Automatic and Scalable Safety Verification using Interval Reachability with Subspace Sampling

Brendan Gould, Akash Harapanahalli, and Samuel Coogan

***Abstract*— Interval refinement is a technique for reducing the conservatism of traditional interval based reachability methods by *lifting* the system to a higher dimension using new auxiliary variables and exploiting the introduced structure through a *refinement* procedure. We present a novel, efficiently scaling, automatic refinement strategy based on a subspace sampling argument and motivated by reducing the number of interval operations through sparsity. Unlike previous methods, we guarantee that refined bounds shrink as additional auxiliary variables are added. This additionally encourages automation of the lifting phase by allowing larger groups of auxiliary variables to be considered. We implement our strategy in JAX, a high-performance computational toolkit for Python and demonstrate its efficacy on several examples, including regulating a multi-agent platoon to the origin while avoiding an obstacle.**

***Index Terms*— Safety-Critical Systems, Reachability Analysis, Interval Arithmetic**

I. Introduction

Engineered systems are subject to a wide variety of uncertainties, including measurement errors, mechanical failures, and unmodeled dynamics. These uncertainties may cause the realized behavior of the system to deviate significantly from its nominal behavior, which is unacceptable in safety-critical systems. To combat this problem, many approaches have been developed to verify the safety of a system in the presence of such uncertainty. One such strategy is *reachable set over-approximation* [1], [2], where safety is guaranteed for an entire set containing all possible states a system may attain under admissable uncertainties and disturbances.

The set representation chosen for this over-approximation is of great importance—typically resulting in a tradeoff between bound computation time and accuracy. Some commonly used set representations include zonotopes [3], constrained zonotopes [4], hybrid zonotopes [5], [6], polytopes [7], Taylor models [8], ellipsoids [9], and generalized star sets [10]. Intervals [11] are a particularly simple (and therefore fast to compute) representation of sets, and multiple approaches [12], [13] have been proposed for efficient interval reachable set computation. The efficiency of interval methods has allowed them to, *e.g.*, be incorporated directly to train neural networks controllers with certified robustness guarantees [14]. Other reachability approaches have also been proposed, like Hamilton-Jacobi reachability [15], level set methods [16], and contraction-based reachability [17].

The simplicity of interval-based approaches causes them to suffer from excessive over-conservatism through the wrapping effect, preventing them from producing useful bounds in many cases. Recent work [12] has developed methods to mitigate this conservatism by artificially introducing additional structure to a system through model redundancies, which was used in [14] to study invariant polytopes in neural network controlled systems. In Section II, we recall this process through the lens of a lifted system, and describe how domain knowledge can be used to select which lifted structure to create. The new structure can be exploited to *refine* bounds for the lifted system, often giving tighter estimates than standard interval reachability.

In Section III, we present the main contribution of this work: a novel, automated refinement strategy based on sampling certain subspaces. Through clever parameterization of these subspaces, we induce a desirable sparsity pattern, which heuristically reduces conservatism by cancelling interval operations performed during refinement. Furthermore, we explicitly bound the growth of refinement costs in both the size of the original system and the amount of added auxilliary variables with Proposition 2, and show how refined bounds are monotonically shrinking as additional redundancies are introduced in Theorem 1. These results together represent a step towards the automation of interval refinement, allowing engineers to quickly test many alternative lifting strategies.

Section IV demonstrates our technique on several examples. First, we present a framework that fully automates the lifting and refinement for two dimensional systems. Next, we show how lifted systems can use some domain knowledge with our automatic refinement procedure to successfully compute useful reachable set bounds on concentrations of a chemical reaction and positions of agents in a multi-agent platoon. Finally, Section V gives a brief conclusion and suggests directions for future work.

A. Notation

We use δ_i to denote the ith standard Euclidean basis vector. In all other cases, vector subscripts denote indexing: y_j is the jth component of the vector y. Let $\leq$ be the elementwise order on $\mathbb{R}^n$ so that $x \leq y$ when $x_i \leq y_i$ for every i. Let $[\underline{y}, \overline{y}] := \{y : \underline{y} \leq y \leq \overline{y}\}$ be a closed interval subset of $\mathbb{R}^n$ and $\mathbb{IR}^n$ denote the set of all n-dim intervals. The symbol ${}^mP_n$ is the set of permutations of n elements from a list of size m, and $|{}^mP_n|$ is the size of this set.

This work was supported in part by the National Science Foundation under awards #2219755 and #2333488 and by the Air Force Office of Scientific Research under Grant FA9550-23-1-0303.

The authors are affiliated with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA {bgould6, aharapan, sam.coogan}@gatech.edu.

II. Polytope Reachability

In this section, we present an approach for over-approximating the reachable sets of continuous-time systems. Our technique is a generalization of the procedure presented in [12], using interval arithmetic with a refinement step.

A. Lifting and Refinement

Consider the nonlinear system

$$\dot{x} = f(x, w), \tag{1}$$

for $x \in \mathbb{R}^n$ as the state, $w \in \mathbb{R}^p$ as a disturbance, and continuous $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$ locally Lipschitz in x for fixed w. Let $H \in \mathbb{R}^{m\times n}$ (with $m > n$) be full rank and H^+ satisfy $H^+H = \mathbf{I}_n$. We define the (H, H^+)-*lifted system*:

$$\dot{y} = Hf(H^+y, w). \tag{2}$$

Since H is full rank, it must have at least n linearly independent rows. Without loss of generality, we assume that the first n rows of H form an invertible block H_V (allowing us to recover x from y) and call the remaining block H_A. H_V acts as a state transformation on the *base states* $x \in \mathbb{R}^n$, H_A is another transformation giving a vector of *auxiliary states*, and their concatenation gives the *lifted states* $y \in \mathbb{R}^m$.

By [14, Proposition 11], the linear subspace $\mathcal{H} = \{Hx : x \in \mathbb{R}^n\}$ is forward invariant for the lifted dynamics (2). Next, we recall the definition of an interval refinement operator [12], which will allow us to exploit this information.

