$\mathbb{R}$eal Problems on the Road

Soonho Kong
soonho.kong@tri.global

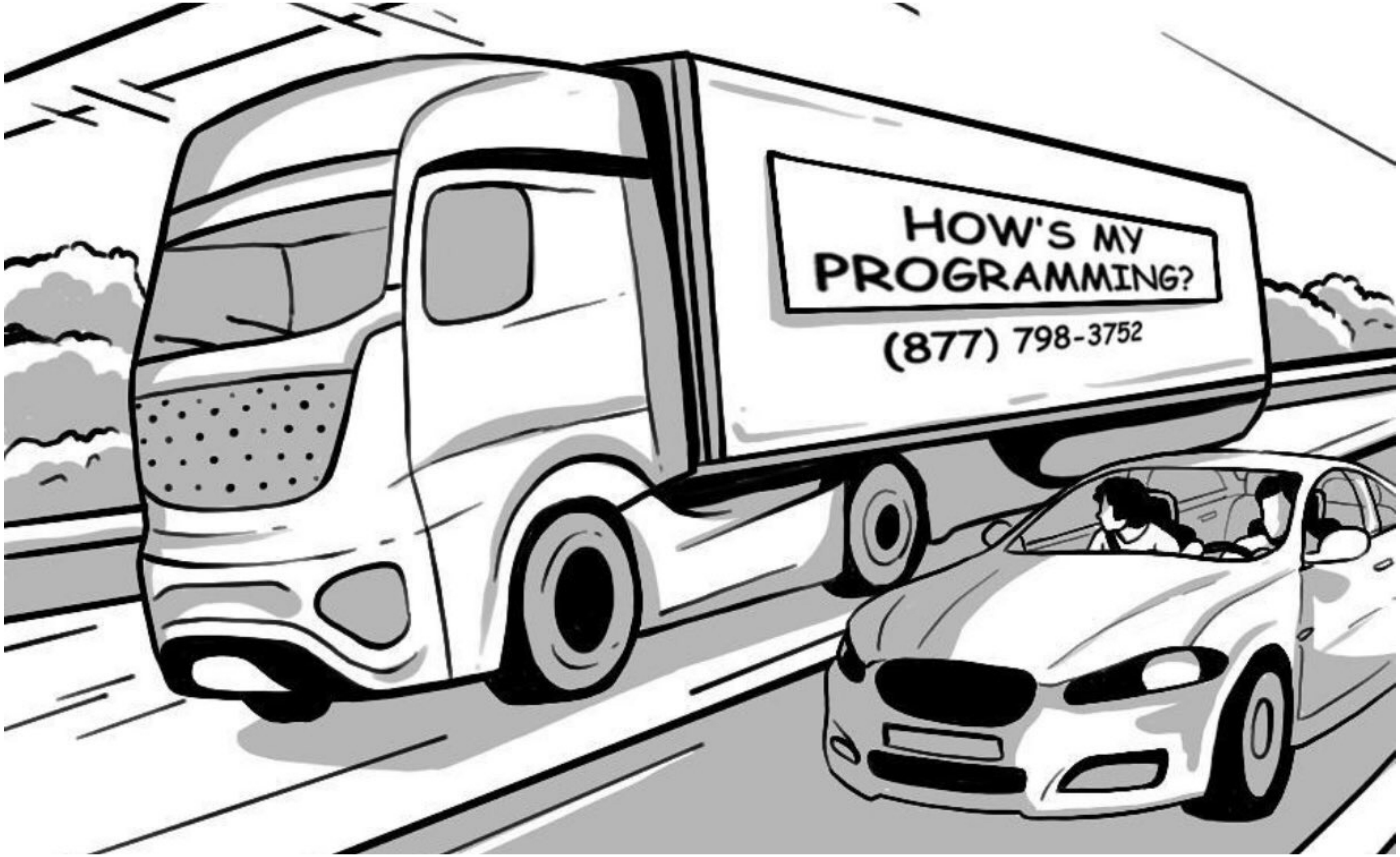
June 23, 2018

Applications of Formal Methods to
Control Theory and Dynamical Systems

Carnegie Mellon University

1. Problem: How to show that our system is safe?

How do we know that our system is safe?



How do we know that our system is safe?

Road Testing?

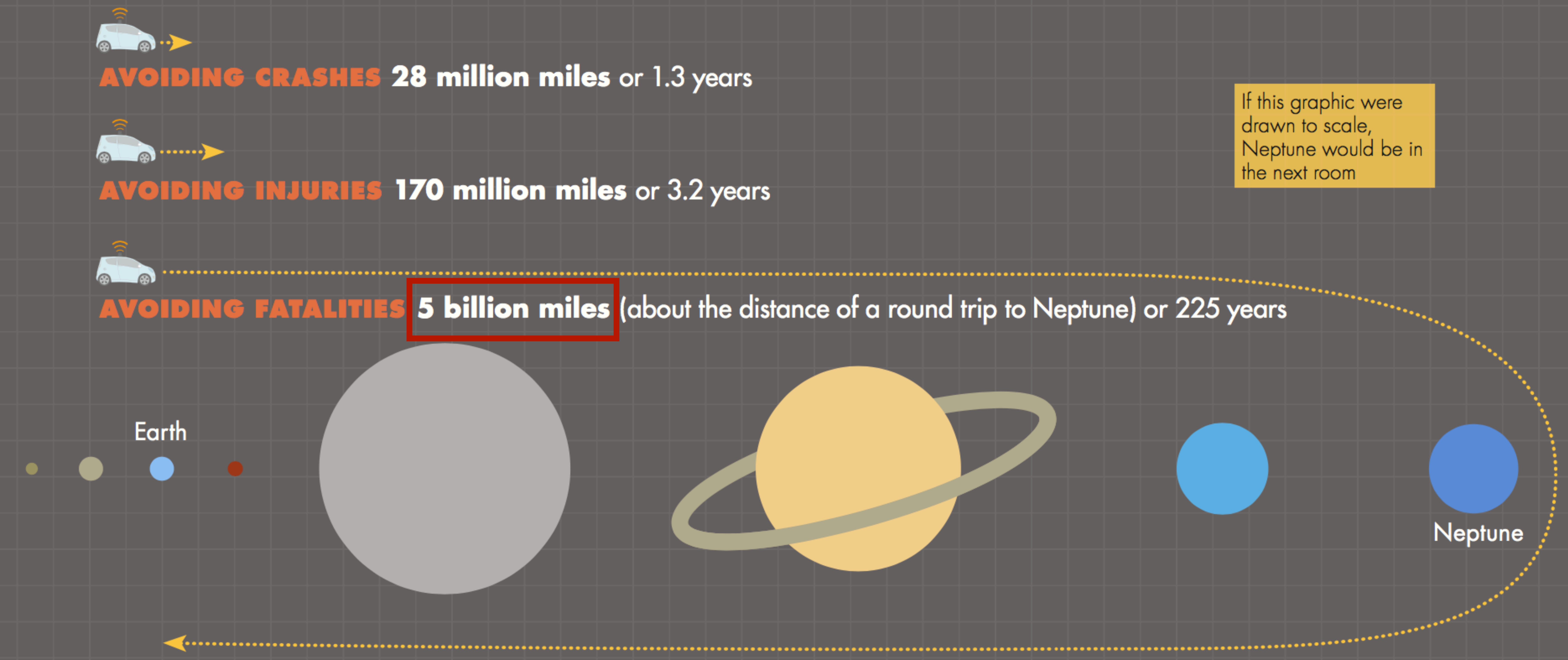
Driving to Safety

How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?

Nidhi Kalra, Susan M. Paddock

https://www.rand.org/pubs/research_reports/RR1478.html

AVs WOULD HAVE TO BE TEST-DRIVEN ASTRONOMICAL DISTANCES
TO SHOW THEY ARE 20 PERCENT BETTER THAN HUMAN DRIVERS AT . . .



NOTE: Confidence interval = 95%. Times provided assume a fleet of 100 test AVs driving continuously at an average of 25 mph. Crash and injury rates used here include police-reported incidents and estimates of unreported incidents.

GREEN CAR: MONICAODO/GETTYIMAGES; SMALL CAR: CHOMBOSAN/FOTOLIA

2015 fatality data are from the Bureau of Transportation Statistics

RAND CORPORATION

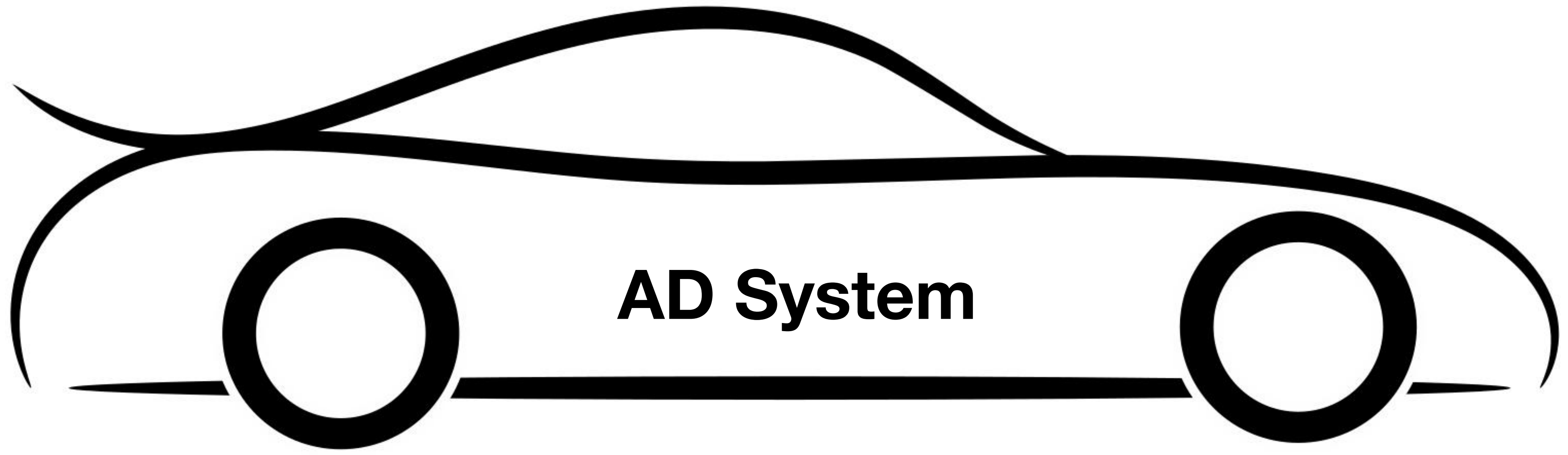
Adapted from *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* by Nidhi Kalra and Susan M. Paddock, RR-1478-RC, 2016, available at www.rand.org/t/RR1478. The RAND Corporation is a research organization that develops solutions to public policy challenges to help make communities throughout the world safer and more secure, healthier and more prosperous. RAND is nonprofit, nonpartisan, and committed to the public interest.

www.rand.org

IG-128

https://www.rand.org/pubs/research_reports/RR1478.html

How to accelerate the testing process?


AD System


AD System
Plant / Physics Model


Traffic
Behavior Model
AD System
Plant / Physics Model


Road Objects
Traffic
Behavior Model
AD System
Plant / Physics Model

Need a simulator to test a system

Automotive and Transportation

Accelerate innovation using Unity’s real-time rendering platform plus PiXYZ’s best-in-class CAD data solutions, to empower your teams. Bring great ideas and products to life.

Talk to a Unity expert

AutoTech summit

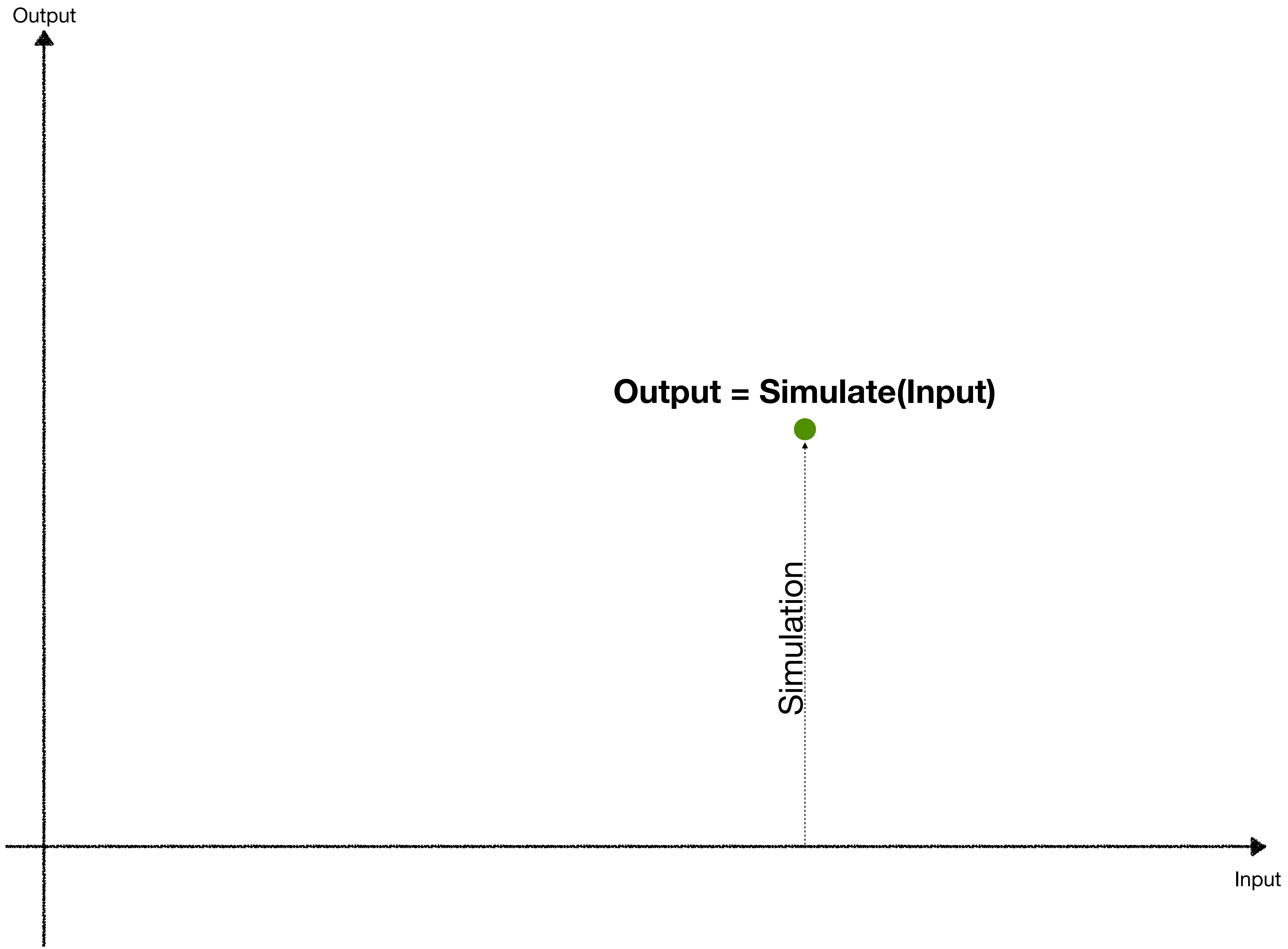

Rendered in real-time with Unity

CAN YOU SIMULATE THE WORLD?

WE CAN.

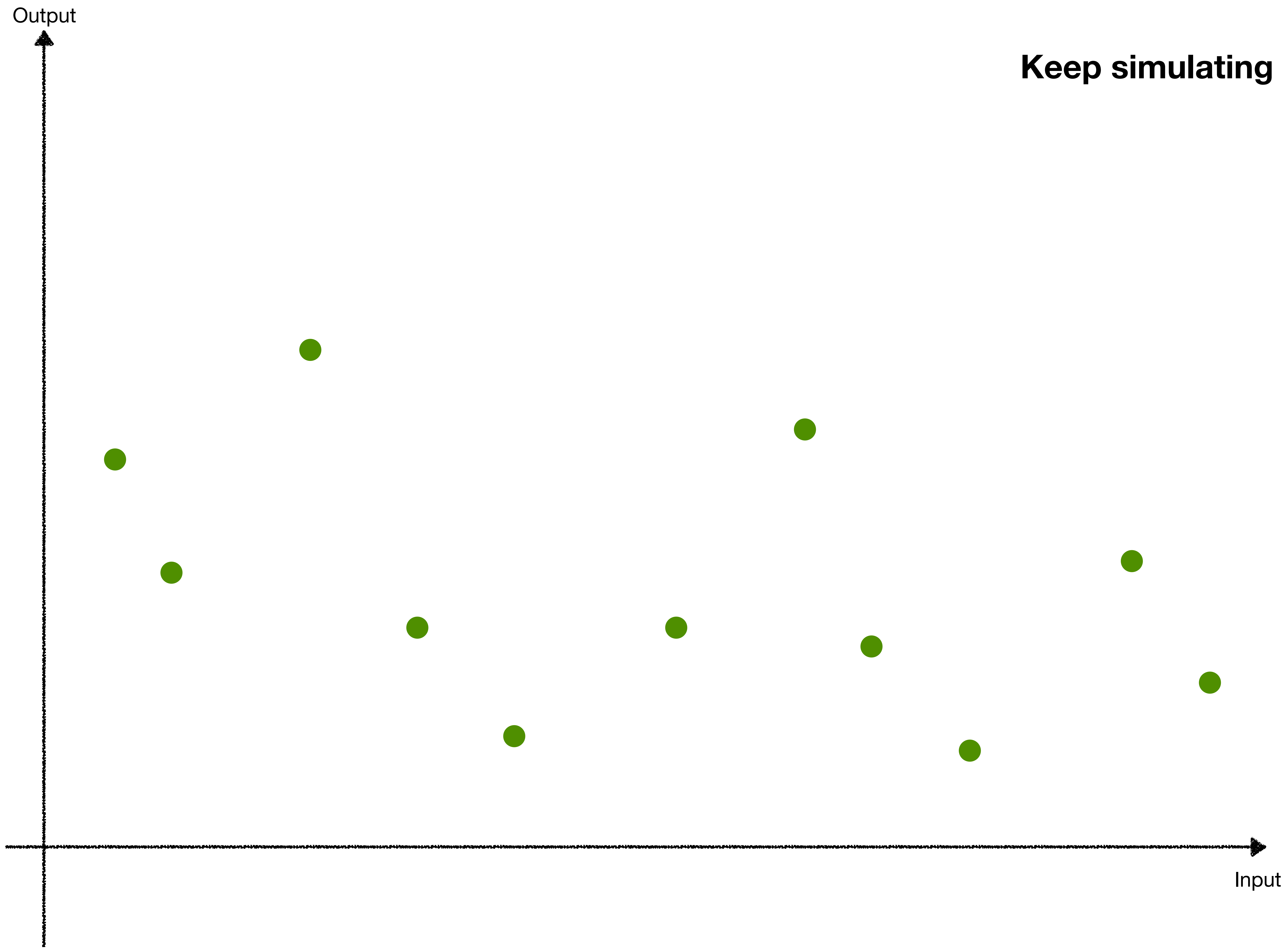

Devonshire St
Sir Francis Drake Blvd
EXIT
STOP AHEAD
SYNCITY
PRG6: Essentials
FRM 3975
Speed
64.1 km/h
Gear
3
RPM
2917.2 rpm
Steering
0 %
Brake
0 %
Throttle
94 %
60
70
80
RGB

