HybridSAL Relational Abstractor: A New Abstractor for Hybrid Systems

Ashish Tiwari

September 7, 2011

1 Detailed Example and the HybridSal Relational Abstractor Tool

Consider a simple 2-dimensional continuous system defined by

$$\begin{aligned} \frac{dx}{dt} &= -y + x \\ \frac{dy}{dt} &= -y - x \end{aligned}$$

Assume we are given the initial condition $x = 1, y = 2$ and the invariant that y is always non-negative. We wish to prove that x always remains non-negative.

This example can be encoded in HybridSAL as shown in Figure 1. The HybridSAL syntax is almost identical to the syntax of SAL [2], but for a few modifications that enable encoding of continuous dynamical systems. The key changes in HybridSAL are:

- Variables whose name ends in *dot* denote the derivative. In Figure 1, there are two state variables x, y, and their derivatives are denoted by variables named $xdot, ydot$ respectively. These special dot variables can only be used as left-hand sides of simple definitions (equations) that appear in guarded commands. Thus, the equation $xdot' = -y + x$ denotes the differential equation $dx/dt = -y + x$.
- The guard of the guarded command that encodes the system of differential equation denotes the state invariant; that is, the system is forced to remain inside the invariant set while evolving as per the differential equations. This meaning is consistent with the usual semantics of guards in SAL. In Figure 1, the guard $y \geq 0 \wedge y' \geq 0$ says that $y \geq 0$ is the mode invariant for the only mode in the system.

The definition of contexts, modules, and properties (theorems) are exactly as in the SAL language [2].

```
Linear1: CONTEXT =
BEGIN

control: MODULE =
BEGIN
 LOCAL x,y:REAL
 LOCAL xdot,ydot:REAL
 INITIALIZATION
  x = 1; y = 2
 TRANSITION
 [
 y >= 0 AND y' >= 0 -->
  xdot' = -y + x ;
  ydot' = -y - x
 ]
END;

% proved using sal-inf-bmc -i -d 2 Linear1 helper
helper: LEMMA
 control |- G(0.9239 * x >= 0.3827 * y);

% proved using sal-inf-bmc -i -d 2 -l helper Linear1 correct
correct : THEOREM
 control |- G(x >= 0);
END
```

Figure 1: HybridSAL file describing a simple two-dimensional continuous dynamical system, along with two safety properties of that system.

The user creates a HybridSAL model, similar to the one shown in Figure 1, using a text editor. Following the SAL convention for naming files, the HybridSAL file containing the model in Figure 1 is stored as file `Linear1.hsal`. Thereafter, to prove the two properties contained in `Linear1.hsal`, the user executes the following commands.

`bin/hsal2hasal examples/Linear1.hsal` . This command is run from the root directory of the HybridSAL relational abstraction tool. It will create a set of files in the subdirectory `examples/`, including the file `Linear1.sal` that contains the relational abstraction of the original model. The process of going from `Linear1.hsal` to `Linear1.sal` involves the following stages (that are all performed in a single run of the above command):

$$\texttt{Linear1.hsal} \longrightarrow \texttt{Linear1.hxml} \longrightarrow \texttt{Linear1.haxml} \longrightarrow \texttt{Linear1.xml} \longrightarrow \texttt{Linear1.sal}$$

The hsal2hxml converter is in the subdirectory `hybridsal2xml`. It is simply the HybridSAL parser that parses a files and outputs it in `.hxml`

format. The program `bin/hsal2hasal` can take as input either a `.hsal` file or a `.hxml` file. It creates `.hasal` and `.haxml` – which is a file in *extended HybridSAL* syntax – that contains the original model as well as its relational abstraction. It is an intermediate file that is useful only for debugging purposes at the moment. In the last stage, the final `.xml` and `.sal` files are easily extracted from the `.haxml` files.

