HybridSAL Relational Abstracter

Ashish Tiwari*

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

Abstract. This paper describes 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 $(\mathbb{X}, \stackrel{a}{\rightarrow})$ with state space $\mathbb{X}$ and transition relation $\stackrel{a}{\rightarrow} \subseteq \mathbb{X}\times\mathbb{X}$ is a *relational abstraction* of another dynamical system $(\mathbb{X}, \stackrel{c}{\rightarrow})$ if the two systems have the same state space and $\stackrel{c}{\rightarrow} \subseteq \stackrel{a}{\rightarrow}$. 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:

$$\boldsymbol{y}_0 \rightarrow_{cont} \boldsymbol{y}_1 \text{ if there is a } t_1 \in \mathbb{R}^{\geq 0} \text{ and a function } F \text{ from } [0, t_1] \text{ to } \mathbb{R}^n \text{ s.t.}$$

* Supported in part by DARPA under subcontract No. VA-DSR 21806-S4 under prime contract No. FA8650-10-C-7075, and NSF grants CSR-0917398 and SHF:CSR-1017483.

P. Madhusudan and S.A. Seshia (Eds.): CAV 2012, LNCS 7358, pp. 725–731, 2012.

$$\boldsymbol{y}_0 = F(0), \boldsymbol{y}_1 = F(t_1), \text{ and}$$
$$\forall t \in [0, t_1] : \left(\frac{dF(t)}{dt} = f(F(t)) \wedge \phi(F(t)) \right) \qquad (1)$$

The concrete semantics is defined using the "solution" F of the system of differential equations. As a result, it is difficult to directly work 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 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. By default, HybridSal generates conservative linear approximations of these nonlinear relational invariants. HybridSal generates the (more precise) nonlinear abstraction (as described in Figure 1) when invoked using an appropriate command line flag. Note that most inf-BMC tools can only handle linear constraints. 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.

`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 ϕ over the variables $\boldsymbol{x}, \boldsymbol{x}'$

1. identify all variables $x_1, \ldots, x_k$ s.t. $\frac{dx_i}{dt} = b_i$ where $b_i \in \mathbb{R}\ \ \forall i$
   let E be $\{\frac{x_i'-x_i}{b_i} \mid i = 1, \ldots, k\}$
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 ϕ to be *True*
4. let $\boldsymbol{c}$ be a real left eigenvector of matrix A_1 and let λ 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 λ, then update ϕ 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 ϕ

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

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 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: if x is a real-valued variable, a differential equation $\frac{dx}{dt} = e$ can be written by assigning e to the dummy identifier *xdot*. 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 `SimpleEx.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

```
SimpleEx: CONTEXT = BEGIN
 SimpleHS: MODULE = BEGIN
  LOCAL x,y: 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

of the relational abstracter. A notable (nontrivial) example in the distribution is a hybrid model of an automobile's automatic transmission from [2]. 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 (Table 1).

4 Related Work and Conclusion

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 [5], (b) tools that directly search for correctness certificates (such as inductive invariants or Lyapunov function) [9], or (c) tools that compute an abstraction and then analyze the abstraction [6,1,3]. 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. In [8] we had defined relational abstractions and proposed many different techniques (not all completely automated at that time) to construct the relational abstraction.

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.) The former is used for constructing high quality relational abstractions and the latter is used for verifying the abstract system.

Table 1. Performance on the 27 navigation benchmarks [4]: The HybridSal models, on purpose, enumerate all modes explicitly so that it becomes clear that the time (RA) for constructing relational abstraction grows linearly with the number of modes (modes). Inf-bmc starts to time out (TO) at 5 minutes at depth (d) 20 for examples with ≥ 25 modes. Ideally, one wants to perform inf-bmc with depth equal to number of modes. $N100$ means inf-bmc returned after 100 seconds with no counter-examples and $C160$ means inf-bmc returned after 160 seconds with a counter-example.

nav	1-5	6	7-8	9	10-11	12	13-15	16-18	19-21	22-24	25-27
modes	9	9	16	16	25	25	42	81	144	225	400
RA	2	2	3	3	5	5	9	20	40	80	180
d=4	N0	N0	N1	N1	N1	C1	N1	N2	N4	N6	N20
d=8	N1	C2	C100	C5	C10	C15	N20	N10	N25	N10	N60
d=12	N5	C3	TO	C18	C20	C50	C150	N10	TO	N40	T0
d=16	N40	C10	TO	C50	C50	C180	TO*	240*	TO	TO	TO
d=20	N100	C80	TO	C160	C80	TO	TO	TO	TO	TO	TO

We note that our tool is the first relational abstracter for hybrid systems and is under active development. We hope to enhance the tool by improving precision of the abstraction using mode invariants and other techniques, providing alternative to inf-bmc, and handling nonlinear differential equations.

References

1. Alur, R., Dang, T., Ivančić, F.: Counter-Example Guided Predicate Abstraction of Hybrid Systems. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 208–223. Springer, Heidelberg (2003)
2. Chutinan, A., Butts, K.R.: SmartVehicle baseline report: Dynamic analysis of hybrid system models for design validation. Ford Motor Co., Tech. report, Open Experimental Platform for DARPA MoBIES, Contract F33615-00-C-1698 (2002)
3. Clarke, E., Fehnker, A., Han, Z., Krogh, B.H., Stursberg, O., Theobald, M.: Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 192–207. Springer, Heidelberg (2003)
4. Fehnker, A., Ivančić, F.: Benchmarks for Hybrid Systems Verification. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004)
5. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable Verification of Hybrid Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
6. Hybridsal: Modeling and abstracting hybrid systems, `http://www.csl.sri.com/users/tiwari/HybridSalDoc.ps`
7. The SAL intermediate language, Computer Science Laboratory, SRI International, Menlo Park, CA (2003), `http://sal.csl.sri.com/`

8. Sankaranarayanan, S., Tiwari, A.: Relational Abstractions for Continuous and Hybrid Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 686–702. Springer, Heidelberg (2011)
9. Sturm, T., Tiwari, A.: Verification and synthesis using real quantifier elimination. In: ISSAC 2011, pp. 329–336 (2011)
10. Tiwari, A.: Hybridsal relational abstracter, http://www.csl.sri.com/~tiwari/relational-abstraction/