Journal of Machine Learning Research 23 (2022) 1-45 Submitted 7/21; Revised 12/21; Published 3/22

OVERT: An Algorithm for Safety Verification of Neural Network Control Policies for Nonlinear Systems

Chelsea Sidrane* CSIDRANE@STANFORD.EDU
Amir Maleki* AMIR.MALEKI@STANFORD.EDU
Ahmed Irfan IRFAN@CS.STANFORD.EDU
Mykel J. Kochenderfer MYKEL@STANFORD.EDU
Aeronautics & Astronautics Department, Stanford University
496 Lomita Mall, Stanford, CA 94305 USA
(denotes equal contribution)*

Editor: Amos Storkey

Abstract

Deep learning methods can be used to produce control policies, but certifying their safety is challenging. The resulting networks are nonlinear and often very large. In response to this challenge, we present OVERT: a *sound* algorithm for safety verification of nonlinear discrete-time closed loop dynamical systems with neural network control policies. The novelty of OVERT lies in combining ideas from the classical formal methods literature with ideas from the newer neural network verification literature. The central concept of OVERT is to abstract nonlinear functions with a set of optimally tight piecewise linear bounds. Such piecewise linear bounds are designed for seamless integration into ReLU neural network verification tools. OVERT can be used to prove bounded-time safety properties by either computing reachable sets or solving feasibility queries directly. We demonstrate various examples of safety verification for several classical benchmark examples. OVERT compares favorably to existing methods both in computation time and in tightness of the reachable set.

Keywords: neural network verification, reachability, nonlinear systems, control

1. Introduction

Deep learning has found application in a wide variety of fields, from computer vision and natural language processing to control of autonomous agents. In particular, recent innovations in deep reinforcement learning have yielded impressive results, such as training neural networks to play Atari video games (Mnih et al., 2015) and board games like Go (Silver et al., 2016). As a result, there is increasing interest in applying deep learning to safety-critical control problems such as autonomous driving (Chen et al., 2015; Bojarski et al., 2016) and aircraft collision avoidance (Julian et al., 2016). Although neural networks can provide flexible representations of complex control strategies, there is concern that they can result in unexpected behavior (Szegedy et al., 2014; Papernot et al., 2016).

Testing can play an important role in identifying undesirable behavior in a system, but testing alone is not sufficient to prove the absence of failures. It is possible that rare, catastrophic failures exist that intelligent testing schemes are required to discover (Corso et al., 2021). Instead, formal verification can be used on a model of the system in order to prove the absence of failures. More specifically, in this work, we develop an algorithm to prove or disprove safety properties of nonlinear discrete-time dynamical systems with deep neural network control policies. The incorporation of nonlinear functions as well as neural networks makes these systems challenging to verify. Katz et al. (2017) demonstrated that verifying ReLU neural networks alone is an NP-complete problem. Tools like those developed by Cimatti et al. (2018) can verify properties of discrete-time systems with nonlinear functions. However, these tools cannot efficiently handle the complexity of neural networks, which can contain thousands of smooth nonlinear or piecewise-linear activation functions (Katz et al., 2017). This problem has led to the development a family of neural network verification tools that are able to efficiently verify properties of neural networks (Liu et al., 2021). Together with a small but growing body of research, this work aims to bridge the gap between the existing literature on verification of closed-loop systems and literature on neural network verification.

There is extensive literature for formal verification of discrete-time closed-loop systems (also known as *transition systems*) (Clarke et al., 2018). Much of this work focuses on linear systems. Although nonlinear model checkers such as the tool developed by Cimatti et al. (2018) can handle nonlinear dynamics, they are not designed for use with neural networks. Even if a network can be encoded into the tool, which may not be possible, the large number of nonlinear activation functions present difficulties. For example, if the network has ReLU activation functions, each would be expanded eagerly to a disjunction, leading to an intractable runtime.

The inability of existing model checking tools or other automated reasoning tools to verify neural networks has led to a recent surge in development of verification tools that are capable of reasoning about neural networks with ReLU activation functions (Katz et al., 2017; Liu et al., 2021). When the network that we would like to verify represents a control policy, these tools may be used to verify input-output properties of the control policy in isolation. However, verifying properties of the control policy in isolation does not address the entire closed-loop system of which the control policy is just one part. Our work uses these recent developments in neural network verification in order to reason about the closed-loop system.

There is prior work that also uses new neural network verification tools to reason about the closed-loop system. Some of these works can be used for discrete-time systems with ReLU-neural-network controllers (Tran et al., 2020; Julian and Kochenderfer, 2021). The methods presented in both Tran et al. (2020) and Julian and Kochenderfer (2021) work by computing polytopes containing the reachable set of the control policy and the dynamics at each timestep in an iterative fashion. These methods are complementary to ours, but have two main limitations. The first is that they are based on hybrid systems reachability tools, which scale poorly with the number of state variables (Chen et al., 2013). The second is that the underlying reachability algorithm based on iterative computation of concrete reachable sets leads to a compounding looseness of the approximation, known as the wrapping effect (Neumaier, 1993). To fix this, the initial set may be split into smaller cells across each dimension, but this too can lead to poor scaling in the number of state variables. The methods developed here, in contrast, have either no explicit dependence on the number of state variables or only linear dependence. All methods presented mitigate the wrapping effect by preserving a symbolic representation across multiple timesteps.

There is related work that assumes piecewise-linear system dynamics, or approximates (neither underapproximates nor overapproximates) potentially nonlinear system dynamics using data-driven models (Dutta et al., 2018; Akintunde et al., 2018, 2020; Xiang et al., 2018c). However, if the system dynamics are approximated with a data driven model, proving that there are no counterexamples for the approximate system does not allow a sound claim that there are no counterexamples for the original nonlinear dynamics. Our method makes an overapproximation that *does* allow such a sound claim to be made. Finally, there is adjacent work that addresses continuous-time systems (Huang et al., 2019; Dutta et al., 2019) and sigmoid-based neural networks (Ivanov et al., 2019). These methods are closely related but not directly applicable to the problem at hand.

Contributions. We present OVERT: a method to reason about nonlinear discrete-time closed-loop systems that contain neural network control policies. OVERT overapproximates nonlinear dynamical systems using piecewise-linear relations, making the entire closed-loop system a conjunction of piecewise-linear relations. The resultant system of piecewise-linear relations can be efficiently reasoned about using ReLU neural network verification tools. The main features of our work can be summarized as follows: i) The use of an overapproximation makes our algorithm *sound*; a proof of no counterexamples for the approximation implies no counterexamples for the original system. ii) The algorithm provides an optimally tight formulation to overapproximate a nonlinear system with a piecewise linear counterpart. iii) Our formulation is amenable to use with various neural network verification tools including NSVerify (Lomuscio and Maganti, 2017), MIPVerify (Tjeng et al., 2019), and Reluplex (Katz et al., 2017). In our experiments we showcase examples of various complexity and solve the verification problem using a mixed-integer programming approach inspired by Tjeng et al. (2019). iv) We prove bounded-time safety properties for the closed-loop system by unrolling the system in time. Reachability queries can be solved directly as feasibility problems or by explicitly computing the reachable set. Neither approach requires gridding the state space which can lead to exponentially many sub-problems. v) The use of hybrid-symbolic reachable set computation preserves both tightness of the reachable set and tractability of its computation. vi) We have released two Julia packages, OVERT.jl and OVERTVerify.jl, which provide implementations of the sound overapproximation and verification algorithms, respectively.

2. Background

This paper borrows ideas from formal verification, control theory, and machine learning. This background section reviews prerequisite concepts and terminology.

2.1 Closed-Loop Systems

A *closed-loop system* is a *dynamical system* paired with a *feedback control policy.* The term *dynamical system* originates in control theory and describes a model of a stateful system that evolves in time (Franklin et al., 2009). To illustrate, we use a system of an inverted pendulum as a running example. The state, $\mathbf{x}$, of the inverted pendulum may be represented by the angle of the pendulum, θ, and the angular velocity of the pendulum, $\dot{\theta}$: $\mathbf{x} = [\theta, \dot{\theta}]^T$.

The set of all possible states is $[-\pi, \pi] \times \mathbb{R}$. We can also define an initial set of states, also known as an initial condition. A dynamical system evolves in time according to an update function, which is commonly called the equation of motion or the plant in control theory. A dynamical system may be discrete-time or continuous-time. We focus on discrete-time systems. For discrete-time systems, the update function $\mathbf{h}$ takes as input the state at time t and returns the state at time $t+1$: $\mathbf{x}_{t+1} = \mathbf{h}(\mathbf{x}_t)$. Additionally, a dynamical system may be linear or nonlinear. A nonlinear system has a nonlinear update function $\mathbf{h}$. In this work, we consider nonlinear systems where $\mathbf{h}$ may contain polynomials, transcendentals (sin, cos, exp), piecewise-linear functions, and compositions thereof. The dynamical systems that arise in classical mechanics can typically be expressed using a function in this class.

A dynamical system may also incorporate external inputs, called control inputs, that allow us to generate desired behavior. The update function is then $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ where $\mathbf{u}_t$ is the control input at time t produced by the *control policy.* The control policy is also called the controller in control theory or the policy in reinforcement learning. In the inverted pendulum example, a control input could be torque applied at the pivot joint. This torque may be used to produce desired behavior, such as balancing the pendulum upright. A *feedback* control policy produces control input $\mathbf{u}_t$ as a function of the current state, $\mathbf{u}_t = \mathbf{c}(\mathbf{x}_t)$, making it reactive to changes in the system. The prevalence of deep learning and deep reinforcement learning has led to the use of neural network based control policies. Consequently, this work focuses on nonlinear discrete-time closed-loop systems with neural network control policies.

2.2 Transition Systems

Readers from formal verification and model checking may be familiar with the notion of a *transition system.* A discrete-time dynamical system, paired with an initial condition (initial set of states), may be equivalently modeled as a symbolic transition system. A symbolic transition system has states that evolve in discrete time according to a transition relation, beginning from a specific initial set. Such a symbolic transition system is defined by a tuple (X, I, TR), where X is a set of state variables (Baier and Katoen, 2008; Cimatti et al., 2018). Here, $I(X)$ represents the set of initial states using a formula over the variables in X. A *formula* is a Boolean combination (using standard logical operators) of constraints. A constraint is of the form $p \bowtie 0$, where p is a polynomial (summation of monomials) and $\bowtie \in \{\leq, <, =, \geq, >\}$. A *transition relation* TR describes how the state evolves in time. The transition relation is symbolically defined as a formula over the state variables associated with the current and next time step. Let $\mathbf{x}_t$ be the current state at time t and $\mathbf{x}_{t+1}$ be a next state at time $t+1$. The formula $TR(\mathbf{x}_t, \mathbf{x}_{t+1})$ returns true if state $\mathbf{x}_{t+1}$ is a valid successor of state $\mathbf{x}_t$. A transition system does not require that there be a single valid successor state for a given state. The update function of a dynamical system corresponds to the transition relation when modeling a dynamical system as a transition system.

2.3 Proving Properties

We prove safety properties and goal-reaching properties for closed-loop systems, which can equivalently be described as model checking for transition systems; specifically, bounded-time model checking (De Moura et al., 2002). For readers from control theory, this can also be described as solving discrete-time *reachability problems* (Clarke et al., 2018). A reachability problem reasons about whether the set of states reachable over time intersects with an unsafe set, or reaches a goal set. For example, consider modeling the motion of an inverted pendulum as a transition system or discrete-time dynamical system. The properties we prove can be conditions that we would like to always hold, known as invariant properties, e.g. *it is always true that the angle of the pendulum with respect to vertical is greater than* $-5°$, or conditions that we would *eventually* like to hold, such as *the angle of the pendulum with respect to vertical is eventually greater than* $-5°$ (Øhrstrøm and Hasle, 1995). Formally, a property is defined as a Boolean-valued predicate over the states of the system, e.g. $\theta \geq -5°$, which is then combined with a *temporal modal operator* such as G (Globally) or F (Finally) as well as a finite range of timesteps to form a bounded temporal property. For example, $F_{1:10}(\theta \geq -5°)$ expresses that θ must be greater than $-5°$ at some point in the first 10 timesteps. The notation $G_{1:10}(\theta \geq -5°)$ expresses that θ must be greater than $-5°$ at all steps in the first 10 timesteps. OVERT has the capability to reason about a larger fragment of temporal logic than just the F and G operators. In this paper, the G operator is used to express safety properties, and the F operator is used to express goal-reaching properties.

In order to prove that a property (e.g., $G_{1:10}\phi$) holds, the negation of the property must be proven unsatisfiable (UNSAT) (Enderton, 1972). If a *trace* of the system is found that satisfies the negation of the property, this trace is a counter-example demonstrating where the property is violated. A trace may also be described as a trajectory in the control theory context. The F and G operators are dual, meaning that $\neg G\phi$ is equivalent to $F\neg\phi$ as well as $\neg F\phi$ is equivalent to $G\neg\phi$. In order to prove $G_{1:10}\phi$ holds, we would test whether the complement, $F_{1:10}\neg\phi$, is unsatisfiable. At each timestep $t \in 1:10$, the property $\neg\phi$ should be unsatisfiable.

3. Methods

Consider a closed-loop discrete-time dynamical system with update function:

$$\mathbf{x}_{t+1} = \mathbf{h}(\mathbf{x}_t)$$

where $\mathbf{x}_t = [x_{t,1}, x_{t,2}, \ldots, x_{t,n}] \in \mathcal{X} \subseteq \mathbb{R}^n$ denotes the state vector at time t and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^n$ denotes the state update function. The closed-loop update function $\mathbf{h}$ is comprised of the dynamics function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ and the control policy function $\mathbf{u} = \mathbf{c}(\mathbf{x})$:

$$\mathbf{h}(\mathbf{x}) = \mathbf{f}(\mathbf{x}, \mathbf{c}(\mathbf{x}))$$

For now, we assume that the control policy $\mathbf{c}(\mathbf{x})$ is represented by a neural network with piecewise linear activation functions (e.g., ReLU). We will discuss how other activation functions may be used in Section 5. The dynamics function $\mathbf{f}$ may contain nonlinearities that prohibit the application of neural verifications tools such as ReluPlex (Katz et al., 2017), MIPVerify (Tjeng et al., 2019), and ReluVal (Wang et al., 2018) because these tools only support linear and piecewise linear operations. To circumvent this problem, we construct an optimally tight *abstraction* of the dynamics function that only contains linear and piecewise linear relations (see Section 3.1). The abstraction of the dynamics function using linear and piecewise linear relations corresponds to creating an overapproximation of the set of values forming the image of the update function that varies based upon the input set.

Once the dynamics have been abstracted, the closed-loop system is unrolled in time. Reachability queries can then be posed to assess safety properties or goal-reaching properties (see Section 3.5). The resulting system consists only of piecewise-linear constraints and is suitable for translation into many ReLU neural network verification tools. In this work, a framework is developed to encode the unrolled system as a mixed integer program (see Section 3.6), which is the basis of several ReLU neural network verification tools (Tjeng et al., 2019; Lomuscio and Maganti, 2017; Akintunde et al., 2018). The mixed integer program is then solved with Gurobi (Gurobi Optimization (2020)).

3.1 Constructing Multidimensional Overapproximations

Many modern nonlinear model checking methods involve *abstraction* of nonlinear relations and functions. Abstraction is the process of deriving an approximate representation of the transition system that is more computationally tractable for verification (Clarke et al., 2018). Importantly, an abstraction is constructed such that proving certain properties of the *abstracted* system automatically proves that the same properties hold for the original system, also called the *concrete system*. Such an abstraction is also called an *overapproximation* because the set of problem states that the abstracted system may visit, $\mathcal{R}_A$, is a superset of the set that the original system may visit, $\mathcal{R}_O \subseteq \mathcal{R}_A$. In this work, we abstract the closed-loop system using a *relational overapproximation* (also known as a *relational abstraction*). The details of how this is done will be explained in this section.

