dReach: δ-Reachability Analysis for Hybrid Systems

Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke

Computer Science Department, Carnegie Mellon University, USA

Abstract. dReach is a bounded reachability analysis tool for nonlinear hybrid systems. It encodes reachability problems of hybrid systems to first-order formulas over real numbers, which are solved by delta-decision procedures in the SMT solver dReal. In this way, dReach is able to handle a wide range of highly nonlinear hybrid systems. It has scaled well on various realistic models from biomedical and robotics applications.

1 Introduction

dReach is a bounded reachability analysis tool for hybrid systems. It encodes bounded reachability problems of hybrid systems as first-order formulas over the real numbers, and solves them using δ-decision procedures in the SMT solver dReal [13]. dReach is able to handle a wide range of highly nonlinear hybrid systems [17,14,16,3]. Figure 1 highlights some of its features: on the left is an example of some nonlinear dynamics that dReach can handle, and on the right a visualized counterexample generated by dReach on this model.

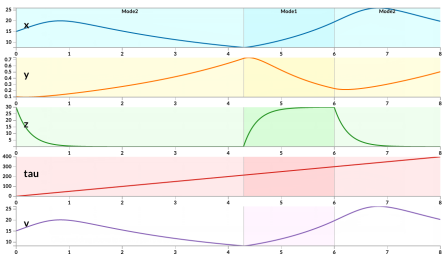
$$\frac{dx}{dt} = \left(\alpha_x\left(k_1 + \frac{(1-k_1)z}{z+k_2}\right) - \beta_x\left(k_3 + \frac{(1-k_3)z}{z+k_4}\right) - m_1\left(1-\frac{z}{z_0}\right)\right)x$$

$$\frac{dy}{dt} = m_1\left(1-\frac{z}{z_0}\right)x + \left(\alpha_y\left(1-d\frac{z}{z_0}\right) - \beta_y\right)y$$

$$\frac{dz}{dt} = \frac{z_0 - z}{\tau}$$

$$\frac{dv}{dt} = \left(\alpha_x\left(k_1 + \frac{(1-k_1)z}{z+k_2}\right) - \beta_x\left(k_3 + \frac{(1-k_3)z}{z+k_4}\right) - m_1\left(1-\frac{z}{z_0}\right)\right)x + m_1\left(1-\frac{z}{z_0}\right)x + \left(\alpha_y\left(1-d\frac{z}{z_0}\right) - \beta_y\right)y$$

(a) An example of nonlinear hybrid system model: off-treatment mode of the prostate cancer treatment model [17]



(b) Visualization of a generated counterexample. Change in the shade of colors represents discrete mode changes.

Fig. 1: An example of nonlinear dynamics and counterexample-generation.

It is well-known that the standard bounded reachability problems for simple hybrid systems are already highly undecidable [2]. Instead, we work in the

framework of δ-reachability of hybrid systems [11]. Here δ is an arbitrary positive rational number, provided by the user to specify the bound on numerical errors that can be tolerated in the analysis. For a hybrid system H and an unsafe region unsafe (both encoded as logic formulas), the δ-reachability problem asks for one of the following answers:

- safe: H cannot reach unsafe.
- δ-unsafe: H^δ can reach unsafe$^\delta$.

Here, H^δ and unsafe$^\delta$ encode (δ-bounded) overapproximations of H and unsafe, defined explicitly as their syntactic variants. (See Section 4.2 in the Appendix.) It is important to note that the definition makes the answers no weaker than standard reachability: When safe is the answer, we know for certain that H does not reach the unsafe region (no δ is involved); when δ-unsafe is the answer, we know that there exists some δ-bounded perturbation of the system that can render it unsafe. Since δ can be chosen to be very small, δ-unsafe answers in fact discover robustness problem in the system, which should be regarded as unsafe indeed. We have proved that bounded δ-reachabilty is decidable for a wide range of nonlinear hybrid systems, even with reasonable complexity bounds [11]. This framework provides the formal correctness guarantees of dReach.

