Kaa: A Python Implementation of Reachable Set Computation Using Bernstein Polynomials

Edward Kim and Parasara Sridhar Duggirala
Department of Computer Science
UNC- Chapel Hill
Introduction

• Reachable set computation is one of the many important tools available for the verification of dynamical and hybrid systems.

• One of the simpler and easier-to-understand reachable set computation algorithms for polynomial discrete dynamical systems utilizes Bernstein polynomials and parallelotope bundles.

• Tomasso Dreossi, Thao Dang and Carla Piazza implemented a tool called Sapo in C++ which leverages parallelotope bundles and the properties of Bernstein polynomials.

• Kaa is a reimplementation of Sapo using robust Python libraries. The result is a compact implementation with only around ~650 lines of code.
• The state of a system, denoted as $x$, lies in a domain $D \subseteq \mathbb{R}^n$. A discrete-time polynomial nonlinear system is denoted as $x^+ = f(x)$.

• The trajectory is denoted as $\xi(x_0)$, is the sequence $x_0, x_1, \ldots$ where $x_{i+1} = f(x_i)$.

• Given an initial set $\Theta$, the reachable set at time $k$, denoted as $\Theta_k = \{ \xi(x_0, k) \mid x_0 \in \Theta \}$ where $\xi(x_0, k) = x_k$. 
Parallelootope Bundles

- A parallelootope $P$ is a set of states in $\mathbb{R}^n$ denoted as $\langle \Lambda, c \rangle$ where $\Lambda \in \mathbb{R}^{2n \times n}$ and $c \in \mathbb{R}^{2n}$, $\Lambda_{i+n} = -\Lambda_i$ and $i \in \{1, \ldots, n\}$ such that:

  $$x \in P \text{ if and only if } \Lambda x \leq c.$$ 

- $\Lambda$ is called the *direction matrix* where $\Lambda_i$ denotes the $i^{th}$ row of $\Lambda$. The vector $c$ is called the *offset vector* where $c_i$ is the $i^{th}$ element of the vector.

- A parallelootope bundle $Q$ is a set of parallelotopes $\{P_1, \ldots, P_m\}$ where $Q = \cap_{i=1}^m P_i$. Note that any polytope initial set can be expressed as a parallelootope bundle.
Parallelotope Bundles

Figure 1 from Dreossi et. al: Parallelotope Bundles for Polynomial Reachability (2016)
Bernstein Polynomials

- Given two multi-indices $i$ and $d$ of size $n$, where $i \leq d$, the Bernstein polynomial of degree $d$ and index $i$ is

$$\mathcal{B}_{i,d} = \beta_{i_1,d_1}(x_1) \beta_{i_2,d_2}(x_2) \cdots \beta_{i_n,d_n}(x_n)$$

$$\beta_{i_m,d_m}(x_m) = \binom{d_m}{i_m} x_m^{i_m} (1 - x_m)^{d_m - i_m}$$

- Any polynomial function can be expressed in the Bernstein basis.
Bernstein Polynomials

• The corresponding *Bernstein Coefficients* can be explicitly calculated for multi-index \(i\) and polynomial degree \(d\):

\[
b_{i,d} = \sum_{j \leq i} \prod_{r} \binom{i_r}{j_r} \binom{d_r}{j_r} a_j
\]

• The upper and lower bounds of polynomial \(h(x_1, \ldots, x_n)\) over unit box \([0,1]^n\) are bounded by the Bernstein coefficients:

\[
\min_{i \in I} \{b_i\} \leq \inf_{x \in [0,1]^n} h(x) \leq \sup_{x \in [0,1]^n} h(x) \leq \max_{i \in I} \{b_i\}.
\]
• A parallelootope $P$ can also be represented as an affine transformation $T_p$ from $[0,1]^n$ to $P$.

• Therefore, upper bounds on the supremum of a function $h$ over $P$ is equivalent to upper bound of $h \circ T_p$ over $[0,1]^n$.

• We denote the procedures for calculating such upper and lower bounds for a polynomial $h$ over some parallelootope $P$ as $\text{BernsteinUpper}(h, P)$ and $\text{BernsteinLower}(h, P)$ respectively.
Reachable Set Comp.

- Given parallelootope bundle $Q = \{P_1, P_2, \ldots, P_m\}$ and a discrete dynamical system $x^+ = f(x)$, we wish to compute an over-approximation of the image $f(Q)$ as a new bundle $Q' = \{P'_1, P'_2, \ldots, P'_m\}$.

