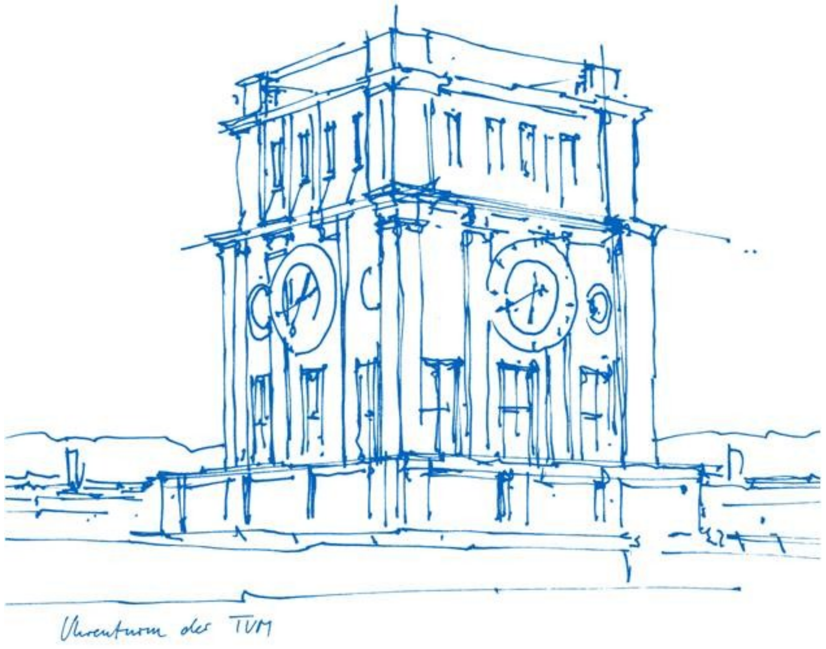
Recent Advances in Model Checking

Miras Sabit, Anton Mai, Amer (Alex‘s group)

Technial University of Munich

Chair of Theoretical Computer Science

Garching, 9th of August



Recap

Paper: Correct Probabilistic Model Checking with Floating-Point Arithmetic - Arnd Hartmanns

Key points:

- Value Iteration can lead to wrong results in model checking
  - Examples given in the Paper
- Solved by using Interval Iteration and controlling the Rounding mode
  - Algorithm with rounding mode shown in the paper
- Experiments showing that the given algorithm works were done in paper

Plan

- Value Iteration can lead to wrong results in model checking
  - Examples given in the Paper
  - → Verify Examples

Plan

- Value Iteration can lead to wrong results in model checking
  - Examples given in the Paper
  - $\rightarrow$ Verify Examples
  -
- Solved by using Interval Iteration and controlling the Rounding mode
  - $\rightarrow$ Algorithm with rounding mode shown in the paper

