

# Artefact of "ClauseSMT: Clause Level NLSAT for Nonlinear Real Arithmetic"

---

Author: Zhonghan Wang ([wangzh@ios.ac.cn](mailto:wangzh@ios.ac.cn))

## Structure of the Artefact

---

1. **binary\_solvers**: The pre-compiled binary files of different versions of our solver and other existing solvers.
2. **experiment\_data**: Our experimental results on the QF\_NRA benchmark.
3. **script**: The scripts for generating the file path list and running the experiments.
4. **source\_code**: The source code of different versions of our solver.
5. **paper.pdf**: The research paper.
6. **README**: This file.
7. **LICENSE.txt**: The license of the artefact.

## 0. Environment Dependencies

---

We recommend the user to conduct the experiment on **Linux or Windows WSL**.

To fully reproduce our experimental results, we highly recommend the user to get an external server for the parallel computing usage.

Besides, several dependencies shown below are required:

1. AR archiving tool
2. Python
3. G++, or other C++ compilers like Clang

## 1. Download Benchmark

---

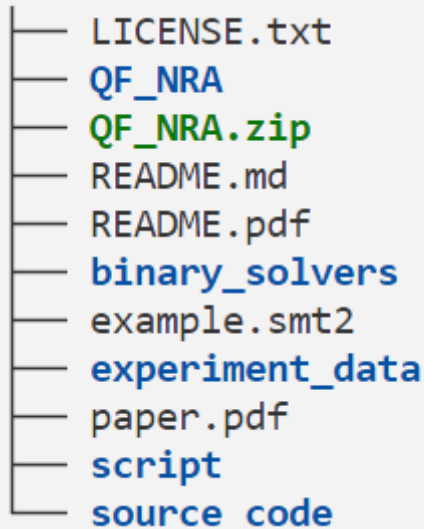
We provide the benchmark in our Google drive link

<https://drive.google.com/file/d/1Nc7IQEeeTFXFdrVp6pKLkMbww7ISc99/view?usp=sharing>

One should unzip the file to get the `QF_NRA` directory.

```
unzip QF_NRA.zip
```

After this step, the folder structure should look like this:



- LICENSE.txt
- QF\_NRA
- QF\_NRA.zip
- README.md
- README.pdf
- binary\_solvers
- example.smt2
- experiment\_data
- paper.pdf
- script
- source\_code

## 2. Generate File Path List

---

To fully evaluate our solver on the whole benchmark, we should generate the absolute paths for all instances.

```
cd script
python generate_list.py ../QF_NRA/
```

The generated absolute paths are stored in `QF_NRA/list.txt` with 12134 lines.

```
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E1.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E11.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E12.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E13.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E14.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E15.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E16.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E17.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E18.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E19.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E20.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E21.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E22.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E23.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E24.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E25.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E26.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E27.smt2
/home/wangzh/ClauseSMT/QF_NRA/20161105-Sturm-MBO/mbo_E10E28.smt2
```

## 3. Compilation and Binary Files

---

There are two ways to use our smt solver, either compile the source code or just use the binary files.

## 3.1 Compilation from Source Code

To compile the source code, we provide a script `script/mk_make.py` to generate the makefile and compile the source code.

```
cd source_code/clauseSMT
python scripts/mk_make.py
cd build
make -j <thread_num>
```

After this, a binary file `z3` will be generated in the `build` directory.

## 3.2 Binary Files

We also provide the pre-compiled binary files of our solver and other SMT solvers in `binary_solvers/`.

## 3.3 Test the Binary Files

We provide a simple smt instance `example.smt2` for testing the binary files.

### 1. Compiled Binary Files

```
./source_code/clauseSMT/build/z3 example.smt2
```

### 2. Pre-compiled Binary Files

```
./binary_solvers/clauseSMT example.smt2
```

## 4. Run clauseSMT on the Benchmark

To fully get the experimental results, we provide a script `script/parallel_run.cpp` to run our solver on all instances in the benchmark with multiple threads. We strongly recommend the user to **use an external server** for the parallel computing usage.

```
mkdir self_data
cd script
g++ -O3 -o parallel_run parallel_run.cpp
./parallel_run [instance_list_path] [solver_path] [output_path] [time_limit]
[memory_limit] [max_process_num]
```

where `instance_list_path` is the path to the list file of test cases, `solver_path` is the path to the solver binary file, `output_path` is the path to collect the results, `time_limit` is the time limit (seconds) for each instance, `memory_limit` is the memory limit (MB) for each instance, and `max_process_num` is the maximum number of processes to run in parallel.

A recommended setting for the external server is:

```
./parallel_run ../QF_NRA/list.txt ../source_code/clauseSMT/build/z3 ../self_data/
1200 30720 120
```

or

```
./parallel_run ../QF_NRA/list.txt ../binary_solvers/claueSMT ../self_data/ 1200
30720 120
```

if you would like to use the pre-compiled binary files.

