# GSPeeDI – a Verification Tool for Generalized Polygonal Hybrid Systems

Hallstein A. Hansen[1] and Gerardo Schneider[2]

[1] Buskerud University College, Kongsberg, Norway
Hallstein.Asheim.Hansen@hibu.no
[2] Dept. of Informatics, University of Oslo, Oslo, Norway.
gerardo@ifi.uio.no

**Abstract.** The GSPeeDI tool implements a decision procedure for the reachability analysis of GSPDIs, planar hybrid systems whose dynamics is given by differential inclusions, and that are not restricted by the goodness assumption from previous work on the so-called SPDIs.
Unlike SPeeDI (a tool for reachability analysis of SPDIs) the underlying analysis of GSPeeDI is based on a breadth-first search algorithm, and it can handle more general systems.

## 1 Introduction

Hybrid systems combine dynamic and discrete behavior, and mathematical models can be defined for systems arising from real scenarios (e.g., a chemical plant) as well as for artificial constructions (e.g. by *hybridizing* a complex differential equation into connected piece-wise smaller equations). These systems are generally hard to analyze: most important verification problems are undecidable for non-trivial classes of hybrid systems. In this paper we deal with a class of planar hybrid systems whose dynamics is given by differential inclusions: *generalized polygonal hybrid systems* (GSPDIs). The reachability problem for GSPDI has been shown to be decidable [7].

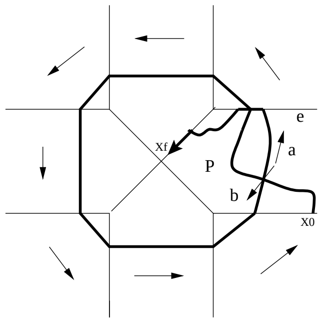


**Fig. 1.** GSPDI.

A GSPDI is a pair $\mathcal{H} = \langle \mathbb{P}, \mathbb{F} \rangle$, where $\mathbb{P}$ is a finite partition of the plane (each $P \in \mathbb{P}$ being a convex polygon), called the *regions* of the GSPDI, and $\mathbb{F}$ is a function associating a pair of vectors to each polygon: $\mathbb{F}(P) = (\mathbf{a}_P, \mathbf{b}_P)$. In a GSPDI every point on the plane has its dynamics defined according to which polygon it belongs to: if $\mathbf{x} \in P$, then $\dot{\mathbf{x}} \in \angle_{\mathbf{a}_P}^{\mathbf{b}_P}$. The angle $\angle_{\mathbf{a}}^{\mathbf{b}}$ denotes a differential inclusion, meaning that the tangent vector at any point of a given trajectory must be a linear combination of vectors $\mathbf{a}$ and $\mathbf{b}$.

A complicating factor in the reachability analysis of GSPDIs is the presence of regions where the trajectory is allowed to enter and leave the region through the same edge, to slide along, or *bounce* off a given edge. For instance, in the example shown in Fig. 1 the dynamics of region $P$ allows the trajectory to slide and bounce off the edge $e$. A region where no trajectory can enter and leave through the same edge is said to be *good*, and a GSPDI where all the regions are good is called an SPDI (we say that that SPDI satisfies the *goodness assumption*).

The tool GSPeeDI is the only tool we know of that implements a decision procedure to solve the reachability problem for this particular class of hybrid systems.

## 2 GSPeeDI

The tool GSPeeDI[3] is a collection of utilities to manipulate and reason mechanically about GSPDIs. It is implemented in 3000 lines of Python code.

The tool takes as input a GSPDI, together with a source and a target interval, on given edges. The tool operates on a graph whose nodes are the edges of the polygons (and not the polygons themselves) which are connected with directed arcs labelled with edge-to-edge one-dimensional successor functions over edge intervals. An *edge interval* represents an interval on a given edge. In order to check for reachability, we use a standard breadth-first search (BFS) model checking approach. We start from a set $A$ containing an initial edge interval. Then we iteratively apply the possible transitions from the current set, adding the resulting edge intervals to $A$. The search ends if an edge interval from $A$ contains the sought-after edge interval, or it ends when, if the sought-after edge interval cannot be reached, the fix-point is reached.

There are three kinds of possible transitions that one may take:

- Edge-to-edge transitions. The result of following the dynamics of a region, and represented as one step transition on the graph.
- 'Cycle' transitions. We only need to analyze simple cycles, which are then converted into *meta-transitions* in the graph. Using acceleration of such simple cycles we are able to compute all the edge intervals reachable by taking the cycle any number of times, without iterating the cycle in most cases.
- Sink transitions. We know how to identify those simple cycles that cannot be exited. These transitions will only be applied at the end since no other continuation is possible.

Using acceleration techniques for analyzing cycles instead of iterating them, we can sucessfully analyze large systems, even though the algorithm has a worst case complexity which is doubly exponential.

