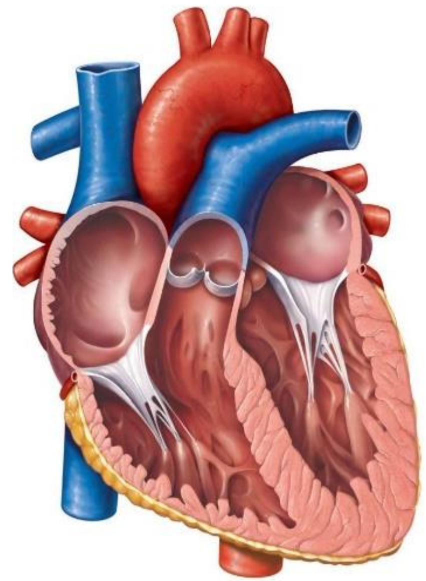
C2E2: A Verification Tool For Stateflow Models

<u>Parasara Sridhar Duggirala,</u>
Sayan Mitra,
Mahesh Viswanathan,
Matthew Potok

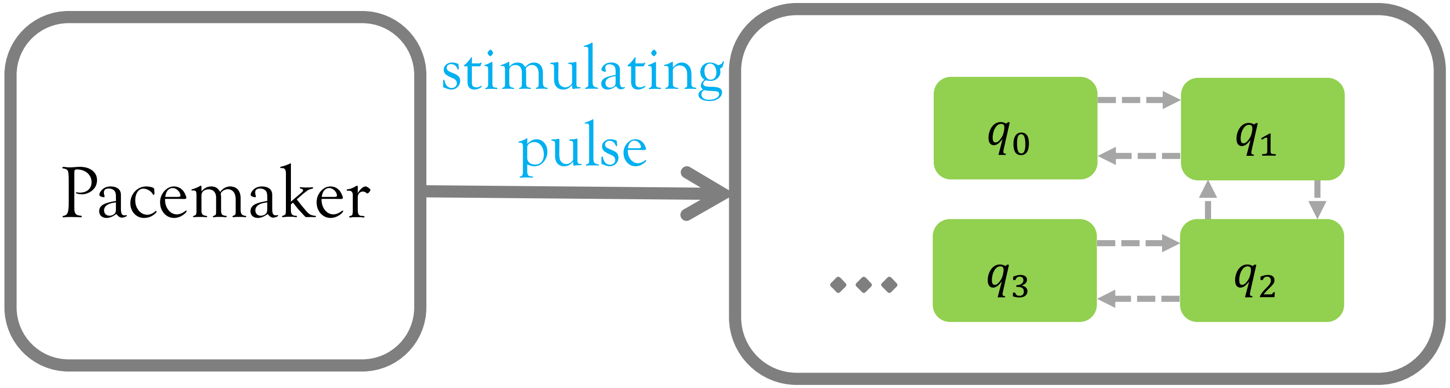
ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Pacemaker – Cardiac Cell System

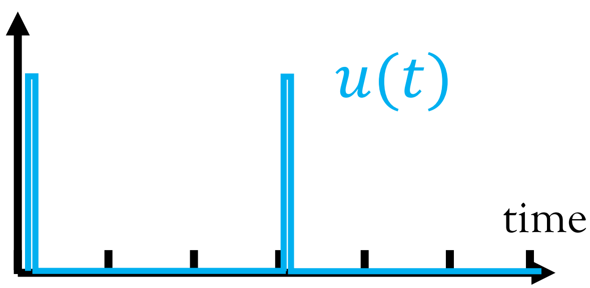
+



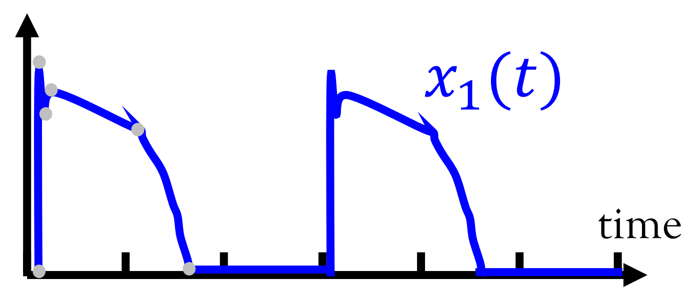
Pacemaker – Cardiac Cell System



HA = Finite State Machine + Differential Equation



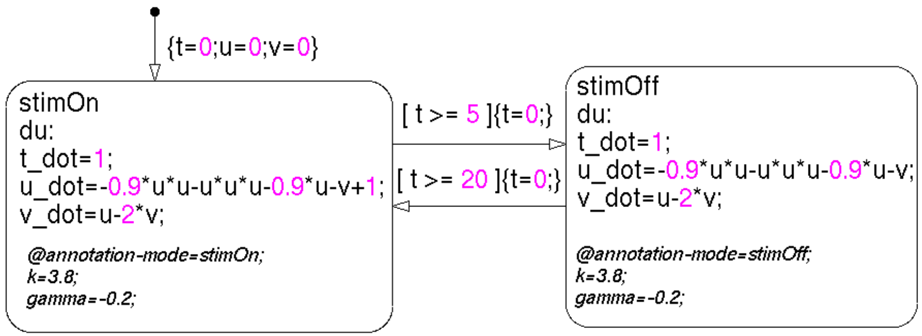
Stimulus from pacemaker

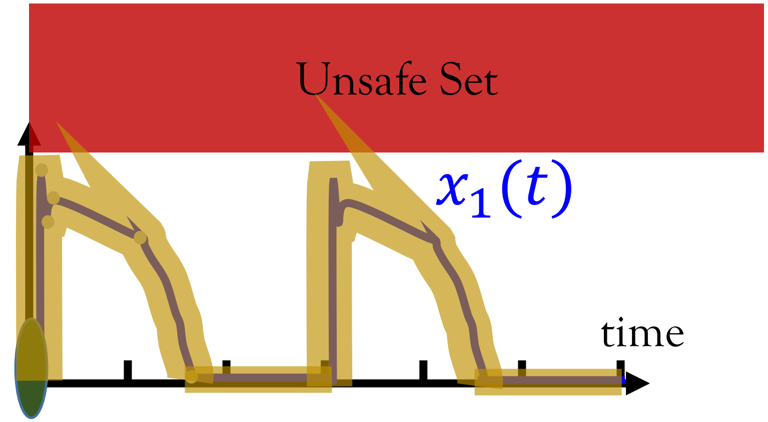


Behavior of a cardiac cell

Features of the Model

Safety Verification





Stateflow Model of Pacemaker – Cardiac Cell system

Features: **Invariants**, **Guards**, and **Resets**

- Inputs:
  1. Model of the system A,
  2. Initial States Θ, and
  3. Unsafe States U
- Output: If the system is safe or unsafe