OVERT can overapproximate functions with both multidimensional inputs and multidimensional outputs. The update function $\mathbf{f}$ may be represented as a vector of functions $[f_1, f_2, \dots, f_n]^T$ each of which maps $\mathbb{R}^n \to \mathbb{R}$. The update function for the ith component of the state $\mathbf{x}_{t,i}$ is denoted $\mathbf{x}_{t+1,i} = f_i(\mathbf{x}_t, \mathbf{c}(\mathbf{x}_t))$. The algorithm presented is applied to each update function component f_i separately. The subscript is dropped for simplicity in the description below. The algorithm consists of two main steps: rewriting f from a single complex equation into a series of simpler constraints and then overapproximating any nonlinear constraints. Both the rewriting and approximation steps are described below.

3.1.1 Rewriting

We rewrite the dynamics function f as a conjunction of constraints (relations), where each constraint is either an elementary function $e(x)$ (such as $\sin(x)$, $\log(x)$, $\exp(x)$, x^3), an algebraic binary operation (addition $(+)$ or subtraction $(-)$), or a single-juncture piecewise linear function (such as $\max(x, y)$, $\min(x, y)$). Many common dynamics functions that arise in classical mechanics can be rewritten this way by introducing auxiliary variables. For example, consider the function

$$f(x, y, z) = \sin(x^2 + y - \log z) \quad \text{defined over} \quad \mathcal{X} := \{(x, y, z) \mid 1 \leq x, y, z \leq 2\}$$

The first column of Table 1 shows the outcome of the rewriting step, where function f is split into a number of linear and nonlinear equality and non-equality relations. Algorithm 1 specifies *how* this step is executed iteratively. On each iteration, we isolate the operator and operands using the function Split. Depending on the type of operator, the algorithm proceeds. Apart from scalar multiplication, we cannot use multiplication ($\times$) and division ($\div$) operations because it will lead to nonlinear constraints in the approximation step. Therefore, the function ConvertMultiplication converts these operations into an exponential of a sum of logarithms using the following two identities:

$$x \times y = \exp\left(\log(x) + \log(y)\right), \quad \forall x, y > 0$$
$$x/y = \exp\left(\log(x) - \log(y)\right), \quad \forall x, y > 0$$

A transformation of variables on x and y is used so that the domain for each log is $[\xi, 1+\xi]$, where ξ is a small positive constant, so that log may be applied.

Algorithm 1 Rewriting

```
1:  function Rewrite(expr, container)
2:      if expr is a variable or number
3:          return expr, container
4:      else if expr is affine
5:          $v \leftarrow$ generate new variable
6:          Add ($v$ = expr) to container
7:          return $v$, container
8:      else if operator of expr is $\times$ or $\div$
9:          Convert expr using log and exp
10:         return Rewrite(expr, container)
11:     else if operator of expr is $+$ or $-$
12:         operator, operands = split(expr)
13:         $x$, container $\leftarrow$ Rewrite(operands[1], container)
14:         $y$, container $\leftarrow$ Rewrite(operands[2], container)
15:         $v \leftarrow$ generate new variable
16:         Add (operator($x$,$y$) = $v$) to container
17:         return $v$, container
18:     else if  operator is a supported nonlinear unary function
19:         operator, operand $\leftarrow$ split(expr)
20:         $x$, container $\leftarrow$ Rewrite(operand, container)
21:         $v \leftarrow$ generate new variable
22:         Add (operator(x) = $v$) to container
23:         return $v$, container
```

3.1.2 Approximation

This step is comprised of processing the constraints obtained from the re-writing step that contain nonlinear functions. For each nonlinear equation such as $v_1 = \sin(x)$, the equation is replaced with two relations forming upper and lower bounds: $v_1 \leq g_{\sin_{UB}}(x)$ and $v_1 \geq g_{\sin_{LB}}(x)$. Because each nonlinear *equation* is replaced with two piecewise linear *relations* we refer to the result as a *relational overapproximation.* The second column in Table 1 shows the result of the approximation step on an example.

Algorithm 2 specifies how this process is performed. The two functions GetUpperBound and GetLowerBound compute piecewise linear upperbound and lowerbound functions, respectively. More precisely, for each nonlinear elementary function $e(x)$ defined over the interval $[a, b]$, functions GetUpperBound and GetLowerBound compute $g_{e_{UB}}$ and $g_{e_{LB}}$, respectively, such that

$$g_{e_{LB}}(x) \leq e(x) \leq g_{e_{UB}}(x) \quad \forall x \in [a, b] \tag{1}$$

Algorithm 2 Approximation

```
1: function APPROXIMATE(container, parameters)
2:     approximation ← $\emptyset$
3:     for ($v$ = expr) in container
4:         if expr is nonlinear
5:             Add ($v \leq$ GETUPPERBOUND(expr, parameters)) to approximation
6:             Add ($v \geq$ GETLOWERBOUND(expr, parameters)) to approximation
7:         else
8:             Add ($v$ = expr) to approximation
9:     return approximation
```

Table 1: OVERT applied to f defined in Section 3.1.1.

Step 1	**Step 2**
$f(x, y, z) = v_{11}$	$f(x, y, z) \approx v_{11}$
$v_{11} = \sin(v_8)$	$v_{11} \leq v_9$
	$v_{11} \geq v_{10}$
	$v_9 = g_{\sin_{UB}}(v_8)$
	$v_{10} = g_{\sin_{LB}}(v_8)$
$v_8 = v_4 - v_7$	$v_8 = v_4 - v_7$
$v_7 = \log(z)$	$v_7 \leq v_5$
	$v_7 \geq v_6$
	$v_5 = g_{\log_{UB}}(z)$
	$v_6 = g_{\log_{LB}}(z)$
$v_4 = v_3 + y$	$v_4 = v_3 + y$
$v_3 = x^2$	$v_3 \leq v_1$
	$v_3 \geq v_2$
	$v_1 = g_{x^2{}_{UB}}(x)$
	$v_2 = g_{x^2{}_{LB}}(x)$

The algorithm for finding the upper-bound piecewise linear function ($g_{e_{UB}}$) and the lower-bound piecewise linear function ($g_{e_{LB}}$) used in GetUpperBound and GetLowerBound is described in Section 3.2 and Section 3.3, with a derivation in Appendix A. This step is important because the overapproximation is useful only if it is *practically* tight; i.e. the approximation error is small. Otherwise, OVERT may be unnecessarily conservative and generate spurious counter examples. A spurious counter example is when the solver reports SAT and returns a point in state space where the property does not hold for the abstracted system, but the property does hold for the true system. However, reducing approximation error requires more linear segments in the piecewise linear bound, which leads to a more computationally expensive representation. The algorithm we derive finds *optimally tight* bounds $g_{e_{LB}}(x)$ and $g_{e_{UB}}(x)$ for any one-dimensional function $e(x)$. This tight bound is efficient in its use of piecewise linear constraints.

The relational overapproximation of the dynamics together with the ReLU neural network control policy forms a system of linear and piecewise linear constraints. Such a system can be analyzed using recent algorithms introduced for ReLU neural network verification. As we will discuss in Section 3.5, we solve this system using an approach inspired by the MIPVerify algorithm (Tjeng et al., 2019). We initially prototyped using the Reluplex algorithm (Katz et al., 2017), but the MIP formulation proved faster.

3.2 Constructing One-Dimensional Overapproximations

Consider a one-dimensional nonlinear function, which in slight abuse of notation, we will call f. The upper and lower bounds described in Eq. (1) are constructed by first separating the given function f into convex and concave regions. To ensure continuity across regions of differing convexity, we enforce that the endpoints of a bound for a region of uniform convexity must be coincident with the function f. The current implementation can compute the convex and concave intervals of the following set of unary functions: $\{\cos, \sin, \exp, \log, \tanh, c/x, c^x, x^c\}$. This set of functions is expressive, however support for additional functions may be easily added. Two additional automated options also exist: implementing symbolic differentiation and using a certified interval root finding method, and independently verifying the overapproximations generated by OVERT.

Once the convex and concave regions of a function have been obtained, the process of constructing the lower bound for a convex region and the upper bound for a concave region are homologous, as is the process of constructing an upper bound for a convex region and a lower bound for a concave region. Both are described below.

3.2.1 Upper Bound for a Convex Region

An upper bound $g(x)$ for a convex region of the function $f(x)$ can be formed using the secant line connecting the ends of the interval, a and b, which follows from the definition of convexity. However, each convex interval is divided into n subintervals, $[x_{i-1}, x_i]$, for $i \in 1 : n$, and the secant line $g_i(x)$ connecting the endpoints of the subinterval is used to bound the function, as shown in Fig. 1. To ensure a tight bound, the placement of intermediate points x_i are optimized to obtain the upper bound that minimizes the area between the bound and the function:

$$\min_{x_1:x_{n-1}} \int_a^b [g(x; x_{0:n}) - f(x)] \, \mathrm{d}x$$

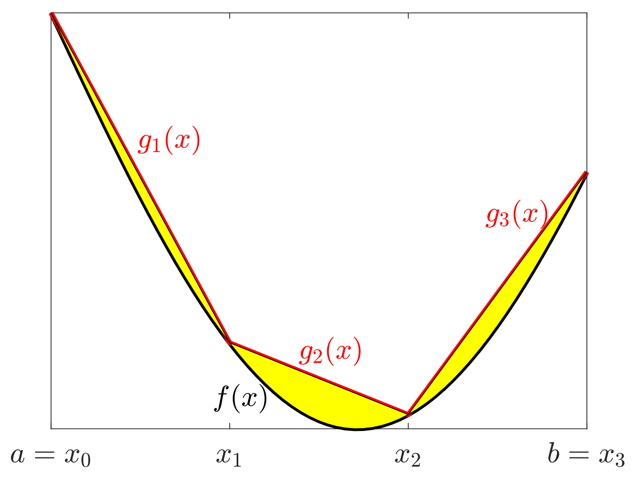


Figure 1: A convex function $f(x)$ and its upper bound $g(x)$, which is composed of $n = 3$ linear pieces: $g_1(x), g_2(x)$, and $g_3(x)$.

where the endpoints, $g(x_0 = a) = f(a)$ and $g(x_n = b) = f(b)$, are fixed. Evaluating this expression yields the optimality condition:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{2}$$

In other words, the slope of f at point x_i must match the slope of a secant connecting points x_{i-1} and x_{i+1}. A derivation can be found in Appendix A.2.1. Points (x_i) that satisfy the optimality conditions in Eq. (2) are found using NLsolve.[1] The values y_i are specified by the function:

$$y_i = f(x_i)$$

If the numerical routine incurs errors in selecting points x_i this may affect the optimality of the bound but will not affect the validity, as any secant connecting points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$ is a valid upper bound over the interval $[x_{i-1}, x_i]$. An illustration of a possible bound $g(x)$ is shown in Fig. 1.

3.2.2 Upper Bound for a Concave Region

An upper bound $g(x)$ for a concave region of the function $f(x)$ can be formed using any tangent line, which comes from the definition of convexity. However, using a single tangent line to bound the entire extent of a concave region of the function would not result in a tight bound. Consequently, each concave interval is divided into n subintervals, $[x_{i-1}, x_i]$, $i \in 1 : n$, and a tangent line $g_i(x; \alpha)$ is used to bound the function over each subinterval, where α is the point of tangency. To ensure a tight bound, the parameters of this piecewise linear bound are optimized. For any given subinterval $[x_{i-1}, x_i]$ bounded by a single tangent line segment $g_i(x; \alpha)$, placing the tangent point α at the midpoint of the subinterval yields the tightest bound. Appendix A.1 provides a proof.

1. `https://github.com/JuliaNLSolvers/NLsolve.jl`

OVERT also optimizes the placement of the points x_i , $i \in 1 : n-1$ that divide the concave interval into subintervals. The points x_i are optimized to obtain an upper bound that minimizes the area between the bound and the function:

$$\min_{x_1:x_{n-1}} \int_a^b [g(x; x_{0:n}) - f(x)] \, \mathrm{d}x$$

Minimizing this objective yields an optimality condition for the x_i:

$$x_i = h\left(\frac{x_{i-1}+x_i}{2}, \frac{x_i+x_{i+1}}{2}\right) \tag{3}$$

$$h(\alpha,\beta) = \frac{\beta f'(\beta) - \alpha f'(\alpha)}{f'(\beta) - f'(\alpha)} - \frac{f(\beta) - f(\alpha)}{f'(\beta) - f'(\alpha)} \tag{4}$$

This condition implies that the optimal bound is continuous within a region of uniform concavity: $g_i(x_i) = g_{i+1}(x_i)$. Proof as well as deriviation of equations Eq. (3) and Eq. (4) are provided in Appendix A.2. In order to ensure continuity across regions of different concavity, the bound is constrained to be tangent at the endpoints: $[a, b]$. Consequently, the bound is given by points x_i that satisfy:

$$x_0 = a \tag{5a}$$

$$x_1 = h\left(a, \frac{x_1+x_2}{2}\right) \tag{5b}$$

$$x_i = h\left(\frac{x_{i-1}+x_i}{2}, \frac{x_i+x_{i+1}}{2}\right), \quad i \in \{2, \dots, n-2\} \tag{5c}$$

$$x_{n-1} = h\left(\frac{x_{n-1}+x_n}{2}, b\right) \tag{5d}$$

$$x_n = b \tag{5e}$$

where h is given by Eq. (4). The y_i values are given by the line that is tangent at the midpoint of the ith interval,

$$g_i(x) = f'\left(\frac{x_{i-1}+x_i}{2}\right)\left(x - \frac{x_{i-1}+x_i}{2}\right) + f\left(\frac{x_{i-1}+x_i}{2}\right)$$

except for the first two and last two y_i:

$$\begin{aligned} y_0 &= f(a) \\ y_1 &= f'(a)(x_1 - a) + f(a) \\ y_i &= f'\left(\frac{x_{i-1}+x_i}{2}\right)\left(\frac{x_i - x_{i-1}}{2}\right) + f\left(\frac{x_{i-1}+x_i}{2}\right), \ i = 2, ..., (n-2) \\ y_{n-1} &= f'(b)(x_{n-1} - b) + f(b) \\ y_n &= f(b) \end{aligned}$$

In practice, we use a numerical nonlinear solver to produce points x_i satisfying the optimality constraints in Eq. (5). We then check the solution to ensure that the optimality conditions have been met. If the optimality conditions have not been met, the resultant bound is still sound, but may not be continuous over the interval. In order to restore continuity, the bound may be repaired by taking y_i to be $\max(g_i(x_i), g_{i+1}(x_i))$. Any line segment adjusted in this way is greater than or equal to the original bound $g_i(x)$ for $x \in [x_{i-1}, x_i]$ and therefore still a valid upper bound. For a lower bound for a convex function, the bound may be analogously repaired by taking $y_i = \min(g_i(x_i), g_{i+1}(x_i))$.

3.2.3 Implementation Details

Once the bounds have been constructed, an ϵ gap is added to each bound such that the bounds do not "touch" the original function. All points (x_i, y_i) that make up the the upper bound are shifted by $+\epsilon$ and all points (x_i, y_i) that make up the lower bound are shifted by $-\epsilon$. This gap helps ensure that any errors incurred during floating point computations do not compromise the validity of the bounds.

