dReal & dReach

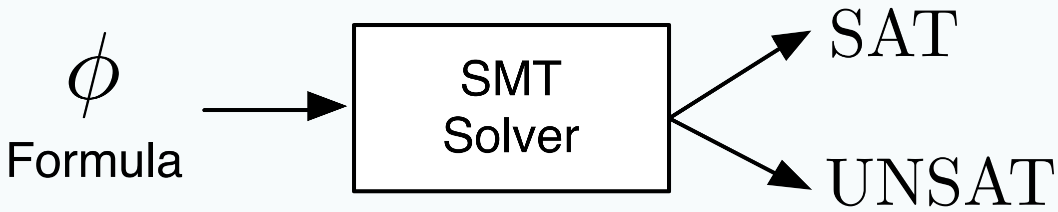
THE DELTA DECISION TOOLS

Soonho Kong / **Sicun Gao** / **Edmund Clarke**

2013/11/22, CMACS/AVACS Workshop

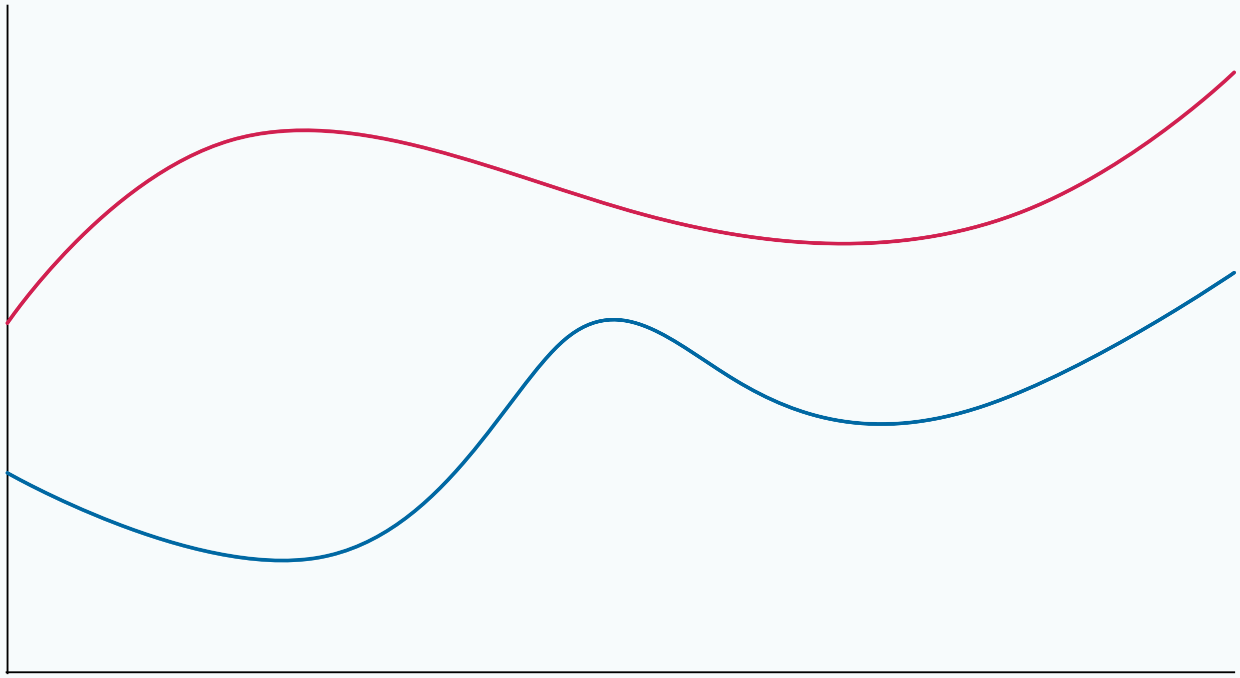
http://dreal.cs.cmu.edu

SMT Problem



- Complexity results of **non-linear** arithmetic over the **reals**
  - **Decidable** if ϕ only contains **polynomials** [Tarski51]
  - **Undecidable** if ϕ contains trigonometric functions
- **Real-world problems** contain **complex nonlinear functions** ($\sin$, $\exp$, ODEs)

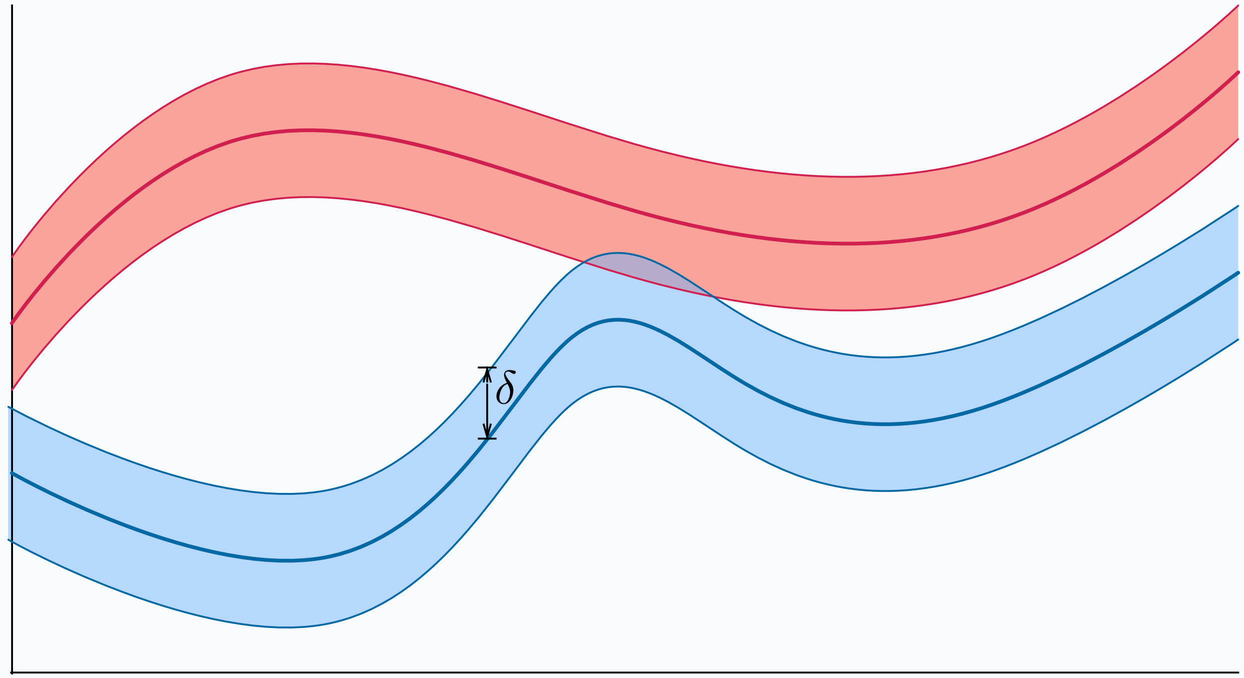
Decision Problem



Standard Form

$$\phi := \exists^{\mathbf{I}} \mathbf{x} \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} f_{ij}(\mathbf{x}) = 0$$

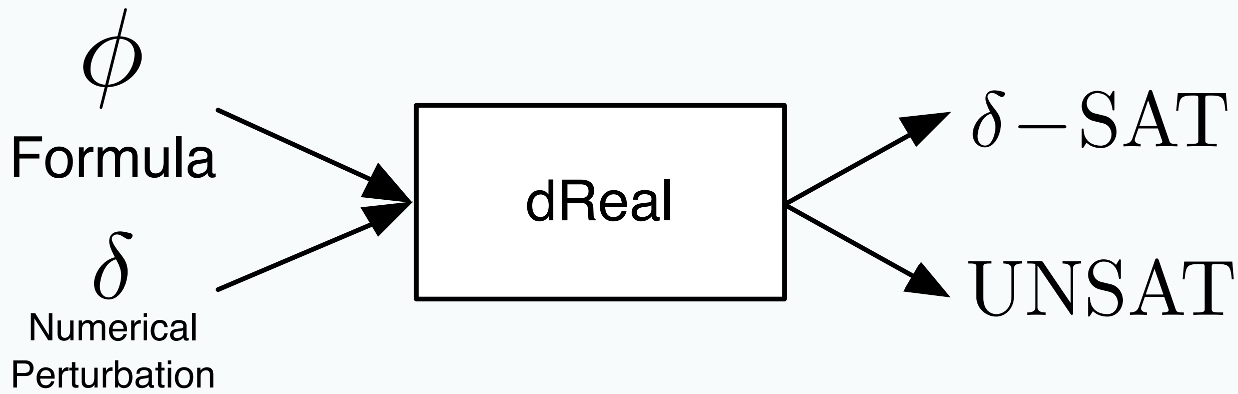
δ-decision Problem



δ-Weakening of ϕ

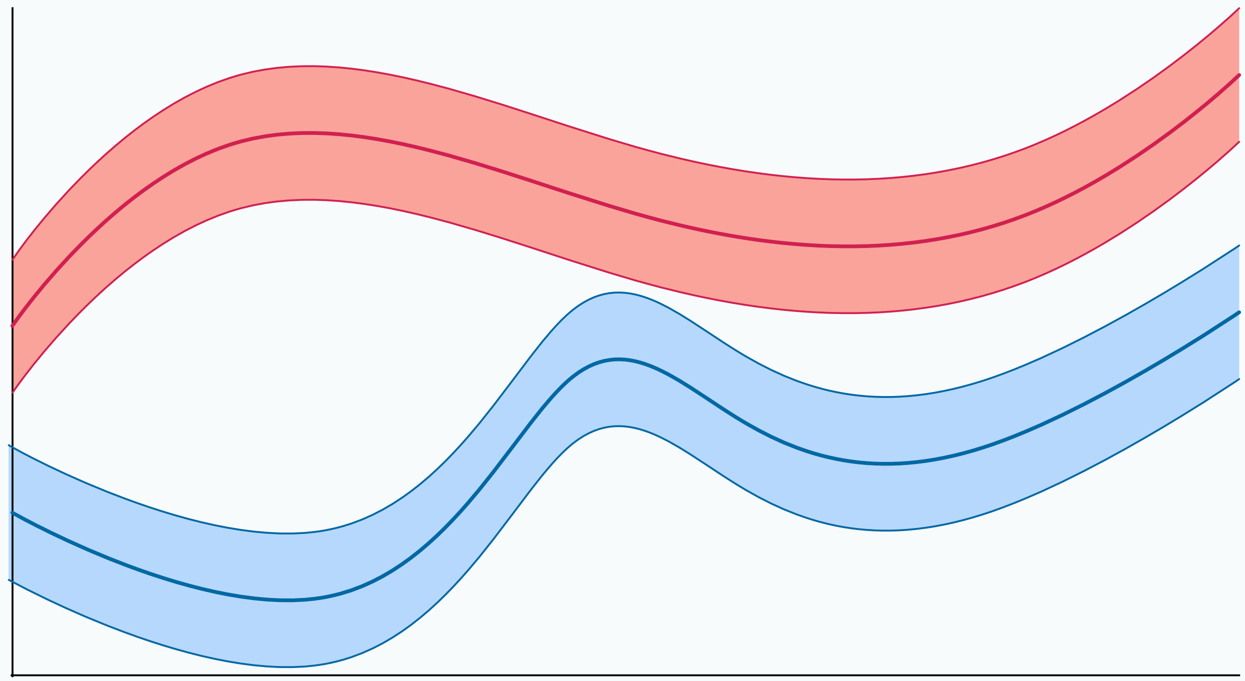
$$\phi^{\delta} := \exists^{\mathbf{I}}\mathbf{x} \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} | f_{ij}(\mathbf{x}) | \leq \delta$$

δ-decision Problem



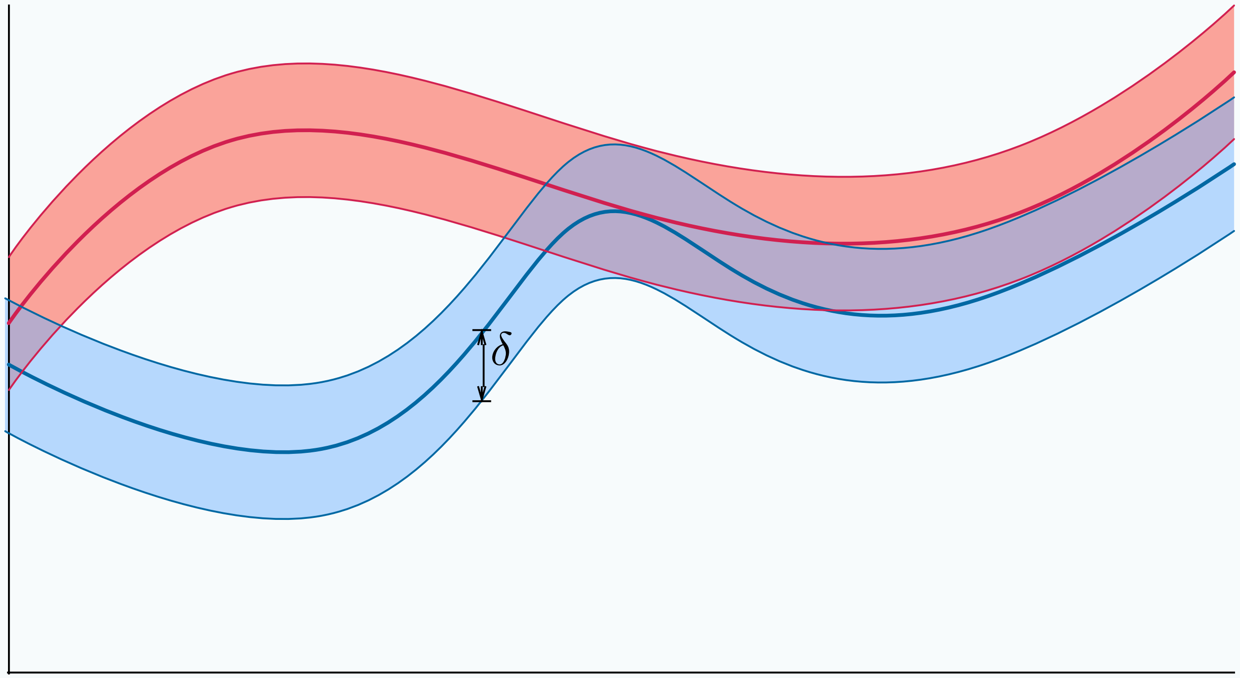
- **UNSAT**: ϕ^δ is unsatisfiable
- **δ-SAT**: ϕ^δ is satisfiable
- **Decidable** [LICS'12, IJCAR'12]
  - **NP**-complete: $\mathcal{F} = \{+, \times, \exp, \sin, \ldots\}$
  - **PSPACE**-complete: $\mathcal{F} = \{\text{ODEs with P-computable rhs}\}$

δ-decision Problem



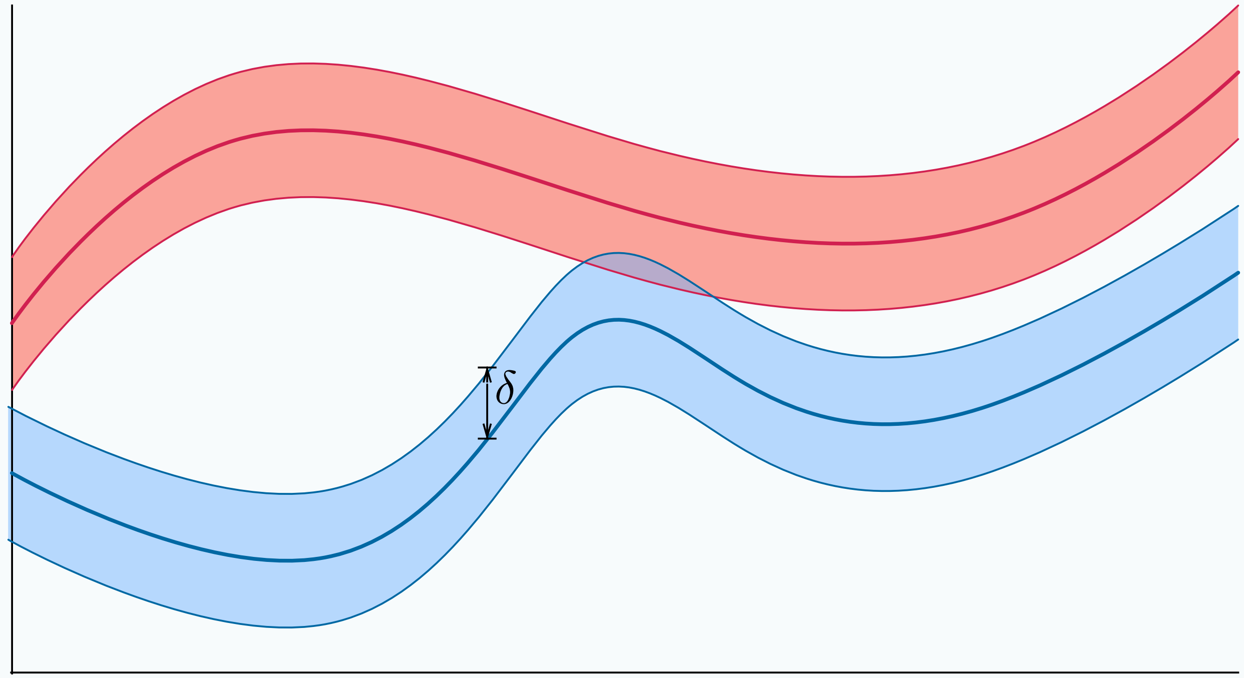
$$\phi^{\delta} : UNSAT \implies \phi : UNSAT$$