Apart from solving δ-reachability, the following key features of dReach distinguish it from other existing tools in this domain [7,9,1,8,15,5,6].

1. Expressiveness. dReach allows the user to describe hybrid systems using first-order logic formulas over real numbers with a wide range of nonlinear functions. This allows the user to specify the continuous flows using highly nonlinear differential equations, and the jump and reset conditions with complex Boolean combinations of nonlinear constraints. dReach also faithfully translates mode invariants into $\exists\forall$ logic formulas, which can be directly solved under certain restrictions on the invariants.

2. Property-guided search. dReach maintains logical encodings (the same approach as [6]), whose size is linear in the size of the inputs, of the reachable states of a hybrid system [11]. The tool searches for concrete counterexamples to falsify the reachability properties, instead of overapproximating the full reachable states. This avoids the usual state explosion problem in reachable set computation, because the full set of states does not need to be explicitly stored. This change is analogous to the difference between SAT-based model checking and BDD-based symbolic model checking.

3. Tight integration of symbolic reasoning and numerical solving. dReach delegates the reasoning on discrete mode changes to SAT solvers, and uses numerical constraint solving to handle nonlinear dynamics. As a result, it can combine the full power of both symbolic reasoning and numerical analysis algorithms. In particular, all existing tools for reachable set computation can be easily plugged-in as engines for solving the continuous part of the dynamics, while logic reasoning tools can overcome the difficulty in handling complex mode transitions.

The paper is structured as follows. We describe the system architecture in Section 2, and give some details about the logical encoding in the tool in Section

3. We then explain the input format and usage in Section 4. More details and examples are given in the Appendix.

2 System Description

The system architecture of dReach is given in Figure 2. We ask the user to provide the following input file and two parameters:

- The input file specifies the hybrid system, the reachability properties in question, and some time bounds on the continuous flow in each mode. The grammar is described in Section 4.1.
- A bound on the number of mode changes.
- A numerical error bound δ, as explained in Section A5. in the Appendix.

From these inputs, dReach generates a logical encoding that involves existential quantification and universal quantification on the time variables. The logical encoding is compact, always linear in the size of the inputs. The tool then makes iterative calls to the underlying solver dReal [13] to decide the reachability properties. When the answer is δ-reachable, dReach generates a counterexample and its visualization. When the answer is unreachable, no numerical error is involved and a (partial, for now) logical proof of unsatisfiability can be provided [12].

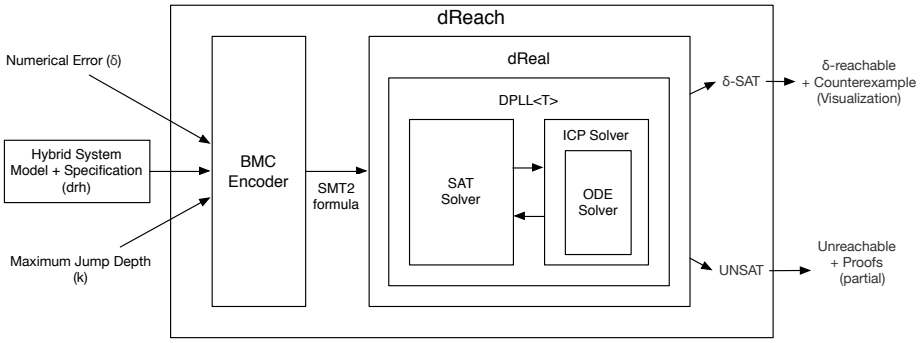


Fig. 2: Architecture of dReach: It consists of an bounded model-checking module and an SMT solver, dReal. In the first phase, the Encoder module translates an input hybrid system into a logic formula. In the second phase, an SMT solver, dReal, solves the encoded δ-reachability problem using a solving framework that combines DPLL(T), Interval Constraint Propagation, and reliable (interval-based) numerical integration.

3 Logical Encoding of Reachability

The details of our encoding scheme is given in [11]. Here we focus on explaining how differential equations and the universal quantifications generated by mode

invariants are encoded, as an extension of the SMT-LIB [4] standard. Although such formulas are automatically generated by dReach from the hybrid system descrpition, the explanation below can be helpful for understanding the inner mechanism of our solver.