  $\forall x \in \Theta, \xi(x,t) \notin U$

Solution

Reachable Set Computation

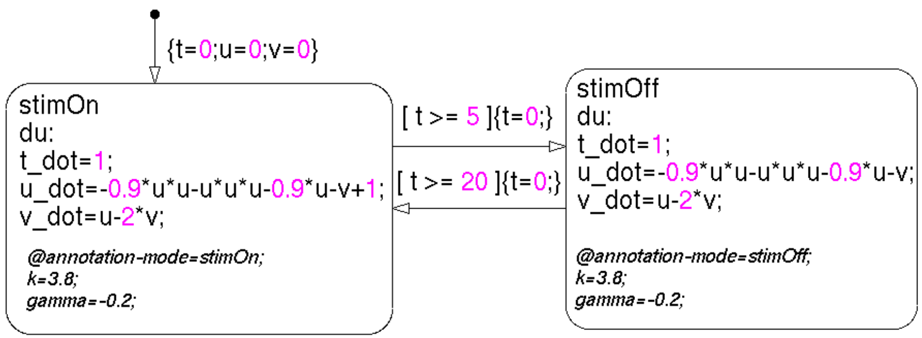
Contributions

- Simulation based verification algorithm for *Fully Hybrid Systems*
- Theoretical guarantees – *Soundness and Relative Completeness*
- Tool Features
  - Stateflow Models, *hyxml* intermediate format
  - Graphical User Interface
  - Visualizing the reachable set

Overview

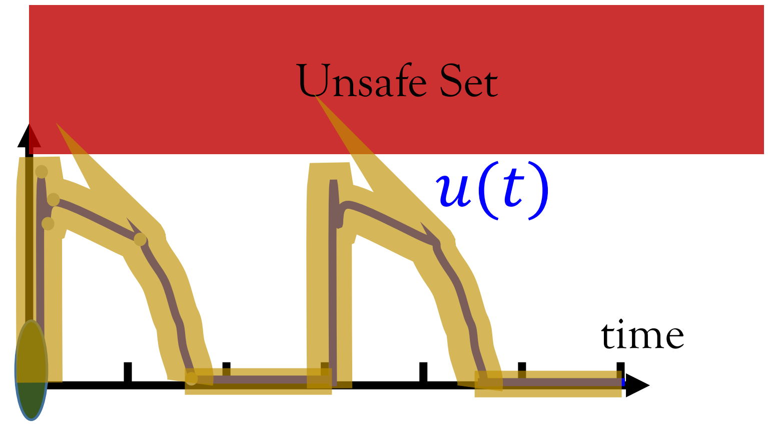
- ✓ Motivation and Problem Statement
- Challenges in Verification
- Building Blocks and Algorithm
- Soundness and Relative Completeness Guarantees
- Tool Features
- Annotations
- Future Work

Safety Verification



Stateflow Model of Pacemaker – Cardiac Cell system

<u>Features</u>: **Invariants**, **Guards**, and **Resets**



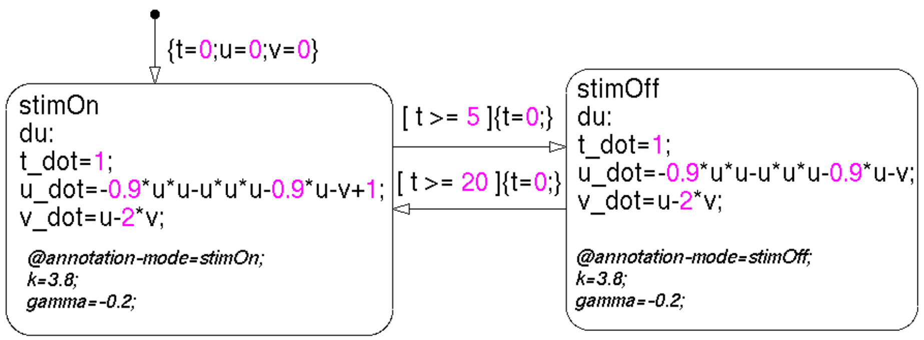
- Inputs:
  1. Model of the system A,
  2. Initial States Θ, and
  3. Unsafe States U
- Output: If the system is safe or unsafe