`sal-inf-bmc -i -d 2 Linear1 correct` . Once the relational abstraction has been created, it can be model checked. Note that the relational abstraction (in file `Linear1.sal`) is an *infinite* state system. Hence, we can not use finite state model checkers. We can, however, use the SAL infinite bounded model checker (`sal-inf-bmc`) and the k-induction prover (`sal-inf-bmc -i`). The k-induction prover can sometimes fail to prove a correct assertion because the assertion is not inductive. In such a case, auxiliary lemmas may be needed to complete a proof. In the running example, we need a helper lemma. Using the lemma `helper`, the property `correct` can be proved using the command:
`sal-inf-bmc -i -d 2 -l helper Linear1 correct`
The lemma `helper` can itself be proved using k-induction as:
`sal-inf-bmc -i -d 2 Linear1 helper`

This completes the discussion of the application of the HybridSAL relational abstraction tool on the running example. For more complex examples, including examples of hybrid systems, the reader is referred to the `examples/` subdirectory in the tool. We note a few points here.

Initialization The initial state of the system need not be a single point. It can be a region of the state space. For example, in Figure 1, the initialization section can be replaced by the initialization

```
INITIALIZATION
 x IN {z:REAL|0 <= z AND z <= 1};
 y IN {z:REAL|2 <= z AND z <= 3};
```

The new model can again be verified using the same set of commands given above.

Composition The model need not be a single module, and it can be a composition of modules. Modules can be purely discrete – they need not all have dynamics given by differential equations. For example, the example in the File `TGC.hsal` describes a model of the train-gate-controller in HybridSAL that is a composition of five modules. Only one of the five has continuous differential equations in the dynamics.

Apart from the syntax for writing differential equations, the HybridSAL input language supports two additional features that are not part of the SAL language [2]. These features are:

Invariant Apart from `INITIALIZATION` and `TRANSITION` blocks, each basemodule can also have an `INVARIANT` block. The invariant block contains a formula that is an (assumed) invariant of the system.

In our running example, we can add the Invariant block:

```
INVARIANT y >= 0
```

in the basemodule, for example, just before/after the `INITIALIZATION` block. Then, we could replace the guard $y \geq 0 \wedge y' \geq 0$ by the new guard *True* in the (only) transition. If ϕ is declared as the invariant, then it has the effect of adding the formula $\phi \wedge \phi'$ in the guards of *all* transitions. This is performed as a preprocessing step by the HSal Relational Abstractor.

INITFORMULA Instead of `INITIALIZATION` block, a HybridSAL input file can contain a `INITFORMULA` block that has a formula as the initialization predicate.

In our running example, the initialization block shown above can be replaced by

```
INITFORMULA 0 <= x AND x <= 1 AND 2 <= y AND y <= 3
```

without changing the meaning of the HybridSAL model.

The `INITFORMULA` block is also handled during the preprocessing phase. The preprocessing phase also handles *defined constants*.

2 HSal Relational Abstractor: Flags

The HybridSAL Relational Abstractor tool accepts the following options/flags.

-n, –nonlinear With this option, the tool creates a more precise relational abstraction, but it may be nonlinear. Even when the input model is a linear continuous dynamical system, the output could be a nonlinear discrete time system.

Currently, the SAL model checker can not analyze nonlinear discrete time systems. Hence, this flag is useful only if using other backend tools that can handle discrete time nonlinear systems.

By default, this option is *not* turned on, and the tool creates linear relational abstractions that can be analyzed by SAL infinite bounded model checker.

-c, –copyguard With this option, the tool explicitly handles the guard in the continuous dynamics as state invariants. Recall the HybridSAL code for the differential equations $dx/dt = -y + x, dy/dt = -y - x$.

```
[
y >= 0 AND y' >= 0 -->
 xdot' = -y + x ;
 ydot' = -y - x
]
```

Here the guard $y \geq 0 \wedge y' \geq 0$ says that y should be nonnegative both *before* and *after* the transition. In other words, $y \geq 0$ is the mode invariant. We could have written the same dynamics as follows:

```
[
y >= 0 -->
 xdot' = -y + x ;
 ydot' = -y - x
]
```

The new HybridSAL file, when processed with the flag `-c`, produces the same output as the original file would produce without the `-c` flag. Thus, the `-c` flag causes copying of the guard of the continuous transitions, but with variables replaced by their prime forms.

By default, this option is *not* turned on, and the tool assumes that the input HybridSAL file already contains guards on prime variables.

-o, –opt This flag turns on some optimizations in the relational abstractor. Currently, this flag causes the tool to construct a relational abstraction that assumes that a certain amount of time is always spent in each mode. This is an unsound assumption. Hence, the output of the relational abstractor can be unsound when the `-o` flag is used.

