol{y}$ is a n-dimensional vector of real-valued variables. Specifically, the semantics of such a system of differential equations is defined as:

$$\begin{aligned} \boldsymbol{y}_0 \rightarrow_{cont} \boldsymbol{y}_1 \quad & \text{if there is a } t_1 \in \mathbb{R}^{\geq 0} \text{ and a function } F \text{ from } [0, t_1] \text{ to } \mathbb{R}^n \text{ s.t.} \\ & \boldsymbol{y}_0 = F(0), \boldsymbol{y}_1 = F(t_1), \text{ and} \\ & \forall t \in [0, t_1] : \left(\frac{dF(t)}{dt} = f(F(t)) \wedge \phi(F(t)) \right) \end{aligned} \tag{1}$$

The concrete semantics is defined using the "solution" F of the system of differential equations. As a result, it is difficult to work directly with it.

The relational abstraction of a hybrid system $(\mathbb{X}, \overset{c}{\rightarrow}_{cont} \cup \overset{c}{\rightarrow}_{disc})$ is a discrete state transition system $(\mathbb{X}, \overset{a}{\rightarrow})$ such that $\overset{a}{\rightarrow} = \overset{a}{\rightarrow}_{cont} \cup \overset{c}{\rightarrow}_{disc}$, where $\overset{c}{\rightarrow}_{cont} \subseteq \overset{a}{\rightarrow}_{cont}$. In other words, the discrete transitions of the hybrid system are left untouched by the relational abstraction, and only the transitions defined by differential equations are abstracted.

The HybridSal relational abstracter tool computes such a relational abstraction for an input hybrid system. In this paper, we describe the tool, the core algorithm implemented in the tool, and we also provide some examples.

2 Relational Abstraction of Linear Systems

Given a system of linear ordinary differential equation, $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{b}$, we describe the algorithm used to compute the abstract transition relation $\overset{a}{\rightarrow}$ of the concrete transition relation $\overset{c}{\rightarrow}$ defined by the differential equations.

The algorithm is described in Figure 1. The input is a pair (A, b), where A is a $(n \times n)$ matrix of rational numbers and $\boldsymbol{b}$ is a $(n \times 1)$ vector of rational numbers. The pair represents a system of differential equations $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{b}$. The output is a formula ϕ over the variables $\boldsymbol{x}, \boldsymbol{x}'$ that represents the relational abstraction of $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{b}$. The key idea in the algorithm is to use the eigenstructure of the matrix A to generate the relational abstraction.

The following proposition states the correctness of the algorithm.

Proposition 1. *Given (A, b), let ϕ be the output of procedure* `linODEabs` *in Figure 1. If $\rightarrow_{cont}$ is the binary relation defining the semantics of $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + b$ with mode invariant True (as defined in Equation 1), then $\rightarrow_{cont} \subseteq \phi$.*

By applying the above abstraction procedure on to the dynamics of each mode of a given hybrid system, the HybridSal relational abstracter constructs a relational abstraction of a hybrid system. This abstract system is a purely discrete infinite state space system that can be analyzed using infinite bounded model checking (inf-BMC), k-induction, or abstract interpretation.

We make two important remarks here. First, the relational abstraction constructed by procedure `linODEabs` is a Boolean combination of linear *and nonlinear* expressions. The nonlinear expressions can be replaced by their conservative linear approximations. The HybridSal relational abstracter performs this approximation by default. It generates the (more precise) nonlinear abstraction (as described in Figure 1) when invoked using an appropriate command line flag. Both inf-BMC and k-induction provers rely on satisfiability modulo theory (SMT) solvers. Most SMT solvers can only reason about *linear* constraints, and hence, the ability to generate linear relational abstractions is important. However, there is significant research effort going on into extending SMT solvers to handle nonlinear expressions. HybridSal relational abstracter and SAL inf-BMC have been used to create benchmarks for linear *and nonlinear* SMT solvers.