δ-decision Problem



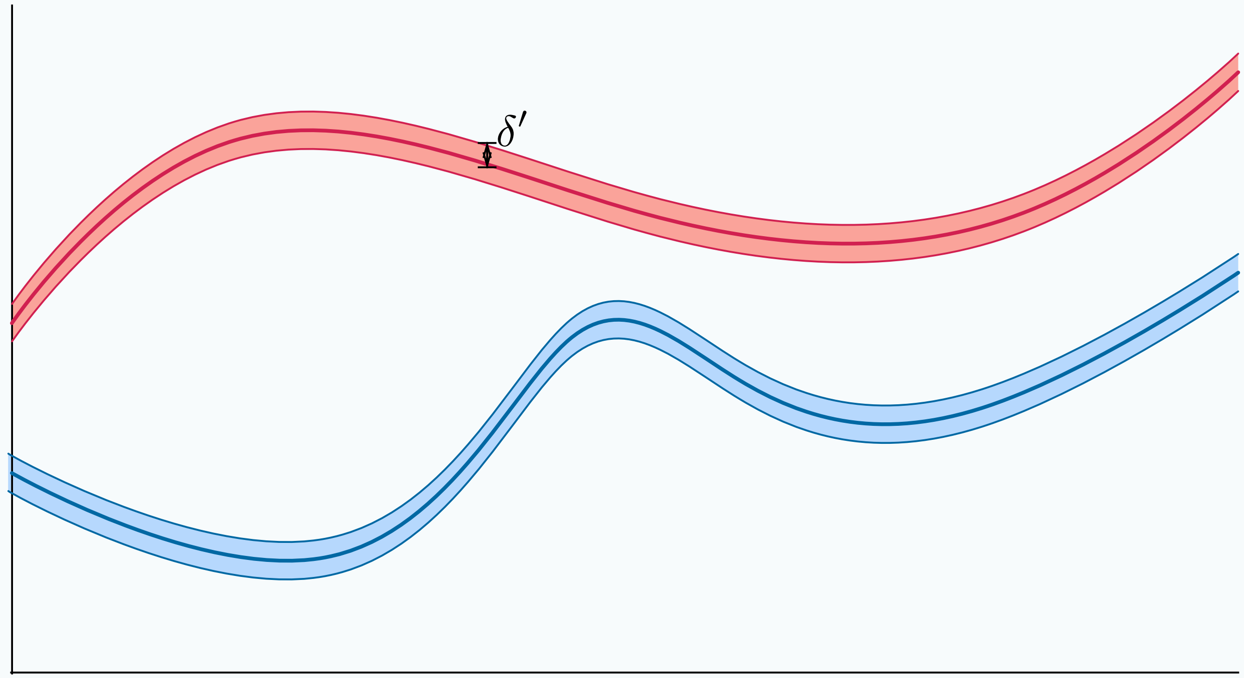
$$\phi^{\delta} : SAT \implies \phi : SAT \vee \phi : UNSAT$$

δ-decision Problem



$$\phi^{\delta} : SAT \implies \phi : SAT \vee \phi : UNSAT$$

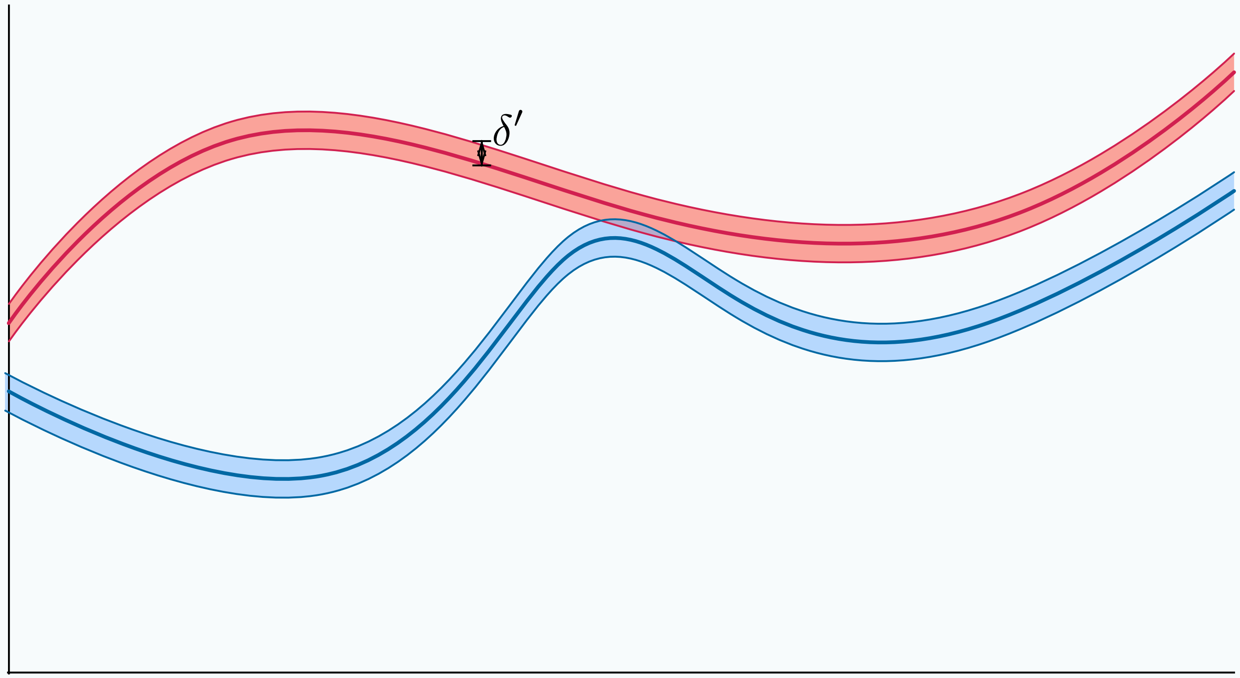
δ-decision Problem



May find a smaller $\delta' < \delta$ such that

$$\phi^{\delta'} : UNSAT \implies \phi : UNSAT$$

δ-decision Problem



$\phi^{\delta'}$: SAT with a **reasonably small** δ'

may indicate a **robustness** problem of the system in verification.

"Small perturbation on the system may **violate** safety properties"

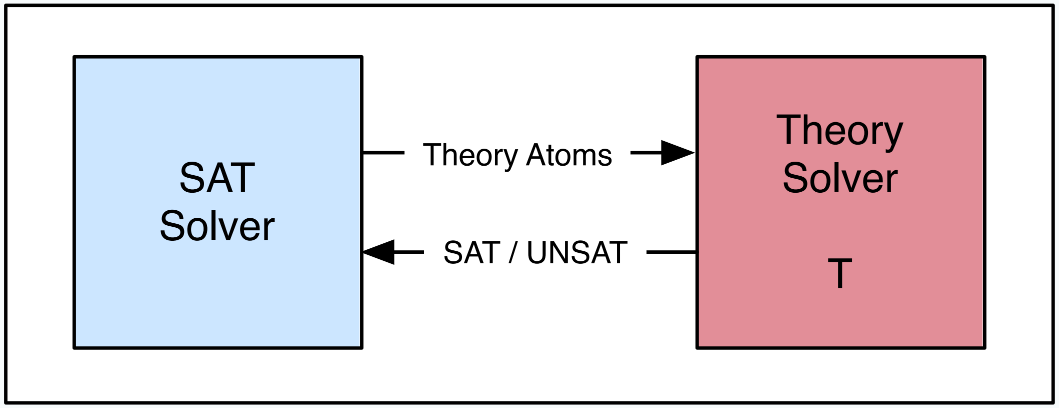
dReal

- **δ-complete** SMT solver
- Can handle various **nonlinear real functions** such as

$$\sin, \cos, \tan, \arcsin, \arccos, \arctan, \log, \exp, \ldots$$

- Can handle **ODEs** (Ordinary Differential Equations)
- **Open-source**: **http://dreal.cs.cmu.edu**

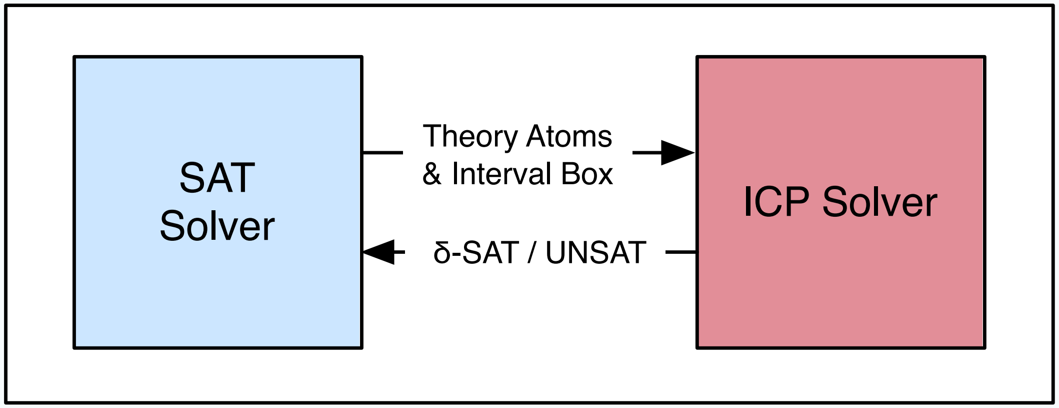
Design of **dReal**



General **DPLL⟨T⟩** Framework

- SAT Solver: provide Boolean abstraction
- Theory Solver **T**: check T-satisfiability of assignment.

Design of **dReal**

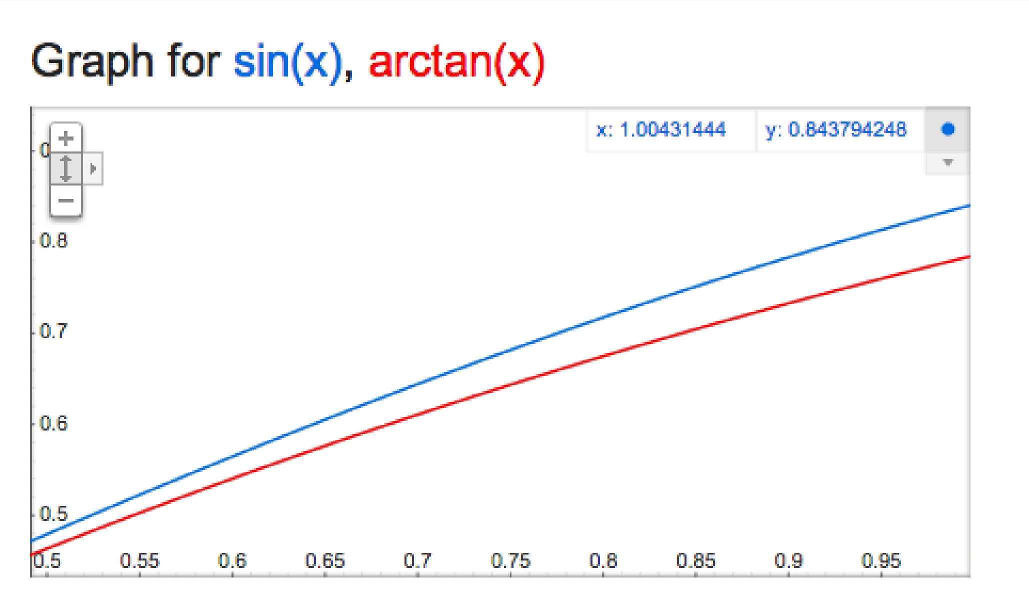


DPLL⟨ICP⟩ Framework

- **ICP** (Interval Constraint Propagation) solver
- Uses "**Branch & Prune**" algorithm

ICP: Pruning

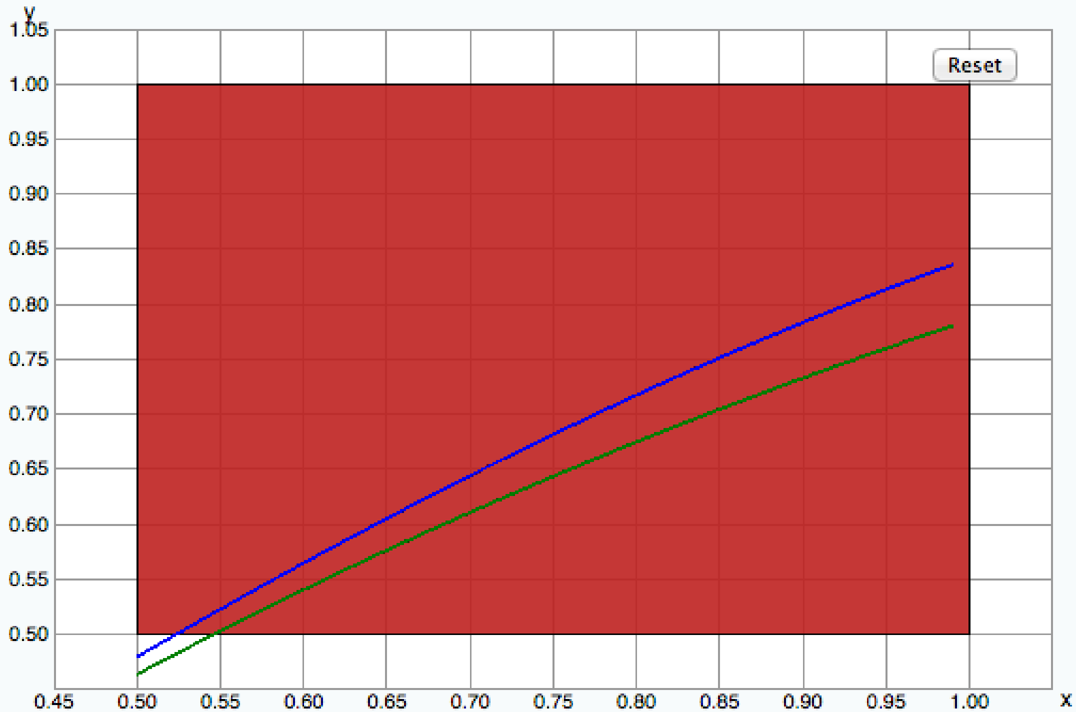
$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \mathrm{atan}(x)$$



