δ-Complete Reachability Analysis for Nonlinear Hybrid Systems

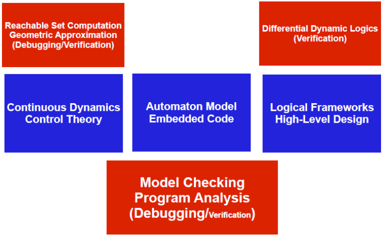
Sicun (Sean) Gao

Carnegie Mellon University

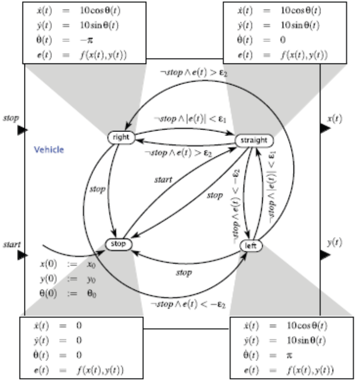
CMACS PI Meeting
4-27-2012

Joint work with Edmund Clarke and Jeremy Avigad

- Cyber-physical Systems: Combine physics and information-processing.
- Different verification techniques are suitable for different tasks.
  - Differential Logics: Verify the logical frameworks.
  - Reachable Set Computation: Visualize complete dynamics.
  - Model Checking: Return counterexamples.



- Simplified Controller of an automated guided vehicle [Lee and Seshia, 2011]



$\mathcal{H} = \langle X, Q, \mathsf{Init}, \mathsf{Flow}, \mathsf{Jump} \rangle$

- A state space $X \subseteq \mathbb{R}^k$ and a finite set of modes Q.
- Init $\subseteq Q \times X$: initial configurations
- Flow $:\subseteq Q \times X \to TX$: continuous flows
  - Each mode q is equipped with differential equations $\dfrac{d\vec{x}}{dt} = \vec{f}_q(\vec{x}, t)$.
- Jump $:\subseteq Q \times X \to 2^{Q \times X}$: discrete jumps
  - The system can be switched from $(q, \vec{x})$ to (maybe multiple) $(q', \vec{x}')$, resetting modes and variables.

Continuous flows are interleaved with discrete jumps.

Bounded Model Checking

The idea of bounded model checking is the following:

- The behavior of a transition system can be encoded as logic formulas.
    - $\mathsf{Init}(\vec{x}_0) \wedge \bigwedge_{i=0}^{k}(\mathsf{Transition}(\vec{x}_i, \vec{x}_{i+1})) \wedge \mathsf{Unsafe}(\vec{x}_{k+1})?$
- Very fast (SAT/SMT) solvers can be used for *deciding* such formulas (say yes or no).
- The usual point of view is that it only works for discrete systems.
    - Extremely successful in the hardware design domain.

- Continuous Dynamics: $\dfrac{d\vec{x}(t)}{dt} = \vec{f}(\vec{x}(t), t)$
  - The solution curve:
    $\alpha : \mathbb{R} \to X,\ \alpha(t) = \alpha(0) + \int_0^t \vec{f}(\alpha(s), s)ds.$
  - Define the predicate (probably no analytic forms)
    $[\![\mathsf{Flow}_f(\vec{x}_0, t, \vec{x})]\!]^{\mathcal{M}} = \{(\vec{x}_0, t, \vec{x}) : \alpha(0) = \vec{x}_0, \alpha(t) = \vec{x}\}$

Reachability:

$$\exists \vec{x}_0, \vec{x}, t.\ (\mathsf{Init}(\vec{x}_0) \wedge \mathsf{Flow}_f(\vec{x}_0, t, \vec{x}) \wedge \mathsf{Unsafe}(\vec{x}))\ ?$$

Encoding Bounded Reachability for Hybrid Systems

For hybrid systems we combine continuous and discrete behaviors:

- "$\vec{x}$ is reachable after after 0 discrete jumps":

$$\text{Reach}^0(\vec{x}) := \quad \exists \vec{x}_0, t.\ [\text{Init}(\vec{x}_0) \wedge \text{Flow}(\vec{x}_0, t, \vec{x})]$$

- Inductively, "$\vec{x}$ is reachable after $k+1$ discrete jumps":

$$\text{Reach}^{k+1}(\vec{x}) := \exists \vec{x}_k, \vec{x}'_k, t.\ [\text{Reach}^k(\vec{x}_k) \wedge \text{Jump}(\vec{x}_k, \vec{x}'_k) \wedge \text{Flow}(\vec{x}'_k, t, \vec{x})]$$

(Some details are omitted.)