Second, Procedure `linODEabs` can be extended to generate even more precise *nonlinear* relational abstractions of linear systems. Let $p_1, p_2, \ldots, p_k$ be k (linear and nonlinear) expressions found by Procedure `linODEabs` that satisfy

```
linODEabs(A, b): Input: a pair (A, b), where $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^{n\times 1}$.
Output: a formula $\phi$ over the variables $\boldsymbol{x}, \boldsymbol{x}'$

 1. identify all variables $x_1, \dots, x_k$ s.t. $\frac{dx_i}{dt} = b_i$ where $b_i \in \mathbb{R}\ \forall i$
    let $E$ be $\{\frac{x_i' - x_i}{b_i} \mid i = 1, \dots, k\}$
 2. partition the variables $\boldsymbol{x}$ into $\boldsymbol{y}$ and $\boldsymbol{z}$ s.t. $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{b}$ can be rewritten as
    $$\begin{bmatrix} \frac{d\boldsymbol{y}}{dt} \\ \frac{d\boldsymbol{z}}{dt} \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} + \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix}$$
    where $A_1 \in \mathbb{R}^{n_1\times n_1}$, $A_2 \in \mathbb{R}^{n_1\times n_2}$, $\boldsymbol{b_1} \in \mathbb{R}^{n_1\times 1}$, $\boldsymbol{b_2} \in \mathbb{R}^{n_2\times 1}$, and $n = n_1 + n_2$
 3. set $\phi$ to be True
 4. let $\boldsymbol{c}$ be a real left eigenvector of matrix $A_1$ and let $\lambda$ be the corresponding real
    eigenvalue, that is, $\boldsymbol{c}^T A_1 = \lambda \boldsymbol{c}^T$
 5. if $\lambda == 0 \wedge c^T A_2 == 0$: set $E := E \cup \{\frac{\boldsymbol{c}^T(\boldsymbol{y}' - \boldsymbol{y})}{\boldsymbol{c}^T \boldsymbol{b_1}}\}$; else: $E := E$
 6. if $\lambda \neq 0$: define vector $\boldsymbol{d}$ and real number $e$ as: $\boldsymbol{d}^T = \boldsymbol{c}^T A_2/\lambda$ and $e = (\boldsymbol{c}^T \boldsymbol{b_1} + \boldsymbol{d}^T \boldsymbol{b_2})/\lambda$
    let $p(\boldsymbol{x})$ denote the expression $\boldsymbol{c}^T\boldsymbol{y} + \boldsymbol{d}^T\boldsymbol{z} + e$ and let $p(\boldsymbol{x}')$ denote $\boldsymbol{c}^T\boldsymbol{y}' + \boldsymbol{d}^T\boldsymbol{z}' + e$
    if $\lambda > 0$: set $\phi := \phi \wedge [(p(\boldsymbol{x}') \leq p(\boldsymbol{x}) < 0) \vee (p(\boldsymbol{x}') \geq p(\boldsymbol{x}) > 0) \vee (p(\boldsymbol{x}') = p(\boldsymbol{x}) = 0)]$
    if $\lambda < 0$: set $\phi := \phi \wedge [(p(\boldsymbol{x}) \leq p(\boldsymbol{x}') < 0) \vee (p(\boldsymbol{x}) \geq p(\boldsymbol{x}') > 0) \vee (p(\boldsymbol{x}') = p(\boldsymbol{x}) = 0)]$
 7. if there are more than one eigenvectors corresponding to the eigenvalue $\lambda$, then
    update $\phi$ or $E$ by generalizing the above
 8. repeat Steps (4)–(7) for each pair $(\boldsymbol{c}, \lambda)$ of left eigenvalue and eigenvector of $A_1$
 9. let $\boldsymbol{c} + \imath\boldsymbol{d}$ be a complex left eigenvector of $A_1$ corresponding to eigenvalue $\alpha + \imath\beta$
10. using simple linear equation solving as above, find $\boldsymbol{c_1}$, $\boldsymbol{d_1}$, $e_1$ and $e_2$ s.t. if $p_1$
    denotes $\boldsymbol{c}^T\boldsymbol{y} + \boldsymbol{c_1}^T\boldsymbol{z} + e_1$ and if $p_2$ denotes $\boldsymbol{d}^T\boldsymbol{y} + \boldsymbol{c_2}^T\boldsymbol{z} + e_2$ then
    $$\frac{d}{dt}(p_1) = \alpha p_1 - \beta p_2 \qquad \frac{d}{dt}(p_2) = \beta p_1 + \alpha p_2$$
    let $p_1'$ and $p_2'$ denote the primed versions of $p_1, p_2$
11. if $\alpha \leq 0$: set $\phi := \phi \wedge (p_1^2 + p_2^2 \geq {p_1'}^2 + {p_2'}^2)$
    if $\alpha \geq 0$: set $\phi := \phi \wedge (p_1^2 + p_2^2 \leq {p_1'}^2 + {p_2'}^2)$
12. repeat Steps (9)–(11) for every complex eigenvalue eigenvector pair
13. set $\phi := \phi \wedge \bigwedge_{e_1, e_2 \in E} e_1 = e_2$;   return $\phi$
```