Encoding integrations. In each mode of a hybrid system, we need to specify continuous flows defined by systems of ordinary differential equations. We extend SMT-LIB with a command `define-ode` to define such systems. For instance, we use `define-ode` as follows to assign a name flow_1 to a group of ODE, $\frac{\mathrm{d}x}{\mathrm{d}t} = v$ and $\frac{\mathrm{d}v}{\mathrm{d}t} = -x^2$.

```
(define-ode flow1 ((= d/dt[x] v) (= d/dt[v] (- 0 (^ x 2)))))
```

We then allow integration terms in the formula. We view the solution of system of differential equations as a constraint between the initial-state variables, time duration, and the end-state variables. We can then write

```
(= [x_t_1 ... x_t_n] (integral 0 t [x_0_1 ... x_0_n] flow_i)),
```

to represent $\boldsymbol{x} = \boldsymbol{x}_0 + \int_0^t \mathit{flow}_i(\boldsymbol{x}(s))\mathrm{d}s$. Note that we do not need to explicitly mention $\boldsymbol{x}(s)$ as a function in the encoding, which can be inferred by the solver.

Universal quantification for mode invariant constriants. To encode mode invariants in hybrid systems, we need $\exists\forall^t$-formulas [14] which is a restricted form of $\exists\forall$ formula where the universal quantifications are limited to the time variables. In dRh, we introduce a new keyword `forall_t` to encode $\exists\forall^t$ formulas. Given a time bound $[0, \mathit{time}_i]$, mode invariant f at mode n is encoded into `(forall_t n [0 time_i] f)`.

4 Using dReach

4.1 Input Format

The input format for describing hybrid systems and reachability properties consists of five sections: macro definitions, variable declarations, mode definitions, and initial condition, and goals. We focus on intuitive explanations here, and the formal grammar is given in the Appendix. Figure 3 shows how to describe a small example hybrid system, an inelastic bouncing ball with air resistance.

- In macro definitions, we allows users to define macros in C preprocessor style which can be used in the following sections. Macro expansions occur before the other parts are processed.
- A variable declaration specifies a real variable and its domain in a real interval. dReach requires special declaration for *time* variable, to specify the upperbound of time duration.

– A mode definition consists of mode id, mode invariant, flow, and jump. *id* is a unique positive interger assigned to a mode. An invariant is a conjuction of logic formulae which must always hold in a mode. A flow describes the continuous dynamics of a mode by providing a set of ODEs. The first formula of *jump* is interpreted as a guard, a logic formula specifying a condition to make a transition. Note that this allows a transition but does not force it. The second argument of *jump*, n denotes the target mode-id. The last one is *reset*, a logic formula connecting the old and new values for the transition.
– *initial-condition* specifies the initial mode of a hybrid system and its initial configuration. *goal* shares the same syntactic structure of *initial-condition*.

```
1  #define D 0.45
2  #define K 0.9
3  [0, 15] x; [9.8] g; [-18, 18] v; [0, 3] time;
4  {   mode 1;
5      invt: (v <= 0);  (x >= 0);
6      flow: d/dt[x] = v; d/dt[v] = -g - (D * v ^ 2);
7      jump: (x = 0) ==> @2 (and (x' = x) (v' = - K * v)); }
8  {   mode 2;
9      invt: (v >= 0); (x >= 0);
10     flow: d/dt[x] = v; d/dt[v] = -g + (D * v ^ 2);
11     jump: (v = 0) ==> @1 (and (x' = x) (v' = v)); }
12 init: @1 (and (x >= 5) (v = 0));
13 goal: @1 (and (x >= 0.45));
```

Fig. 3: An example of `drh` format: Inelastic bouncing ball with air resistance. Lines 1 and 2 define a drag coefficient $D = 0.45$ and an elastic coefficient $K = 0.9$. Line 3 declares variables x, g, v, and $time$. At lines 4 - 7 and 8 - 11, we define two modes – the falling and the bouncing-back modes respectively. At line 12, we specify the hybrid system to start at mode 1 (`@1`) with initial condition satisfying $x \geq 5 \wedge v = 0$. At line 13, it asks whether we can have a trajectory ending at mode 1 (`@1`) while the height of the ball is higher than 0.45.