  $\forall x \in \Theta, \xi(x,t) \notin U$

<u>Solution</u>

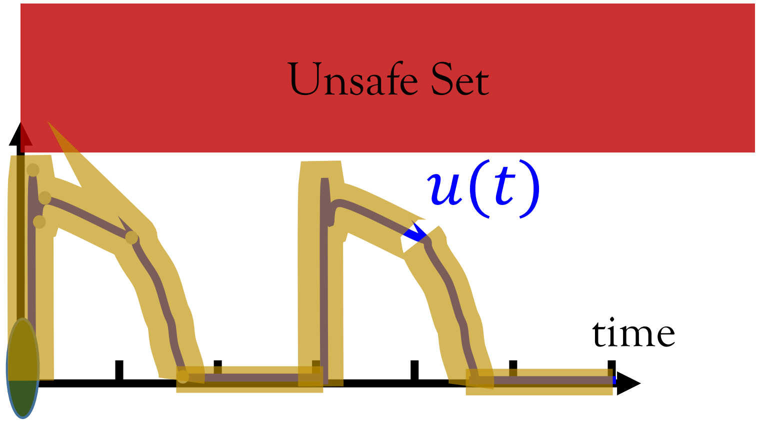
Reachable Set Computation

Challenges In Reachable Set Computation



Stateflow Model of Pacemaker – Cardiac Cell system

<u>Features</u>: **Invariants**, **Guards**, and **Resets**

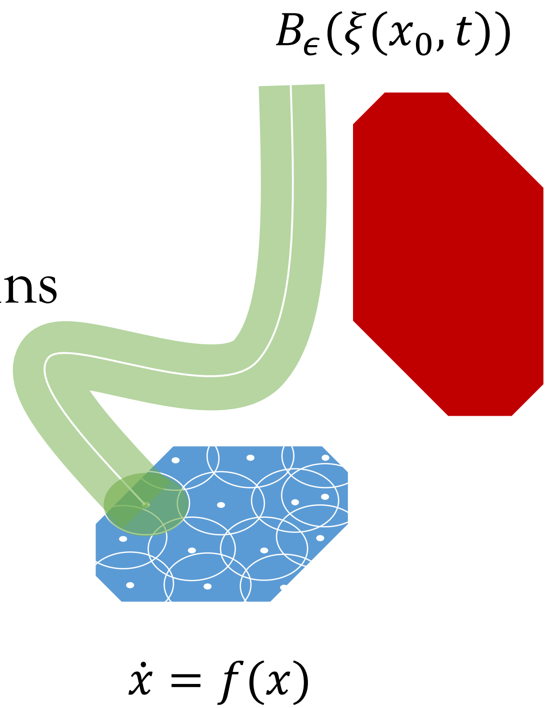


- Nonlinear ODEs – do not even have a closed form solution
- Switching conditions – predicates on variables (nondeterminism)

Our Technique: Use simulations for computing *Reachable Set*

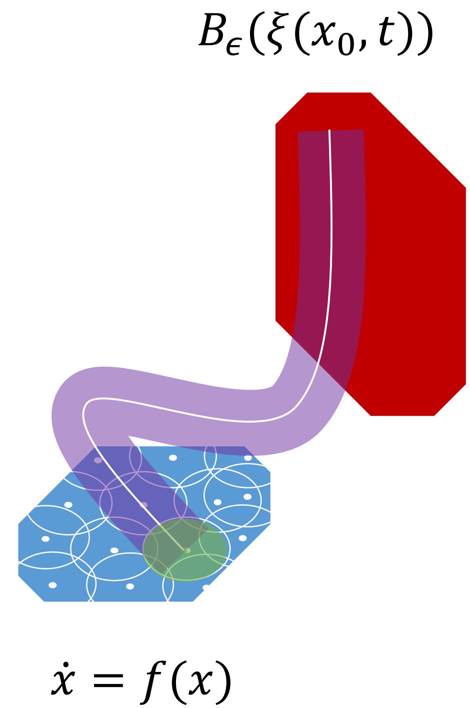
A Simple (Often The Only) Strategy

- Given start Θ and unsafe U
- Compute finite cover of initial set
- Simulate from the center x_0 of each cover
- **Bloat** simulation so that bloated tube contains trajectories from the cover
- Union = over-approximation of reach set



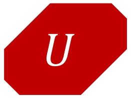
A Simple (Often The Only) Strategy

- Given start Θ and unsafe U
- Compute finite cover of initial set
- Simulate from the center x_0 of each cover
- Bloat simulation so that bloated tube contains trajectories from the cover
- Union = over-approximation of reach set
- Check intersection/containment with U
- Refine



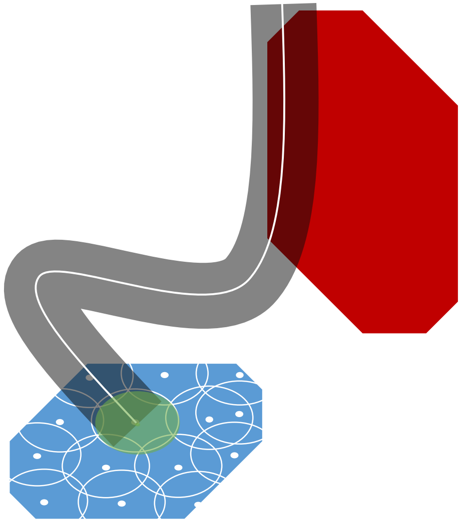
A Simple (Often The Only) Strategy

- Given start 

and unsafe 

- Compute finite cover of initial set
- Simulate from the center x_0 of each cover
- Bloat simulation so that bloated tube contains trajectories from the cover
- Union = over-approximation of reach set
- Check intersection/containment with U
- Refine

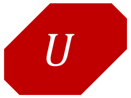
$B_\epsilon(\xi(x_0, t))$



$\dot{x} = f(x)$

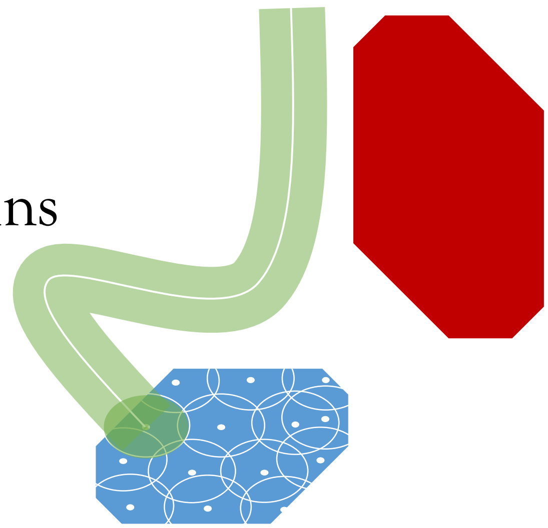
A Simple (Often The Only) Strategy

- Given start 

 and unsafe 

- Compute finite cover of initial set
- Simulate from the center x_0 of each cover
- **Bloat** simulation so that bloated tube contains trajectories from the cover
- Union = over-approximation of reach set
- Check intersection/containment with U
- Refine