Definition 1 (Interval refinement operator [12])**.** Given $\mathcal{S} \subseteq \mathbb{R}^m$, we say $\mathcal{I}_\mathcal{S} : \mathbb{IR}^m \to \mathbb{IR}^m$ is an *interval refinement operator* if for every interval $[\underline{y}, \overline{y}] \in \mathbb{IR}^m$ we have

$$\mathcal{S} \cap [\underline{y}, \overline{y}] \subseteq \mathcal{I}_\mathcal{S}([\underline{y}, \overline{y}]) \subseteq [\underline{y}, \overline{y}]. \tag{3}$$

In our context, $\mathcal{I}_\mathcal{H}$ tightens known bounds on lifted system trajectories; if $y(t) \in [\underline{y}, \overline{y}]$, then also $y(t) \in \mathcal{I}_\mathcal{H}([\underline{y}, \overline{y}])$. $\mathcal{I}_\mathcal{H}([\underline{y}, \overline{y}])$ is often significantly smaller than $[\underline{y}, \overline{y}]$ itself, improving bounds. Below, we present two refinement operators for $\mathcal{H}$, varying in tightness and computational complexity.

The tightest refinement operator for a lifted system is $\mathcal{I}^{\mathrm{LP}}_\mathcal{H} := [\underline{\mathcal{I}^{\mathrm{LP}}_\mathcal{H}}([\underline{y}, \overline{y}]), \overline{\mathcal{I}^{\mathrm{LP}}_\mathcal{H}}([\underline{y}, \overline{y}])]$, where

$$\underline{\mathcal{I}^{\mathrm{LP}}_\mathcal{H}}([\underline{y}, \overline{y}])_j = \min_{x\in\mathbb{R}^n} \delta_j{}^\top Hx \quad \text{s.t.} \quad \underline{y} \le Hx \le \overline{y}, \tag{4}$$

$$\underline{\mathcal{I}^{\mathrm{LP}}_\mathcal{H}}([\underline{y}, \overline{y}])_j = \max_{x\in\mathbb{R}^n} \delta_j{}^\top Hx \quad \text{s.t.} \quad \underline{y} \le Hx \le \overline{y}, \tag{5}$$

for each $j \in \{1, 2, \ldots, m\}$, as originally proved in [7]. To refine an interval in $\mathbb{IR}^m$ with $\mathcal{I}^{\mathrm{LP}}_\mathcal{H}$, $2m$ linear programs must be solved. As the number of states in the lifted system (2) grows, this computation becomes burdensome.

To remedy this issue, we consider another refinement operator. For any $N \in \mathbb{N}$, let $A \in \mathbb{R}^{N\times m}$ such that $AH = 0$, and define the *sampling strategy* refinement operator:

$$\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}([\underline{y}, \overline{y}])_j = [\underline{y}_j, \overline{y}_j] \bigcap_{\substack{i\in\{1,\ldots,N\},\\ A_{i,j}\neq 0}} -\frac{1}{A_{i,j}} \sum_{k\neq i} A_{i,k}[\underline{y}_k, \overline{y}_k], \tag{6}$$

for all $j \in \{1, 2, \ldots, m\}$. By (3), if $\mathcal{H} \cap [\underline{y}, \overline{y}] = \emptyset$, then (6) is vacuous and we may choose any output, *e.g.*, the singleton set $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}([\underline{y}, \overline{y}]) = [\underline{y}, \underline{y}]$. This generalizes a refinement operator proposed in [12] and used in [14, Appendix 3] by allowing A to be any matrix in the left null space of H rather than just a basis. Our contribution is a clever parameterization of A, allowing us to theoretically characterize both the complexity (Proposition 2) and tightness (Theorem 1) of (6).

Though the bounds provided by $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ may not be as tight as those of $\mathcal{I}^{\mathrm{LP}}_\mathcal{H}$, they are computed by simple arithmetic operations and therefore much faster. In Section III, we discuss a principled approach for selecting A and show that it exhibits desirable growth properties.

B. Embedding Systems

As in the previous section, let $\dot{y} = g(y, w) := Hf(H^+y, w)$ be an (H, H^+)-lifted system, and let $\mathcal{I}_\mathcal{H}$ be a refinement operator on $\mathcal{H} \subseteq \mathbb{R}^m$. An *inclusion function* G for g is a mapping $\mathsf{G} = [\underline{\mathsf{G}}, \overline{\mathsf{G}}] : \mathbb{IR}^n \times \mathbb{IR}^m \to \mathbb{IR}^n$ satisfying $g(x, w) \in \mathsf{G}([\underline{x}, \overline{x}], [\underline{w}, \overline{w}])$ for all $x \in [\underline{x}, \overline{x}]$, $w \in [\underline{w}, \overline{w}]$, or

$$\underline{\mathsf{G}}([\underline{y}, \overline{y}], [\underline{w}, \overline{w}]) \le g(y, w) \le \overline{\mathsf{G}}([\underline{y}, \overline{y}], [\underline{w}, \overline{w}]).$$

Inclusion functions are easily constructed through composition of intermediate operations [11]. For instance, `immrax` [18] is an efficient toolbox for JAX that automatically constructs inclusion functions for general dynamics. Throughout, we use a natural inclusion function G [11] which `immrax` obtains by algorithmically replacing nodes in a computation tree with their interval extensions [18].