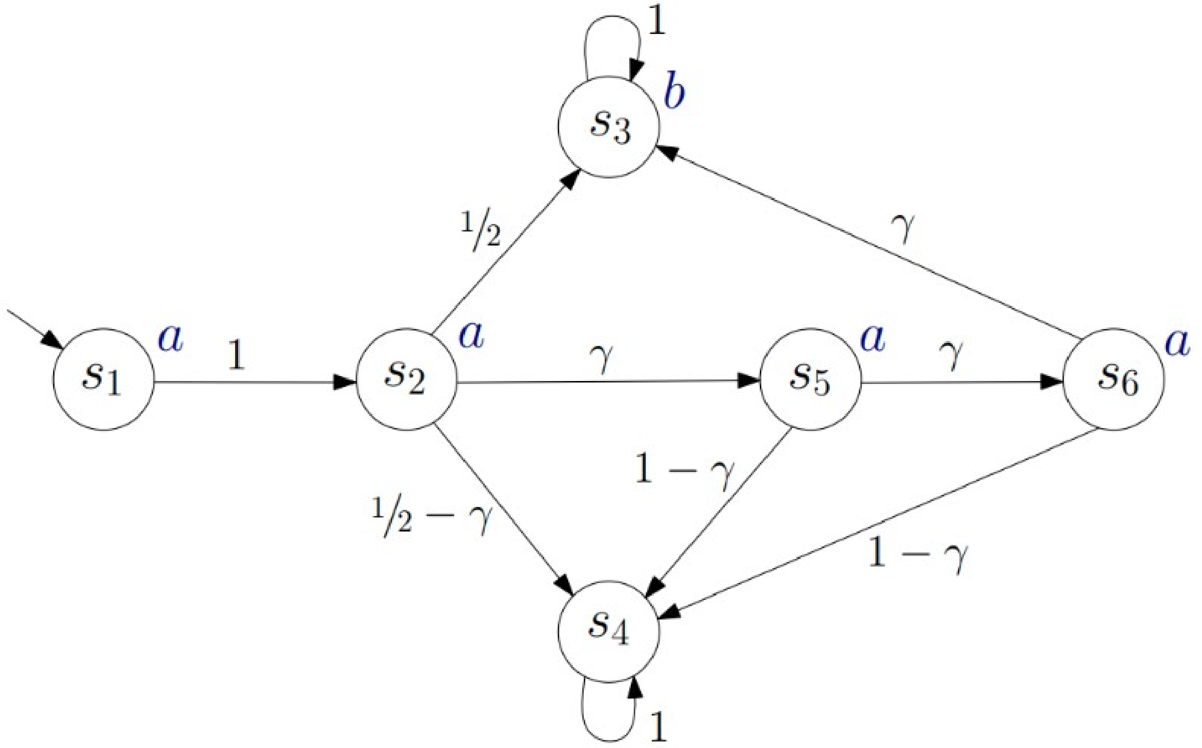
Plan

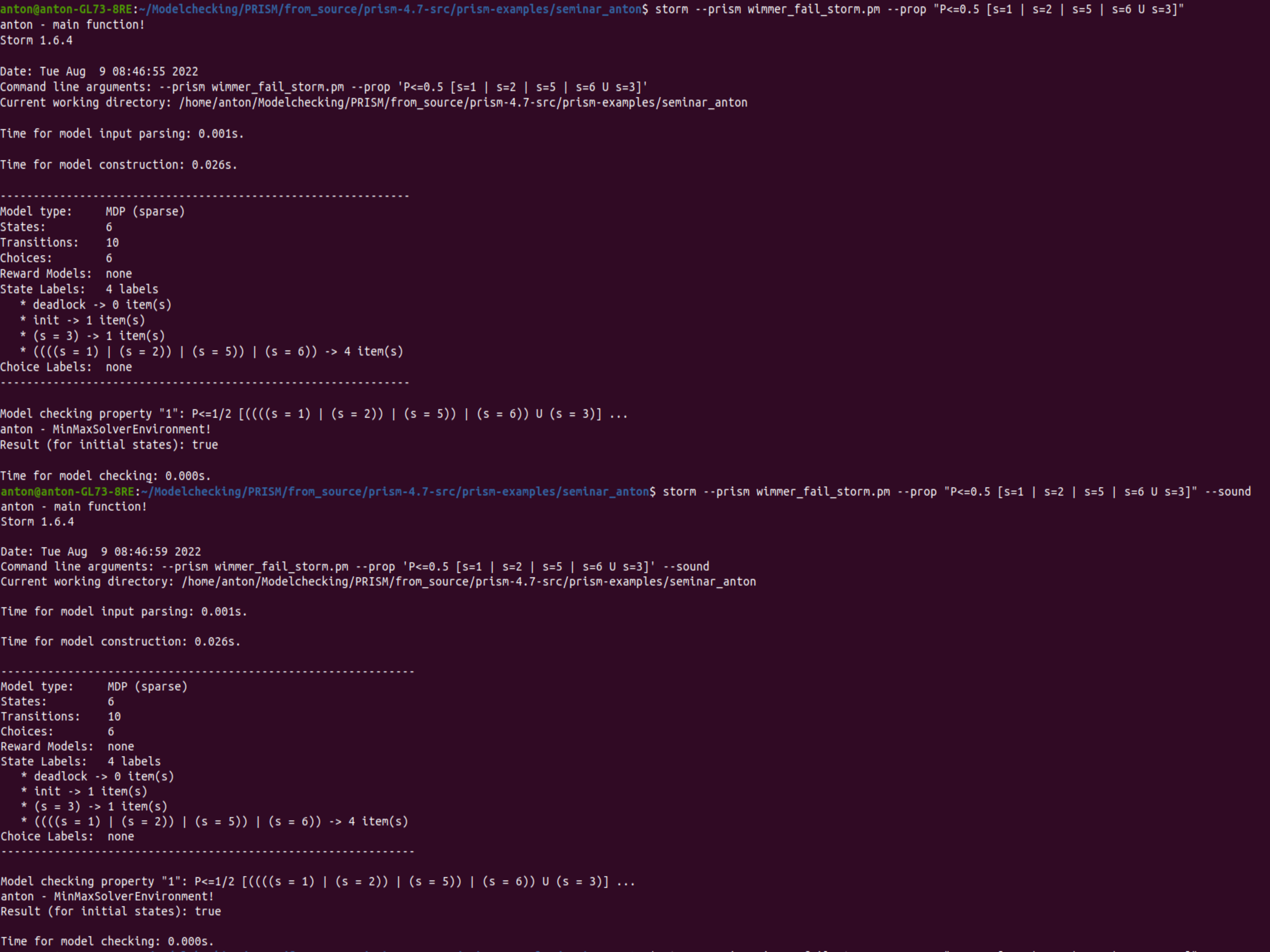
- Value Iteration can lead to wrong results in model checking
  - Examples given in the Paper
  - $\rightarrow$ Verify Examples
  -
- Solved by using Interval Iteration and controlling the Rounding mode
  - $\rightarrow$ Algorithm with rounding mode shown in the paper
  - $\rightarrow$ Present our modified code
- There are some graphics in the paper that shows results of some benchmark tests
  - $\rightarrow$ Present our results on the graphics

Example Testing with Storm

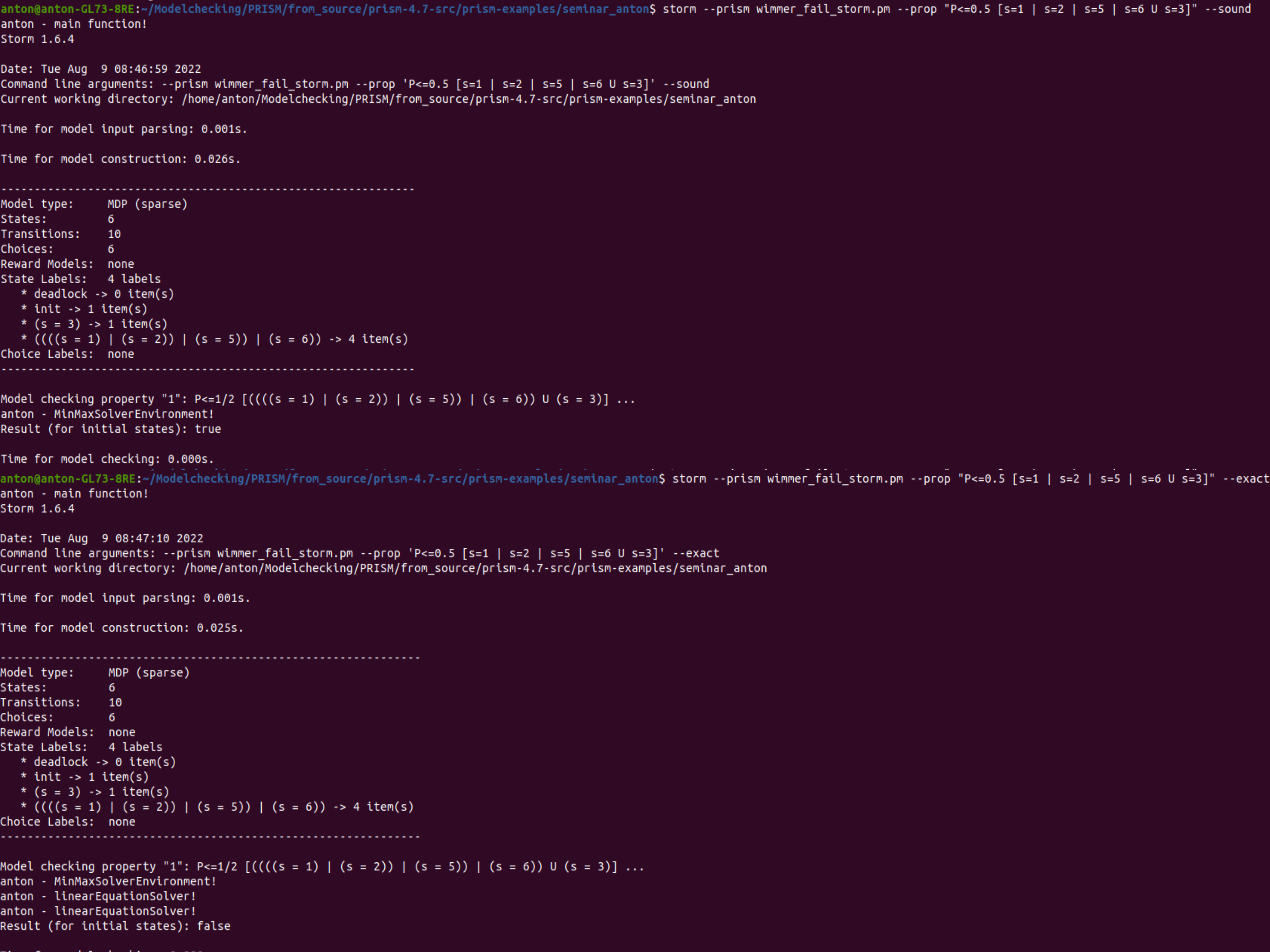
Example from „Probabilistic Model Checking and Reliability of Results“ (Wimmer et al)