$B_\epsilon(\xi(x_0, t))$



$\dot{x} = f(x)$

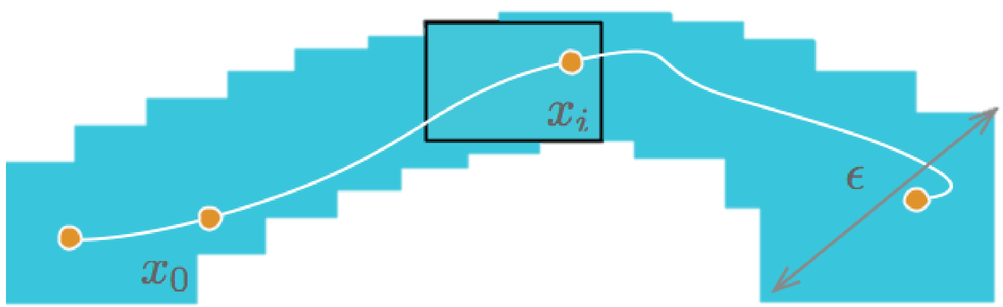
1. How do we get the simulations?
2. How much to bloat?
3. *<u>How to handle mode switches?</u>*

Building Blocks : Simulations

Simulation from x_0 given as $\xi(x_0, t)$ – no closed form!

simulation (x_0, h, ϵ, T) gives a sequence $S_0, \ldots, S_k$:

1. at any time $t \in [ih, (i+1)h]$, $\xi(x_0, t) \in S_i$
2. $dia(S_i) \leq \epsilon$

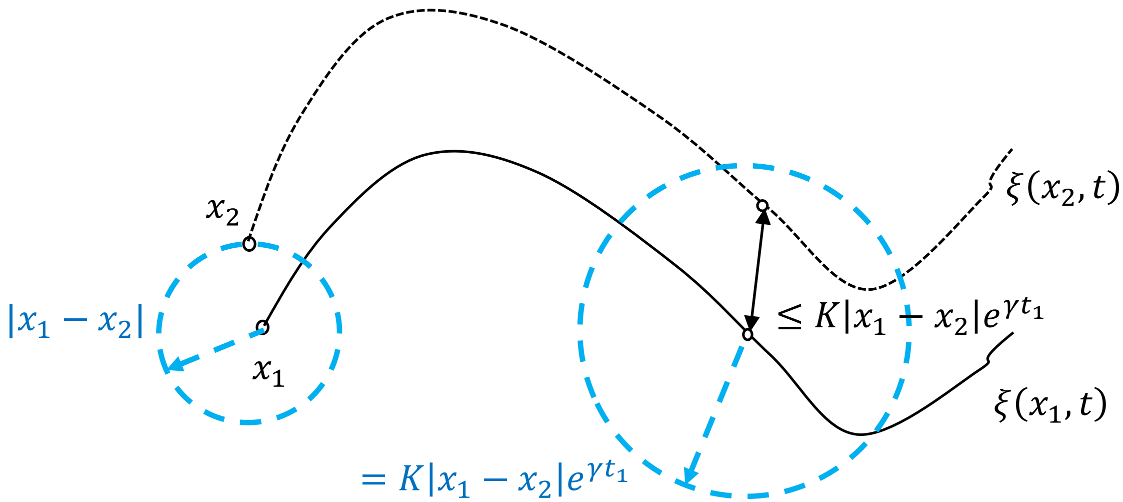


valSim (x_0, T, f) generates such simulations (CAPD)

Building Blocks : Discrepancy Function

<u>Discrepancy Function</u>: capturing the continuity of ODE solutions
executions that start close, stay close

$\langle K, \gamma \rangle$ is called an exponential discrepancy function of the system if for any two states x_1 and $x_2 \in X$, for any t $|\xi(x_1, t) - \xi(x_2, t)| \leq K|x_1 - x_2|e^{\gamma t}$



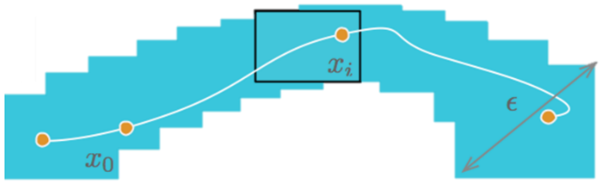
<u>Discrepancy functions are given as model annotations, i.e. $\langle K, \gamma \rangle$ is given by the user</u>

Simulations + Discrepancy Functions = *ReachTubes*

$\boldsymbol{\psi = reachtube(S, \epsilon, T)}$ of $\dot{x} = f(x)$ is a sequence $R_0, \ldots, R_k$ such that $dia(R_i) \leq \epsilon$ and from any $x_0 \in S$, for each time $t \in [ih, (i+1)h]$, $\xi(x_0, t) \in R_i$.

How to compute a ReachTube from validated simulation and annotation?

$\langle S_0, \ldots, S_k, \epsilon_1 \rangle \leftarrow \boldsymbol{valSim}(x_0, T, f)$



Simulations + Discrepancy Functions = *ReachTubes*

$\boldsymbol{\psi = reachtube(S, \epsilon, T)}$ of $\dot{x} = f(x)$ is a sequence $R_0, \ldots, R_k$ such that $dia(R_i) \le \epsilon$ and from any $x_0 \in S$, for each time $t \in [ih, (i+1)h]$, $\xi(x_0, t) \in R_i$.

How to compute a ReachTube from validated simulation and annotation?