4.2 Command Line Options

dReach follows the standard unix command-line usage:

```
dReach <options> <drh file>
```

It has the following options:

– If `-k <N>` is used, set the unrolling bound k as N (Default: 3). It also provides `-u <N>` and `-l <N>` options to specify upper- and lower-bounds of unrolling bound.
– If `--precision <p>` is used, use precision p (Default: 0.001).
– If `--visualize` is set, dReach generates extra visualization data.

We have a web-based visualization toolkit[1] which processes the generated visualization data and shows the counterexample trajectory. It provides a way to

[1] The detailed instructions are available at `https://github.com/dreal/dreal/blob/master/doc/ode-visualization.md`.

navigate and zoom-in/out trajectories which helps understand and debug the target hybrid system better.

References

1. M. Althoff and B. H. Krogh. Reachability analysis of nonlinear differential-algebraic systems. *IEEE Trans. Automat. Contr.*, 59(2):371–383, 2014.
2. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992.
3. H. u. Asad, K. D. Jones, and F. Surre. Verifying robust frequency domain properties of non linear oscillators using SMT. In *Design and Diagnostics of Electronic Circuits Systems, 17th International Symposium on*, pages 306–309, April 2014.
4. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.
5. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS*, pages 183–192, 2012.
6. A. Cimatti, S. Mover, and S. Tonetta. Smt-based verification of hybrid systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.
7. M. Fränzle, T. Teige, and A. Eggers. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *J. Log. Algebr. Program.*, 79(7):436–466, 2010.
8. G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.
9. 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 *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 379–395, 2011.
10. S. Gao, J. Avigad, and E. M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2012.
11. S. Gao, S. Kong, W. Chen, and E. M. Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *CoRR*, abs/1404.7171, 2014.
12. S. Gao, S. Kong, and E. Clarke. Proof generation from delta-decisions. In *SYNASC*, 2014.
13. S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE*, pages 208–214, 2013.
14. S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo ODEs. In *FMCAD*, pages 105–112, 2013.
15. C. Herde, A. Eggers, M. Fränzle, and T. Teige. Analysis of hybrid systems using hysat. In *ICONS*, pages 196–201, 2008.
16. J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Arechiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In *HSCC'14, Berlin, Germany, April 15-17, 2014*, pages 133–142, 2014.
17. B. Liu, S. Kong, S. Gao, and E. Clarke. Parameter identification using delta-decisions for biological hybrid systems. In *CMSB*, 2014.

Appendix

A1. How to Install

dReach is an open source project under GPL-3 license. We bundled dReal and dReach together and host them at `http://dreal.github.io`. The BMC encoder module is written in OCaml and uses Oasis and OCaml Batteries library. At the release page[2], we host pre-compiled static-binaries for Linux and OS X, which do not require any compilation to use dReach in those platforms.

A2. Syntax Grammar of drh

$$
\begin{aligned}
\mathit{drh} &:= \mathit{macro\_def}^* \ \mathit{variable\_decl}^+ \ \mathit{mode\_def}^+ \ \mathit{initial\_cond} \ \mathit{goal}^+ \\
\mathit{macro\_decl} &:= \texttt{\#define} \ \mathit{var} \ (\mathit{expr} \mid \mathit{formula}) \\
\mathit{variable\_decl} &:= \texttt{[}l\texttt{, }u\texttt{]} \ \mathit{var}\texttt{;} \\
\mathit{mode\_def} &:= \{\texttt{mode} \ \mathit{id}\texttt{;} \quad \texttt{invt} : (\mathit{formula}\texttt{;})^+ \quad \texttt{flow} : \mathit{ode}^+ \quad \texttt{jump} : \mathit{jump}^+\} \\
\mathit{ode} &:= \texttt{d/dt[}x\texttt{]=}\mathit{exp} \\
\mathit{jump} &:= \mathit{formula} \ \texttt{==>} \ \texttt{@}n \ \mathit{formula} \\
\mathit{initial\_cond} &:= \texttt{@}\mathit{mode\_id} \ \mathit{formula}\texttt{;} \\
\mathit{goal} &:= \texttt{@}\mathit{mode\_id} \ \mathit{formula}\texttt{;}
\end{aligned}
$$

Note that we use the standard definitions for *formlua* and *expr* here.