- for small values of γ, the model checkers give wrong results



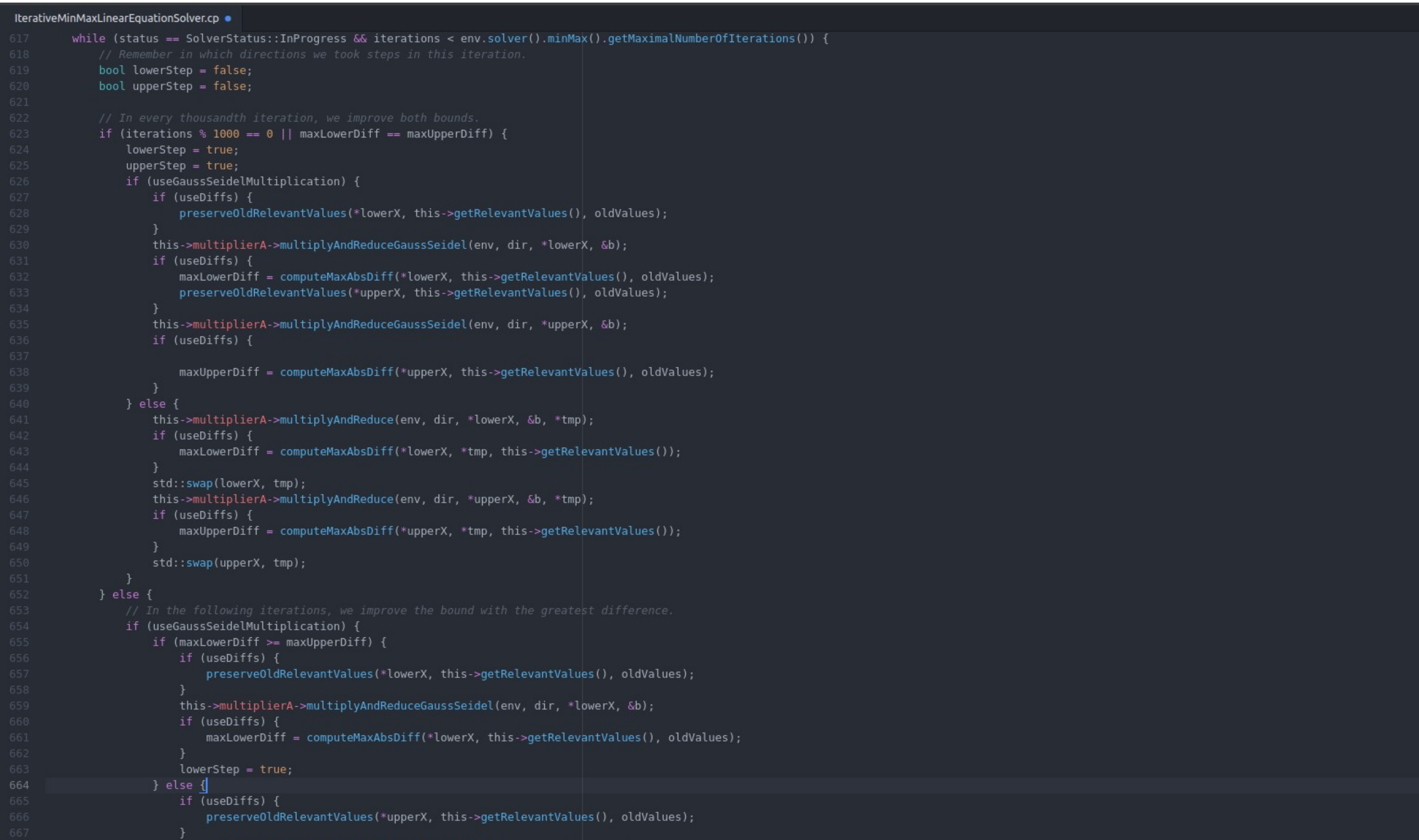


```
anton@anton-GL73-8RE:~/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton$ storm --prism wimmer_fail_storm.pm --prop "P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]"
anton - main function!
Storm 1.6.4

Date: Tue Aug  9 08:46:55 2022
Command line arguments: --prism wimmer_fail_storm.pm --prop 'P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]'
Current working directory: /home/anton/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton

Time for model input parsing: 0.001s.

Time for model construction: 0.026s.

--------------------------------------------------------------
Model type:     MDP (sparse)
States:         6
Transitions:    10
Choices:        6
Reward Models:  none
State Labels:   4 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * (s = 3) -> 1 item(s)
   * ((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) -> 4 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "1": P<=1/2 [((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) U (s = 3)] ...
anton - MinMaxSolverEnvironment!
Result (for initial states): true

Time for model checking: 0.000s.
anton@anton-GL73-8RE:~/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton$ storm --prism wimmer_fail_storm.pm --prop "P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]" --sound
anton - main function!
Storm 1.6.4

Date: Tue Aug  9 08:46:59 2022
Command line arguments: --prism wimmer_fail_storm.pm --prop 'P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]' --sound
Current working directory: /home/anton/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton

Time for model input parsing: 0.001s.

Time for model construction: 0.026s.

--------------------------------------------------------------
Model type:     MDP (sparse)
States:         6
Transitions:    10
Choices:        6
Reward Models:  none
State Labels:   4 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * (s = 3) -> 1 item(s)
   * ((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) -> 4 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "1": P<=1/2 [((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) U (s = 3)] ...
anton - MinMaxSolverEnvironment!
Result (for initial states): true

Time for model checking: 0.000s.
```



```
anton@anton-GL73-8RE:~/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton$ storm --prism wimmer_fail_storm.pm --prop "P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]" --sound
anton - main function!
Storm 1.6.4

Date: Tue Aug  9 08:46:59 2022
Command line arguments: --prism wimmer_fail_storm.pm --prop 'P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]' --sound
Current working directory: /home/anton/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton

Time for model input parsing: 0.001s.

Time for model construction: 0.026s.

--------------------------------------------------------------
Model type:     MDP (sparse)
States:         6
Transitions:    10
Choices:        6
Reward Models:  none
State Labels:   4 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * (s = 3) -> 1 item(s)
   * ((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) -> 4 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "1": P<=1/2 [((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) U (s = 3)] ...
anton - MinMaxSolverEnvironment!
Result (for initial states): true

Time for model checking: 0.000s.
anton@anton-GL73-8RE:~/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton$ storm --prism wimmer_fail_storm.pm --prop "P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]" --exact
anton - main function!
Storm 1.6.4

Date: Tue Aug  9 08:47:10 2022
Command line arguments: --prism wimmer_fail_storm.pm --prop 'P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3]' --exact
Current working directory: /home/anton/Modelchecking/PRISM/from_source/prism-4.7-src/prism-examples/seminar_anton

Time for model input parsing: 0.001s.

Time for model construction: 0.025s.

--------------------------------------------------------------
Model type:     MDP (sparse)
States:         6
Transitions:    10
Choices:        6
Reward Models:  none
State Labels:   4 labels
   * deadlock -> 0 item(s)
   * init -> 1 item(s)
   * (s = 3) -> 1 item(s)
   * ((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) -> 4 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "1": P<=1/2 [((((s = 1) | (s = 2)) | (s = 5)) | (s = 6)) U (s = 3)] ...
anton - MinMaxSolverEnvironment!
anton - linearEquationSolver!
anton - linearEquationSolver!
Result (for initial states): false
```