CARLA: Open-source simulator for autonomous driving research

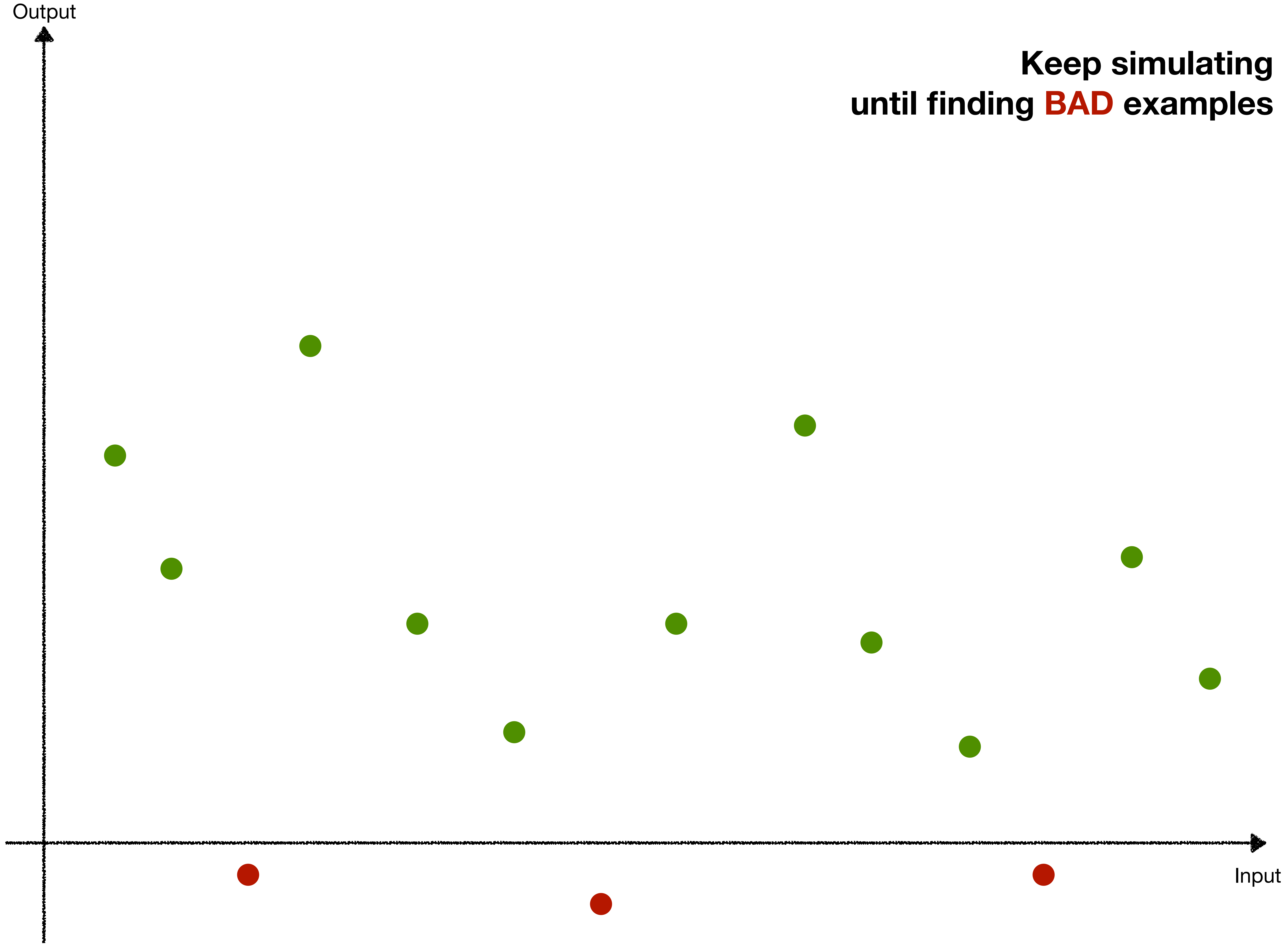


Simulation = Black-Box Testing

Keep simulating



Simulation = Black-Box Testing



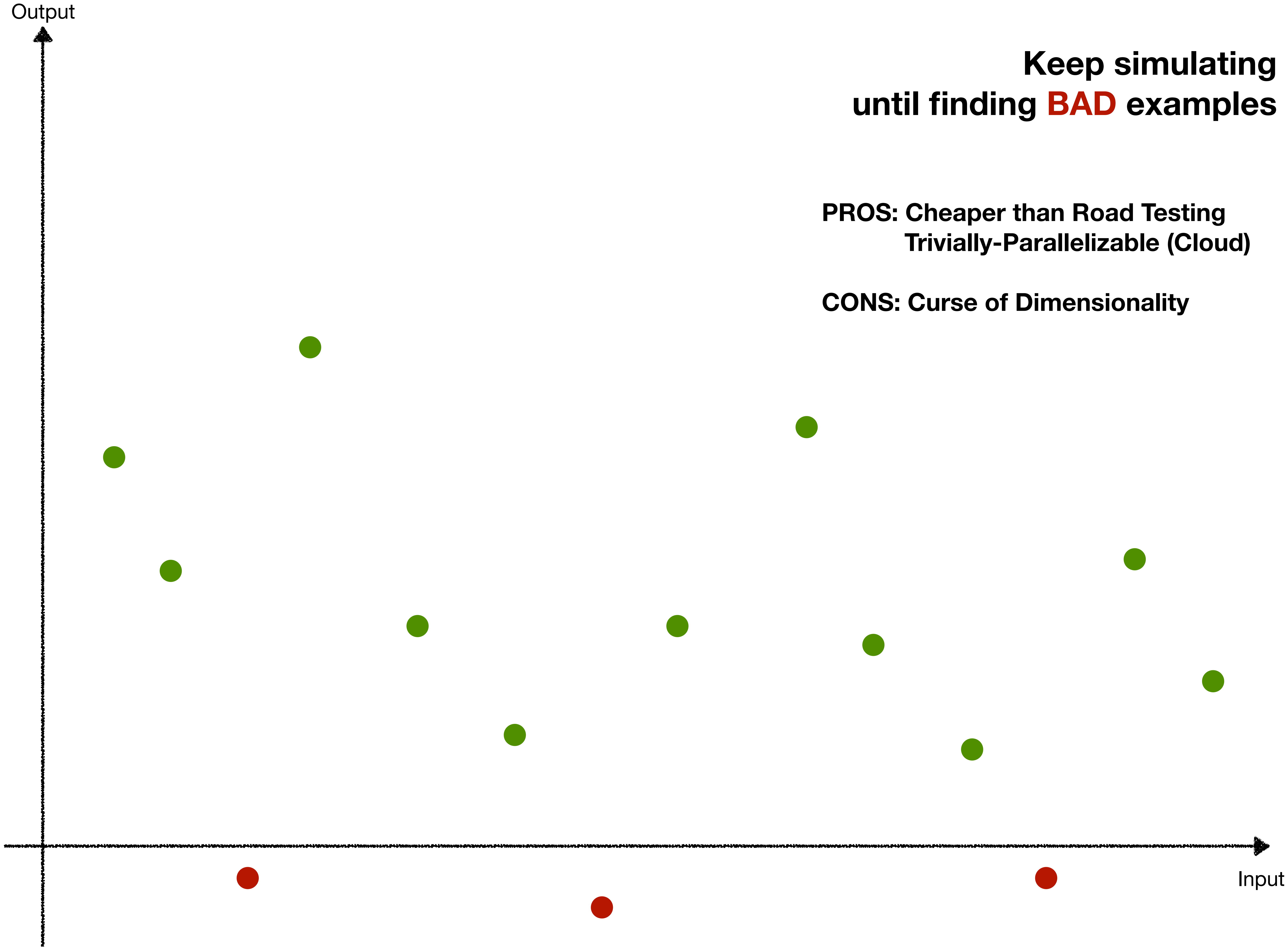
Keep simulating
until finding BAD examples

Simulation = Black-Box Testing

Keep simulating
until finding BAD examples

PROS: Cheaper than Road Testing
Trivially-Parallelizable (Cloud)

CONS: Curse of Dimensionality



Simulation = Black-Box Testing

APEX: Autonomous Vehicle Plan Verification and Execution

Matthew O'Kelly, Houssam Abbas
University of Pennsylvania

Sicun Gao
MIT

Shin'ichi Shiraishi
Toyota Info Technology Center USA

Shinpei Kato
Nagoya University

Rahul Mangharam
University of Pennsylvania

Randomized testing, where the configurations are sampled from hypercubes of parameters, is not a scalable solution: suppose we decide to sample only 10 points in the range of every state variable. For our 7D model, and with 2 cars, this yields a total of 10^{14} simulations. Say we wish to simulate 10 seconds. Even if a simulation runs in real-time, this still requires $10 * 10^{14}$ seconds = 30 million years to complete.

APEX: Autonomous Vehicle Plan Verification and Execution

Matthew O'Kelly, Houssam Abbas
University of Pennsylvania

Sicun Gao
MIT

Shin'ichi Shiraishi
Toyota Info Technology Center USA

Shinpei Kato
Nagoya University

Rahul Mangharam
University of Pennsylvania

Randomized testing, where the configurations are sampled from hypercubes of parameters, is not a scalable solution: suppose we decide to sample only 10 points in the range of every state variable. For our 7D model, and with 2 cars, this yields a total of 10^{14} simulations. Say we wish to simulate 10 seconds. Even if a simulation runs in real-time, this still requires $10 * 10^{14}$ seconds = 30 million years to complete.

Want to do: Accelerated Testing (e.g. concolic testing) / Verification / Synthesis

=> Need symbolic representation of a system

2. Modeling Language

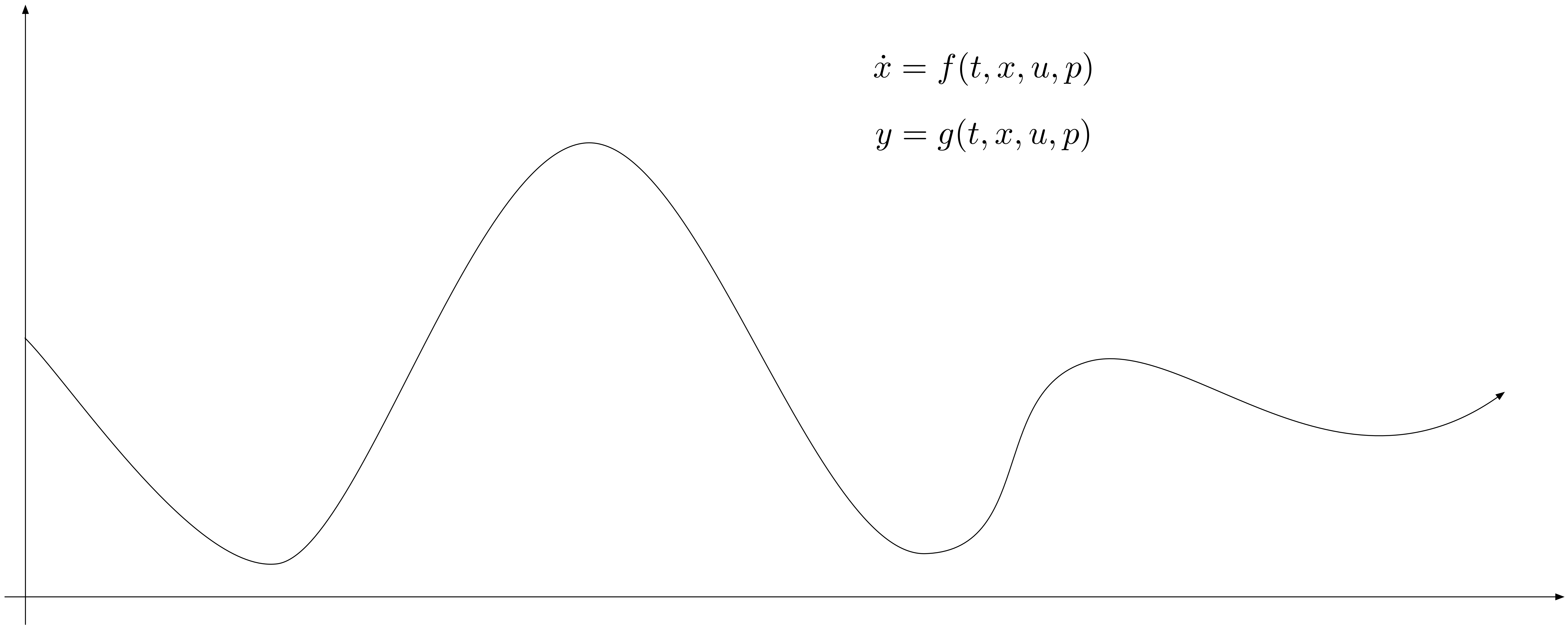
DRAKE

A PLANNING, CONTROL, AND ANALYSIS TOOLBOX FOR NONLINEAR DYNAMICAL SYSTEMS

C++ Library which you can:

- **Model** dynamical systems
- **Simulate** dynamical systems with a suite of numerical integration routines
- Construct and **solve optimization** problems
- **Analysis** dynamical systems
  (local stability/controllability/observability analysis)
- Planning, Controller design, ...

Dynamical Systems
(Continuous)