ANSWER: **UNSAT**

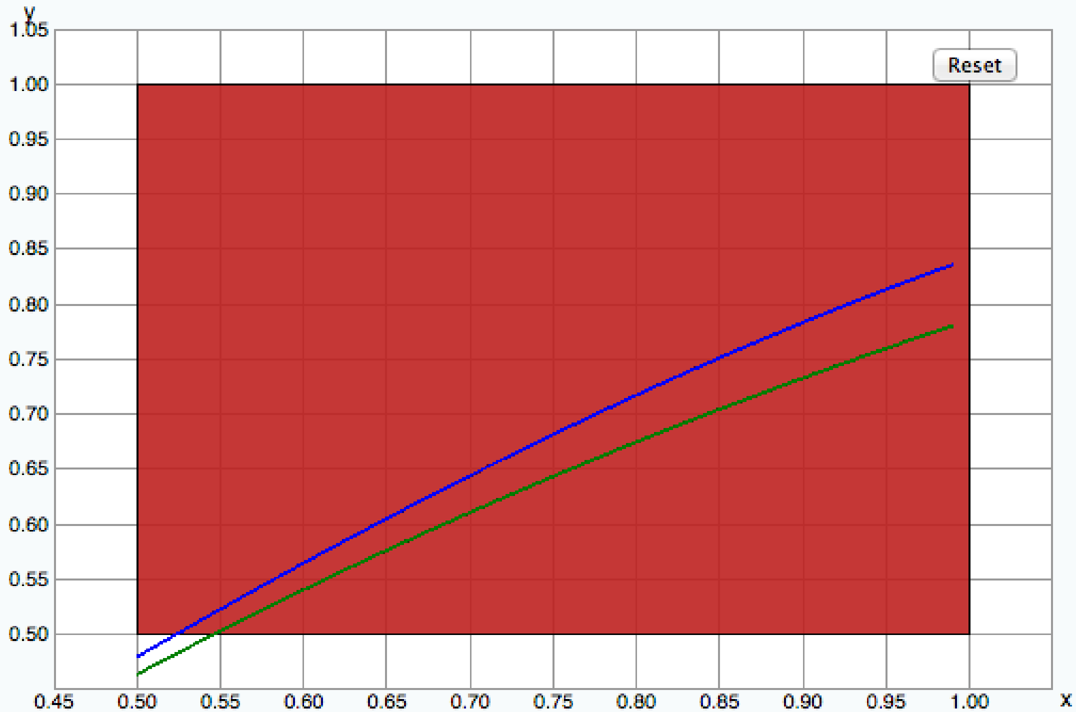
ICP: Pruning

$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \text{atan}(x)$$



ICP: Pruning

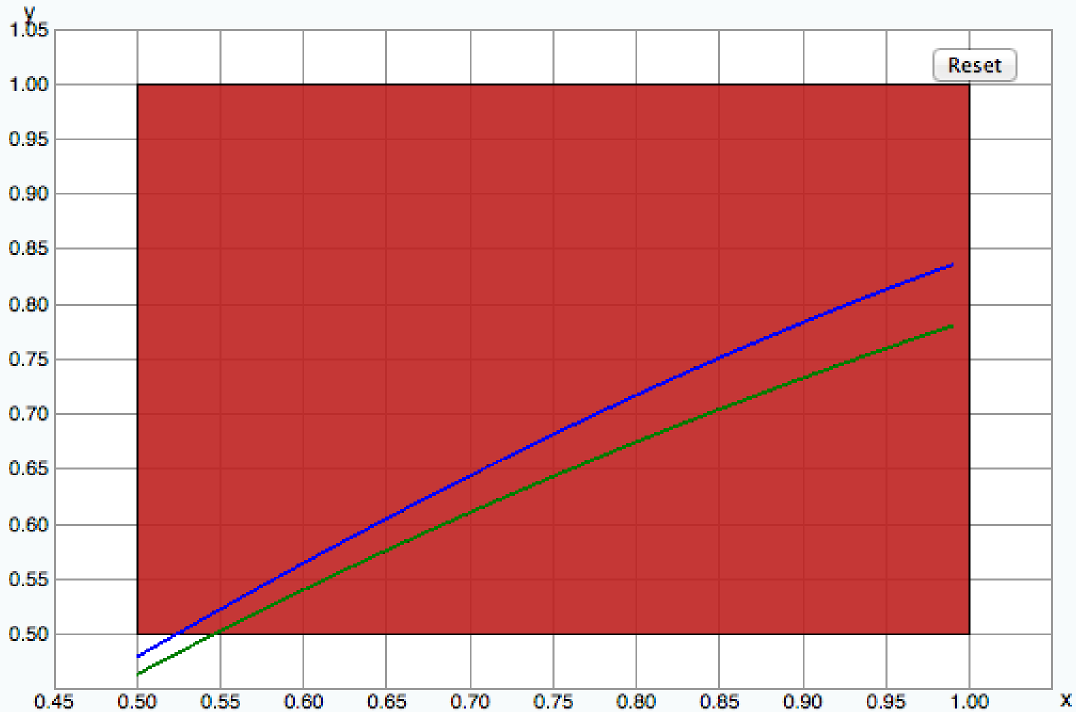
$$\exists x, y \in [0.5, 1.0] : {\color{blue} y = \sin(x)} \wedge y = \mathrm{atan}(x)$$



$$\begin{aligned} y' &= y \cap \sin(x) \\ &= [0.5, 1.0] \cap \sin([0.524, 1.0]) \\ &= [0.5, 1.0] \cap [0.5, 0.841] \\ &= [0.5, 0.841] \end{aligned}$$

ICP: Pruning

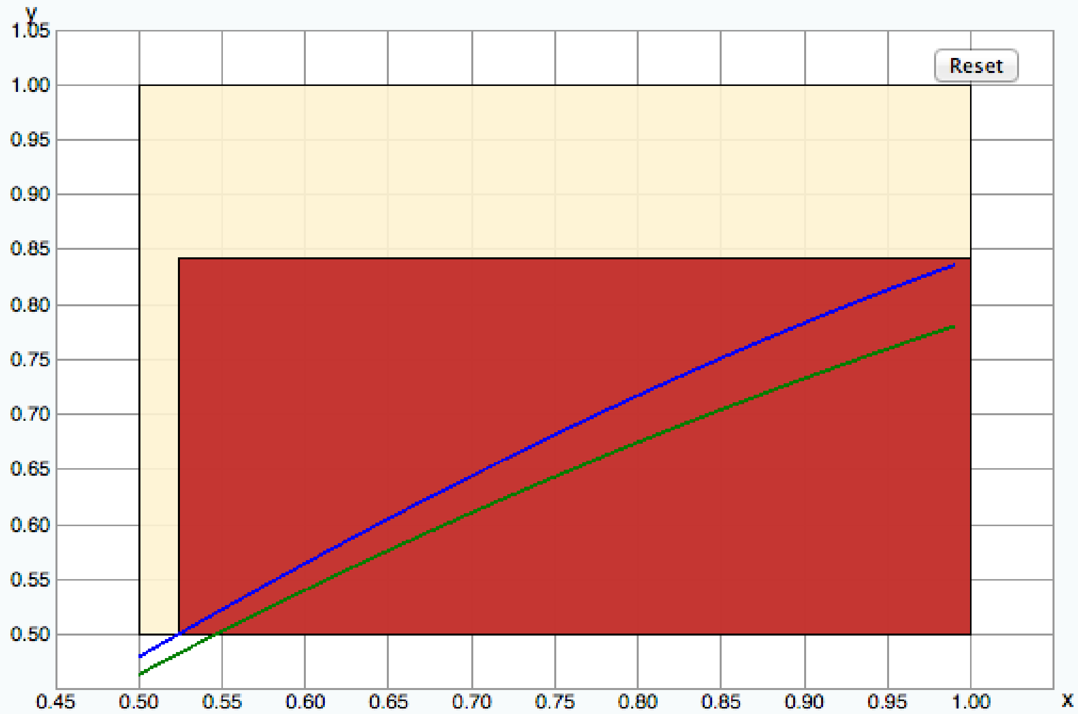
$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \operatorname{atan}(x)$$



$$\begin{aligned} x' &= x \cap \sin^{-1}(y) \\ &= [0.5, 1.0] \cap \sin^{-1}([0.5, 1.0]) \\ &= [0.5, 1.0] \cap [0.524, 1.570] \\ &= [0.524, 1.0] \end{aligned}$$

ICP: Pruning

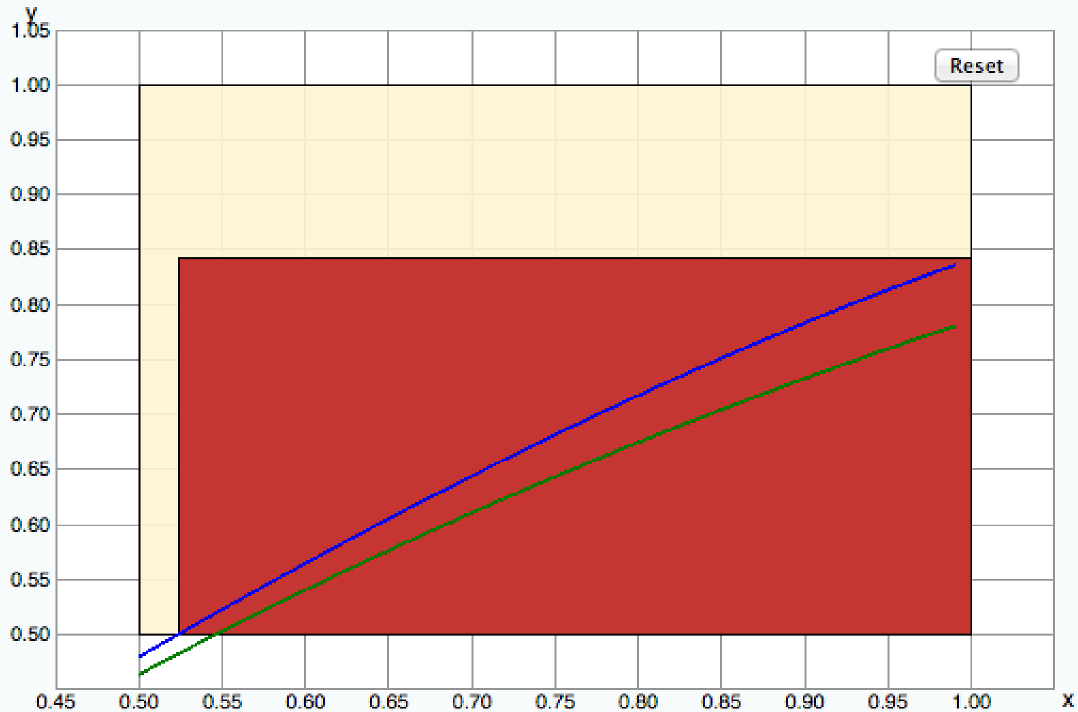
$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \mathrm{atan}(x)$$



$$x : [0.524, 1.0], y : [0.5, 0.841]$$

ICP: Pruning

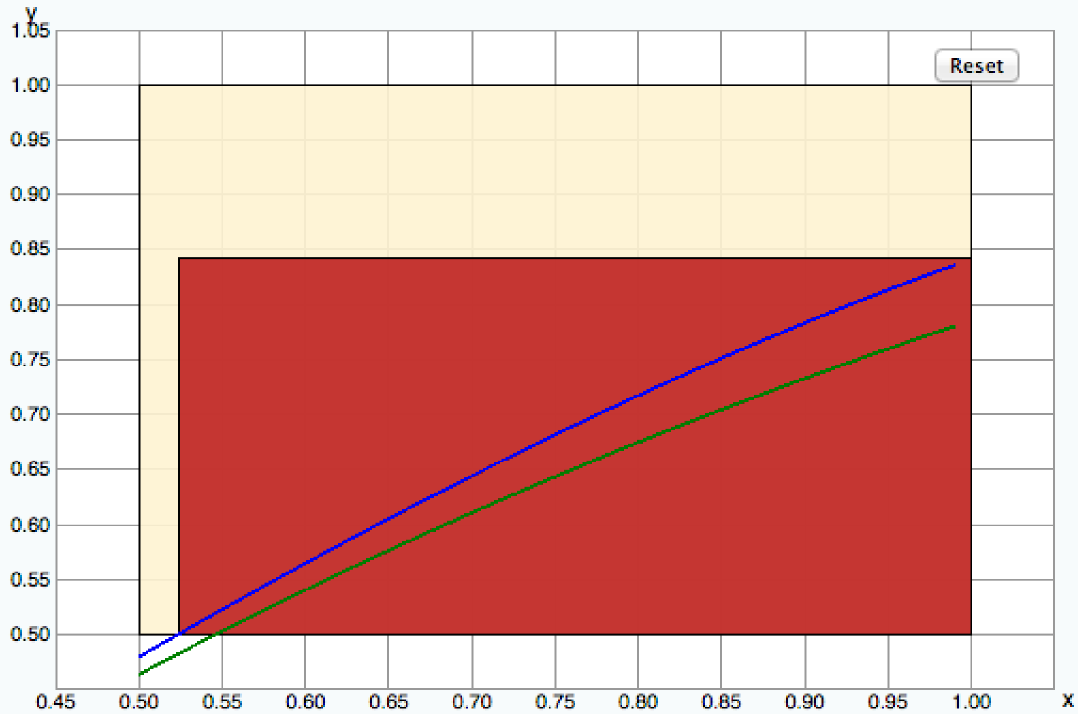
$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \text{atan}(x)$



$$
\begin{aligned}
x' &= x \cap \text{atan}^{-1}(y) \\
&= [0.524, 1] \cap \text{atan}^{-1}([0.5, 0.841]) \\
&= [0.524, 1] \cap [0.546, 1.117] \\
&= [0.546, 1.0]
\end{aligned}
$$

ICP: Pruning

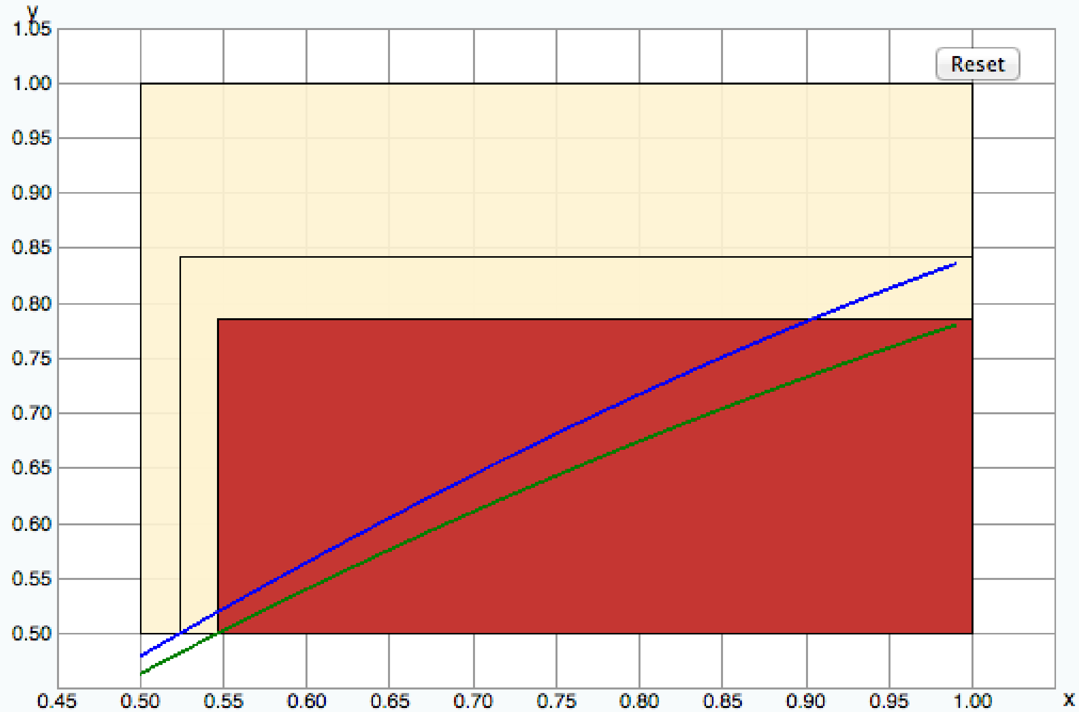
$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \text{atan}(x)$$



$$\begin{aligned} y' &= y \cap \text{atan}(x) \\ &= [0.5, 0.841] \cap \text{atan}([0.546, 1.0]) \\ &= [0.5, 0.841] \cap [0.5, 1.0] \\ &= [0.5, 0.785] \end{aligned}$$

