Reachable Set Computation for Uncertain Linear Systems

Bineet Kumar Ghosh
Department of Computer Science and Engineering
University of Connecticut
bineet.ghosh@uconn.edu

Parasara Sridhar Duggirala
Department of Computer Science and Engineering
University of Connecticut
psd@uconn.edu

***Abstract*—The reasons why computing reachable set for uncertain linear systems is important.**

- **The system model might have errors.**
- **The system has a parameter that varies such as biological systems. For example, the insulin delivery model has several parameters and these parameters are different for each individual.**

I. Superposition Principle on Discrete Systems

• Let, ξ: $\mathbb{R}^n \times \mathbb{R}^+ \rightarrow \mathbb{R}^n$
• $\xi(c,t)$: Denotes the trajectory after time t starting from the set c of a dynamical system $x^+ = A^k x$
• Let x_t denote the set of states reached by the system $x^+ = A^k x$ after time t.

Lemma 1: Given a system dynamics $x^+ = Ax$. For any set of states c of the system, the trajectory starting at c after time t will be $\xi(c,t) = A^t c$.

Proof: The trjectory starts with the set c (at some instant i as per as the universal clock) and after time t it reaches $A^t c$ as dictated by the system dynamics and denoted by $\xi(c,t) = A^t c$. ■

Theorem 2: Given a system dynamics $x^+ = Ax$ and the intial set x represented using *star* $\langle C, V, P\rangle$; then x^+ after time k can be represented using the *star* $\langle C''', V''', P\rangle$, where $C''' = A^k C$, $V''' = A^k V$.

Proof: Base case: Let the initial set be represented as center $C = x$ and basis vectors $V = \{v_1, v_2, \cdots, v_n\}$ *i.e.* the *star* $\langle C, V, P\rangle$. Applying superposition principle:

$$\xi(x + \sum_{i=1}^{n} \alpha_i v_i, 1) = \xi(x,1) + \sum_{i=1}^{n} \alpha_i \big(\xi(x + v_i, 1) - \xi(x,1)\big)$$

By Lemma 1,

$$\xi(x,1) = Ax$$

$$\xi(x + v_i, 1) = A(x + v_i)$$

Therefore,

$$\sum_{i=1}^{n} \alpha_i \big(\xi(x + v_i, 1) - \xi(x,1)\big) = \sum_{i=1}^{n} \alpha_i \cdot Av_i = A \sum_{i=1}^{n} \alpha_i v_i$$

This can be represented by the star $\langle C', V', P\rangle$ where $C' = AC$ and $V' = AV$. Therefore, $x_1 = Ax$ can be represented as $\langle C', V', P\rangle$

Induction Hypothesis: Let, $x_k = A^k x = \xi(x,k)$. And the *star* representation for x_k be $\langle C'', V'', P\rangle$ where $C'' = A^k C$ and $V'' = A^k V$

Induction Step: Let, $x_{k+1} = \xi(x, k+1) = A^{k+1} x$. From Induction Hypothesis, $\xi(x,k)$ is represented using the *star* $\langle C'', V'', P\rangle$ where $C'' = A^k C$ and $V'' = A^k V$
Applying superposition principle:

$$\xi(x, k+1) = \xi\big(\xi(x,k), 1\big) =$$

$$\xi\left(C'' + \sum_{i=1, v_i' \in V''}^{n} \alpha_i v_i', 1\right) = \xi(C'', 1) + \sum_{i=1, v_i' \in V''}^{n} \alpha_i \big(\xi(C'' + v_i', 1) - \xi(C''$$

By Lemma 1,

$$\xi(C'', 1) = AC'' = A^{k+1} C$$

$$\xi(C'' + v_i', 1) = A(C'' + v_i') = A^{k+1}(C + v_i)$$

Therefore,

$$\sum_{i=1, v_i' \in V''}^{n} \alpha_i \big(\xi(C'' + v_i', 1) - \xi(C'', 1)\big) =$$

$$\sum_{i=1}^{n} \alpha_i \cdot Av_i' = \sum_{i=1}^{n} \alpha_i \cdot A \cdot A^k v_i = A^{k+1} \sum_{i=1}^{n} \alpha_i v_i$$