$$\dot{x} = f(t, x, u, p)$$

$$y = g(t, x, u, p)$$

Diagram

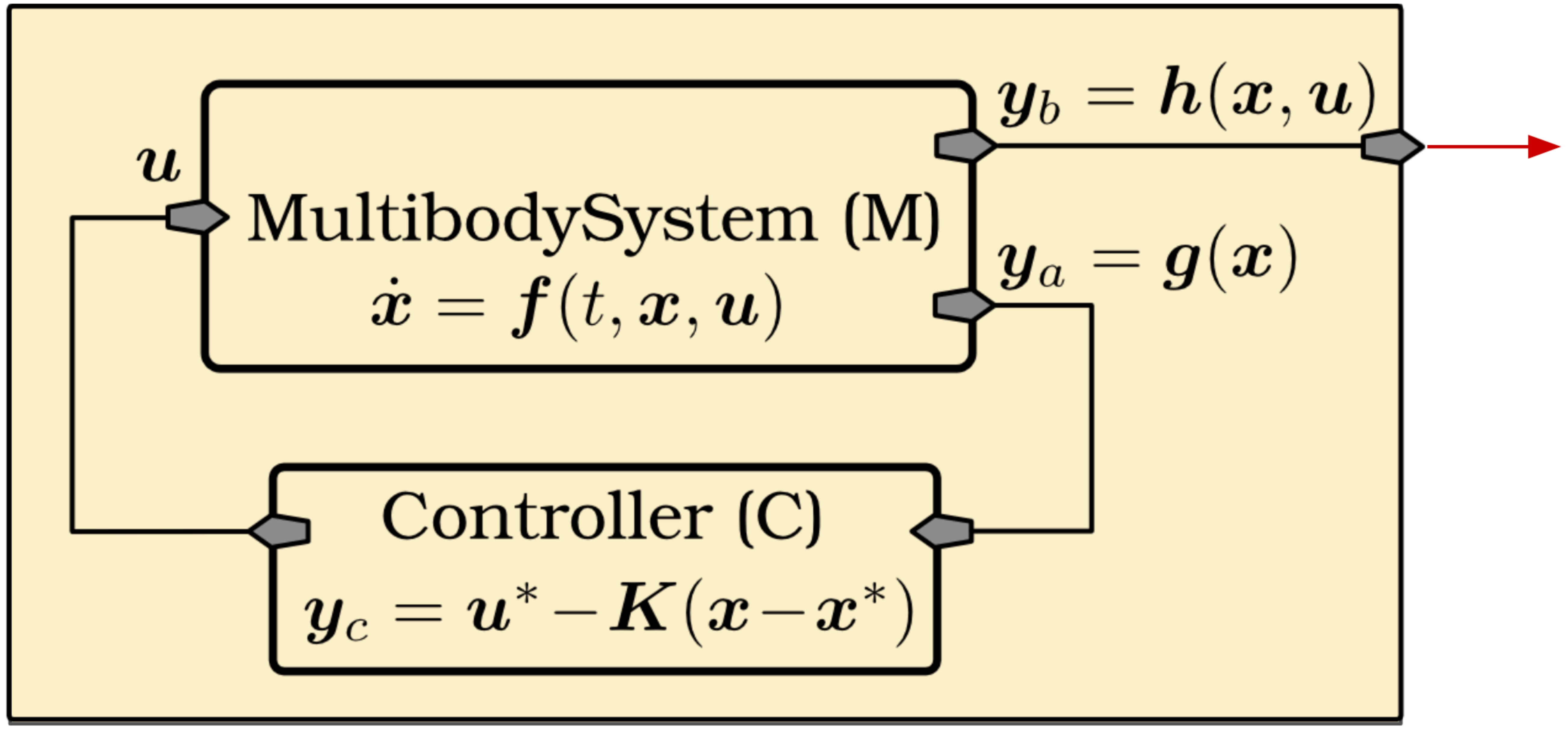


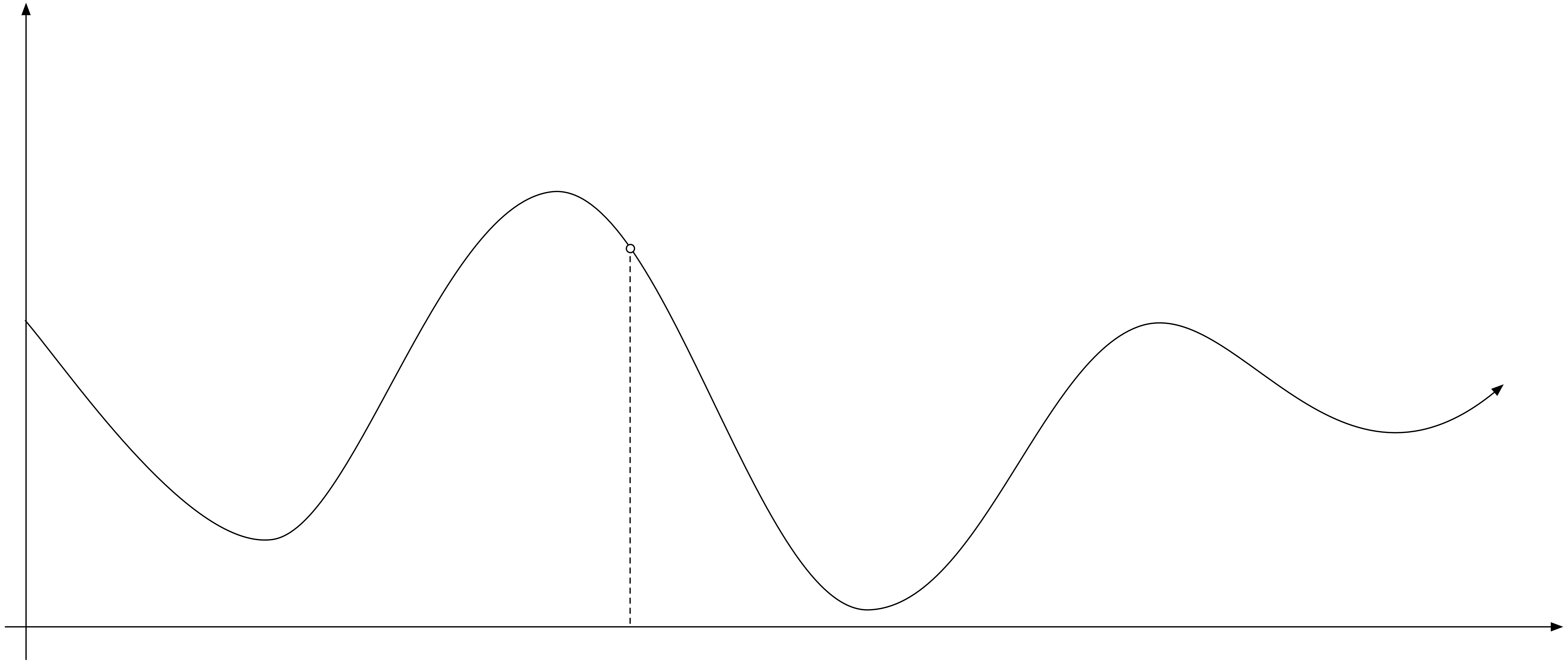
Diagram = A Graph of Systems = A System

Photo Credit: Alejandro Castro @ TRI

Templated System Framework

`System<T>` **where** `T` **can be:**

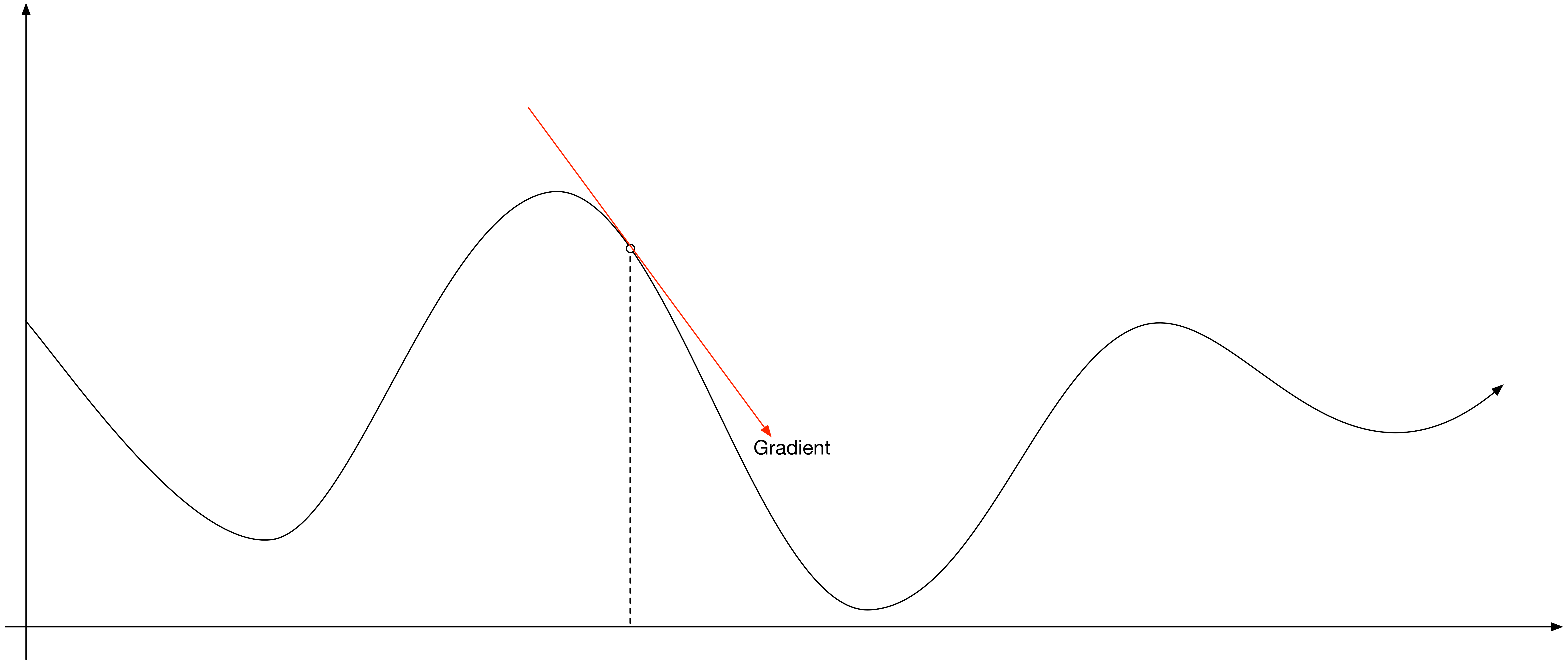
- `double` **for Simulation / Testing**



Templated System Framework

`System<T>` **where** `T` **can be:**

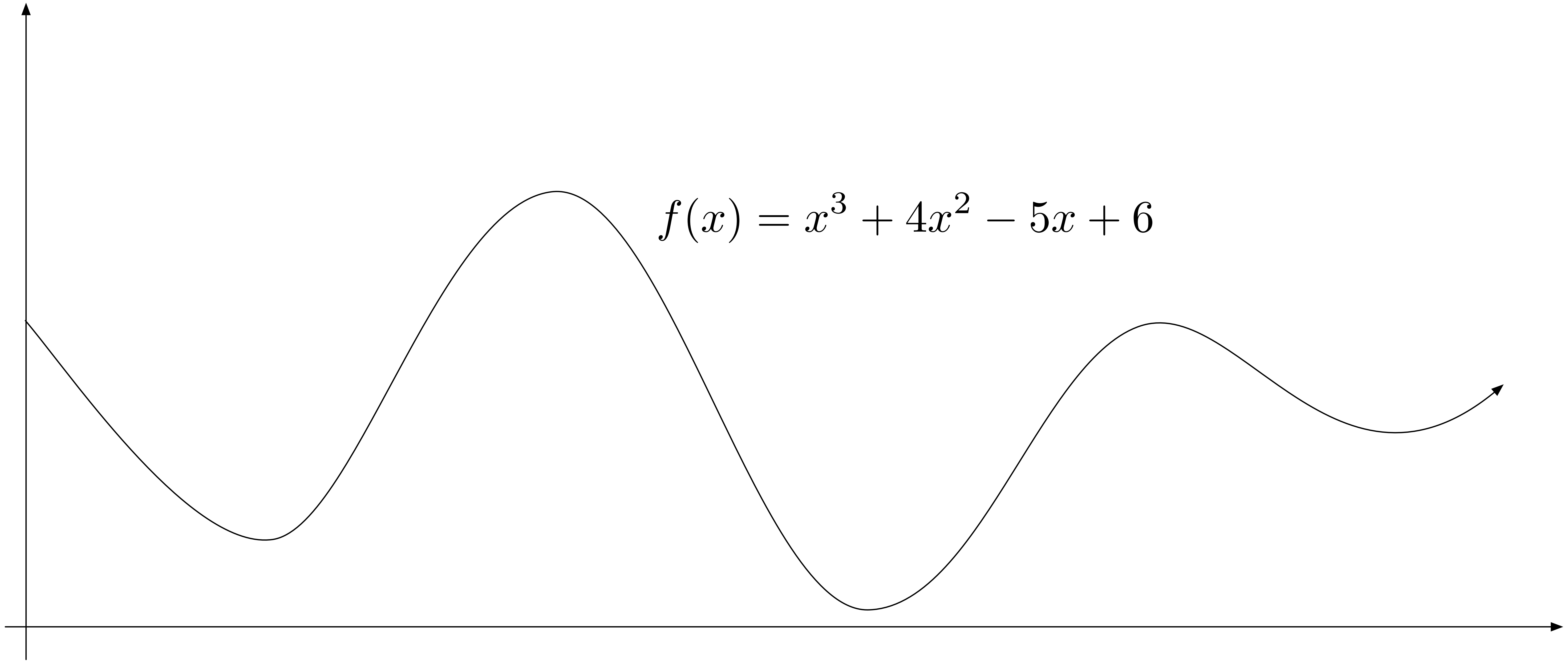
- `double` **for Simulation / Testing**
- `AutoDiff` **for Optimization-based Analysis & Design**



Templated System Framework

`System<T>` **where** `T` **can be:**

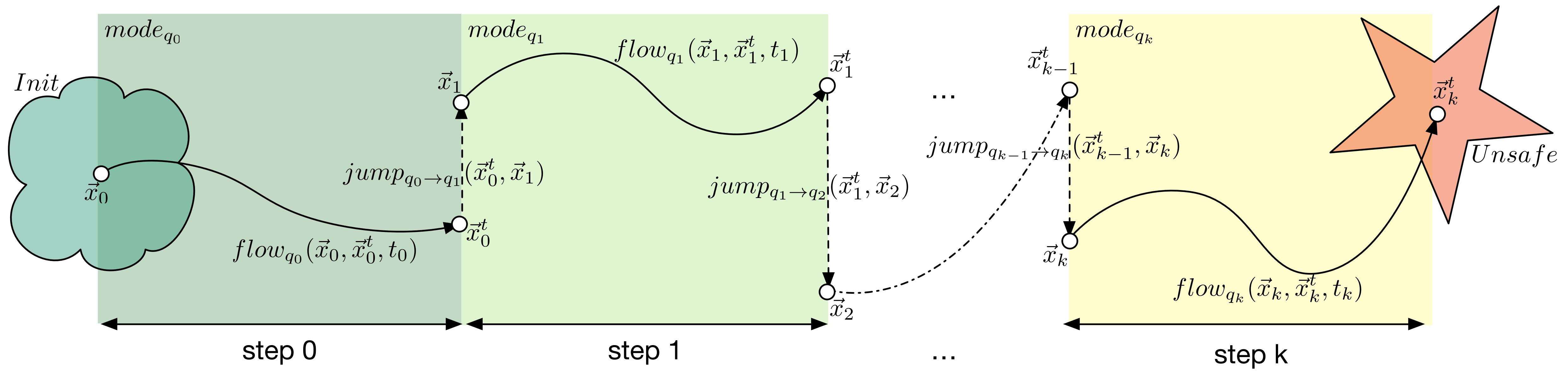
- `double` **for Simulation / Testing**
- `AutoDiff` **for Optimization-based Analysis & Design**
- `symbolic::Expression` **for Symbolic Analysis & Verification (e.g. SMT)**



3. $\mathbb{R}$eal Problems

First-order encoding of $\mathbb{R}$eal problems in control domain

- Planning / Reachability



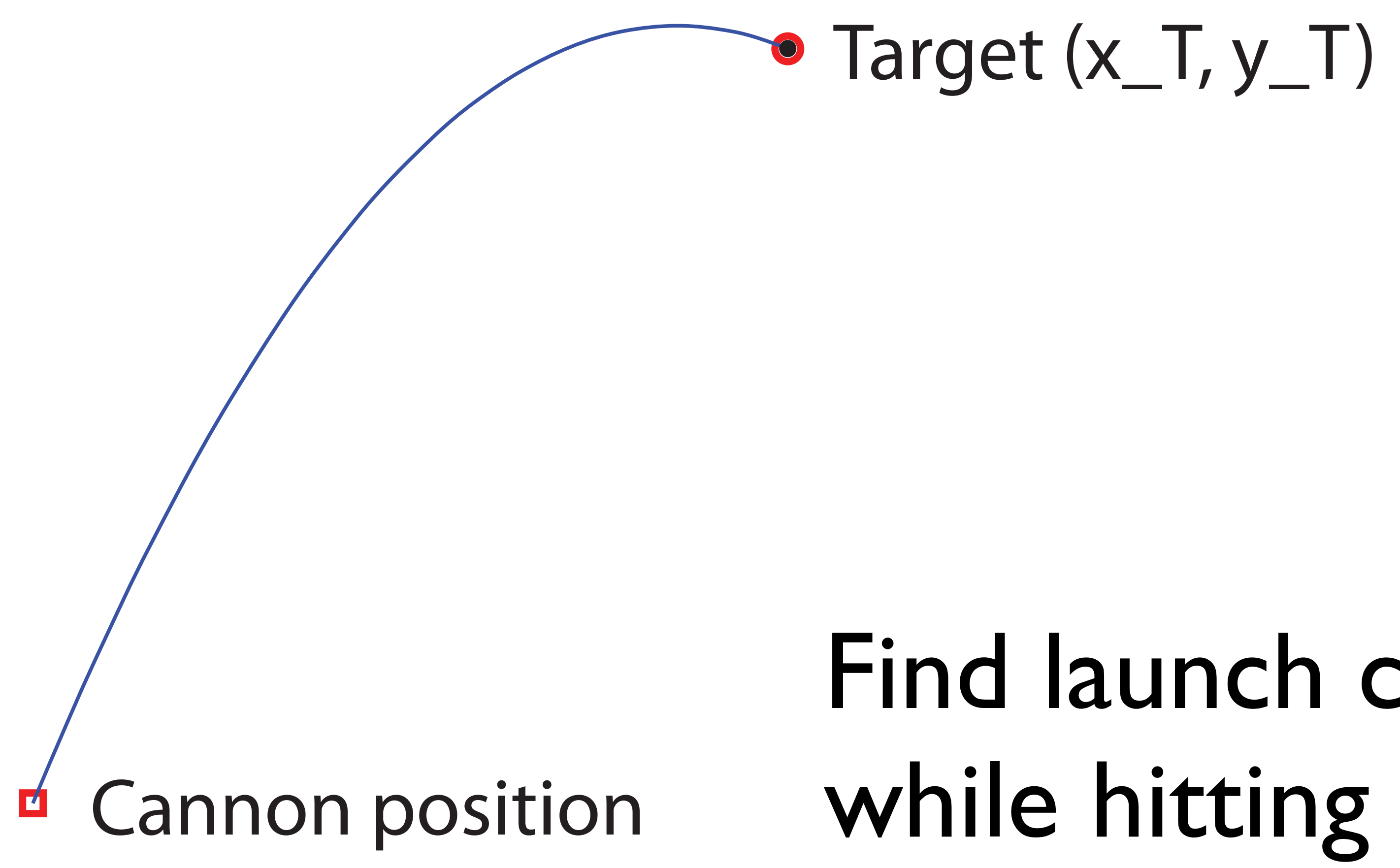
$$\exists \vec{x}_0, \vec{x}_1, \ldots, \vec{x}_k \exists \vec{x}^t_0, \vec{x}^t_1, \ldots, \vec{x}^t_k \exists t_0, t_1, \ldots, t_k$$

$$\begin{aligned}
Init(\vec{x}_0) \wedge flow_{q_0}(\vec{x}_0, \vec{x}^t_0, t_0) \wedge jump_{q_0 \to q_1}(\vec{x}^t_0, \vec{x}_1) \wedge \\
flow_{q_1}(\vec{x}_1, \vec{x}^t_1, t_1) \wedge jump_{q_1 \to q_2}(\vec{x}^t_1, \vec{x}_2) \wedge \\
\ldots \\
flow_{q_k}(\vec{x}_k, \vec{x}^t_k, t_k) \wedge Unsafe(\vec{x}^t_k)
\end{aligned}$$

First-order encoding of problems in control domain

- Planning / Reachability (∃)

