TEAL: synThesizing Efficiently monitorAble mtL

Introduction

TEAL is a Python-based tool for synthesizing formulas in Metric Temporal Logic (MTL) for efficient Runtime monitoring. To synthesize MTL formulas, it relies on solving constraint satisfaction problems using the SMT solver Z3. *TEAL* is written in Python 3.9 and has been tested to work on Linux machines.

Installation

Clone the repository and switch to the root directory using the following:

```
git clone https://github.com/ritamraha/Teal.git
cd TEAL/
```

Now, set up a Python virtual environment for running *TEAL*:

```
virtualenv -p python3 venv
source venv/bin/activate
```

Finally, install the required Python packages using the following:

```
python -m pip install --upgrade pip
pip install -r requirements.txt
```

TEAL is now ready to use.

Running

One can run *TEAL* by simply running `python3 learn_mtl.py`. By default, this will run *TEAL* on `example.signal` with a future-reach bound of 2.

Parameters

There are a variety of arguments that one can use to run *TEAL*, as listed below:

- `-i <input_sample>`: For specifying the input sample; default is `example.signal`.
- `-f <fr_bound>`: For specifying the future-reach bound of the prospective formula; default is 2.
- `-t <timeout>`: For specifying the timeout; default is 600 sec.
- `-o <outputcsv>`: For specifying the CSV file with the output results; default is `results.csv`.
- `-m`: For specifying whether the prospective formula should be *globally* separating or only separating; default is globally separating.

Input sample format

The input sample file consists of three parts separated using `---`. The first part contains the list of positive signals, the second part the negative signals, and the third part the end time point of the signals. The signals in the sample file are piecewise-constant signals. Each signal is represented as a sequence, separated using `;`, of a timepoint and a letter that holds at that timepoint, separated using `:`. Each letter represents the truth value of atomic propositions. An example of a trace is `0.0:1,0;1.2:1,1` which consists of two timepoints 0.0 and 1.2. In the first timepoint only `p` holds, while in the second timepoint both `p,q` hold.

```
0.0:1,1;1.0:0,1;2.5:1,1;2.8:1,1
0.0:0,0;0.2:0,0;1.7:0,1;2.0:1,1
0.0:0,1;2.0:1,0;2.8:0,1;2.9:0,0
0.0:1,0;2.8:1,1;3.0:0,0;3.6:1,0
0.0:1,0;2.1:1,0;3.4:0,0;3.5:0,0
---
0.0:0,0;1.9:0,1;2.3:0,1;3.9:0,0
0.0:0,0;1.2:1,1;1.4:0,0;1.8:0,1
0.0:0,0;2.6:0,0;2.9:0,1;3.7:0,0
0.0:0,1;0.4:0,1;2.9:1,0;3.3:0,1
0.0:1,1;0.1:0,1;2.5:0,0;3.1:1,0
---
4
```

The input file must use the extension `.signal`.

Running using script (on benchmarks)

One can run *TEAL* on the benchmarks used for the research questions in the paper using a convenience script. For this, run the following command `python3 rq-scripts.py` which allows the following parameters:

- `-r <rq_num>`: For specifying experiments from which research question should be run; default is 1.
- `-a`: For specifying whether all benchmarks from the research question should be run; by default, it only runs on a subset.
- `-t <timeout>`: For specifying the timeout for each run of the experiment; default is 600 sec.

The above script produces a CSV file with the results of the experiments for the specified research question.

Resource Considerations

While it is possible to run all the benchmarks using the provided script, please be aware that this task demands a substantial amount of resources both in terms of hardware and time. The benchmark suite contains 144 samples for each of the research questions, and it was executed in parallel (20 runs in each iteration) with a timeout set to 5400 seconds (1.5 hours). The computations were performed on a Computer Grid (using a slurm script `grid-batch.sh`) consisting of 3 Spyder nodes each with an AMD EPYC 7702 64-Core Processor, clock speed of 2.0 GHz using up to 10GB RAM for each run.

Running the complete benchmark suite involves executing the TEAL tool 720 times (once for RQ1 and RQ3, and twice for RQ2 for each sample). When run sequentially, this comprehensive set of benchmarks can potentially consume more than 150 CPU hours.

To expedite the benchmarking process, we recommend running the tool on the provided benchmark subset chosen to keep the runtime within 8 hours. The subset comprises samples of the smallest size for each formula, configured to run with three different future-reach bounds. These results are representative of the outcomes obtained from the full benchmark suite, making them a more efficient choice for most use cases.

How to Interpret the Results

If the convenience script `rq-scripts.py` is used, then the results will be compiled in a CSV file named according to the research question. The CSV file will contain the following columns:

- `file_name`: Name of the input sample file, which contains the formula number with which the sample was generated
- `Fr bound`: The future-reach bound used for the run
- `Number of examples`: Number of total signals in the input sample, adding up both positive and negative signals
- `Example length`: Number of observation timepoints in a signal
- `Formula`: The synthesized formula
- `Formula Size`: The size of the synthesized formula
- `Correct?`: Whether the verification check of the soundness of the formula passes
- `Total Time`: Total Running time of the run
- `Timeout`: The timeout chosen for the run

Note that the samples are generated synthetically using a random generation method from certain ground-truth formulas. As a result, for a run, it is possible that the output formula is simpler (i.e., smaller in size) than the ground-truth formula. However, the output formula should never be more complex (i.e., larger in size) than the ground-truth formula due to the minimality guarantee of *TEAL*.

Also, if the future-reach bound used in a run is less than the future-reach of the ground-truth formula, then *TEAL* might not return any prospective formula (as is seen often in RQ2). This is because, the prospective formula with a small future-reach might be quite large in size, resulting in timeout, or such a prospective formula might not even exist.