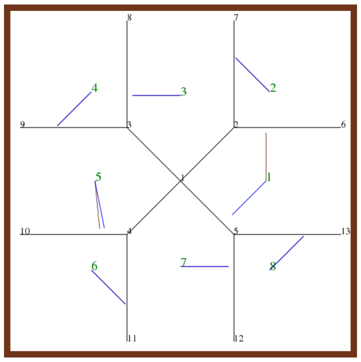
On the left of Fig. 2 we illustrate a typical input file containing a GSPDI composed of 8 regions. From this file we can generate the edge-to-edge transitions, and then create a picture of the GSPDI as shown in the upper right part of the figure. The lower right hand part shows a typical use scenario, where the imprecision is caused by a built-in floating point conversion routine in Python.

[3] http://heim.ifi.uio.no/hallstah/gspeedi/

**Input file**

```
# Points
P  1 1.5 1.5
P  2 2 2
P  3 1 2
P  4 1 1
P  5 2 1
P  6 3 2
P  7 2 3
P  8 1 3
P  9 0 2
P 10 0 1
P 11 1 0
P 12 2 0
P 13 3 1

# Regions

R 1 1 9 1 2 6 13 5
R 2 2 2 2 7 6
R 3 3 3 1 3 8 7 2
R 4 4 4 3 9 8
R 5 5a 5b 1 4 10 9 3
R 6 6 6 4 11 10
R 7 7 7 1 5 12 11 4
R 8 8 8 5 13 12

# Vectors

V 1 0 1
V 2 -1 1
V 3 -1 0
V 4 -1 -1
V 5b 0.2 -1
V 5a 0.1 -1
V 6 1 -1
V 7 1 0
V 8 1 1
V 9 -1.1 -1.1
```

**Generated figure**



**Session log**

```
./search.py data/swimmer-g.graph '2->3~2-7' 0.5 '2->3~2-7' 0.2
Searching from:
   N2->3~2-7   at   [[0.5, 0.5]]
to
   N2->3~2-7   at   [[0.20000000000000001, 0.20000000000000001]]
True
```

In this example the search refers to a node in the graph, `N2->3 2-7`, which represents the edge `2-7`. Since any edge may be traversed in both directions, we uniquely identify that we traverse the edge going from region 2 to region 3.

**Fig. 2.** Example.

While the example contains only 8 regions, it has 84 simple edge cycles (including permutations). We may use various optimization techniques to reduce the search space, but even so, the number of possible paths reachable from a single edge may number in the thousands when we combine the cycles with edge-to-edge transitions.

## 3 Comparing and Contrasting with SPeeDI

A comparison with HyTech [3] is not meaningful since HyTech semi-decides more general hybrid systems than GSPeeDI, but it runs out of memory very quickly for very simple GSPDIs for which GSPeeDI gives an almost immediate answer due to acceleration. The obvious tool to compare GSPeeDI with is the tool SPeeDI [1], since GSPeeDI generalizes SPeeDI.

Our tool contains two major enhancements over SPeeDI, which justified a completely new implementation of reachability analysis for GSPDIs: We can analyze systems that are not restricted by the goodness assumption, and we do so using breadth-first (instead of depth-first) search.

Being able to analyze systems where the goodness asumption does not hold increases the number of analyzable systems. The practical implications for the design of the tool are considerable, and include a more complex vector/function library, and looser restrictions on what constitutes a feasible path of traversed edges and cycles. This in turn leads to a larger search space, so if all the regions are good, then SPeeDI performs much better, but SPeeDI cannot handle systems with non-good regions.

Another difference is that SPeeDI's algorithm is based on depth-first generation of feasible paths. While the depth-first algorithm may not necessarily generate the shortest possible counter example, it does have the advantage of generating the counter example as part of the algorithm itself.

## 4 Complexity

There are two main factors contributing to the run-time complexity of the tool. One is the computation of all the simple cycles in the directed GPSDI graph. The other is the execution of the breadth-first search algorithm. The latter has been shown to have a doubly exponential time complexity in the worst case. However, in practice we can apply a set of heuristics which reduce this complexity considerably, as explained below.

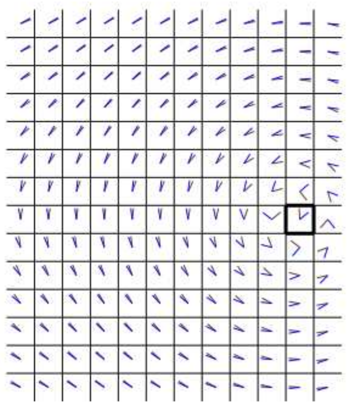


**Fig. 3.** Larger GSPDI example

Computing all simple cycles may be infeasible for large graphs: The number of simple cycles in a complete, directed graph with $n$ nodes is exactly

$$\sum_{i=1}^{n-1} \binom{n}{n-i+1} (n-i)!$$

For computing all simple cycles we have implemented the algorithm due to Tarjan [8], which has a time bound of $O((n+e)(c+1))$, where $e$ is the number of edges and $c$ the number of cycles in the graph. Clearly, the number of cycles is the factor determining the point at which a problem becomes infeasible.