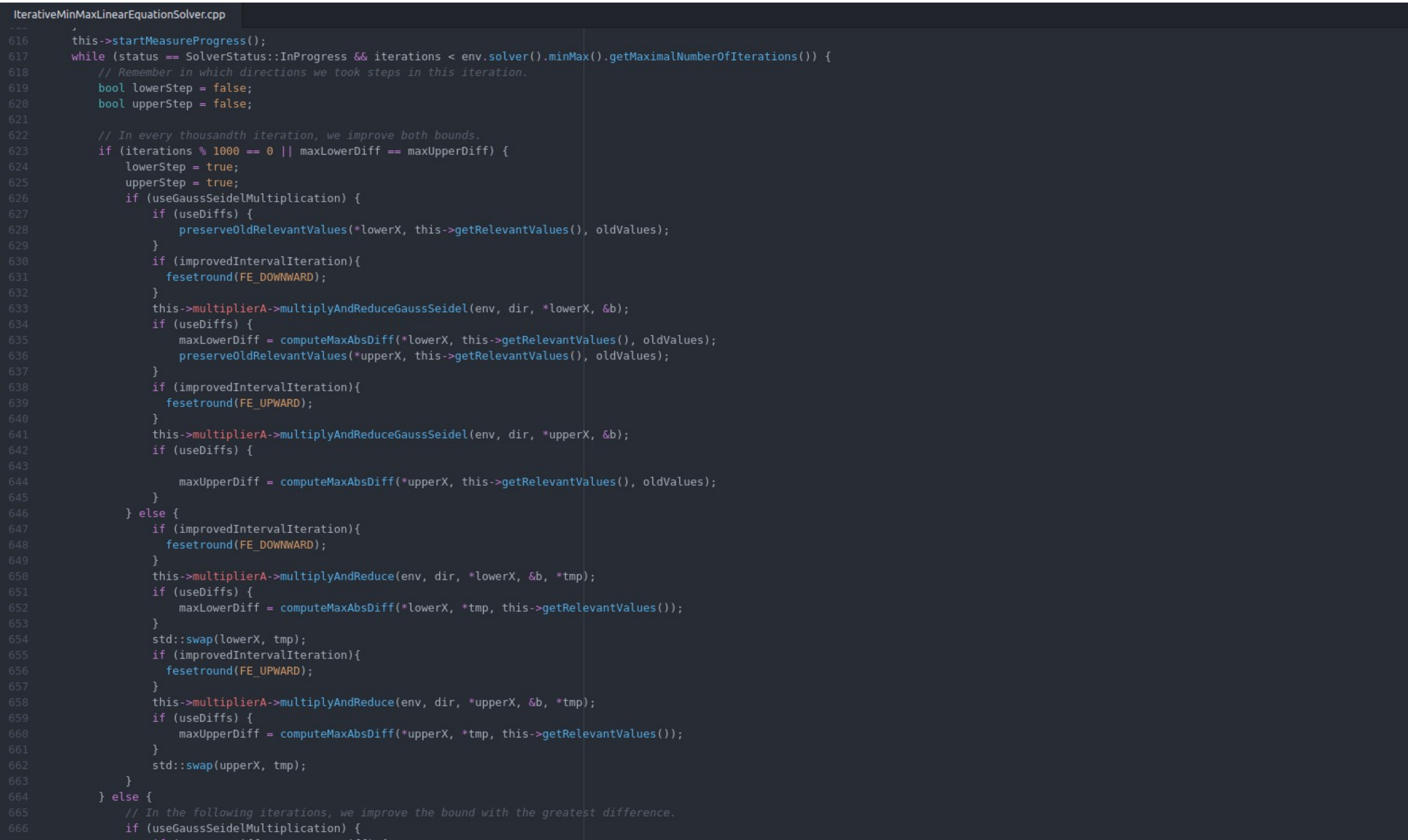
How the Interval Iteration code looks like



IterativeMinMaxLinearEquationSolver.cp

```
    while (status == SolverStatus::InProgress && iterations < env.solver().minMax().getMaximalNumberOfIterations()) {
        // Remember in which directions we took steps in this iteration.
        bool lowerStep = false;
        bool upperStep = false;

        // In every thousandth iteration, we improve both bounds.
        if (iterations % 1000 == 0 || maxLowerDiff == maxUpperDiff) {
            lowerStep = true;
            upperStep = true;
            if (useGaussSeidelMultiplication) {
                if (useDiffs) {
                    preserveOldRelevantValues(*lowerX, this->getRelevantValues(), oldValues);
                }
                this->multiplierA->multiplyAndReduceGaussSeidel(env, dir, *lowerX, &b);
                if (useDiffs) {
                    maxLowerDiff = computeMaxAbsDiff(*lowerX, this->getRelevantValues(), oldValues);
                    preserveOldRelevantValues(*upperX, this->getRelevantValues(), oldValues);
                }
                this->multiplierA->multiplyAndReduceGaussSeidel(env, dir, *upperX, &b);
                if (useDiffs) {

                    maxUpperDiff = computeMaxAbsDiff(*upperX, this->getRelevantValues(), oldValues);
                }
            } else {
                this->multiplierA->multiplyAndReduce(env, dir, *lowerX, &b, *tmp);
                if (useDiffs) {
                    maxLowerDiff = computeMaxAbsDiff(*lowerX, *tmp, this->getRelevantValues());
                }
                std::swap(lowerX, tmp);
                this->multiplierA->multiplyAndReduce(env, dir, *upperX, &b, *tmp);
                if (useDiffs) {
                    maxUpperDiff = computeMaxAbsDiff(*upperX, *tmp, this->getRelevantValues());
                }
                std::swap(upperX, tmp);
            }
        } else {
            // In the following iterations, we improve the bound with the greatest difference.
            if (useGaussSeidelMultiplication) {
                if (maxLowerDiff >= maxUpperDiff) {
                    if (useDiffs) {
                        preserveOldRelevantValues(*lowerX, this->getRelevantValues(), oldValues);
                    }
                    this->multiplierA->multiplyAndReduceGaussSeidel(env, dir, *lowerX, &b);
                    if (useDiffs) {
                        maxLowerDiff = computeMaxAbsDiff(*lowerX, this->getRelevantValues(), oldValues);
                    }
                    lowerStep = true;
                } else {
                    if (useDiffs) {
                        preserveOldRelevantValues(*upperX, this->getRelevantValues(), oldValues);
                    }
```

