Performance Heuristics for GR(1) Realizability Checking and Related Analyses

Artifact Evaluation Guide

Contents

1 Introduction

This document provides a comprehensive guide on how to run and evaluate the experiments associated with the paper *"Performance Heuristics for GR(1) Realizability Checking and Related Analyses"*. It includes details about the artifact contents, run configurations, execution instructions, results format, and the structure of configuration files.

Navigate to the Example Execution (Section 6.3) for a quick execution of a single specification.

2 Artifact Contents

The provided zip file contains the following directories and files:

1. **test**: This directory contains tools to run the evaluation and reproduce the results. It is runnable by executing the `run.bat` file. Refer to Section 5 for the complete information.
2. **results**: This directory contains the CSV results.
    1. **memoryless**: Contains CSV files with the results for the "Partial memoryless vs. Spectra" row in Table 1.
    2. **ordering**: Contains CSV files with the results for the "Static variable ordering vs. partial memoryless" and "Justice ordering vs. partial memoryless" experiments in Table 1.
    3. **sfa**: Contains CSV files with the results for the "Simplified automata + auxiliary grouping vs. partial memoryless" in Table 1.
    4. **alg_reduction**: Contains CSV files with the results for "WS vs. partial memoryless" and "Inherent vacuity vs. partial memoryless" experiments in Table 1.
    5. **final**: Contains CSV files with the results for the experiments in Table 2.
3. **data**: This directory contains the **SYNTECH** corpus, its filtered versions and the **PARAMETRIC** corpus.
    1. **original_subfolders**: The datasets are organized into multiple folders for tracking which category does each spec belong to.
    2. **onefolder**: Contains all the dataset folders used in the experiments, where each folder doesn't store the specs inside subfolders for manageability.
    3. **scripts**: Scripts for filtering and preprocessing of the datasets.
4. **analysis**: This directory contains the Python code for filling the results tables in the evaluation section.

3 Run Configurations

Due to the use of multiple heuristics, their combinations and variants, we used a modular experiment framework. A configuration allows the adjustment and combination of every setting for an experiment.**Each configuration file specifies the settings for an experiment, allowing a combination of any possible argument of every setting.**

3.1 Structure of a Config File

A configuration file contains general settings and multiple configuration blocks.

3.1.1 General Settings

The first four lines are general settings for the run:

- `NAME`: The name of the run.
- `FOLDER_PATH`: The path to the specifications.
- `OUT_PATH`: The path to the output results.
- `RUNS_PER_CONFIG`: Number of runs for each configuration for each specification.

3.1.2 Configuration Blocks

Each configuration block defines a set of settings for a particular experiment. The structure is as follows:

- The block starts with `CONFIG_X`, where `X` is a unique identifier.
- The next line is a descriptive name for the configuration.
- Subsequent lines are key-value pairs specifying settings, in the format `SETTING_NAME: VALUE`.
- The block ends with a single line containing an asterisk (`*`).

3.1.3 Example Config File

Below is an example configuration file:

```
NAME: just_ordering
FOLDER_PATH: ./SYNTECH
OUT_PATH: ./OUT
RUNS_PER_CONFIG: 3

CONFIG_1
SORTED_JUST
ACTION_TYPE: REALIZABILITY
MEMORYLESS: true
INCREMENTAL_JUSTICES: INCREMENTAL_GRAPH
*
CONFIG_2
NOTHING
ACTION_TYPE: REALIZABILITY
MEMORYLESS: true
*
```

In this example, two configurations are defined to compare realizability checking with and without incremental justice ordering in memoryless mode.

3.2 Explanation of Configuration Blocks

Each configuration block contains the following elements:

- **Configuration Identifier**: Starts with `CONFIG_X`, where `X` is the configuration serial number.
- **Name**: A descriptive name for the configuration (e.g., `SORTED_JUST`, `NOTHING`).
- **Action Type**: The action to perform (e.g., `REALIZABILITY`).
- **Settings**: Key-value pairs specifying settings for this configuration. Not including a setting sets it to the default value. Common settings include:
  - `MEMORYLESS`: A boolean value (`true` or `false`) indicating whether to use the partial memoryless algorithm.
  - `INCREMENTAL_JUSTICES`: Specifies the justice ordering method. Possible values are `NO_INCREMENTAL` and `INCREMENTAL_GRAPH`.
- **End of Configuration Block**: A single line containing an asterisk (`*`) indicates the end of the configuration block.

4 Arguments for ACTION_TYPE and Settings

4.1 ACTION_TYPE Options

Possible arguments for ACTION_TYPE include:

- NOTHING
- REALIZABILITY
- KIND_REALIZABILITY
- VACUITY_SYS_BEHAVIORS
- VACUITY_ENV_BEHAVIORS
- VACUITY_CHECK_SATISFIABILITY
- VACUITY_ENV_REDUNDANT_VALS
- VACUITY_SYS_REDUNDANT_VALS
- WELL_SEPARATED_WITH_GAR
- WELL_SEPARATED_NO_GAR

4.2 Configuration Settings

For each setting, the default value is the first one listed.