- We ensure that direction matrix $\Lambda_{-i}$ of $P'_i$ is same as $P_i$ and the computation is required only to compute the offsets of the directions according to the following non-linear optimization problems:

$$
c_{j,i} = \max_{x \in P_i} \Lambda_{j,i} \cdot f(x)
$$

$$
c_{j+n,i} = \max_{x \in P_i} - \Lambda_{j,i} \cdot f(x)
$$

- Here, $c_{j,i}$ is the $j^{th}$ offset of parallelootope $P_i$. Similarly, $\Lambda_{j,i}$ is the $j^{th}$ row of the directions matrix for $P_i$. 
Reachable Set Comp.

- We can invoke \( \text{BernsteinUpper}(h, P) \) and \( \text{BernsteinLower}(h, P) \) to update the offsets according to the solutions found over all parallelotopes in the bundle \( Q = \{P_1, P_2, \ldots, P_m\} \):

\[
c_{j,i} = \min_{l=1}^{m} \left\{ \text{BernsteinUpper}(\Lambda_{j,i} \cdot f(x), P_l) \right\} \quad \text{if } j \leq n.
\]

\[
c_{j+n,i} = \max_{l=1}^{m} \left\{ \text{BernsteinLower}(\Lambda_{j,i} \cdot f(x), P_l) \right\} \quad \text{otherwise.}
\]

- We iterate this over a certain number of time steps to produce the reachable set.
Sapo Drawbacks


• Current implementation is verbose. The main core of algorithm takes over a thousand lines of C++ code.

• It does not have native plotting functionality. Sapo generates MATLAB code which must be separately run through either MATLAB or Octave. Simultaneous visualization is clunky at best.

• Suffers from little to no documentation. The curious reader must delve into previously published papers to find an explanation of the inner workings.

• Consequently, it becomes difficult to accommodate experimentation.
Motivations for Kaa

- Python is known for its powerful, well-tested symbolic and matrix-computation libraries.

- *NumPy* libraries are popular matrix-computation libraries which allow higher-level manipulation of matrices. This gives us an avenue of overcoming the verbosity and the possibility of memory leaks inherent in implementing identical features in C++.

- The library of *SymPy* has powerful symbolic manipulation tools which allow us to comfortably perform many sensitive symbolic substitutions into polynomials.

- *Matplotlib* library has intuitive plotting facilities that we integrate into our tool for visualizing the reachable set. In particular, *Matplotlib* facilitates the ability to visualize several reachable sets simultaneously.
• We offer a Jupyter Notebook to rapidly introduce the interested reader to the techniques and tools we offer through Kaa.

• Jupyter notebooks are simple to create and well-known for their straightforward user interface.

• By leveraging the Matplotlib library for visualizing the reachable set, we were able to design an engaging interactive tutorial and experimentation platform for visualizing reachable sets of non-linear systems.

• We document the code extensively and offer resources for learning the internals of Kaa.
Results: SIR Model

- The SIR epidemic model is a 3-dimensional dynamical system governed by the following dynamics:

\[
\begin{align*}
s_{k+1} &= s_k - (\beta s_k i_k)\Delta \\
i_{k+1} &= i_k + (\beta s_k i_k - \gamma i_k)\Delta \\
r_{k+1} &= r_k + (\gamma i_k)\Delta
\end{align*}
\]

\[
\beta = 0.34, \gamma = 0.05, \Delta = 0.5
\]

<table>
<thead>
<tr>
<th>Time Steps</th>
<th>Kaa (offu)</th>
<th>Sapo (offu)</th>
<th>Kaa (offl)</th>
<th>Kaa (offl)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.470716</td>
<td>0.470716</td>
<td>-0.435191</td>
<td>-0.43519</td>
</tr>
<tr>
<td>51</td>
<td>0.475839</td>
<td>0.475839</td>
<td>-0.439599</td>
<td>-0.439599</td>
</tr>
<tr>
<td>52</td>
<td>0.480906</td>
<td>0.480906</td>
<td>-0.443945</td>
<td>-0.443945</td>
</tr>
<tr>
<td>53</td>
<td>0.485915</td>
<td>0.485915</td>
<td>-0.448227</td>
<td>-0.448227</td>
</tr>
<tr>
<td>54</td>
<td>0.490862</td>
<td>0.490862</td>
<td>-0.452443</td>
<td>-0.452443</td>
</tr>
<tr>
<td>55</td>
<td>0.495747</td>
<td>0.495747</td>
<td>-0.456591</td>
<td>-0.456591</td>
</tr>
<tr>
<td>56</td>
<td>0.500566</td>
<td>0.500566</td>
<td>-0.460669</td>
<td>-0.460669</td>
</tr>
<tr>
<td>57</td>
<td>0.505317</td>
<td>0.505317</td>
<td>-0.464675</td>
<td>-0.464675</td>
</tr>
<tr>
<td>58</td>
<td>0.509999</td>
<td>0.509999</td>
<td>-0.4686075</td>
<td>-0.468608</td>
</tr>
<tr>
<td>59</td>
<td>0.514610</td>
<td>0.514610</td>
<td>-0.472465</td>
<td>-0.472465</td>
</tr>
<tr>
<td>60</td>
<td>0.519147</td>
<td>0.519147</td>
<td>-0.476246</td>
<td>-0.476246</td>
</tr>
</tbody>
</table>
Results: SIR Model