Reachability within n discrete jumps:

$$\exists \vec{x}.\ (\bigvee_{i=0}^{n} \text{Reach}^i(\vec{x}) \wedge \text{Unsafe}(\vec{x}))\ ?$$

- The Flow and Jump predicates in the formulas require a rich signature of nonlinear functions.
  - polynomials
  - exponentiation and trigonometric functions
  - solutions of ODEs, mostly no analytic forms
- To handle naive unrolling, the arithmetic theory is way less than enough.
  - In realistic systems at least some linear dynamical systems occur.
  - Various techniques have been developed to encode interesting behaviors in arithmetic; but of course not a large part of them. (We will discuss invariant-based reasoning later.)

We are all aware of difficulties of symbolic decision procedures over reals when nonlinear functions are involved.

- For the theory of nonlinear arithmetic:
  - Double-exponential lower bound for quantifier elimination (PSPACE for Σ_1).
  - Very active research in the past thirty years.
  - Available solvers: challenged by formulas with ten variables.
- The general first-order theory over exp, sin, ODEs, ...
  - Wildly undecidable.
  - Is formal verification impossible because of this?

Scaling up: Use Numerical Methods?

However, large systems of real equalities/inequalities/ODEs are **numerically** solved routinely in scientific computing.

- They are usually regarded inappropriate for verification because of the inevitable numerical errors.
  - (Platzer and Clarke, HSCC 2008)
- But isn't there any way to use them?

Let's start by formalizing "numerical algorithms".

What does it mean to say a function f over reals is "numerically computable"?

- There exists an algorithm M_f, such that given a good approximation of x, M_f can find a good approximation of $f(x)$.
  - "A real function is computable if we can draw it faithfully."
- This leads to Computable Analysis (a.k.a. Type-II Computability) over real numbers. [A. Turing, A. Grzegorczyk, K. Weihrauch, S. Cook]

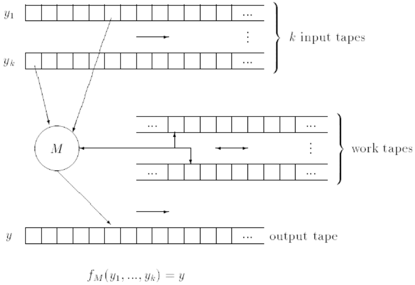
- Any real number a is encodable by a name $\gamma_a : \mathbb{N} \to \mathbb{Q}$ satisfying

$$\forall i, \ |a - \gamma_a(i)| < 2^{-i}$$

- A Type-II Turing machine extends the ordinary by allowing input and output tapes to be both infinite. The working tape remains finite.
- Although output tape is infinite, each symbol needs to be written down after finitely many operations.

- A function f is Type-II computable, if there exists a Type-II Turing machine $\mathcal{M}_f$, s.t.:

  Given any $\gamma_{\vec{x}}$ of $\vec{x} \in dom(f)$, $\mathcal{M}_f$ outputs a $\gamma_{f(\vec{x})}$ of $f(\vec{x})$.



Example

- e^x is Type-II computable over $[-1, 1]$.
  - Suppose we want to compute e^x at some $x \in [-1, 1]$ with an error bound 2^{-n} on the output. Since

    $$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

    We only need to expand the series to $n+1$ terms, and the error is controlled by

    $$(e^x - \sum_{k=0}^{n} \frac{x^k}{k!}) \leq \sum_{k=n+1}^{\infty} \frac{1}{k!} < 2^{-(n+1)}.$$

  - We then use a 2^{-m} rational approximation of x to evaluate the truncated series, where $m \leq n+4$.
- It is computable because the number of terms, $n+1$, is computed from the error bound 2^{-n}, and the truncated series is a computable function in the usual sense over rational representation of x.

First-order Formulas with Computable Functions

- Let $\mathcal{F}$ be any recursive set of Type-II computable functions.
  - This is a very general framework: $\mathcal{F}$ can contain polynomials, $\exp$, $\sin$, and solutions of Lipschitz-continuous ODEs.
- Consider $\mathbb{R}_{\mathcal{F}} = \langle \mathbb{R}, 0, 1, \mathcal{F}, < \rangle$ and the corresponding $\mathcal{L}_{\mathcal{F}}$.
- **Can we solve (decide the truth value of) logic formulas in $\mathcal{L}_{\mathcal{F}}$ over $\mathbb{R}_{\mathcal{F}}$?**
  - This would allow us to solve formulas that arise in bounded model checking of hybrid systems, *almost in its full generality*.
  - The obvious answer is of course NO.

