d/dt Manual

Thao Dang
VERIMAG, Centre Equation,
2, av de Vignate, 38610 GIERES, France
thao.dang@imag.fr

April 1, 2002

d/dt is a tool for reachability analysis of continuous and hybrid systems. The tool can also be used to solve the safety verification and controller synthesis problems for such systems. It is a C++ implementation of the algorithms presented in [2, 6]. We present first the main features of the tool and then a short user guide.

1 Overview

The current version of the tool **d/dt** accepts as input a hybrid automaton [1][1] in which

- *Continuous dynamics* are *linear* possibly with *uncertain*, *bounded input* of the form $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}$ where $\mathbf{u}$ ranges inside a convex polyhedron. The input $\mathbf{u}$ can model under-specified control or external disturbance.
- All the *staying conditions* ("invariants") and *transition guards* are specified as *convex polyhedra*.

Given a hybrid automaton and a safety property specified as a set of unsafe ("bad") states, the tool can solve the following problems:

1. Reachability: the tool performs forward reachability analysis from the initial set. The output is an over-approximation of the reachable set.
2. Safety Verification: using forward reachability analysis, the tool checks whether the system starting from the initial set can reach the bad set. The output is a yes/no answer accompanied by a set of bad states that the system has reached, in case the answer is yes.
3. Switching Controller Synthesis[2]: by computing an under-approximation of the maximal invariant set [3], the tool synthesizes a switching controller so that the system always

[1]Note that one can specify a purely continuous system as a hybrid automaton with only one location (or discrete state) and no transitions at all.
[2]The controller synthesis module will be included in the downloadable version soon.

remains inside the safe set. The output is the under-approximation of the maximal invariant set and the synthesized automaton.

2 Specifying Input Hybrid Automata

The input hybrid automaton and the specification are defined in a model file (with `.hyb` extension). Since the reachability algorithms of **d/dt** use polyhedral approximations, some computation parameters can be specified and fine-tuned by the user in order to find a good compromise between computation time and accuracy. If the model file is `example1.hyb`, the computation parameters should be defined in file `example1.par`. If such a file does not exist, default parameters will be used, but the approximation quality can be bad.

The grammar of the input language is simple and can be easily understood through an example (see details in Appendix A). The textual description of a 2-state hybrid automaton is shown below. Constant parameters and formulas referring to them can be used to describe the system, and comments are written between /* and */, as in C syntax.

```
dimension : 2; /* system dimension, i.e. the number of continuous variables */
constants : /* constant parameters in the system description */
a0   =   -0.32,
a1   =   -0.28;

initloc : 0; /* initial location */
initset : /* initial set can be an orthogonal, convex polyhedron, or a union of convex polyhedra */
  type griddy /* orthogonal polyhedron specified by a list of extreme vertices */
      -0.2   0.2, /* coordinates of an extreme vertex */
       0.2   0.2,
      -0.2   0.6,
       0.2   0.6;

/* The bad set is a list of symbolic states of the form (q, P) where q is a location and P is a */
/* polyhedon. Here the bad set is the union of (0, P1) and (1, P2) where */
/* P1 is the rectangle [0.62, 0.67] × [−0.1, 0.1], P2 is the halfspace x[0] ≤ x[1] */
badset :
  loc_id: 0
  type rectangle /* rectangle P1 specified by intervals */
     0.62    0.67,    /* 0.62 ≤ x[0] ≤ 0.67 */
     -0.1    0.1;     /* −0.1 ≤ x[1] ≤ 0.1 */
  or
  loc_id: 1
  type convex_constr /* convex polyhedron P2 specified by a set of constraints */
     1.0   -1.0   0.0;   /* x[0] − x[1] ≤ 0.0 */
```