3.2.4 Examples

Figure 2 shows three different functions with their associated upperbound function $g(x)$. In Fig. 2a, $f(x) = x^2$ is plotted in black over the interval $[a, b] = [-1, 2]$. The function $f(x) = x^2$ is convex. Three upper bound function $g(x)$ are shown in color for three different values of n. One may notice that the accuracy of approximations improves significantly when n increases. In Fig. 2b, we repeat this example for $f(x) = \cos(x)$ which is concave over the interval $[a, b] = [-\pi/3, \pi/2]$. Finally, in Fig. 2c, the function $f(x) = \tanh(x)$ is bounded over the interval $[a, b] = [-\pi/2, \pi/2]$. In this case, $f(x)$ has an inflection point at $x = 0$, which means the interval is split into two sub-intervals $[-\pi/2, 0]$ and $[0, \pi/2]$. The blue and red curves show the upper bound and the lower bound of the function, respectively. In both cases, $n = 2$ linear segments were used over each sub-interval. Notice that both the upper and lower bound are continuous over $[-\pi/2, \pi/2]$.

3.3 Closed Form Expressions of Piecewise Linear Functions

The final step is to transform the set of points (x_i, y_i) representing the lower and upper bound into closed form expressions. While each individual piece of a bound $g_i(x)$ could be written as a line segment connecting points (x_i, y_i):

$$g_i(x) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} (x - x_i) + y_i, \quad x_{i-1} \le x \le x_i$$

representing the bounds in closed form is useful because we can then write the overapproximation of a nonlinear function using two linear inequalities: $g_{\sin_{LB}}(x) \le \sin(x) \le g_{\sin_{UB}}(x)$. The closed-form expression must satisfy the following two conditions: i) $g(x)$ is piecewise linear and continuous and ii) $g(x_i) = y_i$ for $n + 1$ points (x_i, y_i). The points are already computed following either algorithm in Section 3.2. The function $g(x)$ is defined to be a weighted sum of $n + 1$ piecewise linear basis functions $\beta_i(x)$ with the *property* that:

$$\beta_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases}$$

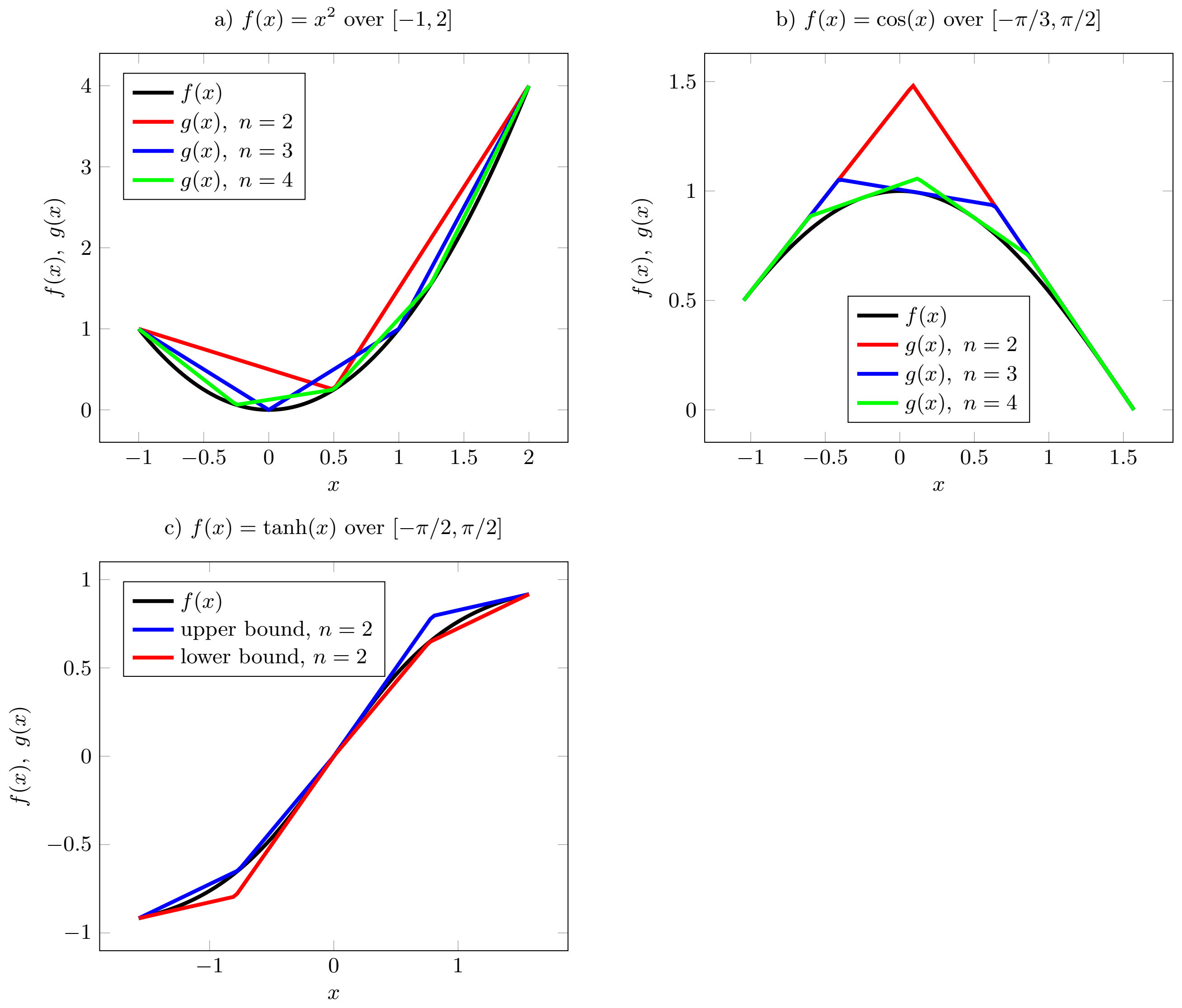


Figure 2: Examples of piecewise linear over-approximations.

Then, we can write the function $g(x)$ as:

$$g(x) = \sum_{i=0}^{n} \beta_i(x) y_i$$

Notice that $g(x)$ is piecewise linear because it is a weighted sum of piecwise linear functions $\beta_i(x)$. In addition,

$$g(x_i) = y_0\beta_0(x_i) + \cdots + y_i\beta_i(x_i) + \cdots + y_N\beta_N(x_i) = 0 + \cdots + y_i + \cdots + 0 = y_i$$

The problem is therefore reduced to finding the basis functions $\beta_i(x)$. Our idea for constructing the basis functions is inspired by first order basis functions in finite element analysis. The basis functions $\beta_i(x)$ can be defined using the following piecewise relation:

$$\beta_i(x) = \begin{cases} 0 & x < x_{i-1} \\ L_l^i & x_{i-1} < x < x_i \\ L_r^i & x_i < x < x_{i+1} \\ 0 & x > x_{i+1} \end{cases} \tag{7}$$

where

$$L_l^i = \frac{x - x_{i-1}}{x_i - x_{i-1}}, \qquad L_r^i = \frac{x - x_{i+1}}{x_i - x_{i+1}}$$

Equation (7) can be rewritten in closed form as:

$$\beta_i(x) = \min\Big(\max\big(L_l^i, 0\big), \max\big(L_r^i, 0\big)\Big) = \max\Big(0, \min\big(L_l^i, L_r^i\big)\Big)$$

For the boundary cases of $i = 0$ and $i = n$, we only have L_r^0 and L_l^n, respectively, and therefore,

$$\beta_0 = \max\big(L_r^0, 0\big), \quad \text{and} \quad \beta_n = \max\big(L_l^0, 0\big)$$

As an example, Eqs. (8) and (9) are the closed form expressions for the upper bound (blue curve) and lower bound (red curve) linear piecewise over-approximations shown in Fig. 2c.

$$\begin{aligned} g_{\text{ub}}(x) =& (-0.917 \max(0, -1.244(x - -0.767)) + \\ & -0.645 \max(0, \min(1.244(x - -1.571), -1.304(x - 0.0))) + \\ & 0.0 \max(0, \min(1.304(x - -0.767), -1.260(x - 0.794))) + \\ & 0.794 \max(0, \min(1.260(x - 0.0), -1.287(x - 1.571))) + \\ & 0.917 \max(0, 1.287(x - 0.794))) \end{aligned} \tag{8}$$

$$\begin{aligned} g_{\text{lb}}(x) =& (-0.917 \max(0, -1.287(x - -0.794)) + \\ & -0.794 \max(0, \min(1.287(x - -1.571), -1.260(x - 0.0))) + \\ & 0.0 \max(0, \min(1.260(x - -0.794), -1.304(x - 0.767))) + \\ & 0.645 \max(0, \min(1.304(x - 0.0), -1.244(x - 1.571))) + \\ & 0.917 \max(0, 1.244(x - 0.767))) \end{aligned} \tag{9}$$

3.4 Single Pendulum Example

To illustrate the procedure for obtaining a relational overapproximation, we use the inverted single pendulum dynamical system. We derive the discrete-time dynamical system from the governing differential equations. A *simple* single pendulum is comprised of a point mass m and a massless rod of length ℓ. The pendulum is actuated by a motor that exerts a torque of u. The goal is to keep the pendulum in the up-right position. Neglecting air resistance, the governing differential equation for this dynamical system is:

$$\ddot{\theta} = \frac{g}{\ell} \sin\theta + \frac{1}{m\ell^2} u \tag{10}$$

where θ denotes the angle made by the pendulum arm and vertical direction and g is the gravitational acceleration. We use a first-order Euler scheme with a temporal step size of $\Delta\tau$ to produce the discrete-time equations:

$$\begin{aligned}
x_{t+1,1} =& x_{t,1} + \Delta\tau \ x_{t,2} \\
x_{t+1,2} =& x_{t,2} + \Delta\tau \left(\frac{g}{\ell} \sin x_{t,1} + \frac{1}{m\ell^2} u_t \right)
\end{aligned}$$

where $\mathbf{x}_t = [x_{t,1}, x_{t,2}]^T = [\theta_t, \dot{\theta}_t]^T$ and u_t represents the discrete state variable and control input, respectively, at time step t. In order to obtain the relational overapproximation of this system, we need to specify the domain of input parameters (states and control parameters). For the state parameters, the domain is typically specified by the user. The control signal u_t would normally be specified by a neural network, and the range of u_t for the dynamics approximation could be found using any method for estimating the output range of a neural network, such as interval arithmetic, linear relaxation, or Lipschitz constant estimation of the network (Tjeng et al., 2019; Xiang et al., 2018a). However in this simple example, we simply specify a set that u_t lies within. Assuming the initial set of states $(x_{0,1}, x_{0,2})$ lie within $[-1, 1]^2$, and initial control policy input lies within $[-2, 2]$, the relational overapproximation of this system that specifies states at the next time step ($t = 1$) is given by:

$$\begin{aligned}
& x_{1,1} = x_{0,1} + \Delta\tau \ x_{0,1} \\
& -1 \leq x_{0,1} \leq 1 \\
& -1 \leq x_{0,2} \leq 1 \\
& x_{1,2} = x_{0,2} + \Delta\tau \left(\frac{g}{\ell} v_1 + \frac{1}{m\ell^2} u_0 \right) \\
& v_1 \leq g_{\sin_{UB}}(x_{0,1}) \\
& v_1 \geq g_{\sin_{LB}}(x_{0,1}) \\
& -2 \leq u_0 \leq 2
\end{aligned}$$

The two functions $g_{\sin_{LB}}$ and $g_{\sin_{UB}}$ are the lower and upper bounds of $\sin(x_{0,1})$ over domain $[-1, 1]$, which can be computed using the GetUpperBound and GetLowerBound functions. For example, if we use piecewise linear functions with two linear segments, we obtain

$$\begin{aligned}
g_{\sin_{UB}}(x) =& -1.00\max(0, -1.449(x + 0.881)) \\
& - 0.761\max(0, \min(1.449(x + 1.570), -1.135x)) \\
& + 0.01\max(0, \min(1.135(x + 0.881), x - 1.0)) \\
& + 1.01\max(0, \min(x, -1.752(x - 1.571)) \\
& + 1.01\max(0, 1.752(x - 1.0)), \\
g_{\sin_{LB}}(x) =& -1.01\max(0, -1.752(x + 1.0)) \\
& - 1.01\max(0, \min(1.752(x + 1.571), -x) \\
& - 0.01\max(0, \min(x + 1.0, -1.135(x - 0.881))) \\
& + 0.761\max(0, \min(1.135x, -1.449(x - 1.571)) \\
& + 0.99\max(0, 1.449(x - 0.881))
\end{aligned}$$

3.5 Solving Reachability Problems

OVERT solves reachability problems, meaning it reasons about the set of states reachable over time and whether this set of states intersects with an unsafe set or reaches a goal set. One way to frame the reachability problem is to explicitly compute the reachable set of the closed-loop system. The *reachable set* at a given timestep, $\mathcal{R}_t$, is comprised of all possible states that the system could visit at time t (Clarke et al., 2018). If the intersection of the reachable set and the unsafe set is empty, $\mathcal{R}_t \cap \mathcal{S}_{\text{unsafe}} = \varnothing$, the safety property holds at time t. If the reachable set is a subset of the goal set, $\mathcal{R}_t \subseteq \mathcal{S}_{\text{goal}}$, the system is guaranteed to reach the goal at time t.

Another way to frame the reachability problem is by solving what we call *feasibility problems*. Feasibility problems can directly encode the unsafe set (complement of the safe set) and return SAT indicating that the unsafe set is reachable or UNSAT indicating that the unsafe set is not reachable, without explicitly computing the reachable set.

OVERT can solve reachability problems using either an explicit reachable set computation framing or a feasibility framing, and these two approaches are often complementary. Since reachable sets are typically overapproximated by simple geometric shapes such as hyperrectangles or polytopes, reachable set computation introduces some looseness into the approximation. Conversely, solving feasibility problems incurs less looseness and less wrapping effect because the reachable set is instead represented implicitly. Consequently, solving feasibility problems results in fewer spurious counter examples and allows more properties to be proven. However, computing reachable sets provides intuition about the the evolution of the system and allow the reachability problem to remain tractable over long time horizons.

We can compute reachable sets in two different ways. The first approach is what we call the *concrete* approach. Beginning with all state variables $\mathbf{x}$ constrained to lie in the input set $\mathcal{I}$ at time $t = 0$, this approach computes a concrete reachable set $\mathcal{R}_1$ of the system at timestep $t = 1$. For the next timestep, a new problem instance is created with the state variables $\mathbf{x}$ now constrained to lie within $\mathcal{R}_1$, and the concrete reachable set $\mathcal{R}_2$ is calculated, which is an overapproximation of the reachable set at timestep $t = 2$. Here, we define the concrete reachable set as an explicit representation of reachable set (e.g., $1.56 \leq x \leq 4.67$). The term *concretizing* refers to computing such a concrete reachable set. This approach effectively resets the reachable set computation problem, yielding a 1-step problem where only the initial set changes at each step. Concretizing after just one step incurs compounding looseness after several iterations, rendering the reachable set more conservative. This looseness allows for more spurious counter examples. However, computing 1-step reachable sets is very fast.

The second approach is what we call the *symbolic* approach. In this approach, an implicit, symbolic representation of the reachable set is unrolled for several timesteps, and only concretized at the final timestep in the sequence. The symbolic approach produces sets that are significantly tighter, but the computational complexity of this approach can grow quickly. Each additional timestep encoded in the symbolic representation adds an approximately equal number of additional constraints. If one were to use only this approach, the length of time over which one could check properties would be severely constrained.