The `-o` flag is useful if the (sound) relational abstraction is too large to be analyzable (in reasonable time) by the model checker. In that case, the user could try to create a simpler, but unsound, abstraction using the `-o` flag and model checking it.

By default, this option is *not* turned on.

The tool is expected to include more options in the future as new extensions and features are implemented.

3 HSal Relational Abstractor: Background

Hybrid dynamical systems are formal models of complex systems that have both discrete and continuous behavior. It is well-known that the problem of verifying hybrid systems for properties such as safety and stability is quite hard, both

in theory and in practice. There are no automated, scalable and compositional tools and techniques for formal verification of hybrid systems.

HybridSAL is a framework for modeling and analyzing hybrid systems. HybridSAL is built as an extension of SAL (Symbolic Analysis Laboratory). SAL consists of a language and a suite of tools for modeling and analyzing discrete state transition systems. HybridSAL extends SAL by allowing specification of continuous dynamics in the form of differential equations. Thus, HybridSAL can be used to model hybrid systems. These models can be abstracted into discrete finite state transition systems using the HybridSAL abstractor. The abstracted system is output in the SAL language, and hence SAL (symbolic) model checkers can be used to model check the abstraction. While HybridSAL can be used to verify intricate hybrid system models, it is based on constructing qualitative abstractions – which can be very coarse at times. Furthermore, the HybridSAL abstractor is not compositional, and can not abstract modules independently separately without being very coarse.

To alleviate the shortcomings of the HybridSAL abstractor, we developed the concept of relational abstractions of hybrid systems. A relational abstraction transforms a given hybrid system into a purely discrete transition system by summarizing the effect of the continuous evolution using relations. The state space of system and its discrete transitions are left unchanged. However, the differential equations describing the continuous dynamics (in each mode) are replaced by a relation between the initial values of the variables and final values of the variables. The abstract discrete system is an infinite-state system that can be analyzed using standard techniques for verifying systems such as k-induction and bounded model checking.

Relational abstractions can be constructed compositionally by abstracting each mode separately. Abstraction and compositionality are crucial for achieving scalability of verification. We have also developed techniques for constructing good quality relational abstractions. The details are technical and can be found in papers [3, 4]. The HybridSAL verification framework has been extended by an implementation of relational abstraction.

3.1 Qualitative versus Relational Abstraction

Qualitative and predicate abstraction are techniques for abstracting a system that work by simplifying the state space of the system. Specifically they reduce the state space of the system to a finite set of (qualitative) states defined by certain (qualitative) predicates. In contrast, relational abstraction does not simplify the state space but only simplifies the presentation of the dynamics by replacing hard-to-analyze differential equations by discrete transitions. In principle, predicate and qualitative abstraction can be used on a relational abstraction of a system to further approximate the system, if needed. The original HybridSAL tool implements qualitative abstraction. The new HybridSAL Relational Abstractor implements relational abstraction.

4 HSal Relational Abstractor: Technical Background

We extended the HybridSAL tool by implementing a relational abstractor for HybridSAL models. The input to the tool is a specification of a hybrid system in the HybridSAL language.

The **verification workflow** for using the HybridSAL relatioal abstractor is as follows:

1. Create a model of a hybrid system in HybridSAL: The user creates such a model in a file, say in file filename.hsal, using any text editor.

2. Construct a relational abstraction: The user runs the HybridSAL relational abstractor to create a discrete, but infinite-state relational abstraction of the original model. The relational abstraction is output in the SAL language as filename.sal.

3. Analyze the SAL model: The user runs the SAL infinite bounded model checker or k-induction prover to analyze the SAL model in filename.sal

We now discuss the implementation of the HybridSAL relational abstractor. We first define relational abstraction of a continuous dynamical system.