Informal testing have shown that running an unmodified algorithm on examples with hundreds of nodes quickly becomes infeasible, both due to the execution time of the algorithm, and the number of (unpermutated) cycles.

Because of this the tool includes several domain specific optimizations to the algorithm. In particular we only investigate prefixes to cycles where:

– There actually are trajectories that make a complete cycle.
– The cycles will not be redundant. Cycles where the trajectories *bounce* off edges are not required to be analyzed.
– The generation of a cycle will not help analysis. This happens if a node represents an interval that may be reached in its entirety by any trajectory.

We will demonstrate the optimizations' effectiveness on a bigger example (partially shown in Fig. 3). The GSPDI contains 334 nodes (in the reachability graph). A run of the unmodified algorithm finds that the total number of cycles (without permutations) is 181398.

If we apply only the first optimization, we reduce that number to 1041 cycles. Adding the second optimization reduces the number to 112 cycles, and applying all three leaves us with 85 cycles. The optimizations cut off 811, 229, and 183 prefixes respectively.

So, for this particular example, we find that more than 99% of the possible cycles are redundant. Computing the number of permutations gives us a total of 1100 cycles subsequently used as meta-transitions in the breadth-first search.

The execution time of the program which generates the cycles is less than a minute on a low-end, modern CPU. On the same system a reachability search returning *false* (thus having computed the entire reach-set for a particular start-interval) finishes execution in slightly over ten seconds.

## 5 Discussion

We have presented a prototype tool for solving the reachability problem for generalized polygonal hybrid systems. The tool implements a BFS algorithm (as presented in [4]), following the theoretical results published in [7]. The algorithm is based on the analysis of a finite number of possible qualitative behaviors, including only simple loops which may be accelerated in most cases. Since the number of such behaviors may be extremely big, the tool uses several powerful heuristics that exploit the topological properties of planar trajectories for considerably reducing the set of actually explored paths on the reach-graph.

The main applications of GSPDIs is to over-approximate non-linear differential equations on the plane. Then we can apply GSPeeDI to perform reachability analysis. There is ongoing and future work in the area of automatically[4] partitioning the plane and generating GSPDIs based on such equations, based on whether properties such as Lipschitz continuity applies and can be exploited. This will allow for analysis of larger, real-world problems. The application of GSPeeDI to over-approximate planar differential equations could be combined with simulation techniques in order to further refine parts of the hybridized equation to make more precise analysis. This, together with the use of the phase

[4] A simple, ad-hoc application for automatic partitioning is distributed with the tool.

portrait (see below) will produce less 'do not know' answers and increase the number of 'yes' and 'no' answers.

One line of future work is incorporating support for enhancements, optimizations and utilities currently available for SPeeDI, that have been already explored theoretically for SPDIs. This include the computation of the phase portrait of a system [2], which may allow both optimizations [6] and compositional parallelization [5] of the reachability analysis algorithm. Note that the implementation of such features will not add to the complexity of the tool as all the information needed to compute the phase portrait (invariance, viability and controllability kernels, and semi-separatrices) is already computed when analyzing simple cycles (see [5, 6] for more details).

We conjecture that the cycle generation and breadth-first search may mutually benefit from running in parallell and working with a shared state. There may, for example, be no need to generate cycles for sufficiently explored parts of the graph.

*Acknowledgments.* We would like to thank Gordon Pace for useful suggestions on how to improve the efficiency of the tool.

## References

[1] E. Asarin, G. Pace, G. Schneider, and S. Yovine. SPeeDI: a verification tool for polygonal hybrid systems. In *CAV'02*, volume 2404 of *LNCS*, pages 354–358, 2002.

[2] E. Asarin, G. Schneider, and S. Yovine. Towards computing phase portraits of polygonal differential inclusions. In *HSCC'02*, number 2289, 2002.

[3] T. Henzinger, P.-H.Ho, and H.Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1), 1997.

[4] G. Pace and G. Schneider. Model checking polygonal differential inclusions using invariance kernels. In *VMCAI'04*, volume 2937 of *LNCS*, pages 110–121, 2003.

[5] G. Pace and G. Schneider. A compositional algorithm for parallel model checking of polygonal hybrid systems. In *ICTAC'06*, volume 4281 of *LNCS*, pages 168–182, 2006.

[6] G. Pace and G. Schneider. Static analysis for state-space reduction of polygonal hybrid systems. In *FORMATS'06*, volume 4202 of *LNCS*, pages 306–321, 2006.

[7] G. J. Pace and G. Schneider. Relaxing goodness is still good. In *ICTAC'08*, volume 5160 of *LNCS*, pages 274–289, Istanbul, Turkey, September 2008.

[8] R. E. Tarjan. Enumeration of the elementary circuits of a directed graph. Technical report, Ithaca, NY, USA, 1972.