- Trajectory Optimization / Optimal Control

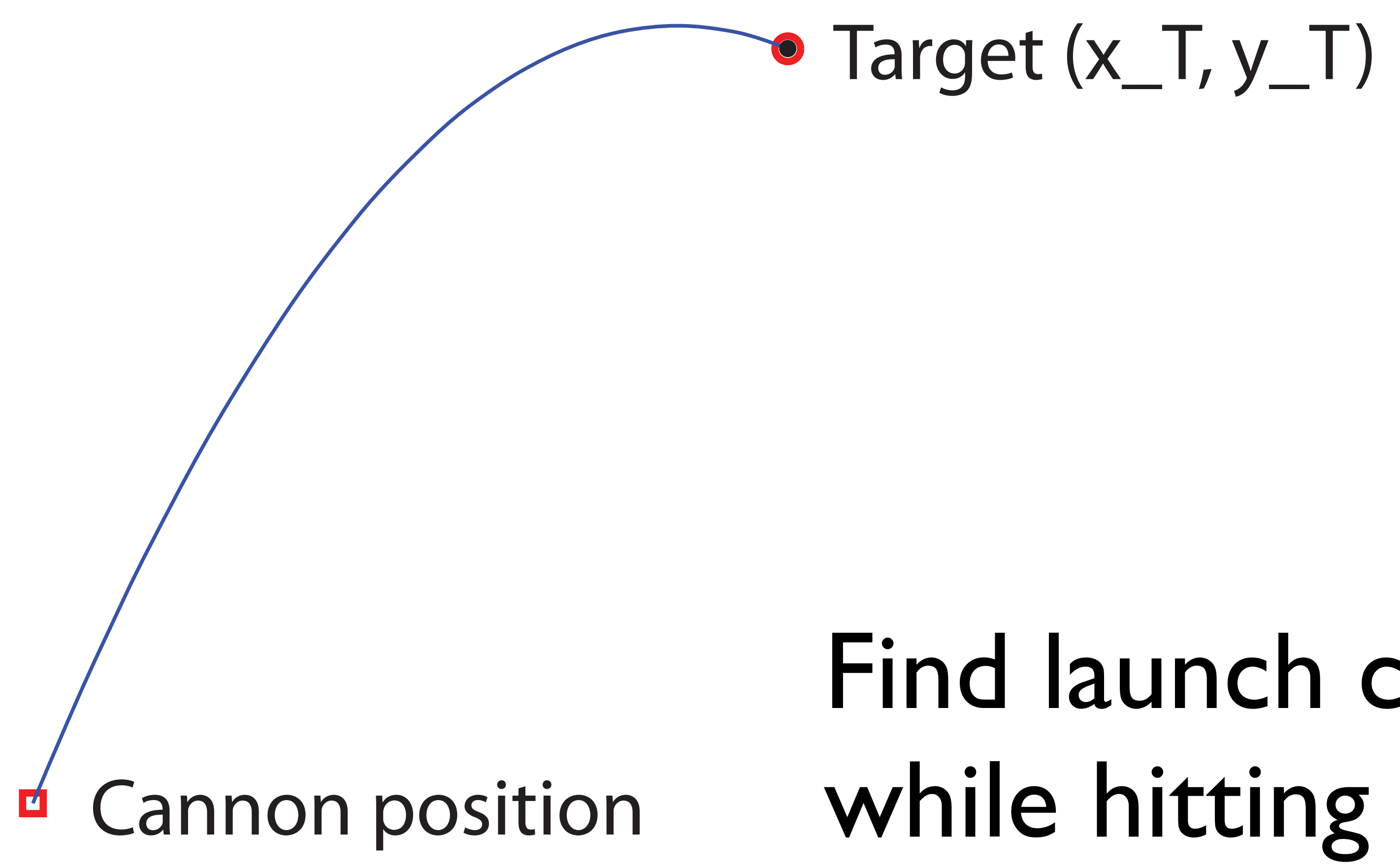


Find launch conditions that minimize launch energy while hitting the target

First-order encoding of problems in control domain

- Planning / Reachability (∃)

- Trajectory Optimization / Optimal Control



Find launch conditions that minimize launch energy while hitting the target

=> Constrained Optimization Problem

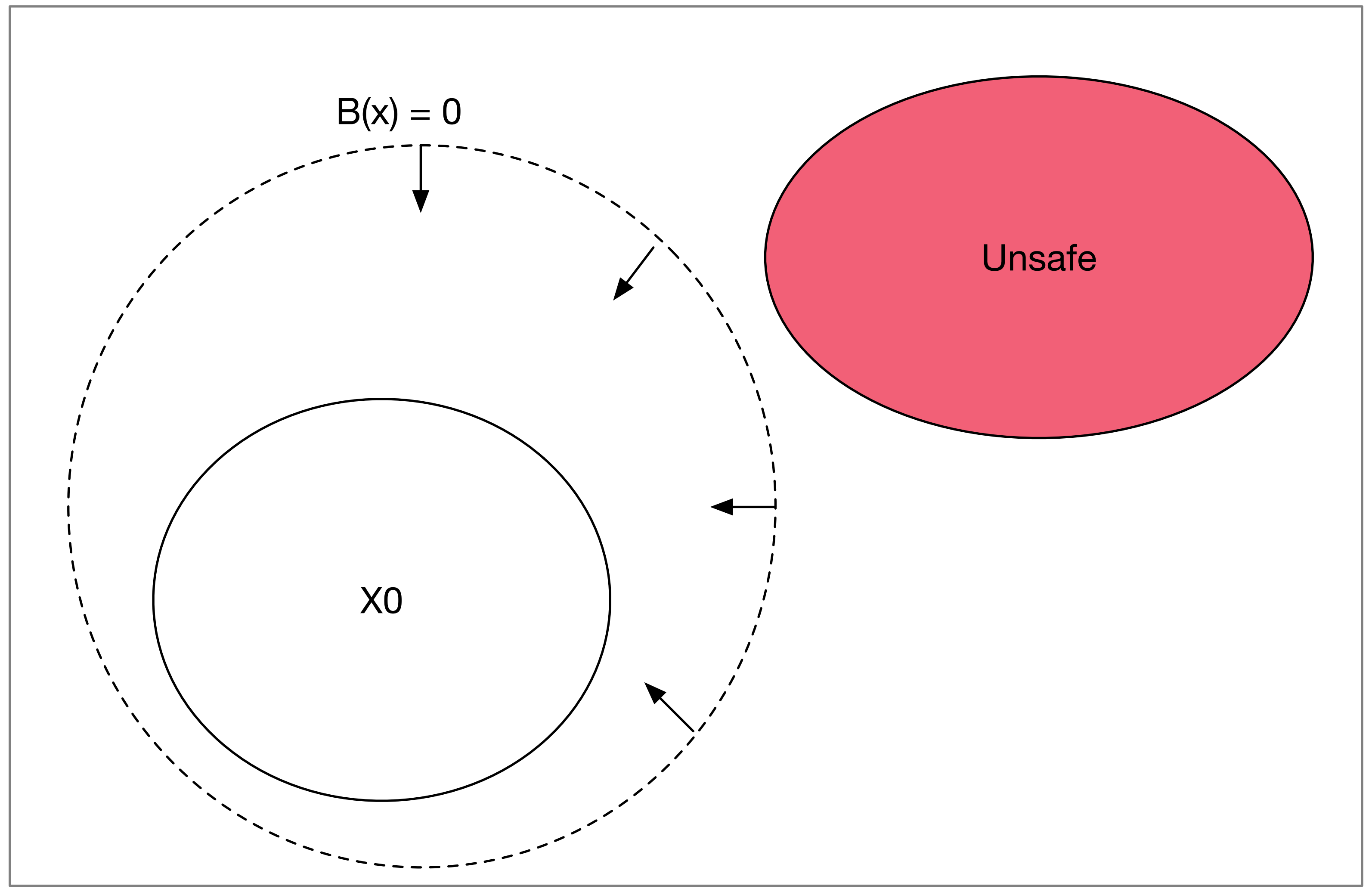
$$\min f(\mathbf{x}) \text{ s.t. } \phi(\mathbf{x})$$

$\Updownarrow$ is logically

$$\exists \mathbf{x}. \forall \mathbf{y}.\ \phi(\mathbf{x}) \wedge \phi(\mathbf{y}) \rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$$

First-order encoding of problems in control domain

- Planning / Reachability (∃)
- Trajectory Optimization / Optimal Control (∃∀)
- Lyapunov-function (stability) / Barrier certificate (safety)



Find a function, **B** : X → R, which satisfies the following conditions to show that it is **not** possible to have a trajectory starting in **X0** and reaching **Unsafe:**

$$\forall x \in X_0 \implies B(x) \leq 0$$

$$\forall x \in U \implies B(x) > 0$$

$$\forall x \,.\, B(x) = 0 \implies \frac{\partial B}{\partial x} f(x) \leq 0$$

First-order encoding of problems in control domain

- Planning / Reachability (∃)
- Trajectory Optimization / Optimal Control (∃∀)
- Lyapunov-function / Barrier certificate (∃∀)

First-order encoding of problems in control domain

- Planning / Reachability ($\exists$)
- Trajectory Optimization / Optimal Control ($\exists\forall$)
- Lyapunov-function / Barrier certificate ($\exists\forall$)
- Robust Optimal Control / Robust Optimization ($\exists\forall\exists$)

Minimize $c(x)$
s.t. $g(x, u)$ for all $u \in U$.

Encoding:

$\exists x. (\forall u \in U.\ g(x, u)) \wedge (\forall y. (\forall u \in U.\ g(y, u)) \Rightarrow c(x) \leq c(y))$

=> $\exists x. (\forall u \in U.\ g(x, u)) \wedge (\forall y. (\exists u \in U.\ \neg g(y, u)) \vee (c(x) \leq c(y)))$

4. How to Solve it?

Decision Problems over the Reals

Given an arbitrary first-order sentence over $\langle \mathbb{R}, \geq, \mathcal{F} \rangle$, such as

$$\varphi = Q_1^{[l_1,u_1]} x_1 \ldots Q_n^{[l_n,u_n]} x_n . \bigwedge_i \left(\bigvee_j f_{i,j}(\vec{x}) > 0 \vee \bigvee_k f_{i,k}(\vec{x}) \geq 0 \right)$$

where $f \in \mathcal{F}$, can we compute whether φ is true or false?

- Complexity results of **non-linear** arithmetic over the **Reals**
  - **Decidable** if φ only contains polynomials [Tarski51]
  - **Undecidable** if φ includes trigonometric functions (i.e. sin)
- **Real-world problems** contain **complex nonlinear functions**
  (trigonometric functions, log, exp, ODEs)

Delta-decision Problem

- Given a first-order formula over the Real, φ , and a positive rational number δ, **delta-decision problem** asks for one of the following answers:

    - **UNSAT**: φ is unsatisfiable
    - **δ-SAT** : $\varphi^{-\delta}$ is satisfiable.

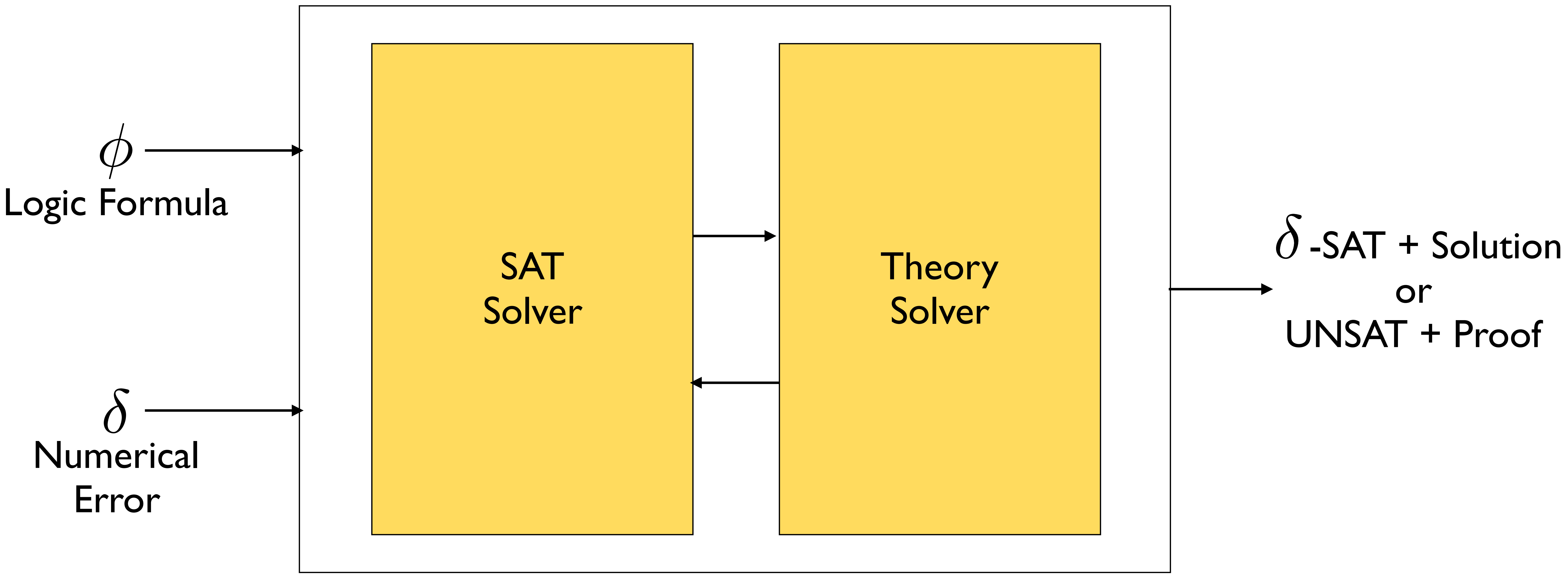
  where $\varphi^{-\delta}$ is called the **δ-weakening** of φ which is formally defined as follows:

$$\varphi^{-\delta} = Q_1^{[l_1,u_1]}x_1 ... Q_n^{[l_n,u_n]}x_n. \bigwedge_i \left(\bigvee_j f_{i,j}(\vec{x}) > -\delta \vee \bigvee_j f_{i,k}(\vec{x}) \geq -\delta \right)$$

- It is shown that this problem is **decidable** for signatures with computable functions [LICS12]

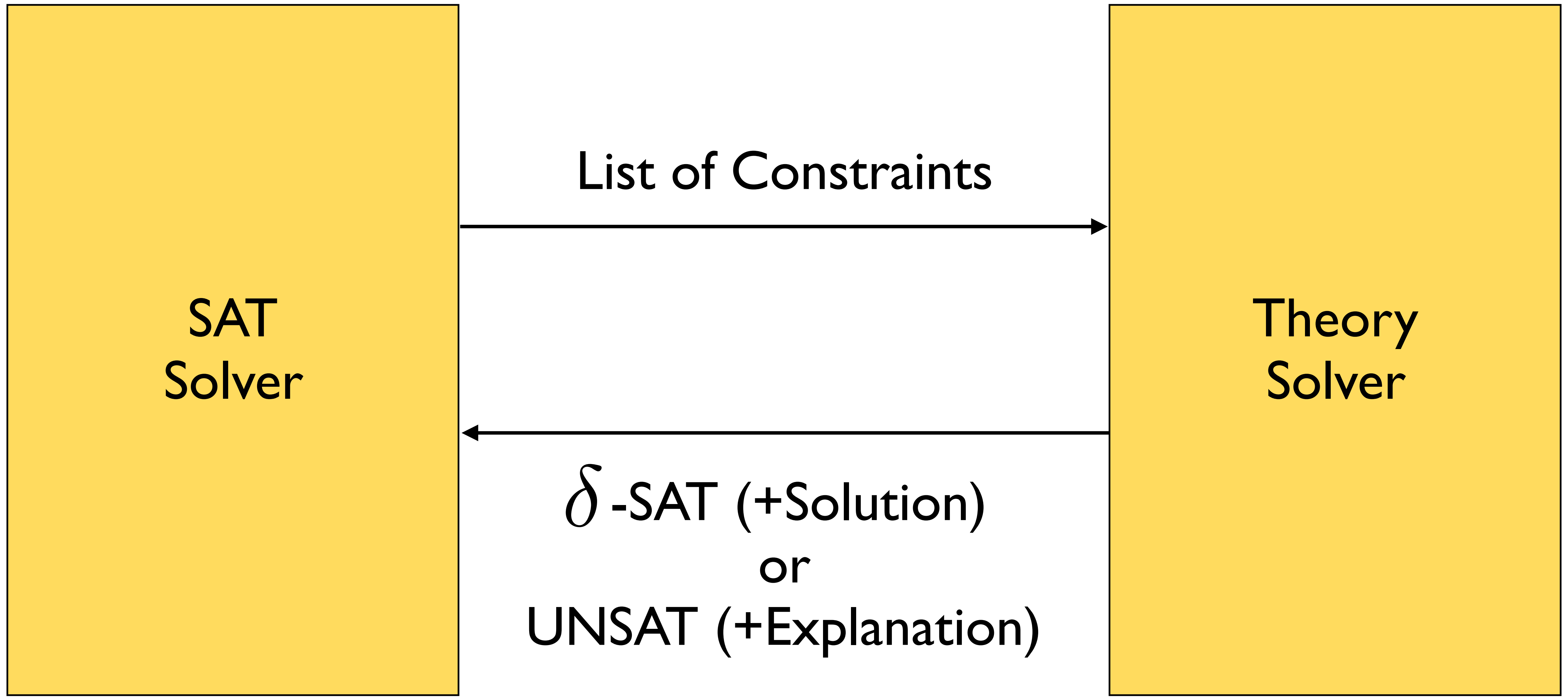
- The **complexity** for existential problems is **NP** (with P-time computable functions) or **PSPACE** (with Lipschitz ODEs) [LICS12]

Design of Solver: Big Picture



- **SAT solver** finds a satisfying **Boolean** assignment
- **Theory solver** checks whether the assignment is feasible under the first order theory of **Real**

Design of Solver: Big Picture

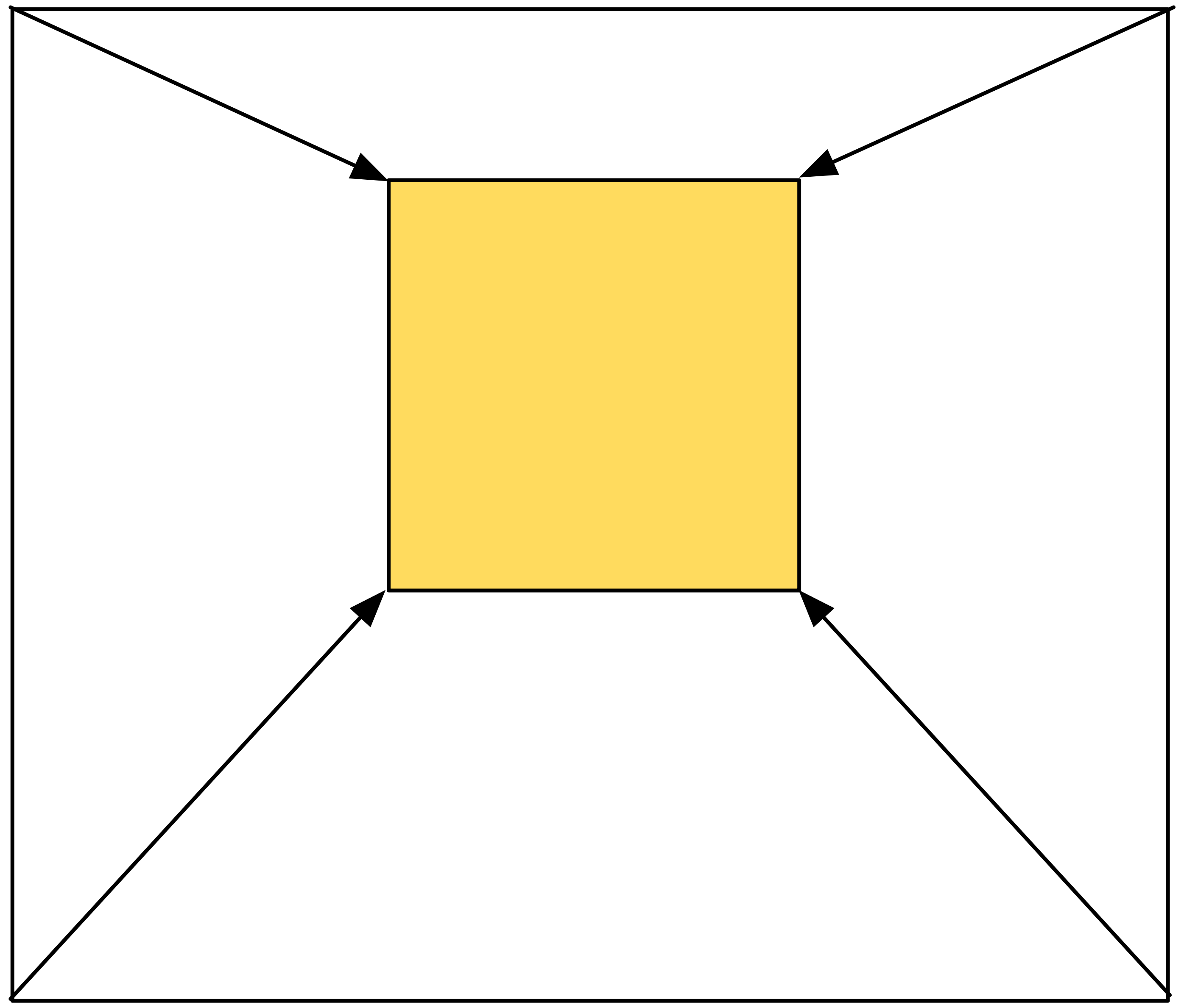


Boolean Search
Non-chronological Backtracking
Learning
...

