AVERIST: An Algorithmic Verifier for Stability

Pavithra Prabhakar [1]

IMDEA Software Institute, Madrid, Spain

Miriam García Soto [2]

IMDEA Software Institute, Madrid, Spain & Universidad Politécnica de Madrid

Abstract

In this paper, we present the tool `AVERIST`, which implements an algorithmic approach for the stability analysis of polyhedral switched systems. It implements an abstraction based model-checking approach proposed in the earlier work of the authors. The tool constructs a weighted graph abstracting the polyhedral hybrid system, and analyzes the graph to infer stability. In the case that the hybrid system is unstable, it also returns an abstract counterexample in the weighted graph indicating a potential reason for failure.

Keywords: Stability analysis, hybrid systems, polyhedral switched systems, abstraction, model-checking.

1 Overview

Stability is a fundamental property in control system design. It captures the notion that small perturbations in the initial state of the system result in only small perturbations in the eventual behavior of the system. We consider stability with respect to an equilibrium point — a state of the system which does not change with time. Two classical notions of stability in control theory are Lyapunov stability and asymptotic stability. Lyapunov stability captures the notion that executions starting close to the equilibrium point remain close to it; and asymptotic stability, in addition, requires that the execution starting in a small neighborhood of the equilibrium point converge to the equilibrium point.

`AVERIST` implements an abstraction based analysis technique for stability analysis of polyhedral switched systems (*PSS*s) [8,9]. *PSS*s are hybrid systems [5] in which the invariants and guards are polyhedral sets; and the dynamics is given by

[1] Email: pavithra.prabhakar@imdea.org
[2] Email: miriam.garcia@imdea.org

a polyhedral differential inclusion $\dot{x} \in P$, where P is a polyhedral set. In addition, there are no resets of variables during a discrete transition. Polyhedral hybrid systems are an important class of hybrid systems, since hybrid systems with more complex dynamics can be reduced to this class using hybridization techniques [10,7].

`AVERIST` implements a "quantitative" predicate abstraction for stability analysis. It constructs an abstract weighted graph from a *PSS* using a set of predicates. The nodes of the graph correspond to the boundaries "faces" of the regions created by the predicates; an edge between two nodes indicates the presence of an execution from the face corresponding to the first node to that of the second; and the weight on an edge provides an upper bound on the "scaling" — the ratio of the distance to the origin in the target face with respect to the distance to the origin in the source face — associated with any execution corresponding to the edge. The weighted graph is then analysed for cycles such that the product of the weights on its edges is > 1. Absence of such cycles and absence of infinite weight edges implies Lyapunov stability; in addition, if there are no cycles with weight ≥ 1, then the system is asymptotically stable. If the graph does not satisfy the above property, then our tool returns such a cycle as an abstract counterexample, indicating a potential reason for instability. In contrast to the classical methods for stability analysis based on exhibiting Lyapunov functions, which are deductive, `AVERIST` implements an algorithmic approach.

(a) Polyhedral switched system

(b) Abstract weighted graph

Fig. 1. Illustration of quantitative predicate abstraction.

The weighted graph construction is illustrated in Figure 1. Figure 1a shows a *PSS* consisting of four modes whose invariants correspond to the four quadrants of the plane, respectively. In each of the quadrants, the system evolution is such that its derivative is a vector between the pair of vectors depicted on the quadrant, that is, the dynamics is given by the polyhedral inclusion dynamics $\dot{x} \in P$, where P is the polyhedral set corresponding to the convex cone of the two vectors. The system takes a discrete transition at the boundary between the two adjacent quadrants. A sample execution is shown in Figure 1a as a dotted piecewise linear trajectory. The weighted graph constructed from that *PSS* is shown in Figure 1b, where the nodes correspond to the positive x-axis (x^+), negative x-axis (x^-), positive y-axis (y^+) and negative y-axis (y^-). An edge between two nodes indicates the existence of an execution between the axes corresponding to the nodes. For instance, an edge from x^+ to y^+ indicates that there is an execution from the positive x-axis through *quad*1 to the positive y-axis. Every edge is labelled with a weight which indicates the maximum scaling. For instance, in the first quadrant, any execution starting

on the positive x-axis at distance d from the origin will reach the positive y-axis at distance at most $2d$ from the origin. Hence, the weight on the corresponding edge is 2.