Definition 1 (Relational Abstraction Without Time) *Consider a continuous dynamical system* $\frac{d\vec{x}}{dt} = f(\vec{x})$*, where* $\vec{x}$ *is a* $n \times 1$ *vector of real-valued variables. The relation* $R \subseteq \mathbb{R}^{2n}$ *is a relational abstraction of this continuous system if for all time trajectories* $\tau : [0, T) \mapsto \mathbb{R}^n$ *of this system, it is the case that* $(\forall\ t \in [0, T))\ (\tau(0), \tau(t)) \in R$*.*

Thus, a relational abstraction R captures all pairs of states $(\vec{x}_0, \vec{x})$ such that it is possible to reach $\vec{x}$ from $\vec{x}_)$ in a finite amount of time by evolving according to the continuous dynamics. A relational abstraction of a hybrid system is constructed by replacing each constituent continuous system by its relational abstraction and keeping the discrete transitions unchanged.

We briefly describe the techniques implemented in the HybridSAL relational abstractor. Currently, HybridSAL relational abstractor can only construct relational abstractions of hybrid systems in which all continuous dynamics are given by linear ordinary differential equations. For linear systems, such as $d\vec{x}/dt = A\vec{x}$, whenever A has real eigenvalues, useful relational abstractions can be generated using the eigenvectors of A' corresponding to those real eigenvalues [4]. Here, A' denotes the transpose of matrix A. Specifically, if $\vec{c}$ is such that $A'\vec{c} = \lambda\vec{c}$, then by simple algebraic manipulation, we obtain $\frac{d}{dt}(c_1x_1 + \ldots + c_nx_n) = \lambda(c_1x_1 + \ldots + c_nx_n)$ where $\vec{c} := [c_1; \ldots; c_n]$ and $\vec{x} := [x_1; \ldots; x_n]$. Let p denote the linear expression $c_1x_1 + \ldots + c_nx_n$ and let p_0 denote the linear expression $c_1x_{10} + \ldots + c_nx_{n0}$. Here, x_{i0} denotes the old value of x_i. If $\lambda < 0$, then we know that the value of p approaches zero monotonically. Consequently, we get the relational abstraction

Benchmark	Affine Invs			Affine+Eigen Invs			Affine+Eigen+Box Invs		
	depth	status	time(s)	depth	status	time(s)	depth	status	time(s)
nav01	4	F	0.63	4	F	0.88	4	F	1.91
nav01	5	P	0.75	5	P	0.91	5	P	1.36
nav02	4	F	0.64	4	F	0.87	4	F	1.8
nav02	5	P	0.68	5	P	1.04	5	P	3.33
nav03	4	F	0.60	4	F	0.91	4	F	1.72
nav03	5	P	0.67	5	P	1.05	5	P	2.7
nav04	3	CE	0.49	8	F	3.21	8	F	34.883
nav04				4	P	0.75+0.99	4	P	0.98+2.21
nav05	2	CE	0.47	8	F	3.85	8	F	37.31
nav05				8	P	2.15+2.50	8	P	5.38+11.05
nav06	4	CE	0.61	8	F	18.01	8	F	494.5
nav06*	4	CE	1.03	8	P	21.80+7.42	8	P	40.22+35.08
nav07	5	CE	0.66	-	-	-	5	F	69.9
nav07				-	-	-	6	P	6.25
nav08	4	CE	0.52	-	-	-	6	CE	0.95
nav09	4	CE	0.57	4	CE	1.45	4	CE	19.87
nav10	3	CE	0.44	3	CE	0.99	3	CE	0.95

Table 1: Comparison of various abstractions over the NAV benchmarks. All experiments were performed on an Intel Xeon E5630 2.53GHz single-core processor (x86_64 arch) with 4GB RAM running Ubuntu Linux 2.6.32-26. Legend — **depth:** k-induction depth, **time:** time taken by verifier, **status: P**: Proved Property, **CE:** k-induction base case fails and counterexample is produced, **F**: inductive step fails, no proofs or counterexample. **Note:** Relational eigeninvariants are inapplicable for nav07, nav08 (indicated by -). k-induction timings reported as $t_1 + t_2$ indicate that an auxiliary lemma was used. t_1 is the time to prove the property, and t_2 to discharge the lemma.

$(p_0 \leq p < 0) \vee (p_0 \geq p > 0) \vee (p = p_0 = 0)$. Similarly, we can write the relational invariants for the case when $\lambda > 0$ and $\lambda = 0$. This key idea also generalizes to the case when $\frac{d\vec{x}}{dt} = A\vec{x} + \vec{b}$. We call such relational abstractions eigen-invariants.