/****************/
/* first location */
/****************/
location : 0; /* id of the location */
  /* dynamics $\dot{\mathbf{x}} = A_0\mathbf{x} + b_0\mathbf{u}$, $\mathbf{u} \in U_0$ */
  matrixA : /* matrix A_0 */
    0.0 $\quad$ -6.0,
    $[-a0/(1 + a0)]$ $\quad$ 0.0;
  scalB : 1.0; /* $b_0 = 1.0$ */
  inputset : /* U_0 is a triangle with vertices $(0.5, 0.5)$, $(0.5, 1.0)$, $(1.0, 0.5)$ */
   type convex_vert
     0.5 $\quad$ 0.5, /* vertex $(0.5, 0.5)$ */
     0.5 $\quad$ 1.0,
     1.0 $\quad$ 0.5;
  stayset : /* all staying sets must be convex polyhedra */
   type rectangle
     -0.15 $\quad$ 1.0,
     -1.0 $\quad$ 1.0;

  /* first outgoing transition */
  transition :
    label *0to1* : /* label */
    if in guard : /* all guard sets must be convex polyhedra */
    type rectangle
      $[-2.0/(a0 - a1)]$ $\quad$ $[-a0/(a0 - a1)]$,
      $[a0/(a0 - a1)]$ $\quad$ $[-1.0/(a0 - a1)]$;
    goto 1; /* id of the target location is 1 */

  /* second outgoing transition */
  transition :
    label *0to2* : /* label */
    if in guard : /* guard set */
    type convex_constr
      1.0 $\quad$ 2.0 $\quad$ -0.02, $\quad$ /* $x[0] + 2x[1] \leq -0.02$ */
      -1.5 $\quad$ 0.8 $\quad$ -1.5,
      -1.0 $\quad$ 1.0 $\quad$ 1.5;
    goto 2; /* id of the target location is 2 */

/*****************/
/* second location */
/*****************/
location : 1;

```
/* dynamics ẋ = A1x + b1u, u ∈ U1 */
matrixA : /* matrix A1 */
    -2.0  -3.0,
     3.0  -2.0;
 scalB : 0.8; /* b1 = 0.8 */
 inputset : /* set U1 */
   type convex_vert
    0.0  1.0; /* the input is constant: u = (0.0, 1.0), U1 is a singleton */
stayset : type rectangle
     -1.0  -0.02,
     -1.0   1.0;
transition :
   label 1to0 :
   if in guard : type convex_constr
      1.0   1.0  -0.05,
     -1.5   1.0  -1.5,
     -1.0   1.0   1.5;
   goto 0; /* id of the target location */

 ; /* semicolon */

/* the analysis set defined by a set of inequalities */
limits:
   x[0] >= -1.0  and
   x[0] <= 1.0   and
   x[1] >= -1.0  and
   x[1] <= 1.0
```

2.1 Specifying Polyhedral Sets

Polyhedral sets in the system description can be of the following three types.

- `Convex Polyhedra`: convex polyhedra can be specified using two formats: a set of vertices (`convex_vert`) and a set of constraints (`convex_constr`). In case a polyhedron is a hyper-rectangle, it can be defined simply by intervals.
- `Union of Convex Polyhedra`: a union of convex polyhedra can be specified by listing all the polyhedra separated by the keyword `OR` (see the bad set of the example above).
- `Orthogonal Polyhedra`: orthogonal polyhedra are specified by lists of extreme vertices following the keyword `griddy` (see Chapter 3 of [6], and more details can be found in [5]).

While all the staying and guard sets must be convex polyhedra, the initial set can be orthogonal. The analysis set (`limits`) can be specified either as a conjunction of inequalities, as in the example, or as a convex polyhedron. Note that no computation is performed outside the analysis set.

3 Specifying Computation Parameters

We first briefly explain the meaning of some important computation parameters that are used for error control purposes.

- `Time step`: the choice of the time step for each location depends on the desired accuracy and also on the matrices of the continuous dynamics (see Chapter 4 of [6]).