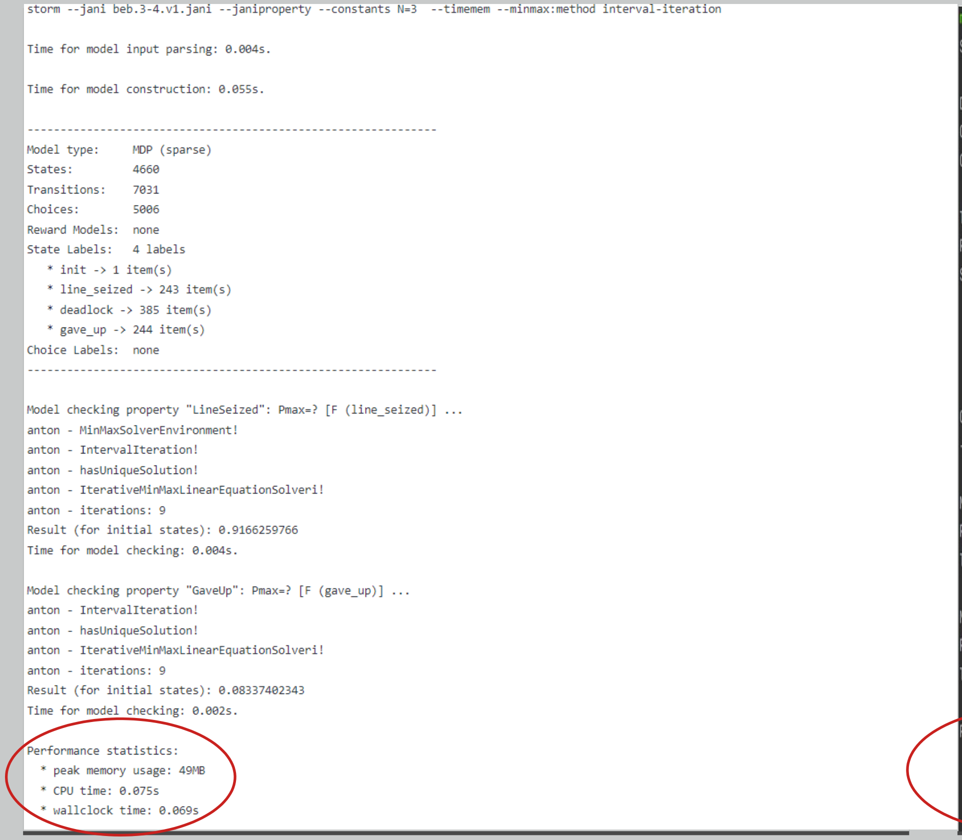
A modified Interval Iteration code



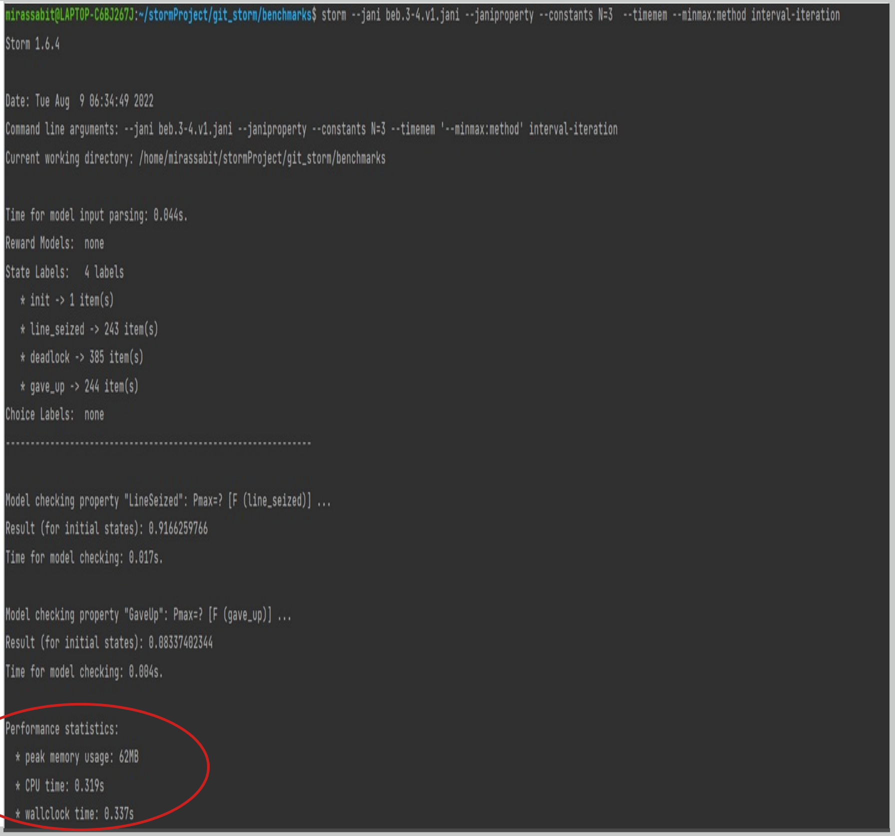
IterativeMinMaxLinearEquationSolver.cpp

```cpp
    this->startMeasureProgress();
    while (status == SolverStatus::InProgress && iterations < env.solver().minMax().getMaximalNumberOfIterations()) {
        // Remember in which directions we took steps in this iteration.
        bool lowerStep = false;
        bool upperStep = false;

        // In every thousandth iteration, we improve both bounds.
        if (iterations % 1000 == 0 || maxLowerDiff == maxUpperDiff) {
            lowerStep = true;
            upperStep = true;
            if (useGaussSeidelMultiplication) {
                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);
                }
            } else {
                if (improvedIntervalIteration){
                  fesetround(FE_DOWNWARD);
                }
                this->multiplierA->multiplyAndReduce(env, dir, *lowerX, &b, *tmp);
                if (useDiffs) {
                    maxLowerDiff = computeMaxAbsDiff(*lowerX, *tmp, this->getRelevantValues());
                }
                std::swap(lowerX, tmp);
                if (improvedIntervalIteration){
                  fesetround(FE_UPWARD);
                }
                this->multiplierA->multiplyAndReduce(env, dir, *upperX, &b, *tmp);
                if (useDiffs) {
                    maxUpperDiff = computeMaxAbsDiff(*upperX, *tmp, this->getRelevantValues());
                }
                std::swap(upperX, tmp);
            }
        } else {
            // In the following iterations, we improve the bound with the greatest difference.
            if (useGaussSeidelMultiplication) {
```