Fig. 1. Algorithm implemented in HybridSal relational abstracter for computing relational abstractions of linear ordinary differential equations.

the equation $\frac{dp_i}{dt} = \lambda_i p_i$. Suppose further that there is some λ_0 s.t. for each i $\lambda_i = n_i\lambda_0$ for some *integer* n_i. Then, we can extend ϕ by adding the following relation to it:

$$p_i(\boldsymbol{x}')^{n_j} p_j(\boldsymbol{x})^{n_i} = p_j(\boldsymbol{x}')^{n_i} p_i(\boldsymbol{x})^{n_j} \tag{2}$$

However, since p_i's are linear or quadratic expressions, the above relations will be highly nonlinear unless n_i's are small. So, they are not currently generated by the relational abstracter. It is left for future work to see if good and useful linear approximations of these highly nonlinear relations can be obtained.

3 The HybridSal Relational Abstracter

The HybridSal relational abstracter tool, including the sources, documentation and examples, is freely available for download [10].

The input to the tool is a file containing a specification of a hybrid system and safety properties. The HybridSal language naturally extends the SAL language by providing syntax for specifying ordinary differential equations. SAL is a guarded command language for specifying discrete state transition systems and supports modular specifications using synchronous and asynchronous composition operators. The reader is referred to [7] for details. HybridSal inherits all the language features of SAL. Additionally, HybridSal allows differential equations to appear in the model as follows: for each real-valued variable x, the user defines a dummy variable *xdot* which represents $\frac{dx}{dt}$. A differential equation can now be written by assigning to the *xdot* variable. Assuming two variables x, y, the syntax is as follows:

```
guard(x,y) AND guard2(x,x',y,y') --> xdot' = e1; ydot' = e2
```

This represents the system of differential equations $\frac{dx}{dt} = e1$, $\frac{dy}{dt} = e2$ with mode invariant $guard(x, y)$. The semantics of this guarded transition is the binary relation defined in Equation 1 conjuncted with the binary relation $guard2(x, x', y, y')$. The semantics of all other constructs in HybridSal match exactly the semantics of their counterparts in SAL.

Figure 2 contains sketches of two examples of hybrid systems modeled in HybridSal. The example in Figure 2(left) defines a module `SimpleHS` with two real-valued variables x, y. Its dynamics are defined by $\frac{dx}{dt} = -y + x$, $\frac{dy}{dt} = -y - x$ with mode invariant $y \geq 0$, and by a discrete transition with guard $y \leq 0$. The HybridSal file `SimpleHS.hsal` also defines two safety properties. The latter one says that x is always non-negative. This model is analyzed by abstracting it

```
bin/hsal2hasal examples/SimpleEx.hsal
```

to create a relational abstraction in a SAL file named `examples/SimpleEx.sal`, and then (bounded) model checking the SAL file

```
sal-inf-bmc -i -d 1 SimpleEx helper
sal-inf-bmc -i -d 1 -l helper SimpleEx correct
```

The above commands prove the safety property using k-induction: first we prove a lemma, named `helper`, using 1-induction and then use the lemma to prove the main theorem named `correct`.

The example in Figure 2(right) shows the sketch of a model of the train-gate-controller example in HybridSal. All continuous dynamics are moved into one module (named `timeElapse`). The `train`, `gate` and `controller` modules define the state machines and are pure SAL modules. The `observer` module is also a pure SAL module and its job is to enforce synchronization between modules on events. The final system is a complex composition of the base modules.

