`Posto`

About The Tool

With the increasing autonomous capabilities of cyber-physical systems, the complexity of their models also increases significantly, thus continually posing challenges to existing formal methods for safety verification. In contrast to model checking, monitoring emerges as an effective lightweight, yet practical verification technique capable of delivering results of practical importance with better scalability. Monitoring involves analyzing logs from an actual system to determine whether a specification (such as a safety property) is violated. Although current monitoring techniques work well in some areas, it has largely been unable to cope with the growing complexity of the models. Monitoring techniques, such as those using reachability methods, may fail to produce results when dealing with complex models like Deep Neural Networks (DNNs). We propose here a novel statistical approach for monitoring that is able to generate results with high probabilistic guarantees.

`Posto` is a Python-based prototype tool that implements the proposed statistical monitoring technique, enabling an effective monitoring of complex systems, including non-linear systems with DNN-based components, while providing results with high probabilistic guarantees.

`Posto` provides three main command-line operations:

1. `behavior` – draw multiple random trajectories and visualise system evolution under uncertainty.
2. `generateLog` – simulate one trajectory, probabilistically sample it, and save it as a .lg log for later analysis.
3. `checkSafety` – verify whether logged trajectories satisfy user-defined safety constraints.

`Posto` supports three model types:

- `Equation mode` – update equations supplied through a `JSON` model.
- `ANN mode` – system dynamics represented by a trained `.h5` neural network model.
- `Development mode (dev)` – supply a custom Python `getNextState` function without modifying core files.

Installation

The tool can be used in one of two ways: **(1) via a local installation** or **(2) by running it in a VirtualBox environment** using the provided OVA file. The detailed steps for each option are outlined below.

1. Local Installation (Recommended)

Dependencies

- `Python 3.9.x`
  - To install this on Ubuntu, one can follow the following steps (note: this step requires the user to have `sudo` privileges):

```
sudo apt update
sudo apt install python3.9 python3.9-venv python3.9-dev -y
```

- `NumPy`

  ```
  pip install numpy
  ```

- `SciPy`

  ```
  pip install scipy
  ```

- `mpmath`

  ```
  pip install mpmath
  ```

- `mpl_toolkits`

  ```
  pip install matplotlib
  ```

- `tqdm`

  ```
  pip install tqdm
  ```

- `TensorFlow` (required for ANN mode)

  ```
  pip install tensorflow
  ```

- `docopt` (for command-line argument parsing)

  ```
  pip install docopt
  ```

Verify Installation (optional)

- To verify if the above dependencies are correctly installed, one can run the following:

  ```
  python -c "import numpy, scipy, mpmath, tqdm, mpl_toolkits, tensorflow, docopt;
  print('All dependencies installed successfully')"
  ```

- If all the dependencies are correctly installed, the above command should run without any error, and display `All dependencies installed successfully` in the terminal.

Downloading the tool

1. Once the dependencies are installed, download the repository to your desired location `/path/to/Posto`
2. Once the repository is downloaded, the user needs to set the variable `POSTO_ROOT_DIR=` to `/path/to/Posto`. To do so, we recommend adding this to `bashrc` (see **Step 2.1**). For users who do not wish to add it to their `bashrc`, can set the variable each time they open the terminal session to run the

tool (see **Step 2.2**). Users choosing step 2.2 are gently reminded to perform this step every time they intend to run the tool.

1. ***[Recommended]*** Once the repository is downloaded, please open `~/.bashrc`, and add the line `export POSTO_ROOT_DIR=/path/to/Posto`, mentioned in the following steps:

    1.
       ```
       vi ~/.baschrc
       ```
    2. Once `.bashrc` is opened, please add the location, where the tool was downloaded, to a path variable `POSTO_ROOT_DIR` (This step is crucial to run the tool):

        1.
           ```
           export POSTO_ROOT_DIR=/path/to/Posto
           ```
2. *[Alternate Approach]* Run this command every time a new terminal session is opened to run the tool:

    1.
       ```
       export POSTO_ROOT_DIR=/path/to/Posto
       ```

2. Using Virtual Box

This artifact is distributed as a pre-configured VirtualBox virtual machine to ensure full reproducibility of the experimental results reported in the paper.

1. Install Oracle VirtualBox (version 7.0 or later) from the official website: https://www.virtualbox.org/wiki/Downloads Please ensure that the **VirtualBox Extension Pack** corresponding to the same version is also installed.
2. Download the Posto zip from the following link and unzip the contents:

   https://alabama.box.com/s/6gn0u0anx0wm8tavhff21qy7clrjgs5b
3. Open **VirtualBox Manager**
4. Select **File** → **Import Appliance**
5. Choose the downloaded `.ova` file from the unzipped contents
6. Click **Next**, then **Import**
7. Start the imported virtual machine

No additional configuration or installation is required.