(Discrete Domain)

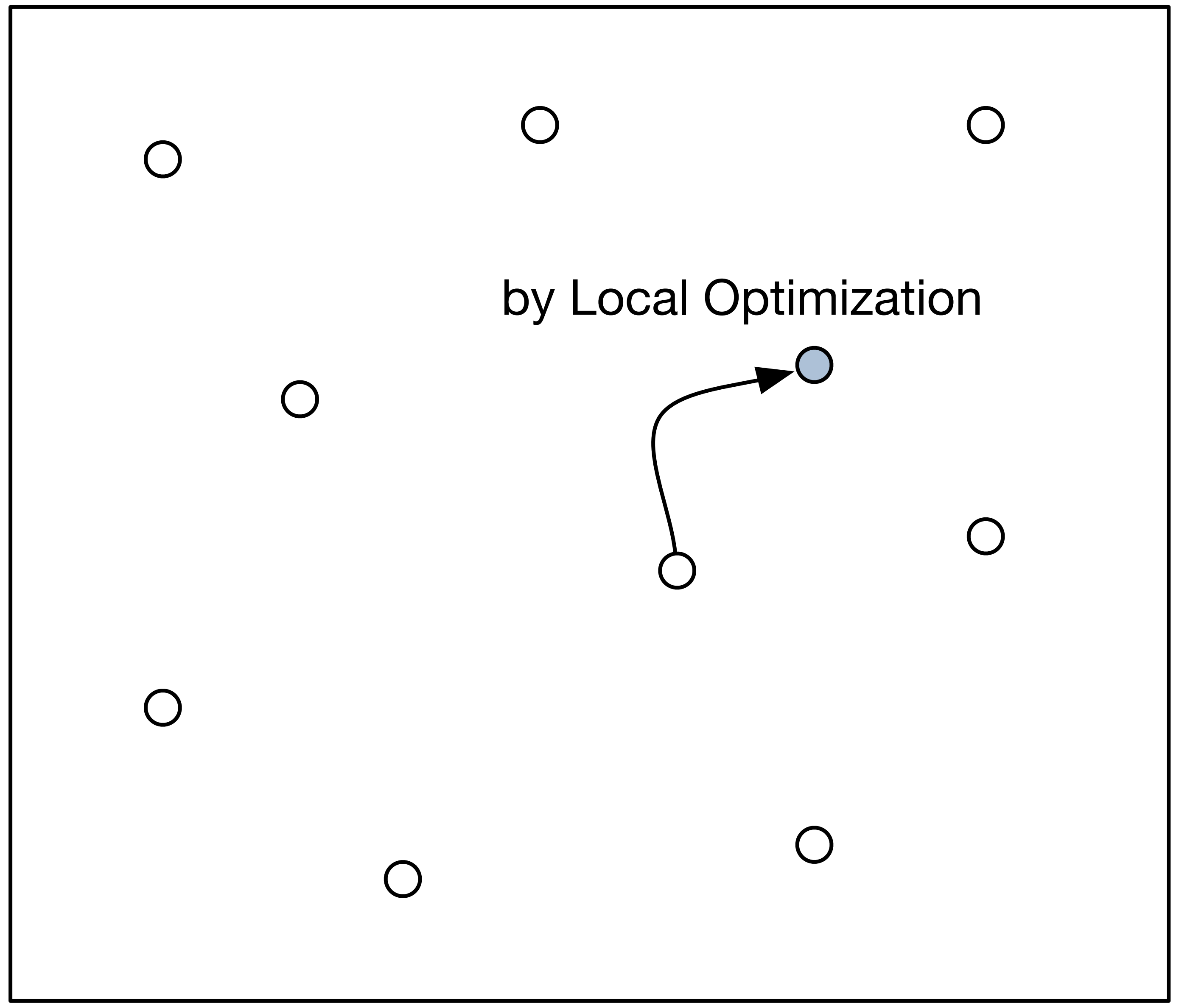
Constraints Solving
Validated Numerics
Optimization
Simulation/Sampling
...
(Continuous Domain)

Top-down/Bottom Approaches in Theory Solver



Top-Down Approach

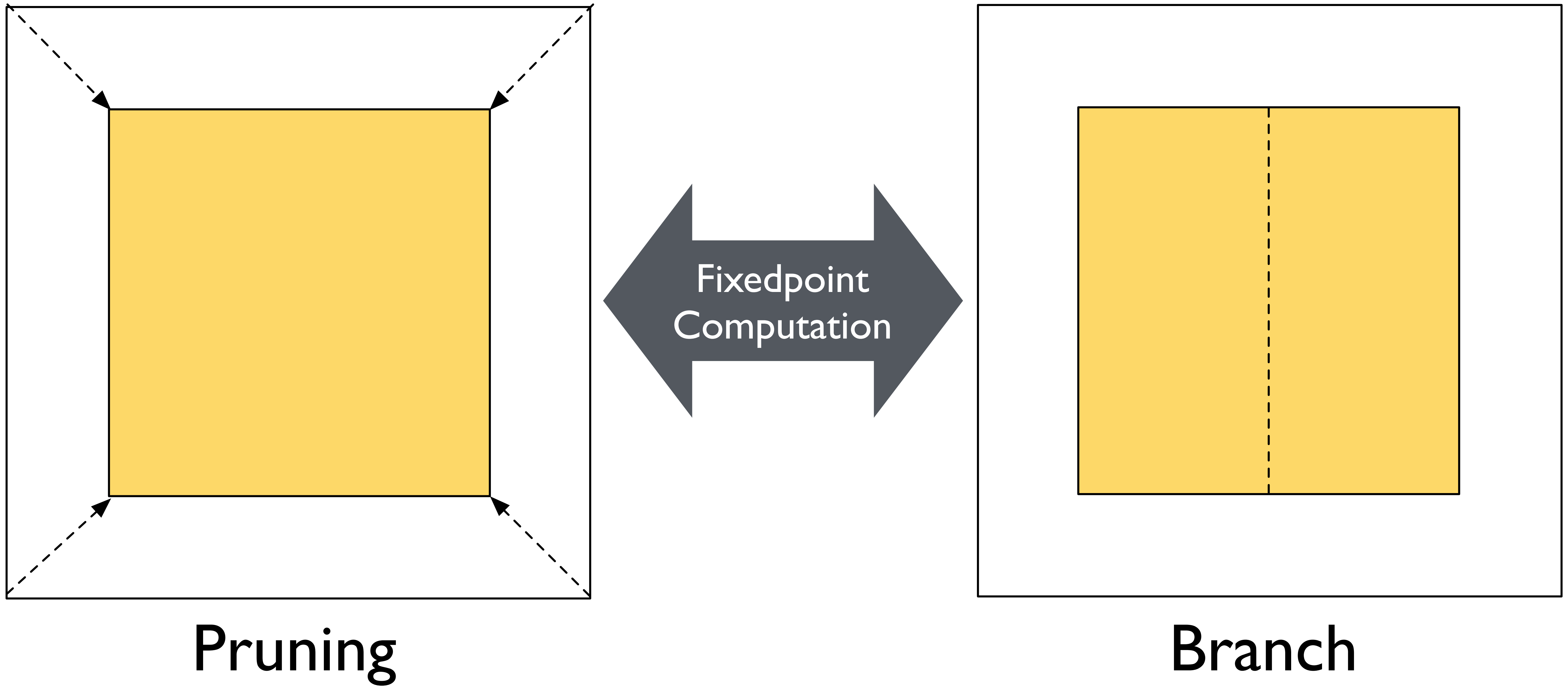
Maintain a set of possible solutions
Useful to show **UNSAT**
Validated Numerics
(i.e. Interval-based methods)



Bottom-Up Approach

Sample points and test them
Useful to show **SAT**
Use local-optimization to improve

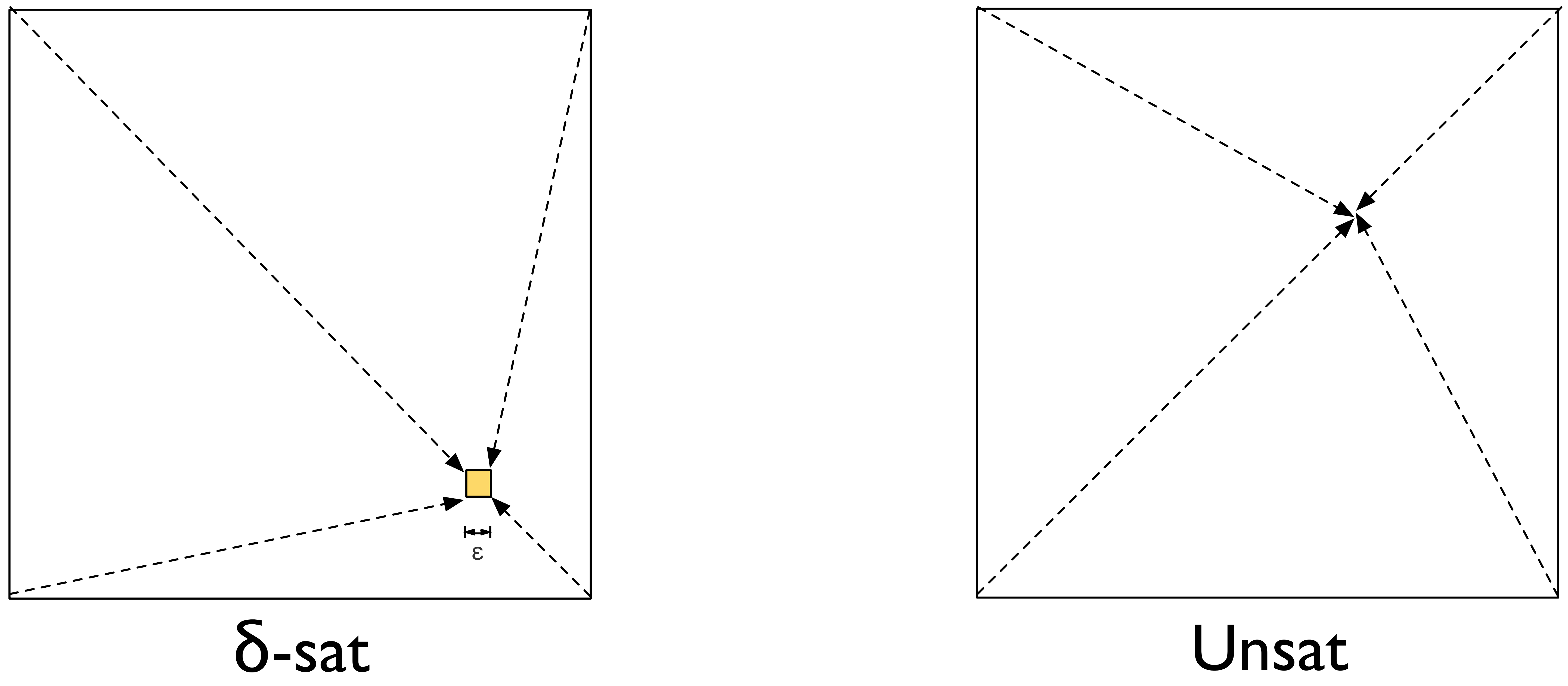
An Algorithm in Theory Solver: ICP(Interval Constraint Propagation)



Safely **reduce** a search space without removing solutions

Partition a search space into two sub-spaces

Two Termination Conditions of ICP



ICP Algorithm

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;

Pruning

Branching

5. How to Solve $\exists\forall$

CAV'18

Delta-Decision Procedures for Exists-Forall Problems over the Reals

Soonho Kong[1], Armando Solar-Lezama[2], and Sicun Gao[3]

[1] Toyota Research Institute
soonho.kong@tri.global
[2] Massachusetts Institute of Technology, USA
asolar@csail.mit.edu
[3] University of California, San Diego, USA
sicung@ucsd.edu

Abstract. We propose δ-complete decision procedures for solving satisfiability of nonlinear SMT problems over real numbers that contain universal quantification and a wide range of nonlinear functions. The methods combine interval constraint propagation, counterexample-guided synthesis, and numerical optimization. In particular, we show how to handle the interleaving of numerical and symbolic computation to ensure delta-completeness in quantified reasoning. We demonstrate that the proposed algorithms can handle various challenging global optimization and control synthesis problems that are beyond the reach of existing solvers.

Problem

$$\exists x \,.\, \forall y \,.\, \varphi(x, y)$$

where ϕ can include an arbitrary **Boolean** combination of **numerically computable functions** (e.g. sin, cos, exp)

Problem

$$\exists x \,.\, \forall y \,.\, \varphi(x, y)$$

We can encode the following problems:

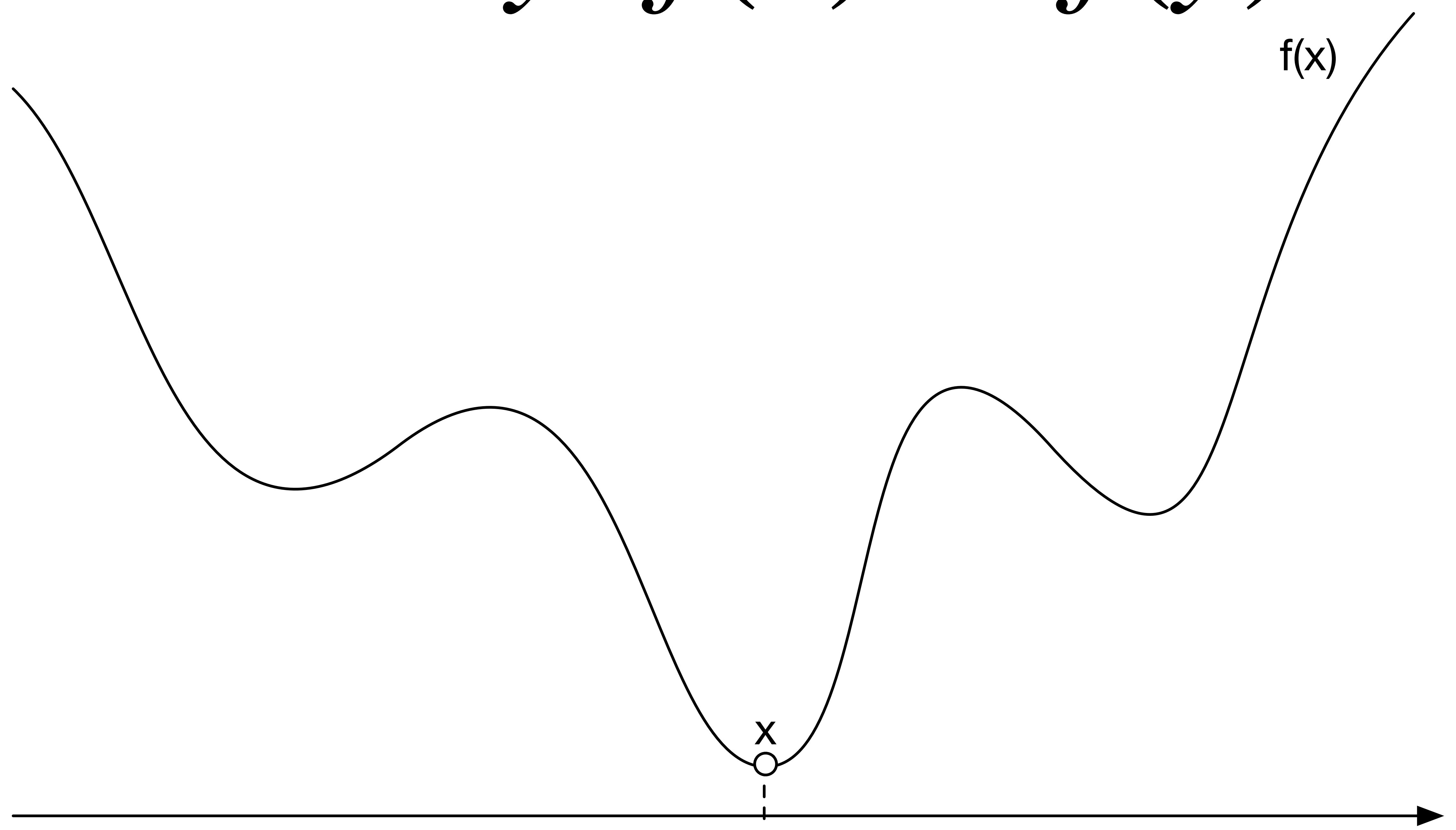
- **Optimization**
    - Non-linear / Non-convex
    - Global
    - Constrained
- **Synthesis**
    - Program
    - Controller (e.g. Lyapunov function, Barrier function)