Using G, we define a *lifted embedding system* by composing the interval refinement operator with the inclusion function evaluated on each lower ($[\underline{y}, \overline{y}_{i:\underline{y}}]$) and upper ($[\underline{y}_{i:\overline{y}}, \overline{y}]$) face of the hyperrectangle $[\underline{y}, \overline{y}]$[1], for every $i \in \{1, \ldots, m\}$,

$$\dot{\underline{y}}_i = \underline{\mathsf{G}}_i(\mathcal{I}_\mathcal{H}([\underline{y}, \overline{y}_{i:\underline{y}}]), [\underline{w}, \overline{w}]), \tag{7a}$$

$$\dot{\overline{y}}_i = \overline{\mathsf{G}}_i(\mathcal{I}_\mathcal{H}([\underline{y}_{i:\overline{y}}, \overline{y}]), [\underline{w}, \overline{w}]). \tag{7b}$$

Under some mild regularity assumptions, it follows from [19, Thm. 2] that for any $x_0 \in \{x \in \mathbb{R}^n : \underline{y}_0 \le Hx \le \overline{y}_0\}$, and $t \mapsto \mathbf{w}(t) \in [\underline{\mathbf{w}}(t), \overline{\mathbf{w}}(t)]$, we have

$$x(t) \in \{x \in \mathbb{R}^n : \underline{y}(t) \le Hx \le \overline{y}(t)\},$$

where $t \mapsto x(t)$ is the trajectory of (1) with initial condition x_0 and input map $\mathbf{w}$, and $t \mapsto [\underline{y}(t), \overline{y}(t)]$ is the trajectory of any (H, H^+)-lifted embedding system (7) from initial condition $[\underline{y}_0, \overline{y}_0]$ under input map $[\underline{\mathbf{w}}, \overline{\mathbf{w}}]$.

Remark 1. In [12], the authors tighten bounds by re-expressing dynamics in terms of the lifted variables, which is represented in our lifted system framework through different left inverses H^+ [14, Remark 15]. If $H = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{smallmatrix}\right]$, the choice $H^+ = \left[\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{smallmatrix}\right]$ keeps the first 2 components of the lifted dynamics the same, while $H^+ = \left[\begin{smallmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \end{smallmatrix}\right]$ replaces x_2 with $-x_1 + y_3$. However, we note that the choice of H^+ does not directly result in the term cancellations also proposed in [12]. For the rest of this paper, we use $H^+ = \begin{bmatrix} H_V^{-1} & 0_{n,m-n} \end{bmatrix}$.

[1]For $x, y \in \mathbb{R}^n$, $x_{i:y} \in \mathbb{R}^n$ is the vector *replacing* the ith component of x with that of y: $(x_{i:y})_j = x_j$ for every $j \neq i$ and $(x_{i:y})_i = y_i$.

III. Refinement Strategy

In the previous section, we defined the sampling strategy refinement operator $\mathcal{I}^{\text{SS}}_{\mathcal{H},A}$ but left the parameters $N \in \mathbb{N}$ and $A \in \mathbb{R}^{N\times m}$ in (6) undetermined. There are many suitable A matrices, and importantly each may yield different refined bounds. We therefore wish to characterize choices of these parameters that produce good refinements efficiently.

First, we show that when auxiliary variables are considered individually, the optimal refinement can be parameterized by a single $a \in \mathbb{R}^{m-n}$. Combining these vectors for each auxiliary variable gives a basis for the left null space of H with desirable properties that are made clear in Example 1.

Example 1 (Basis computation)**.** Given a matrix H, the MATLAB code `null(H.')'` uses SVD to compute $L_{\text{M}}(H)$ with rows forming a basis for the left null space of H. Applying L_{M} to two very similar matrices, we see

$$L_{\text{M}}\left(\begin{bmatrix}1&0\\0&1\\1&1\end{bmatrix}\right) = \begin{bmatrix}-0.5774 & -0.5774 & 0.5774\end{bmatrix},$$
$$L_{\text{M}}\left(\begin{bmatrix}1&0\\0&1\\1&1\\1&0.5\end{bmatrix}\right) = \begin{bmatrix}-0.9957 & -0.7071 & 0.4184 & 0.5773\\ 0.0917 & -0.4082 & 0.9082 & -1.0000\end{bmatrix}.$$

By contrast, the bases computed in Algorithm 1, Line 2 are

$$L\left(\begin{bmatrix}1&0\\0&1\\1&1\end{bmatrix}\right) = \begin{bmatrix}-1 & -1 & 1\end{bmatrix}, \quad L\left(\begin{bmatrix}1&0\\0&1\\1&1\\1&0.5\end{bmatrix}\right) = \begin{bmatrix}-1 & -1 & 1 & 0\\ -1 & -0.5 & 0 & 1\end{bmatrix}.$$

Note that with Algorithm 1, the first basis appears as a sub-block of the second (but not with L_{M}). This is the key property that ensures adding auxiliary variables does not worsen refinement (Theorem 1). Furthermore, the right block of our bases is an identity matrix, forcing sparsity (especially when $m-n$ is large). This causes many terms in (6) to cancel entirely, reducing the number of interval operations, each of which contribute to the conservatism of the refinement.

Finally, we generate additional rows for A by combining each *pair* of optimal refinement vectors in L. This gives an N (and induced complexity of (6)) much lower than naive, uniform sampling the left null space of H (Proposition 2).

A. Algorithm 1: Subspace Sampling

First, we parameterize the sample space of A, applying the assumption in (6) that $AH = 0$.

Proposition 1. Let $H \in \mathbb{R}^{m\times n}$ be full rank and $L^\top \in \mathbb{R}^{m\times(m-n)}$ be a basis for the left null space $\mathcal{N}(H^\top)$. If $N \in \mathbb{N}$ and $A \in \mathbb{R}^{N\times m}$ with $AH = 0$, there exists a $\Gamma \in \mathbb{R}^{N\times(m-n)}$ with $A = \Gamma L$.