In order to keep the problem computationally tractable, and yet obtain a reasonably accurate reachable set, we employ a hybrid approach, where concretization is performed only when the reachable set begins to get too large. We refer to this approach as the *hybrid-symbolic* approach, and it is one aspect that sets OVERT apart from related work by Julian and Kochenderfer (2021) and Tran et al. (2020). The hybrid-symbolic approach is implemented by the function ComputeReachSets($\mathbf{f}, \mathcal{I}, \mathrm{cI}$) (Algorithm 3), where $\mathbf{f}$ is the closed-loop system, $\mathcal{I}$ is an input set, and cI stands for *concretization intervals* and is an array such as $[5, 10, 5]$ indicating that we concretize at timesteps 5, 15, 20. For example, we might compute nine one-step concrete sets, and then calculate a tight symbolic reachable set at timestep 10, beginning from a concrete set at timestep 1. This alternation is then repeated, calculating one-step concrete sets and symbolic sets when needed. The decision of when to compute symbolic reachable sets affects how long it takes to solve the reachability problem. For the experiments that were run here, we ran initial explorations for each problem and then manually chose a concrete vs. symbolic unroll schedule. We leave the task of automatically picking an unroll schedule for a given problem for future work.

Algorithm 3 Hybrid-Symbolic Reachable Set Computation

```
 1: function ComputeReachSets($\mathbf{f}$, $\mathcal{I}$, cI)
 2:     sym_input_set ← $\mathcal{I}$
 3:     $\mathcal{R} \leftarrow \mathcal{I}$
 4:     reach_sets ← $\emptyset$
 5:     approximations ← $\emptyset$
 6:     for $n$ in cI
 7:         for $i \leftarrow 1 : n-1$
 8:             $\hat{\mathbf{f}}$ ← Overapproximate($\mathbf{f}$, $\mathcal{R}$)
 9:             Add $\hat{\mathbf{f}}$ to approximations
10:             $\mathcal{R}$ ← OneStepReach($\hat{\mathbf{f}}$,$\mathcal{R}$)
11:             Add $\mathcal{R}$ to reach_sets
12:         $\hat{\mathbf{f}}$ ← Overapproximate($\mathbf{f}$, $\mathcal{R}$)
13:         Add $\hat{\mathbf{f}}$ to approximations
14:         $\mathcal{R}$ ← SymbolicReach(approximations, sym_input_set, $n$)
15:         Add $\mathcal{R}$ to reach_sets
16:         sym_input_set ← $\mathcal{R}$
17:     return reach_sets
```

Algorithm 4 Feasibility of Invariant Properties

```
function $\textsc{Feasibility}_G(\mathbf{f}, \mathcal{I}, n, P)$
    input_set $\leftarrow \mathcal{I}$
    $\mathcal{R} \leftarrow \mathcal{I}$
    approximations $\leftarrow \emptyset$
    holds $\leftarrow$ true
    $i \leftarrow 1$
    while $i \leq n$ and holds
        $\hat{\mathbf{f}} \leftarrow \textsc{Overapproximate}(\mathbf{f}, \mathcal{R})$
        Add $\hat{\mathbf{f}}$ to approximations
        holds $\leftarrow$ holds and $\textsc{MultiStepFeas}$(approximations, input_set, $i$, $P$)
        $\mathcal{R} \leftarrow \textsc{OneStepReach}(\hat{\mathbf{f}}, \mathcal{R})$
        $i \leftarrow i + 1$
    return holds
```

Algorithm 5 Hybrid-Symbolic Feasibility

```
function $\textsc{HSFeasibility}_G(\mathbf{f}, \mathcal{I}, n, P)$
    input_set $\leftarrow \mathcal{I}$
    $\mathcal{R} \leftarrow \mathcal{I}$
    approximations $\leftarrow \emptyset$
    holds $\leftarrow$ true
    $i \leftarrow 1$
    while $i \leq n$ and holds
        $\hat{\mathbf{f}} \leftarrow \textsc{Overapproximate}(\mathbf{f}, \mathcal{R})$
        Add $\hat{\mathbf{f}}$ to approximations
        holds $\leftarrow$ holds and $\textsc{MultiStepFeas}$(approximations, input_set, $i$, $P$)
        if time to reset
            $\mathcal{R} \leftarrow \textsc{SymbolicReach}$(approximations, input_set, $i$)
            input_set $\leftarrow \mathcal{R}$
            $n \leftarrow n - i$
            $i \leftarrow 1$
        else
            $\mathcal{R} \leftarrow \textsc{OneStepReach}(\hat{\mathbf{f}}, \mathcal{R})$
            $i \leftarrow i + 1$
    return holds
```

The function $\textsc{Overapproximate}(\mathbf{f}, \mathcal{R})$ calculates an overapproximation of the closed-loop system $\mathbf{f}$ over the domain specified by $\mathcal{R}$. The function $\textsc{OneStepReach}(\hat{\mathbf{f}}, \mathcal{R})$ computes a one step reachable set for the system from $\mathcal{R}$, and the function $\textsc{SymbolicReach}$(approximations, sym_input_set, n) computes a reachable set for the system n steps into the future using symbolic relations between timesteps. The function $\textsc{OneStepReach}$ is defined $\textsc{OneStepReach}(\hat{\mathbf{f}}, \mathcal{R}) = \textsc{SymbolicReach}(\hat{\mathbf{f}}, \mathcal{R}, 1)$. More detailed on how the functions are implemented will be discussed in Section 3.6.

OVERT solves feasibility problems by iteratively unrolling the closed-loop system and checking if the complement of the desired property is UNSAT at the final time. However, OVERT requires a domain for the state variables in order to overapproximate the dynamics at each subsequent timestep. OVERT provides this domain for feasibility problems by calculating 1-step reachable sets. This is illustrated in Algorithm 4. The function MultiStepFeas(approximations, $\mathcal{I}$, n, P) performs a symbolic feasibility query to determine if property P holds n steps into the future beginning from set $\mathcal{I}$. Feasibility for goal-reaching properties, or F properties, would be written very similarly to Algorithm 4, except it would be checking for the first instance of the property holding, instead of checking that it holds at every step.

In our experiments, the feasibility approach shown in Algorithm 4 ran fast enough that a hybrid-symbolic-feasibility approach was not necessary. However, one such algorithm mixing the use of feasibility problems and symbolic reachable set computation for a property of the form $G\ P$ (globally P) is described in Algorithm 5. Such an approach would keep the problem tractable for long time histories.

3.6 MIP Formulation

Once the system dynamics have been abstracted, the closed-loop system consists of a conjunction of linear and piecewise linear constraints. The piecewise linear constraints used are min, max and relu. The key insight of several neural network verification tools for ReLU activations (Akintunde et al., 2018; Tjeng et al., 2019) is that piecewise linear activation functions can be encoded into an optimization problem using binary variables, resulting in a mixed integer program (MIP).

We compute reachable sets by constructing an optimization problem where the abstracted closed-loop system dynamics are encoded as constraints. An initial set for the state variables is also encoded as a constraint. We then find the minimum and maximum value of each state variable at subsequent timesteps, subject to these constraints. The symbolic reachability procedure, SymbolicReach, solves the following two optimization problems for the kth state component in order to compute the reachable set n steps in the future:

$$\begin{aligned} \text{minimize} \quad & \mathbf{x}_{n,k} \\ \text{subject to} \quad & \mathbf{x}_i \in \mathcal{S}, \\ & \mathbf{x}_{t+1} \bowtie \hat{\mathbf{f}}(\mathbf{x}_t, \mathbf{c}(\mathbf{x}_t)),\ t = i, \ldots, n-1 \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}_{n,k} \\ \text{subject to} \quad & \mathbf{x}_i \in \mathcal{S}, \\ & \mathbf{x}_{t+1} \bowtie \hat{\mathbf{f}}(\mathbf{x}_t, \mathbf{c}(\mathbf{x}_t)),\ t = i, \ldots, n-1 \end{aligned}$$

Here, timestep i is the initial timestep and set $\mathcal{S}$ is the starting set. The symbol $\hat{\mathbf{f}}$ represents the overapproximated system dynamics, $\mathbf{c}$ represents the control policy, and $\bowtie$

represents that there is a relation between $\mathbf{x}_{t+1}$ and $\hat{\mathbf{f}}(\mathbf{x}_t, \mathbf{c}(\mathbf{x}_t))$. Computing the minimum and maximum values specify the reachable set as a hyper-rectangle. This process is repeated for all components of the state. For example, to compute the reachable set 10 steps in the future beginning from timestep 0, we would set $i = 0$, $\mathcal{S} = \mathcal{I}$, and $n = 10$. Once the reachable sets have been computed, they are then intersected with the unsafe sets or goal sets to determine if the desired property holds.

In order to ensure soundness in the finite-precision regime, the solution to the dual optimization problem, rather than the primal problem, is used to construct the overapproximate reachable set. For minimization problems, the dual solution is guaranteed to be a lower bound, and for maximization problems, the dual solution is guaranteed to be an upper bound. Thus using the dual bound increases the volume of the hyperrectangle and still represents an overapproximation of the true reachable set. Most modern optimization software provides access to the dual as well as the primal solution. In practice, the duality gap between the primal and dual solutions is small, and using the dual does not add significant looseness.

In order to solve the feasiblity framing of reachability, a similar problem is constructed where the closed-loop system dynamics are encoded as constraints and an initial set is specified for the state variables. In this case, however, no objective is specified and the negation of the desired property, $\neg P$, is encoded as an additional constraint. The symbolic feasibility procedure, MultiStepFeas, solves the following feasibility problem:

$$\begin{aligned} \text{minimize} \quad & 0 \\ \text{subject to} \quad & \mathbf{x}_i \in \mathcal{S}, \\ & \mathbf{x}_{t+1} \bowtie \hat{\mathbf{f}}(\mathbf{x}_t, \mathbf{c}(\mathbf{x}_t)),\ t = i, \ldots, n-1, \\ & \neg P(\mathbf{x}_n) \end{aligned}$$

If no feasible solution can be found, the negation of the property is *unsatisfiable* and the property is proven, meaning that the system, for example, never visits (at a given time n) the unsafe set. Unlike explicit reachable set computation, this approach does not need to solve optimality problems and therefore is typically less computationally expensive.

There are many efficient tools for solving MIPs, which allows for tractable verification of reachability properties. An important factor contributing to the speed of verification is the encoding used to represent the ReLUs and other piecewise linear activations in the MIP. In this work, a ReLU encoding inspired by the MIPVerify algorithm (Tjeng et al., 2019) is used. This encoding is a tighter alternative to the *big-M* formulations that are commonly used for encoding min and max operations (Akintunde et al. (2018)). The resultant MIP is then solved using Gurobi (Gurobi Optimization, 2020).

The ReLU encoding presented in the MIPVerify algorithm requires a bound for the values of all neurons in the neural network. Such bounds can be computed using interval arithmetic or a linear programming relaxation (Tjeng et al., 2019). In this work, we use a Lipschitz-constant-based bounding algorithm designed by Xiang et al. (2018a). An encoding inspired by MIPVerify is also used for the piecewise-linear min and max functions in the abstraction of the dynamics. This encoding is particularly suitable for OVERT, as bounds on each variable are already calculated during the abstraction of the dynamics, and these bounds can be used for the MIP encoding. OVERT is currently implemented in

Julia. We have open-sourced two Julia packages containing implementations of the sound overapproximation[2] and verification algorithms.[3]

4. Experiments

Below we demonstrate application of OVERT to a number of benchmark examples. All simulations are conducted with a machine with two 14-core Intel(R) Xeon(R) CPU E5-2690 v4 CPUs @ 2.60GHz (28 cores total), 128 GB RAM, and Ubuntu 18.04.

4.1 Benchmark Examples

For the purpose of demonstration, we have chosen four classical control systems as described below. These four examples were part of the ARCH-COMP AINNC 2020 competition (Johnson et al., 2020). We have also trained additional control policies to facilitate comparison.

1. **Single Pendulum (S)**: The governing equation for an inverted single pendulum is Eq. (10). We will use the discrete-time version shown in Eq. (14) with state variables $\mathbf{x}_t = [x_{t,1}, x_{t,2}]^T = \left[\theta_t, \dot{\theta}_t\right]^T$ and input control torque u_t which is repeated below for the sake of completeness:

$$x_{t+1,1} = x_{t,1} + \Delta\tau \ x_{t,2} \tag{14a}$$
$$x_{t+1,2} = x_{t,2} + \Delta\tau \left(\frac{g}{\ell}\sin x_{t,1} + \frac{1}{m\ell^2}u_t\right) \tag{14b}$$

The parameter values are $m = 0.5$, $\ell = 0.5$, $g = 1.0$ and $\Delta\tau = 0.1$. The state space $[x_1, x_2]$ is two-dimensional, and the control input u is one-dimensional.

Control policies are trained to stabilize the pendulum upside-down using behavior cloning, a supervised learning approach for training control policies. Here, a neural

2. `https://www.github.com/sisl/OVERT.jl`
3. `https://www.github.com/sisl/OVERTVerify.jl`

Table 2: Benchmark Problems.

Problem	Dynamics	Neural Network Controller Size	Number of Timesteps
S1	single pendulum	$2 \times 25 \times 25 \times 1$	25
S2	single pendulum	$2 \times 50 \times 50 \times 1$	25
T1	tora	$4 \times 25 \times 25 \times 25 \times 1$	15
T2	tora	$4 \times 50 \times 50 \times 50 \times 1$	15
T3	tora	$4 \times 100 \times 100 \times 100 \times 1$	15
C1	car	$4 \times 100 \times 1$	10
C2	car	$4 \times 200 \times 1$	10
C3	car	$4 \times 300 \times 1$	10
ACC	adaptive cruise control	$6 \times 3 \times 20 \times 20 \times 20 \times 1$	55

network is trained to replicate expert demonstrations. We initially generate a set of expert control inputs for trajectories originating from different initial states of the system. Expert control inputs are defined as those leading the system to reach to its goal state, 0 radians, in finite time. The expert control inputs are generated using optimal control techniques. Specifically, we have used an implementation of the LQR (Linear Quadratic Regulator) algorithm to generate expert control inputs. Two control policies with different architectures were trained using this data; details are given in Table 2.
Property: The angle of the pendulum must stay above ≈ -15 degrees, or -0.2167 radians, for 25 discrete time steps:

$$G_{1:25}\ x_1 \geq -0.2167$$

beginning from the initial set

$$x_1, x_2 \in [1, 1.2] \times [0, 0.2]$$

2. **TORA (T)**: This example is a model for rotational actuators. TORA stands for Transalation Oscillations of a Rotational Actuator. The system consists of a cart that can roll along a frictionless surface and is attached to the wall by a spring. Inside the cart, there is a mass on a rod that can rotate to actuate the cart. The state may be described by the x-displacement of the cart and the angular displacement of the rotational actuator. However, Jankovic et al. (1996) derive the following simpler equations of motion using more complex state variables:

$$\begin{aligned}\dot{x}_1 =& x_2\\ \dot{x}_2 =& -x_1 + \epsilon \sin x_3\\ \dot{x}_3 =& x_4\\ \dot{x}_4 =& u\end{aligned}$$

The model has a four-dimensional state space $[x_1, x_2, x_3, x_4]$ and one-dimensional control input u. We will use the following discrete form of the model:

$$\begin{aligned}x_{t+1,1} =& x_{t,1} + \Delta\tau x_{t,2}\\ x_{t+1,2} =& x_{t,2} + \Delta\tau\left(\epsilon \sin x_{t,3} - x_{t,1}\right)\\ x_{t+1,3} =& x_{t,3} + \Delta\tau x_{t,4}\\ x_{t+1,4} =& x_{t,4} + \Delta\tau u_t\end{aligned}$$

The parameter values are $\epsilon = 0.1$ and $\Delta\tau = 0.1$. We tested this benchmark with three different neural network control policies, as shown in Table 2. The largest neural network is that of Dutta et al. (2019). This network is trained to stabilize the cart at $x = 0$. We prepared the two smaller networks using supervised learning on data generated from the larger network. This way, we ensure that the three networks are trained to do a similar task, and therefore the difference in computational time is due to the different neural network architecture.
Property: The first state variable, x_1 must stay close to the origin:

$$G_{1:15}\ x_1 \in [-2, 2]$$

beginning from the initial set

$$x_1, x_2, x_3, x_4 \in [0.6, 0.7] \times [-0.7, -0.6] \times [-0.4, -0.3] \times [0.5, 0.6]$$

3. **Car (C)**: This example uses a kinematic bicycle model to approximate car dynamics (Rajamani, 2011; Kong et al., 2015). The state of the system may be described by the position of the car (x_1, x_2) as well as the yaw angle (x_3) and speed (x_4). The ordinary differential equations governing motion are:

$$\begin{aligned} \dot{x}_1 &= x_4 \cos(x_3) \\ \dot{x}_2 &= x_4 \sin(x_3) \\ \dot{x}_3 &= u_2 \\ \dot{x}_4 &= u_1 \end{aligned}$$

   The state space $[x_1, x_2, x_3, x_4]$ is four-dimensional, and control inputs $[u_1, u_2]$ are two-dimensional. We will use the following discrete form of the model:

$$\begin{aligned} x_{t+1,1} &= x_{t,1} + \Delta\tau x_{t,4} \cos(x_{t,3}) \\ x_{t+1,2} &= x_{t,2} + \Delta\tau x_{t,4} \sin(x_{t,3}) \\ x_{t+1,3} &= x_{t,3} + \Delta\tau u_{t,2} \\ x_{t+1,4} &= x_{t,4} + \Delta\tau u_{t,1} \end{aligned}$$

   We set $\Delta\tau = 0.2$.