$\langle S_0, \ldots, S_k, \epsilon_1 \rangle \leftarrow \boldsymbol{valSim}(x_0, T, f)$

For each $i \in [k]$

$\quad \epsilon_2 \leftarrow \max_{t \in T_i} K e^{\gamma t} \delta;$

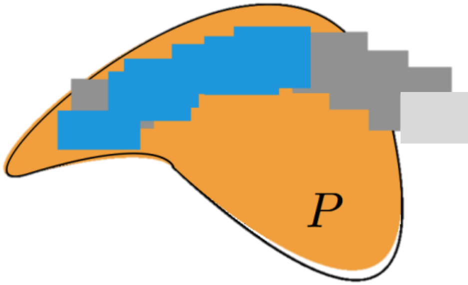
$\quad R_i \leftarrow B_{\epsilon_2}(S_i)$

$\langle R_0, \ldots, R_k \rangle$ is a ***reachtube***$(\boldsymbol{B_\delta(x_0), \epsilon_1 + \epsilon_2, T})$

- ✓ How do we get the simulations?
- ✓ How much to bloat?
- *How to handle mode switches?* → Invariants, Guards

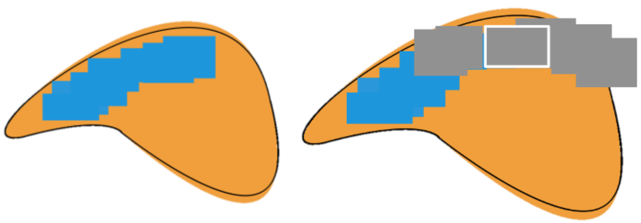
Handling Invariants

Tagging: track a region based on a predicate P



$$\textbf{tagRegion}(R,P) = \begin{cases} must & R \subseteq P \\ may & R \cap P \neq \emptyset, \bar{R} \cap P \neq \emptyset \\ not & R \cap P = \emptyset \end{cases}$$

<u>Goal: **Reachtube** that respects the invariant of the mode</u>



$\boldsymbol{\phi = invariantPrefix(\psi, Invariant)}$ is

$\langle R_0, tag_0, \ldots, R_m, tag_m \rangle$, such that either

$tag_i = must$ if all the $R_j's$ before it are must

$tag_i = may$ if all the $R_j's$ before it are tagged may or must and at least one of them is not must

Handling Guards & Resets

<u>Goal: Compute set of states in **Reachtube** that change mode based on **Guard**</u>

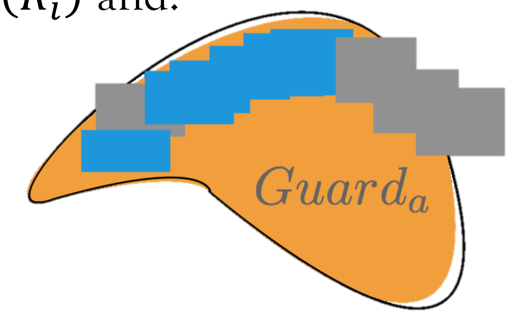
$\boldsymbol{nextRegions(\phi)}$ returns a set of tagged regions N.

$\langle R', tag' \rangle \in N$ iff $\exists\ a \in A, \langle R_i, tag_i \rangle \in \phi$ such that $R' = Reset_a(R_i)$ and:

$R_i \subseteq Guard_a, tag_i = tag' = must$

$R_i \cap Guard_a \neq \emptyset, R_i \notin Guard_a, tag_i = must, tag' = may$

$R_i \cap Guard_a \neq \emptyset, tag_i = tag' = may$



Tagging is essentially **bookkeeping**

1. $invariantPrefix$ discards the invalid trajectories (violating invariant)
2. $nextRegions$ tags the regions based on the feasibility of discrete transition

Utility of tagging

1. Reachable set is contained in union of *may* and *must* regions – inferring safety
2. There exists at least one reachable state in every *must* region – inferring violation of safety

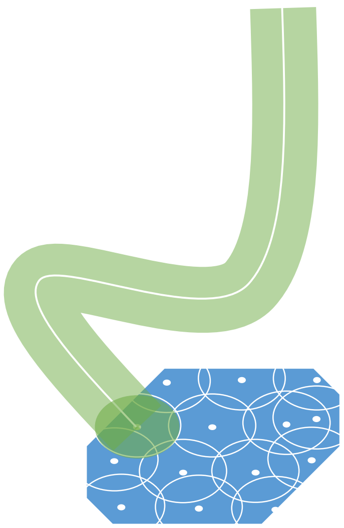
Algorithm for *Hybrid Systems*

Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N

$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag \rangle \in partition$

$\psi \leftarrow reachTube(S, T)$



end;

Algorithm for *Hybrid Systems*

Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N

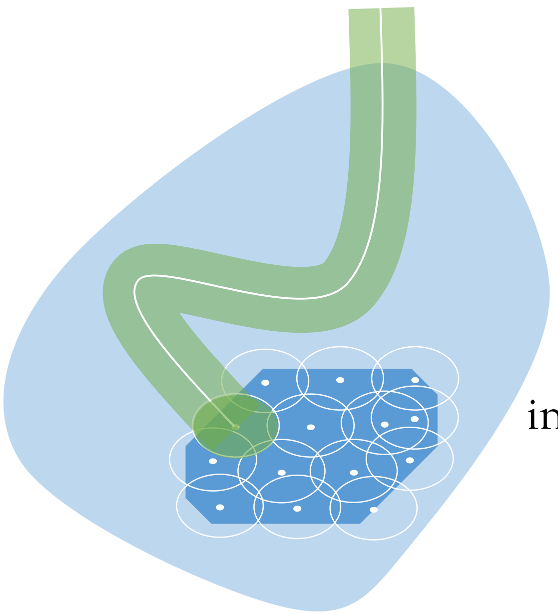
$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag\rangle \in partition$

$\psi \leftarrow reachTube(S, T)$

$\phi \leftarrow invariantPrefix(\psi)$

end;



Algorithm for *Hybrid Systems*

Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N

$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag \rangle \in partition$

$\psi \leftarrow reachTube(S, T)$

$\phi \leftarrow invariantPrefix(\psi)$

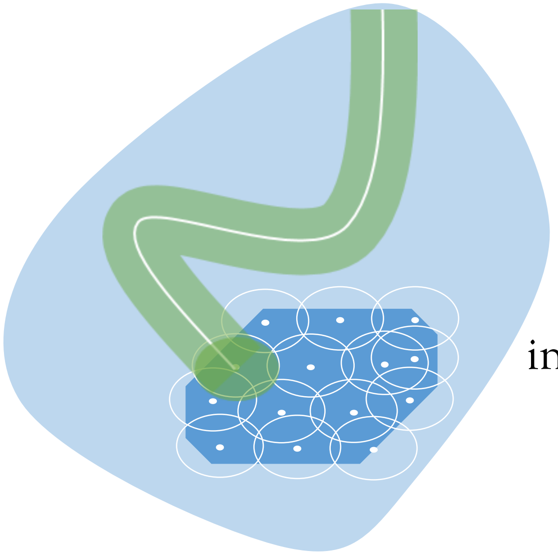
if (ϕ is safe) then continue;

if (ϕ is unsafe and tag is $must$) return unsafe;

else *refine* tagged cover;

end;

return safe;



