Recent Advances in Model Checking (Alex‘s group)

by Anton Mai, Miras Sabit, Mohammad Aamer Alsmail

Github-Link to files: https://github.com/3nt0n/git_storm

Table of Contents

Introduction:

The current implementation of the Interval Iteration algorithm in Model Checkers like Storm can return wrong results on some models due to rounding.
Our modified implementation of the Interval Iteration uses the ideas of Arnd Hartmanns from his paper “Correct Probabilistic Model Checking with Floating-Point Arithmetic” to control the rounding mode which returns correct results in these situations.

How to run:

Installing and running storm with our modified algorithm:

-Install storm from source: https://www.stormchecker.org/documentation/obtain-storm/build.html (we used Linux to install Storm)

-In your installation of Storm, replace the file *IterativeMinMaxLinearEquationSolver.cpp* in the folder *storm/src/storm/solver/* with the *IterativeMinMaxLinearEquationSolver.cpp* file we provided in the Zip-Folder/Github *storm_files/*

-Then compile Storm again (in the folder *storm/build*, run ‘make’ on your console)

-To run our implementation of the Interval Iteration algorithm on any model, just run storm on a model and property with the option ‘--minmax:method interval-iteration’ added to your command.

Running the tests and benchmarks:

Additionally, we ran tests to on a model that was referenced in Hartmanns paper (he referenced “Probabilistic Model Checking and Reliability of Results” by Wimmer et al).
We used both Storm and PRISM, so you need to also install PRISM if you want to do all tests (http://prismmodelchecker.org/download.php).
These tests can be run on the standard installation of PRISM and Storm.
The tests we ran can be found in the folder paper_tests (commands ran are provided in *terminal.txt*)

The benchmarks we used to create the plots are listed in the file *plots/info_benchmarks.ods*. All the benchmarks we used are provided in the benchmarks folder. The commands we ran as well as the results we obtained are provided in the file *benchmarks/0_all_infos_benchmarks*.

Developer Guide:

The important function to run the algorithm is ‘solveEquationsIntervalIteration()’ in **line 557** of the file *storm_files/IterativeMinMaxLinearEquationSolver.cpp*.
We modified this function to obtain our improved version of the Interval Iteration algorithm.

Standard-Algorithm:

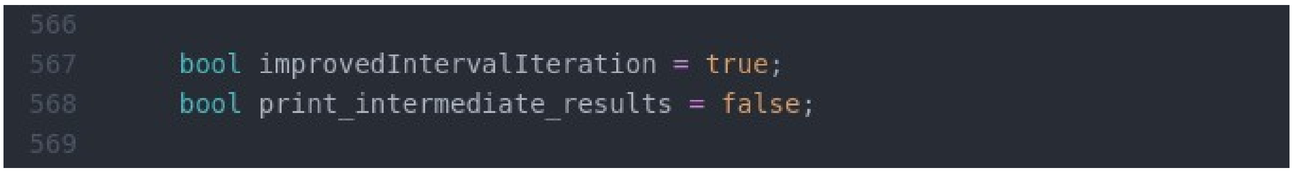
Lines 573 to 612 set up all variables needed for the algorithm.

In line 613 the while-loop starts, in which we improve lower and upper bounds until we reach a certain precision. The if/else branches test the number of iterations and whether the Gauss-Seidel-Multiplication is used.
Because the content of each branch is very similar we will used the first branch from line 622 to 641 as an example representative for each branch.
The new values for the lower and upper bounds are calculated here with the algorithm this→multiplierA→multiplyAndReduceGaussSeidel().
If needed, the function also saves the old values with preserveOldRelevantValues() to calculate the difference computeMaxAbsDiff() between the old and new bounds.

Lines 718 to 737 check whether the algorithm converged and therefore should return a result.

Our modifications:

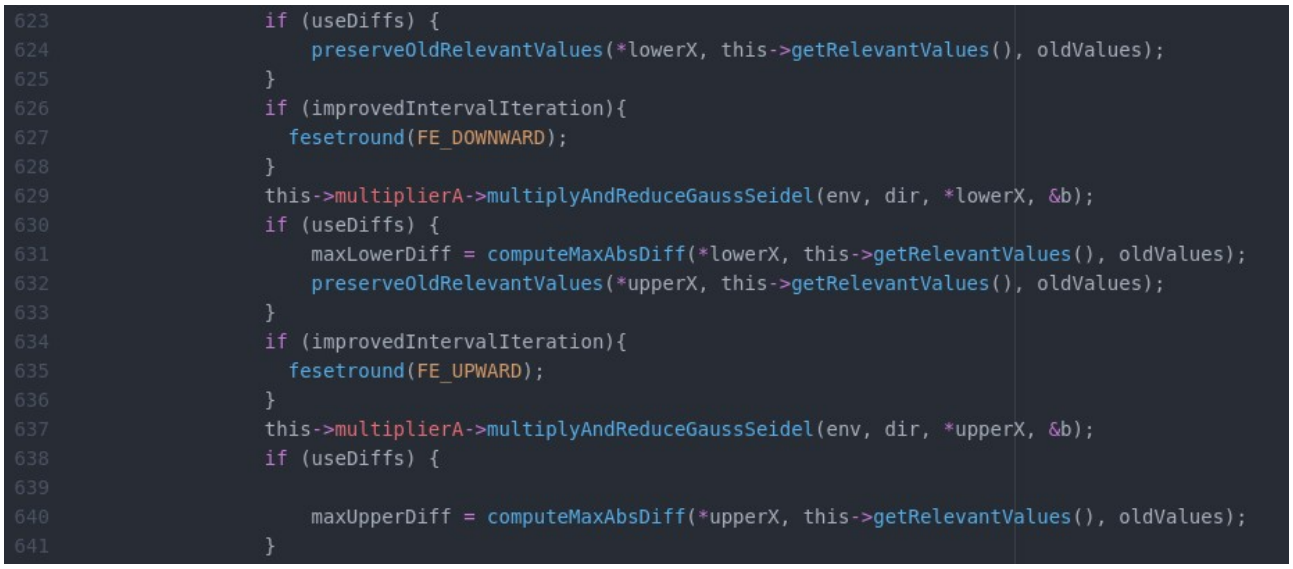
In line 567 and 568 we implemented two booleans, ‘improvedIntervalIteration’ and ‘print_intermediate_results’. These booleans make it more convenient to use/not use some of the code we added.
If ‘improvedIntervalIteration’ is true, our modified algorithm is run, otherwise the changes of the rounding mode is ignored.
If ‘print_intermediate_results’ is true, the algorithm also prints intermediate results (upper and lower bounds) to the console after each iteration. Otherwise it prints nothing.



```
566
567       bool improvedIntervalIteration = true;
568       bool print_intermediate_results = false;
569
```

