FastForward

This is the FastForward tool, which implements the semi-decision procedure for Petri net reachability described in the paper *Directed Reachability for Infinite-State Systems*.

The artifact was tested on the TACAS21 Artifact Evaluation VM, available under https://zenodo.org/record/4041464, using VirtualBox on a machine running Windows 10 with 16GB of physical memory and an Intel Core i7 3770 CPU @ 3.4GHz.

IMPORTANT NOTE

Our approach is parameterized by the chosen heuristic. In the paper, we present three such heuristics:

- state equation over the rationals (D_Q)
- state equation over the integers (D_Z)
- continuous reachability (D_Q>=0).

The two state-equation heuristics are implemented via the mathematical optimization tool *Gurobi*. Due to licensing issues, we **cannot include** it here. **We regret that without Gurobi, the two state-equation heuristics cannot be run, and a majority of results will not be reproducible.** After discussing with the AEC chairs, we provide instructions for reproducing a subset of results without Gurobi. However, for reviewers working at a degree-granting institution, it is easy to obtain a so-called **academic license** for Gurobi, which at the time of writing is free of charge. However, activating the license requires an Internet connection, and the reviewers have to give identifying information to a third-party to obtain the license for Gurobi, so **this is optional.** The rest of the readme will explicitly note how to compile the program with and without Gurobi, and how to reproduce results with and without it. The authors are not affiliated with Gurobi.

Installation

Note: In case you download this `.zip` folder on a Windows machine, move the `.zip` onto the Ubuntu-VM before unpacking it - the line endings might be wrong otherwise.

Gurobi

To install Gurobi, head to https://www.gurobi.com/downloads/gurobi-software/ and download the 64-bit linux version of "Gurobi Optimizer". At the time of writing, the current version is gurobi9.0.3 with the md5 checksum 832040cce622ba7f267e26645fcd200d. Follow the instructions in the Gurobi documentation to install the software on the machine you wish to run FastForward on.

If Gurobi was installed successfully, we recommend running `gurobi_cl` in order to make sure Gurobi is responsive. Right now, the output should look like this:

```
Failed to set up a license

ERROR 10009: No Gurobi license found (user tacas21, host tacas, hostid
279fc479, cores 1)
```

Next, head to https://www.gurobi.com/academia/academic-program-and-licenses/ and follow the instructions for obtaining and activating an academic license. At the time of writing, this requires running the `grbgetkey` command on the machine, which in turn requires an Internet connection.

Next, place the file `gurobi903/linux64/lib/gurobi90.netstandard20.dll` from the downloaded folder into the folder `artifact/src/gurobi/`. This ensures that the library is accessible for FastForward.

Gurobi might check the validity of the license while running the tool, which again requires an Internet connection, so you should leave the Internet connection of the VM active during the benchmarking process when running with Gurobi.

Quick Setup

This section provides quick instructions to install and run the artifact, assuming Gurobi was installed, as per the previous section. A random subset of benchmark instances will be reproduced; see the section 'Reproducing Results' for detailed information.

Run the following commands from the same folder that this README document is located in.

```
sh install.sh
cd artifact/src/
./republish_with_gurobi.sh
cd ../benchmark/
./run_all_on_10.sh
./plot_pruning.sh
./plot_heuristics.sh
cd plots
latexmk -g -pdf main.tex
evince main.pdf
```

Detailed Setup

Preparing dependencies

In the root folder of the artifact, execute the install script:

```
sh install.sh
```

Note that you may be prompted for your password. The process can take up to 30 minutes.

Compiling FastForward

Whether you installed Gurobi or not, the next step is to compile the program itself. In the directory `artifact/src/`, run the command

```
./republish.sh
```

(if Gurobi was not installed) or the command

```
./republish_with_gurobi.sh
```

(if Gurobi was installed).

If everything went fine, your output should look similar to this:

```
Microsoft (R) Build Engine version 16.7.0+7fb82e5b2 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  fastforward -> artifact/src/bin/Release/netcoreapp3.1/linux-
x64/fastforward.dll
  fastforward -> artifact/src/bin/Release/netcoreapp3.1/linux-x64/publish/
```

Warnings about unused variables can be ignored. Also, .NET Core may show a welcome message - this can be safely ignored as well.

Optional: Testing FastForward

You can also let some tests verify that the tool is working as expected. Move to the `artifact/tests` folder and execute

```
dotnet test -v n
```

(without Gurobi) or

```
GUROBI=true dotnet test -v n
```

(with Gurobi).

Testing can take up to 5 minutes without Gurobi, and up to 10 minutes with it. Many tests should be very fast, but some may take a bit longer. In rare cases, we have observed that the test task can get stuck. This

appears to stem from dotnet itself - if the task has not finished, but no new output has appeared for about a minute, cancel it with `Ctrl+C` and restart it. If everything went fine, you should see output similar to this:

```
Test Run Successful.
Total tests: 7608
     Passed: 7608
 Total time: 2.2080 Minutes
```

The exact number of tests depends on whether Gurobi is enabled or not.