   Similar to the Tora benchmark, we tested this example with three different neural network control policies. The largest neural network is that of Dutta et al. (2019) and we prepared the two smaller networks using supervised learning.

   Property: The position of the car will reach a goal set near the origin within 10 timesteps:

$$F_{1:10}\ (x_1, x_2) \in [-0.6, 0.6] \times [-0.2, 0.2]$$

   beginning from the initial set

$$x_1, x_2, x_3, x_4 \in [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51]$$

4. **Adaptive Cruise Control (ACC)**: This system models an adaptive cruise control policy for a car on a highway. The system has a target velocity, but there is a lead vehicle ahead of the ego vehicle, and the ego must maintain a safe distance from the lead vehicle. The neural network control policy adjusts the longitudinal acceleration of the ego vehicle. The position, velocity and acceleration of the lead vehicle are x_1, x_2, and x_3. The position, velocity, and acceleration of the ego vehicle are x_4, x_5, and x_6. The continuous-time dynamics of the system (Johnson et al., 2020) are:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -2x_3 + 2a - 2\mu x_2^2 \\ \dot{x}_4 &= x_5 \\ \dot{x}_5 &= x_6 \\ \dot{x}_6 &= -2x_6 + 2u - 2\mu x_5^2 \end{aligned}$$

where $[x_1, x_2, x_3, x_4, x_5, x_6]$ is the 6-dimensional state space and u is the single control input (acceleration of the ego car). Here, a and μ are the lead car acceleration and friction factor, respectively. We use the following discrete form of the system:

$$\begin{aligned}
x_{t+1,1} =& x_{t,1} + \Delta\tau x_{t,2} \\
x_{t+1,2} =& x_{t,2} + \Delta\tau x_{t,3} \\
x_{t+1,3} =& x_{t,3} + \Delta\tau \left(-2x_{t,3} + 2a - 2\mu x_{t,2}^2\right) \\
x_{t+1,4} =& x_{t,4} + \Delta\tau x_{t,5} \\
x_{t+1,5} =& x_{t,5} + \Delta\tau x_{t,6} \\
x_{t+1,6} =& x_{t,6} + \Delta\tau \left(-2x_{t,6} + 2u_t - 2\mu x_{t,5}^2\right)
\end{aligned}$$

Similar to Johnson et al. (2020), we choose $a = -2$, which models the lead vehicle decelerating, and a friction parameter $\mu = 10^{-4}$. The discretization timestep is set to $\Delta\tau = 0.1$.
This benchmark was tested with the neural network control policy from Johnson et al. (2020). **Property:** A safe distance between the ego vehicle and the lead vehicle must be maintained:

$$G_{1:55}\ x_{lead} - x_{ego} \geq D_{safe} \tag{21}$$

where $D_{safe} = T_{gap} * v_{ego} + D_{min}$, $x_{lead} = x_1$, $x_{ego} = x_4$, $v_{ego} = x_5$, $D_{min} = 10$, and $T_{gap} = 1.4$. The initial set is

$$x_1, x_2, x_3, x_4, x_5, x_6 \in [90, 91] \times [10, 11] \times [30, 30.2] \times [30, 30.2] \times [0, 0.01] \times [0, 0.01]$$

Each benchmark example contains nonlinearities that are overapproximated. One may wonder what the right choice for n, the number of linear segments within each region of constant convexity, is for each benchmark. On one hand, as n increases, the overapproximation gets tighter. On the other hand, this overapproximation requires more binary variables in the mixed integer representation. Interestingly, we experimentally observe that increasing the number of binary variables by increasing n does not immediately make the MIP problem harder to solve. In other words, in our experiments, the MIP program was significantly more sensitive to number of binary variables originating from the neural network, and less so to those originating from the abstraction of the dynamics. In all of our experiments, we automatically chose n large enough such that the maximum relative error between $f(x)$ and $g(x)$ is less than 2% of the range of $f(x)$ over the specified domain.

4.2 Comparison with other tools

Here, we will compare the performance of OVERT with two other tools that can be used for closed-loop verification of discrete-time problems. It is imperative to note that discrete-time problems are inherently different than continuous-time problems. Even if a discrete-time problem is obtained by discretizing a continuous system, these two systems are fundamentally different, and not directly comparable. Therefore, it is not appropriate to compare OVERT to tools designed for continuous-time problems such as Sherlock (Dutta et al., 2019) or Verisig (Ivanov et al., 2019).

As mentioned earlier, OVERT can solve a reachability problem by framing it as either a direct reachable set computation or as a feasibility problem. To show its performance on feasibility problems, we compare it with dReal (Gao et al., 2013), which is an automated reasoning tool designed for solving nonlinear decision problems over the real numbers. As dReal is not designed to perform model checking for transition systems, a wrapper was written to translate the bounded time model checking problem that OVERT solves into the SMT2 format that is readable by dReal. For direct reachable set computation, we compare OVERT with the Neural Network Verification (NNV) tool (Tran et al., 2020), which is designed for computing reachable sets of closed-loop systems with a neural network control policy. We emphasize here that at the time of writing, NNV was designed for use only with continuous-time systems, and we had to implement support for discrete-time systems. The approximate star-based method was used within NNV.

The results of the comparisons are shown in Table 3 on the S1 and T1 problems. For each example, we have assumed a 12-hour timeout. For the reachable set computation framing of the problem, NNV finished the S1 problem in 483 seconds but was unable to prove the desired property as the approximate reachable set computed was too loose. In comparison, OVERT took just 99 seconds and was able to prove the property. For the S1 problem, we perform a symbolic query with OVERT at timesteps 10, 20, and 25, and propagate one-step concrete sets in between those timesteps. On the T1 problem, NNV did not finish computation within 12 hours, but OVERT finished the task in less than 4 minutes and was able to prove the property. For the T1 problem, we perform a symbolic query with OVERT at timesteps 5, 10 and 15. It was noted that by timestep 8 in problem T1, NNV was tracking the reachable set of nearly 2 million polytopes, due to disjunctions introduced by the ReLU activations; this explosion in polytopes likely caused the timeout.

Next, we compare OVERT to dReal for solving the feasibility framing of the problem. dReal times out on the S1 problem, while OVERT takes less than two minutes to solve the feasibility framing of the problem. For the T1 problem, dReal is able to show that the property holds, but it takes approximately 3.5 hours, while OVERT takes less than two minutes to solve the feasibility framing. For both the S1 and T1 feasibility framing versions of the problem, Algorithm 4 was used, where no symbolic queries are performed to "reset" the time horizon. If a symbolic query had been used, making the resultant feasibility problem e.g. only 10 steps long, the feasibility results would likely be even faster. This remains to be explored more fully in future work. The results clearly indicate that general purpose nonlinear reasoning tools such as dReal do not scale to handle ReLU neural networks, and specialized tools such as OVERT are necessary for the verification of systems containing ReLU neural networks.

OVERT outperforms both dReal and NNV in terms of computation time, and the final reachable set obtained by OVERT are also significantly tighter than those found by NNV. Figure 3 shows the reachable sets obtained by OVERT and NNV for the S1 problem. As shown by Xiang et al. (2018b), splitting the input set can produce tighter reachable sets. This phenomenon benefits the performance of both NNV and OVERT. To allow for fair comparison, we do not split the input set for either tool. As can be seen by comparing the top two plots in Fig. 3, for the first 20 timesteps, the dimensions of the sets produced by NNV are up to 3 to 4 orders of magnitude larger than those produced by OVERT. For timesteps 21 to 25, however, shown in the bottom two plots, the sets appear to be

Table 3: Comparison between OVERT, NNV, and dReal

Problem	Timesteps	DReal (s)	NNV (s)	$\text{OVERT}_{\text{reach}}$ (s)	$\text{OVERT}_{\text{feas}}$ (s)
S1	25	Time Out, N/A	483, fails	99, HOLDS	**77**, HOLDS
T1	15	12655, HOLDS	Time Out, N/A	232, HOLDS	**79**, HOLDS

degenerate as they cease to contain the true reachable set, which is still roughly centered near (0,0), and reduce to lines in the case of timesteps 24 and 25. As the sets for timesteps 21 to 25 grow very large with dimensions on the order of 10^6, it is likely that numerical errors are creating the observed degeneracy.

4.3 Benchmarking Performance

In this section, we report the results of reachability problems analyzed with OVERT. As aforementioned, each reachability problem can be posed either as a reachable set computation and then set intersection problem, or by directly encoding the property and solving a feasibility problem. We explore each of OVERT's two capabilities in the following two subsections.

4.3.1 Reachable Set Computation Problems

We start with the single pendulum benchmark. Fig. 4 illustrates the reachability results for the single pendulum benchmark with two different neural network control policies. For both S1 and S2, symbolic reachable sets were computed at timesteps 10, 20, and 25. It can be observed for both S1 and S2 that the hybrid-symbolic approach is capable of proving the safety property, but the naive 1-step concrete sets are too loose, and cannot prove the property. The convex hull of Monte Carlo simulations represent an underapproximation of the true reachable set, and one can observe that the sets generated by OVERT's hybrid approach hug the true system trajectories tightly. This suggests that few spurious counter examples will be generated.

While one might observe that performing more and longer symbolic queries would produce even tighter reachable sets, this also adds to the complexity of the MIP to be solved. For example, in the case of S2, the naive one-step reachable set computation took less than 4 minutes, while the hybrid symbolic approach with two 10-step queries, one 5-step query, and 22 one-step queries required just under 12 minutes to solve.

When the update function of a dynamical system includes more nonlinear components, the complexity of the problem increases. Each nonlinear component requires piecewise linear overapproximation, which in turn translates into an MIP formulation with more binary variables. While there is some variability from one problem to another, additional binary variables generally add complexity to the problem. Furthermore, a dynamical system with more nonlinearity may require a more complex control policy; e.g. a deeper neural network, which further increases the complexity of the problem. In order to handle this additional complexity, in the Tora and Car examples, we reduced the number of timesteps and increased the frequency at which symbolic queries were performed to reset the time horizon.

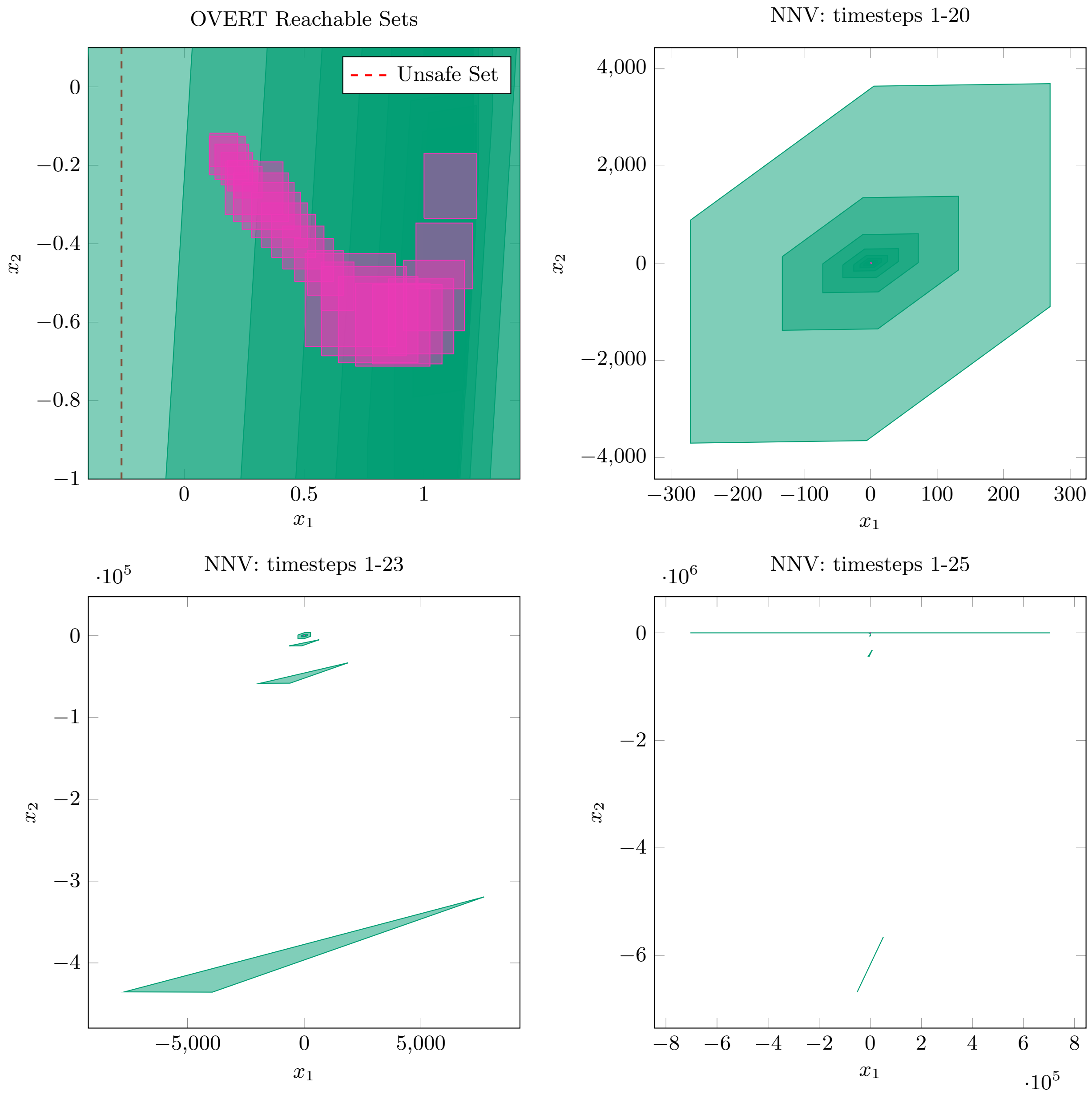


Figure 3: Comparison of reachable set for the S1 problem using NNV (green polygons) and OVERT (pink polygons). The dashed red line in the top left plot shows the boundary of the unsafe set.