Results under differenct circumstances

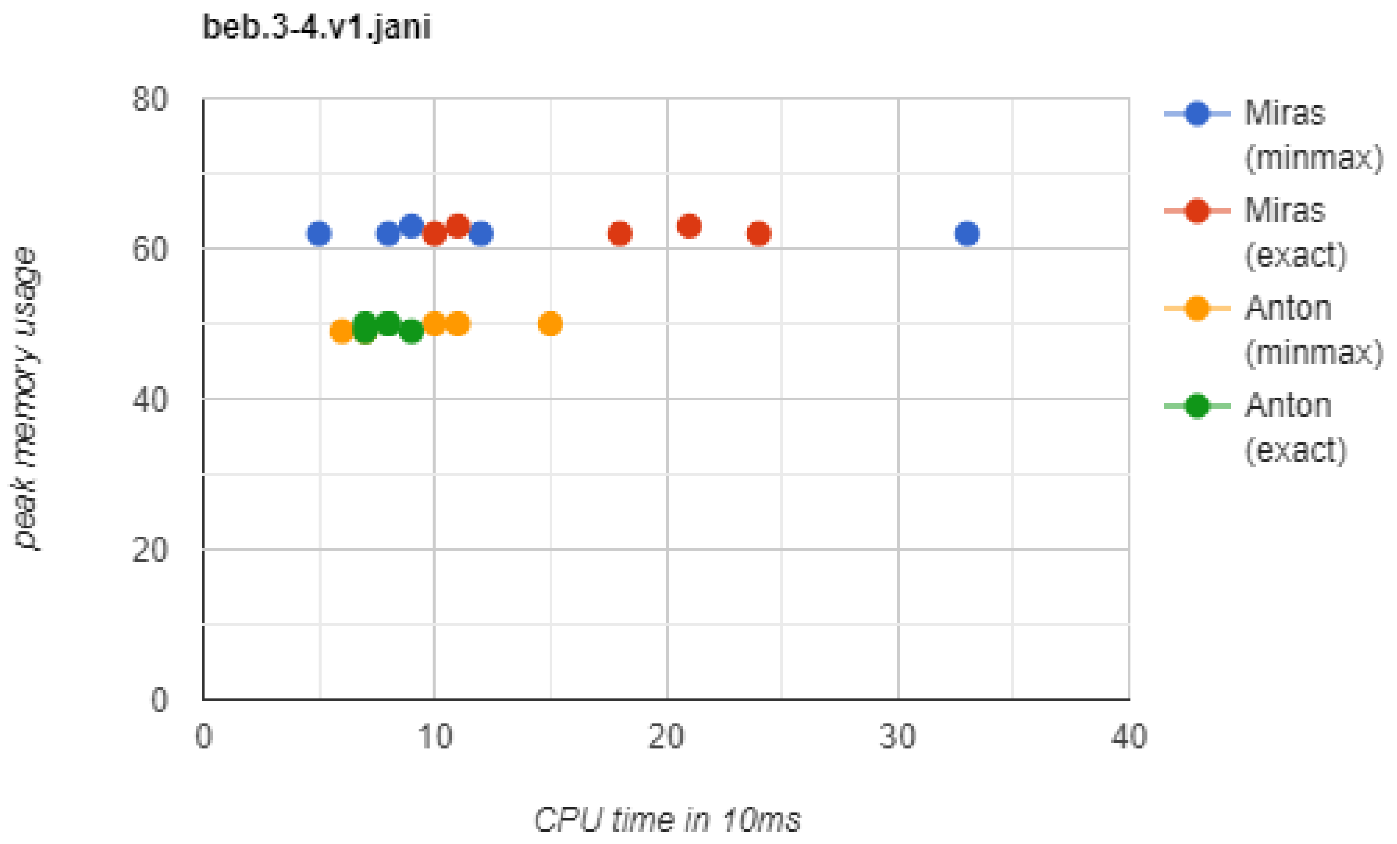


```
storm --jani beb.3-4.v1.jani --janiproperty --constants N=3  --timemem --minmax:method interval-iteration

Time for model input parsing: 0.004s.

Time for model construction: 0.055s.

--------------------------------------------------------------
Model type:     MDP (sparse)
States:         4660
Transitions:    7031
Choices:        5006
Reward Models:  none
State Labels:   4 labels
   * init -> 1 item(s)
   * line_seized -> 243 item(s)
   * deadlock -> 385 item(s)
   * gave_up -> 244 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "LineSeized": Pmax=? [F (line_seized)] ...
anton - MinMaxSolverEnvironment!
anton - IntervalIteration!
anton - hasUniqueSolution!
anton - IterativeMinMaxLinearEquationSolveri!
anton - iterations: 9
Result (for initial states): 0.9166259766
Time for model checking: 0.004s.

Model checking property "GaveUp": Pmax=? [F (gave_up)] ...
anton - IntervalIteration!
anton - hasUniqueSolution!
anton - IterativeMinMaxLinearEquationSolveri!
anton - iterations: 9
Result (for initial states): 0.08337402343
Time for model checking: 0.002s.

Performance statistics:
  * peak memory usage: 49MB
  * CPU time: 0.075s
  * wallclock time: 0.069s
```

