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"
```

NOTE 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
```

NOTE

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
```

TIP 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 numerical objectives.

NOTE	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.

NOTE	The `exportstrat` flag may not be used when computing a pareto curve (i.e. more than one numerical 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 (numeric) 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 numerical reward properties in an `mlessmulti` query.

The `unichain` keyword allows the same properties as a standard `multi` query with a maximum of one numeric 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 numeric 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 boolean (non-numeric) (`>=`,`<=`) or numeric (`max=?`, `min=?`) nature. Numeric reward specifications aim to optimise the corresponding rewards value. If more than one numeric specifications are included, **MultiGain 2.0** will compute 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.

TIP	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"])
```

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

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)