- `Bloating amount`: this parameter is used to ensure over-approximations for verification problems and under-approximations for synthesis problems. If its value is positive, over-approximations are produced, otherwise under-approximations. The bloating amount for a location depends on the time step chosen for the location and the matrix of its continuous dynamics. See [6] for the estimation of this parameter.

- `Mesh size`: orthogonal approximations are computed over a mesh, and the approximation error depends on the granularity of this underlying mesh. By reducing the mesh size one can achieve better approximations at the price of more computation time.

- `Integration error tolerances`: the desired integration accuracy is specified by a scalar relative tolerance `rel_tol` and an absolute tolerance vector `abs_tol`.

We now explain how to specify computation parameters. There are four types of computation parameters: parameters specific to each location, underdying mesh size, polyhedral error control parameters, and interface parameters. They are separated by a semicolon, and parameters of the same type can be given in any order. Note that if some parameters are not specified, default values will be used. However, the parameters related to error control, which are mentioned above, should be properly chosen in order to guarantee the desired accuracy.

```
dimension : 2; /* system dimension */

 /* Parameters specific to each location */
 location : 0, /* id of the location */
    time_step 0.2 / * time step */
    bloat 0.005 /* bloating amount */
    abs_tol 10 10 /* absolute integration error tolerance */
    rel_tol 6 /* relative integration error tolerance */
    grid_method lp /* griddization method, see explanation below */
    hull 0 /* do not use convex-hull approximation, see explanation below */
    itermax 300 /* search order parameter, see explanation below */
 ; /* semicolon */
```

```
/* Underlying mesh */
mesh_size 0.01 0.01 /* granularity of the underlying mesh */
; /* semicolon */

/* Polyhedral error control parameters */
dblmin 1E-20 /* user-defined DBL_MIN */
polylib_priority qhull /* or cdd */
cdd_zero 1E-12 /* "almost zero constant" for the library CDD */
; /* semicolon */

/* Interface and visualization parameters */
file_out oogl_format /* ddt_format, nosave; default value is nosave or 0 */
display geomview /* noview, or ddt_viewer */
verbose s /* s: print out a summary, i: interactive mode, q: quiet) */
projection 0 1 0 /* viewing projection axes */
zoom 1 1 0
view_angle 60 /* viewing angle of the 3D display window, in degree */
zmin −0.8 /* zmin,.., ymax are the coordinates of the display window */
zmax 0.8 /* if unspecified, the bounding box of the analysis set in the model file */
xmin −0.4
xmax 0.8
ymin −0.4
ymax 0.8
colour 2 /* integer ranging from 0 to 16 */
rotation 0 90 0 /* the default values are 0 0 0 */
translation −0.1 −0.1 −0.1
viewing_mode 1 /* 0 for Filled, 1 for Wireframe */
refresh 1 /* 1: clear the display window after each step; 0 by default */
height 900 /* height of the display window */
width 900 /* width of the display window */
```

3.0.1 Additional Options

- **Orthogonal approximation**: there are several "griddization" methods for orthogonal approximations implemented in **d/dt**. The parameter `grid_method` of a location specifies the method to be used for the continuous reachability computations at the location. The default choice is `bsp`. The stability and efficiency of these methods depend on the form of reachable sets. Note that orthogonal approximations are used because orthogonal polyhedra are in general easier to manipulate than lists of convex polyhedra. However, there are cases where treating reachable sets as lists of convex polyhedra is less expensive than as orthogonal polyhedra. The possible values of `grid_method` are:

  – `bsp`: binary-space-partition based method

- `lp`: linear programming based method
- `enu`: enumeration method
- `sim`: skip the orthogonal approximation step and store the reachable set as a list of convex polyhedra.

- **Convex-hull approximation**: in order to speed up the continuous reachability computation in a location, the user has an option to over-approximate the initial sets to explore by their convex hull by setting parameter `hull` to 1. This obviously reduces the approximation accuracy.