Algorithm for *Hybrid Systems*

Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N

$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag \rangle \in partition$

$\psi \leftarrow reachTube(S, T)$

$\phi \leftarrow invariantPrefix(\psi)$

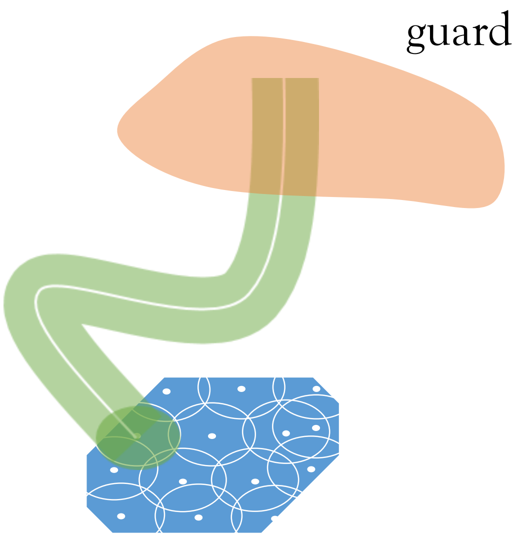
if (ϕ is safe) then continue;

if (ϕ is unsafe and tag is $must$) return unsafe;

else *refine* tagged cover;

end;

return safe;



Algorithm for *Hybrid Systems*

Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N

$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag \rangle \in partition$

$\psi \leftarrow reachTube(S, T)$

$\phi \leftarrow invariantPrefix(\psi)$

$Next \leftarrow nextRegions(\phi)$

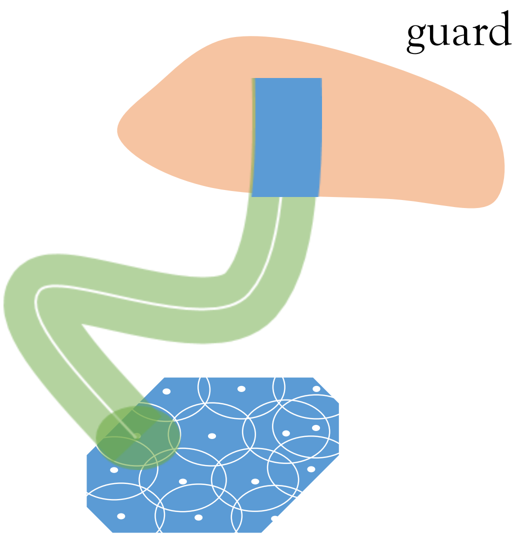
if (ϕ is safe) then check $Next$;

if (ϕ is unsafe and tag is $must$) return unsafe;

else *refine* tagged cover;

end;

return safe;



Algorithm for *Hybrid Systems*

<u>Input: Initial Set Θ, Unsafe set U, Time T, Number of Switches N</u>

$partition \leftarrow taggedCover(\Theta)$

$\forall \langle S, tag\rangle \in partition$

  $queueRegions \leftarrow \{\langle S, tag\rangle\}$

  $\forall \langle S, tag\rangle \in queueRegions$ until N steps and T time

    $\psi \leftarrow reachTube(S, T)$

    $\phi \leftarrow invariantPrefix(\psi)$

    $Next \leftarrow nextRegions(\phi)$

    if (ϕ is safe) *enque* $Next$ *to* $queueRegions$;

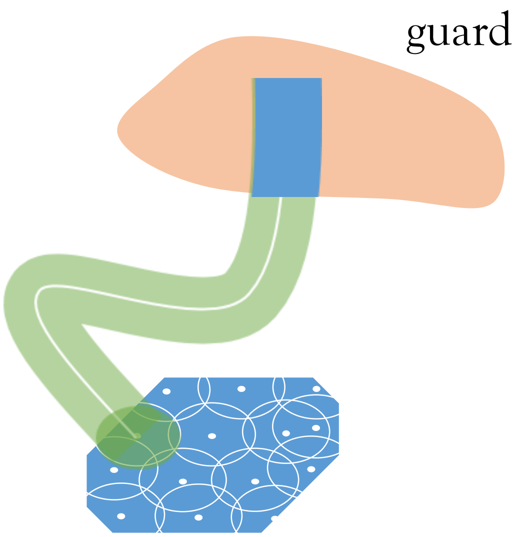
    if (ϕ is unsafe and tag is $must$) return unsafe;

    else *refine* tagged cover;

  end;

end;

return safe;



Soundness & Relative Completeness

[Soundness]: If the algorithm returns safe(or unsafe), then the system is indeed safe(or unsafe).

Proof sketch:

1. Union of *May* and *Must* regions contains the reachable set
2. Algorithm returns safe only when all the *May* and *Must* regions are safe
3. Algorithm returns unsafe only when a *Must* region is contained in the unsafe set

Soundness & Relative Completeness

[Relative Completeness]: If the system is *robustly safe* or *robustly unsafe*, then the algorithm will terminate with correct answer.