Figure 1: Booleans for easier change of settings

With fesetround(), we control the rounding mode. fesetround(FE_UPWARD) makes the rounding go towards +∞, fesetround(FE_DOWNWARD) makes the rounding go towards -∞.
Between lines 626 and 702 we added the function ‘fesetround(FE_UPWARD);’ every time before the variable ‘*upperX’ is used and ‘fesetround(FE_DOWNWARD);’ every time before the variable ‘*lowerX’ Is used.

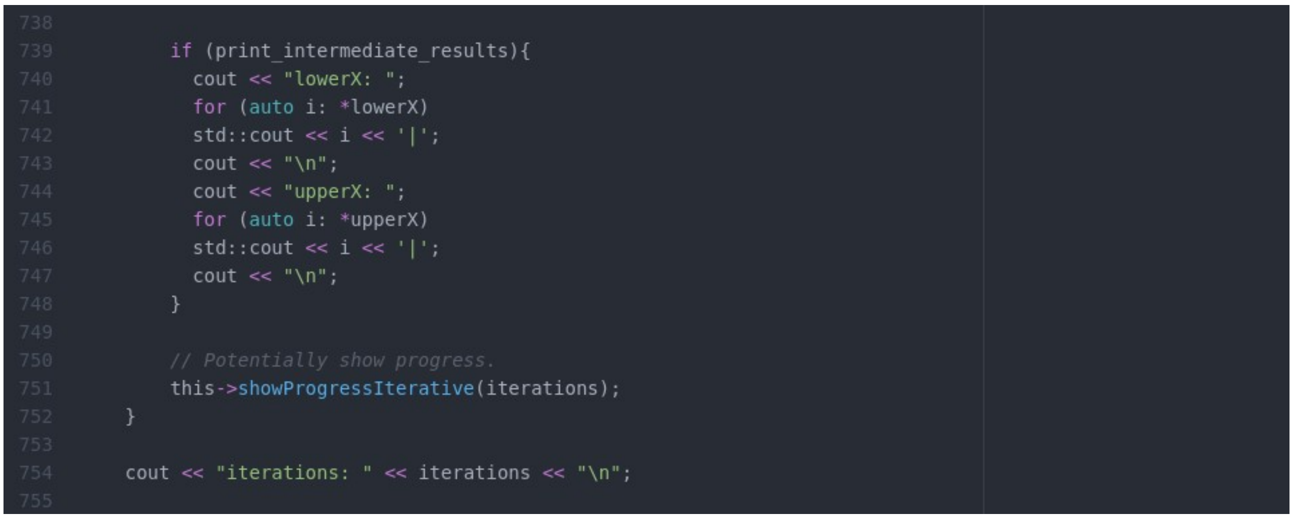


```
                if (useDiffs) {
                    preserveOldRelevantValues(*lowerX, this->getRelevantValues(), oldValues);
                }
                if (improvedIntervalIteration){
                  fesetround(FE_DOWNWARD);
                }
                this->multiplierA->multiplyAndReduceGaussSeidel(env, dir, *lowerX, &b);
                if (useDiffs) {
                    maxLowerDiff = computeMaxAbsDiff(*lowerX, this->getRelevantValues(), oldValues);
                    preserveOldRelevantValues(*upperX, this->getRelevantValues(), oldValues);
                }
                if (improvedIntervalIteration){
                  fesetround(FE_UPWARD);
                }
                this->multiplierA->multiplyAndReduceGaussSeidel(env, dir, *upperX, &b);
                if (useDiffs) {

                    maxUpperDiff = computeMaxAbsDiff(*upperX, this->getRelevantValues(), oldValues);
                }
```

Figure 2: Fesetround() function in modified code

In the lines 739 to 748 we print the intermediate results and in line 754 the number of iterations.



```
738
739         if (print_intermediate_results){
740           cout << "lowerX: ";
741           for (auto i: *lowerX)
742           std::cout << i << '|';
743           cout << "\n";
744           cout << "upperX: ";
745           for (auto i: *upperX)
746           std::cout << i << '|';
747           cout << "\n";
748         }
749
750         // Potentially show progress.
751         this->showProgressIterative(iterations);
752     }
753
754     cout << "iterations: " << iterations << "\n";
755
```

Figure 3: Printing intermediate results and iterations

Future Work:

- Currently our implementation overwrites the vanilla implementation of the Interval Iteration in storm, so you can’t run both algorithms at the same time. A future improvement to this would be to make both the vanilla Interval Iteration and our modified Interval Iteration available to run at the same time.

- We have only tested our algorithm on a small number of benchmarks during this course. In the future more tests could be run, especially on specific benchmarks that have small values close to zero (which are more prone to rounding errors).