- **Search order**: the current version of **d/dt** performs breadth first search. If after `itermax` integration steps the computation of continuous-successors has not yet terminated, it will be delayed until next global iteration and other locations are then explored. If unspecified, this option is ignored.

- **Polyhedral error control parameters**:

  - `polylib_priority`: for convex polyhedral manipulation, **d/dt** combines two libraries QHULL [4] and CDD [7], which allows better precision control. The library CDD can handle degenerate cases such as non-full dimensional polyhedra but it is less time-efficient than QHULL. If the value of `polylib_priority` is `qhull`, the library QHULL will be used first and if it fails, the library CDD will then be used, and vice versa.
  - `cdd_zero`: "almost zero constant" that the library CDD uses to control error.
  - `precision`: user-defined DBL_MIN used by **d/dt** for various operations related to membership testing.

- **Visualization**: the visualization paramamenters in the example above define a perspective window. The angle of its field of view in direction y is 80 degrees. `(xmin, ymin, zmin)` define the coordinates of the bottom left corner of the display window. "projection 0 1 2" means that the object is projected on the first, second, and third coordinate axes. To visualize higher than three dimensional polyhedra, the coordinate axes in which the object will be displayed should be specified. In the above example, "`translation` -0.1 -0.1 -0.1" means that the object is rotated 0 degree about the axis x, 90 about y, and translated by $(-0.1, -0.1, -0.1)$. See the OpenGL documentation [9] for more details on 3D viewing.

- **Output**: if the parameter `file_out` is set to `ddt_format`, the analysis results will be saved in a file. The default output filename is the same as that the model file but with `.res` as extension. The results are stored in form of a sequence of (q, P) where P is the computed set at location q.

4 Running the tool

The command to run **d/dt** is:

```
ddt [options] [ModelName]
```

`ModelName` is the name of the model file (with `.hyb` extension, e.g. `example1.hyb`) which contains the description of the input hybrid automaton. As mentioned above, computation parameters will be read from the file `example1.par`.

The following options are available. Note that *the options in the command line override those defined in the parameter files.*

- `-p PbType` (type of the problem to solve)
  `PbType = reach`: the tool computes an over-approximation of the set of all states reachable from the initial set.
  `PbType = verif`: the tool checks whether the system satifies the safety property specified as a set of bad states. If no such sets are specified, only the computation of the reachable set is performed.

- `-r RepType` (data structure for orthogonal approximations)
  `RepType = uni`: orthogonal polyhedra are represented using matrices, and the underlying grid is uniform with fixed size.
  `RepType = grid`: orthogonal polyhedra are encoded by their extreme vertices, and the size of the underlying mesh is variable.
  If unspecified the option `griddy` will be used.

- `-v ViewMode`
  `ViewMode = noview`: no run-time visualization of the results.
  `ViewMode = geomview`: evoke GeomView[3] [8] to display the results (which are sent to GeomView via pipe communication). To use this option, GeomView must be priorily installed.
  `ViewMode = ddt_viewer`: use the viewer of **d/dt**. This viewer does not include advanced 3D features (in order to save computation time). For a better visualization quality, use `geomview` option.
  If unspecified, the viewer of **d/dt** will be used.

- `-o [ddt_format, oogl_format, nosave]`
  Write the analysis results to a file. If the model file name is `example1.hyb`, then the result file is `example1.res` in case `ddt_format` is chosen. As we shall see later in this section, you can then visualize data stored in example1.res using the View menu of the menu bar (see below). If `oogl_format` is chosen and the model has, for example, two locations, then the results will be written to 2 files: `example-0.list` and `example-1.list`, each of which contains the results for a location, saved in OOGL[4] format. These files can be then input to the 3D viewer GeomView.
  If unspecified, the results will not be saved, which is equivalent to `-o nosave` (this option can be used to override the output option given in the parameter file).