Simple Case:
Unconstrained Global Optimization

$$\min f(x)$$

$$\Downarrow$$

$$\exists x \,.\, \forall y \,.\, f(x) \leq f(y)$$

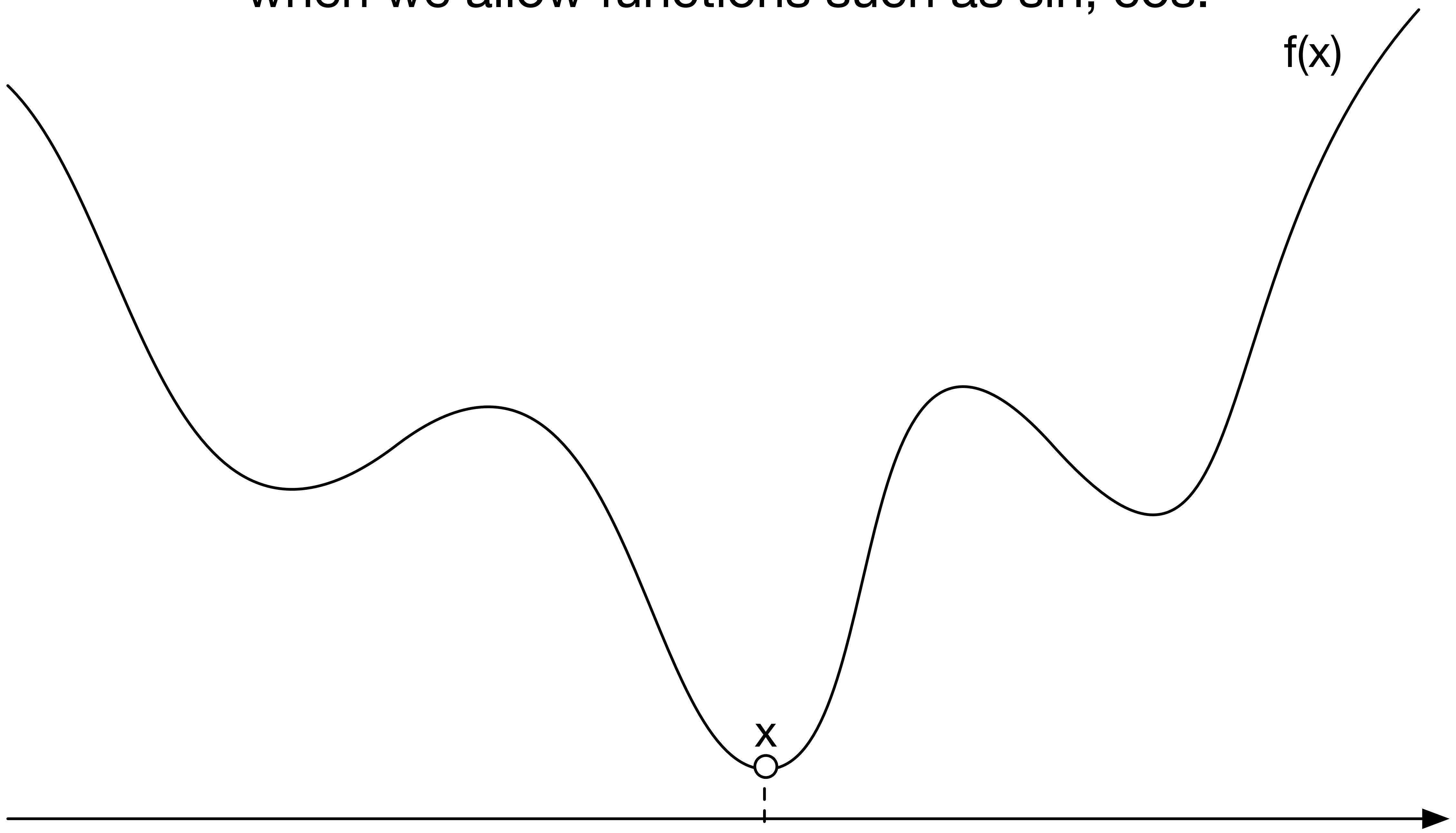


Simple Case:
Unconstrained Global Optimization

$$\exists x \,.\, \forall y \,.\, f(x) \leq f(y)$$

Remark:

Finding the **exact** global optimum is **undecidable**
when we allow functions such as sin, cos.

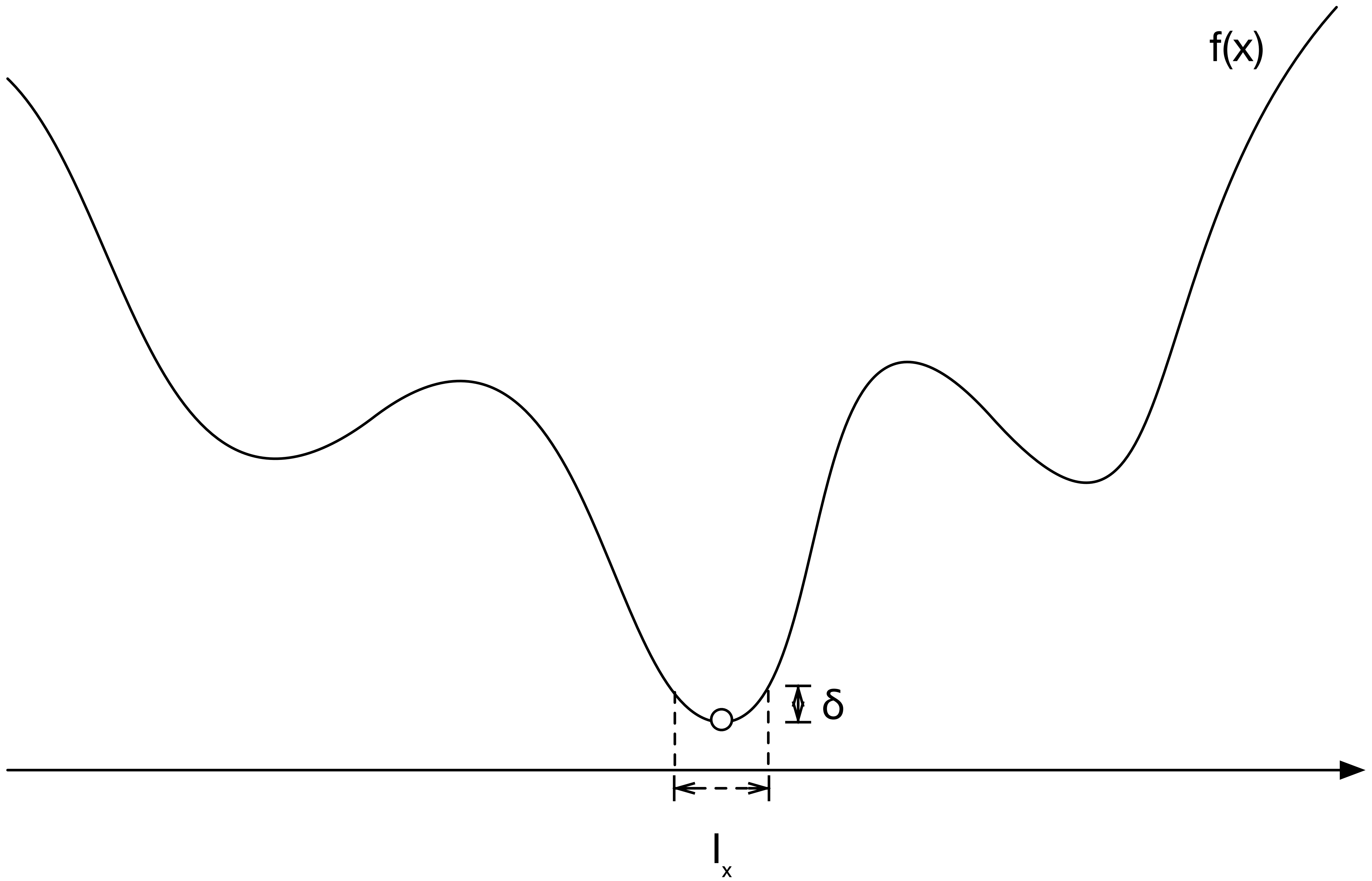


Simple Case:
Unconstrained Global Optimization

Instead, we want to find a small interval I_x such that forall x in I_x :

$$\forall y . f(x) \leq f(y) + \delta$$

Note that this problem is **decidable** (Σ_1)

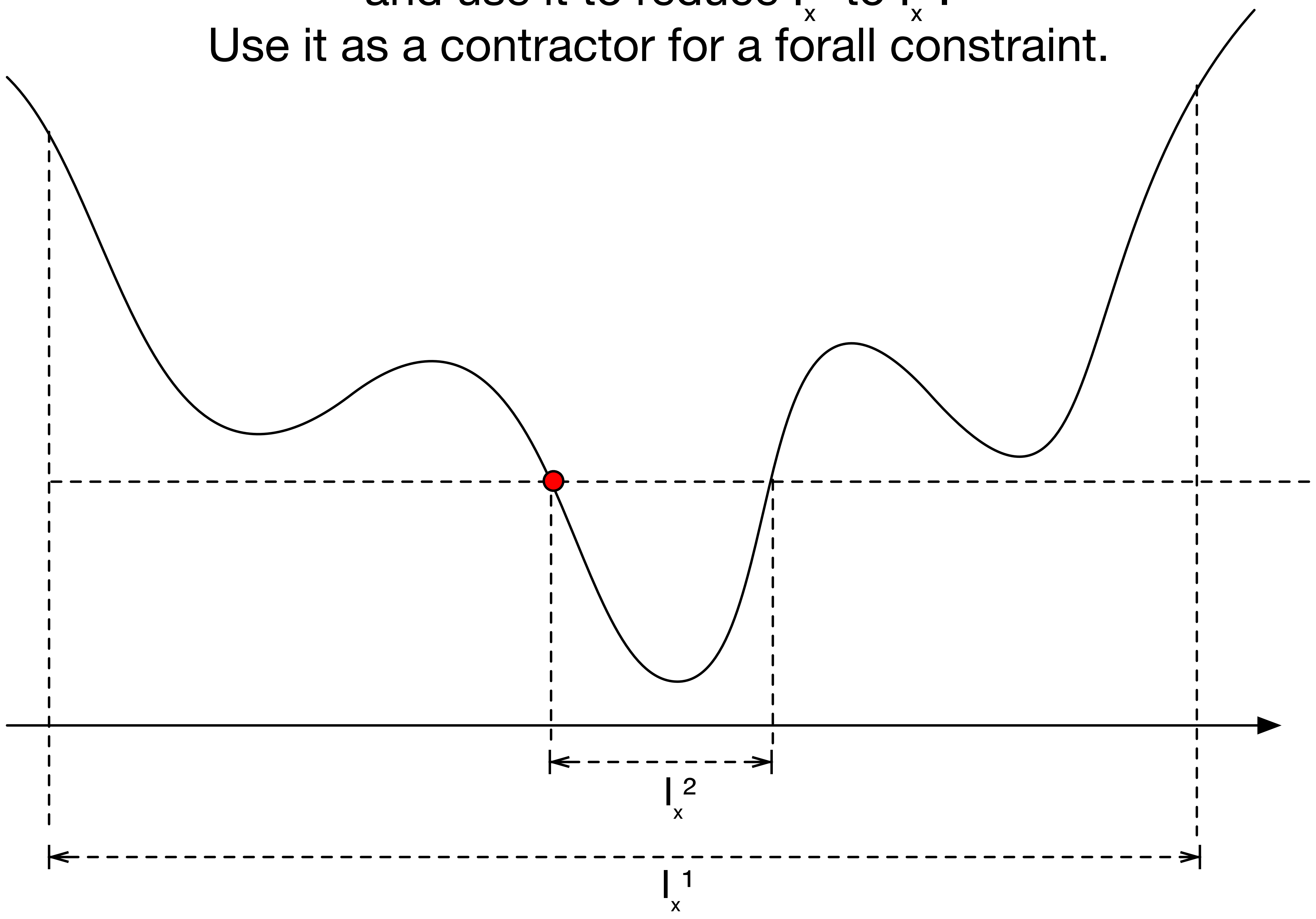


Idea 1: Counterexample Refinement

Find a **counterexample** y such that for an x in I_x^1

$$f(x) > f(y)$$

and use it to reduce I_x^1 to I_x^2.
Use it as a contractor for a forall constraint.



Finding a Counterexample

$$f(x) > f(y)$$

How do we find such y?
Note that the problem is in general **undecidable** again.

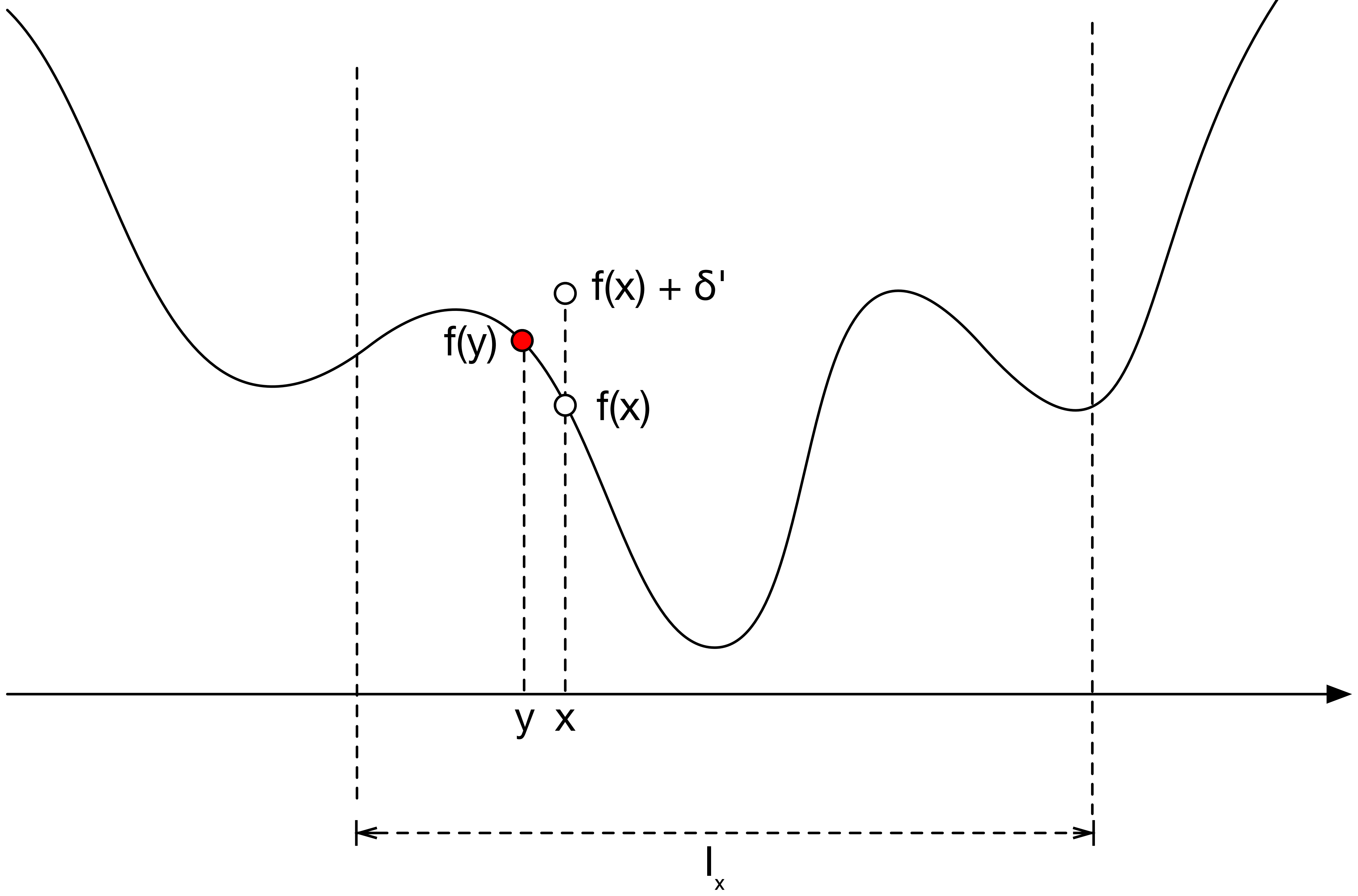
We use **delta-decision solver** to find a counterexample:

$$\text{Solve}(f(x) > f(y), \delta')$$

Problem of spurious counterexamples

Consider a δ-SAT case where Solve(f(x) > f(y), δ') finds (x, y) such that:

$$f(x) + \delta' > f(y) \quad ✓$$
$$f(x) > f(y) \quad ✗$$



Spurious counterexamples give **NO** pruning power.

Idea 2: Double-sided Error Control

Instead of solving the following to counterexample:

$$\mathrm{Solve}(f(x) > f(y), \delta')$$

Solve the following which strengthened the CE query by ε:

$$\mathrm{Solve}(f(x) > f(y) + \epsilon, \delta')$$

Q: How to pick δ' and ε given δ?