But what if we take into account the numerical computability of $\mathcal{F}$?

Suppose we want to decide a formula in $\mathcal{L}_{\mathcal{F}}$:

$$\exists^I x.(f(x) = 0 \wedge g(x) = 0).$$

($I \subseteq \mathbb{R}$ is a bounded interval where f and g are defined).

- Numerical algorithms can never compute $f(x)$ and $g(x)$ precisely for all x.
- But Type-II computability implies that it is possible to fix any error bound δ, and numerically decide the relaxed formula:

$$\exists^I x.(|f(x)| < \delta \wedge |g(x)| < \delta)$$

Consequently, we could consider formulas whose satisfiability is invariant under numerical perturbations.

- Consider any formula $\varphi := \bigwedge_i (\bigvee_j f_{ij}(\vec{x}) = 0)$.
  - Inequalities are turned into interval bounds on slack variables.
- A δ-perturbation on φ is a constant vector $\vec{c}$ satisfying $||\vec{c}||_\infty < \delta$, and a δ-perturbed φ is:

$$\varphi^{\vec{c}} := \bigwedge_i (\bigvee_j |f_{ij}(\vec{x})| = c_{ij})$$

δ-Robustness

- We say satisfiability of φ is δ-robust (over some bounded $\vec{I}$), if:

  For any δ-perturbation $\vec{c}$, $\exists^{\vec{I}}\vec{x}.\varphi \leftrightarrow \exists^{\vec{I}}x.\varphi^{\vec{c}}$.

- Observations:
  - If robust for bigger δ, then robust for smaller ones.
  - Strict and non-strict inequalities are inter-changeable in robust formulas. (But negations can still be encoded.)

As it turns out, robust formulas in $\mathcal{L}_{\mathcal{F}}$ have nice computational properties.

- Theorem:

  Satisfiability of robust bounded first-order over $\mathbb{R}_{\mathcal{F}}$ is decidable.

- This is significant given the richness of $\mathcal{F}$: exp, sin, ODEs, ...

- **Theorem:**

  Suppose all the functions in $\mathcal{F}$ are in Type-II complexity class $\mathbf{C}$, then satisfiability of bounded SMT in $\mathcal{L}_{\mathcal{F}}$ can be decided in $\mathbf{NP}^{\mathbf{C}}$.

- Corollaries:
  - $\mathcal{F} = \{+, \times, \exp, \sin\}$: $\mathbf{NP}$-complete.
  - $\mathcal{F} = \{$Lipschitz-continuous ODEs$\}$: $\mathbf{PSPACE}$-complete.

- **Theorem:** There exists decision algorithms that, on any φ in $\mathcal{L}_{\mathcal{F}}$, returns "sat/unsat" satisfying:
  - If φ is decided as "unsat", then it is indeed unsatisfiable.
  - If φ is decided as "sat", then:

    Under some δ-perturbation $\vec{c}$, $\varphi^{\vec{c}}$ is satisfiable.

- If a decision procedure satisfies this property, we say it is δ-complete.

Recall that when bounded model checking a hybrid system $\mathcal{H}$, we ask if

$$\varphi : \mathsf{Reach}^{\leq n}_{\mathcal{H}}(\vec{x}) \wedge \mathsf{Unsafe}(\vec{x})$$

is satisfiable.

- If φ is unsatisfiable, then $\mathcal{H}$ is safe up to depth n.
- If φ is satisfiable, then $\mathcal{H}$ is unsafe.

Delta-Complete BMC

Consequently, using a δ-complete decision procedure we can guarantee:

- If φ is "unsatisfiable", then $\mathcal{H}$ is safe up to depth n.
  - It is possible to make even stronger claims, that it is safe up to n under any δ'-perturbation, where $\delta' < \delta$ is also specified by the user. In this case we say it is (δ, δ')-complete.
- If φ is "satisfiable", then

$\mathcal{H}$ is unsafe under some δ-perturbation.

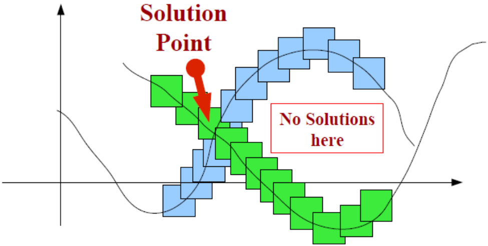
Consequently, if a system can become unsafe under some δ-perturbation, we will be able to detect such unsafety.