A3. An Example of Encoded SMT2 Formula

Example encoding The bounded reachability problem of a bouncing ball example (when $k = 3$) is encoded into the following shortened SMT2 formula.

```
(set-logic QF_NRA_ODE)
(declare-fun x_0_0 () Real) ...
(declare-fun v_0_t () Real) ...
(declare-fun time_0 () Real) ...
(define-ode flow_1 ((= d/dt[x] v)
                    (= d/dt[v] (+ (- 0.0 9.8) (* -0.45 (^ v 2.0))))))
(define-ode flow_2 ((= d/dt[x] v)
                    (= d/dt[v] (+ (- 0.0 9.8) (* +0.45 (^ v 2.0))))))
(assert (<= 0.0 x_0_0)) ...
(assert (<= v_10_t 18.0))
(assert (<= 0.0 time_0))
(assert (and (and (= v_0_0 0.0) (>= x_0_0 5.0)) (= mode_0 1.0) (=
[x_0_t v_0_t] (integral 0. time_0 [x_0_0 v_0_0] flow_1)) (= mode_0
1.0) (forall_t 1.0 [0.0 time_0] (<= v_0_t 0.0)) (<= v_0_t 0.0) (<=
...
x_9_t) (= [x_10_t v_10_t] (integral 0. time_10 [x_10_0 v_10_0]
flow_1)) (= mode_10 1.0) (forall_t 1.0 [0.0 time_10] (<= v_10_t 0.0))
(<= v_10_t 0.0) (<= v_10_0 0.0) (forall_t 1.0 [0.0 time_10] (>= x_10_t
0.0)) (>= x_10_t 0.0) (>= x_10_0 0.0) (= mode_10 1.0) (>= x_10_t
0.45))) (check-sat) (exit)
```

Benchmark	#Mode	#Depth	#ODEs	#Vars	Delta	Result	Time(s)	Trace
AF-GOOD	4	3	20	53	0.001	SAT	0.425	793K
AF-BAD	4	3	20	53	0.001	UNSAT	0.074	—
AF-TO1-GOOD	4	3	24	62	0.001	SAT	2.750	224K
AF-TO1-BAD	4	3	24	62	0.001	UNSAT	5.189	—
AF-TO2-GOOD	4	3	24	62	0.005	SAT	3.876	553K
AF-TO2-BAD	4	3	24	62	0.001	UNSAT	8.857	—
AF-TSO1-TSO2	4	3	24	62	0.001	UNSAT	0.027	—
AF8-K7	8	7	40	101	0.001	SAT	10.478	3.8M
AF8-K23	8	23	40	293	0.001	SAT	135.29	11M
EO-K2	3	2	18	48	0.01	SAT	3.144	1.9M
EO-K11	3	11	99	174	0.01	UNSAT	0.969	—
QUAD-K1	2	1	34	89	0.01	SAT	2.386	10M
QUAD-K2	2	2	34	125	0.01	SAT	4.971	13M
QUAD-K3	4	3	68	161	0.01	SAT	13.755	42M
QUAD-K3U	4	3	68	161	0.01	UNSAT	2.846	—
CT	2	2	10	41	0.005	SAT	345.84	3.1M
CT	2	2	10	41	0.002	SAT	362.84	3.1M
BB-K10	2	10	22	66	0.01	SAT	8.057	123K
BB-K20	2	20	42	126	0.01	SAT	39.196	171K

Table 1: Summary of the running time of the tool on various hybrid system models: #Mode = Number of modes in the hybrid system, #Depth = Unrolling depth, #ODEs = Number of ODEs in the unrolled formula, #Vars = Number of variables in the unrolled formula, Result = Bounded Model Checking Result (delta-SAT/UNSAT) Time = CPU time (s), Trace = Size of the ODE trajectory, AF = Atrial Filbrillation, EO = Electronic Oscillator, QUAD = Quadcopter Control, CT = Cancer Treatment, BB = Bouncing Ball with Drag.