Kaa

Sapo
Results: SIR Model

Kaa

Sapo
The Rossler model is another 3-dimensional system governed under the dynamics:

\[
\begin{align*}
x_{k+1} &= x_k + -(y - z) \Delta \\
y_{k+1} &= y_k + (x_k + ay_k) \Delta \\
z_{k+1} &= z_k + (b + z_k(x_k - c)) \Delta
\end{align*}
\]

\[a = 0.1, \quad b = 0.1, \quad c = 14\]
\[\Delta = 0.025\]

<table>
<thead>
<tr>
<th>Time Steps</th>
<th>Kaa (offu)</th>
<th>Sapo (offu)</th>
<th>Kaa (offl)</th>
<th>Kaa (offl)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1.95908</td>
<td>1.96043</td>
<td>-1.9209</td>
<td>-1.9193</td>
</tr>
<tr>
<td>51</td>
<td>1.83552</td>
<td>1.83688</td>
<td>-1.7963</td>
<td>-1.7947</td>
</tr>
<tr>
<td>52</td>
<td>1.71044</td>
<td>1.71181</td>
<td>-1.67016</td>
<td>-1.6685</td>
</tr>
<tr>
<td>53</td>
<td>1.58392</td>
<td>1.58531</td>
<td>-1.54255</td>
<td>-1.5409</td>
</tr>
<tr>
<td>54</td>
<td>1.45604</td>
<td>1.45744</td>
<td>-1.41355</td>
<td>-1.4119</td>
</tr>
<tr>
<td>55</td>
<td>1.32687</td>
<td>1.32829</td>
<td>-1.28324</td>
<td>-1.2815</td>
</tr>
<tr>
<td>56</td>
<td>1.19649</td>
<td>1.19792</td>
<td>-1.15168</td>
<td>-1.1500</td>
</tr>
<tr>
<td>57</td>
<td>1.06499</td>
<td>1.06642</td>
<td>-1.01896</td>
<td>-1.0172</td>
</tr>
<tr>
<td>58</td>
<td>0.932432</td>
<td>0.933877</td>
<td>-0.885157</td>
<td>-0.88339</td>
</tr>
<tr>
<td>59</td>
<td>0.798905</td>
<td>0.800358</td>
<td>-0.75035</td>
<td>-0.74857</td>
</tr>
<tr>
<td>60</td>
<td>0.664489</td>
<td>0.665949</td>
<td>-0.614619</td>
<td>-0.61282</td>
</tr>
</tbody>
</table>

Upper and lower offsets for variable y
Results: Rossler Model

Kaa

Reachable Set for x

\[ t: \text{time steps} \]

Sapo

\[ X \]

\[ T \]
Results: Rossler Model

Kaa

Sapo
Results: Quadcopter Model

Kaa

Sapo
Performance Drawbacks

• While the current implementation in Python is very intuitive and concise, it incurs severe performance penalties. We believe this is due to some extraneous library calls in the core loop of the reachable set computation.

• An immediate next step is to deploy extensive profiling to find performance bottlenecks and subsequently improve on them.

<table>
<thead>
<tr>
<th>Model</th>
<th>Kaa</th>
<th>SAPO (C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIR</td>
<td>11.41 sec</td>
<td>0.16 sec</td>
</tr>
<tr>
<td>Rossler</td>
<td>41.92 sec</td>
<td>1.17 sec</td>
</tr>
<tr>
<td>Quadcopter</td>
<td>78.21 sec</td>
<td>11.98 sec</td>
</tr>
<tr>
<td>Lotka-Volterra</td>
<td>18 min 95.05 sec</td>
<td>57.48 sec</td>
</tr>
<tr>
<td>Phosphoraley</td>
<td>103.81 sec</td>
<td>24.86 sec</td>
</tr>
</tbody>
</table>
Conclusions

• We present Kaa, a Python implementation of reachable set computation of nonlinear systems which is focused towards accessibility and pedagogical use.

• We include Juypter Notebooks and documentation through: https://github.com/Tarheel-Formal-Methods/kaa

• While we do incur performance drawbacks from selecting Python for implementing this algorithm, we believe that it aids in fast prototyping and enables students to easily build on top of the library.

• Immediate future work includes improving on the running time and creating a more streamlined format for defining models and visualizing reachable sets.