When A has complex eigenvalues, we get other kinds of relational abstractions; for details, the reader is referred to [4].

We evaluate the HybridSAL relational abstractor over the navigation benchmarks [1]. The navigation benchmarks model a vehicle moving in a 2-dimensional rectangular space $[0, m-1] \times [0, n-1]$. This space is partitioned in $m \times n$ cells. Let x, y denote the position of the vehicle and v_x, v_y denote its velocity. Then the dynamics of the vehicle in any particular cell are given by the ODEs:

$$\begin{array}{rclcrcl} \frac{dx}{dt} & = & v_x & & \frac{dv_x}{dt} & = & a_{11}(v_x - b) + a_{12}(v_y - c) \\ \frac{dy}{dt} & = & v_y & & \frac{dv_y}{dt} & = & a_{21}(v_x - b) + a_{22}(v_y - c) \end{array}$$

where the matrix $A := [a_{11}, a_{12}; a_{21}, a_{22}]$ and the direction (b, c) are parameters that can potentially vary (for each of the cells)[1].

[1]The matrix A is Hurwitz: the dynamics for (v_x, v_y) asymptotically converge to (b, c).

Every benchmark in the suite is specified by fixing the matrix A, the number of cells $m \times n$, the direction (b, c) in each cell, and initial intervals for each of the four state variables x, y, v_x, v_y. Our experiments focus on proving the unreachability of a distinct cell marked B for each benchmark instance [1].

In our experiments, we verify the safety property for the navigation benchmarks using k-induction over the relational abstraction. We use the SAL infinite bounded model checker, with the k-induction flag turned on (`sal-inf-bmc -i`), which uses the SMT solver Yices in the back end. Table 1 reports the results. For each benchmark, we report the depth used for performing k-induction (under "depth"), the output of k-induction (under "status"), and the time it took (under "time"). There are three possible outputs: (a) the base case of k-induction fails and a counterexample is found (denoted by "CE"), (b) the base case is proved, but the induction step fails; i.e., no counterexample is found, but no proof is found either (denoted by "F"), (c) the base case and the induction step are successfully proved (denoted by "P"). Since we perform k-induction on an abstraction, the counterexamples may be spurious, but the proofs are not. As Table 1 indicates, relational abstractions are sufficient to establish safety of the benchmarks nav01–nav05, nav06*, and nav07. The system nav06* is the same as nav06 but with a slightly smaller set of initial states. However, the proof fails on nav06 and nav08–nav10. There are two reasons for failure: (a) poor quality of abstraction, which is reflected in entries "CE" in Table 1, and (b) inability to find suitable k-inductive lemmas. This happens in the case of nav06, where the proof fails without yielding a counterexample. We employed three kinds of relational abstractions for each mode: affine, eigen, and box. Table 1 also shows performance of each of these techniques.

For further details on relational abstraction, and for downloading the HybridSAL relational abstractor and documentation, the reader is referred to the website `http://www.csl.sri.com/~tiwari/relational-abstraction/`.

5 Conclusion

Compositional verification of complex systems is a challenging problem. We developed a new approach that enables compositional analysis of continuous and hybrid dynamical systems. It is based on computing relational abstractions of continuous dynamics of a complex system.

Relational abstraction replaces the differential equations in the system description by sound abstract discrete transitions, thus enabling application of discrete verification tools. The HybridSAL relational abstractor automatically computes such abstractions for linear hybrid systems. The extension to nonlinear dynamics is left for future work.

References

[1] Rajeev Alur and George J. Pappas, editors. *Benchmarks for Hybrid Systems Verification*, volume 2993 of *Lecture Notes in Computer Science*. Springer, 2004.

[2] SRI Intl. Formal Methods Group. SAL: Symbolic Analysis Laboratory, 2011. `http://sal.csl.sri.com/`.

[3] Sriram Sankaranarayanan and Ashish Tiwari. Relational abstractions for continuous and hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proc. 23rd Intl. Conf. on Computer Aided Verification, CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 686–702. Springer, 2011.

[4] A. Tiwari. Approximate reachability for linear systems. In *Proc. 6th Intl. Workshop on Hybrid Systems: Computation and Control, HSCC 2003*, volume 2623 of *Lecture Notes in Computer Science*, pages 514–525. Springer, 2003.