# Performance Heuristics for GR(1) Synthesis and Related Algorithms

README with materials overview

Elizabeth Firman, Shahar Maoz and Jan Oliver Ringert

# Research Paper Abstract

Reactive synthesis for the GR(1) fragment of LTL has been implemented and studied in many works. In this work we present and evaluate a list of heuristics to potentially reduce running times for GR(1) synthesis and related algorithms. The list includes several heuristics for controlled predecessor computation and BDDs, early detection of fixed-points and unrealizability, fixed-point recycling, and several heuristics for unrealizable core computations. We have implemented the heuristics and integrated them in our synthesis environment Spectra Tools, a set of tools for writing specifications and running synthesis and related analyses.

We evaluate the presented heuristics on SYNTECH15, a total of 78 specifications of 6 autonomous Lego robots, on SYNTECH17, a total of 149 specifications of 5 autonomous Lego robots, all written by 3rd year undergraduate computer science students in two project classes we have taught, as well as on benchmarks from the literature. The evaluation investigates not only the potential of the suggested heuristics to improve computation times, but also the difference between existing benchmarks and the robot's specifications in terms of the effectiveness of the heuristics.

Our evaluation shows positive results for the application of all the heuristics together, which get more significant for specifications with slower original running times. It also shows differences in effectiveness when applied to different sets of specifications. Furthermore, a comparison between Spectra, with all the presented heuristics, and two existing tools, RATSY and Slugs, over two well-known benchmarks, shows that Spectra outperforms both on most of the specifications; the larger the specification, the faster Spectra becomes relative to the two other tools.

# Overview of Files and Folders

- Python test script:
  - **TestRuntimes.py** – a script to execute different tests as presented in the paper
- Executable jar files:
  - **spectra_realizability_test.jar** – the jar file to run the realizability test
  - **spectra_ddmin_test.jar –** the jar file to run the DDmin test
- Specifications zip files:
  - **SYNTECH15.zip** – autonomous Lego robots specifications in Spectra format (created by students)
  - **SYNTECH17.zip** – autonomous Lego robots specifications in Spectra format (created by students)
  - **AMBA_GenBuf_specs.zip** – AMBA and GenBuf specifications generated in various formats
- Performance tests raw data:
  - **PerfTestsRawData.zip** – excel files containing all the raw data that was used for the tables presented in the paper in the evaluation section.
- Libraries used by the executable jar files:
  - **cudd3.dll** – CUDD3.0 library and Spectra C code in a dynamic-link library for execution on Windows OS.
  - **libcudd3.so** – CUDD3.0 library and Spectra C code in a shared object for execution on Linux OS.

# Materials Inspection

- Using TestRuntimes.py tool:
  - Run ***python3.6 TestRuntimes.py –h*** and execute a test according to the presented usage
  - Examples:
    1. Run realizability test with 2 iterations on a realizable AMBA specification, with GR1 game algorithm and all of the optimizations, with log files written to 'results' folder:
       *python3.6 TestRuntimes.py -i amba_ahb_01.spectra -o results --single -r 2 --gr1 --realizability --type optimized*
    2. Run the same test but only with partitioned transition relation and fixed-point recycling:
       *python3.6 TestRuntimes.py -i amba_ahb_realizable_amba_ahb_1.spectra -o results --single -r 2 --gr1 --realizability --type prt fpr*
    3. Run DDMin test with 1 iteration on an unrealizable SYNTECH17 specification, with Rabin game algorithm and all the fixed-point and BDD optimizations (denoted by opt1 in the evaluation section of the paper):
       *python3.6 TestRuntimes.py -i* roboticarm_Robot_224.spectra *-o results --single -r 1 --rabin --DDMin --type opt1*

# Materials Inspection

- Using the jar files directly:
  - Realizability tests: Run ***java -jar realizability_test.jar*** *and e*xecute a test according to the presented usage
    - Example: run realizability test on an unrealizable GenBuf specification, with GR1 game algorithm and grouping of variables and their primed copies:

      *java -jar realizability_test.jar --game_type gr1 -group*

      *--input genbuf_w_guar_fairness_05.spectra --output genbuf_w_guar_fairness_05*
  - DDMin tests: Run ***java -jar ddmin_test.jar*** *and e*xecute a test according to the presented usage
    - Example: run DDMin test on an unrealizable SYNTECH15 specification, with Rabin game algorithm and the optimization of contained sets:

      *java -jar ddmin_test.jar --game_type rabin -sets*

      *--input HumanoidLTL_531_Humanoid_unrealizable.spectra*

      *--output HumanoidLTL_531_Humanoid_unrealizable*

# System Requirements

- Requirements for the tests execution:
  - Java 8 64Bit
  - Pyhon3.6
- For evaluation, we ran on a PC with the following specs:
  - Tests on windows:
    - Intel Xeon W-2133 Processor with 32 GB RAM
    - Windows 10 64bit
  - Tests on Ubuntu:
    - VM of a KVM hypervisor, on host OS Windows 10.
    - The hardware used is Intel Xeon W-2133 Processor with 32 GB RAM.
    - The VM runs Ubuntu 18.04.1 LTS
    - The VM configured with 10 GB RAM and 6 cores
    - Before running tests must run *export LD_LIBRARY_PATH='/path/to/libcudd3'*