This can be represented by the star $\langle C''', V''', P\rangle$ where $C''' = A^{k+1} C$ and $V''' = A^{k+1} V$. Therefore, $x_{k+1} = A^{k+1} x$ can be represented as $\langle C''', V''', P\rangle$ ■ Note that though

Theorem 1 only proves about dynamics $x^+ = Ax$, but it can be easily extended to $x^+ = Ax + B$ using the same technique and $x_k = A^k x + B^k$.

II. Operation on Matrix $A_{n \times n}$

From *Theorem* 1 we see that calculating exponents of matrix A is the key operation for verification of Linear Discrete Systems. If A is a matrix containing variables then A^k might be depedent on higher degree polynomials of the variables, then safety verification of such systems will require

us to perform non-Linear Programming and we know that non-Linear Programming is NP-Hard. For computational ease, the primary question of our investigation is to find *scenarios* where A^k is only lineraly dependent on the variables and then the safety verification $x_k \cap Unsafe$ can be performed using *bi-linear programming*

Definition 2.1: If $A_{n\times n}$ is a matrix with k variables, it can be represented as $A = N_0 + N_1x_1 + \cdots N_kx_k$. $\forall_i$ $N_i \in \mathbb{R}^{n\times n}$ and N_i represents the coefficent matrix for the variable x_i in A and N_0 represents only the constants of A. Therefore, we define our matrix A as a k tuple of matrices $\langle N_0, N_1, \cdots, N_k \rangle$

With respect to *Definition* 2.1 we need to define the multiplication (exponentiation) of A accordingly, this motivates the next definition $w.r.t$ to the operation on these matrices.

Definition 2.2: Given the represntation of two matrices A and B with k variables as $\langle N_0, N_1, \cdots, N_k \rangle$ and $\langle M_0, M_1, \cdots, M_k \rangle$ $s.t.$
$\forall_{ij}$ $1 \leq i, j \leq k$, $N_i \times M_i = 0$ and $N_i \times M_j) + (M_i \times N_j) = 0$
we define the product of the two matrices as follows:
$\langle N_0, N_1, \cdots, N_k\rangle \otimes \langle M_0, M_1, \cdots, M_k\rangle = \langle L_0, L_1, \cdots, L_k\rangle$
$L_0 = N_0 \times M_0$, $L_i = (N_i \times M_0) + (N_0 \times M_i)$

III. Computational Aspects of the Problem

Recall if a matrix A containins variables then A^k might be depedent on higher degree polynomials of the variables. For the computational ease the primary question of our investigation is to find *scenarios* where A^k is only lineraly dependent on the variables. Findng out such scenarios is a significantly tough problem (Open?).

For example:

$$\begin{bmatrix} 1 & \alpha \\ 0 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & \alpha \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 3 & -\alpha \\ 0 & 4 \end{bmatrix}$$

preserves the lineraity.
This is not only for this particular example, infact this represents a broad class of matrices where linearity is preserved after multiplication. This is a crucial example that lays out the some interesting properties that will serve as a platform for the rest of computation in finding more general classes. If a matrix have the following structure and they are multiplied together they will always preserve linearity and univarite property:

$$\begin{bmatrix} Q_1 & R_1 \\ \mathbf{0} & T_1 \end{bmatrix} \times \begin{bmatrix} Q_2 & R_2 \\ \mathbf{0} & T_2 \end{bmatrix} = \begin{bmatrix} Q_1 \times Q_2 & (Q_1 \times R_2) + (R_1 \times T_2) \\ \mathbf{0} & T_1 \times T_2 \end{bmatrix}$$

If all the variable are confined within R, then the result is univariate linear matrix, also another nice property is the structural closure *i.e. the structure is preserved after the multiplication*. This observation is crucial for the compuational ease. When performing matrix multiplication k times it is hard to deal with k different structures.