In SMT-COMP, the standard timelimit is 1200 seconds and the memory limit is 30720 MB for each instance.

If the user can not get an advanced external server and would like to narrow the test set, please manually modify the `instance_list_path` to a smaller list files containing a subset of the benchmark.

If everything goes well, the `self_data` folder would contain txt files for each instance, each containing the solver's output and statistics.

```
├── sqrt-problem-12vars3-chunk-0087.txt
├── sqrt-problem-12vars3-chunk-0088.txt
├── sqrt-problem-12vars3-chunk-0089.txt
└── sqrt-problem-12vars3-chunk-0091.txt
```

## 5. Collecting Results

We provide a script `script/collect.py` to collect the results of all solvers on all test cases, and generate a csv file. The script takes two arguments:

- **folder\_path:** the path to the folder containing the results (for example: `../self_data/`)
- **output\_file:** the path to the output csv file (for example: `csv/claueSMT.csv`)

```
mkdir csv
cd script
python collect.py ../self_data/ csv/claueSMT.csv
```

Then the `csv/claueSMT.csv` file would contain the results of our solver on all test cases, including the instance name, solver's output, time and memory consumption.

Benchmark	Result	Time	Memory
sqrt-problem-12vars3-chunk-0087	sat	0.0	17.15
sqrt-problem-12vars3-chunk-0088	unsat	0.01	17.14
sqrt-problem-12vars3-chunk-0091	sat	0.01	17.15
sart-problem-12vars3-chunk-0089	unsat	0.0	17.15

## 6. Comparison with our results

We provide the experimental results of our solver on the whole benchmark in `experiment_data/`. The user can compare their results with ours to see the performance of our solver.

## 7. Other SMT Solvers

We also provide the source code and binary files of other SMT solvers, including NLSAT, Z3, CVC5, Yices2, dReal, and MathSAT. The user can use these solvers to compare their results with ours, following the steps in the previous sections.

# Appendix: Description of Solvers and Experimental Results

## Comparison with Existing SMT Solvers

Solver	Path	Data	Usage	Sat	Unsat	Solved
NLSAT	<a href="#">NLSAT</a>	<a href="#">NLSAT result</a>	./NLSAT <*.smt2>	5541	5191	10732
Z3	<a href="#">z3</a>	<a href="#">z3 result</a>	./z3 <*.smt2>	5569	5379	10948
CVC5	<a href="#">cvc5</a>	<a href="#">cvc5 result</a>	./cvc5 <*.smt2>	5475	5809	11284
Yices2	<a href="#">yices2</a>	<a href="#">yices2 result</a>	./yices2 <*.smt2>	5372	5612	10984
dReal (delta=0.001)	<a href="#">dReal</a>	<a href="#">dReal result</a>	./dReal -- precision 0.001 <*.smt2>	4811	4294	9105
MathSAT	<a href="#">mathsat</a>	<a href="#">mathsat result</a>	./mathsat <*.smt2>	2772	4583	7355
clauseSMT (Ours)	<a href="#">clauseSMT</a>	<a href="#">clauseSMT result</a>	./clauseSMT <*.smt2>	5608	5397	11005

## Effect of Proposed Techniques (Ablation Study)

### Effect of Look-Ahead Mechanism

Solver	Description	Path	Data	Usage	Sat	Unsat	Solved
NLSAT	Decide Lowest Degree Literal	<a href="#">NLSAT</a>	<a href="#">NLSAT result</a>	./NLSAT <*.smt2> -st	5541	5191	10732
random_decide	Decide Random Literal	<a href="#">random_decide</a>	<a href="#">random_decide result</a>	./random_decide <*.smt2> -st	5505	5147	10652
static-look-ahead	Feasible-set based Look-Ahead	<a href="#">static-look-ahead</a>	<a href="#">static-look-ahead result</a>	./static-look-ahead <*.smt2> -st	5555	5223	10778

### Effect of Clause-Level Propagation based Branching Heuristic

Solver	Description	Path	Data	Usage	Sat	Unsat	Solved
static-look-ahead	Static order based on degree	<a href="#">static-look-ahead</a>	<a href="#">static-look-ahead result</a>	./static-look-ahead <*.smt2> -st	5555	5223	10778
vsids-look-ahead	Dynamic order based on VSIDS	<a href="#">vsids-look-ahead</a>	<a href="#">vsids-look-ahead result</a>	./vsids-look-ahead <*.smt2> -st	5599	5321	10920
clauseSMT (Ours)	Dynamic order based on clause-level propagation	<a href="#">clauseSMT</a>	<a href="#">clauseSMT result</a>	./clauseSMT <*.smt2> -st	5608	5397	11005