Proof. Since $AH = 0$, every row A_i, $i \in \{1, 2, \ldots, N\}$ must lie in the left null space of H. By assumption, there exist scalars $\Gamma_{i,1}, \ldots \Gamma_{i,m-n}$ with $A_i = \sum_{j=1}^{m-n} \Gamma_{i,j} L_j$. Defining $\Gamma_i = \begin{bmatrix}\Gamma_{i,1} & \cdots & \Gamma_{i,m-n}\end{bmatrix}$, we have $A_i = \Gamma_i L$. Repeating the above argument for all i and letting $\Gamma = \begin{bmatrix}\Gamma_1 & \cdots & \Gamma_N\end{bmatrix}^\top$ gives the desired result. □

Note that Proposition 1 reduces the problem of sampling a subspace of $\mathbb{R}^m$ to unconstrained sampling of $\mathbb{R}^{m-n}$. Compared to prior work [12], which chose a *preconditioning vector* from $\mathbb{R}^m$ to generate A (without leveraging the subspace $\mathcal{H}$), this simplification reduces the dimension of the search space. For systems with a large base dimension n, this result is especially important, since only the dimensions corresponding to the auxiliary variables need to be sampled.

Algorithm 1 Subspace-based construction of A

Input: Full rank $H \in \mathbb{R}^{m\times n}$, subspace sample size $s \in \mathbb{N}$

1: $\begin{bmatrix}H_V\\H_A\end{bmatrix} \leftarrow H$ ▷ $H_V \in \mathbb{R}^{n\times n}, H_A \in \mathbb{R}^{(m-n)\times n}$
▷ *Construct basis for left null space*
2: $L = [-H_A H_V^{-1}\ I_{m-n}]$ ▷ $LH = 0$
▷ *Sample pairwise subspaces*
3: Allocate $A_2 \in \mathbb{R}^{s\cdot|^{m-n}P_2|\times m}$
4: $W_i \leftarrow \begin{bmatrix}\cos(\frac{i\pi}{s+1}) & \sin(\frac{i\pi}{s+1}) & 0 & \cdots & 0\end{bmatrix}$ for $i \in \{1, 2, \ldots, s\}$ ▷ $W \in \mathbb{R}^{s\times m-n}$
5: **for** $p \in {}^{m-n}P_2$ **do**
6: $\quad \Omega_p \leftarrow$ Apply permutation p to columns of W
7: $\quad$ Append $\Omega_p L$ to A_2
8: **return** $\begin{bmatrix}L\\A_2\end{bmatrix}$

Next, we observe that (6) is invariant under scalings of A.

Lemma 1. Let $H \in \mathbb{R}^{m\times n}$ be full rank, and $AH = 0$. Then, for any interval $[\underline{y}, \overline{y}]$ and $\lambda \in \mathbb{R} \setminus \{0\}$, we have

$$\mathcal{I}^{\text{SS}}_{\mathcal{H},A}([\underline{y}, \overline{y}]) = \mathcal{I}^{\text{SS}}_{\mathcal{H},\lambda A}([\underline{y}, \overline{y}]) \tag{8}$$

This can be quickly seen by examining the form of (6); the multiplication of every $A_{i,k}$ term within the sum is exactly offset by the overall division by $A_{i,j}$.

Lemma 1 further reduces the sample space to just the positive half of a hypersphere in $\mathbb{R}^{m-n}$. We note that if $m - n = 1$, then Lemma 1 guarantees the existence of a canonical refinement $\mathcal{I}^{\text{SS}}_{\mathcal{H},A}$ since there is only one vector in the left null space, up to scalings. Thus, by considering every auxiliary variable individually, we may construct a basis consisting of these canonical refinement vectors for each. This is the core idea of Algorithm 1.

Sampling the full domain of Lemma 1 uniformly requires exponentially many samples [20, Corollary 4.2.11]. However, submanifolds of this hypersphere can be covered with only polynomially many, since the term $|^{m-n}P_k|$ grows polynomially with degree k (Figure 1). Algorithm 1 implements this method up to degree two. Though this could be iterated to higher dimensions, we note that in practice pairwise sampling or even just using the basis points is often sufficient.

Proposition 2. For $A \in \mathbb{R}^{N\times m}$ generated by Algorithm 1, the number of rows N is worst-case upper bounded by $\mathcal{O}(s(m-n)^2)$, and hence the complexity of the refinement operator (6) is $\mathcal{O}(sm^2(m-n)^2)$.

Proof. Algorithm 1 constructs A in two phases. First, it generates a basis L (Line 2) to parameterize the left null space of H as in Proposition 1. By Lemma 1, it suffices consider only constant magnitude linear combinations of these basis vectors. Algorithm 1 considers $s\cdot|^{m-n}P_2|$ such vectors, where $s \in \mathbb{N}$ is fixed (Line 5). By definition, $|^{m-n}P_2| = \frac{(m-n)!}{(m-n-2)!}$, so the size of this second block grows quadratically in $m-n$. Thus, the size N of the overall matrix A grows with the sum

of the sizes of its two blocks: $\mathcal{O}(m-n)+\mathcal{O}(s(m-n)^2) = \mathcal{O}(s(m-n)^2)$.

Finally, note that (6) requires computing N bounds for m interval components, each using m multiplications and additions and one division. Therefore, the final complexity is $\mathcal{O}(m^2N) = \mathcal{O}(sm^2(m-n)^2)$, as claimed. □

Adding base states increases m but keeps $m-n$ constant. This causes the complexity of (6) to increase quadratically, allowing our technique to scale to systems with many base states. When adding auxiliary variables, m increases and n is constant, inducing quartic runtime growth. Both cases are significant improvements over the exponential complexity of a dense sampling of the hemisphere in $\mathbb{R}^{m-n}$ (Figure 1).

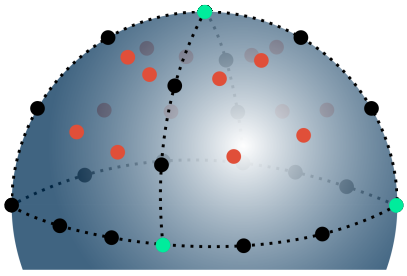

Fig. 1. Points sampled on the surface of a hemisphere. Basis samples (corresponding to rows of L in Algorithm 1) are shown in green, those along angular subspaces are in black (corresponding to rows of A_2 in Algorithm 1), and ones distributed over the entire surface in red. The number of samples needed to densely cover the surface is exponential in dimension.