ICP: Pruning

$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \text{atan}(x)$$

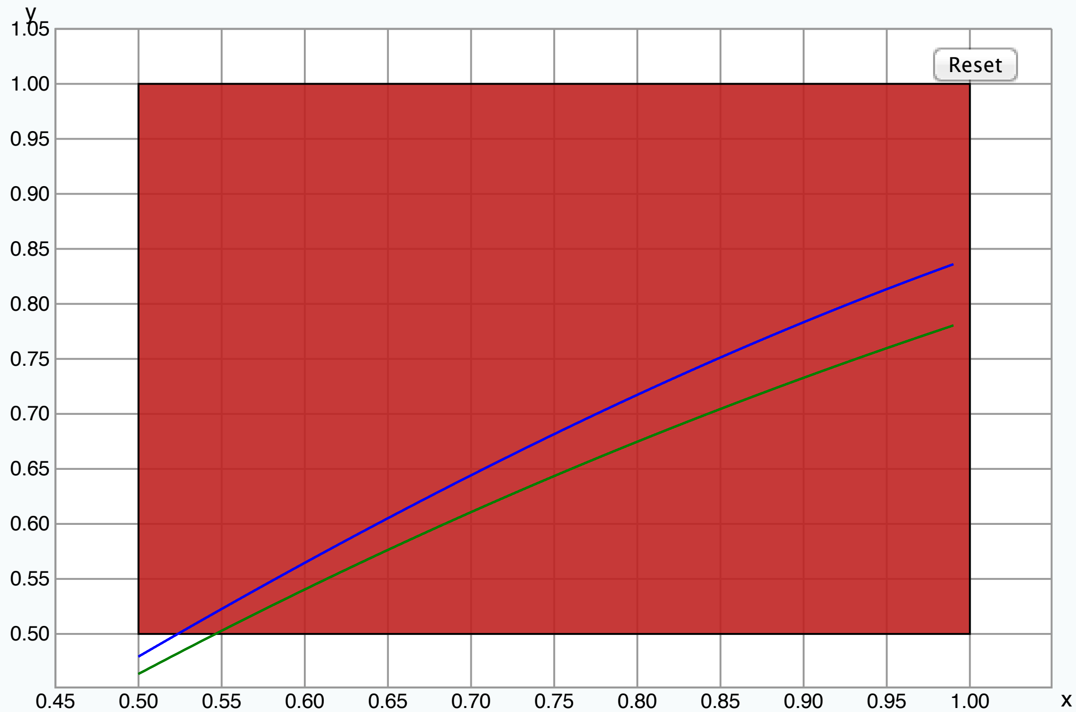


$$x : [0.524, 1.0], y : [0.5, 0.785]$$

ICP: Pruning

$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \operatorname{atan}(x)$$

Begin x dim : x y dim : y Next

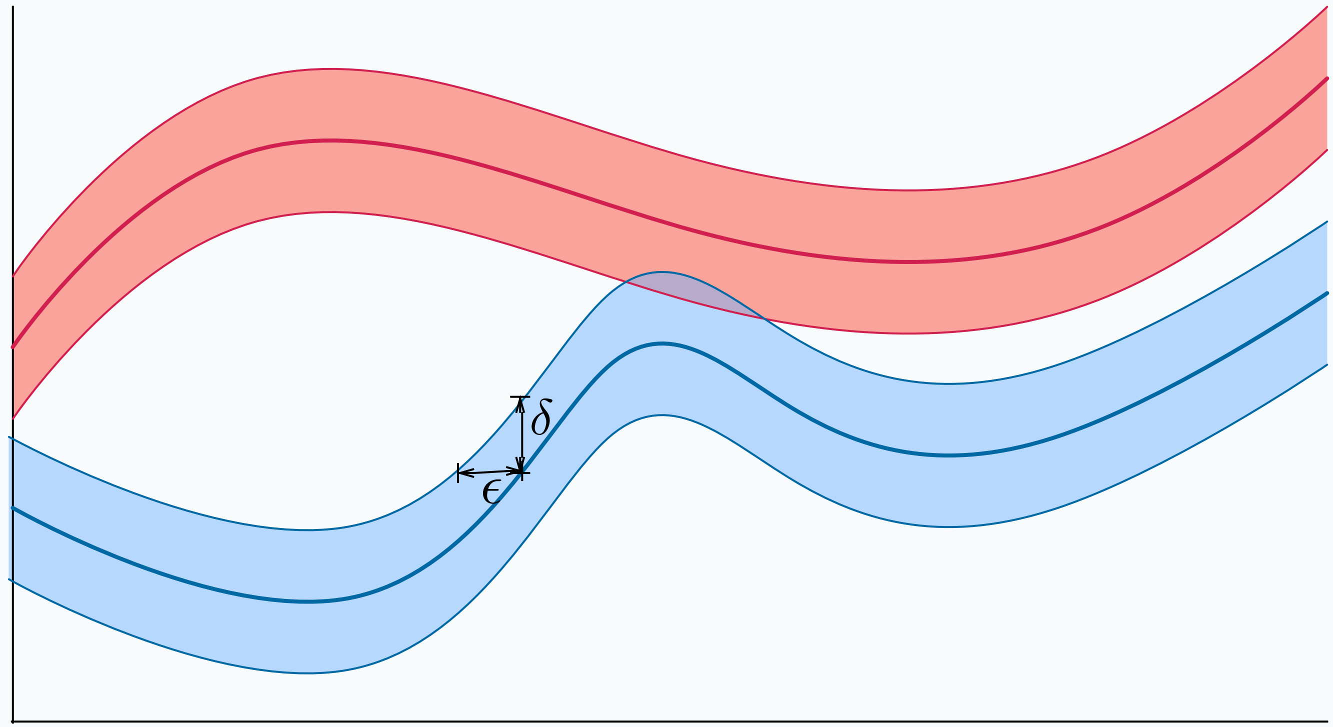


ANSWER: **UNSAT**

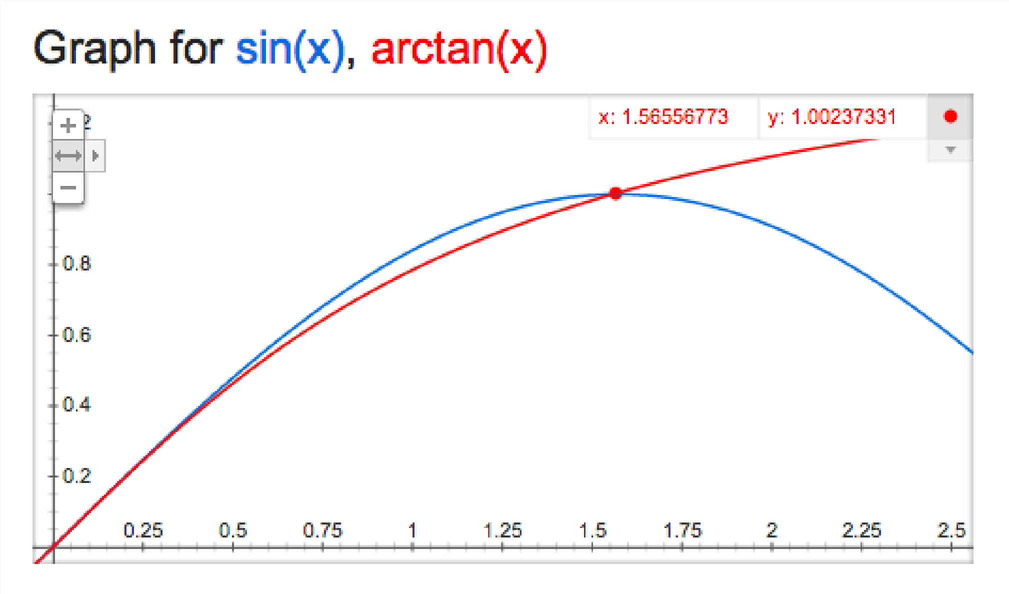
ICP: Branching

- Divide the search space and try each one
- **Stop** when the size of box is smaller than ϵ:

$$|B| < \epsilon$$



ICP: Branching

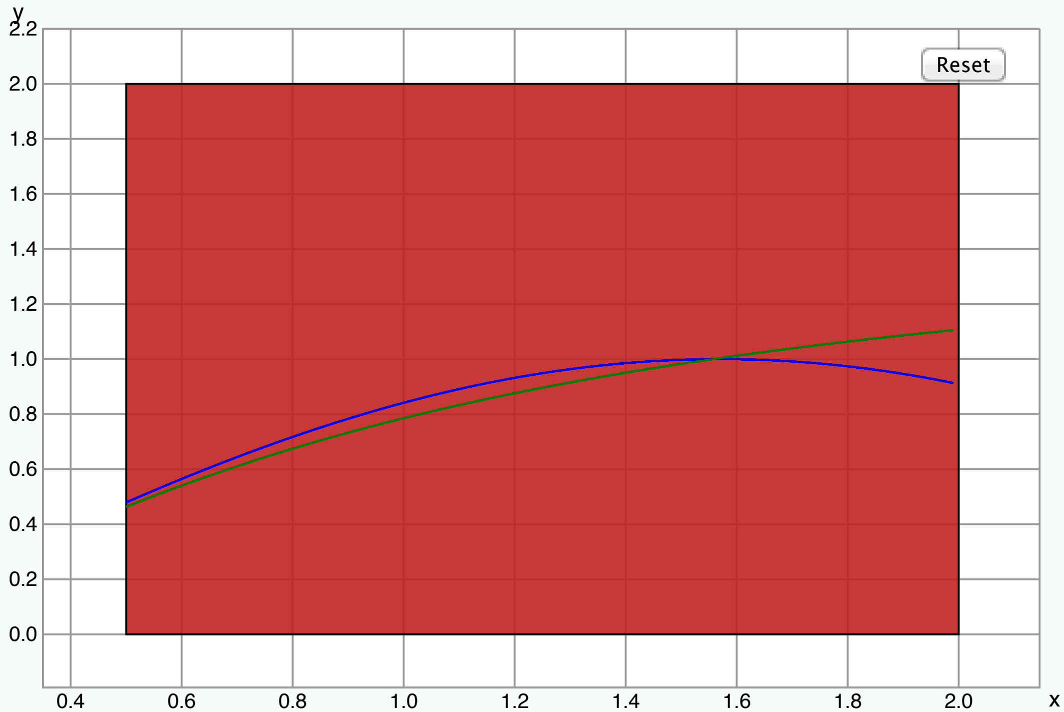


ANSWER: **SAT**

ICP: Branching

$$\exists x \in [0.5, 2.0], y \in [0.0, 2.0] : y = \sin(x) \wedge y = \text{atan}(x)$$

Begin x dim : x y dim : y Next



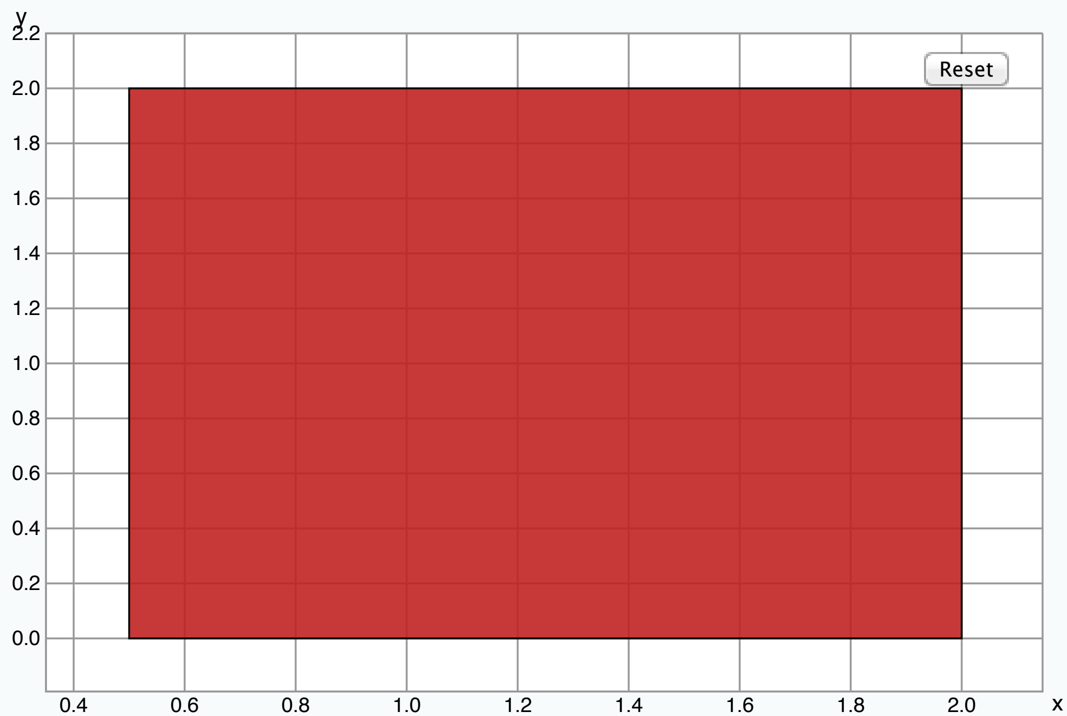
$$\epsilon = 0.001$$

ANSWER: **δ-SAT**

ICP: Branching

$$\exists x \in [0.5, 2.0], y \in [0.0, 2.0] : y = \sin(x) \wedge y = \operatorname{atan}(x)$$

Begin x dim : x y dim : y Next



$$\epsilon = 0.001$$

ANSWER: **δ-SAT**, $x = [1.556, 1.557], y = [1.000, 1.000]$

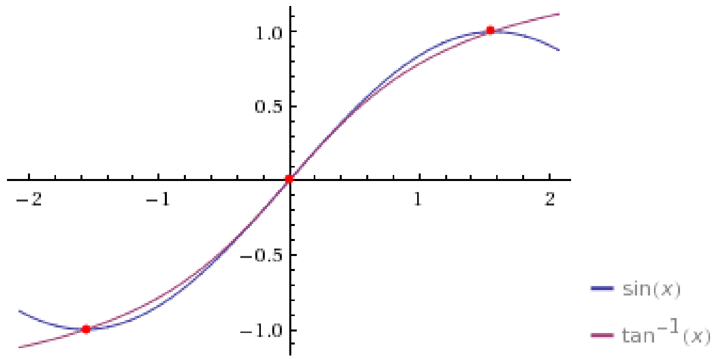
ICP: Branching

Input:

$\sin(x) = \tan^{-1}(x)$

$\tan^{-1}(x)$ is the inverse tangent function »

Plot:



Alternate form:

$$\frac{1}{2}\, i\, e^{-i x} - \frac{1}{2}\, i\, e^{i x} = \frac{1}{2}\, i \log(1 - i x) - \frac{1}{2}\, i \log(1 + i x)$$

$\log(x)$ is the natural logarithm »

Integer solution: Step-by-step solution

$x = 0$

Numerical solution: More digits

$x \approx \pm 1.55708581552472\ldots$

Computed by Wolfram *Mathematica* Download page

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL⟨ICP⟩

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
  $B \leftarrow S$.pop() ;
  while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
    //Pruning without branching, used as the assert() function.
    $B \leftarrow \text{Prune}(B, c_i)$;
  end
  //The ε below is computed from δ and the Lipschitz constants of functions beforehand.
  if $B \neq \emptyset$ **then**
    if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
      $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; //Splitting on the intervals
      S.push($\{B_1, B_2\}$);
    else
      return δ-sat; //Complete check() is successful.
    end
  end
end
return unsat;

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL⟨ICP⟩

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
    $B \leftarrow S$.pop() ;
    while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
        `//Pruning without branching, used as the assert() function.`
        $B \leftarrow \text{Prune}(B, c_i)$;
    end
    `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
    if $B \neq \emptyset$ **then**
        if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
            $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; `//Splitting on the intervals`
            S.push($\{B_1, B_2\}$);
        else
            return δ-sat; `//Complete check() is successful.`
        end
    end
end
return unsat;

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL⟨ICP⟩

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
  $B \leftarrow S$.pop() ;
  while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
    `//Pruning without branching, used as the assert() function.`
    $B \leftarrow \text{Prune}(B, c_i)$;
  end
  `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
  if $B \neq \emptyset$ **then**
    if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
      $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; `//Splitting on the intervals`
      S.push($\{B_1, B_2\}$);
    else
      return δ-sat; `//Complete check() is successful.`
    end
  end
end
return unsat;

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL⟨ICP⟩

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
  $B \leftarrow S$.pop() ;
  while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
    `//Pruning without branching, used as the assert() function.`
    $B \leftarrow \text{Prune}(B, c_i)$;
  end
  `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
  if $B \neq \emptyset$ **then**
    if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
      $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; `//Splitting on the intervals`
      S.push($\{B_1, B_2\}$);
    else
      return δ-sat; `//Complete check() is successful.`
    end
  end
end
return unsat;

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL$\langle$ICP$\rangle$

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
    $B \leftarrow S$.pop() ;
    while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
        `//Pruning without branching, used as the assert() function.`
        $B \leftarrow \text{Prune}(B, c_i)$;
    end
    `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
    if $B \neq \emptyset$ **then**
        if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
            $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; `//Splitting on the intervals`
            S.push($\{B_1, B_2\}$);
        else
            return δ-sat; `//Complete check() is successful.`
        end
    end
end
return unsat;

ICP in **dReal**

Algorithm 1: Theory Solving in DPLL$\langle$ICP$\rangle$

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.

output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
    $B \leftarrow S$.pop() ;
    while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
        `//Pruning without branching, used as the assert() function.`
        $B \leftarrow \text{Prune}(B, c_i)$;
    end
    `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
    if $B \neq \emptyset$ **then**
        if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
            $\{B_1, B_2\} \leftarrow$ Branch(B, i); `//Splitting on the intervals`
            S.push($\{B_1, B_2\}$);
        else
            return δ-sat; `//Complete check() is successful.`
        end
    end
end
return unsat;

dReal Demo

Handling Differential Equations

An ODE system

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{f}(\vec{x}, t)$$

when put in Picard–Lindelöf form:

$$\vec{x}_t = \vec{x}_0 + \int_0^t f(\vec{x}, s)\mathrm{d}s$$

is seen as a **constraint** between $\vec{x}_0, \vec{x}_t$, and t.