The above two examples, as well as, several other simple examples are provided in the HybridSal distribution to help users understand the syntax and working of the relational abstracter. A notable (nontrivial) example in the distribution is a hybrid model of an automobile's automatic transmission from [2].

```
SimpleEx: CONTEXT = BEGIN
 SimpleHS: MODULE = BEGIN
  LOCAL x,y,xdot,ydot:REAL
  INITIALIZATION
   x = 1; y IN {z:REAL| z <= 2}
  TRANSITION
   [ y >= 0 AND y' >= 0 -->
       xdot' = -y + x ;
       ydot' = -y - x
   [] y <= 0 --> x' = 1; y' = 2]
 END;
 helper: LEMMA SimpleHS |-
   G(0.9239*x >= 0.3827*y);
 correct : THEOREM
   SimpleHS |- G(x >= 0);
END
```

```
TGC: CONTEXT = BEGIN
 Mode: TYPE = {s1, s2, s3, s4};
 timeElapse: MODULE = BEGIN
  variable declarations
  INITIALIZATION x = 0; y = 0; z = 0
  TRANSITION
   [mode invariants -->
     --> xdot' = 1; ydot' = 1; zdot' = 1]
 END;
 train: MODULE = ...
 gate: MODULE = ...
 controller: MODULE = ...
 observer: MODULE = ...
 system: MODULE = (observer || (train []
   gate [] controller [] timeElapse));
 correct: THEOREM system |- G ( ... ) ;
END
```

Fig. 2. Modeling hybrid systems in HybridSal: A few examples.

Users have to separately download and install SAL model checkers if they wish to analyze the output SAL files using k-induction or infinite BMC.

The HybridSal relational abstracter constructs abstractions compositionally; i.e., it works on each mode (each system of differential equations) separately. It just performs some simple linear algebraic manipulations and is therefore very fast. The bottleneck step in our tool chain is the inf-BMC and k-induction step, which is orders of magnitude slower than the abstraction step (we have not tried abstract interpretation yet). The performance of HybridSal matches the performance reported in our earlier paper [8] on the navigation benchmarks (which are included with the HybridSal distribution). In [8] we had used many different techniques (not all completely automated at that time) to construct the relational abstraction.

4 Discussion: Strengths and Weaknesses

The HybridSal relational abstracter is a tool for verifying hybrid systems. The other common tools for hybrid system verification consist of (a) tools that iteratively compute an overapproximation of the reachable states [4], (b) tools that directly search for correctness certificates (such as inductive invariants or Lyapunov function) [6, 9], or (c) tools that compute an abstraction and then analyze the abstraction [5, 1, 3]. Our relational abstraction tool falls in category (c), but unlike all other abstraction tools, it does not abstract the state space, but abstracts only the transition relation.

The key benefit of relational abstraction is that it cleanly separates reasoning on continuous dynamics (where we use control theory or systems theory) and reasoning on discrete state transition systems (where we use formal methods.) Concepts such as Lyapunov functions or inductive invariants (aka barrier certificates) for continuous systems are used to construct very precise relational abstractions, and formal methods is used to verify the abstracted system. In fact, for several classes of simple continuous dynamical systems, lossless relational abstractions can be constructed, and hence all incompleteness in verification then comes from incompleteness of k-induction provers.

The relational abstraction methodology and tool have certain weaknesses, which we now enumerate. (a) Relational abstraction generates verification problem on a discrete, infinite state space system, which are difficult to automatically handle. (b) Our tool does not use the mode invariants when creating relational abstractions. (c) Our tool performs calculations using floating point arithmetic and hence the computed eigenvalues and eigenvectors can have numerical errors. (d) There are other techniques for discovering relational invariants that are not automated in our tool presently. (e) Our tool can not handle nonlinear differential equations presently. (f) Our tool can not efficiently handle platform constraints imposed on control systems, such as sampling frequency, sensing and actuating delays, etc [11].We note that this is the first version of the tool and we hope to enhance the tool to address some of the above concerns in future releases of the tool.