In this section we will lay out all the neccessary definitions and in the next section we will discuss an algorithm to compute the such set of matrices $w.r.t$ *Definition* 2.1 and obeying *Definition* 2.2.

Let $[\![i, j]\!] = \{ i, i+1, \cdots, i+j-1\}$

(*Definition* 3.1: *Structure of a matrix*: $\mathfrak{C}_{m\times n} = \{(\Omega_1, R_1), (\Omega_2, R_2), \cdots, (\Omega_k, R_k)\} \subseteq \mathbb{R}^{m\times n}$
$s.t$
1. $\Omega_i = [\![c_1, l]\!] \times [\![c_2, b]\!]$; $1 \leq c_1 \leq m$, $1 \leq c_2 \leq n$, $1 \leq (c_1 + l) \leq m$, $1 \leq (c_2 + b) \leq n$
2. $\bigcup_{i=1}^{k} \Omega_i = [\![1, m]\!] \times [\![1, n]\!]$
3. $\forall_{i\neq j}$ $\Omega_i \cap \Omega_j = \emptyset$;
4. $\forall$ $A \in \mathfrak{C}_{m\times n}$; if $(i, j) \in \Omega_l$ then $A_{i,j} \in R_l$

Let $\mathbb{S}$ denote the set of all possible structures. And, structure of a matrix A is defined by $struct(A)$, $struct(A)$ $\in \mathbb{S}$

Recall from *Definition* 2.2, $\forall_i$ $struct(M_i), struct(N_i) \in \mathbb{S}$

Let the set of matrices satisfying a *Structure* (as defined in *Definition* 3.1) $K \in \mathbb{S}$ be denoted by M_K.

Definition 3.2: *0 Product Structures*: Two *Structures* S_1 and S_2 are said to be *0 Product Structures* iff $\forall_{M\in M_{S_1}, N\in M_{S_2}}$ $M \times N = 0 \wedge N \times M = 0$

Let $S_{p0} = \{(S_i, S_j) \mid S_i$ and S_j are *0 Product Structures*$\}$

As discussed earlier in this section motivated with the given example to enforce computational simplicity it is important to have a structural closure *i.e.* when any matrix of structure S is multiplied with any matrix of structure S the resulting matrix is also of the same structure S.

Next we discuss how to find a subset of matrices that are
1. Given $\langle N_0, N_1, \cdots, N_k \rangle$ and $\langle M_0, M_1, \cdots, M_k \rangle$ (*Definition* 2), we need to ensure $\forall_{i,j}$ $N_i \cdot M_i = 0 \wedge \big(N_i \cdot M_j + N_j \cdot M_i = 0\big)$.
2. Structural Closure.

The second property is actually easy to achieve if we fix a particular structure for computational ease and only deal with those structures. The first point needs a lot of computation so we fix a structure that will ensure point 2 and then try to compute sub-structures satisfying 1. This motivates the next definition.

Definition 3.3 *Block Structure matrix*: Is a special case (subset) of *Structure* (*Definition* 3.1) of $n \times n$ matrices denoted by $\mathfrak{B}_{n\times n} = \{(\Omega_1, R_1), (\Omega_2, R_2)\}$ *s.t*
1. $\Omega_i = [\![c_1, l]\!] \times [\![c_2, b]\!]$; $1 \leq c_1 \leq m$, $1 \leq c_2 \leq n$, $1 \leq (c_1 + l) \leq m$, $1 \leq (c_2 + b) \leq n$
2. $\Omega_2 = ([\![1, m]\!] \times [\![1, n]\!]) - \Omega_1$;
3. $R_2 = \{0\}$;
4. $R_1 = \mathbb{R}$ The coefficients come from R only, right?