Definition

Robustly safe: If there is ϵ separation between reachable set and U

Robustly unsafe: If ϵ shrinkage of invariants, guards, and initial set Θ, is unsafe with respect to ϵ shrinkage of U

Proof sketch:

1. Refining the cover enough will ensure that overapproximation is less than ϵ, so if the system is robustly safe, the algorithm returns safe
2. If the ϵ shrinkage of invariants, guards, Θ, and U is unsafe, then $\exists\ R_i$ tagged $must$ in the reachable that is unsafe

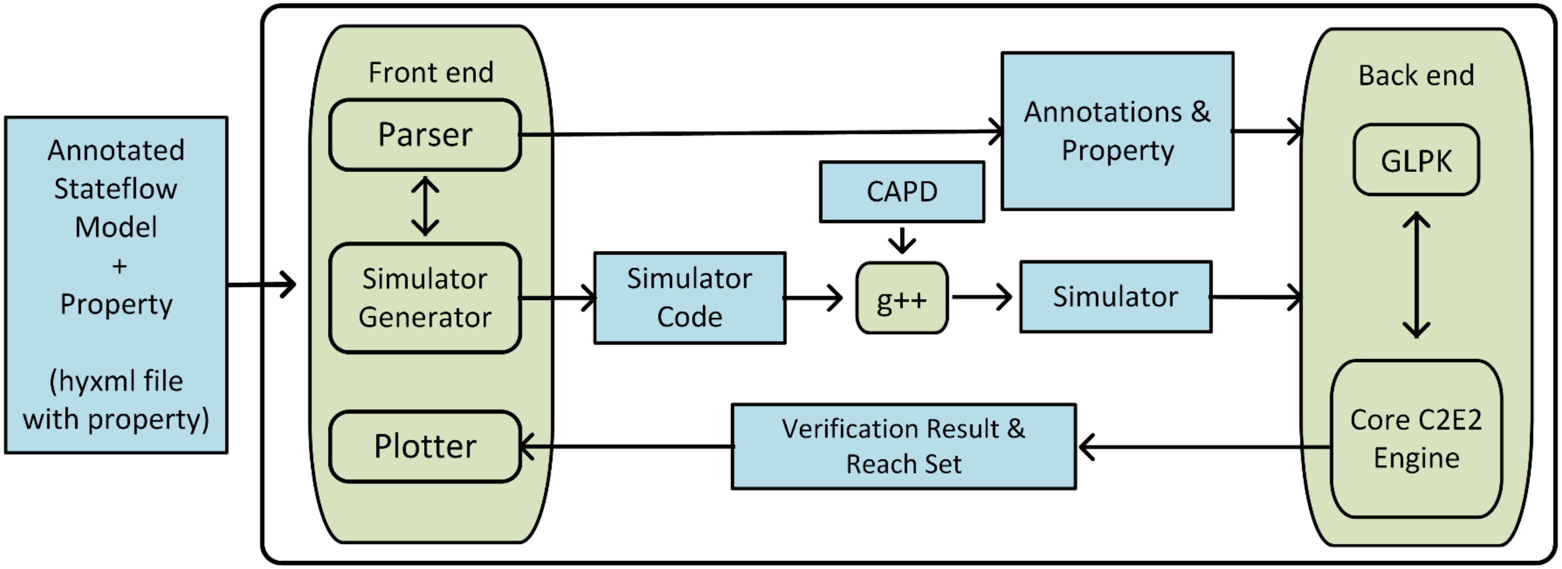
Overview

- ✓ Motivation and Problem Statement
- ✓ Challenges in Verification
- ✓ Building Blocks and Algorithm
- ✓ Soundness and Relative Completeness Guarantees
- Tool Features
- Annotations
- Future Work

C2E2 : Compare-Execute-Check-Engine

Features:

- Stateflow models
- Graphical User Interface
- Plotting



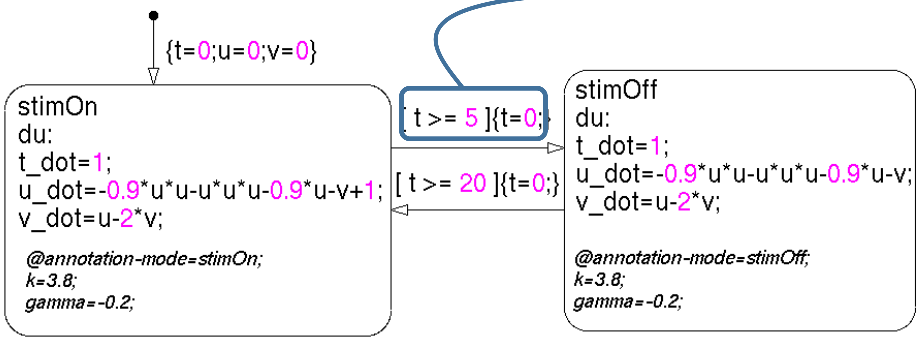
Architecture of C2E2

C2E2: Features, Architecture, & Usability

Stateflow models: No formal semantics from MATHWORKS, Hybrid automata semantics by Tiwari [‘02], Manamcheri et.al.[‘10]

Urgent semantics:

$t \geq 5; t = 0$

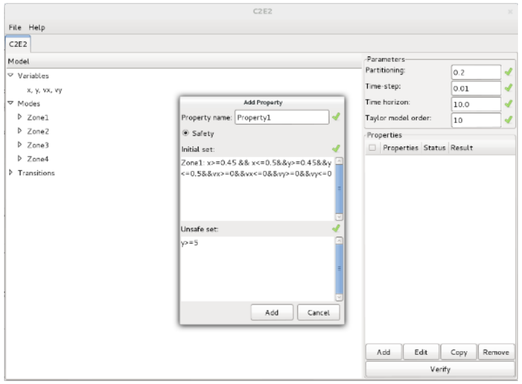


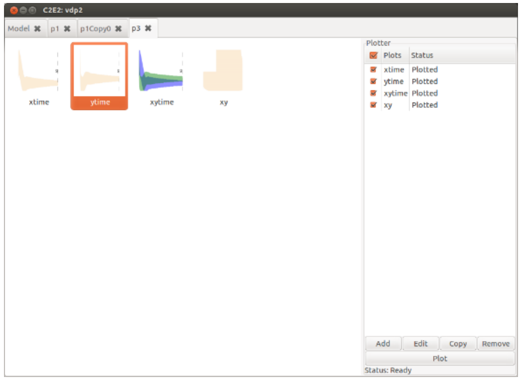
Bloating the guard set: for providing robust counterexamples