Posto Location inside the Virtual Machine

After logging into the virtual machine, the Posto tool is located at:

```
~/Desktop/Posto
```

To access it, open a terminal and run:

```
cd ~/Desktop/Posto
```

From this directory, all commands described in the paper and appendices (including artifact evaluation scripts) can be executed directly.

Recreating Results

The results to be reproduced are described in the draft. Detailed instructions for recreating these results are provided in **Appendices A and B** (originally proposed in this paper)

Jet Model

The main results of the Jet Model study are shown in **Figures A.2 and A.3** in the above draft.

Once the tool is downloaded and properly set up, these experimental results can be reproduced using the `artEval.py` script. Detailed steps are provided in **Appendix A**.

For example, to recreate the result shown in **Figure A.2a** (and similarly **Figures A.2b, A.2c, ..., A.3c, and A.3d**), execute the following command:

```
python artEval.py --fig=A2a
```

Van der Pol Oscillator

The main results of the Van der Pol Oscillator study are shown in **Figure A.4** in the above draft.

The experimental results for the Van der Pol Oscillator can be reproduced in a manner similar to the Jet Model case study. Detailed steps are provided in **Appendix A**.

For example, to recreate the result shown in **Figure A.4a**, execute the following command:

```
python artEval.py --fig=A4a
```

Mountain Car

We also present additional experiments using a DNN-based controller for the Mountain Car benchmark (details in **Appendix B**).

```
python artEvalNN.py --fig=B6a
python artEvalNN.py --fig=B6b
python artEvalNN.py --fig=B6c
python artEvalNN.py --fig=B6d
```

Other Usage: Command-Line

All operations use:

```
python posto.py <operation> [arguments]
```

Argument	Required In	Description
`--log=<directory or logfile>`	`behavior`, `generateLog`, `checkSafety`	**Behavior:** path to a **directory** where plots will be saved; an `img/` folder is created inside it. **GenerateLog / CheckSafety:** path to the **.lg logfile** to write or read; plots are saved in an `img/` folder next to the logfile.
`--init=<initialSet>`	`behavior`, `generateLog`	Initial set for state sampling, e.g., `"[0.8,1],[0.8,1]"`. One `[lo, hi]` pair per dimension.
`--timestamp=<T>`	`behavior`, `generateLog`	Time horizon for the simulation (integer ≥ 0).
`--mode=<mode>`	All commands	Specifies model type: • `equation` — load system from a JSON equation model • `ann` — load system from a `.h5` neural network model
`--model_path=<model_path>`	All commands	Path to model file. Use `.json` for equation mode and `.h5` for ann mode.
`--prob=<prob>`	`generateLog`	Logging probability per step during log generation (float ≥ 0).
`--dtlog=<dtlog>`	`generateLog`	Time interval between logged entries when generating a log (float ≥ 0).
`--states=<states>`	Optional in equation mode; required in ann mode	Comma-separated list of state variable names. Needed for mapping ANN inputs/outputs.
`--constraints=<constraints>`	Required in `checkSafety` for ann mode; optional otherwise	Safety constraint specification (JSON file or inline list).

1. Behavior Mode

Generate random trajectories and visualise projections.

```
posto.py behavior \
    --log=<directory> \
    --init=<initialSet> \
    --timestamp=<T> \
    --mode=<mode> \
    --model_path=<model_path> \
    [--states=<states>]
```

2. Generate Log

Simulate a single trajectory, apply probabilistic sampling, and store it in a .lg file.

```
posto.py generateLog \
    --log=<logfile> \
    --init=<initialSet> \
    --timestamp=<T> \
    --mode=<mode> \
    --model_path=<model_path> \
    --prob=<prob> \
    --dtlog=<dtlog> \
    [--states=<states>]
```

3. Check Safety

Evaluate whether logged trajectories satisfy constraints.

```
posto.py checkSafety \
    --log=<logfile> \
    --mode=<mode> \
    --model_path=<model_path> \
    [--states=<states>] \
    [--constraints=<constraints>]
```

Other Usage: Development Mode

Custom next-state function without modifying core `Posto` code.

Example:

```python
from System import System

def my_getNextState(state):
    x, y = state
    x_next = x + 0.1 * (y - x)
    y_next = y + 0.1 * (x - y)
    return (x_next, y_next)

sys = System(
    log_path="/path/to/output.lg",
    states=["x", "y"],
    constraints=[(0, "le", 1)]
)

sys.getNextState = my_getNextState

sys.behaviour([[0, 1], [0, 1]], T=100)
sys.generateLog([[0, 1], [0, 1]], T=100, prob=0.5, dtlog=0.1)
sys.checkSafety()
```

Run the above using the command:

```
python dev/Model.py
```

Run the above using the command:

```
python dev/Model.py
```

Required Packages

- `numpy`
- `scipy`
- `mpmath`
- `matplotlib`
- `docopt`
- `tensorflow`
- `tqdm`