B. Monotonicity With Respect to Extra Rows in H

The specific basis L in Algorithm 1 is also important. The j-th row L_j corresponds exactly to the canonical vector in the left null space of the matrix $\begin{bmatrix} H_V \\ (H_A)_j \end{bmatrix}$, since $L_jH = -\delta_j^T H_A H_V^{-1} H_V + (H_A)_j = 0$. By isolating each auxiliary variable in its own row, it ensures the corresponding A retains the same rows when new auxiliary variables are added. This property is not shared by other methods for generating such a basis, *e.g.* SVD, which prioritize orthonormality.

Theorem 1. Let $H^0 \in \mathbb{R}^{m_0 \times n}$ be full rank, and $[\underline{y}^0, \overline{y}^0] \in \mathbb{IR}^{m_0}$. For any $h \in \mathbb{R}^{1\times n}$ and $\underline{\gamma}, \overline{\gamma} \in \mathbb{R}$ with $\underline{\gamma} < \overline{\gamma}$, define

$$H^1 = \begin{bmatrix} H^0 \\ h \end{bmatrix}, \quad [\underline{y}^1, \overline{y}^1] = \left[\begin{bmatrix} (\underline{y}^0)^\top & \underline{\gamma} \end{bmatrix}^\top, \begin{bmatrix} (\overline{y}^0)^\top & \overline{\gamma} \end{bmatrix}^\top \right].$$

Let $A^0 \in \mathbb{R}^{N_0 \times m_0}$, $A^1 \in \mathbb{R}^{N_1 \times m_0+1}$ be given by Algorithm 1 for H^0 and H^1, respectively. For $j \in \{1, 2, \ldots, m_0\}$,

$$\mathcal{I}^{\mathrm{SS}}_{\mathcal{H}^1, A^1}([\underline{y}^1, \overline{y}^1])_j \subseteq \mathcal{I}^{\mathrm{SS}}_{\mathcal{H}^0, A^0}([\underline{y}^0, \overline{y}^0])_j. \tag{9}$$

Proof. We prove monotonicity of the lower bound; the upper case is identical. By (6), $\underline{\mathcal{I}}^{\mathrm{SS}}_{\mathcal{H}^1, A^1}([\underline{y}^1, \overline{y}^1])_j \geq \underline{y}^0_j$ always. Thus, if $\underline{\mathcal{I}}^{\mathrm{SS}}_{\mathcal{H}^0, A^0}([\underline{y}^0, \overline{y}^0])_j = \underline{y}^0$, then the proof is trivial.

Else, again by (6), there exists an $i^* \in \{1, 2, \ldots, N_0\}$ with

$$\underline{\mathcal{I}}^{\mathrm{SS}}_{\mathcal{H}^0, A^0}([\underline{y}^0, \overline{y}^0])_j = -\frac{1}{A^0_{i^*,j}} \sum_{k \neq i^*} \min\left\{ A^0_{i^*,k}\underline{y}_k, A^0_{i^*,k}\overline{y}_k \right\},$$

by definition of interval multiplication [11]. Note that the $m_0 - n$ rows of L^0 on Line 2 in Algorithm 1 are linearly independent by construction, and therefore form a basis for the left null space of H^0. Thus, there exist coefficients $\xi_1, \ldots, \xi_{m_0-n}$ with $A^0_{i^*} = \sum_{i=1}^{m_0-n} \xi_i L^0_i$. Since $A^0_{i^*}$ is a row of A^0, these ξ_is are a permuted row of Ω_p, as in Line 6.

Consider the linear combination $A'_{i^*} := \sum_{i=1}^{m_0-n} \xi_i L^1_i$. Since ${}^{m_0-n}P_2 \subset {}^{m_0-n+1}P_2$, A'_{i^*} is a row of A^1, say A^1_κ. By construction, $A^1_{\kappa,q} = A^0_{i^*,q}$ for all $q \in \{1, 2, \ldots, m_0\}$ and (since $L^1_{i,m_0+1} = 0$ for all $i \leq m_0 - n$) $A^1_{\kappa,m_0+1} = 0$. Applying (6) once more,

$$\underline{\mathcal{I}}^{\mathrm{SS}}_{\mathcal{H}^1, A^1}([\underline{y}^1, \overline{y}^1])_j \geq -\frac{1}{A^1_{\kappa,j}} \sum_{k \neq \kappa} \min\left\{ A^1_{\kappa,k}\underline{y}_k, A^1_{\kappa,k}\overline{y}_k \right\} =$$

$$-\frac{1}{A^0_{i^*,j}} \sum_{k \neq i^*} \min\left\{ A^0_{i^*,k}\underline{y}_k, A^0_{i^*,k}\overline{y}_k \right\} = \underline{\mathcal{I}}^{\mathrm{SS}}_{\mathcal{H}^0, A^0}([\underline{y}^0, \overline{y}^0])_j,$$

as desired. □

These results enable us to apply Algorithm 1 to a wide variety of systems. Since each additional variable incurs a reasonable computation cost (Proposition 2) and can only improve bounds (Theorem 1), good bounds can be achieved by simply constructing a lifted system with lots of auxiliary variables, rather than carefully selecting them from domain knowledge (as in previous work [12]).

IV. Examples

We now demonstrate the efficacy of our approach with several examples[2]. We include a classic nonlinear system, a chemical reaction, and a multi-agent platoon. Additionally, we address uncertainty of many types, including unknown initial conditions, parameter values, or persistent disturbances. Our contribution reduces the need to manually tune parameters for each of these domains, successfully applying the same calculations in each. Each of the simulations in this section is implemented in `immrax` [18], an interval arithmetic and reachability analysis module for the Python framework JAX [21] supporting GPU parallelization, Just-in-Time Compilation, and Automatic Differentiation. They were performed on an Arch Linux system with an Intel i5-12600K CPU, NVIDIA RTX 3050 GPU, and 64GB of RAM.