A4. Experiments

All benchmarks and data shown here are also available on the tool website[3]. Due to space limit, we only highlight the typical nonlinear differential equations in the models. All models are hybrid systems with multiple modes containing these equations All experiments were conducted on a machine with a 3.4GHz octa-core Intel Core i7-2600 processor and 16GB RAM, running 64-bit Ubuntu 12.04LTS. Table 1 is a summary of the running time of the tool on various hybrid system models.

Atrial Fibrillation. We studied the Atrial Fibrillation model. The model has four discrete control locations, four state variables, and nonlinear ODEs. A typical set of

[2] `http://dreal.github.io/download`
[3] `http://dreal.github.io`

ODEs in the model is:

$$\begin{aligned}
\frac{du}{dt} &= e + (u - \theta_v)(u_u - u)vg_{fi} + wsg_{si} - g_{so}(u) \\
\frac{ds}{dt} &= \frac{g_{s2}}{(1 + \exp(-2k(u - us)))} - g_{s2}s \\
\frac{dv}{dt} &= -g_v^+ \cdot v \qquad \frac{dw}{dt} = -g_w^+ \cdot w
\end{aligned}$$

The exponential term on the right-hand side of the ODE is the sigmoid function, which often appears in modelling biological switches.

Prostate Cancer Treatment. The Prostate Cancer Treatment model [17] exhibits more nonlinear ODEs. The reachability questions are

$$\begin{aligned}
\frac{dx}{dt} &= (\alpha_x(k_1 + (1 - k_1)\frac{z}{z + k_2} - \beta_x((1 - k_3)\frac{z}{z + k_4} + k_3)) - m_1(1 - \frac{z}{z_0}))x + c_1x \\
\frac{dy}{dt} &= m_1(1 - \frac{z}{z_0})x + (\alpha_y(1 - d\frac{z}{z_0}) - \beta_y)y + c_2y \\
\frac{dz}{dt} &= \frac{-z}{\tau} + c_3z \\
\frac{dv}{dt} &= (\alpha_x(k_1 + (1 - k_1)\frac{z}{z + k_2} - \beta_x(k_3 + (1 - k_3)\frac{z}{z + k_4})) \\
&\quad -m_1(1 - \frac{z}{z_0}))x + c_1x + m_1(1 - \frac{z}{z_0})x + (\alpha_y(1 - d\frac{z}{z_0}) - \beta_y)y + c_2y
\end{aligned}$$

Electronic Oscillator. The EO model represents an electronic oscillator model that contains nonlinear ODEs such as the following:

$$\begin{aligned}
\frac{dx}{dt} &= -ax \cdot sin(\omega_1 \cdot \tau) \\
\frac{dy}{dt} &= -ay \cdot sin((\omega_1 + c_1) \cdot \tau) \cdot sin(\omega_2) \cdot 2 \\
\frac{dz}{dt} &= -az \cdot sin((\omega_2 + c_2) \cdot \tau) \cdot cos(\omega_1) \cdot 2 \\
\frac{\omega_1}{dt} &= -c_3 \cdot \omega_1 \quad \frac{\omega_2}{dt} = -c_4 \cdot \omega_2 \quad \frac{d\tau}{dt} = 1
\end{aligned}$$