Idea 2: Double-sided Error Control

$$\text{Solve}(f(x) > f(y) + \epsilon, \delta')$$

UNSAT CASE:

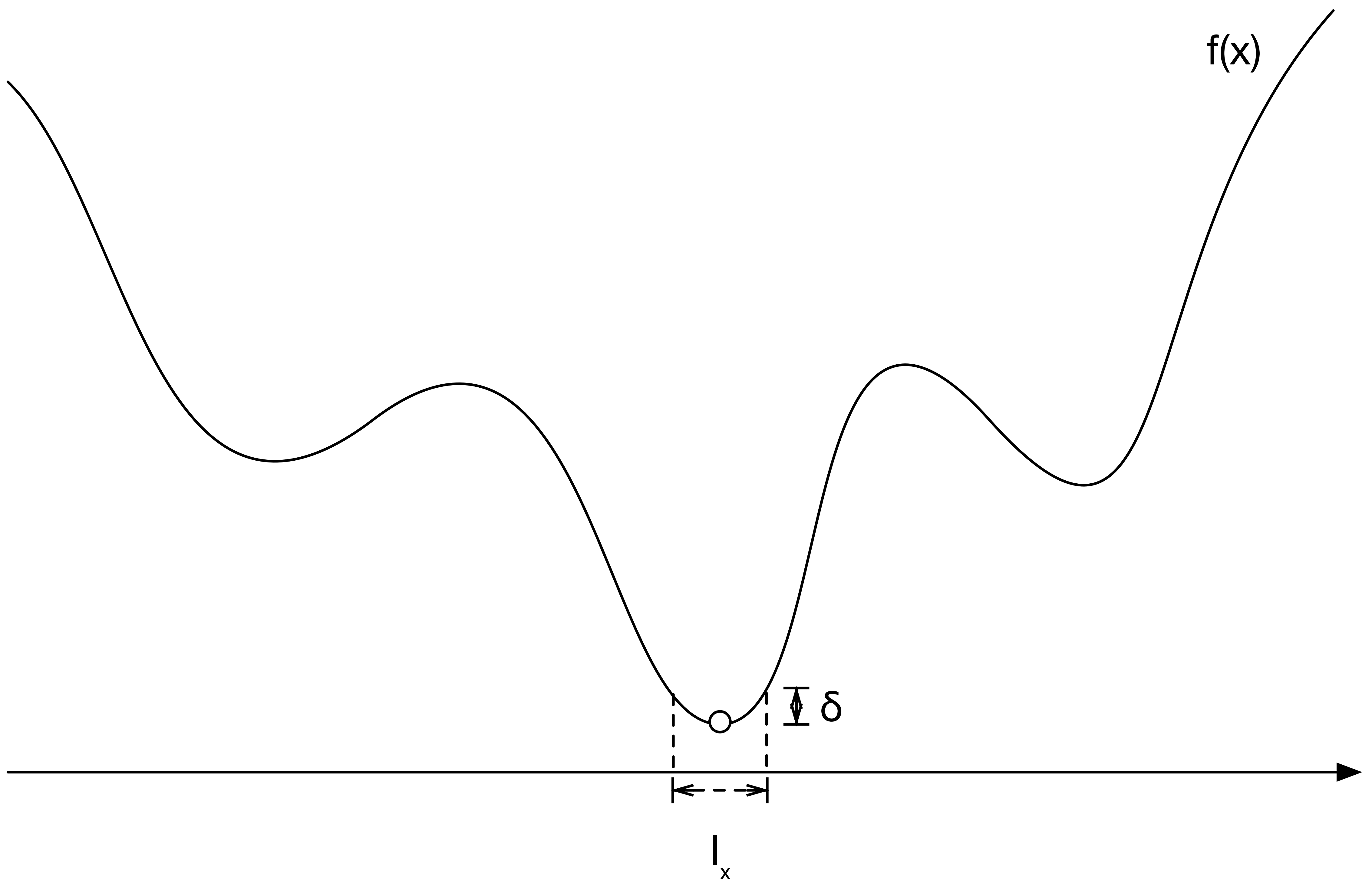
$$\forall y\,.\,f(x) \leq f(y) + \epsilon$$

Note that we wanted to satisfy:

$$\forall y\,.\,f(x) \leq f(y) + \delta$$

So we have:

$$\epsilon < \delta$$



Idea 2: Double-sided Error Control

$$\text{Solve}(f(x) > f(y) + \epsilon, \delta')$$

δ-SAT CASE: We have (x, y) such that:

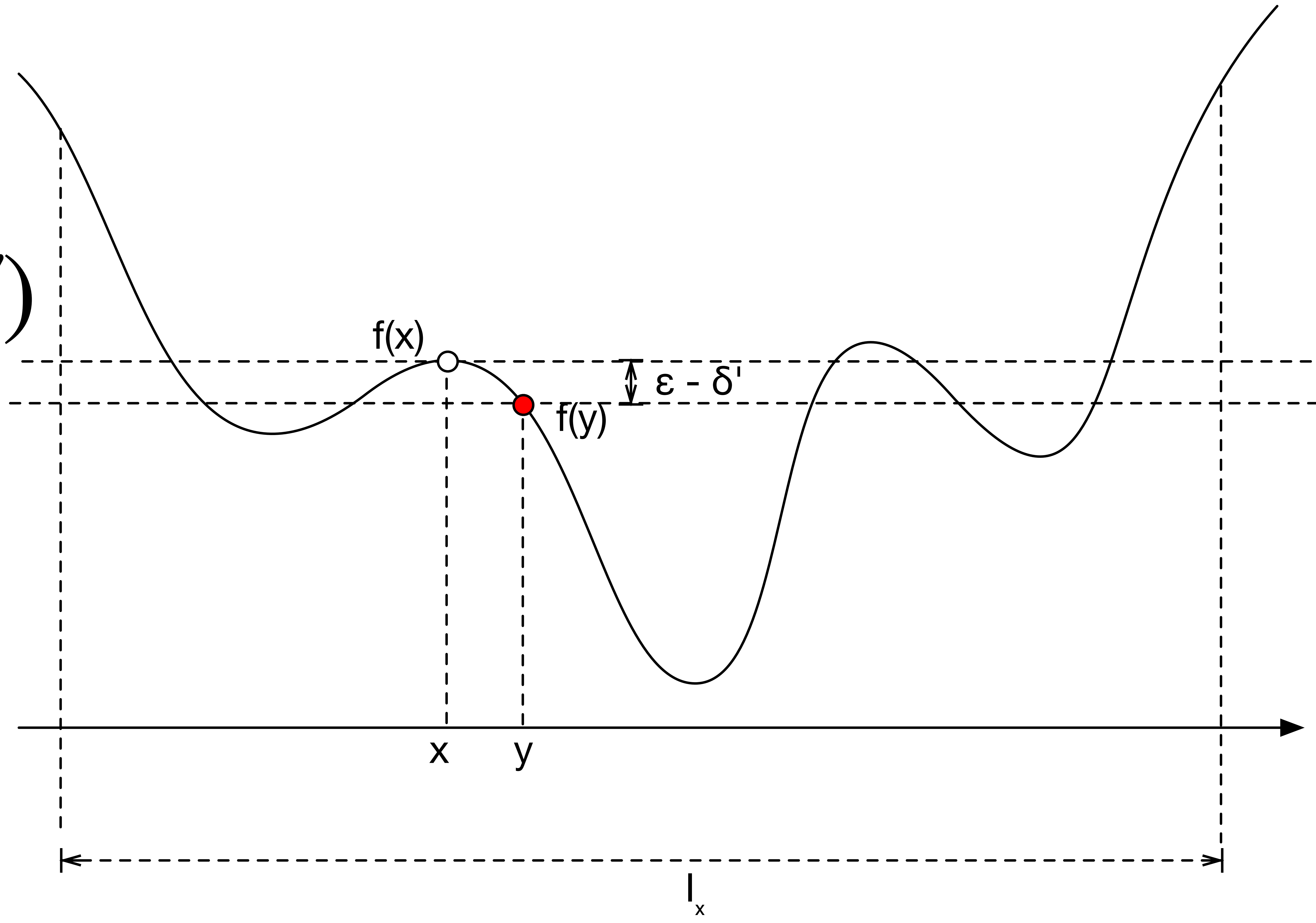
$$f(x) + \delta' > f(y) + \epsilon$$
$$f(x) > f(y) + (\epsilon - \delta')$$

We want this to be a true counterexample:

$$\epsilon - \delta' > 0$$

That is,

$$\delta' < \epsilon$$



Idea 2: Double-sided Error Control

Instead of solving the following to counterexample:

$$\mathrm{Solve}(f(x) > f(y), \delta')$$

Solve the following which strengthened the CE query by ε:

$$\mathrm{Solve}(f(x) > f(y) + \epsilon, \delta')$$

Q: How to pick δ' and ε given δ?

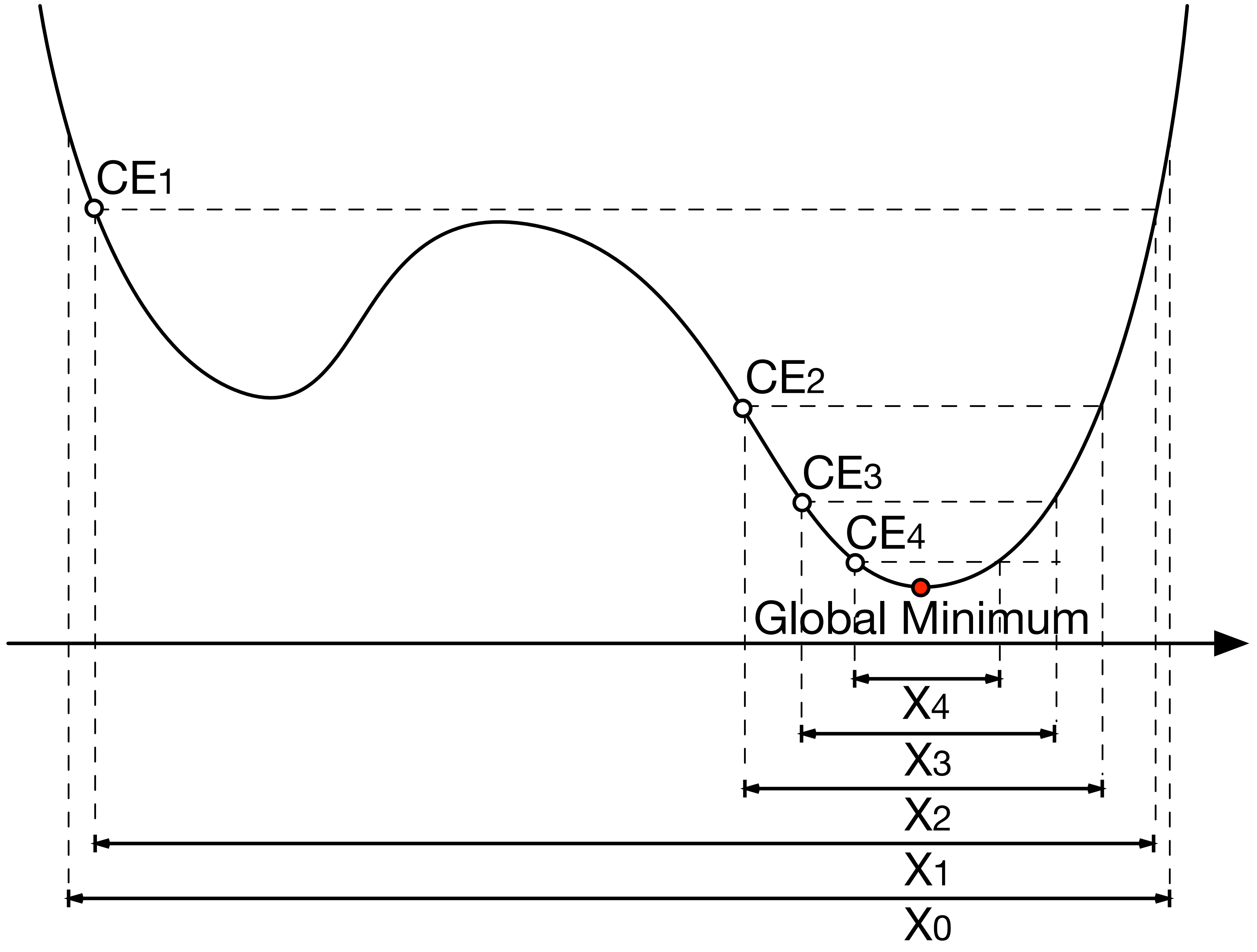
$$\delta' < \epsilon < \delta$$

Idea 2: Double-sided Error Control

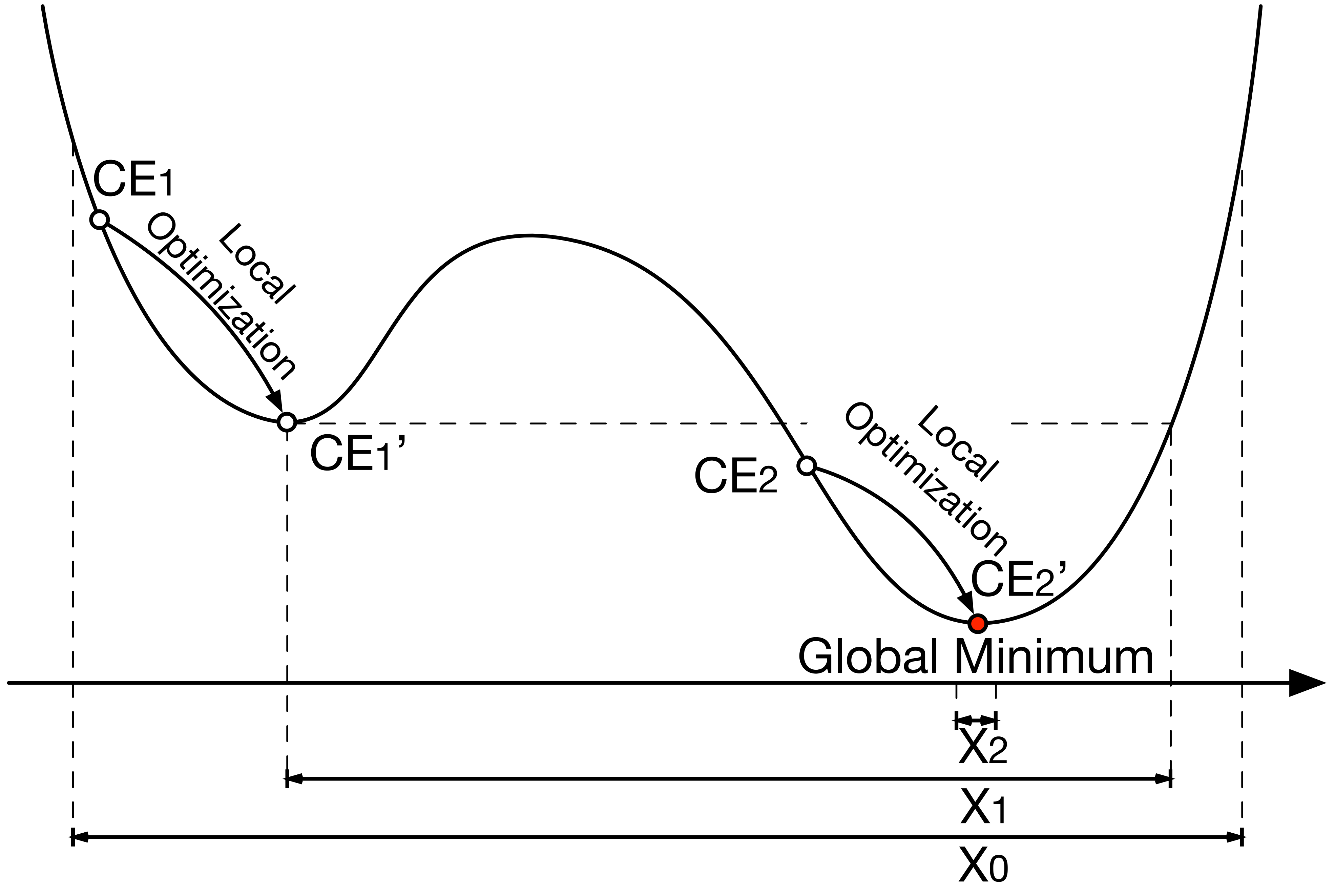
Algorithm 2 $\forall$-Clause Pruning

function $\textsc{Prune}(B_x,\ B_y,\ \forall y \bigvee_{i=0}^{k} f_i(x,y) \geq 0,\ \delta',\ \varepsilon,\ \delta)$
    repeat
        $B_x^{\text{prev}} \leftarrow B_x$
        $\psi \leftarrow \bigwedge_i f_i(x,y) < 0$
        $\psi^{+\varepsilon} \leftarrow \mathsf{Strengthen}(\psi, \varepsilon)$
        $b \leftarrow \mathsf{Solve}(y, \psi^{+\varepsilon}, \delta')$ ▷ $0 < \delta' < \varepsilon < \delta$ should hold.
        if $b = \emptyset$ **then**
            return B_x ▷ No counterexample found, stop pruning.
        end if
        for $i \in \{0, ..., k\}$ **do**
            $B_i \leftarrow B_x \cap \mathsf{Prune}\Big(B_x, f_i(x,b) \geq 0\Big)$
        end for
        $B_x \leftarrow \bigsqcup_{i=0}^{k} B_i$
    until $B_x \neq B_x^{\text{prev}}$
    return B_x
end function

Idea 3: Local Optimization



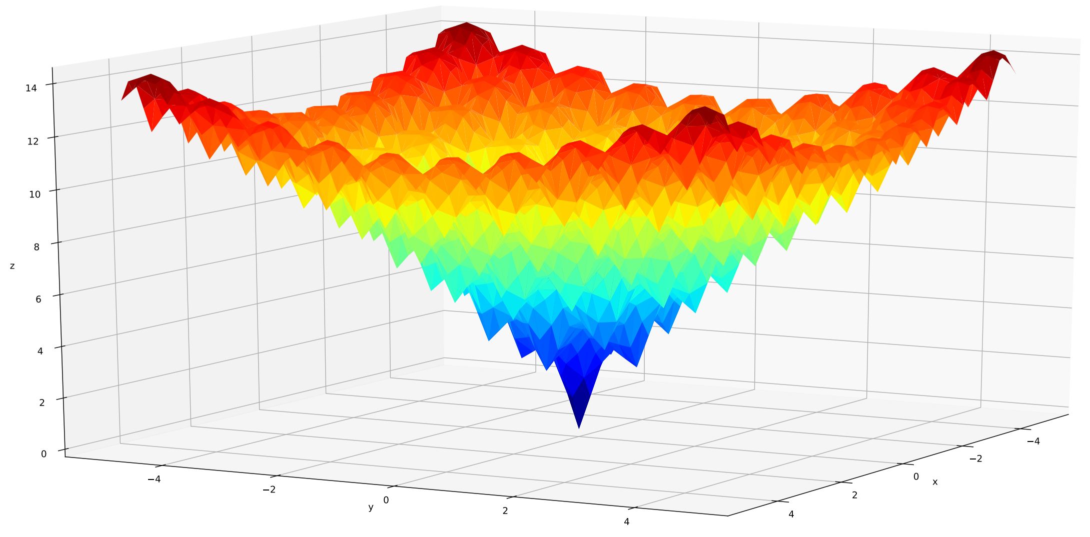
(a) Without local optimization.