- **This can not be achieved using precise algorithms.**

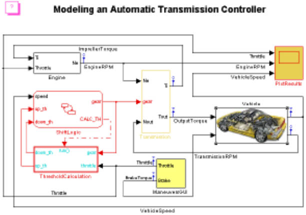
Practical δ-Complete Decision Procedures

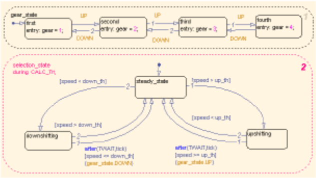
- We have shown a general framework for deciding logic formulas in a rich theory over reals, and their applicability in verification problems.
- No restriction on use of specific numerical algorithms.
  - Interval Constraint Propagation, Semi-definite Programming, Convex Optimization, CORDIC, Boundary-Value Solvers for ODEs, ...
- The obligation is to prove δ-completeness (rather than using them just as heuristics).

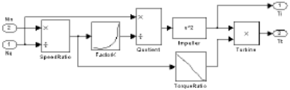
- Interval Arithmetic + Constraint Programming.
  - Starting from initial intervals on all variables, maintain an over-approximation of the constraints using interval arithmetic. (Use floating point arithmetic, outward-rounding.)
  - Reduce (contraction+splitting) the size of intervals until some limit is reached (say, 10^{-7}). Return "unsat" if conflicts arise in the process (i.e., intervals on the same variable become disjoint).



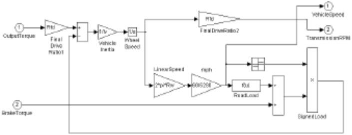
- Example:
  - Solve $\{x = y, x^2 = y\}$ for $x \in [1,4]$ and $y \in [1,5]$:
  - $I^x : [1,4] \to [1,\sqrt{5}] \to [1,\sqrt[4]{5}] \to [1,\sqrt[8]{5}] \to [1,\sqrt[16]{5}] \to \cdots \to [\mathbf{1},\mathbf{1}]$
  - $I^y : [1,5] \to [1,\sqrt{5}] \to [1,\sqrt[4]{5}] \to [1,\sqrt[8]{5}] \to [1,\sqrt[16]{5}] \to \cdots \to [\mathbf{1},\mathbf{1}]$
- Simple algorithm, but can solve large systems of nonlinear constraints.
  - Many papers report solving constraints with thousands of variables (robotics, planning, etc.).
  - HySAT and [Gao et al. FMCAD2010] can solve many interesting benchmarks.
- **DPLL(ICP) is δ-complete.**



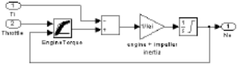




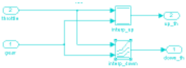
Torque Converter



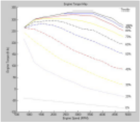
Vehicle



Engine



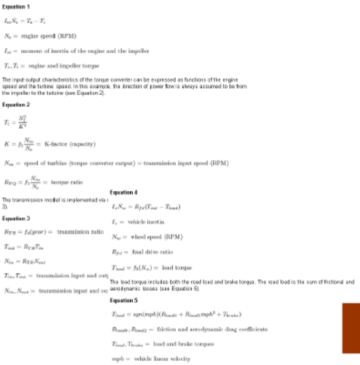
Treshold Calculation



The Automatic Transmission Model

- Four main control locations (four gears) plus six transition modes.

Equations in the model



Equation 1

$I_{ei}\dot{N}_e = T_e - T_i$

$N_e =$ engine speed (RPM)

$I_{ei} =$ moment of inertia of the engine and the impeller

$T_e, T_i =$ engine and impeller torque

The input-output characteristics of the torque converter can be expressed as functions of the engine speed and the turbine speed. In this example, the direction of power flow is always assumed to be from the impeller to the turbine (see Equation 2).

Equation 2

$T_i = \frac{N_e^2}{K^2}$

$K = f_2 \frac{N_{in}}{N_e} =$ K-factor (capacity)

$N_{in} =$ speed of turbine (torque converter output) = transmission input speed (RPM)

$R_{TQ} = f_3 \frac{N_{in}}{N_e} =$ torque ratio

The transmission model is implemented via ...
3).

Equation 3

$R_{TR} = f_4(gear) =$ transmission ratio

$T_{out} = R_{TR} T_{in}$

$N_{in} = R_{TR} N_{out}$

$T_{in}, T_{out} =$ transmission input and out...

