A Symbolic Justice Violations Transition System for Unrealizable GR(1) Specifications

README with materials overview

Aviv Kuvent, Shahar Maoz and Jan Oliver Ringert

Research Paper Abstract

One of the main challenges of reactive synthesis, an automated procedure to obtain a correct-by-construction reactive system, is to deal with unrealizable specifications. Existing approaches to deal with unrealizability, in the context of GR(1), an assume-guarantee expressive fragment of LTL that enables efficient synthesis, include the generation of concrete counter-strategies and the computation of an unrealizable core. Although correct, such approaches produce large and complicated counter-strategies, often containing thousands of states. This hinders their use by engineers.

In this work we present the Justice Violations Transition System (JVTS), a novel symbolic representation of counter-strategies for GR(1). The JVTS is much smaller and simpler than its corresponding concrete counter-strategy. Moreover, it is annotated with invariants that explain how the counter-strategy forces the system to violate the specification. We compute the JVTS symbolically, and thus more efficiently, without the expensive enumeration of concrete states.

Finally, we provide the JVTS with an on-demand interactive concrete and symbolic play.We implemented our work, validated its correctness, and evaluated it on 14 unrealizable specifications of autonomous Lego robots as well as on benchmarks from the literature. The evaluation shows not only that the JVTS is in most cases much smaller than the corresponding concrete counter-strategy, but also that its computation is orders of magnitude faster.

Suggested Steps for Materials Inspection

- We suggest to begin by running the script described here on small specifications in order to get the results quickly
  - In ./specifications/AMBA, create a new directory called “AMBA_1”. Place in it the files
    - amba_ahb_w_guar_fairness_amba_ahb_1.spectra
    - amba_ahb_w_guar_trans_amba_ahb_1.spectra
    - amba_ahb_wo_ass_fairness_amba_ahb_1.spectra
  - Run:
    ```
    python sym_debug_eval.py ../specifications/AMBA/AMBA_1/ ./amba_1.csv –s –c -j
    ```
  - Compare the results in the amba_1_space.csv to the ones in ./results/amba_no_reorder_space.csv for these specifications
- Then run all of the specifications, some of which might take some time to complete

Overview of Files and Folders

- ./**bin**/: Contains the executables and script needed for running the evaluation
- ./**results**/: Contains CSV files with the evaluation results for the specifications in the ./specifications/ folder.
- ./**specifications**/: Contains the Spectra files of the specifications for evaluation.

specifications Folder

- Contains 2 sub-folders:
  - ./**AMBA**/: the Spectra files of the AMBA (1-4) specifications taken from "Diagnostic Information for Realizability"[1].
  - ./**workshop**/: the Spectra files of the specifications created by students for a workshop.
- All specifications are in Spectra format (with extension ".spectra")

[1] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In VMCAI, volume 4905 of LNCS, pages 52–67. Springer, 2008

results Folder

- Contains 4 CSV files with the results of running the evaluation on the specifications in the ./**specifications**/ folder:
  - Size evaluation: amba_no_reorder_space.csv, workshop_no_reorder_space.csv
  - Time evaluation: amba_no_reorder_time.csv, workshop_no_reorder_time.csv
- All results are from running the evaluation without re-order of BDDs during the JVTS computation and the concrete counter-strategy extraction.
- Time results are in milliseconds.
  - Results which are over 10 minutes are given the value of -1

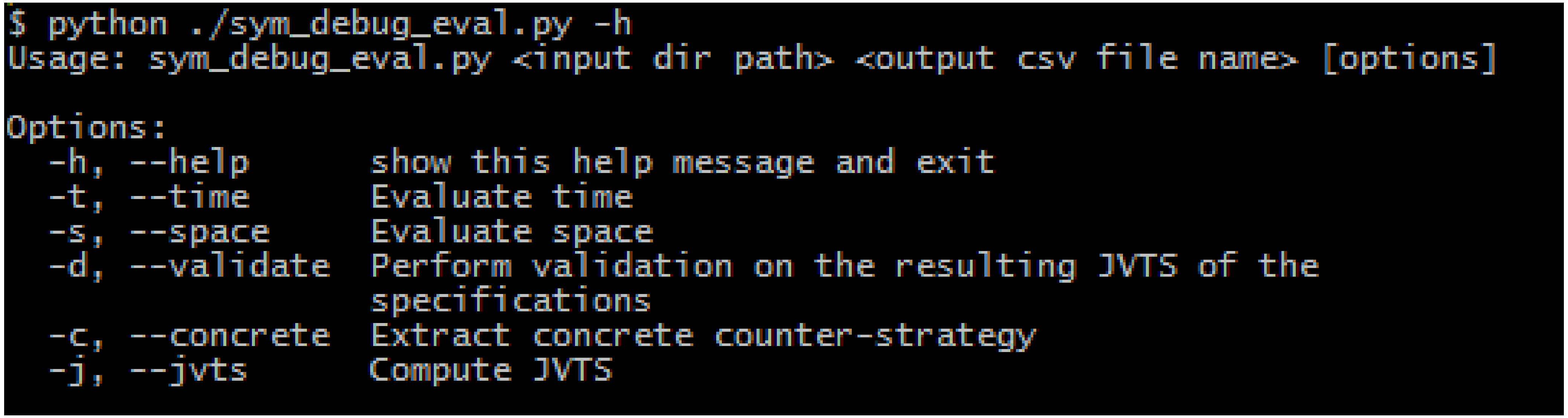
bin Folder

- Contains the following:
  - ./**jvts_lib**/: the libraries needed to run the **jvts.jar** file
  - ./**cudd3.dll**: a DLL of the BDD engine used in the **jvts.jar** file
  - ./**jvts.jar**: the executable for computing the JVTS and a concrete counter-strategy and for evaluating both.
  - ./**sym_debug_eval.py**: the python script used to run the **jvts.jar** file and perform the relevant evaluation.

sym_debug_eval.py Requirements

- Allows running size and/or time evaluation on JVTS and/or concrete counter-strategy
- Requires:
  - Python 2.7
  - Java 8 64Bit
  - Microsoft Visual C++ 2015 Redistributable (x64) – https://www.microsoft.com/en-us/download/details.aspx?id=48145
- Run on Linux (or on Windows via Cygwin - https://www.cygwin.com/)
- For evaluation, we ran on a PC with the following specs:
  - Inter i7 CPU 3.4GHz (using only a single core of the CPU)
  - 16GB RAM
  - Windows 7 64bit
  - Python 2.7
  - Java 8 SE 64Bit

sym_debug_eval.py Parameters



```
$ python ./sym_debug_eval.py -h
Usage: sym_debug_eval.py <input dir path> <output csv file name> [options]

Options:
  -h, --help      show this help message and exit
  -t, --time      Evaluate time
  -s, --space     Evaluate space
  -d, --validate  Perform validation on the resulting JVTS of the
                  specifications
  -c, --concrete  Extract concrete counter-strategy
  -j, --jvts      Compute JVTS
```

- Running validation might take a very long time on some of the specifications.
- Output csv file name will be concatenated with "_space" or with "_time" if performing size / time evaluation.
  - Performing both will result in 2 output files.

sym_debug_eval.py Usage Examples

- To run time evaluation of AMBA specifications for JVTS and concrete counter-strategy use:

```
python sym_debug_eval.py ../specifications/AMBA/ ./amba_output.csv –t –c -j
```

- To run size evaluation of workshop specifications for JVTS and concrete counter-strategy use:

```
python sym_debug_eval.py ../specifications/workshop/ ./workshop_output.csv –s –c -j
```