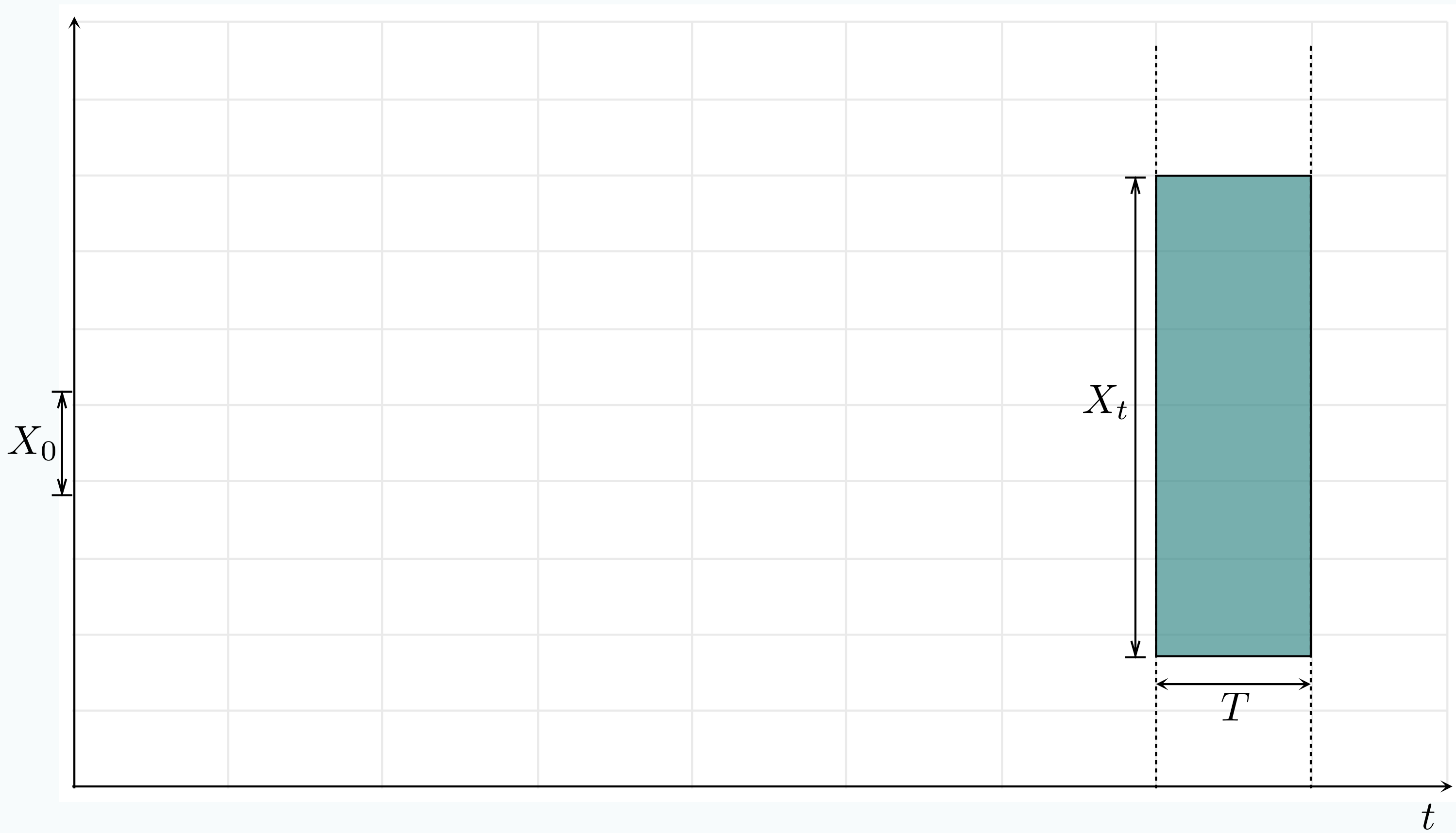
ODE Pruning

Starting with big intervals for

$$\vec{x}_t, \vec{x}_0, t$$

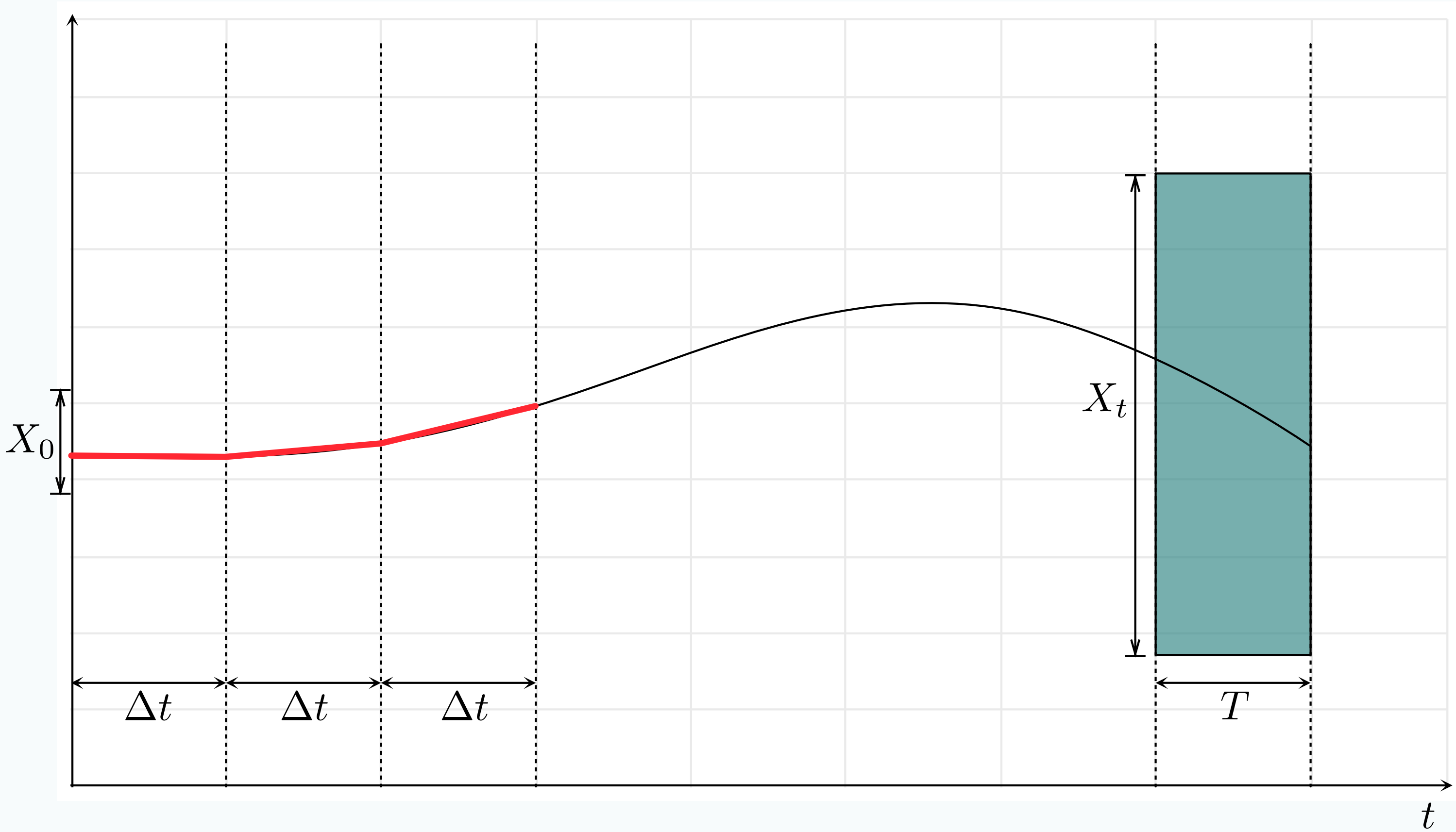
use the **ODE constraints** to find smaller intervals for them.

Forward Pruning (on X_t)



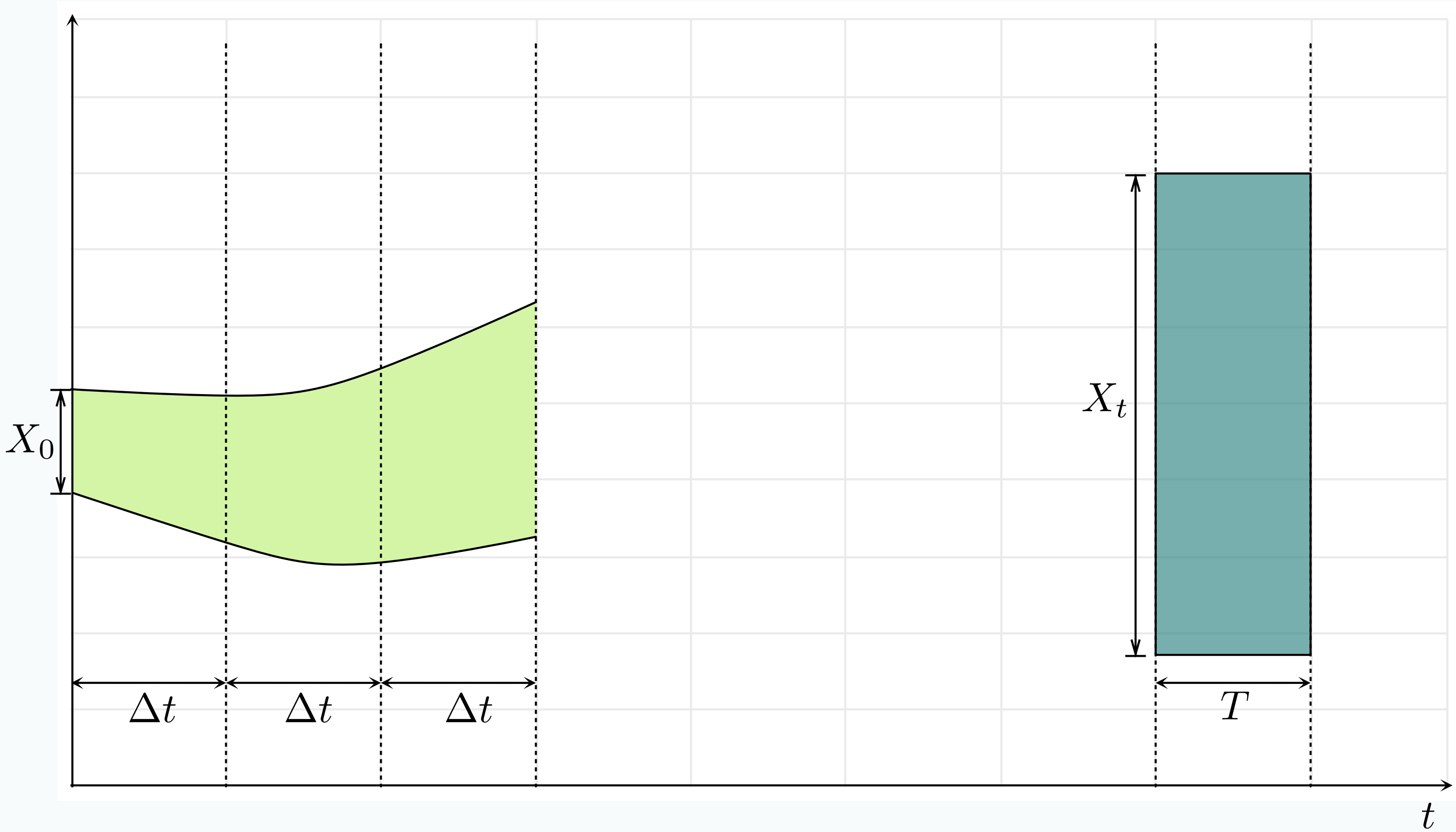
$\Rightarrow$

Forward Pruning (on X_t)



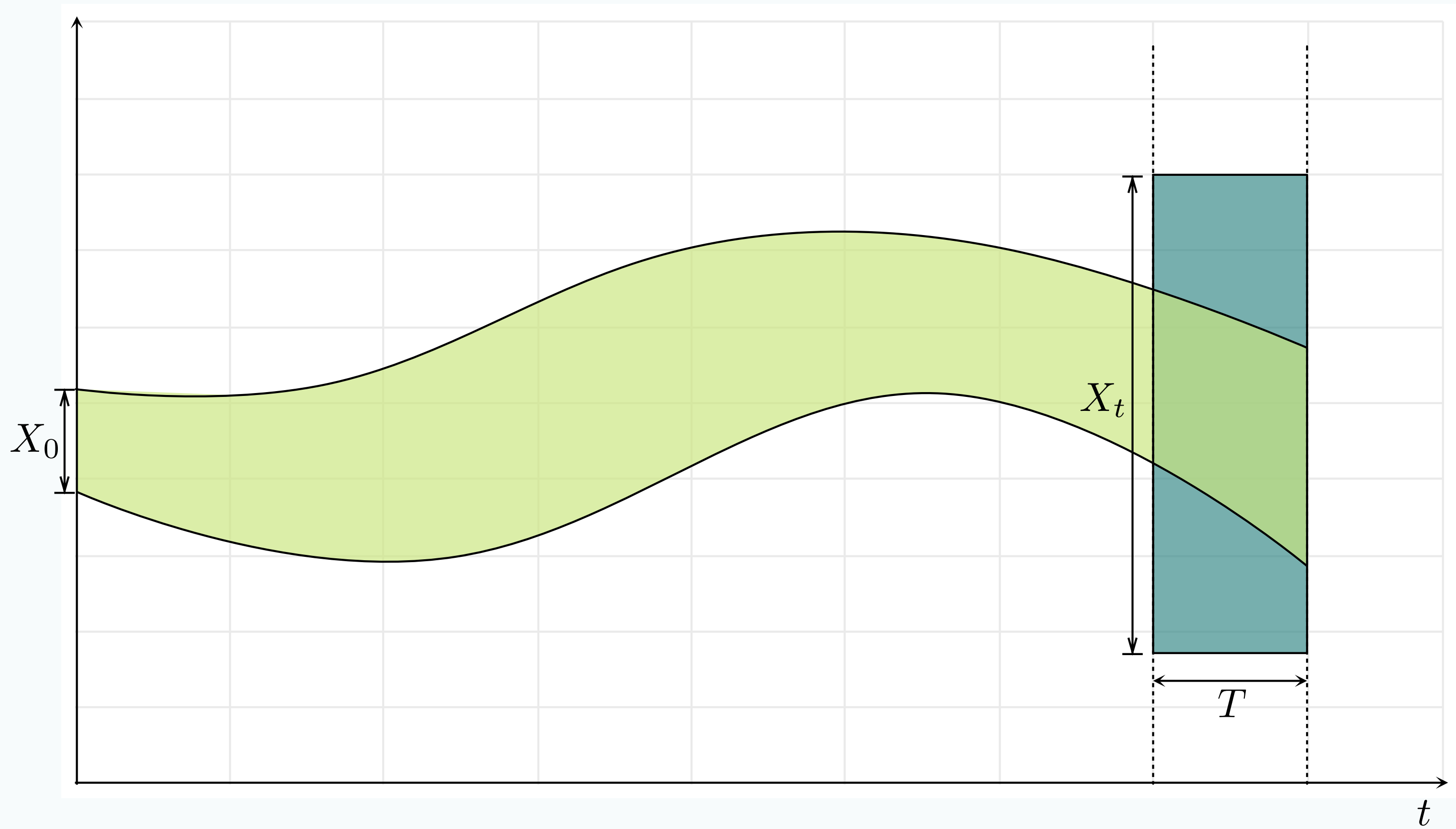
$\Rightarrow$

Forward Pruning (on X_t)



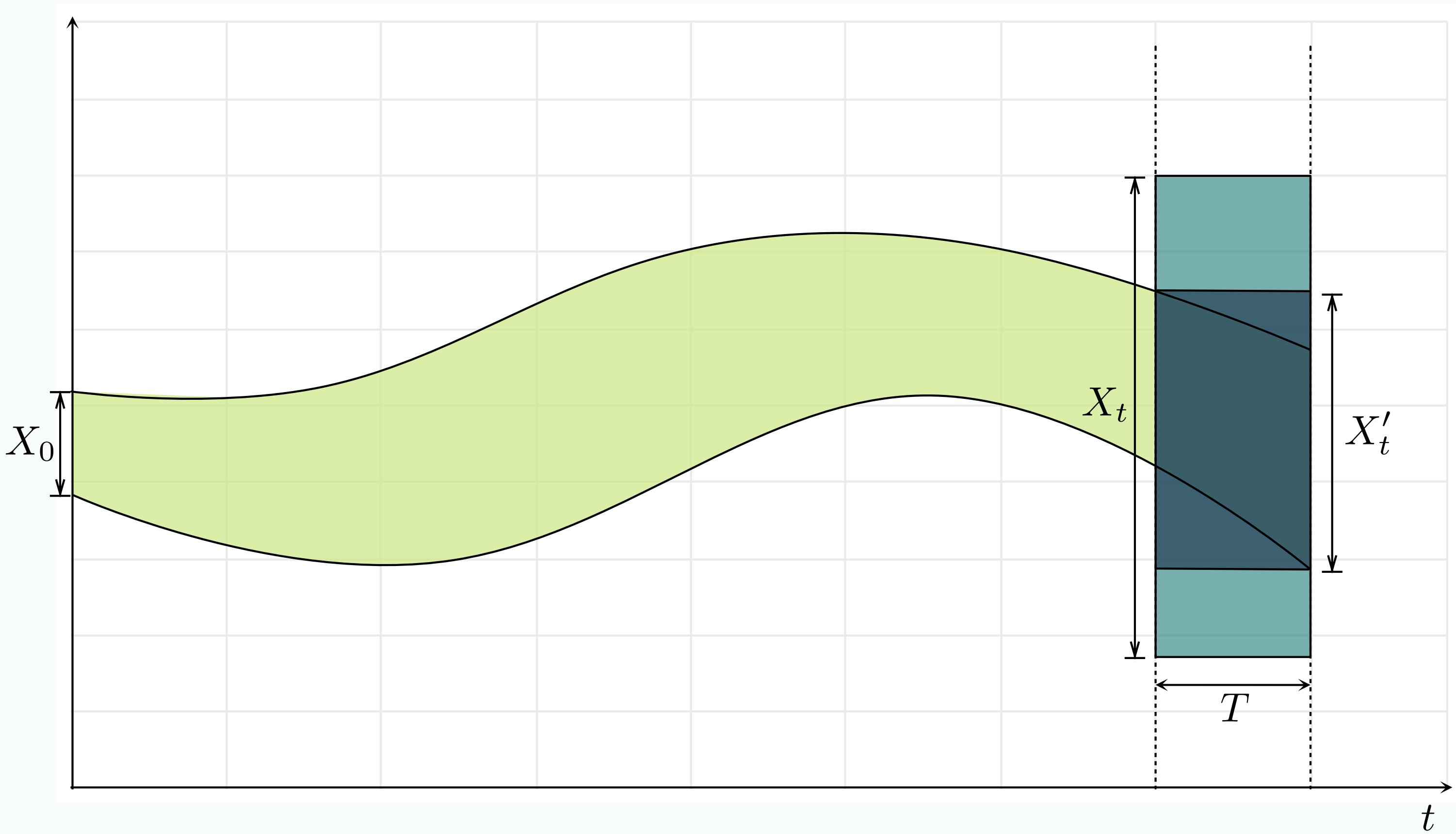
$\Rightarrow$

Forward Pruning (on X_t)