Figure 5 shows the reachability results for the Tora T3 example and Car C3 example. The difference between the concrete and hybrid-symbolic approaches is not as significant as in the single pendulum example, in part because we had to concretize the symbolic queries earlier to avoid having queries that were too large. Symbolic sets were calculated for all of the TORA problems (T1, T2, T3) at timesteps 5, 10, and 15. For the car problems (C1, C2, C3), symbolic sets were calculated at timesteps 5 and 10. The top left plot shows the 2-

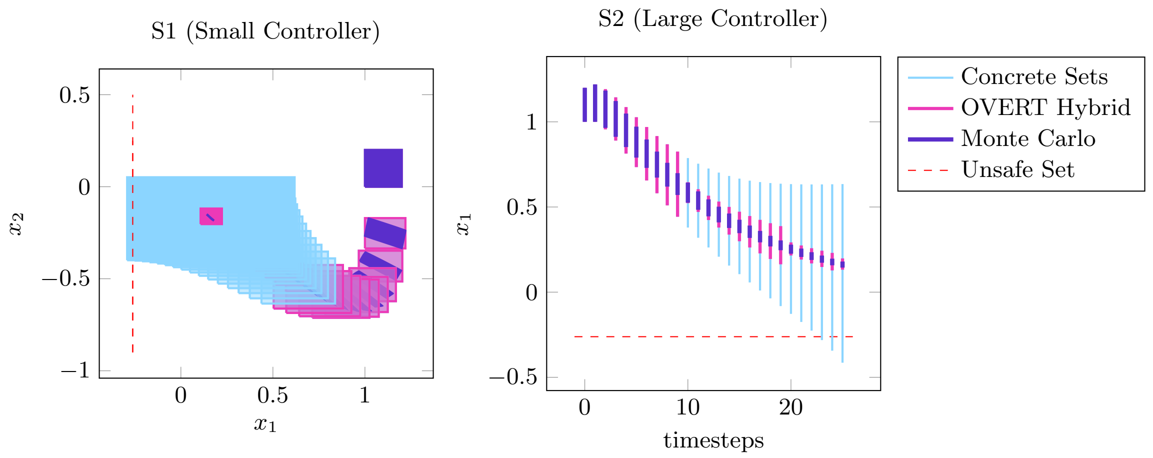


Figure 4: Reachability problem for the single pendulum benchmarks S1 and S2. The OVERT hybrid symbolic approach is compared to a naive approach where 1-step concrete sets are computed at every timestep as well as to the convex hull of of one million Monte Carlo simulations.

dimensional reachable sets for variables $x_{t,2}$ and $x_{t,3}$ for the T3 benchmark, and the bottom left plot show the 1-dimensional reachable sets for variable $x_{t,1}$ for the T3 benchmark. The reachable sets do not leave the safe set, indicated with dashed red lines, and so the safety property is proven.

The plots on the right side of Fig. 5 show the reachability results for example C3. On the top right, the reachable sets of states $x_{t,1}$ and $x_{t,2}$ are shown. We observe that the reachable state sets are never a subset of the goal set indicating that we cannot prove the goal-reaching property holds, but we can show that the intersection of the reachable state sets and the goal set is empty, meaning we can prove that the goal-reaching property never holds. If the sets partially overlapped, the results would be inconclusive. This problem illustrates the versatility of computing the reachable state set concretely. Once the sets have been computed, multiple queries can be answered with minimal additional computation. In contrast, a whole new feasibility problem would have to be solved to address a second query.

The final example is the Adaptive Cruise Control benchmark which is shown in Figs. 6 and 7. Figure 6a shows reachable sets of the measurement:

$$y = x_{lead} - x_{ego} - T_{gap}v_{ego} \tag{22}$$

over 55 timesteps, where symbolic queries were performed at timesteps 20, 40 and 55. Equation (22) is simply a re-arranged version of Eq. (21). In order to maintain safety, we must have that $y > D_{min} = 10$. One can see from the figure that this is indeed the case; the reachable sets do not cross the red dotted line indicating the edge of the unsafe set. One may also observe that the hybrid-symbolic sets offer modest tightness improvements over the concrete sets. This is due to inherent properties of the closed-loop system. Figure 6b shows a top-down view of the ego and lead car positions, as well as the D_{safe}, computed using OVERT's hybrid symbolic approach, and is intended to help the reader visualize the roadway scenario. In this depiction, the property could still hold even

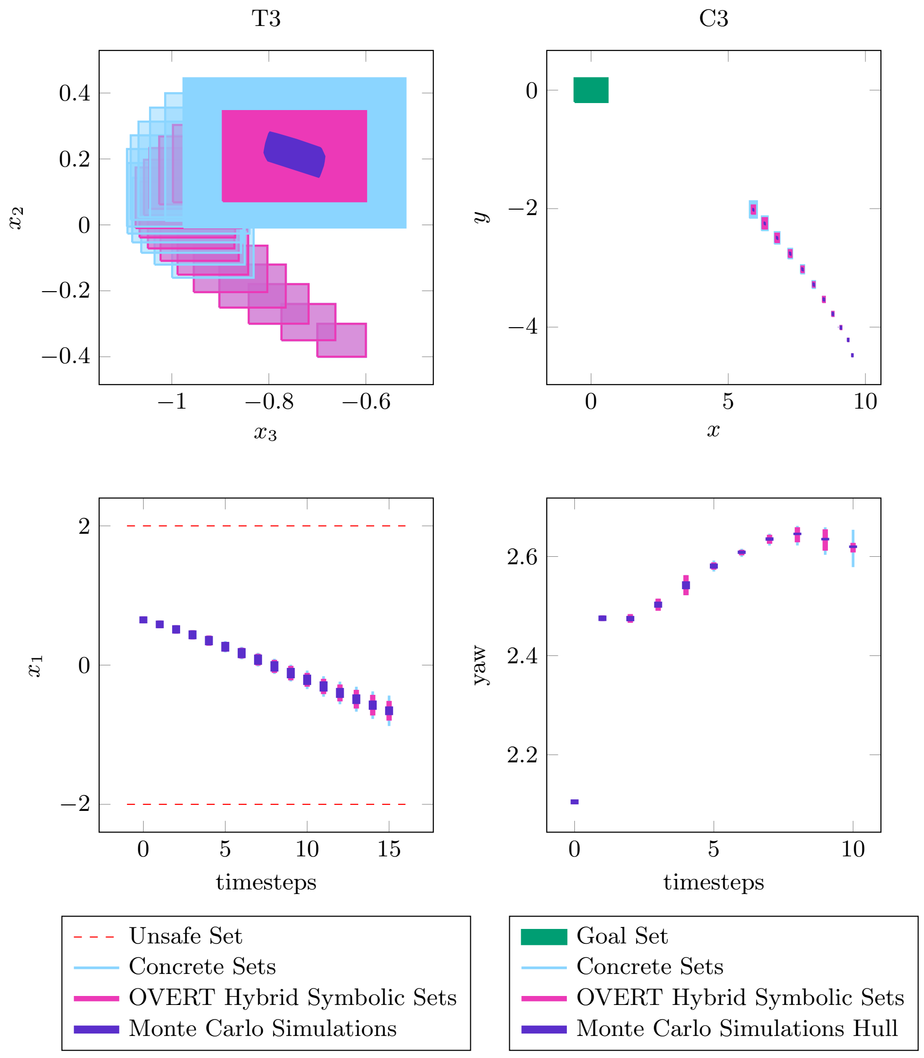
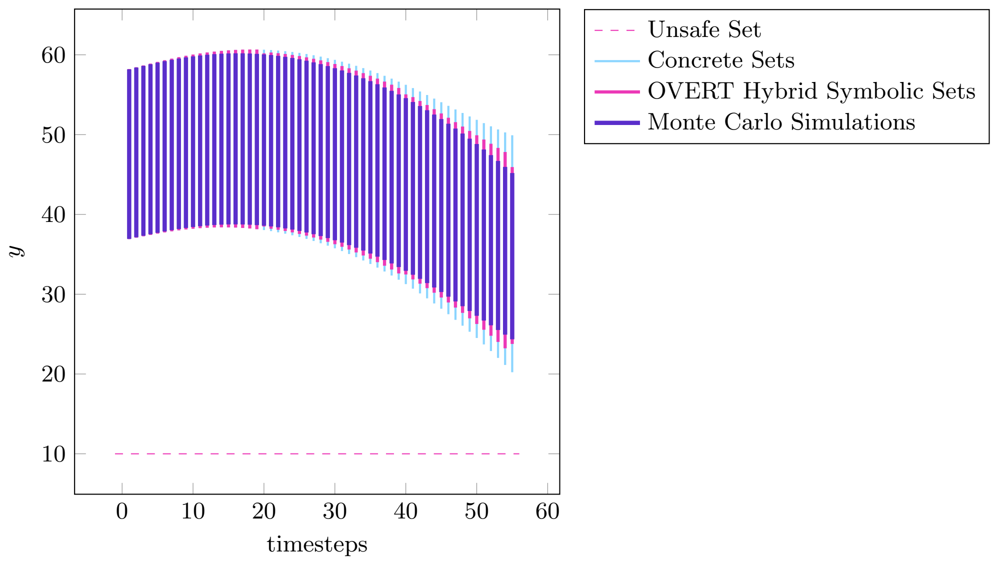
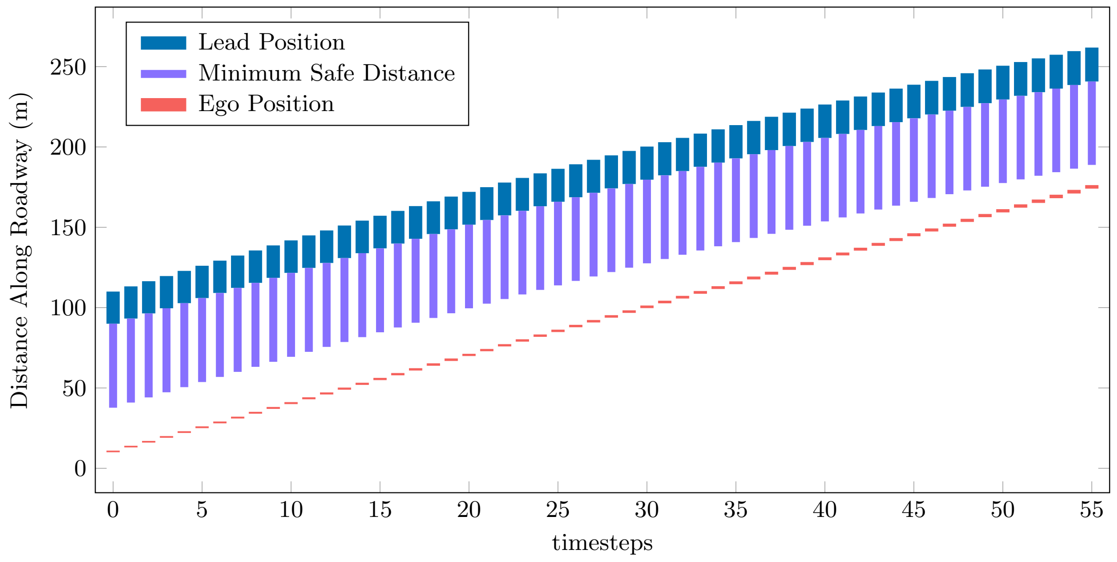


Figure 5: Reachable sets for benchmark examples T3 (left) and C3 (right). Light blue indicates the reachable sets obtained via 1-step concretization and pink indicates those found with hybrid symbolic computation. The convex hull of 1 million Monte Carlo simulations of the original system is shown in purple.



(a) Measurement sets. It can be observed that the property holds, and that the ego vehicle does not get too close to the lead vehicle.



(b) State sets. A top-down view of the reachable sets of the ego car position, lead car position, and safe distance, which is based on ego velocity.

Figure 6: Reachable sets for the Adaptive Cruise Control (ACC) benchmark example.

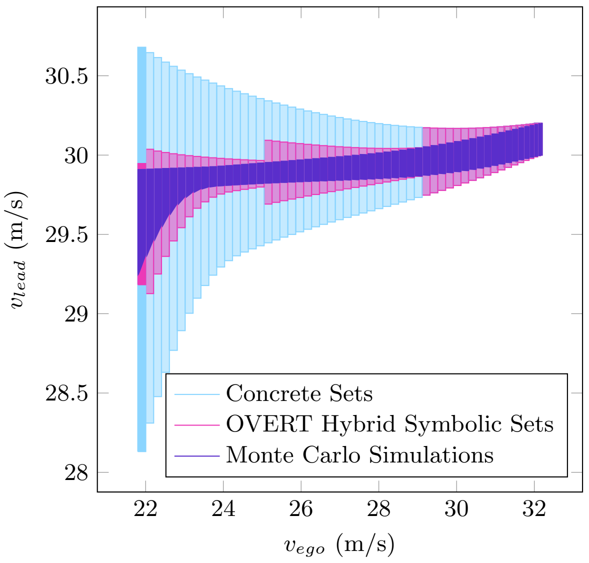


Figure 7: Reachable sets of velocity for the lead and ego vehicles for the Adaptive Cruise Control (ACC) benchmark example. Solid sets indicate the final timestep.

if there were overlap between the ego vehicle position and the safe distance threshold, as the ego vehicle maximum forward position, maximum safe distance, and lead car minimum forward position may not occur simultaneously. Finally, Fig. 7 shows 2-dimensional state sets of the ego vehicle and lead vehicle velocities. This plot demonstrates how tightly the OVERT hybrid-symbolic approach can hug the true system trajectories, compared to the looseness of naively computing 1-step reachable sets at every timestep.

The compute times for each of the reachability problems are shown in Fig. 8a. For smaller problems such as S1 and T1, the compute time is less than 5 minutes. For S2, C1, C2, C3, and ACC, the compute time is less than 15 minutes. However for the larger Tora examples, T2 and T3, the computation time for the reachable set computation framing begins to grow to the order of hours, with T3 taking a little over 18 hours. While the Tora problems have control policies with similar numbers of ReLU activations as the other problems' control policies, the number of weights in the Tora problem control policies is larger than that of the other problems' control policies (Table 2).

Additionally, note that the experimental timing results do not display a sharp trend with state dimension, as S2 is 2-dimensional problem, and ACC is a 6 dimensional problem with more than twice as many timesteps, yet ACC requires only 1.24 times as long to solve. This is as expected, as OVERT's reachable set computation has only explicit linear dependence on the state dimension (the number of optimization problems to be solved is $2n$, where n is the state dimension). In the worst case, the number of activation regions in a network could be exponential in the input dimension (state dimension) (Katz et al., 2017), which could make higher dimensional problems much slower, but our results do not exhibit this worst-case trend.