$\mathfrak{B}_{n\times n} \subset \mathbb{S}$

Let, $blockStr(c_1, c_2, l, b)$ be a shorthand notation for the block structure $\Big\{\big([\![c_1, l]\!] \times [\![c_2, b]\!], R\big), \big(([\![1, m]\!] \times [\![1, n]\!]) - \Omega_1, \{0\}\big)\Big\}$

$L_M(N) = NM$
$ker(L_M) = \{N | L_M(N) = 0\}$

Definition 3.4 *Mutual Kernel Structure*: Two *Structure* S_1 and S_2 are said to be *Mutual Kernel Structure* iff it satisfies the follwoing condition:

$$\forall_{\substack{X_1 \in M_{S_1} \\ X_2 \in M_{S_2}}} X_1X_2 = 0 \wedge X_2X_1 = 0$$

Let, $kerStr(S_1, S_2)$ be a shorthand notation for representing that *Structures* S_1 and S_2 are *Mutual Kernel Structure* *i.e.* it satisfies:

$$\forall_{\substack{X_1 \in M_{S_1} \\ X_2 \in M_{S_2}}} X_1X_2 = 0 \wedge X_2X_1 = 0$$

We can check if two structures S_1 represented by $blockStr(c_1, c_2, l_0, b_0)$ and S_2 represented by $blockStr(d_1, d_2, l_1, b_1)$ are *Mutual Kernel* of each other by the following formula:
Let, ϕ: *Block Structure* $\times$ *Block Structure* $\rightarrow \{\top, \bot\}$

$$\phi(S_1, S_2) = \begin{cases} \top & ([\![c_2, b_0]\!] \cap [\![d_1, l_1]\!] = \emptyset) \wedge ([\![d_2, b_1]\!] \cap [\![c_1, l_0]\!] = \emptyset) \\ \bot & ([\![c_2, b_0]\!] \cap [\![d_1, l_1]\!] \neq \emptyset) \wedge ([\![d_2, b_1]\!] \cap [\![c_1, l_0]\!] \neq \emptyset) \end{cases}$$

kerStr diag to be added

The set of 0 *Product Structures* (definition 3.2) can be computed from the formula:

$$C = \Bigg\{ \Big(S_1, S_2\Big) \Big| S_1 = blockStr(c_1, c_2, l_0, b_0), \\ 1 \leq c_1 \leq n, 1 \leq c_2 \leq n, 1 \leq l_0 \leq n, 1 \leq b_0 \leq n \\ \bigwedge \\ S_2 = blockStr(d_1, d_2, l_1, b_1), \\ 1 \leq d_1 \leq n, 1 \leq d_2 \leq n, 1 \leq l_1 \leq n, 1 \leq b_1 \leq n \\ \bigwedge \\ \phi(S_1, S_2) \Bigg\}$$

A. The Relation Graph

In the previous section we got the set of 0 *Product Structures*. We define a graph from these 0 *Product Structures*. Let, G_R be the relation graph representing the graph of 0 *Product Structures*.
• *Vertices*: $V = \mathbb{S}$
• *Edges* E: There is an edge from $u \in V$ to $v \in V$ iff $(u, v) \in C$ (edges are bi-directional).

diag or example??

Let, R be the set of complete sub-graphs in $G_R(V, E)$.

A given dynamics A is represented as s a k tuple of matrices $< N_0, N_1, \cdots, N_k >$ (*Definition* 2.1)
Recall we need to ensure $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0 (*Definition* 2.2). So, if this N_1, N_2, $\cdots$, N_k are of the structures that are represented by the vertices of the sub-graphs in R we ensure that each of them when multiplied with any of them gives a 0 matrix, thus ensuring the property $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0 (*Definition* 2.2). Another important property of block matrix is when two matrices of the same structures are multiplied the same structure is preserved. This property is very crucial in terms of computational ease as discussed in section 3. So, when computing A_k, and A is represented as defined in *Definition* 2.1 and the operation as defined in *Definition* 2.2, the structure is always preserved.