Quadcopter Control. We developed a model that contains the full dynamics of a quadcopter. We use the model to solve control problems by answering reachability

questions. A typical set of the differential equations are the following:

$$\frac{\mathrm{d}\omega_x}{\mathrm{d}t} = L \cdot k \cdot (\omega_1^2 - \omega_3^2)(1/I_{xx}) - (I_{yy} - I_{zz})\omega_y\omega_z/I_{xx}$$
$$\frac{\mathrm{d}\omega_y}{\mathrm{d}t} = L \cdot k \cdot (\omega_2^2 - \omega_4^2)(1/I_{yy}) - (I_{zz} - I_{xx})\omega_x\omega_z/I_{yy}$$
$$\frac{\mathrm{d}\omega_z}{\mathrm{d}t} = b \cdot (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)(1/I_{zz}) - (I_{xx} - I_{yy})\omega_x\omega_y/I_{zz}$$
$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = \omega_x + \frac{\sin(\phi)\sin(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_y + \frac{\sin(\theta)}{\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)}\omega_z$$
$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\left(\frac{\sin(\phi)^2\cos(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)}\omega_y + \cos(\phi)\cos(\theta)\right)\cos(\phi)^2} + \frac{1}{\cos(\phi)}\right)\omega_y - \frac{\sin(\phi)\cos(\theta)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_z$$
$$\frac{\mathrm{d}\psi}{\mathrm{d}t} = \frac{\sin(\phi)}{\left(\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)\right)\cos(\phi)}\omega_y + \frac{1}{\frac{\sin(\phi)^2\cos(\theta)}{\cos(\phi)} + \cos(\phi)\cos(\theta)}\omega_z$$
$$\frac{\mathrm{d}xp}{\mathrm{d}t} = (1/m)(\sin(\theta)\sin(\psi)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot xp)$$
$$\frac{\mathrm{d}yp}{\mathrm{d}t} = (1/m)(-\cos(\psi)\sin(\theta)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot yp)$$
$$\frac{\mathrm{d}zp}{\mathrm{d}t} = (1/m)(-g - \cos(\theta)k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) - k \cdot d \cdot zp$$
$$\frac{\mathrm{d}x}{\mathrm{d}t} = xp, \frac{\mathrm{d}y}{\mathrm{d}t} = yp, \frac{\mathrm{d}z}{\mathrm{d}t} = zp$$

A5. Bounded δ-reachability

Let $H = \langle X, Q, \mathsf{flow}, \mathsf{jump}, \mathsf{inv}, \mathsf{init}\rangle$ be a hybrid system as standardly defined. We use first-order formulae over the real numbers to represent H, by writing

$$H = \langle X, Q, \varphi_{\mathsf{flow}}, \varphi_{\mathsf{jump}}, \varphi_{\mathsf{inv}}, \varphi_{\mathsf{init}}\rangle$$

where $\varphi_{\mathsf{flow}}, \varphi_{\mathsf{jump}}, \varphi_{\mathsf{inv}}$ and φ_{init} are logic formulae that define the corresponding predicates in the standard definition. Now, let $\delta \in \mathbb{Q}^+$ be a chosen error bound, we define the δ-perturbation of H to be

$$H^\delta = \langle X, Q, \varphi^\delta_{\mathsf{flow}}, \varphi^\delta_{\mathsf{jump}}, \varphi^\delta_{\mathsf{inv}}, \varphi^\delta_{\mathsf{init}}\rangle.$$

Here, φ^δ is a syntactic variant of φ which relaxes the numerical terms in φ up to an error bound δ. The notion is formally defined in our recent work [10,11]. We now define the bounded δ-reachability problem that dReach solves.

Let $n \in \mathbb{N}$ be a bound and $T \in \mathbb{R}^+$ be an upper bound of time duration. We write unsafe to denote a subset of the state space of H defined by a first-order formula. The bounded δ-reachability problem asks for one of the following answers

- H cannot reach unsafe in n steps within time T.
- H^δ can reach unsafe^δ in n steps within time T.

Note that these answers are not weaker than the precise ones. When safe is the answer, we know for certain that H does not reach the unsafe region; when δ-unsafe is the answer, there exists some δ-bounded perturbation in the system that *would* render it unsafe. Note that the error-bound δ can be chosen to be arbitrarily small, so that the δ-unsafe answer discovers robustness problem in the system, which should be regarded as unsafe indeed.