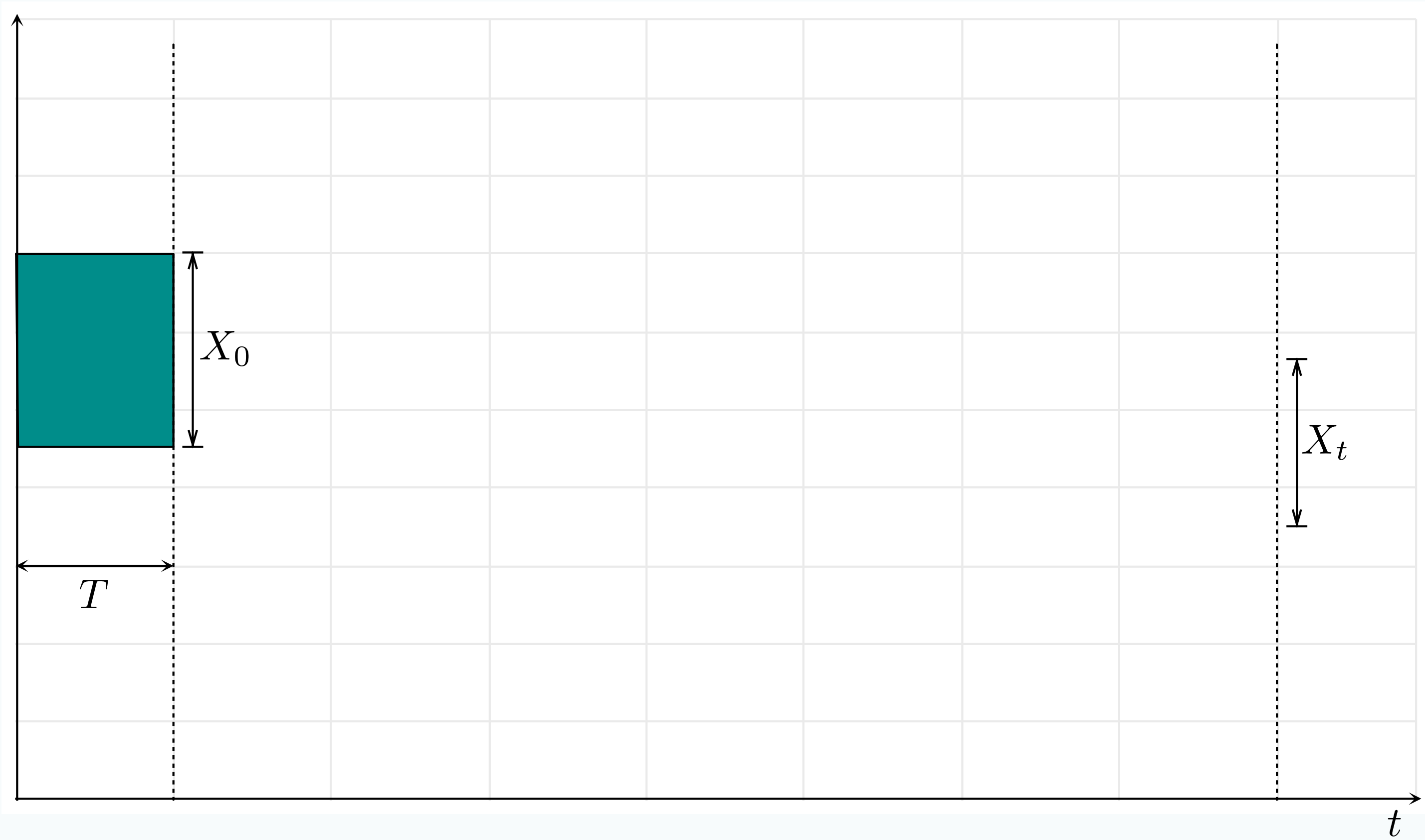
$\Rightarrow$

Forward Pruning (on X_t)



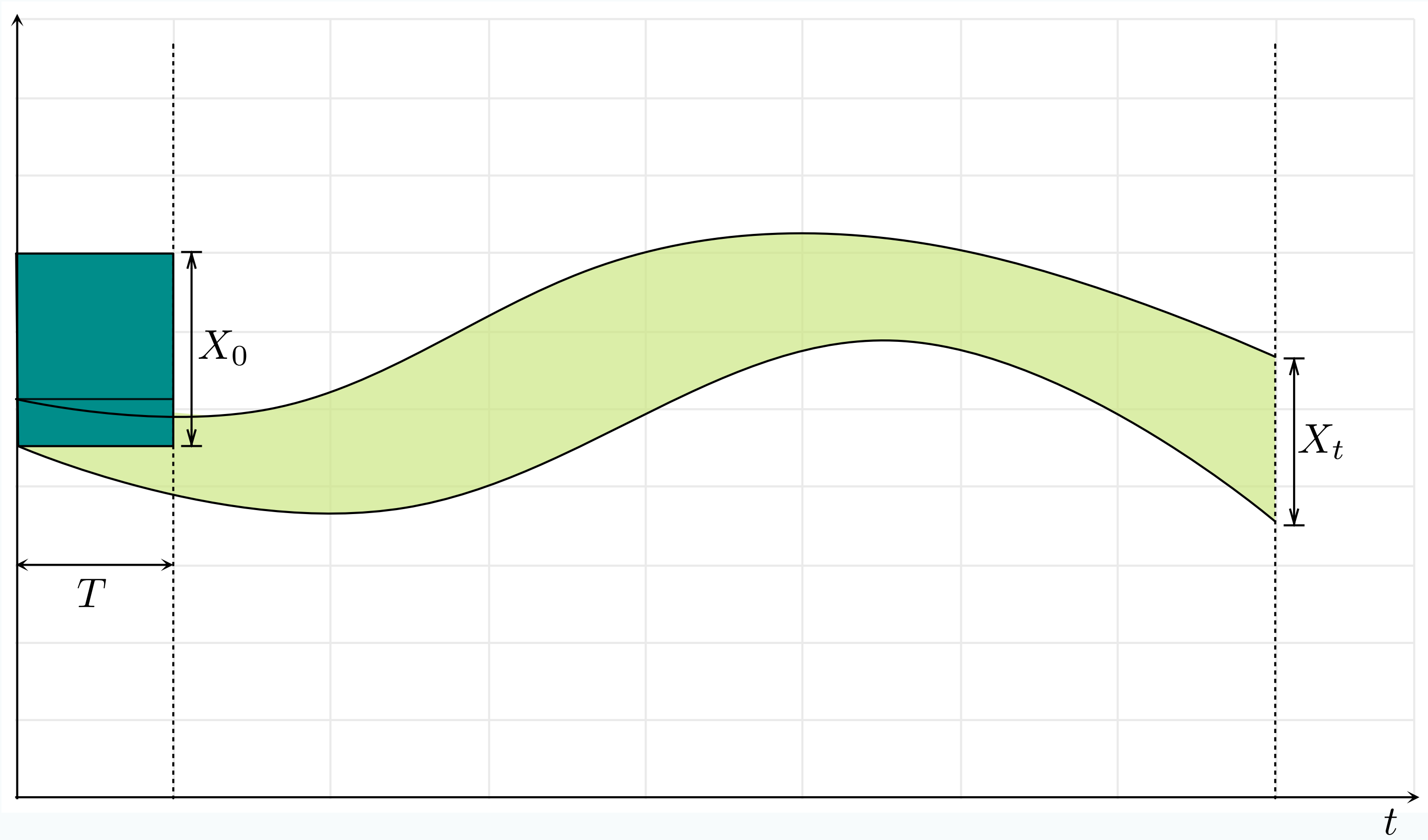
$\Rightarrow$

Backward Pruning (on X_0)



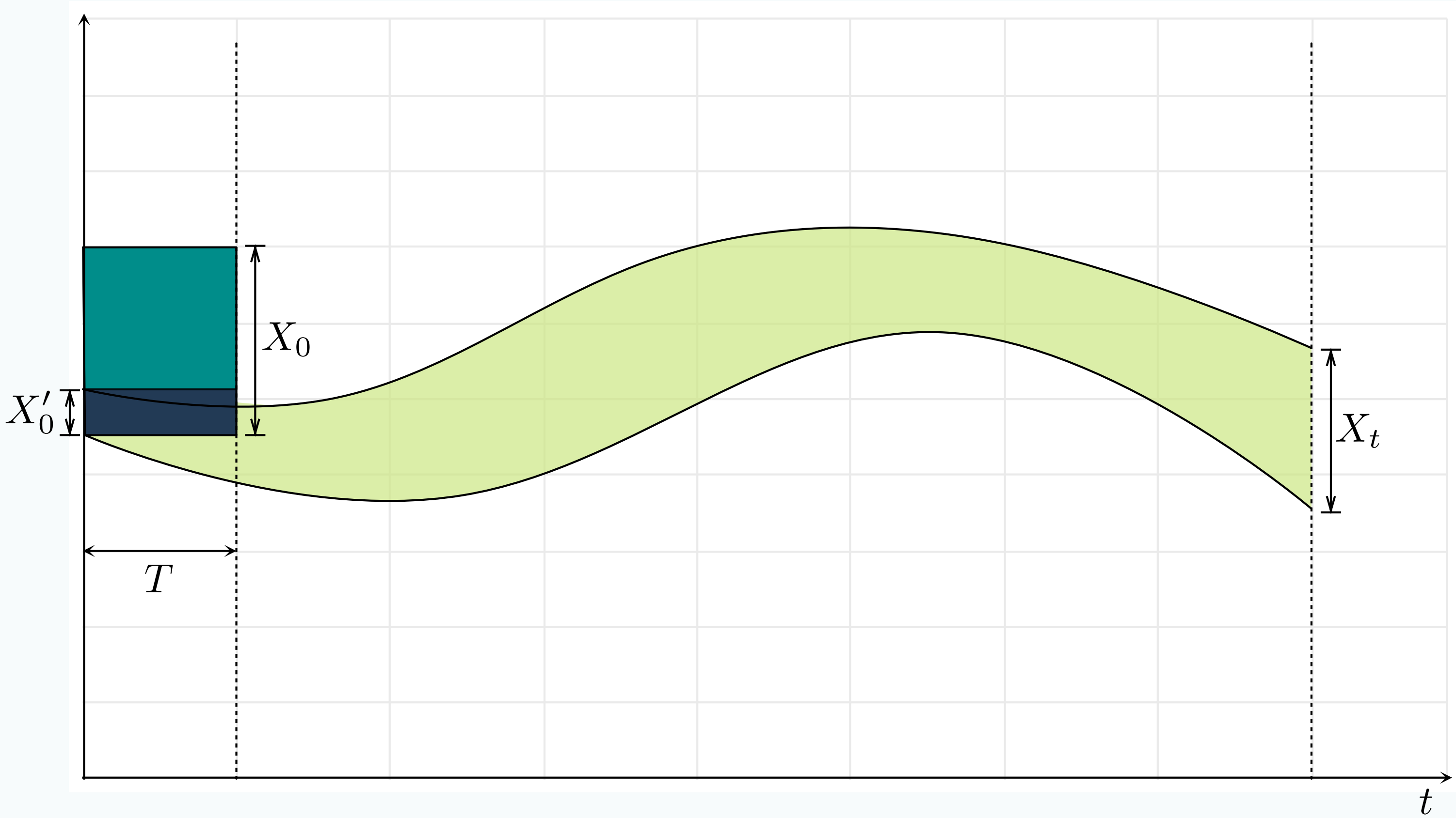
$\Leftarrow$

Backward Pruning (on $\boldsymbol{X_0}$)



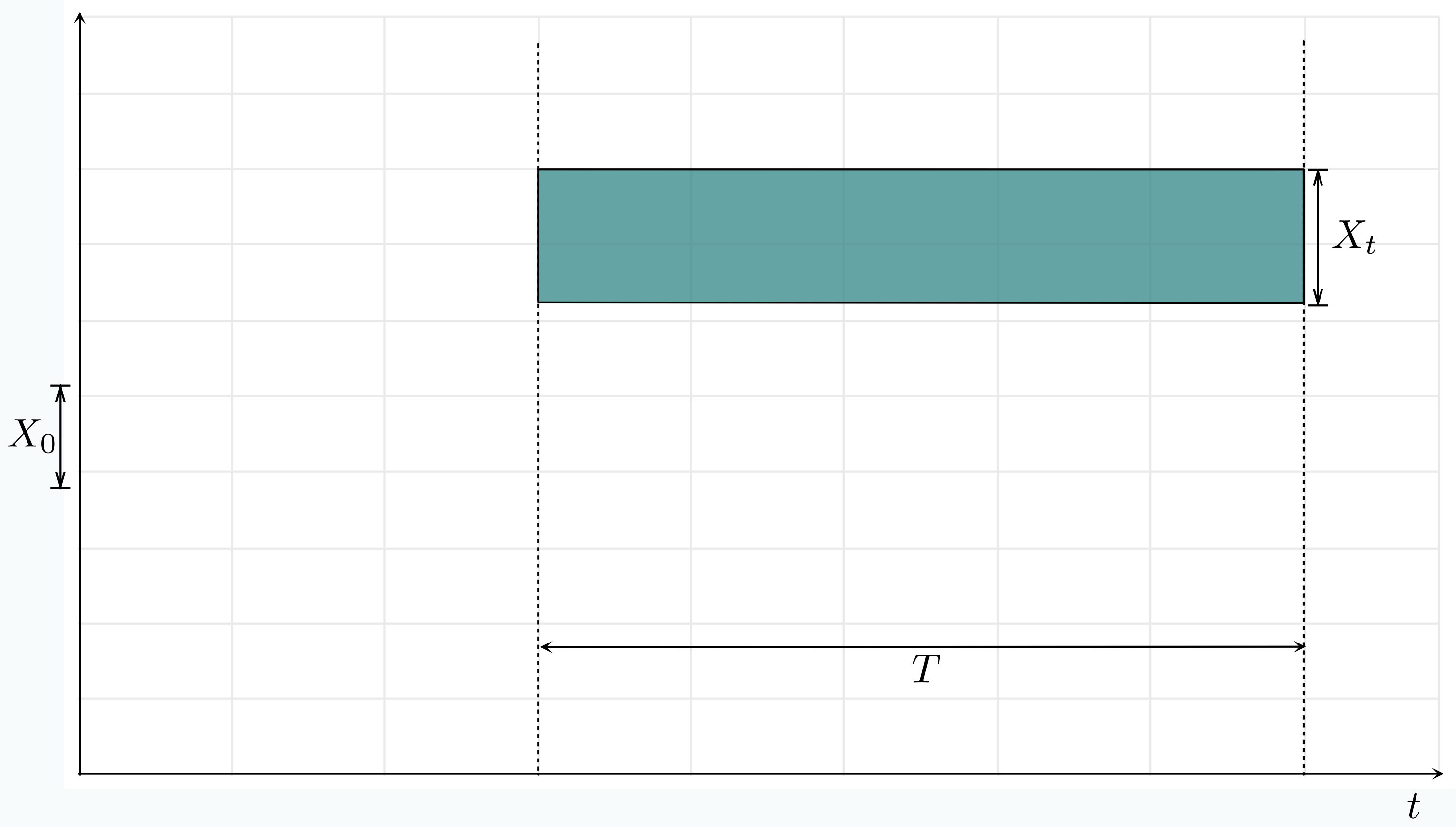
$\Leftarrow$

Backward Pruning (on $\boldsymbol{X_0}$)