2 Software architecture and implementation

The software architecture of AVERIST is illustrated in Figure 2. Below we explain some of the elements:

- The input to the tool is a polyhedral switched system (*PSS*) that we want to analyse for stability. In addition, one can provide a set of linear expressions, which will be used to partition the state-space.
- The main internal functions of AVERIST are as follows:
  - *State-space partition* is the first step in the quantitative predicate abstraction. It takes a finite set of linear expressions and creates a finite partition of the state-space into regions. The nodes of the abstract graph will correspond to the facets of these regions.
  - *Graph construction* is the main step of the algorithm which constructs the weighted graph that abstracts the polyhedral switched system.
  - *Graph analysis* is essentially the model-checking phase, wherein, the weighted graph constructed in the previous step is analysed for cycles with product of weights on its edges > 1 or ≥ 1, depending on whether the user wants to analyse for Lyapunov or asymptotic stability, respectively.
- *Output* of AVERIST is the output from the graph analysis. It either says that the system is stable (Lyapunov or asymptotic, as requested by the user), or outputs a counterexample — a cycle with product of weights > 1 or ≥ 1 — indicating a potential reason for instability.

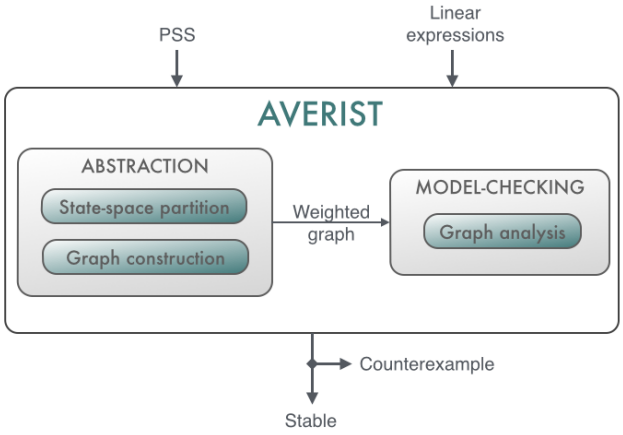

Fig. 2. AVERIST architecture

AVERIST has been implemented in Python and uses several software packages. It uses Parma Polyhedra Library (PPL) [6] to represent the invariants and guards of the polyhedral switched system and to perform the polyhedral operations in the state-space partition. The GNU Linear Programming Kit GLPK [2] solver is used

for solving the optimization problems involved in the weight computation. The NetworkX Python package [3] is used to manage graphs and perform graph analysis in the model-checking phase. All these utilities are included in the free open-source mathematics software system `sage` [4].

3 Running AVERIST

We describe the input and output formats and explain how to run `AVERIST`. Here, the main features are highlighted; the details can be found in [1].

3.1 Input format

The polyhedral switched system is specified using a programming language like syntax (see Figure 3 for an example); it consists of the following components:

(i) *Variables*: They are used to capture the continuous dynamics of the system and represent the continuous state of the system. They are specified as strings separated by commas with the tag 'var'. The number of variables determines the dimension of the system:

```
var:  x, y;
```

(ii) *Locations*: They correspond to the modes of the system and are defined as strings separated by commas with the tag 'location':

```
location:  quad1, quad2, quad3, quad4;
```

For each location a block is defined, which contains the invariant, dynamics and guards related to this location. The block begins with the name of the location with the 'loc' tag:

```
loc:  quad1;
```

The block consists of the following components:

(a) *Invariants*: These are conditions over the variables that are required to hold while the control is in the location. They are defined by a conjunction of linear constraints over the variables representing a polyhedral set, and are specified with the tag 'inv':

```
inv:  x >= 0 AND y >= 0;
```

(b) *Dynamics*: This specifies the evolution of the continuous state as a solution of a polyhedral inclusion dynamics and is specified as a conjunction of linear constraints over the derivatives of the variables specified by adding a prefix 'd' to the variables. The tag 'dyn' is used to specify the dynamics:

```
dyn:  dx == -1 AND dy >=1 AND dy <= 2;
```