- **INCREMENTAL_JUSTICES**: NO_INCREMENTAL, INCREMENTAL_GRAPH
- **GROUP_OPTION**: NO_GROUPING, GROUP_FIXED, GROUP_NOTFIXED
- **GROUP_AUX**: NO_GROUPING, GROUP_DOM, GROUP_PAIR, GROUP_PAIR_SEPARATED
- **REDUCED_ALGORITHM**: FALSE, TRUE, PARTIAL
- **STATIC_ORDER**: NO_STATIC_ORDERING, REVERSE_CUTHILL
- **MEMORYLESS**: false, true
- **NEW_TRIGGER**: false, true
- **NEW_PATTERN**: false, true
- **VERIFICATION**: false, true
- **PARALLEL**: false, true

5 The Running Framework

The running framework is located in the test directory and includes several important files and folders.

5.1 Important Files

- run.jar: The Java archive containing the experiment framework.
- run.bat: The batch file for running the experiment framework.
- Cudd.dll: The CUDD C native code for BDD operations.
- StaticVariableOrdering.dll: The C++ native code for the Cuthill-McKee ordering algorithm.
- main.py: A Python script for graph clustering and metrics calculations.

5.2 Important Folders

- `CONFIGS`: Contains all configuration files for experiments.
- `OUT`: The directory where output results are saved. Each run outputs a CSV and serialized files inside a new folder.
- `TEMP`: Stores temporary files used during execution and the Dwyer `.spectra` files before and after pattern simplification.
- `DATA`: Contains example specifications for testing.

6 Executing the Experiments

This section provides practical instructions on how to run the experiments.

6.1 Prerequisites

- Windows operating system.
- Up-to-date JDK.

6.2 Execution Steps

1. **Navigate to the `test` Directory**

   Open a command prompt and navigate to the `test` folder provided in the artifact.

2. **Run the `run.bat` File**

3. **Select Configuration**

   Upon running, you will be prompted to select settings:

   a. **Select Config File**
   Choose the desired configuration file from the `CONFIGS` folder. For example, select `example.txt` to select the example configuration.

   b. **Set Timeout**
   Enter the run timeout in seconds. A run will be terminated after this time. The default used in the paper is **600** seconds.

   c. **Select Isolation Option**
   To prevent possible interference between runs that might influence runtime, we allow performing each run in isolation:

   - `NO_ISOLATION`: No isolation.
   - `THREAD_ISOLATION`: Thread isolation.
   - `PROCESS_ISOLATION_JAR`: Process isolation using a separate JAR.

   The setting used in the paper is `PROCESS_ISOLATION_JAR`.

4. **Execution**

   The experiment will run according to the configurations specified. Useful information will be displayed during the run.

   The framework performs a total of `(Amount of Specs × Amount of Configs × Runs per Config)` runs. It processes each specification sequentially, and for each specification, it performs `(Amount of Configs × Runs per Config)` runs in an interleaved manner, cycling between the configurations (e.g., Config 1, Config 2, Config 1, Config 2, etc.).

   This interleaving attempts to improve the accuracy of runtimes relative to each other by reducing the impact of external system factors on a single configuration.

5. **Retrieve Results**

   Upon completion, the CSV result files will be generated in the `OUT` directory specified in the configuration file. The path to the result file is provided at the end of the run.

6. **Validation Step (Optional)**

   After generating the results, an optional validation step is automatically performed. This step checks that, for each specification, the Game Model and Z objects are identical across all runs.

   The Z objects are verified separately for different types of realizability checks: GR(1), GR(1) with kind realizability, and Rabin realizability runs.

6.3 Example Execution

For a quick test, run an example experiment using the pre-defined "example.txt" configuration:

1. Navigate to the `test` folder.
2. Run `run.bat`.
3. When prompted, select `example.txt` as the configuration file.
4. Use the default settings for timeout and isolation, **600** and **PROCESS_ISOLATION_JAR** respectively.
5. The experiment should complete in less than 2 minutes.

7 CSV Results

A CSV result file contains measurement data for each run of the experiment. Each row represents a single run and includes useful information.

7.1 Main Columns

The main columns of a CSV result file include:

1. **Spec**: Name of the specification.
2. **RunConfig**: The configuration used.
3. **ActionType**: The type of work performed.
4. **Result**: The result of the work (e.g., `SYS_REAL`, `SYS_UNREAL`).
5. **TOTAL_TIME**: The total time of the run, in milliseconds.

7.2 Additional Data Groups

The data is grouped into the following sections:

1. **Run Metadata**: Details such as unique identifier, timeout settings, and configuration file used.
2. **Time Measurements**: Timing information for different phases (e.g., BDD initialization, heuristic methods, game model construction).
3. **Ordering Information**: Variable ordering after each significant phase.
4. **Node Size Data**: BDD node sizes after each phase.
5. **Fixpoints**: Number of fixpoints reached during execution.

6. **Reordering Statistics**: Total reordering time, number of reordering calls, average reordering time, and reordering gain.
7. **Specification Metadata**: Data about counters, triggers, patterns, monitors, and variables.
8. **SAT Counts**: SAT counts of the game model and results.
9. **Game Model Information**: Graph and cluster data, domain graphs, constraint graphs, and more.