For example, when executing

[3]GeomView is a professional 3D viewer. It is freely available at http://www.geomview.org.
[4]Object Oriented Graphics Language

```
ddt -v geomview -o data/example1.hyb
```

d/dt performs reachability analysis on the system described in file `example1.hyb` in directory `data`. GeomView is evoked to visualize the reachable set, and once the computation terminates the results are saved in `data/example1.res`. As another example, when executing

```
ddt -p verif -v noview data/example1.hyb
```

in addition to the reachability computation, **d/dt** checks whether the system satisfies the safety property. No run-time visualization is used, and the results wiil not be saved.

A more interactive way of using **d/dt** is to evoke its menu bar by executing the command '`ddt`' (without options and `ModelName`). The menu bar consists of five sub-menus:

- `Input`: used to select the model file.
- `Run`: used to select the type of analysis (reachability, verification) and start the analysis.
- `Preferences`: During the analysis of a system, some computation parameters and options can be modified via the `Preferences` menu without changing the parameter file.
- `View`: this menu allows to display the results from the output files (`.res`). This is useful when the user might wish to skip run-time visualization in order to reduce computation time. The reachable set at each location will be displayed. Press the right button of the mouse in the display window to change the location to display.
- `OOGL-Save`: this menu is used to transform results stored in output files (`.res`) into OOGL format data, which can be then input to GeomView.

References

[1] 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*, LNCS 736, pages 209–229. Springer-Verlag, 1993.

[2] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 20–31. Springer-Verlag, 2000.

[3] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88:1011–1025, 2000.

[4] C.B. Barber, D.P. Dobkin, and H.T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[5] O. Bournez. *Complexité algorithmique des systèmes dynamiques continus et hybrides.* PhD thesis, École Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme, 1999.

[6] T. Dang. *Verification and Synthesis of Hybrid Systems.* PhD thesis, INPG, Verimag, 2000.

[7] K. Fukuda. *cdd/cdd++ manual.* Kluwer, 1988.

[8] M. Phillips, S. Levy, and T. Munzner. Geomview: An interactive geometry viewer. Notices of the American Mathematical Society, October 1993.

[9] C. Walnum. *3D Graphics Programming with OpenGL.* Que Corp, 1995.

A Input Language

A.1 Hybrid Automata

Formally, the input hybrid automata are described using the following grammar.

Table 3: Grammar for specifying input hybrid automata

HybAuto	::= dim ';' Constants ';' SpecDecl LocationList ';' Limits
dim	::= '`dimension :`' Integer ';'
SpecDecl	:= InitSet Safeset Badset Targetset
Constants	::= '`constants :`' ConstantList | None
ConstantList	::= ConstantList ',' ConstantDecl | ConstantDecl
ConstantDecl	::= Name '=' Double
InitSet	::= '`initloc :`' Integer ';' '`initset :`' PolyhedronList ';' | None
SafeSet	::= '`safeset :`' SymbolicState ';' | None
BadSet	::= '`badset :`' SymbolicState ';' | None
TargetSet	::= '`targetset :`' SymbolicState ';' | None
LocationList	::= LocationList LocationDecl | LocationDecl
LocationDecl	::= '`location :`' Integer ';' Dynamics StaySet OutTransitions
Dynamics	::= '`matrixA :`' Matrix ';' '`scalB :`' Double ';' InputSet
InputSet	::= '`stayset :`' Polyhedron ';' | None
StaySet	::= '`stayset :`' Polyhedron ';'

```
                 | None

OutTransitions  ::= 'transition :' TransitionList
                  | None

TransitionList  ::= TransitionList Transition
                  | None

Transition      ::= 'label' Name ': if in' Polyhedron 'goto' Integer ';'

Limits          ::= 'limits :' Polyhedron
                  | 'limits :' InequalityList

SymbolicState   ::= 'loc_id :' Integer ';' PolyhedronList

PolyhedronList  ::= PolyhedronList 'or' Polyhedron
                  | Polyhedron

Polyhedron      ::= PolyType Matrix
                  | None

PolyType        ::= 'convex_vert'
                  | 'convex_constr'
                  | 'rectangle'
                  | 'griddy'

InequalityList  ::= InequalityList 'and' Inequality
                  | Inequality

Inequality      ::= Variable InequalityType ArithmExpr
                  | None

InequalityType  ::= '<='
                  | '>='

Variable        ::= Name '[' Integer ']'

Matrix          ::= Matrix ',' Vector
                  | Vector

Vector          ::= Vector ArithmExpr
                  | ArithmExpr
```