(c) *Guards*: These capture constraints over the variables, that need to be satisfied for a switching to a certain mode to be enabled. They are tagged with 'guards' and consist of the switching condition specified as a conjunction of linear constraints over the variables and the target mode of the switching.

```
guards:
    when y == 0 goto quad2;
```

In addition, one can optionally provide a set of linear expressions to use in the

```
var:  x, y;
location:  quad1, quad2, quad3, quad4;

loc:  quad1;
     inv:  x >= 0 AND y >= 0;
     dyn:  dx == -1 AND dy >= 1 AND dy <= 2;
     guards:
          when y == 0 goto quad2;

loc:  quad2
     inv:  x <= 0 AND y >= 0;
     dyn:  dx >= -2 AND dx <= -1 AND dy == -4;
     guards:
          when x == 0 goto quad3;

loc:  quad3;
     inv:  x <= 0 AND y <= 0;
     dyn:  dx == 1 AND dy <= -1 AND dy >= -2;
     guards:
          when y == 0 goto quad4;

loc:  quad4;
     inv:  x >= 0 AND y <= 0;
     dyn:  dx >= 1 AND dx <= 2 AND dy == 4;
     guards:
          when x == 0 goto quad1;
```

Fig. 3. Input format example

state-space partition. These linear expressions are homogeneous. For instance, the syntax for the linear expression corresponding to the axes and the lines diagonal is:

```
x
y
x - y
x + y
```

3.2 Run AVERIST

To run AVERIST, the input polyhedral switched system is stored in `input.dat`, and the optional set of linear expressions for partitioning the state-space is stored in `linearexp.dat`. AVERIST is run in `sage` by the command `load("Main.py")`.

Next, an interactive dialogue is used to determine the set of linear expressions which will partition the state-space, and an example is shown in Figure 4. The user needs to specify if she wants to specify the set of linear expressions to be added or would like it to be generated automatically. In the case of the former, the linear expressions from `linearexp.dat` are used for partition creation. In the case of the latter, additional information regarding the granularity of the regions that is desired, is requested, to generate the appropriate set of linear expressions. Also, there is an option for the user to specify if she wants to add the linear expressions appearing in the system description (invariants and guards) to be added to the set of linear expressions to be used for partition creation.

3.3 Output

The main output of AVERIST consists of answering if the input polyhedral switched system is stable. It also returns a counter-example in the event that the tool is unable to determine stability of the system. This data is stored in the following

```
sage: load('Main.py')

* Please specify the path for the folder in which the experiment data (input.dat) is stored:
/Users/mgarcia/Experiment

* Do you want the linear expressions for creating the regions to be generated automatically
(A) or do you want to add them manually (M)? Enter A/M:
A

* The linear expressions will be generated in a uniform fashion. Please specify the granular
ity -- a natural number (higher number indicates finer partition):
0

* In addition, do you want to add the linear expressions appearing in the input hybrid autom
aton? Enter Y/N:
N

STABILITY ANSWER = Stable
```

Fig. 4. Interactive `AVERIST` dialogue

files:

- `stable.dat`: It contains one of the following answers: "Stable" or "Abstract counterexample".
- `counterexample.dat`: When `stable.dat` contains "Abstract counterexample", this file contains the list of nodes in the counterexample.

References

[1] Algorithmic verifier of stability: http://software.imdea.org/projects/averist/index.html.

[2] Glpk: https://www.gnu.org/software/glpk/.

[3] NetworkX: https://networkx.github.io/.

[4] sage: http://www.sagemath.org/.

[5] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei–Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. pages 209–229, 1992.

[6] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[7] Laurent Doyen, Thomas A. Henzinger, and Jean franois Raskin. Automatic rectangular refinement of affine hybrid systems. pages 144–161. Springer, 2005.

[8] Pavithra Prabhakar and Miriam Garcia Soto. Abstraction based model-checking of stability of hybrid systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 280–295, 2013.

[9] Pavithra Prabhakar and Miriam García Soto. An algorithmic approach to stability verification of polyhedral switched systems. In *American Control Conference*, 2014.

[10] A. Puri, V.S. Borkar, and P. Varaiya. ϵ-approximation of differential inclusions. pages 362–376, 1995.