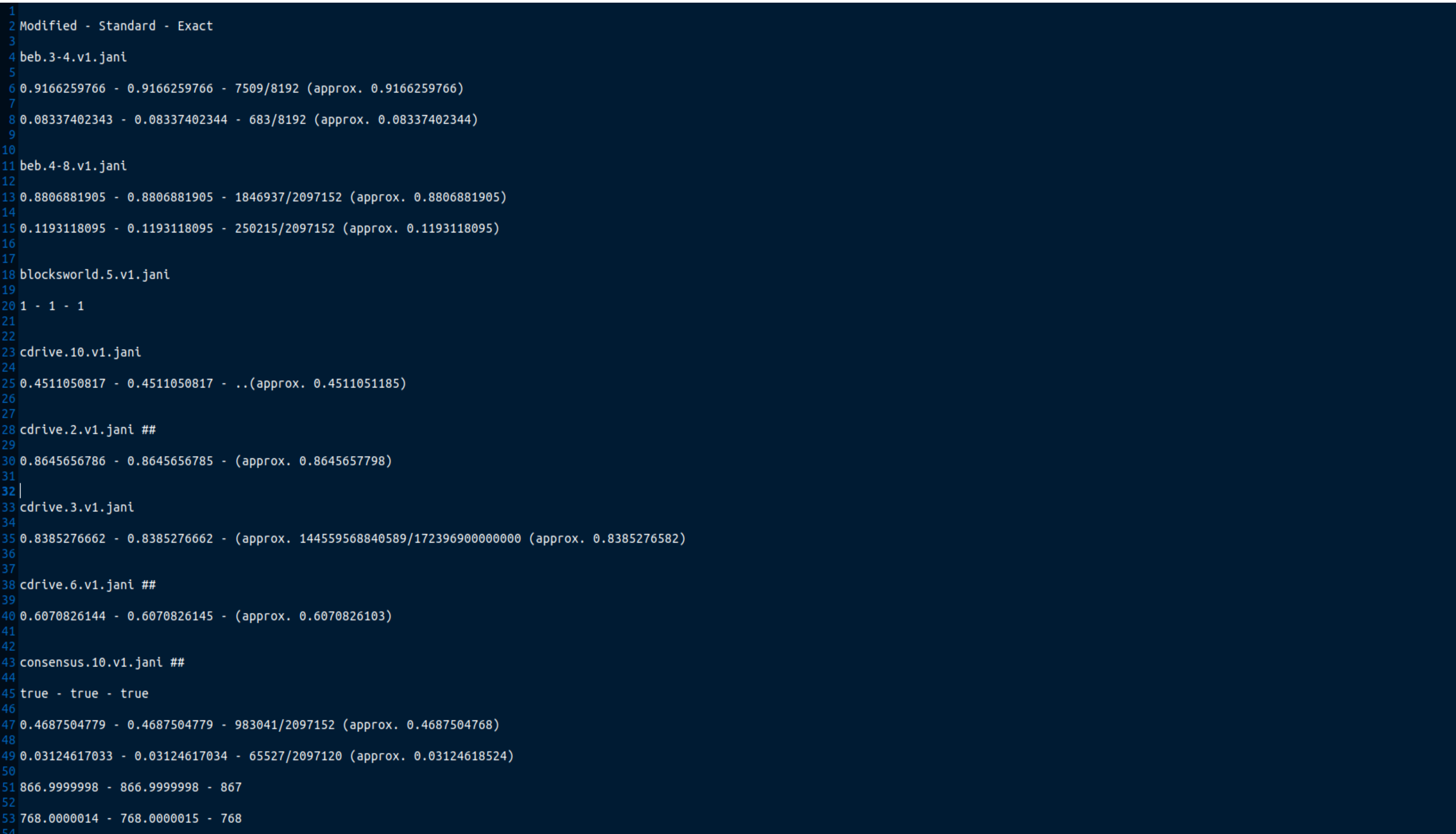


```
mirassabit@LAPTOP-C6BJ267J:~/stormProject/git_storm/benchmarks$ storm --jani beb.3-4.v1.jani --janiproperty --constants N=3  --timemem --minmax:method interval-iteration
Storm 1.6.4

Date: Tue Aug  9 06:34:49 2022
Command line arguments: --jani beb.3-4.v1.jani --janiproperty --constants N=3 --timemem '--minmax:method' interval-iteration
Current working directory: /home/mirassabit/stormProject/git_storm/benchmarks

Time for model input parsing: 0.044s.
Reward Models:  none
State Labels:   4 labels
   * init -> 1 item(s)
   * line_seized -> 243 item(s)
   * deadlock -> 385 item(s)
   * gave_up -> 244 item(s)
Choice Labels:  none
--------------------------------------------------------------

Model checking property "LineSeized": Pmax=? [F (line_seized)] ...
Result (for initial states): 0.9166259766
Time for model checking: 0.017s.

Model checking property "GaveUp": Pmax=? [F (gave_up)] ...
Result (for initial states): 0.08337402344
Time for model checking: 0.004s.

Performance statistics:
  * peak memory usage: 62MB
  * CPU time: 0.319s
  * wallclock time: 0.337s
```