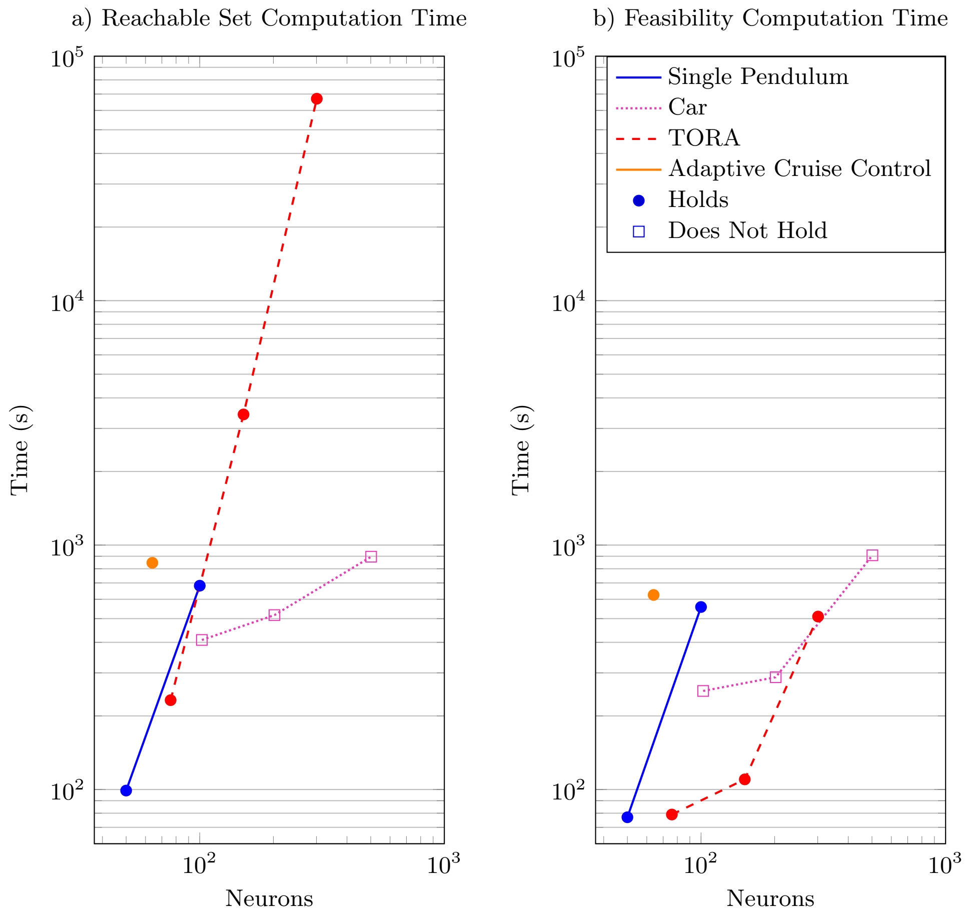


Figure 8: Computation time versus number of ReLU neurons in the neural network control policy.

4.3.2 Feasibility problems

The next set of experiments explore solving the reachability problems described in section 4.1 as feasibility problems. In addition to the closed-loop system and the initial sets for the input parameters, a feasibility query is comprised of a property. Specifically, the complement of the property that we would like to hold is encoded. In the case of the single pendulum for example, the property is staying within the safe set of $x_1 \geq -0.2167$ radians. The unsafe set, $x_1 \leq -0.2167$ radians, is encoded in the feasibility problem.

A feasibility query in OVERT has two possible outcomes. If the solver returns UNSAT, the unsafe set cannot be reached and $x_1 \geq -0.2167$ for the specified time range. If the solver returns SAT, the pendulum may visit a state with angle $\theta \leq -0.2167$ and that the property cannot be guaranteed to hold. In this case, OVERT returns a counter example, which is a trajectory of the system. As OVERT is based on overapproximating the dynamics to create an abstraction of the system, the counter example that the decision procedure finds is an *abstract counter example*, and may be spurious. The abstract counter example may be used to seed an execution of the original, concrete system to generate a true counter example. The novelty of OVERT lies in finding the tightest overapproximation, and therefore one may expect negligible difference between the abstract counter example and the true counter example. If a real counter example is found, then the property is shown to *fail*.

We can see in Fig. 8b that the feasibility approach was able to prove the same number of properties as the explicit reachable set computation approach. We would expect that the feasibility approach should be able to prove the examples that can be proven with explicit reachable set computation, as the feasibility approach incurs less overapproximation. In some problems, the feasibility approach may be able to prove examples that cannot be proven with the explicit reachable set computation approach.

The solver was able to find real counter examples for problems C1, C2, and C3; however, this is trivial due to the fact that the intersection of the reachable sets and goal set was empty, as can be seen in the top right plot of Fig. 5. In other words, every trace from the starting set is a counter example.

Figure 8b also shows the compute time of the feasibility experiments. The compute times for these experiments are reliably shorter than those of the reachable set computation problems, but generally on the same order of magnitude. The TORA problem is a marked exception, where the feasibility framing of the problem is 1 to 2 orders of magnitude faster for problems T1, T2 and T3. The compute time needed to solve problem T3 was reduced from more than 18 hours to less than 9 minutes. We hypothesize that there is some interaction between the wide layers (100 neurons) of the T3 control policy and solving the optimization problems over state variables that are part of reachable set computation. Solving a feasibility problem involves either finding a single feasible solution or showing that no feasible solution exists for each timestep, whereas solving a reachable set computation problem translates to solving $2n$ optimization problems, where n is the state dimension. Given the timing results, if an explicit reachable set is not needed, we recommend solving a feasibility problem instead of explicitly computing the reachable set.

5. Extensions

This section discusses different ways to further improve OVERT, both in terms of finding a tighter reachable set and in terms of compute time.

5.1 Input splitting

Solving reachability problems over large input domains and long time horizons can be computationally expensive. A large input domain means that a large region of the nonlinear dynamics function must be overapproximated, possibly resulting in a larger number of linear and nonlinear constraints. A large input domain may also span a large number of activation patterns in the neural network. Even if the initial set is small, solving over a long time horizon can lead to the reachable sets growing due to overapproximation error, which introduces the same two problems. One possible technique to address this problem is to split the input set, as was demonstrated by Wu et al. (2020). The split problems are independent of each other, and therefore, can be run in parallel, potentially reducing the runtime. Input splitting can also serve to reduce the looseness of the overapproximation of the reachable set (Xiang et al., 2018b). The reachable set is then no longer represented as a single hypercube but as the overlap of several smaller hypercubes. We ran preliminary trials of this strategy and found that the reachable set computed was tighter, and leave a more complete exploration of the effect on runtime to future work.

5.2 Effect of L_1 regularization

It is well-known that an L_1 regularization term in the loss function, when training neural networks, promotes sparsity of the weight matrices. Fewer non-zero weights can lead to a simpler verification problem, as there will be fewer variables or simpler constraints in the verification query (Narodytska et al., 2019; Tjeng et al., 2019). We tested this idea here by training a network with a configuration similar to the network used in the T1 problem (i.e. three hidden layers of 25 neurons each) with L_1-regularizing terms for the weights of the hidden layers. We observed a significant speedup for reachable set computation problems. While this initial finding confirms previous work, more studies are necessary to further elucidate the effect of L_1 regularization on reachable set computation. In addition, this should be studied in conjunction with the effect of L_1 regularization on the performance of the network.

5.3 Smooth Activation Functions

Earlier, we described how OVERT has been designed for networks with piecewise linear activation functions such as ReLUs. It is, however, possible to use OVERT with smooth activation functions such as tanh or sigmoid by overapproximating each smooth nonlinear activation function in the same manner that the system dynamics are overapproximated. We do not expect such an approach to scale as well as problems containing ReLU networks, but ran an initial exploration. We trained a new network for the inverted pendulum problem with tanh activation functions. The network has a single layer with 25 neurons, which is purposefully smaller than those studied earlier. Reachable sets were computed for $n = 20$ timesteps, with symbolic queries performed at timesteps 10 and 20. The concrete and

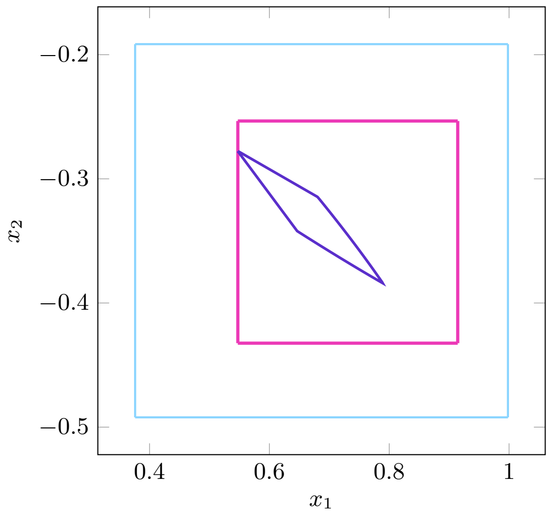


Figure 9: Reachable set computation problem using single pendulum dynamics with a neural network control policy that has only one hidden layer with 25 neurons with tanh activation functions. Boxes indicate reachable sets at timestep $n_t = 20$ obtained via 1-step concretization (blue) and hybrid-symbolic computation (pink). The convex hull of Monte Carlo simulations is shown in purple.

symbolic reachable sets produced approximate the convex hull of Monte Carlo simulations closely, as can be seen in Fig. 9. Although every smooth nonlinearity is approximated, the accuracy provided by OVERT's optimal piecewise-linear overapproximations still yields tight reachable sets.

6. Conclusion

In this paper, we introduced a methodology for analyzing closed-loop discrete-time dynamical systems with neural network control policies from a safety perspective. Such systems are typically governed by nonlinear dynamical updates, which limit the use of existing neural verification tools. To handle this, we overapproximate the nonlinear dynamics with piecewise linear relations. The piecewise linear relations are then encoded into an MIP and solved using a commercial MIP solver. Our experiments on a proof of concept implementation demonstrate that OVERT outperforms naive adaptations of existing tools for use with nonlinear discrete-time closed-loop systems with neural network control policies in both tightness of computed reachable sets and speed of computation. Our work contributes to ongoing efforts to apply verification tools to real-world cyber physical systems, such as autonomous vehicles and airplane collision avoidance systems.

Acknowledgments

The authors would like to thank Clark Barrett, Christopher Strong, Sydney Katz, Aleksandar Zeljic, Kyle Julian, Changliu Liu, Tomer Arnon, Joe Vincent, and Ransalu Senanayake for their support and feedback. This work was supported by AFRL and DARPA under contract FA8750-18-C-0099. A. Maleki acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Appendix A. Algorithms for Overapproximation

This section provides detailed derivations of our algorithms for finding a closed form piecewise-linear overapproximation of any function mapping $\mathbb{R}^n \to \mathbb{R}$. Algorithm 1 breaks the problem down into finding optimal piecewise-linear overapproximations for one-dimensional functions mapping $\mathbb{R} \to \mathbb{R}$ (Appendix A.2). Each piecewise-linear overapproximation is formed using an upper and lower bound as is specified in Algorithm 2. Finally, each piecewise-linear bound is represented as a single closed-form function (Section 3.3).

A.1 Optimality of the Midpoint Tangent Bound

Consider a function f that is strictly concave over the interval $[x_{i-1}, x_i]$. Over this interval, we seek to construct the tightest upper bound for this function that consists of a single tangent line segment. The upper bound $g_i(x, \alpha)$ is defined to be tangent to f at some point α, $\alpha \in [x_{i-1}, x_i]$:

$$g_i(x, \alpha) = f'(\alpha) \cdot (x - \alpha) + f(\alpha)$$

The "tightest" upper bound is defined as the bound which minimizes the area between the bound and the function:

$$\min_{\alpha} \int_{x_{i-1}}^{x_i} (g_i(x, \alpha) - f(x))\, \mathrm{d}x$$

As the area under the function f does not depend on α, we focus on minimizing the area $A(\alpha)$ under the bound:

$$A(\alpha) = \int_{x_{i-1}}^{x_i} g_i(x, \alpha) \mathrm{d}x$$

The area $A(\alpha)$ under the bound can alternately be expressed using the midpoint integral approximation, which is exact for a linear function:

$$A(\alpha) = g(x_{mid}, \alpha) \cdot (x_i - x_{i-1})$$

where α is the tangent point and x_{mid} is the midpoint: $x_{mid} = \frac{x_i + x_{i-1}}{2}$ (dropping the subscript i for simplicity). As the term $(x_i - x_{i-1})$ is fixed, we can further focus on minimizing only $g(x_{mid}, \alpha)$.

Consider choosing $\alpha = x_{mid}$. The term $g(x_{mid}, \alpha)$ evaluates to:

$$\begin{aligned} g_i(x_{mid}, \alpha = x_{mid}) &= f'(x_{mid}) \cdot (x_{mid} - x_{mid}) + f(x_{mid}) \\ &= f(x_{mid}) \end{aligned}$$

Or, in other words, $g_i(x_{mid}, \alpha = x_{mid})$ lies *on* the function.

Next, consider choosing $\alpha = \beta$, where $\beta \neq x_{mid}$ is any other point in the interval $[x_i - x_{i-1}]$ different from x_{mid}. While the bound $g_i(x, \beta)$ lies on the function at $x = \beta$:

$$\begin{aligned} g_i(x = \beta, \beta) &= f'(\beta) \cdot (\beta - \beta) + f(\beta) \\ &= f(\beta) \end{aligned}$$

evaluating $g_i(x, \beta)$ at any other point x along the bound, including x_{mid}, necessarily produces points *above* the function:

$$g_i(x, \beta) > f(x) \ , \ \forall x \neq \beta \ , \ x \in [x_i - x_{i-1}]$$

as the tangent to a strictly concave function touches the function at exactly one point and lies above the function at all other points. Consequently,

$$g_i(x_{mid}, \beta) > f(x_{mid})$$

and because $g_i(x_{mid}, \alpha = x_{mid}) = f(x_{mid})$ this implies that

$$g_i(x_{mid}, \beta) > g_i(x_{mid}, x_{mid})$$

and therefore that $\alpha = x_{mid}$ is the minimizer of $g_i(x_{mid}, \alpha)$ over the interval $[x_i - x_{i-1}]$. This further implies, as reasoned above, that

$$\begin{aligned} x_{mid} &= \arg\min_{\alpha} A(\alpha) \\ &= \arg\min_{\alpha} \int_{x_{i-1}}^{x_i} \left(g_i(x, \alpha) - f(x)\right) \mathrm{d}x \end{aligned}$$

or, in other words, that the tangent line at the midpoint is the tightest tangent bound for a strictly concave function over a fixed interval.

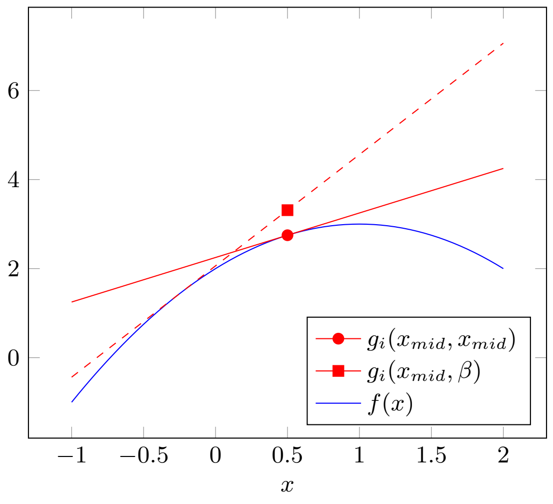


Figure 10: Graphical depiction of midpoint optimality proof.

A.2 Bounding 1D (scalar) functions

Consider a scalar function mapping $f\colon \mathbb{R}\to\mathbb{R}$. Our goal is to find the tightest piecewise-linear upper and lower bound functions over domain $[p, q]$. In order to find such bounds, we need to divide domain $[p, q]$ into intervals within which $f''(x)$ does not change sign. That is, within each interval $[a, b]$, $f(x)$ is either convex or concave. In order to have a continuous function and express the piecewise-linear function in closed form, as described in section Section 3.3, the bound from each convex or concave interval must be coincident to the function at the interval endpoints.