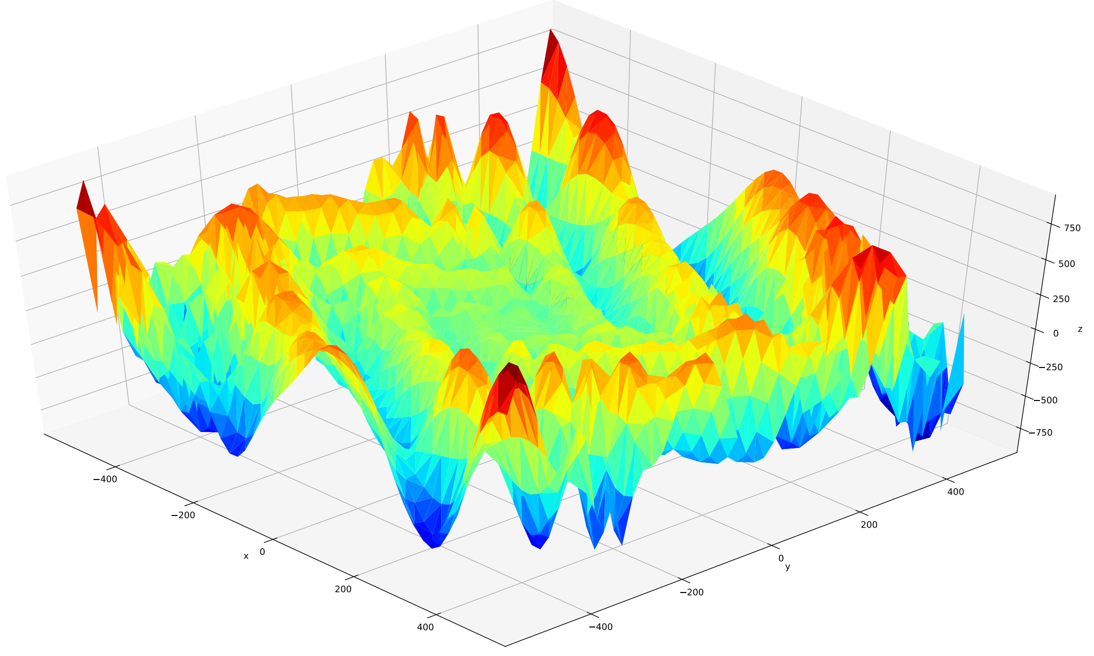
(b) With local optimization.

Fig. 1: Illustrations of the pruning algorithm for CNF$^{\forall}$-formula with and without using local optimization.

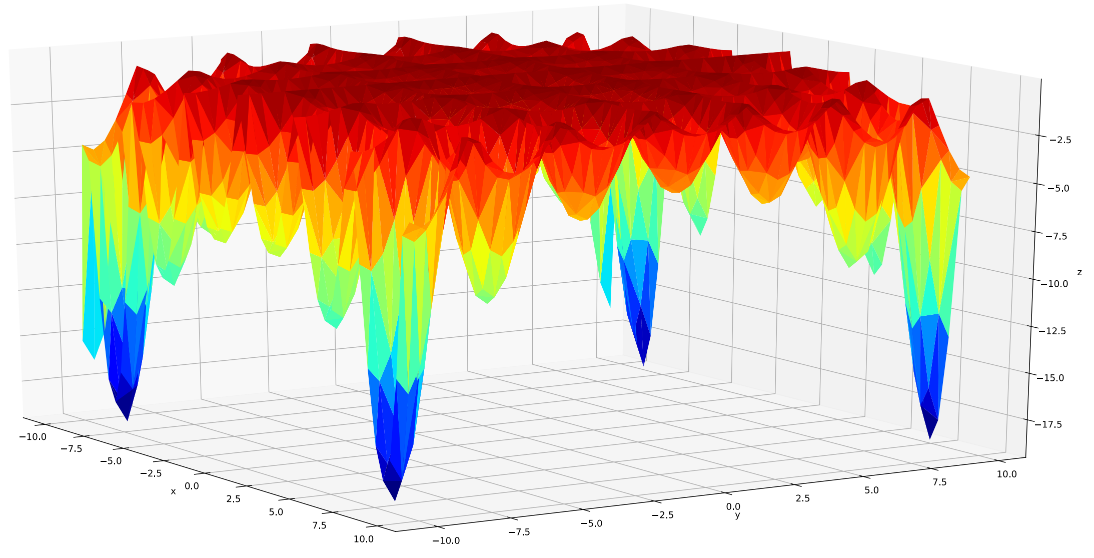
Case Study: Nonlinear Global Optimization



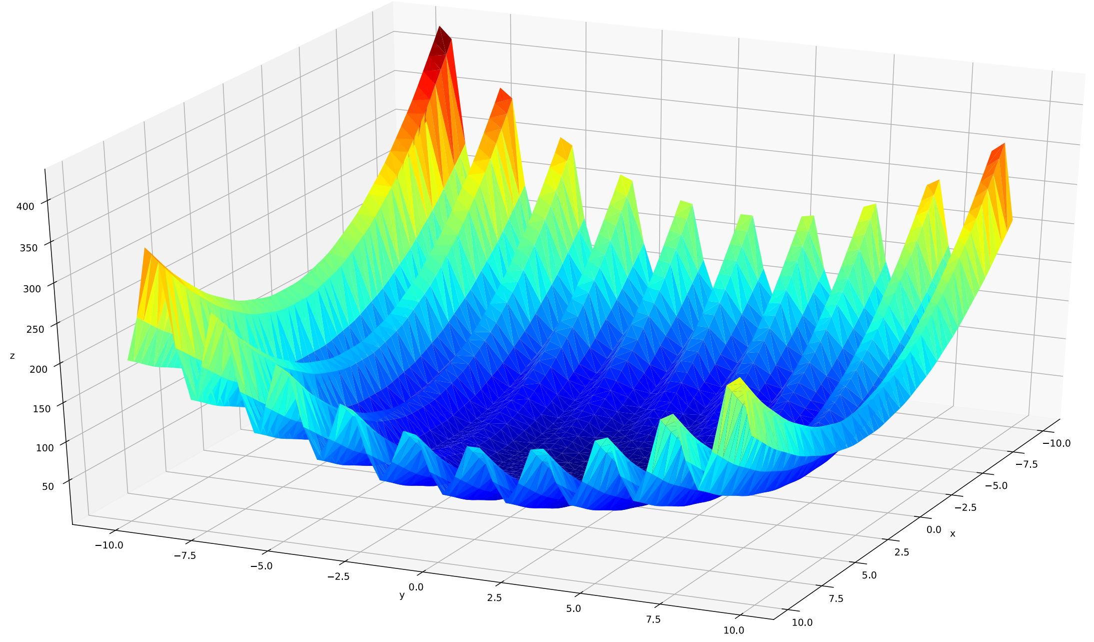
(a) Ackley Function.



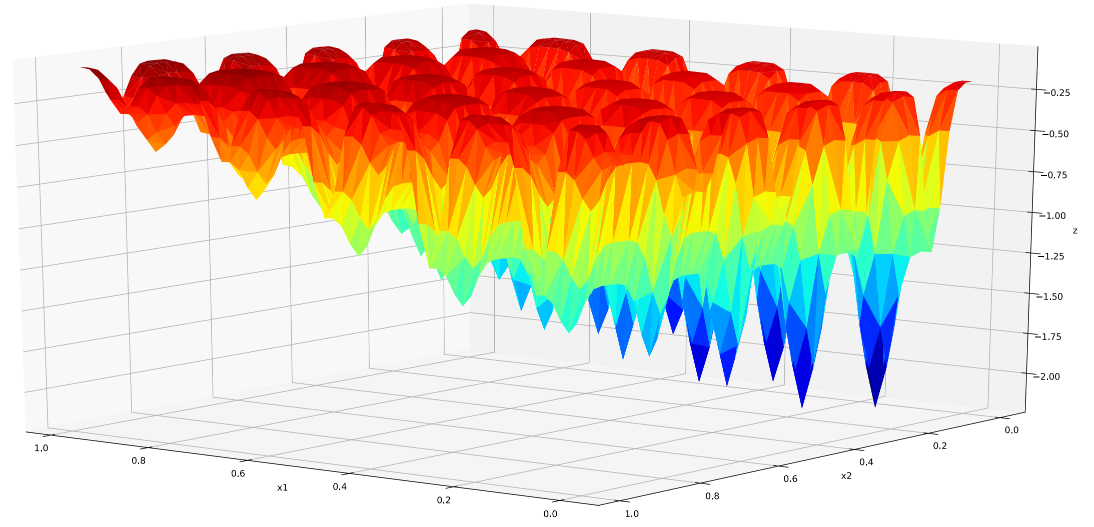
(b) EggHolder Function.



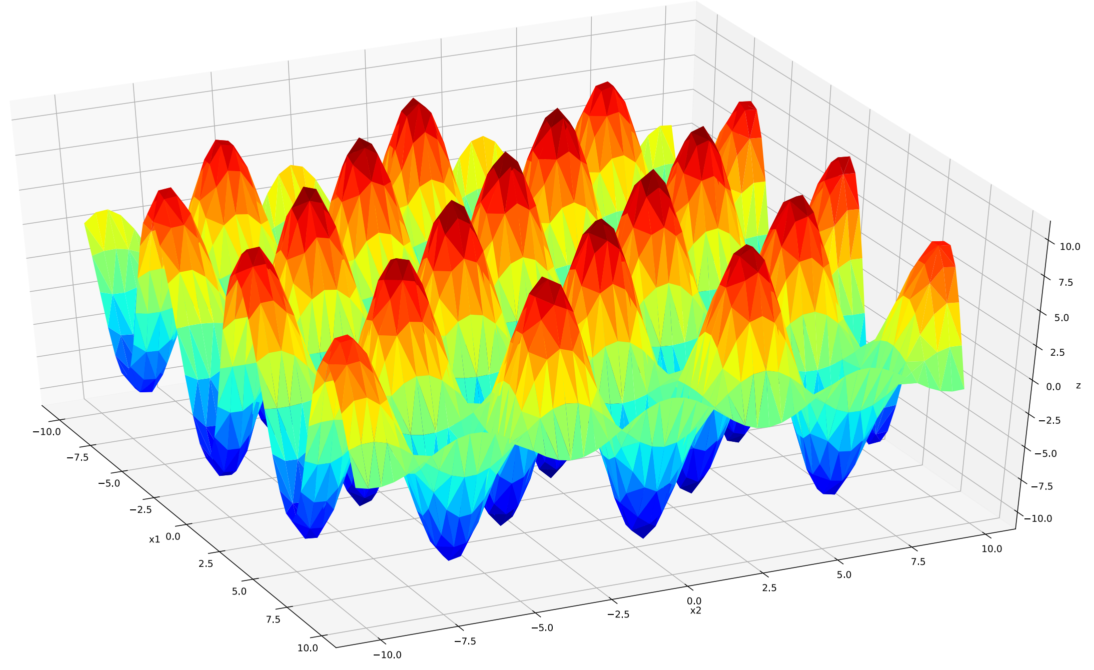
(c) Holder Table2 Function.



(d) Levi N13 Function.



(e) Ripple 1 Function.



(f) Testtube Holder Function.

Case Study: Nonlinear Global Optimization

Name	Solution			Time (sec)		
	Global	No L-Opt.	L-Opt.	No L-Opt.	L-Opt.	Speed Up
Ackley 2D	0.00000	0.00000	0.00000	0.0579	0.0047	12.32
Ackley 4D	0.00000	0.00005	0.00000	8.2256	0.1930	42.62
Aluffi Pentini	-0.35230	-0.35231	-0.35239	0.0321	0.1868	0.17
Beale	0.00000	0.00003	0.00000	0.0317	0.0615	0.52
Bohachevsky1	0.00000	0.00006	0.00000	0.0094	0.0020	4.70
Booth	0.00000	0.00006	0.00000	0.5035	0.0020	251.75
Brent	0.00000	0.00006	0.00000	0.0095	0.0017	5.59
Bukin6	0.00000	0.00003	0.00003	0.0093	0.0083	1.12
Cross in Tray	-2.06261	-2.06254	-2.06260	0.5669	0.1623	3.49
Easom	-1.00000	-1.00000	-1.00000	0.0061	0.0030	2.03
EggHolder	-959.64070	-959.64030	-959.64031	0.0446	0.0211	2.11
Holder Table2	-19.20850	-19.20846	-19.20845	52.9152	41.7004	1.27
Levi N13	0.00000	0.00000	0.00000	0.1383	0.0034	40.68
Ripple 1	-2.20000	-2.20000	-2.20000	0.0059	0.0065	0.91
Schaffer F6	0.00000	0.00004	0.00000	0.0531	0.0056	9.48
Testtube Holder	-10.87230	-10.87227	-10.87230	0.0636	0.0035	18.17
Trefethen	-3.30687	-3.30681	-3.30685	3.0689	1.4916	2.06
W Wavy	0.00000	0.00000	0.00000	0.1234	0.0138	8.94
Zettl	-0.00379	-0.00375	-0.00379	0.0070	0.0069	1.01
Rosenbrock Cubic	0.00000	0.00005	0.00002	0.0045	0.0036	1.25
Rosenbrock Disk	0.00000	0.00002	0.00000	0.0036	0.0028	1.29
Mishra Bird	-106.76454	-106.76449	-106.76451	1.8496	0.9122	2.03
Townsend	-2.02399	-2.02385	-2.02390	2.6216	0.5817	4.51
Simionescu	-0.07262	-0.07199	-0.07200	0.0064	0.0048	1.33

Table 1: Experimental results for nonlinear global optimization problems: The first 19 problems (Ackley 2D – Zettl) are unconstrained optimization problems and the last five problems (Rosenbrock Cubic – Simionescu) are constrained optimization problems. We ran our prototype solver over those instances with and without local-optimization option ("L-Opt." and "No L-Opt." columns) and compared the results. We chose $\delta = 0.0001$ for all instances.

Case Study: Synthesizing Lyapunov Function

Problem: Find a Lyapunov function for a dynamical system, v : X → $\mathbb{R}^+$, which satisfies the following condition:

$$\forall \boldsymbol{x} \in X \setminus \mathbf{0} \ v(\boldsymbol{x})(\mathbf{0}) = 0$$
$$\forall \boldsymbol{x} \in X \ \nabla v(\boldsymbol{x}(t))^T \cdot f_i(\boldsymbol{x}(t)) \leq 0.$$

where the system is described by a system of ODEs:

$$\dot{\boldsymbol{x}}(t) = f_i(\boldsymbol{x}(t)), \quad \forall \boldsymbol{x}(t) \in X_i.$$

Case Study: Synthesizing Lyapunov Function

Normalized Pendulum Given a standard pendulum system with normalized parameters

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) - x_2 \end{bmatrix}$$

and a quadratic template for a Lyapunov function $v(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x} = c_1 x_1 x_2 + c_2 x_1^2 + c_3 x_2^2$, we can encode this synthesis problem into the following $\exists\forall$-formula:

$$\begin{aligned}
\exists c_1 c_2 c_3 \ \forall x_1 x_2 \ & [((50 c_3 x_1 x_2 + 50 x_1^2 c_1 + 50 x_2^2 c_2 > 0.5) \wedge \\
& (100 c_1 x_1 x_2 + 50 x_2 c_3 + (-x_2 - \sin(x_1)(50 x_1 c_3 + 100 x_2 c_2)) < -0.5)) \vee \\
& \neg((0.01 \leq x_1^2 + x_2^2) \wedge (x_1^2 + x_2^2 \leq 1))]
\end{aligned}$$

Our prototype solver takes 44.184 seconds to synthesize the following function as a solution to the problem for the bound $\|\boldsymbol{x}\| \in [0.1, 1.0]$ and $c_i \in [0.1, 100]$ using $\delta = 0.05$:

$$v = 40.6843 x_1 x_2 + 35.6870 x_1^2 + 84.3906 x_2^2.$$

Case Study: Synthesizing Lyapunov Function

Damped Mathieu System Mathieu dynamics are time-varying and defined by the following ODEs:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_2 - (2+\sin(t))x_1 \end{bmatrix}.$$

Using a quadratic template for a Lyapunov function $v(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x} = c_1 x_1 x_2 + c_2 x_1^2 + c_3 x_2^2$, we can encode this synthesis problem into the following $\exists\forall$-formula:

$$\begin{aligned} \exists c_1 c_2 c_3 \; \forall x_1 x_2 t \; [&(50x_1x_2c_2 + 50x_1^2c_1 + 50x_2^2c_3 > 0) \wedge \\ &(100c_1x_1x_2 + 50x_2c_2 + (-x_2 - x_1(2+\sin(t)))(50x_1c_2 + 100x_2c_3) < 0) \\ &\vee \neg((0.01 \le x_1^2 + x_2^2) \wedge (0.1 \le t) \wedge (t \le 1) \wedge (x_1^2 + x_2^2 \le 1))] \end{aligned}$$

Our prototype solver takes 26.533 seconds to synthesize the following function as a solution to the problem for the bound $\|\boldsymbol{x}\| \in [0.1, 1.0]$, $t \in [0.1, 1.0]$, and $c_i \in [45, 98]$ using $\delta = 0.05$:

$$V = 54.6950x_1x_2 + 90.2849x_1^2 + 50.5376x_2^2.$$

To conclude:

To conclude:

1. Road Testing is expensive & not scalable.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.
3. But we still need advanced techniques such as white-box fuzzing, verification & synthesis.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.
3. But we still need advanced techniques such as white-box fuzzing, verification & synthesis.
4. Find a good modeling tool so that you can extract symbolic representations.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.
3. But we still need advanced techniques such as

   white-box fuzzing, verification & synthesis.
4. Find a good modeling tool so that you can extract symbolic representations.
5. Many interesting control problems can be encoded into first-order logic formulas.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.
3. But we still need advanced techniques such as

   white-box fuzzing, verification & synthesis.
4. Find a good modeling tool so that you can extract symbolic representations.
5. Many interesting control problems can be encoded into first-order logic formulas.
6. Delta-decision problems:

   For verification, using delta-weakening allows us to fix “near-failure” systems.

   For synthesis, a dual notion of delta-strengthening produces robust designs.

To conclude:

1. Road Testing is expensive & not scalable.
2. Simulation is the next hope.
3. But we still need advanced techniques such as

   white-box fuzzing, verification & synthesis.
4. Find a good modeling tool so that you can extract symbolic representations.
5. Many interesting control problems can be encoded into first-order logic formulas.
6. Delta-decision problems:

   For verification, using delta-weakening allows us to fix "near-failure" systems.

   For synthesis, a dual notion of delta-strengthening produces robust designs.
7. We have an implementation (dReal) which can handle $\exists$ and $\exists\forall$ formulas.

   ($\exists\forall\exists$ support is work-in-progress).