A. Refinement Procedure Comparison

Consider the Van der Pol oscillator:

$$\dot{x}_1 = \mu\left(x_1 - \tfrac{1}{3}x_1^3 - x_2\right), \quad \dot{x}_2 = \tfrac{1}{\mu}x_1, \tag{10}$$

for some $\mu > 0$. Fix a number $\ell \in \mathbb{N}$ of auxiliary variables to introduce, and define the matrix $K \in \mathbb{R}^{\ell \times 2}$ whose ith row is $\begin{bmatrix} \cos(\frac{i*\pi}{\ell+1}) & \sin(\frac{i*\pi}{\ell+1}) \end{bmatrix}$ for $i \in \{1, 2, \ldots, \ell\}$. Since only the *ratios* of the coefficients of each row of A affect the refinement (6), there is only one degree of freedom to choose these coefficients for two dimensional systems. The matrix K produces rows at even intervals in this space. Let

$$H = \begin{bmatrix} I_2 \\ K \end{bmatrix}, \quad H^+ = \begin{bmatrix} I_2 & 0_{2\times\ell}, \end{bmatrix}$$

and consider the (H, H^+)-lifted system of (10). Note that this lifting can be applied automatically to any 2D system.

[2]The source code for these examples is available at `https://github.com/gtfactslab/Gould_LCSS2025`.

Using Refinement $\mathcal{I}^{\mathrm{LP}}_{\mathcal{H}}$

ℓ	Time (sec)	Final Bound Size
2	$5.78 \pm 1.02\text{e}{-1}$	29.41
4	14.38 ± 0.269	3.991
6	$23.55 \pm 2.63\text{e}{-1}$	1.322

Using Refinement $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$

ℓ	Time (sec)	Final Bound Size
2	$2.62\text{e}{-2} \pm 2.98\text{e}{-3}$	29.41
4	$2.13\text{e}{-2} \pm 2.48\text{e}{-3}$	4.780
6	$2.71\text{e}{-2} \pm 7.81\text{e}{-3}$	1.659

TABLE I. Comparison of refinements $\mathcal{I}^{\mathrm{LP}}_{\mathcal{H}}$ and $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ as used in Example 2. Sample size of 10 runs, each numerically integrated from $t = 0$ to $t = 2\pi$ with a timestep of 0.01 using the `tsit5` solver.

Example 2. We generate the lifted embedding system (7) for (10) to over-approximate the reachable set from initial conditions $x_0 \in \left[\begin{bmatrix}0.9 & -0.1\end{bmatrix}^\top, \begin{bmatrix}1.1, 0.1\end{bmatrix}^\top\right]$ for time $t \in [0, 2\pi]$. We compare the refinements $\mathcal{I}^{\mathrm{LP}}_{\mathcal{H}}$ given in (4)-(5) and $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ based on Algorithm 1 (with $s = 10$).

Table I compares the average runtime and final bound size of both refinements for multiple values of ℓ. Note that the bound generated by $\mathcal{I}^{\mathrm{LP}}_{\mathcal{H}}$ is always smaller than that of $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ for the same ℓ. However, $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ scales better and may generate tighter bounds than $\mathcal{I}^{\mathrm{LP}}_{\mathcal{H}}$ in a shorter time with a larger ℓ.

The scaling of $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ is further supported by the computational capabilities of JAX. Computing (6) requires repeated evaluations of the same formula. The JAX function transformation `vmap` can *vectorize* this, computing many such results in one arithmetic operation. Furthermore, the transformation `jit` compiles Python functions into XLA which is executed on a GPU for parallelism.

B. Enzymatic Reaction

Consider the dynamics of an enzymatic reaction from [12]:

$$\dot{x}_{\mathrm{A}} = -k_1 x_{\mathrm{A}} x_{\mathrm{F}} + k_2 x_{\mathrm{F:A}} + k_6 x_{\mathrm{R:A'}}, \tag{11a}$$
$$\dot{x}_{\mathrm{F}} = -k_1 x_{\mathrm{A}} x_{\mathrm{F}} + k_2 x_{\mathrm{F:A}} + k_3 x_{\mathrm{F:A}}, \tag{11b}$$
$$\dot{x}_{\mathrm{F:A}} = k_1 x_{\mathrm{A}} x_{\mathrm{F}} - k_2 x_{\mathrm{F:A}} - k_3 x_{\mathrm{F:A}}, \tag{11c}$$
$$\dot{x}_{\mathrm{A'}} = k_3 x_{\mathrm{F:A}} - k_4 x_{\mathrm{A'}} x_{\mathrm{R}} + k_5 x_{\mathrm{R:A'}}, \tag{11d}$$
$$\dot{x}_{\mathrm{R}} = -k_4 x_{\mathrm{A'}} x_{\mathrm{R}} + k_5 x_{\mathrm{R:A'}} + k_6 x_{\mathrm{R:A'}}, \tag{11e}$$
$$\dot{x}_{\mathrm{R:A'}} = k_4 x_{\mathrm{A'}} x_{\mathrm{R}} - k_5 x_{\mathrm{R:A'}} - k_6 x_{\mathrm{R:A'}}. \tag{11f}$$

We assume the initial concentration of each reagent is given: $x_0 = \begin{bmatrix}34 & 20 & 0 & 0 & 16 & 0\end{bmatrix}^\top$, and that the parameter k is known within an order of magnitude: $k \in [\hat{k}, 10\hat{k}]$, where $\hat{k} = \begin{bmatrix}0.1 & 0.033 & 16 & 5 & 0.5 & 0.3\end{bmatrix}^\top$.

