MultiGain 2.0 - User Documentation

Table of Contents

Introduction

We present **MultiGain 2.0**, a tool for the formal verification of probabilistic systems that involve a multidimensional reward structure and are subject to steady-state constraints as well as Linear-Temporal-Logic specifications. This tool is built on top of the **PRISM model checker** and incorporates multi-objective capabilities of the controller synthesis tool **MultiGain** [1]. Our tool also provides an approach for finite memory solutions and the capability for 2- and 3-dimensional visualization of Pareto curves to aid the trade-off analysis in multi-objective scenarios.

Getting Started

Requirements

- Java 11+
- gcc 12.1.0+

Gurobi

MultiGain2.0 uses an LP solver as back-end, with `lpsolve` per default included in the sources. There is also the option of using the commercial solver Gurobi, which for licensing reasons cannot be included as part of the distribution. If you want to use Gurobi with MultiGain2.0, you need to follow these steps. Otherwise, skip directly to the installation.

Obtain and download Gurobi and install the licence. Instructions for linux can be found at gurobi.com. It may be noticed, that Gurobi offers free licenses for academic purposes. You can find documentation for your operating system here: gurobi.com/documentation/quickstart.html

Before installing and each use of MultiGain, the `GUROBI_HOME` environment variable must be set.

```
export GUROBI_HOME="opt/gurobi952/linux64"
```

The path may vary depending on the installed version of Gurobi and your operating system. You can find which path is the correct one for your system by browsing the Gurobi Quick start guide: www.gurobi.com/documentation/quickstart.html

You can always check if the variable is set correctly by calling:

```
echo $GUROBI_HOME
```

Installation

Download the provided `multigain2.zip`, open a terminal and switch into the directory the downloaded file is in. Then extract the artefact and install it:

```
unzip multigain2.zip
cd multigain2
cd prism-4.7-src/prism
make clean_all
make
```

If the environment variable `GUROBI_HOME` is not set expect the following message during compilation:

```
GUROBI HOME is not set. Not compiling Gurobi support
make[1]: Leaving directory `.../multiObjective/prism/prism/ext/gurobi'
```

This is not an error message, but a warning.

You can test if the installation finished correctly by running this example:

```
bin/prism examples/example.prism examples/example.props
```

Install with Gurobi

If you want to use Gurobi you additionally have to copy the Gurobi library files into the library folder:

```
cp -r $GUROBI_HOME/lib/* lib/
```

Test your Gurobi installation by running the example with gurobi

```
bin/prism examples/example.prism examples/example.props --gurobi
```

If you want to use Gurobi in an IntelliJ run configuration you have to mark `prism/ext/solver/gurobi` as source

Plotting Pareto Curves

For plotting Pareto curves generated by MultiGain2.0 we recommend installing Anaconda.[2] Then create a new conda environment.[3]

```
conda create -n multigain2 python=3.10
Proceed ([y]/n)? y
```

Activate the environment and install the required packages:

```
conda activate multigain2
pip install argparse matplotlib
```

Assuming you are still located in the prism directory, you can test the installation by plotting one of our pregenerated files:

```
python ./etc/scripts/pareto-plot.py examples/results/pareto2d.pareto
```

User Guide

Basic Usage

Create your own model and run configuration by following the official Prism instructions on the PRISM language and the Prism Property Specification [4].

The now created model can be checked by adapting and running the following general command from the root directory of the project (i.e. the `multigain2` directory):

```
bin/multigain2 path_to_model_file path_to_property_file [--gurobi] [--exportpareto]
[--exportstrat]
```

If the command throws any error messages, try setting up a fresh symlink:

```
rm bin/multigain2
ln -s prism-4.7-src/prism/bin/prism bin/multigain2
```

A selection of example models and benchmarks can be found in the `examples` directory. Applying the command may look as follows:

```
bin/multigain2 examples/meanpayoff/pacman.10.prism examples/meanpayoff/pacman.props
```

Each of the `examples` subdirectories further contains a shell script named `run.sh`, which runs all contained models and logs the output in the corresponding `results` subdirectory.