ArithmExpr	::= Name	
	| Double	
	| ArithmExpr '+' ArithmExpr	
	| ArithmExpr '−' ArithmExpr	
	| ArithmExpr '∗' ArithmExpr	
	| ArithmExpr '/' ArithmExpr	
	| '-' ArithmExpr	
	| ArithmExpr 'pow' '('ArithmExpr')'	
	| '(' ArithmExpr ')'	
	| Fct '(' ArithmExpr ')'	
Fct	::= 'cos'	
	| 'sin'	
	| 'tan'	
	| 'acos'	
	| 'asin'	
	| 'atan'	
	| 'cotan'	
	| 'sqrt'	
Double	::= Integer('.'Integer)?Exposant?	
Exposant	::= [eE][+−]?Integer	
Name	::= Lettre(_ |(Lettre)—Number)*	
Integer	::= (Number)+	
Number	::= [0-9]	
Letter	::= [a-z] | [A-Z]	

A.2 Computation Parameters

Computation parameters are described using the following grammar.

Table 4: Grammar for specifying computation parameters

CompParams	::= dim ';' LocationParams ';' MeshSize ';' PolyCompParams ';' InterfaceParams
dim	::= 'dimension :' Integer ';'
LocationParams	:= LocationParamList

```
PolyCompParams      := PolyCompParamList
                       | None
InterfaceParams     := InterfaceParamList
                       | None

LocationParamList   ::= LocationParamList EachLocationParams
                        | EachLocationParams

EachLocationParams  ::= 'location :' Integer ',' EachLocParamList

EachLocParamList    ::= EachLocParamList ',' LocParamDecl
                        | LocParamDecl

PolyCompParamList   ::= PolyCompParamList ',' PolyCompParamDecl
                        | PolyCompParamDecl

InterfaceParamList  ::= InterfaceParamList ',' InterfaceParamDecl
                        | InterfaceParamDecl

LocParamDecl        ::= 'time_step' Double
                        | 'bloat' Double
                        | 'abs_tol' Double
                        | 'rel_tol' Vector
                        | 'hull' Boolean
                        | 'itermax' Integer
MeshSize            ::= 'mesh_size' Double

PolyCompParamDecl   ::= 'dblmin' Double
                        | 'polylib_priority' PolyLibName
                        | 'cdd_zero' Double

InterfaceParamDecl  ::= 'file_out' OutputType
                        | 'display' DisplayType
                        | 'verbose' VerboseType
                        | 'projection' Integer Integer Integer
                        | 'view_angle' Double
                        | 'xmin' Double                 | 'ymin' Double     | 'zmin' Double
                        | 'xmax' Double
                        | 'ymax' Double
                        | 'zmax' Double
                        | 'colour' Integer
                        | 'rotation' Double Double Double
                        | 'viewing_mode' Integer
```

| 'refresh' Boolean
| 'height' Double
| 'width' Double

Double	::= {Integer('.'{Integer})?{Exposant}?
Exposant	::= [eE][+-]?{Integer}
Integer	::= {Number}+
Boolean	::= '1' — '0'
Number	::= [0-9]
PolyLibName	::= 'qhull' | 'cdd'
VerboseType	::= 'i' | 's' | 'd'
DisplayType	::= 'noview' | 'ddt_viewer' | 'geomview'
OutputType	::= 'nosave' | 'ddt_format' | 'oogl_format'