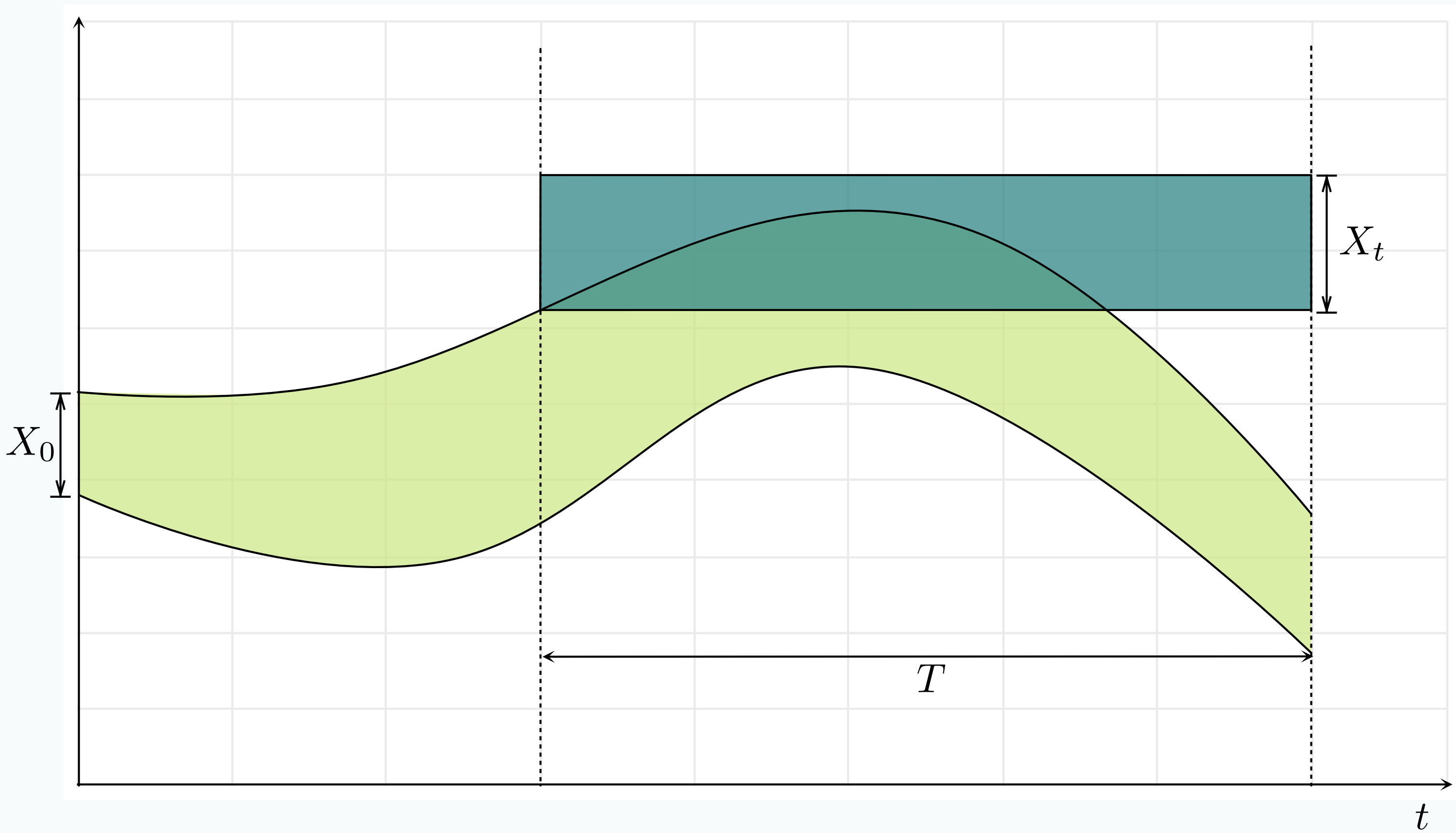


$\Leftarrow$

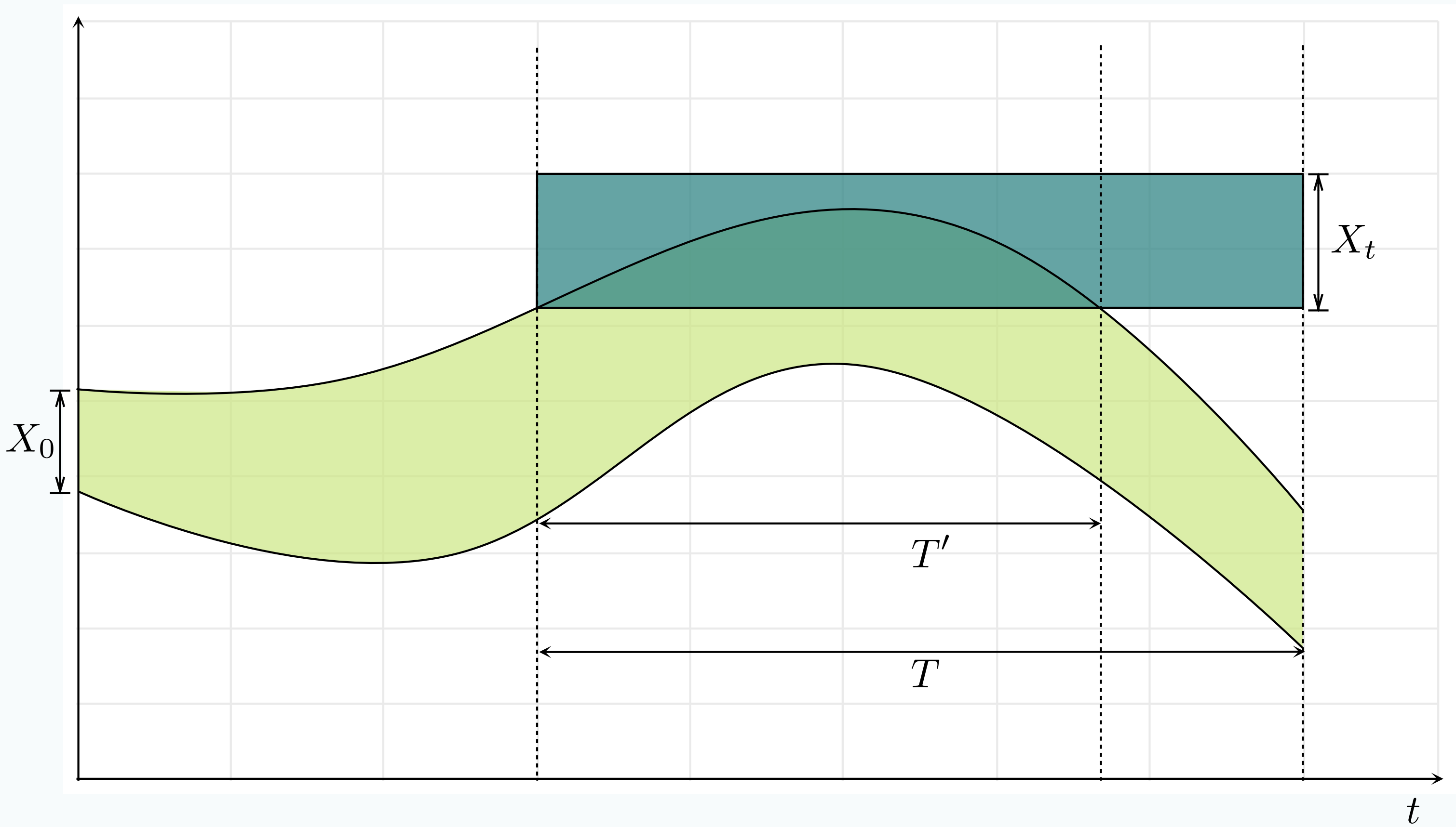
Time Pruning (on T)



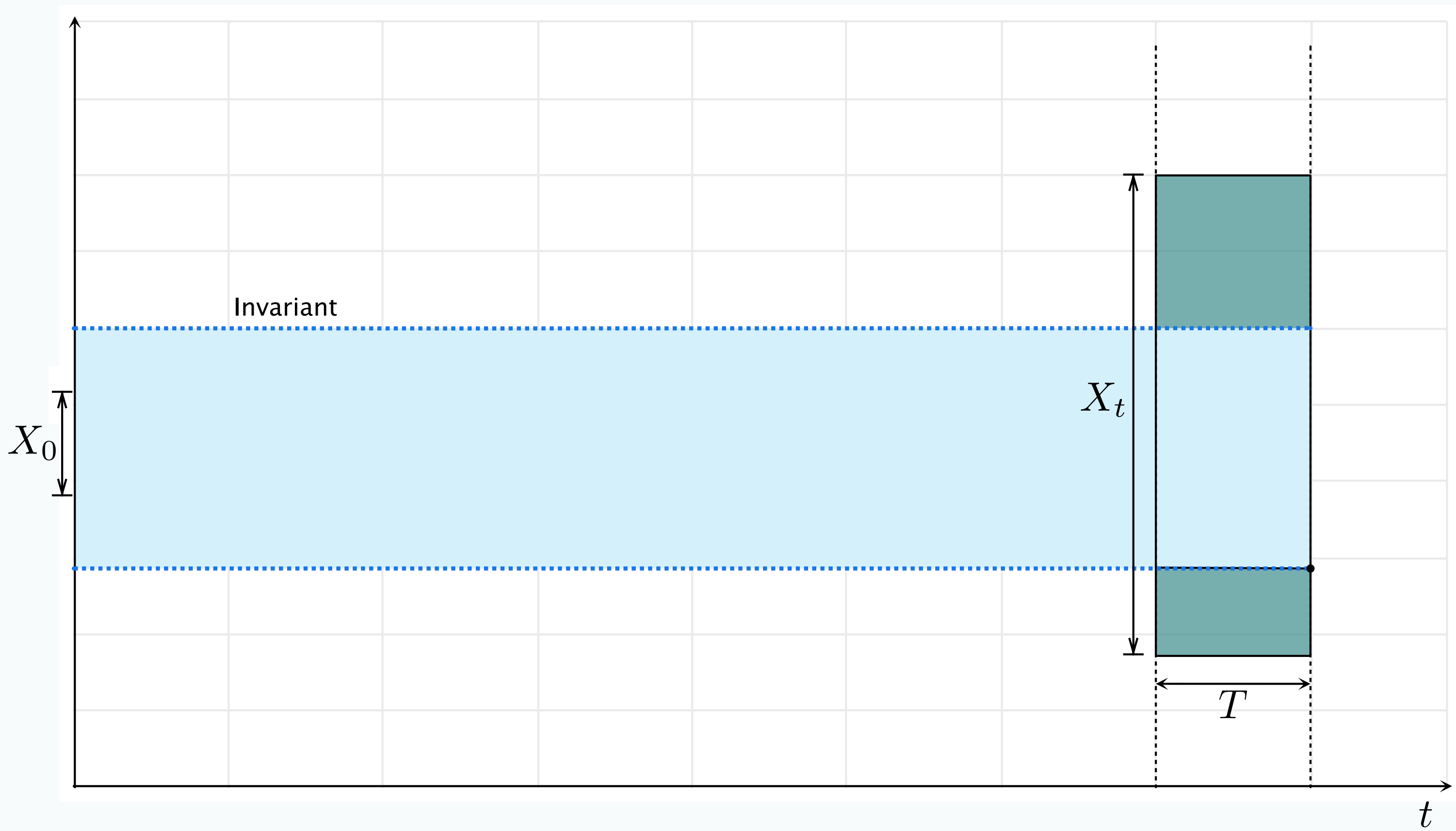
Time Pruning (on T)



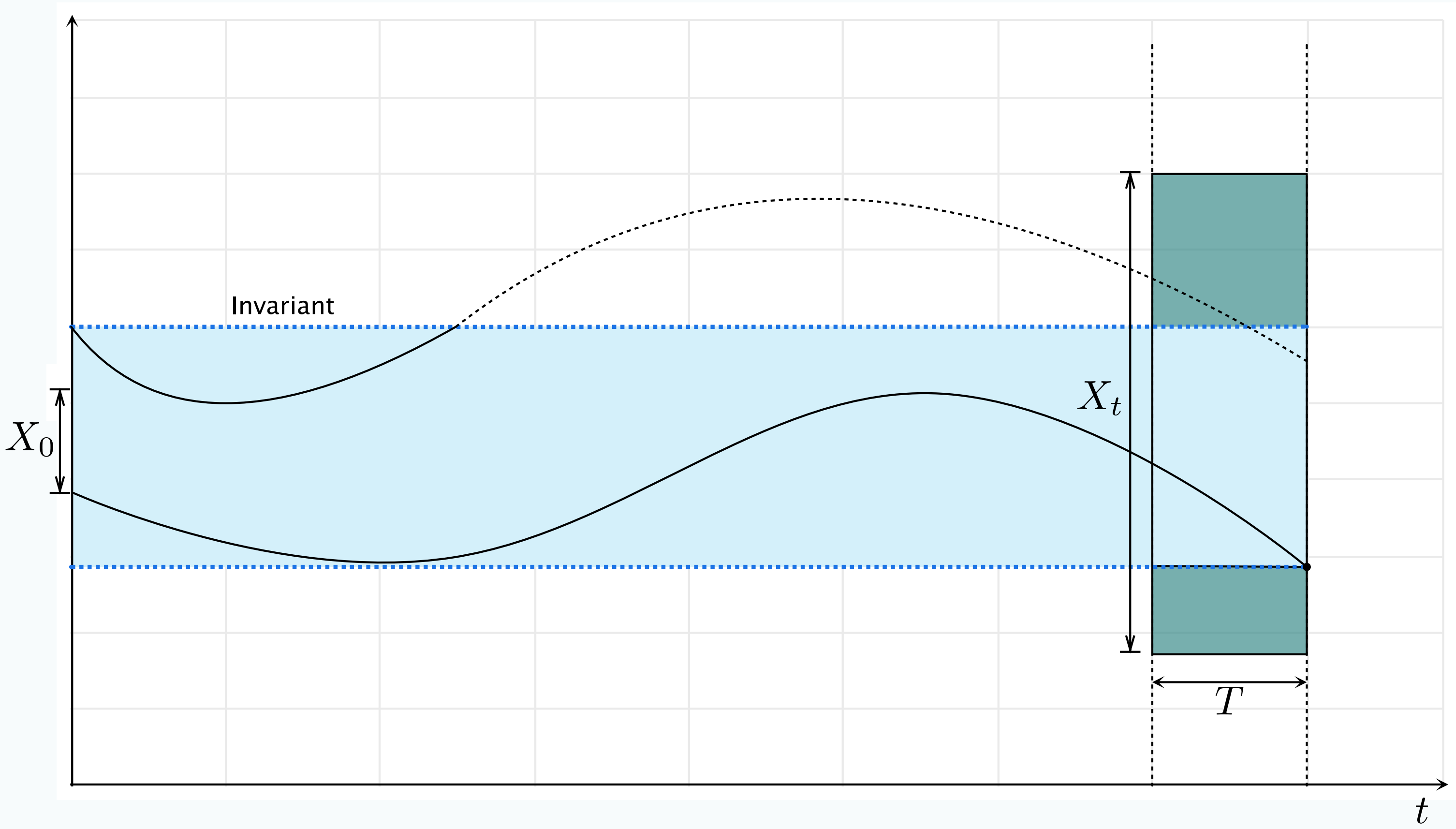
Time Pruning (on $\boldsymbol{T}$)