More formally, given a natural number n, our goal is to find the tightest piecewise-linear upper or lower bound function $g(x)$ for f over interval $[a, b]$ that is composed of n linear pieces (see Fig. 1). More specifically, we choose $n-1$ points $x_1, x_2, \ldots, x_{n-1}$ within the interval $[a, b]$ such that: $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$. Then,

$$g_i(x) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_i) + y_i, \quad x_{i-1} \le x \le x_i \tag{23}$$

where points $(x_i, y_i), i = 1, 2, \ldots n-1$ need to be specified, and $g(x_0 = a) = f(a)$ and $g(x_n = b) = f(b)$. The following algorithms then specify how to *optimally* find points (x_i, y_i) in Eq. (23) within each interval.

A.2.1 Upper (lower) bound for a 1D convex (concave) function

The tightest upper bound function $g(x)$ in Eq. (23) for a convex function $f(x)$ over interval $[a, b]$ is specified by selecting points x_i that satisfy the following system of equations:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}}, \quad \text{for} \quad i = 1, 2, \ldots n-1, \tag{24}$$

$$y_i = f(x_i) \tag{25}$$

To ensure continuity $x_0 = a, x_n = b$, and $y_0 = f(x_0), y_n = f(x_n)$.

Proof: Here we prove that $g(x)$ is a tight upper bound for a convex function. One can similarly show that this is a tight lower bound for a concave function.

Consider function $f(x)$ and its overapproximation $g(x; x_0{:}x_n)$, where $x_0{:}x_n$ denotes $n+1$ points $x_0, x_1, x_2, \ldots, x_n$ along the x-axis. Function g is comprised of n linear segments $g_i(x)$ between x_{i-1} and x_i as shown in Fig. 1. Since $f(x)$ is convex, application of Jensen's inequality straightforwardly yields $g_i(x) \ge f(x)$ for $x_{i-1} \le x \le x_i$, suggesting that $g(x) \ge f(x), \forall x \in [a, b]$. Finding the tightest possible g amounts to minimizing the shaded area in Fig. 1, or equivalently, solving the following minimization problem:

$$\min_{x_1:x_{n-1}} \int_a^b [g(x; x_0{:}x_n) - f(x)]\, \mathrm{d}x \tag{26}$$

Notice that $f(x)$ is a constant with respect to $x_0{:}x_n$, and can be removed from the objective.

We rewrite Eq. (26):

$$\begin{aligned}\min_{x_1:x_{n-1}} \int_a^b [g(x;x_0{:}x_n) - f(x)]\,\mathrm{d}x &= \min_{x_1:x_{n-1}} \sum_{i=0}^{n-1} \int_{x_{i-1}}^{x_i} g_i(x;x_{i-1},x_i)\mathrm{d}x \\ &= \min_{x_1:x_{n-1}} \sum_{i=0}^{n-1} \int_{x_{i-1}}^{x_i} \left[\frac{f(x_i)-f(x_{i-1})}{x_i - x_{i-1}}(x-x_{i-1}) + f(x_{i-1})\right]\mathrm{d}x\end{aligned}$$

Using the fact that the midpoint approximation of the integral of a line is exact, we can write the expression as:

$$= \min_{x_1:x_{n-1}} \sum_{i=0}^{n-1} \left(\frac{f(x_i)+f(x_{i-1})}{2}(x_i - x_{i-1})\right)$$

We take derivative of the objective with respect to $x_j; j = 1, 2, \ldots, n-1$ and set to zero:

$$\begin{aligned}0 =& \frac{\partial}{\partial x_j} \sum_{i=0}^{n-1} \left(\frac{f(x_i)+f(x_{i-1})}{2}(x_i - x_{i-1})\right) \\ =& \frac{\partial}{\partial x_j}\left(\frac{f(x_j)+f(x_{j-1})}{2}(x_j - x_{j-1})\right) + \frac{\partial}{\partial x_j}\left(\frac{f(x_j)+f(x_{j+1})}{2}(x_{j+1} - x_j)\right) \\ =& \frac{1}{2}\left[f'(x_j)(x_j - x_{j-1}) + f(x_j) + f(x_{j-1}) + f'(x_j)(x_{j+1}-x_j) - f(x_j) - f(x_{j+1})\right] \\ =& \frac{1}{2}\left[f'(x_j)(x_{j+1}-x_{j-1}) + f(x_{j-1}) - f(x_{j+1})\right]\end{aligned}$$

which simplifies to

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{x_{j+1} - x_{j-1}} \blacksquare \tag{27}$$

While we do not show here that this stationary point is a minimum, empirically, we have observed that it appears to be. Points (x_i, y_i) satisfying the optimality condition in Eq. (27) are found using a numerical routine provided by NLsolve. If there is error in the numerical routine and the x_i are not optimally placed, the bound produced will still be valid and continuous, as any secant connecting two points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$ of a convex function is a valid upper bound over the interval $[x_{i-1}, x_i]$.

A.2.2 Upper (lower) bound for a 1D concave (convex) function

Consider a function $f{:}\mathbb{R} \to \mathbb{R}$ that is concave over interval $[a, b]$. Similar to the derivation procedure in Appendix A.2.1, we find the optimum points x_i in Eq. (23) by minimizing the area between $g(x)$ and $f(x)$.

The tightest upper bound function $g(x)$ for a concave function $f(x)$ over interval $[a, b]$ is specified by points $x_i, i = 1, 2, \ldots, n-1$ that satisfy:

$$x_i = h\left(\frac{x_{i-1}+x_i}{2}, \frac{x_i + x_{i+1}}{2}\right) \tag{28}$$

where

$$h(\alpha,\beta)=\frac{\beta f'(\beta)-\alpha f'(\alpha)}{f'(\beta)-f'(\alpha)}-\frac{f(\beta)-f(\alpha)}{f'(\beta)-f'(\alpha)} \tag{29}$$

The y_i values are given by

$$\begin{aligned} y_i &= g_i(x_i) \\ &= f'\left(\frac{x_{i-1}+x_i}{2}\right)\left(x_i-\frac{x_i+x_{i-1}}{2}\right)+f\left(\frac{x_i+x_{i-1}}{2}\right) \\ &= f'\left(\frac{x_{i-1}+x_i}{2}\right)\left(\frac{x_i-x_{i-1}}{2}\right)+f\left(\frac{x_i+x_{i-1}}{2}\right) \end{aligned}$$

Proof: Consider line segment $g_i(x)$ that is an upper bound for the concave function $f(x)$ over the interval $[x_{i-1}, x_i]$. Since $f(x)$ is concave, $g_i(x)$ may not intersect $f(x)$ at more than one points; otherwise, between the intersecting points, by Jensen's inequality, $g_i(x) < f(x)$. That means, $g_i(x)$ can at best be tangent to $f(x)$. Denote the tangent point $\alpha \in [x_{i-1}, x_i]$. We showed in Appendix A.1 that if

$$\alpha=\frac{x_{i-1}+x_i}{2}$$

the area between $f(x)$ and $g_i(x)$ is minimized. That means once we find x_{i-1} and x_i, function g_i is the line that is tangent to $f(x)$ at $(x_{i-1}+x_i)/2$. We now turn our attention to optimally distribute the x_i's. Similar to the previous algorithm, we would like to minimize the area between $f(x)$ and $g(x)$:

$$\begin{aligned} \min_{x_1:x_{n-1}} \int_a^b [g(x;x_0{:}x_n)-f(x)]\,\mathrm{d}x &= \min_{x_1:x_{n-1}} \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} g_i(x;x_{i-1},x_i)\mathrm{d}x \\ &= \min_{x_1:x_{n-1}} \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f'\left(\frac{x_{i-1}+x_i}{2}\right)(x-\frac{x_{i-1}+x_i}{2})+f\left(\frac{x_{i-1}+x_i}{2}\right)\mathrm{d}x \\ &= \min_{x_1:x_{n-1}} \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)(x_i-x_{i-1}) \end{aligned}$$

Taking derivative with respect to $x_j, j=1,2,\ldots,n-1$ and setting to zero:

$$\begin{aligned} 0 =& \frac{\partial}{\partial x_j}\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)(x_i-x_{i-1}) \\ =& \frac{\partial}{\partial x_j}\left(f\left(\frac{x_{j-1}+x_j}{2}\right)(x_j-x_{j-1})\right)+\frac{\partial}{\partial x_j}\left(f\left(\frac{x_{j+1}+x_j}{2}\right)(x_{j+1}-x_j)\right) \\ =& \frac{1}{2}f'\left(\frac{x_{j-1}+x_j}{2}\right)(x_j-x_{j-1})+f\left(\frac{x_{j-1}+x_j}{2}\right)+\frac{1}{2}f'\left(\frac{x_{j+1}+x_j}{2}\right)(x_{j+1}-x_j)-f\left(\frac{x_{j+1}+x_j}{2}\right) \end{aligned}$$

which simplifies to

$$\begin{aligned} \frac{1}{2}f'\left(\frac{x_i+x_{i-1}}{2}\right)(x_i-x_{i-1})+f\left(\frac{x_i+x_{i-1}}{2}\right) = \\ \frac{1}{2}f'\left(\frac{x_i+x_{i+1}}{2}\right)(x_i-x_{i+1})+f\left(\frac{x_i+x_{i+1}}{2}\right) \end{aligned} \tag{30}$$

Re-arranging Eq. (30) and solving for x_i, we can reproduce Eq. (28). It is worth showing that the optimality constraint (Eq. (30)) also enforces continuity over points $x_1, x_2, \ldots, x_{n-1}$. To show this, we re-arrange Eq. (30):

$$f'\left(\frac{x_i + x_{i-1}}{2}\right)\left(\frac{x_i - x_{i-1}}{2}\right)+f\left(\frac{x_i + x_{i-1}}{2}\right) = f'\left(\frac{x_i + x_{i+1}}{2}\right)\left(\frac{x_i - x_{i+1}}{2}\right)+f\left(\frac{x_i + x_{i+1}}{2}\right)$$

$$\Rightarrow$$

$$f'\left(\frac{x_i + x_{i-1}}{2}\right)\left(x_i - \frac{x_i + x_{i-1}}{2}\right)+f\left(\frac{x_i + x_{i-1}}{2}\right) = f'\left(\frac{x_i + x_{i+1}}{2}\right)\left(x_i - \frac{x_i + x_{i+1}}{2}\right)+f\left(\frac{x_i + x_{i+1}}{2}\right)$$

$$\Rightarrow$$

$$g_i(x_i) = g_{i+1}(x_i)$$

which indicates that $g(x)$ is continuous within interval (a, b). ■

While we do not show here that the stationary point satisfying Eq. (28) is a minimum, empirically we have observed that the stationary point is a unique minimum. Proof for arbitrary concave f is still open.

If function $f(x)$ contains both convex and concave sub-intervals, the application of Eq. (24) and Eq. (28) leads to discontinuity at the inflection points. Note that continuity is necessary in order to be able to express piecewise-linear function in closed form; see Section 3.3. Discontinuity results because the value of over-approximating function $g(x)$ at the endpoints of concave sub-intervals is not equal to that of $f(x)$. To address this, we enforce that the bound $g(x)$ is tangent to the function f at both a and b, which implies that $g(a) = f(a)$ and $g(b) = f(b)$ on all concave sub-intervals (notice that this condition is already satisfied on convex sub-intervals). Therefore, the bound is given by points (x_i, y_i) that satisfy:

$$\begin{aligned}
x_0 =& a \\
x_1 =& h\left(a, \frac{x_1 + x_2}{2}\right) \\
x_i =& h\left(\frac{x_{i-1} + x_i}{2}, \frac{x_i + x_{i+1}}{2}\right), \quad i = 2, \ldots n-2 \\
x_{n-1} =& h\left(\frac{x_{n-1} + x_n}{2}, b\right) \\
x_n =& b
\end{aligned}$$

where h is given by Eq. (29), and y_i is given by:

$$\begin{aligned}
y_0 =& f(a) \\
y_1 =& f'(a)(x_1 - a) + f(a) \\
y_i =& f'\left(\frac{x_{i-1} + x_i}{2}\right)\left(\frac{x_i - x_{i-1}}{2}\right) + f\left(\frac{x_{i-1} + x_i}{2}\right), \; i = 2, ..., (n-2) \\
y_{n-1} =& f'(b)(x_{n-1} - b) + f(b) \\
y_n =& f(b)
\end{aligned}$$

Satisfying pairs (x_i, y_i) are again found using a numerical routine provided by NLsolve. If the optimality conditions are not met, the bound will be sound but not continuous. The bound may be repaired by taking $y_i = \max(g_i(x_i), g_{i+1}(x_i))$ if it is an upper bound for a concave function. The resulting line segment forming an upper bound may only be shifted upward by this process and thus will still constitute a valid upper bound. For a lower bound for a convex function, the bound may be analogously repaired by taking $y_i = \min(g_i(x_i), g_{i+1}(x_i))$.

References

Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. Reachability analysis for neural agent-environment systems. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2018.

Michael Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. Formal verification of neural agents in non-deterministic environments. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 25–33, 2020.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer-Aided Verification (CAV)*, pages 258–263, 2013.

Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Transactions on Computational Logic (TOCL)*, 19 (3):19, 2018.

Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

Anthony Corso, Robert J. Moss, Mark Koren, Ritchie Lee, and Mykel J. Kochenderfer. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72(2005.02979):377–428, 2021. doi: 10.1613/jair.1.12716.

Leonardo De Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *International Conference on Automated Deduction*, pages 438–455. Springer, 2002.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151–156, 2018.

Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *ACM International Conference on Hybrid Systems: Computation and Control*, pages 157–168, 2019.

Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems.* 2009.

Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

Mrdjan Jankovic, Daniel Fontaine, and Petar V. Kokotovic. Tora example: cascade-and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3): 292–297, 1996.

Taylor Johnson, Diego Lopez, Patrick Musau, Hoang Tran, Tran Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, and Chao Huang. Arch-comp20 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. *EPiC Series in Computing*, 74:107–139, 2020.

Kyle D. Julian and Mykel J. Kochenderfer. Reachability analysis for neural network aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 44(6):1132–1142, 2021.

Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC)*, pages 1–10, 2016.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117, 2017.

Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, 4(3–4):244–404, 2021.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nina Narodytska, Hongce Zhang, Aarti Gupta, and Toby Walsh. In search for a SAT-friendly binarized neural network architecture. In *International Conference on Learning Representations (ICLR)*, 2019.

Arnold Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. In *Validation numerics*, pages 175–190. Springer, 1993.

Peter Øhrstrøm and Per FV Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Kluwer Academic Publishers, 1995.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016.

Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor Johnson. NNV: A tool for verification of deep neural networks and learning-enabled autonomous cyber-physical systems. In *International Conference on Computer-Aided Verification*, 2020.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, pages 1599–1614, 2018.

Haoze Wu, Alex Ozdemir, Aleksandar Zeljic, Kyle Julian, Ahmed Irfan, Divya Gopinath, Sadjad Fouladi, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. Parallelization techniques for verifying neural networks. In *Formal Methods in Computer Aided Design (FMCAD)*, 2020.

Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018a.

Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Specification-guided safety verification for feedforward neural networks. *arXiv preprint arXiv:1812.06161*, 2018b.

Weiming Xiang, Hoang-Dung Tran, Joel A. Rosenfeld, and Taylor T. Johnson. Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In *American Control Conference (ACC)*, pages 1574–1579. IEEE, 2018c.