Results under differenct circumstances

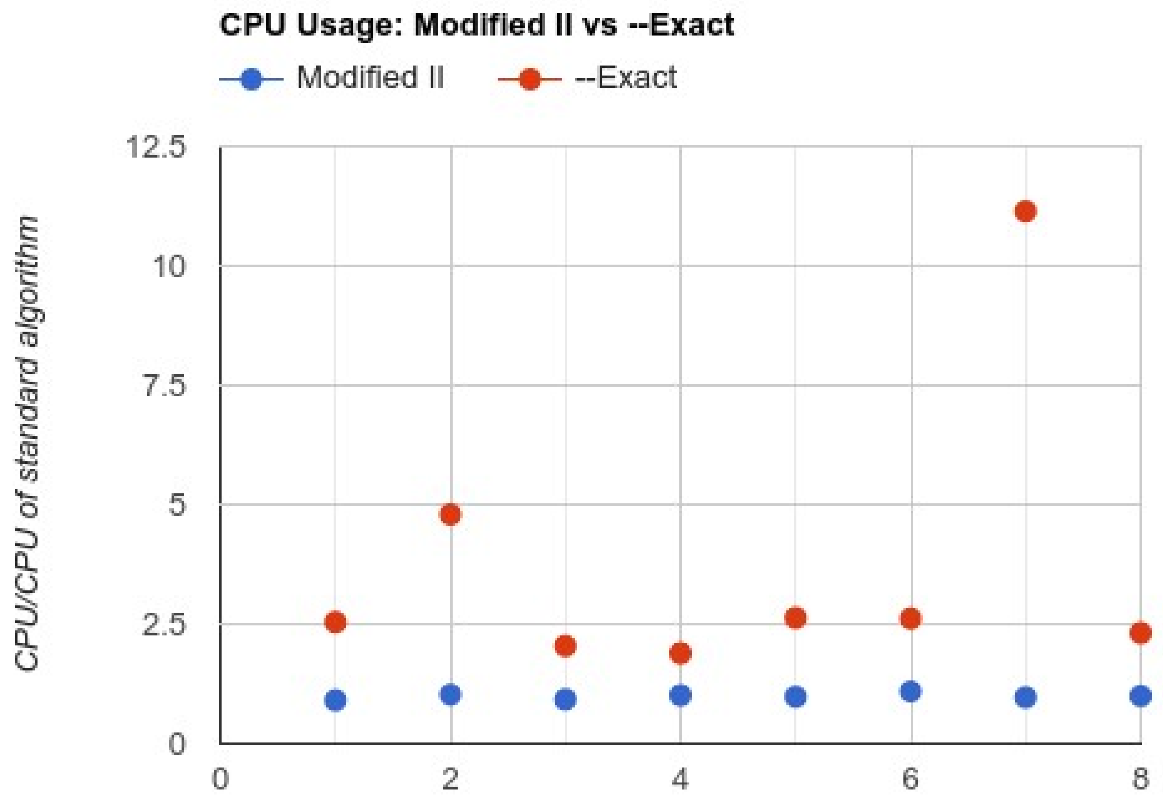


Results for Modified – Standard - Exact

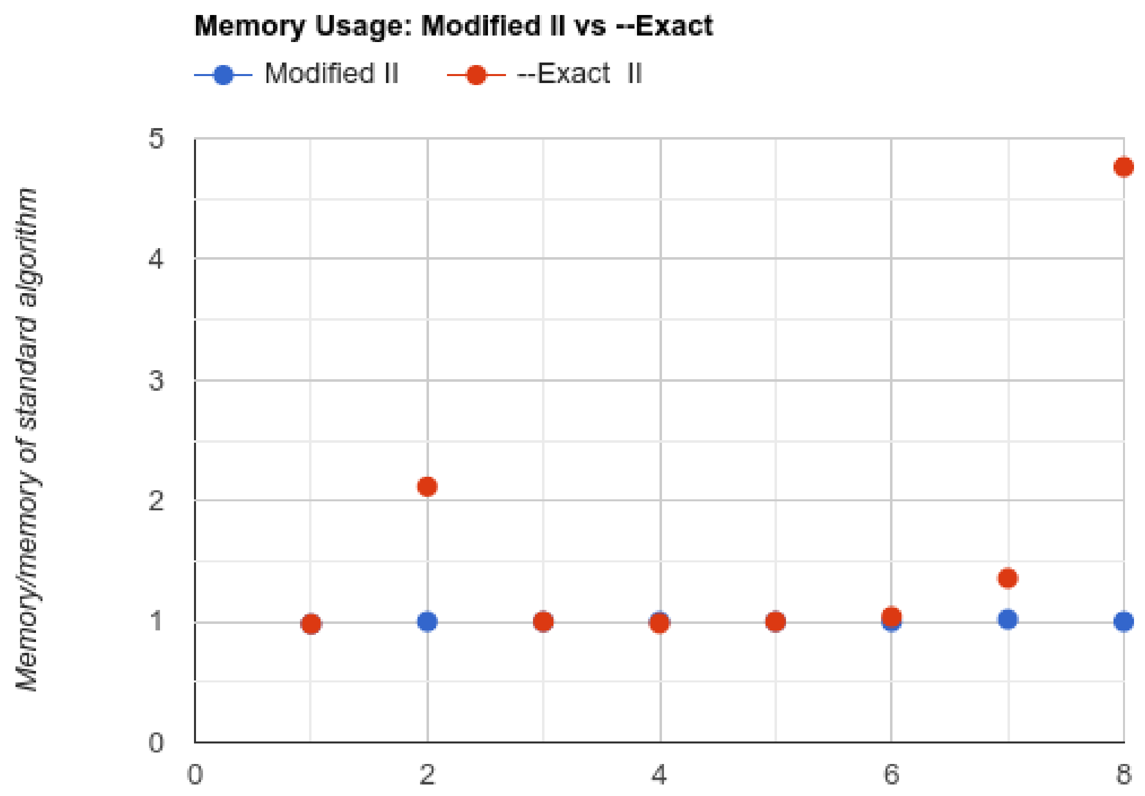


```

Modified - Standard - Exact

beb.3-4.v1.jani

0.9166259766 - 0.9166259766 - 7509/8192 (approx. 0.9166259766)

0.08337402343 - 0.08337402344 - 683/8192 (approx. 0.08337402344)


beb.4-8.v1.jani

0.8806881905 - 0.8806881905 - 1846937/2097152 (approx. 0.8806881905)

0.1193118095 - 0.1193118095 - 250215/2097152 (approx. 0.1193118095)


blocksworld.5.v1.jani

1 - 1 - 1


cdrive.10.v1.jani

0.4511050817 - 0.4511050817 - ..(approx. 0.4511051185)


cdrive.2.v1.jani ##

0.8645656786 - 0.8645656785 - (approx. 0.8645657798)


cdrive.3.v1.jani

0.8385276662 - 0.8385276662 - (approx. 144559568840589/172396900000000 (approx. 0.8385276582)


cdrive.6.v1.jani ##

0.6070826144 - 0.6070826145 - (approx. 0.6070826103)


consensus.10.v1.jani ##

true - true - true

0.4687504779 - 0.4687504779 - 983041/2097152 (approx. 0.4687504768)

0.03124617033 - 0.03124617034 - 65527/2097120 (approx. 0.03124618524)

866.9999998 - 866.9999998 - 867

768.0000014 - 768.0000015 - 768
```

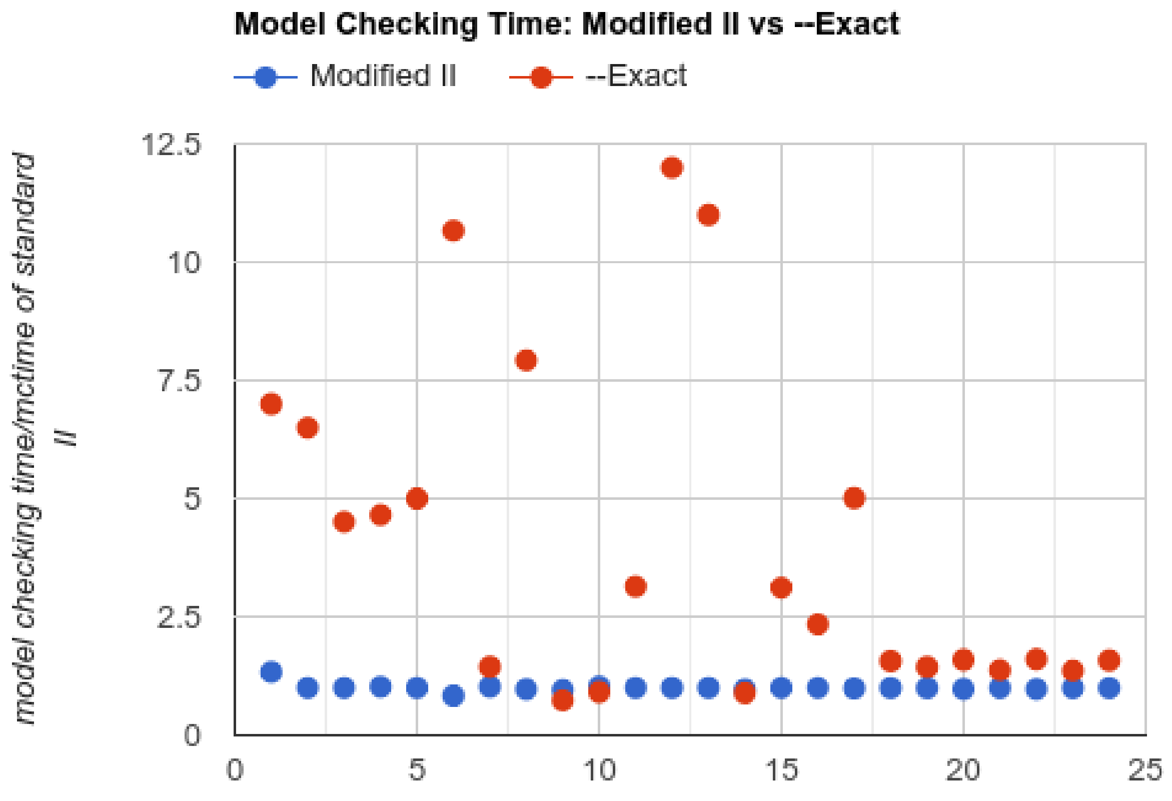
Comparison between Modified Interval Iteration and ‘--exact’ option



Comparison between Modified Interval Iteration and ‘--exact’ option



Comparison between Modified Interval Iteration and ‘--exact’ option



Open Questions – Future Work

- In which cases is the modified Interval Iteration (with better Rounding) relevant ? Are there underlying structural properties in the model that make a difference ?
  → only small really relevant in the benchmarks

- Test it on examples that are more likely to fail due to rounding (eg. results close to 0)
  → create own tests

- QoL Improvement to algorithm – make it possible for both modified and standard Interval Iteration to be run at the same time

- Which option should be used ? Modified Interval Iteration or Exact ?
  → Modified II is a small improvement with almost not downside
  → Exact is more precise, but often much longer runtime in comparison to Modified II