Reproducing results:

Reproducing Figures 2-5

Move to the `artifact/benchmark/` folder for this set of instructions.

Reproducing all results (all tools on all benchmark suites) from the paper is expected to take multiple days. We offer two recommended workarounds:

- Running only approaches implemented in FastForward, i.e. FastForward with A*/GBFS with the state equation over Q, as well as Dijkstra.
- Running all tools, but not running them on all instances.

The benchmark folder contains scripts to facilitate both approaches, as well as a script to run all benchmarks.

The scripts themselves can be run with or without Gurobi. Without Gurobi, the approaches with Gurobi will still be called, and he benchmarking process will automatically handle the resulting errors.

The scripts are as follows:

- Script `run_all_on_10.sh`: Runs all tools on roughly 10% of benchmarks. Expected to take up to 3 hours to complete.
- Script `run_all_on_all.sh`: Runs all tools on all benchmark instances. Expected to take up to 24 hours to complete.
- Script `run_fastforward_on_10.sh`: Runs only the instantiations of FastForward (A*/GBFS with state equation over Q, and Dijkstra) on roughly 10% of benchmarks. Expected to take roughly 1 hour to complete.
- Script `run_fastforward_on_all.sh`: Runs only the instantiations of FastForward (A*/GBFS with state equation over Q, and Dijkstra) on all benchmark instances. Expected to take roughly 10 hours to complete.

The two scripts that run on a subset of instances can be invoked with an additional numerical parameter, e.g. `./run_all_on_10.sh 42`. This parameter is the seed for the probabilistic choice of which instances to run - to run on roughly 10% of benchmark instances, every instance that is encountered is run with a probability of 10%. It follows that with a 10% probability, there is a (tiny) chance to run all benchmarks, and also a (tiny) chance to run no benchmarks at all. This probabilistic way of picking instances was chosen to

ensure that no bias is introduced by e.g. manually picking a subset of instances to run. If you want to run a higher percentage of benchmarks, simply adjust the value of the `PROBABILITY` variable in the script itself.

For all scripts, the timeout is set to 60 seconds. It is possible to adjust to a different timeout by modifying the scripts - if you do, also modify the `TIMEOUT` variable in the python files `generate-plot.py` and `generate-comparison-plot.py`, otherwise the plots will not match the chosen timeout and for example some data may not be shown.

We allow tools to use up to slightly more than 8GB of memory. Note that the original results were also obtained with roughly 8GB of memory - if you wish to change the available memory, in the file `artifact/benchmark/benchmark_utils.py`, modify the lines

```
MAX_VIRTUAL_MEMORY = (1024  # b -> kb
                      * 1024  # kb -> mb
                      * 1024  # mb -> gb
                      * 8.1)  # <- this many gb
```

to any amount of memory. Note that the amount of memory the virtual machine has should also be adjusted accordingly in VirtualBox.

Note that the only way to reproduce the full results of the paper is if all tools are run, including the tools dependant on Gurobi, and all instances are selected. Since this requires a large amount of time, we include the other scripts as a courtesy to reviewers that are not inclined to spend the time to reproduce all results. However, we'd like to stress again that partial results are indeed just partial and may differ from the results in the paper.

After generating the results, the data is present in the `results` directory as `.json` files. Additionally, you can move to the `plots` directory and execute the command

```
latexmk -g -pdf main.tex
```

in order to compile a `.pdf` document containing plots of the results. The output will be the `main.pdf` document, which you can view e.g. with the command

```
evince main.pdf
```

Note: In case none of the preconfigured benchmarking options fit, we invite users to have a look at the scripts in order to modify them. Additionally, invoke

```
python3 benchmark.py -h
```

for more information on how to run benchmarks. The script allows precisely specifying the tools that should be run, as well as which benchmark suites they should run on. If you generate the results under the same names as the preconfigured scripts, you can manually invoke `generate_plots.sh` in order to repopulate the data for the `.pdf` output.

Reproducing Figure 6

To reproduce Figure 6 of the paper, there are two additional scripts, located in the directory `artifact/benchmark`:

- You can run `./plot_pruning.sh` to run pruning for all instances and collect information about the fraction of places and transitions which were pruned per instance. This should take around half an hour to complete.
- You can run `./plot_heuristics.sh` to calculate the initial heuristic estimation for all instances. This should be run after running the main benchmark suite - the estimations will be compared to the shortest witness for instances, and instances where no approach guaranteeing a shortest witness has reported one will be skipped. Note also, this script runs with and without Gurobi, but without it, the output will contain error messages saying that Gurobi needs to be installed to compute the state equation heuristics. These can be ignored if Gurobi is not installed. The whole process should take at most 2 hours to complete.