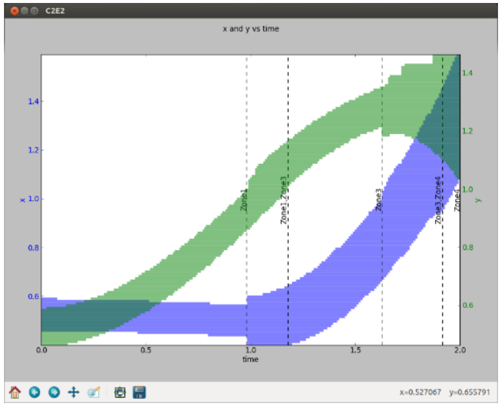
$$t \geq 5 \Rightarrow t \geq 5 - \epsilon, t \leq 5 + \epsilon$$

C2E2: Features, Architecture, & Usability

- GUI for viewing model, properties
- Saving model in *hyxml* format
- Interface for plotting reachable set



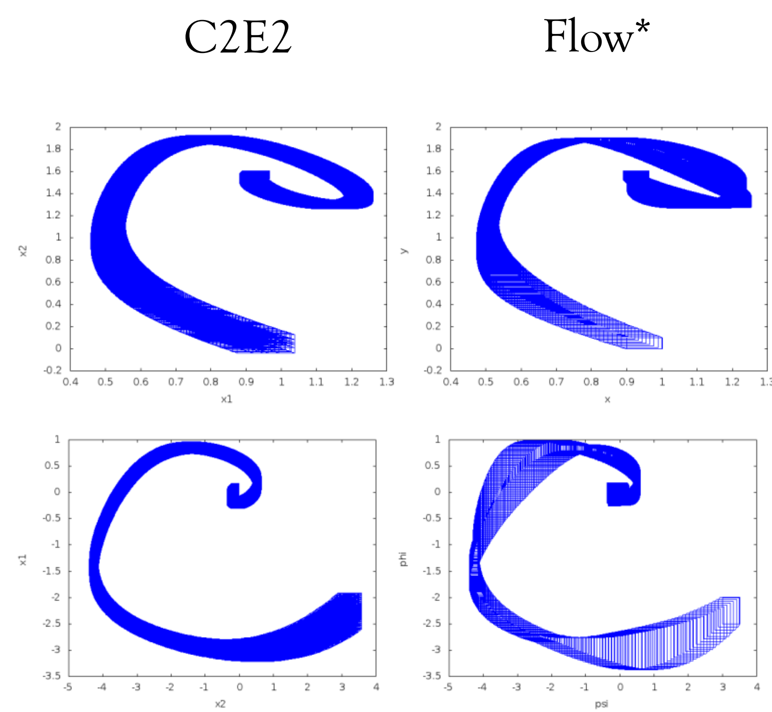




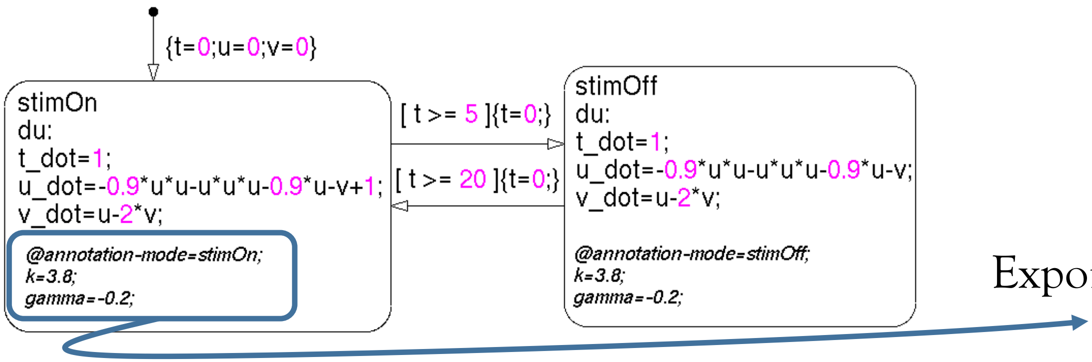
More in the Tool Demo Market

Comparison with Existing Approaches on Academic Benchmarks [DMV'13]

Benchmark	Variables	Sims.	C2E2 (time)	Flow* (time)	Ariadne (time)
Moore-G. Jet Engine	2	36	**1.56**	10.54	56.57
Brussellator System	2	115	**5.26**	16.77	72.75
VanDerPol Oscillator	2	17	**0.75**	8.93	98.36
Coupled VanDerPol	4	62	**1.43**	90.96	270.61
Sinusoidal Tracking	6	84	**3.68**	48.63	763.32
Linear Adaptive	3	16	**0.47**	NA	NA
Nonlinear Adaptive	2	32	**1.23**	NA	NA
Nonlinear Disturbance	3	48	**1.52**	NA	NA



Discrepancy Functions – Model Annotations



Exponential discrepancy function $\langle K = 3.8, \gamma = -0.2 \rangle$

- Sufficient conditions for finding discrepancy functions (borrowed from Control Theory)
  - Lipschitz continuity: $\dot{x} = f(x)$ has Lipschitz constant L, then $|x_1(t) - x_2(t)| \leq |x_1 - x_2|e^{Lt}$
  - Contraction Metric: If $J^T M + M\,J + b_M M \preccurlyeq 0$, then $\exists k, \delta > 0, |x_1(t) - x_2(t)\,|^2 \leq k|x_1 - x_2|^2 e^{-\delta t}$
  - Incremental Lyapunov Function: With function V, then $|x_1(t) - x_2(t)\,| \leq k\,|x_1 - x_2|; k = F(V)$

- Finding such discrepancy function automatically
  - Nonlinear optimization for Lipschitz continuity
  - For $\dot{v} = Av$ that are exponentially stable, compute Lyapunov function
  - Solving LMIs using Sum-Of-Squares tools to compute contraction metric
  - Manual proof methods using coordinate transformation and eigen values of Jacobian

Summary & Future Work

- Simulation based verification algorithm for *Fully Hybrid Systems*
- Soundness and Relative completeness guarantees
- Tool features:
  - Stateflow models
  - GUI and usability enhancements
  - Plotting for visualizing reachable set

Future Work

- Automatically finding discrepancy functions
- Theoretical Result: Minimum number of simulations to verify a given system

Thank You, Questions?