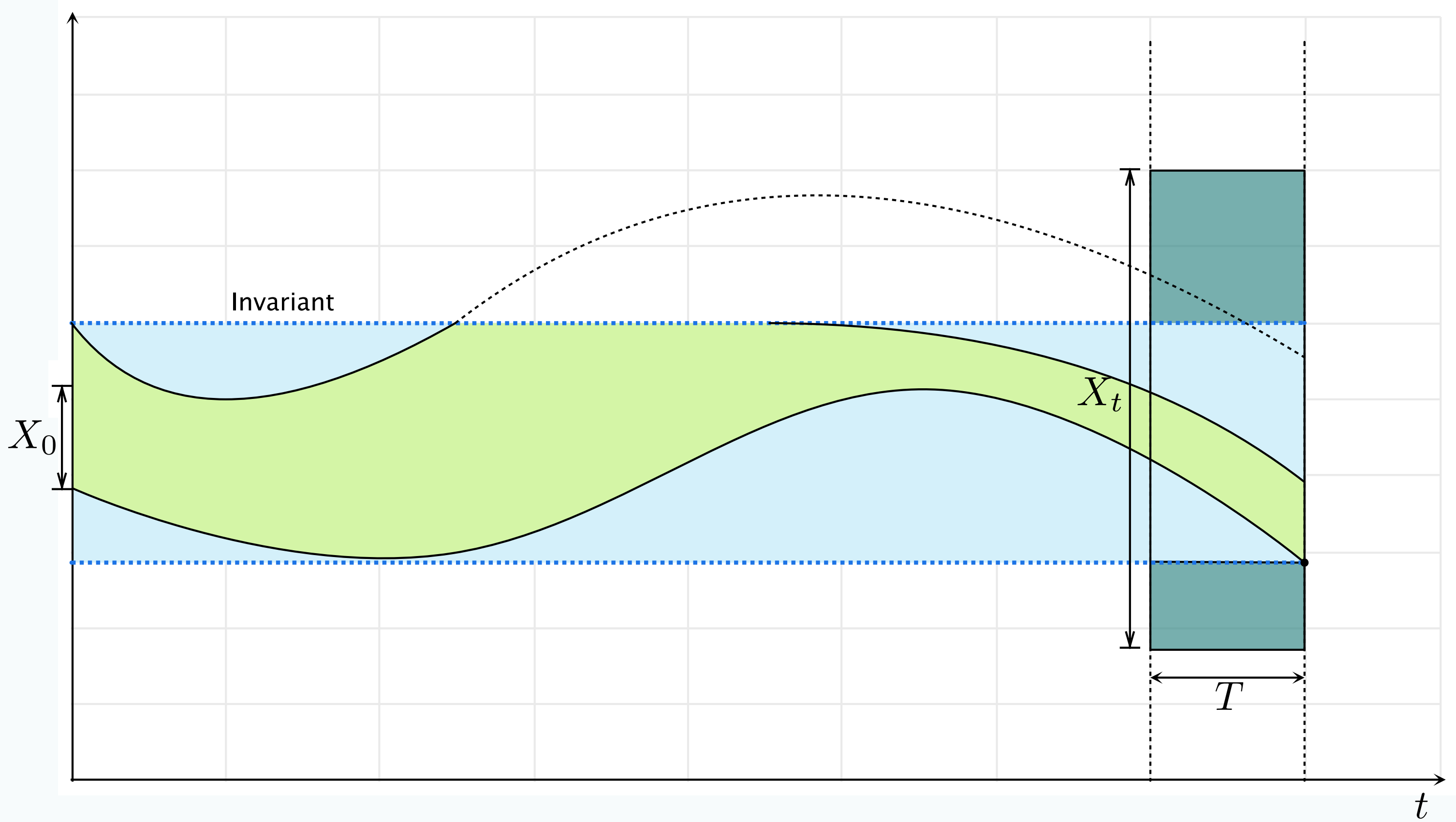
Pruning with Invariant



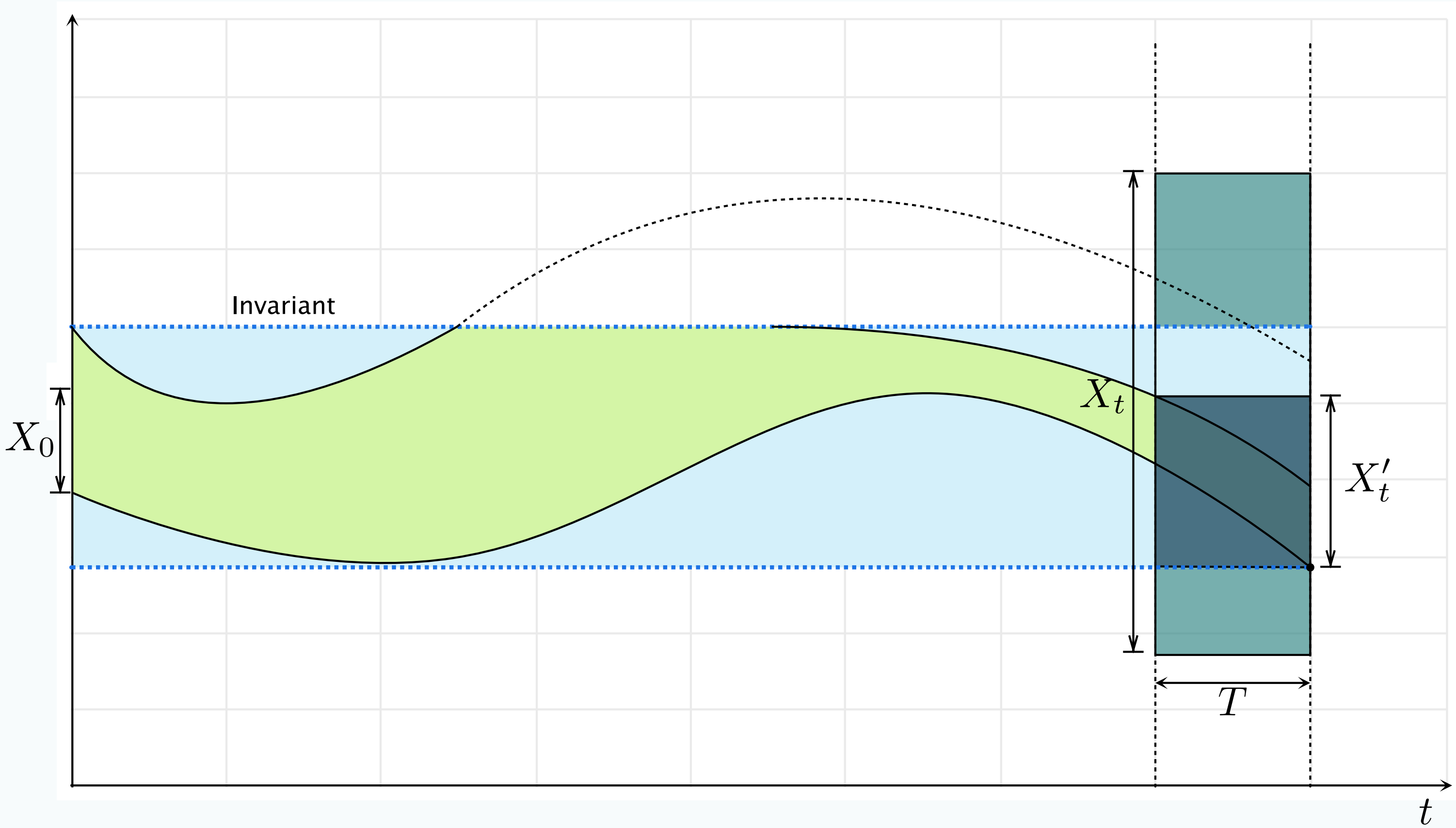
Pruning with Invariant



Pruning with Invariant



Pruning with Invariant



dReach

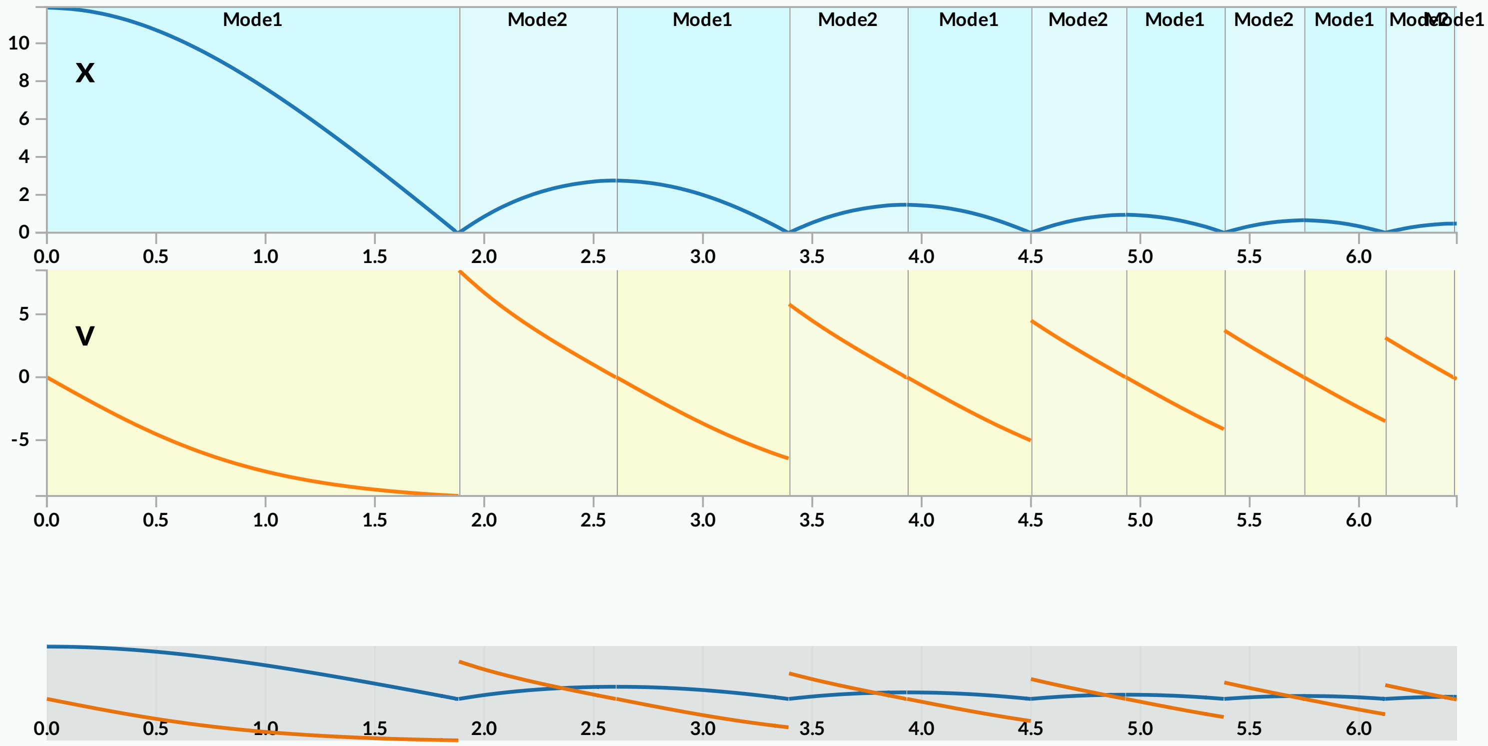
- Tool for safety verification of **hybrid systems**.
- Handle general hybrid systems with nonlinear differential equations and complex discrete mode-changes.
- Performs bounded delta-complete reachability analysis and uses dReal as a computation engine.

Experimental Results: Hybrid System Benchmark

Problem	# of Mode	# of Unrolling Depth	# of ODEs	# of variables	Eps	Result	Time	Size of Trace
Atrial Filbrillation	4	3	20	44	0.001	SAT	43.10	90K
Atrial Filbrillation	8	7	40	88	0.001	SAT	698.86	20M
Atrial Filbrillation	8	23	120	246	0.001	SAT	4528.13	59M
Atrial Filbrillation	8	31	160	352	0.001	SAT	8485.99	78M
Atrial Filbrillation	8	47	240	528	0.001	SAT	15740.41	117M
Atrial Filbrillation	8	55	280	616	0.001	SAT	19989.59	137M
Prostate Cancer	2	2	15	36	0.005	SAT	345.84	3.1M
Prostate Cancer	2	2	15	36	0.002	SAT	362.84	3.1M
Electronic Oscillator	3	2	18	42	0.01	SAT	52.93	998K
Electronic Oscillator	3	2	18	42	0.001	SAT	57.67	847K
Electronic Oscillator	3	11	72	168	0.01	UNSAT	7.75	--
Bouncing Ball	2	10	22	66	0.01	SAT	0.25	123K
Bouncing Ball	2	20	42	126	0.01	SAT	0.57	171K
Bouncing Ball	2	20	42	126	0.001	SAT	2.21	168K
Bouncing Ball	2	40	82	246	0.01	UNSAT	0.27	----
Bouncing Ball	2	40	82	246	0.001	UNSAT	0.26	----
Decay Model	3	2	9	24	0.1	SAT	30.84	72K

Experimental Results: Hybrid System Benchmark

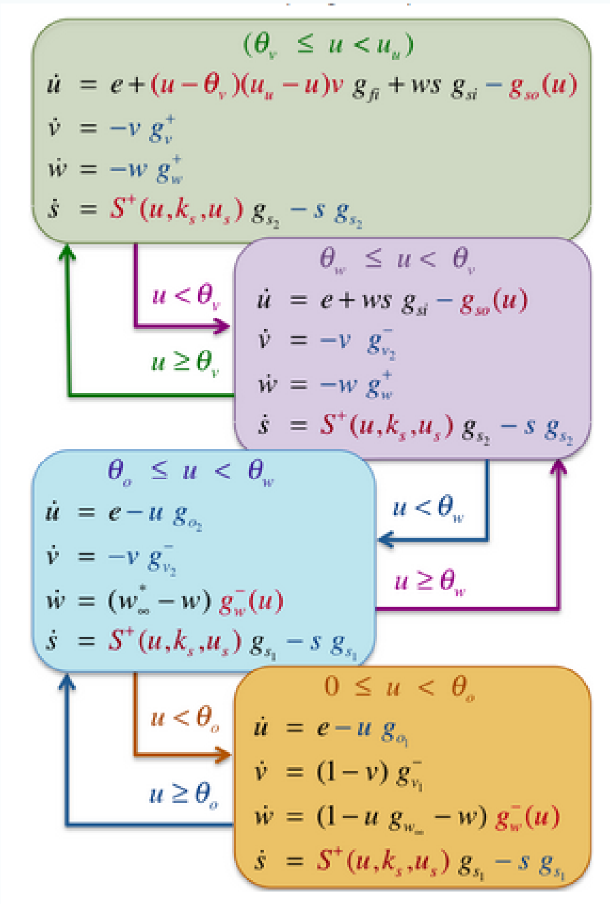
Bouncing Ball



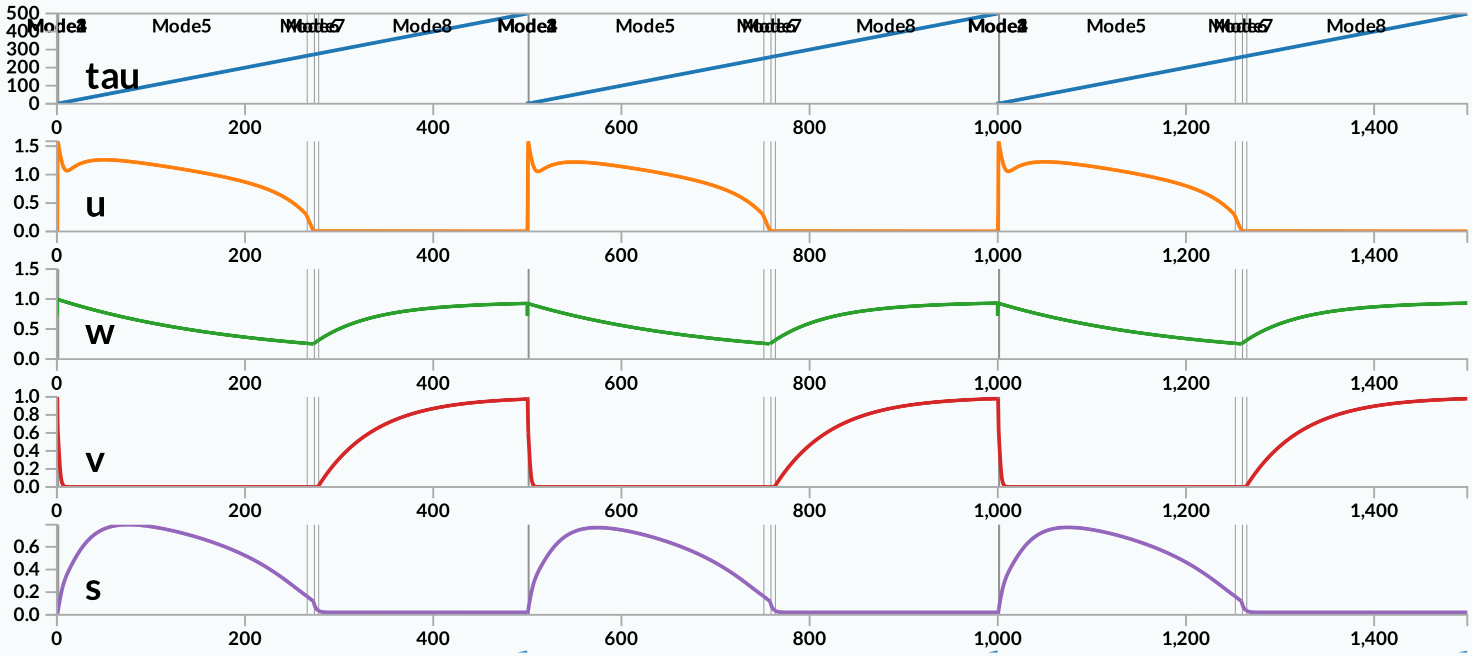
Click and drag above to zoom / pan the data

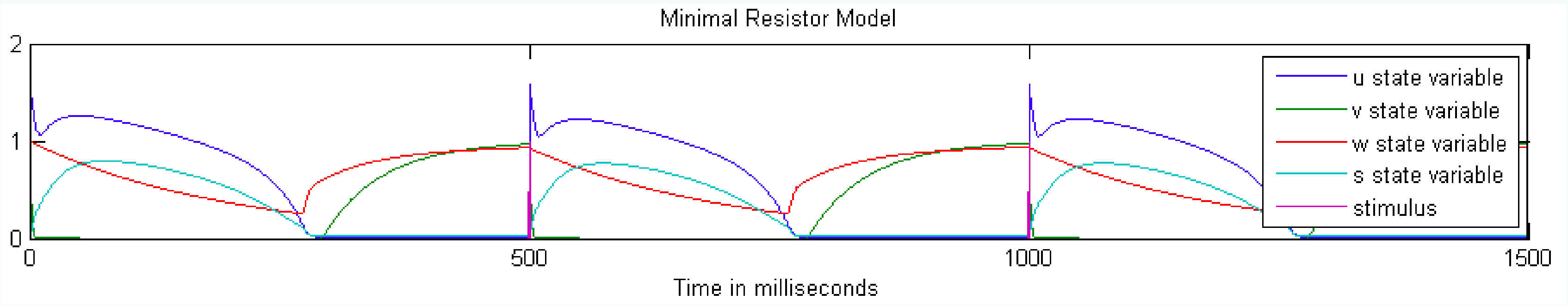
Experimental Results: Hybrid System Benchmark

- Including **Atrial Filbrillation** Model: (R. Groso et al, "From Cardiac Cells to Genetic Regulatory Networks", CAV'11)



Experimental Results: Hybrid System Benchmark





dReach Demo

Conclusion

- **dReal** is an **δ-complete** SMT solver
- **dReach** is a tool for safety verification of **hybrid systems**.
- Support **nonlinear real functions** such as

$$\sin, \cos, \tan, \arcsin, \arccos, \arctan, \log, \exp, \ldots$$

- Handle **ODEs** (Ordinary Differential Equations)
- Based on **DPLL$\langle$ICP$\rangle$** framework
- **Scalable** with our experiments
- **Open-source**: available at **http://dreal.cs.cmu.edu**