References

1. 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.
2. A. Chutinan and K. R. Butts. *SmartVehicle baseline report: Dynamic analysis of hybrid system models for design validation.* Ford Motor Co., 2002. Tech. report, Open Experimental Platform for DARPA MoBIES, Contract F33615-00-C-1698.
3. 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.
4. 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 *CAV*, volume 6806 of *LNCS*, pages 379–395, 2011.
5. Hybridsal: Modeling and abstracting hybrid systems. `http://www.csl.sri.com/users/tiwari/HybridSalDoc.ps`.
6. S. Prajna, A. Papachristodoulou, and P. A. Parrilo. SOSTOOLS: Sum of Square Optimization Toolbox for MATLAB, 2002. Available from `http://www.cds.caltech.edu/sostools` and `http://www.aut.ee.ethz.ch/~parrilo/sostools`.
7. The SAL intermediate language, 2003. Computer Science Laboratory, SRI International, Menlo Park, CA. `http://sal.csl.sri.com/`.
8. S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 686–702, 2011.
9. T. Sturm and A. Tiwari. Verification and synthesis using real quantifier elimination. In *ISSAC*, pages 329–336, 2011.
10. A. Tiwari. Hybridsal relational abstracter. `http://www.csl.sri.com/~tiwari/relational-abstraction/`.
11. A. Zutshi, S. Sankaranarayanan, and A. Tiwari. Timed relational abstractions for sampled data control systems. Submitted, Under review.

A Supplementary Material

Proof. (Proof sketch for Proposition 1) First, let $p(\boldsymbol{x})$ be the linear expression $\boldsymbol{c}^T\boldsymbol{y} + \boldsymbol{d}^T\boldsymbol{z} + e$ discovered in Step (6). Then,

$$\begin{aligned}\frac{dp}{dt} &= \boldsymbol{c}^T(A_1\boldsymbol{y} + A_2\boldsymbol{z} + \boldsymbol{b_1}) + \boldsymbol{d}^T\boldsymbol{b_2} \;=\; \lambda\boldsymbol{c}^T\boldsymbol{y} + \lambda\boldsymbol{d}^T\boldsymbol{z} + \boldsymbol{c}^T\boldsymbol{b_1} + \boldsymbol{d}^T\boldsymbol{b_2}\\ &= \lambda * (\boldsymbol{c}^T\boldsymbol{y} + \boldsymbol{d}^T\boldsymbol{z} + \boldsymbol{c}) \;=\; \lambda * p\end{aligned}$$

Hence, $p(\boldsymbol{x}(t)) = p(\boldsymbol{x}(0))e^{\lambda t}$. Therefore, the relation added in Step (6) to ϕ will hold between an initial state $\boldsymbol{x}$ and a future state $\boldsymbol{x'}$.

Next, consider the quadratic relations added to ϕ in Step (11). Let p_1, p_2 be as defined in Step (10). Then,

$$\frac{d(p_1^2 + p_2^2)}{dt} = 2p_1(\alpha p_1 - \beta p_2) + 2p_2(\beta p_1 + \alpha p_2) \;=\; 2\alpha(p_1^2 + p_2^2)$$

Hence, $p_1(\boldsymbol{x}(t))^2 + p_2(\boldsymbol{x}(t))^2 = (p_1(\boldsymbol{x}(0))^2 + p_2(\boldsymbol{x}(0))^2)e^{2\alpha t}$, and therefore, the relation added in Step (11) to ϕ will hold between an initial state $\boldsymbol{x}$ and a future state $\boldsymbol{x'}$.

Next, consider the relations added in Step (13). It is easy to observe that every expression $s(\boldsymbol{x}, \boldsymbol{x'})$ in the set E is equal to the time t taken to reach $\boldsymbol{x}'$ from $\boldsymbol{x}$ following the linear ODE dynamics. Hence, all these expressions need to be equal, as stated in Step (13).

Finally, the nonlinear relationship in Equation 2 holds for any binary reachable pair of states $(\boldsymbol{x}, \boldsymbol{x'})$ because

$$\left(\frac{p_i(\boldsymbol{x'})}{p_i(\boldsymbol{x})}\right)^{n_j} = \left(\frac{p_j(\boldsymbol{x'})}{p_j(\boldsymbol{x})}\right)^{n_i} = e^{n_i n_j \lambda_0 t}$$

□