After running these scripts, recompile the `.pdf` document by navigating to the `artifact/benchmark/plots` directory and executing

```
latexmk -g -pdf main.tex
```

The file `main.pdf` then contains the relevant data.

Optional: Regenerating the pre-pruned instances

We also provide a script to regenerate the pre-pruned instances from the original instances. To do so, in the `artifact/benchmark` directory, run

```
sh preprune_all_nets.sh
```

Allow up to roughly 20 minutes for the task to finish. Note that cancelling this task while it is in-progress will not generate all, but only some, pre-pruned instances, so to ensure that all instances are present, it is recommended to let this task completely run through once started.

Structure

The artifact is structured as follows, with especially notable directories/files in bold:

- artifact: Contains the artifact itself.

    - benchmark: Contains scripts and files to facilitate benchmarking. To benchmark the tool, this is the main point of interest.
        - icover: The ICover tool, see https://github.com/gsutre/icover. Licensed under Apache 2.0.
        - bfc: The bfc tool, see http://www.cprover.org/bfc/. Additionally, see bfc/LICENSE.txt.
        - kreach: The kreach-tool, see https://github.com/dixonary/kosaraju/blob/master/. Licensed under GPL-3.0.
        - nets: The benchmark suite can be found here.
            - coverability: The coverability benchmark suite
            - random_walk: The random walk benchmark suite
            - sypet: The sypet benchmark suite
            - original_coverability: The original coverability benchmark suite, including safe instances
        - plots: Once benchmarking was run, a `.pdf` document containing the results can be compiled here. See the **Reproducing Results** section for more information on how to populate and compile the results document.
        - results: Once benchmarking was run, the results can be found as `.json` files here.
        - `generate_instances.py`: The entrypoint for generating the random walk instances from the original instances. The functionality has not been tested on the virtual machine, but we include the source code here for curious reviewers.
        - `benchmark.py`: The main entrypoint for benchmarking. It is recommended to call this via one of the `run_x_on_y.sh` scripts in the directory, but a curious reviewer is invited to invoke `python3 benchmark.py -h` to have a look at the interface.
    - **src: Contains the source code of the FastForward tool.** Note that this prototype includes partial implementations of some experimental features, like transfer transitions, which are not mentioned in the paper, and *not used during benchmarking*.
        - Program.cs: The main entrypoint for FastForward.
        - Heuristics: The directory containing most of the code related to computing the different heuristics introduced in the paper.
        - **`SearchAlgorithms/A_Star.cs`: Implements Algorithm 1 (directed search).**
        - PetriNet: Contains the classes used to represent Petri nets and their components, i.e. places, transitions, etc.
- dependencies: Contains dependencies that need to be installed separately.
    - mist: The mist tool, see https://github.com/pierreganty/mist/blob/master/README.md. Licensed under GPL-3.0.
    - lola: The LoLA tool, see http://service-technology.org/lola/. Licensed under the GNU AFFERO GENERAL PUBLIC LICENSE.
    - python: Python3 packages that are needed to run the project.
    - python2: Python2 packages that are needed to run the project.
    - nuget: The nuget packages that FastForward depends on.
    - pkgs: .deb files that the benchmarking process depends on.
    - z3: Precompile binaries for the z3 solver, taken from https://github.com/Z3Prover/z3. Licensed under the MIT license.

Troubleshooting

The benchmarking script outputs "Gurobi needs to be installed"

If Gurobi is not installed, expect to see this error message in the output when calling the `run_x_on_y.sh` script:

```
Gurobi needs to be installed, and the compile flag set, to use this
heuristic! See the README for more information.
```

If Gurobi was indeed not installed, this error message can be ignored.

However, if you expect Gurobi to be installed, you should make sure that you used the script `republish_with_gurobi.sh` and not `republish.sh` while compiling FastForward.

Running FastForward outputs libgurobi90.so is not installed

If you see an error message complaining about the lack of `libgurobi90.so`, it can mean Gurobi is not installed properly, but you compiled with Gurobi. Check whether you can run the `gurobi_cl` command, or compile without Gurobi by using the script `republish.sh` to compile FastForward.

Running FastForward outputs "Failed to create CoreCLR"

The first step when encountering this error is to restart the machine. If the error persists, it can mean that dotnet does not have enough memory available. From our tests, the benchmarking is successfull on the preconfigured virtual machine, but if you encounter this error, you should try adjusting the machine to have more memory.

Computing preprocessing plots outputs "Unhandled exception. Microsoft.Z3.Z3Exception: Overflow encountered when expanding old_vector"

This is expected - Z3 cannot compute continuous reachability for all instances. The error can be ignored, the script will process it and continue without issues.