The standard command may also be extended with optional flags.

--gurobi

Uses Gurobi as the LP solver instead of the default `lpsolve`. PRISM needs to be built with Gurobi support see how to build with Gurobi.

--exportpareto `pareto_file`

Export the Pareto curve to `pareto_file` when checking properties with multiple quantitative LRA objectives.

This will overwrite the file so only one Pareto query should be present in the `prism_property_file`.

To plot the generated Pareto file you can use the provided script as follows:

```
python prism-4.7-src/prism/etc/scripts/pareto-plot.py path_to_pareto_file
```

--exportstrat `"policy_filename":"type=type_name"`

Export the computed policy to `policy_filename`. The parameter `type_name` may be one of the following strings:

`dot`: The policy is exported as two (partial) copies of the (product) MDP representing the transient and recurrent behaviour and actions connecting the two according to the switch probabilities. Each action label is prefixed with [T], [R] or [SW] indicating whether they belong to the transient or recurrent behaviour or the switch between both respectively. The action labels are further extended with the probability value as assigned by the policy. For example an action `act`, which our policy would choose with probability 0.75 in the recurrent run is labelled on the exported MDP as `[R]act:0.75`.

`actions`: The policy is exported as a more lightweight textfile in table format sectioned in transient and recurrent behaviour. Each of the rows consists of a state of the (product) model, the distribution over the state's actions and the switch probability from transient to recurrent behaviour.

The `exportstrat` flag may not be used when computing a Pareto curve (i.e. more than one quantitative reward property specified). Furthermore, since the policy computed by the `detmulti` keyword is deterministic and memoryless, both `type_name` options export the strategy on the original MDP.

Property Specification

Query Wrappers

As entry point for **MultiGain 2.0** functionality every query in the property file has to be wrapped with the `multi(⋯)`, `mlessmulti(⋯)`, `unichain(⋯)` or `detmulti(⋯)` keyword.

The **`multi`** keyword describes the standard functionality of **MultiGain 2.0**. It allows for an LTL-Specification, Steady-State-Specifications and arbitrary many (quantitative) reward specification.

The **`mlessmulti`** keyword may be used in case the `multi` keyword results in an unbound memory policy. The policy allows for an additional integer literal at the front of the property list. This integer fixes the maximum number of steps the policy takes before visiting an accepting state again in the long run. The resulting policy therefore requires only finite memory. The tool will then ouput the minimal relaxation factor `delta` [5] we have to relax the steady-state and boolean reward constraints with. Since the objective function is preoccupied with `delta` it is not possible to specify quantitative reward properties in an `mlessmulti` query.

The **`unichain`** keyword allows the same properties as a standard `multi` query with a maximum of one quantitative reward specifications. **MultiGain 2.0** will then compute if there exists a solution to the query whose corresponding policy constitutes a unichain on the MDP. This is achieved by iteratively searching through the maximum end components of the MDP. If a quantitative reward is specified, **MultiGain 2.0** will return the unichain solution maximizing (or minimizing) the respective reward structure.

The **`detmulti`** keyword allows for an LTL-Specification and arbitrary many Steady-State-Specifications. **MultiGain 2.0** will compute a deterministic unichain policy following the approach by A. Velasquez et al.[6]

Accepting Frequency Bound

This is exclusively allowed at the start of the property list in an `mlessmulti` query. The bound is denoted as a single integer literal, as in the examples below. It specifies the maximum number of steps the policy takes before visiting an accepting state again in the long run.

Reward-Specification

Reward constraints can be specified using the `R` operator. These may be of qualitative (boolean) (`>=` ,`<=`) or quantitative (`max=?`, `min=?`) nature. Quantitative reward specifications aim to optimise the corresponding rewards value. If more than one quantitative specifications are included, **MultiGain 2.0** will approximate the Pareto curve.