$N_{in}, N_{out} =$ transmission input and ou...

Equation 4

$I_v \dot{N}_w = R_{fd}(T_{out} - T_{load})$

$I_v =$ vehicle inertia

$N_w =$ wheel speed (RPM)

$R_{fd} =$ final drive ratio

$T_{load} = f_5(N_w) =$ load torque

The load torque includes both the road load and brake torque. The road load is the sum of frictional and aerodynamic losses (see Equation 5).

Equation 5

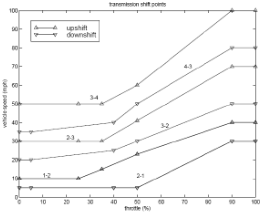
$T_{load} = sgn(mph)(R_{load0} + R_{load2} mph^2 + T_{brake})$

$R_{load0}, R_{load2} =$ friction and aerodynamic drag coefficients

$T_{load}, T_{brake} =$ load and brake torques

$mph =$ vehicle linear velocity

Discrete Transitions



Nonlinear Differential Algebraic Equations

The Automatic Transmission Model [Absmeier 2001; Runde 1984]

- Equations in the first gear:

$$
\begin{aligned}
I_{t1}\dot{\omega} &= T_t - R_1 R_d T_s \\
I_{t1} &= I_t + I_{si} + R_1^2 I_{cr} + \frac{R_1^2}{R_2^2} I_{ci} \\
RT_{12B} &= \frac{R_{sr}}{R_{ci}}\left(\frac{T_t - (I_t + I_{si})\dot{\omega}_t}{R_{si}} - I_{ci}\dot{\omega}_{ci} + (1 - \frac{1}{R_{si}})\right)
\end{aligned}
$$

- 1-2-1 Shift, Torque Phase:

$$
\begin{aligned}
I_{t1}\dot{\omega} &= T_t - R_1 R_d T_s - (1 - \frac{R_1}{R_2})T_{c2} \\
RT_{12B} &= \frac{R_{sr}}{R_{ci}}(T_{c2} - \frac{I_{si}}{R_{si}}\dot{\omega}_{si} - I_{ci}\dot{\omega}_{ci})
\end{aligned}
$$

The Automatic Transmission Model [Absmeier 2001; Runde 1984]

- Second Gear:

$$
\begin{aligned}
I_{t2}\dot{\omega}_t &= T_t - R_2 R_d T_s \\
RT_{c2up} &= T_t - T_{c1} - I_t\dot{\omega}_t \\
RT_{c2down} &= \frac{I_{ci12}}{R_2}\dot{\omega}_{cr} - \frac{R_2}{R_1}T_{c1} + R_2 R_d T_s \\
RT_{12B} &= \frac{R_{sr}}{R_{ci}}(T_t - \frac{I_{si}}{R_{si}}\dot{\omega}_{si} - (I_t + I_{ci})\dot{\omega}_t) \\
I_{t2} &= I_t + I_{ci} + R_2^2 I_{cr} + \frac{R_2^2}{R_1^2} I_{si}
\end{aligned}
$$

- Third Gear:

$$
\begin{aligned}
I_{t2}\dot{\omega}_t &= T_t - R_2 R_d T_s \\
RT_{c2up} &= T_t - T_{c1} - I_t\dot{\omega}_t \\
RT_{c2down} &= \frac{I_{ci12}}{R_2}\dot{\omega}_{cr} - \frac{R_2}{R_1}T_{c1} + R_2 R_d T_s \\
&\ldots
\end{aligned}
$$

- 2-3-2 Shift, Torque phase:

$$
\begin{aligned}
I_{t2}\dot{\omega}_t &= T_t + (1 - \frac{R_2}{R_1})T_{c3} - R_2 R_d T_s \\
RT_{c2up} &= T_t + T_{c3} - I_t\dot{\omega}_t \\
RT_{c2down} &= R_2 I_{cr}\dot{\omega}_{cr} + I_{ci}\omega_{ci} + \frac{R_2}{R_1} I_{si}\dot{\omega}_{si} + R_1 R_d T_s + \frac{R_2}{R_1} T_{c3} \\
RT_{12B} &= \frac{R_{sr}}{R_2}(T_t + \frac{R_{cr}}{R_{si}} T_{c3} - (I_t + I_{ci})\dot{\omega}_t - \frac{I_{si}}{R_{si}}\dot{\omega}_{si})
\end{aligned}
$$