B. Revisiting the Operation

Recall *Definition* 2.2; $< N_0, N_1, \cdots, N_k > \otimes < N_0, N_1, \cdots, N_k > = < L_0, L_1, \cdots, L_k >$
$L_0 = N_0 \times M_0$, $L_i = (N_i \times M_0) + (N_0 \times M_i)$
And, $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0

Section 3 discusses on how to pick $N_1, N_2, \cdots, N_k$. We still need to pick N_0 of such a structure *s.t.* it will preserve the structure after the operation $L_i = (N_i \times M_0) + (N_0 \times M_i)$ as well.

The structure of N_0 is computed as:
$\{struct(N_0) | \forall_{Q \in nodes\ of\ a\ sub\ graph} struct(N_0) \cdot Q \in Q \wedge Q \cdot struct(N_0) \in Q\}$

We now discuss an algorithm to find a $Structure\ S$ given a $Block\ Structure\ blockStr(c_1, c_2, l, b)\ s.t.$ $S \times blockStr(c_1, c_2, l, b) = blockStr(c_1, c_2, l, b) \wedge blockStr(c_1, c_2, l, b) \times S = blockStr(c_1, c_2, l, b)$
Do we need to define multiplication of structures?
Note we find a suitable structure for all the N_i/M_i that will preserve the structure for the operation $(N_i \times M_0)$ + $(N_0 \times M_i)$ using $Algorithm$ 1, and then take intersection of all those structures so that the following is satisfied.
$\forall_{Q \in nodes\ of\ a\ sub\ graph} struct(N_0) \cdot Q \in Q \wedge Q \cdot struct(N_0) \in Q\}$
Do we need to define intersection of structures?

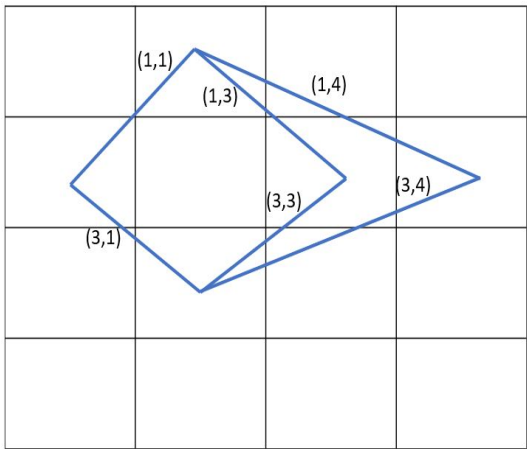
In order to have structural closure, M_0 should be closed under the operation L_0 = $N_0 \cdot M_0$. Next we discuss the algorithm that will ensure structural closure after the operation L_0 = $N_0 \cdot M_0$. Next, we will describe $Algorithm$ 2 that will find out subsets of matrices that will be structurally closed under the operation L_0 = $N_0 \cdot M_0$.

$Definition$ 3.2.1: *Dependency Graph of a Matrix* M: Let I be the set of cells of the matrix that requires a zero in order to be structurally closed under the operation L_0.
Next, we define a graph the following way:
Set of Vertices V: Set of all cells (i, j) of the matrix M.
Edges E: $\forall c \in I$ An edge (u, v) exsits iff either u or v has to be 0 in-order to have a 0 in c.
Example: Let the structure of N_0 be denoted by 0 and R for the sets. So, when L_0 is calculated for the first time, L_0 =N_0 $\cdot$ N_0

$$struct(N_0) = \begin{bmatrix} 0 & R & 0 & 0 \\ R & R & R & R \\ 0 & R & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$struct(L_0) = \begin{bmatrix} R & R & R & R \\ R & R & R & R \\ R & R & R & R \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

I = {(1,1),(1,3),(1,4),(3,1),(3,3),(3,4)}
The dependency graph:



C. The Generalization

In section 3.1 we had discussed about $Relation\ Graph$ $s.t.$ when two matrices are multiplied together they always have the same structure $i.e.$ structurally closed under the operation L_i. In this section we lift that assumption and try to find structures that will preserve the properties as described in $Definition$ 2.2. Note that, we stilll require L_0 to be structurally closed under the operation L_0. Also, L_i should still be a $Block\ Structure\ Matrix$

$Definition$ 3.3.1 *Safe Zone of a block structure* B *w.r.t a relation graph* G: For all neighbors of the node representing the block structure B in the relation graph we do the following:
• For a neighbor $blockStr(c_1, c_2, l_0, b_0)$ of B: c_2 to $c_2 + b_0 - 1$ rows are blocked in B. And, c_1 to $c_1 + l_0 - 1$ columns are blocked in B.
• The above process is done for all the neighbors of the node representing B and the remaining $unblocked$ columns are called *Safe Zone of* B *w.r.t to* G

With the current setup, following is the state of the operations as defined in $Definition$ 2.2:
• L_0 = $N_0 \times M_0$; Struturally closed
• L_i = $(N_i \times M_0)$ + $(N_0 \times M_i)$; The structure calculation of $(N_i \times M_0)$ is required only once. Only the structure of $(N_0 \times M_i)$ changes at every time interval.

When multiplying a structure Q with a $blockStr(c_1, c_2, l_0, b_0)$ only the row values of the columns c_1 and $c_1 + l_0 - 1$ in Q dictates the resulting structure of the matrix.
Example:

$$\begin{bmatrix} R & R & R & R \\ R & 0 & 0 & R \\ R & 0 & 0 & R \\ 0 & R & R & R \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & R & R & 0 \\ 0 & R & R & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & R & R & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & R & R & 0 \end{bmatrix}$$

Notice the problem from the above example. This operation

might not preserve the block structure property. This observation motivates the next definition.

$Definition$ 3.3.2 *Contigous Structured Matrix*: In every column of the matrix there can be no R $s.t$ they are at a distance of greater than 0 cells.
We now only pick the subsets of N_0/M_0 that are *Contigous Structured Matrix*.

D. Revisiting the operation $w.r.t$ to Generalization

$Definition$ 3.4.1 *t-Axle Graph*: When the vertices of the graph can be partioned into t sets $\{X_1, X_2, \cdots X_t\}$ $s.t$,
• Partion X_1 is complete graph.
• $\forall i > 1$ $X_1 \cup X_i$ is a complete bi-partite graph.

Recall $Definition$ 2.2; $< N_0, N_1, \cdots, N_k > \otimes < M_0, M_1, \cdots, M_k >$ = $< L_0, L_1, \cdots, L_k >$
L_0 = $N_0 \times M_0$, L_i=($N_i \times M_0$) + ($N_0 \times M_i$)
And, $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0

With the current setup, L_0 preserves structure, $i.e.$ at every step M_0 will have the same structure of N_0. But now M_i will have different structures.
Recall the Relation Graph $G_R(V, E)$ defined in section 3.1. Now, insted of looking for complete sub-graphs in $G_R(V, E)$ we will look for $t - Axle$ sub-graphs R' in $G_R(V, E)$.
A given dynamics A is represented as s a k tuple of matrices ($Definition$ 2.1)
Recall we need to ensure $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0 ($Definition$ 2.2). So, if this N_1, N_2, $\cdots$, N_k are of the structures that are represented by the vertices of the sub-graphs in R' we ensure that each of them when multiplied with $< M_1, M_2, \cdots, N_k >$ gives a 0 matrix, thus ensuring the property $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0 ($Definition$ 2.2).
When L_i and L_0 is performed in these nodes we get another $< M'_1, M'_2, \cdots, N'_k >$ it goes to another partition of the subgraph. And thus, it ensures $\forall_{i,j}$ ($N_i \times M_j$) + ($M_i \times N_j$) = 0 ($Definition$ 2.2). As it is a $t - Axle$ Graph, this property will be satisfied upto atleast t steps.