LTL-Specification

LTL formulas may be specified using the `P` operator. The notation does not differ from the standard Prism notation for temporal logics. Only one LTL specification may be specified per query.

Steady-State-Specification

Steady-State constraints can be defined with the `S` operator. The notation does not differ from the standard Prism notation for steady-state-constraints. You can specify multiple steady-state-constraints per query.

To specify a Steady-State-Constraint with an equality operator, please default to using two `S` operators, with `<=` and `>=`.

Examples

```
multi(R{"reward1"}max=? [S], R{"reward2"}>=0.25 [S], S>=0.25 ["ssLabel"], P>=1 [F
state!=2])
```

```
multi(R{"reward1"}max=? [ S ], R{"reward2"}max=? [ S ], P>=1 [ G state!=1 ], S>=0.5 [
"someLabel" ])
```

```
mlessmulti(100, P>=1 [ G F "acc" ], S>=1 [ "ss" ])
```

```
unichain(R{"unbalanced"}max=? [S], P>=1 [(F "a") | (F "b")])
```

```
detmulti(P>=0.75 [(! "danger") U "tool"], S>=0.75 ["home"], S<=1 ["home"])
```

More examples can be found in the `examples` directory.

Example Clarification

In the following section we will clarify the models and property files provided in the `examples` directory in the order of the respective subdirectories. The checkmark and cross indicate if the files have been used in experiments of my master's thesis or not respectively.

examples/grid

This directory contains all models used in my thesis and during development, that are based on the gridworld model. The corresponding property files contain commented out queries for a better understanding of how to tune the parameters.

✔ `gardentool_app.prism, gardentool_app.props` These contain the model and property file used for the examplary gridworld application of Chapter 4 of the thesis.

✖ `grid#.prism` These files contain gridworld models with non-Dirac transition distributions. They have not been evaluated in the thesis. The corresponding python script is named `grid_prism.py`.

✔ `griddet#.prism` These comprise the models used for the experimental evaluation of the thesis as described in Setup 1. They have been generated using the `grid_prism_det.py` script.

✔ `griddet#_ss#.prism` These files contain the models used for experiments on steady-state constraint scaling. For a better overview only one such file is pregenerated and included in the run script. The other testcases may be simulated using the generator python script `grid_prism_det_ss.py` and provided property file `grid_ss.props`.

✔ `griddet#_pareto#.prism` These files contain the models used for experiments on LRA constraint scaling. For a better overview only one such file is pregenerated and included in the run script. The other testcases may be simulated using the generator python script `grid_prism_det_pareto.py` and provided property file `grid_pareto.props`.

✔ `python scripts` The `grid_prism.py` and `grid_prism_det.py` scripts have been associated with the previous model files, for the generation of randomized gridworld models as described in Setup 1 of my thesis. Both expect a single commandline argument specifying the gridworld size as an integer. Furthermore `grid_prism_det_ss.py` may be used to generate the model input files for the experiments on scaling the steady-state constraints. A second commandline argument is expected as an integer indicating the number `D` of SS constraints to generate. Note that `D` must be smaller than 1/4 of the amount of cells of the grid model. The `grid_prism_det_pareto.py` script is provided for the generation of randomized models for the experiments on scaling the LRA rewards. Similarly a second commandline argument is expected to specify the number of reward structures to generate.

examples/meanpayoff

This directory contains all models used for the benchmark evaluation and comparison with the PET tool included in the thesis. The provided run script executes all benchmarks of Table 5.2. This is only a single run of each model, not the average over 5 runs. Note that the unichain queries are also not included, since they lead to some timeouts. To verify the times listed in the table, feel free to substitute the `multi` keyword of the property files with the `unichain` keyword.

examples/membound

This directory contains models for the experiments on unbounded-memory policy solutions.

✔ `membound1_1.prism` This file contains the example from Figure 4.4 of my thesis. The corresponding property file is `membound.props`. The provided run script solves the example and exports the policy depicted in Figure 4.5.