- 2-3-2 Shift, Inertia Phase:

$$
\begin{aligned}
I_{t23}\dot{\omega}_t &= T_t + (1 - \frac{1}{R_{si}} T_{c3} - \frac{R_{ci}}{R_{sr}} T_{12B} + I_{s23}\dot{\omega}_{cr}) \\
I_{cr23}\dot{\omega}_{cr} &= \frac{T_{12B}}{R_{sr}} - (1 - \frac{1}{R_{si}})T_{c3} - R_d T_s + I_{s23}\dot{\omega}_t \\
RT_{c2up} &= T_t + T_{c3} - I_t\dot{\omega}_t \\
RT_{c2down} &= R_2 I_{cr}\dot{\omega}_{cr} + I_{ci}\dot{\omega}_{ci} + \frac{R_2}{R_1} I_{si}\dot{\omega}_{si} + R_2 R_d T_s - \frac{R_2}{R_1} T_{c3}
\end{aligned}
$$

Results from dReal

- Reachability Question: $t_1 < 15 \wedge t_2 < 50 \wedge \omega(t_1) > 50 \wedge \omega(t_2) = 0$? (Can the vehicle reach a certain speed and decelerate within a certain time bound?)
- Answer: Yes, with sample trace returned. (Solved in 4.5s)

```
//Simulink Example: Auto Transmission Controller

//the variables have the same name as in the model description.
[0, 1000] R_load0, R_load2, I_v, N_w, R_fd, I_ei;
[-1000, 1000] T_brake, T_load, T_out, T_load, T_in, R_TR, N_in, N_out, K, f_2, f_3, T_e, T_i;
[-200, 200] V;
[0, 1000] t;

%%

{ //1st gear, steady state
mode: 1;

dynamics: //the following equations are taken from Eq 1-5 in model description

d/dt(N_e') = (T_e - T_i)/I_ei;
d/dt(N_w') = R_fd * (T_out - T_load)/I_v ;
T_load = sgn(V)*(R_load0 + R_load2 * V^2 + T_brake) ;
T_out = R_TR * T_in ;
N_in = R_TR * N_out ;
T_i = N_e^2 / K^2 ;
K = f_2 * N_in / N_e ;
R_TQ = f_3 * N_in / N_e ;

//this constant is given in Table 1 in the model description. More constants can be set here.
R_TR = 2.393 ;

jump:
//if the vehicle speed is over up_th, and after TWAIT ticks it is still over up_th, then upshift
V[t] > up_th & V[t+TWAIT] >= up_th ==> @2 *;
//if after TWAIT it becomes lower than up_th, it returns to the current gear
V[t] > up_th & V[t+TWAIT] < up_th ==> @1 *;
}
```

```
jump:

V[t] < down_th & V[t+TWAIT] <= down_th ==> @3 true ;
V[t] < down_th & V[t+TWAIT] > down_th  ==> @4 true ;

}
%%

goal:

V[10]>50 and V[50]<5;
```

```
mode list:
    @0: 1        @1: 1        @2: 2
    @3: 3        @4: 4        @5: 3
    @6: 2        @7: 1

V (real):
    @0: [0, 0]
    @1: [0, 0] to [25.753121131, 25.753132445]
    @2: [25.028642881, 25.103735219] to [45.202938101, 45.223122312]
    @3: [45.202938101, 45.223122312] to [59.328301923, 59.369321301]
    @4: [59.328301923, 59.369321301] to [41.631938123, 41.731138142]
    @5: [41.631938123, 41.731138142] to [21.129301217, 21.130301125]
    @6: [21.129301217, 21.130301125] to [5.471248102, 5.471248019]
    @7: [5.471248102, 5.471248019] to [0.000000000, 0.000000000]

t (real):
    @0: [0, 0]
    @1: [3.068125077, 3.112125121]
    @2: [2.772474941, 2.804660616]
    @3: [5.045497657, 5.412499212]
    @4: [22.274214810, 22.497145603]
    @5: [11.928315213, 11.931316811]
    @6: [6.389129301, 6.391943582]
    @7: [10.541821217, 10.570492172]
```

- Model checking can be used in the context of nonlinear continuous and hybrid systems.
- Our technique relies on recent progress in the underlying decision procedures, combining fast SAT solvers with numerical algorithms.
- We have developed the theory for ensuring the reliablility of such combination.
- Our tool is under active development and will be available soon.
- We are developing tools for using our solver in the context of program analysis of embedded code.