Example 3 ([12, Example 3])**.** Using the lifting matrix

$$K = \begin{bmatrix} -0.48 & -0.14 & -0.62 & -0.48 & 0.24 & -0.24 \\ -0.31 & 0.75 & 0.43 & -0.31 & 0.15 & -0.15 \\ 0 & 0 & 0 & 0 & 0.70 & 0.70 \end{bmatrix}, \quad H = \begin{bmatrix} I_6 \\ K \end{bmatrix},$$

from [12] with the system (11), we compute A as in Algorithm 1 with $s = 10$ and use $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ to compute reachable set approximations on the time interval $t \in [0, 0.04]$. This K is motivated by chemical properties of the system guaranteeing that each auxiliary variable it introduces is invariant. Therefore, we explicitly have $\dot{y}_j = 0$ in (2) for $j \in \{7, 8, 9\}$.

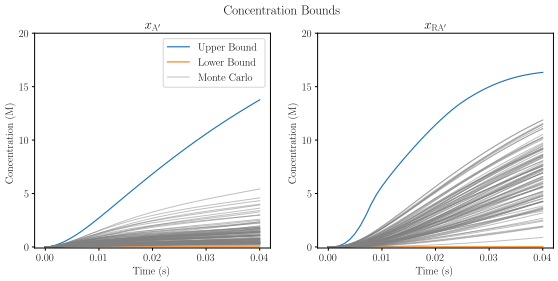

Fig. 2. State bounds for $x_{\mathrm{A'}}$ (left) and $x_{\mathrm{RA'}}$ (right) from Example 3. Gray lines are Monte-Carlo sampled real solutions. The bounds and each sample trajectory were integrated with the `tsit5` solver and a timestep of 10^{-3}.

The bounds on two of the state variables produced by Example 3 are visualized in Figure 2. We recover similar bounds to [12, Figure 5]. Our bounds rely on Algorithm 1 and expressing invariance in the lifted system, rather than the radius based preconditioning and symbolic simplifications performed in that work.

C. Multi-Agent Platoon

Consider the problem of stabilizing a double integrator

$$\begin{aligned} \dot{p}^1_x &= v^1_x, & \dot{p}^1_y &= v^1_y, \\ \dot{v}^1_x &= u_x + w^1_x, & \dot{v}^1_y &= u_y + w^1_y, \end{aligned} \tag{12}$$

subject to an additive input disturbance $w^1_x, w^1_y \in [-0.01, 0.01]$ while avoiding an obstacle. For initial state $p^1_x = 8$, $p^1_y = 7$, $v^1_x = -\sqrt{3}$, and $v^1_y = -1$, we solve this problem by using the Python library `casadi` to generate feedforward inputs $u_{ff} := \{\begin{bmatrix}u_x(t) & u_y(t)\end{bmatrix}^\top\}_{t\in[t_0,t_f]}$ that minimize a quadratic form of position and control. Obstacle avoidance is a hard constraint in this minimization, with an additional 30% padding added to the size of the obstacle.

This system can be easily extended to a platoon of arbitrary length $P \in \mathbb{N}$ by appending states with dynamics

$$\begin{aligned} \dot{p}^i_x &= v^i_x, & \dot{p}^i_y &= v^i_y, \\ \dot{v}^i_d &= k_p \left(p^{i-1}_d - p^i_d - r \frac{v^{i-1}_d}{\|v^{i-1}\|} \right) + k_v (v^{i-1}_d - v^i_d) + w^i_d, \end{aligned} \tag{13}$$

for all $i \in \{2, \ldots, P\}$ and $d \in \{x, y\}$. Here, each additional agent is implementing PD control to track the proceeding one with parameters $k_p = k_v = 5$. To avoid collisions, followers do not track the exact position of the proceeding agent, but aim to follow at a distance of $r = 0.5$. The follower agents start with the same velocity as the leader and offset position $p^i_x = p^1_x + 0.2\sqrt{3}(i-1)$, $p^i_y = p^1_y + 0.2(i-1)$. Finally, the control error of the follower agents is assumed to be similarly bounded: $w^i_x, w^i_y \in [-0.01, 0.01]$.

Example 4. We introduce the lifting matrix

$$L = \begin{bmatrix} {\delta_{4(i-1)+1}}^\top + {\delta_{4(i-1)+3}}^\top \\ {\delta_{4(i-1)+2}}^\top + {\delta_{4(i-1)+4}}^\top \end{bmatrix}, \quad H = \begin{bmatrix} I_{4P} \\ L_1 \\ \vdots \\ L_P \end{bmatrix},$$

to give the (H, H^+) lifted system for (12)-(13). Leveraging the insight from [14, Example 2], it can be advantageous to

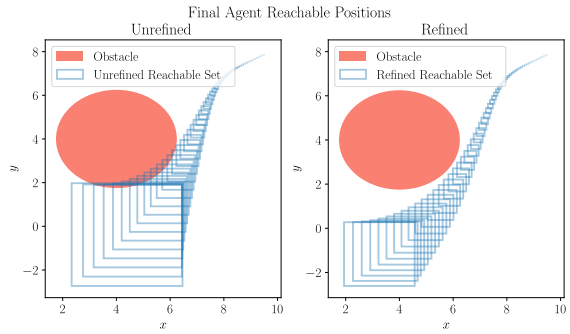

Fig. 3. Interval bounds on the position of the final agent in the platoon of Example 4 ($P = 6$). Bounds produced *without* a refinement operator are shown on the left, and cannot be verified as safe, since they intersect the obstacle. Bounds produced using the refinement operator (6) with Algorithm 1 (shown on the right) are tightened, no longer intersect the obstacle, and verify the safety of the platooned system.