✖ `membound2_1.prism` A "bloated" version of the previous example. It is easily comprehensible and may help for the understanding of the unbounded-memory problem and the `mlessmulti` keyword when used with the membound.props` property file. It is encouraged to play with the memory

bound integer of the property file.

examples/pareto

This directory contains examples for the comprehension and visualization of the Pareto approximation. It is encouraged to use the provided runscript for automatic execution, export and visualization of the examples.

✔ `pareto2d.prism, pareto2d.props` These files are model and property file for an example of a non-trivial 2-dimensional Pareto frontier approximation. The computed Pareto curve matches the one displayed in Figure 4.15 of my thesis.

✔ `pareto3d.prism, pareto3d.props` These files are model and property file for an example of a non-trivial 3-dimensional Pareto frontier approximation. The computed Pareto curve matches the one displayed in Figure 4.15 of my thesis.

examples/thesis

This directory contains the model and property files for the running example of Section 4.4.1 of my thesis as depicted in Figure 4.8. Note that it is only possible to provide property queries for Figure 4.9 and Figures 4.13/4.14 since the other Figures demonstrate theoretical solution not computed with the final version of **MultiGain 2.0**. The included runscript executes the `multi` (Figure 4.9) and `mlessmulti` (Figure 4.13/4.14) query as described in my thesis.

examples/unichain

✘ `unichain.prism, unichain.props` These files contain an example where no policy satisfying the specification may constitute a unichain. Thus the result is `false`. Further details are commented in the files.

✔ `unichain2.prism, unichain2.props` These files contain the example depicted in Figure 4.6 of my thesis. The example is first solved with a `multi` query and subsequently with a `unichain` query. The latter is not able to find a unichain solution satisfying the SS specification. Note that a `multi` query does find a solution and hence returns `true`.

✔ `unichain_reward.prism, unichain_reward.props` These files contain the example depicted in Figure 4.7 of my thesis. Feel free to verify, that the equivalent `multi` query as included in the property file does not achieve a greater mean payoff, and in fact computes the same policy.

examples/velasquez_paper

This directory contains the examplary models of the paper from Velasquez et al. proposing the deterministic policy approach. These have been useful in evaluating the correctness of my implementation.

✔ `figure1.prism, figure1.props` This example is depicted in Figure 5.4(a) of my thesis. For the experiments on changing the constant epsilon contact a developer or try to change the constant directly in the source code and rebuild the project.

✔ `gardentool_agent.prism, gardentool_agent_ss#.props` These contain the model depicted in Figure

5.6 of my thesis. The two property files compute the policy corresponding to the two subfigures (a) and (b).

The provided runscript executes the examples and exports the corresponding deterministic policies in DOT format and table format respectively for better readability.

Common Errors

- ```
  Error: Problem when initialising an LP solver. InvocationTargetException was thrown
  Message: null
  The message of parent exception is: Could not initialize class gurobi.GurobiJni
  ```

  This error may appear if you forgot to copy the Gurobi library files into the Multigain library. The corresponding command as from the installation instructions above is:

  ```
  cp -r $GUROBI_HOME/lib/* lib/
  ```

[1] Brázdil, Tomáš & Chatterjee, Krishnendu & Forejt, Vojtěch & Kucera, Antonín. (2015). MultiGain: A Controller Synthesis Tool for MDPs with Multiple Mean-Payoff Objectives.

[2] www.gurobi.com/documentation/quickstart.html

[3] For details and trouble shooting see: conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#creating-an-environment-with-commands

[4] We offer some new functionality: Property Specification

[5] Křetínský, J.: Ltl-constrained steady-state policy synthesis. In: Zhou, Z.H. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21. pp. 4104–4111

[6] Velasquez, A., Alkhouri, I., Beckus, A., Trivedi, A., Atia, G.: Controller synthesis for omega- regular and steady-state specifications. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. p. 1310–1318. AAMAS '22, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2022)
  ```