P	Unrefined	Refinement $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$	CORA
3	$1.30\mathrm{e}{-2} \pm 2.1\mathrm{e}{-3}$	$2.92\mathrm{e}{-2} \pm 2.3\mathrm{e}{-3}$	3.11
6	$1.46\mathrm{e}{-2} \pm 1.2\mathrm{e}{-3}$	$1.97 \pm 3.1\mathrm{e}{-2}$	5.56
9	$2.48\mathrm{e}{-2} \pm 3.0\mathrm{e}{-3}$	10.3 ± 0.17	10.57
12	$2.85\mathrm{e}{-2} \pm 2.0\mathrm{e}{-3}$	35.0 ± 0.16	12.85

TABLE II. Scaling of refinement runtime with platoon size. Unrefined interval bounds are compared to the subspace sampling refinement strategy of Algorithm 1 and a zonotope-based method. The refined interval bounds and CORA verify platoon safety; unrefined bounds do not. Sample size of 10 runs, each numerically integrated from $t = 0$ to $t = 3$ with a timestep of 0.01 using Euler integration. The reported times for CORA were recorded by MATLAB's `timeit` command.

introduce halfspaces correlating position and velocity terms. Constructing A as in Algorithm 1 with $s = 10$, we use the lifted embedding system (7) with refinement $\mathcal{I}^{\mathrm{SS}}_{\mathcal{H},A}$ to bound the platoon's reachable sets on the time interval $t \in [0, 3]$.

The reachable set bounds computed in Example 4 are visualized in Figure 3 for $P = 6$. Interval refinement tightens these bounds and allows us to verify that every agent in the platoon will avoid the obstacle.

This example demonstrates the scalability of our method. The platooned system contains $4P$ base states, and $2P$ auxiliary states are added for a total of $6P$ lifted states. Despite the inherent high-dimensionality of multi-agent systems, our method is able to efficiently produce useful bounds on reachable sets. Table II lists the time taken to compute embedding trajectories for both the base and lifted systems across several values of P, characterizing runtime growth in platoon size. We additionally compare to zonotopic bounds produced by the CORA toolbox [3]. Initially, our method verifies safety faster. Due to the many auxiliary variables being used, our runtime eventually grows too quickly, and becomes slower than the zonotopic method.

V. Conclusion

We have presented a novel strategy for parameter conditioning in interval refinement reachability analysis. Our method exhibits efficient runtime growth and monotonic bound sizes in the number of auxiliary variables used, enabling its application to a wide variety of systems without the use of domain knowledge. Interesting areas for future work include further automation of the selection of H. In particular, a promising approach is to apply the automatic differentiability inherent to JAX with the interval refinement toolbox [18] to learn an optimal H through gradient descent.

References

[1] X. Chen and S. Sankaranarayanan, "Reachability analysis for cyber-physical systems: Are we there yet?," in *NASA Formal Methods Symposium*, pp. 109–130, Springer, 2022.

[2] L. Geretti, J. A. D. Sandretto, M. Althoff, L. Benet, P. Collins, M. Forets, S. Mitsch, C. Schilling, J. Tillet, and M. Wetzlinger, "Arch-comp24 category report: Continuous and hybrid systems with nonlinear dynamics," in *Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems* (G. Frehse and M. Althoff, eds.), vol. 103 of *EPiC Series in Computing*, pp. 39–63, EasyChair, 2024.

[3] M. Althoff, "An introduction to cora 2015," in *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, pp. 120–151, EasyChair, December 2015.

[4] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz, "Constrained zonotopes: A new tool for set-based estimation and fault detection," *Automatica*, vol. 69, pp. 126–136, 2016.

[5] T. J. Bird, H. C. Pangborn, N. Jain, and J. P. Koeln, "Hybrid zonotopes: A new set representation for reachability analysis of mixed logical dynamical systems," *Automatica*, vol. 154, p. 111107, 2023.

[6] J. Koeln, T. J. Bird, J. Siefert, J. Ruths, H. C. Pangborn, and N. Jain, "zonolab: A matlab toolbox for set-based control systems analysis using hybrid zonotopes," in *2024 American Control Conference (ACC)*, pp. 2513–2520, IEEE, 2024.

[7] S. M. Harwood and P. I. Barton, "Efficient polyhedral enclosures for the reachable set of nonlinear control systems," *Springer London*, Feb. 2016. Accepted: 2016-07-20T19:25:14Z Publisher: Springer London.

[8] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling, "Juliareach: a toolbox for set-based reachability," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 39–44, 2019.

[9] A. B. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *International workshop on hybrid systems: Computation and control*, pp. 202–214, Springer, 2000.

[10] P. S. Duggirala and M. Viswanathan, "Parsimonious, simulation based verification of linear systems," in *International conference on computer aided verification*, pp. 477–494, Springer, 2016.

[11] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied interval analysis*. Springer, 1st edition. ed., 2001.

[12] K. Shen and J. Scott, "Rapid and Accurate Reachability Analysis for Nonlinear Dynamic Systems by Exploiting Model Redundancy," *Computers & Chemical Engineering*, vol. 106, Aug. 2017.

[13] S. Jafarpour, A. Harapanahalli, and S. Coogan, "Efficient interaction-aware interval analysis of neural network feedback loops," *IEEE Transactions on Automatic Control*, 2024.

[14] A. Harapanahalli and S. Coogan, "Certified Robust Invariant Polytope Training in Neural Controlled ODEs," Aug. 2024. arXiv:2408.01273 [cs, eess, math].

[15] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *56th Conference on Decision and Control (CDC)*, pp. 2242–2253, 2017.

[16] I. Mitchell and C. J. Tomlin, "Level set methods for computation in hybrid systems," in *Hybrid Systems: Computation and Control (HSCC)*, pp. 310–323, 2000.

[17] J. Maidens and M. Arcak, "Reachability Analysis of Nonlinear Systems Using Matrix Measures," *IEEE Transactions on Automatic Control*, vol. 60, pp. 265–270, Jan. 2015. Conference Name: IEEE Transactions on Automatic Control.

[18] A. Harapanahalli, S. Jafarpour, and S. Coogan, "immrax: A Parallelizable and Differentiable Toolbox for Interval Analysis and Mixed Monotone Reachability in JAX," Apr. 2024. arXiv:2401.11608 [cs, eess, math].

[19] J. K. Scott and P. I. Barton, "Bounds on the reachable sets of nonlinear control systems," *Automatica*, vol. 49, no. 1, pp. 93–100, 2013.

[20] R. Vershynin, "High-Dimensional Probability: An Introduction with Applications in Data Science," May 2025.

[21] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.