CHC-COMP 2022: Competition Report

Emanuele De Angelis[*]
IASI-CNR, Italy
emanuele.deangelis@iasi.cnr.it

Hari Govind V K
University of Waterloo, Canada
hgvk94@gmail.com

CHC-COMP 2022 is the fifth edition of the competition of solvers for Constrained Horn Clauses. The competition was run in March 2022; the results were presented at the 9th Workshop on Horn Clauses for Verification and Synthesis held in Munich, Germany, on April 3, 2022. This edition featured six solvers, and eight tracks consisting of sets of linear and nonlinear clauses with constraints over linear integer arithmetic, linear real arithmetic, arrays, and algebraic data types. This report provides an overview of the organization behind the competition runs: it includes the technical details of the competition setup as well as presenting the results of the 2022 edition.

1 Introduction

Constrained Horn Clauses (CHCs, for short) are a class of first-order logic formulas where the Horn clause format is extended with *constraints*, that is, formulas of an arbitrary, possibly non-Horn, background theory (such as linear integer arithmetic, arrays, and algebraic data types).

CHCs have been advocated by many researchers as suitable framework to model software systems and to reason about their properties (see, for instance, [2, 4, 8]). In particular, CHCs turn out to be a very flexible and expressive intermediate language to formalize features of different programming and specification languages as well as proof systems used in the development of software systems, thereby making solvers for CHCs (CHC solvers, for short) complementary tools that can be conveniently used to automate the analysis and verification pipeline of software systems.

CHC-COMP 2022[1] is the 5th edition of the competition of solvers for CHCs, affiliated with the 9th Workshop on Horn Clauses for Verification and Synthesis (HCVS 2022[2]) held in Munich, Germany, on April 3, 2022. The goal of the CHC-COMP series is to evaluate the effectiveness and the efficiency of state-of-the-art solvers for CHCs on realistic and publicly available benchmarks.

CHC-COMP is open to proposals for new competition tracks and anyone is welcome to submit candidate benchmarks. The CHC-COMP 2022 deadline for submitting benchmarks to be considered for the competition was March 1, 2022. The deadline for submitting the solvers for test runs (optional, but recommended) was March 8, 2022, while the deadline for submitting the solvers for evaluation was March 18, 2022 (extended to March 21, 2022). The competition was run in the subsequent two weeks, and the results were announced at the HCVS workshop on April 3, 2022.

CHC-COMP 2022 featured 6 solvers, and 8 tracks consisting of sets of linear and nonlinear clauses with constraints over linear integer arithmetic, linear real arithmetic, arrays, algebraic data types, and a few combinations of such theories.

This report is structured as follows. Section 2 presents the competition tracks, the technical resources used to run the competition, and the evaluation model adopted to rank the solvers. Section 3 presents

[*]The author is member of the INdAM Research Group GNCS.
[1]The CHC-COMP 2022 webpage is available at https://chc-comp.github.io/
[2]The HCVS 2022 webpage is available at https://www.sci.unich.it/hcvs22/.

G.W. Hamilton, T. Kahsai, M. Proietti (Eds.):
Joint International Workshops on
Horn Clauses for Verification and Synthesis (HCVS 2022)
Verification and Program Transformation (VPT 2022)
EPTCS 373, 2022, pp. 44–62, doi:10.4204/EPTCS.373.5

the benchmark suite, specifically, the format, the inventory, and how the candidate benchmarks have been processed and selected for the competition runs. Section 4 presents the tools that were submitted to CHC-COMP 2022. Section 5 presents the results of this edition. Section 6 presents a few concluding remarks from the organizers, which also include the suggestions and observations from the participants. Section 7 collects the tool descriptions contributed by the participants. Finally, Appendix A includes the tables with the detailed results about the competition runs.

Acknowledgements

We would like to thank the HCVS 2022 Program Chairs, Temesghen Kahsai and Maurizio Proietti, for hosting the competition this year as well.

CHC-COMP 2022 heavily built on the infrastructure developed by the Organizers of the previous editions, that is, Grigory Fedyukovich, Arie Gurfinkel, and Philipp Rümmer, which also includes the contributions from Nikolaj Bjørner, Adrien Champion, and Dejan Jovanovic.

We are deeply indebted to StarExec [20] that continues to provide the CHC-COMP community the computing resources and evaluation environment to run the competition. In particular, we would like to thank Aaron Stump for his indispensable support with StarExec.

We are also extremely grateful to Philipp Rümmer who kindly helped us by answering our numerous questions about the design and the organization of the competition.

Finally, we would to thank all the CHC-COMP 2022 participants for having submitted their solvers, as well as all who contributed with new benchmarks, and all the HCVS 2022 participants for the profitable discussion we had at the workshop after the presentation of the results.

2 Design and Organization

This section presents (i) the competition tracks, (ii) the technical resources used to run the solvers, and (iii) the evaluation system used to rank the solvers in each track.

2.1 Tracks

Solvers participating in the CHC-COMP 2022 could enter the competition in eight tracks: one track was introduced in this edition, that is, LIA-nonlin-Arrays-nonrecADT, while the remaining tracks were inherited from the previous editions.

The tracks are classified according to type of clauses included in the corresponding benchmarks. In particular, the categories have been defined by considering the following features: (i) the background theory of the constraints, and (ii) the structure of the clauses, that is, linear clauses (clauses with at most one uninterpreted atom in the premise of the clause), and nonlinear clauses (clauses with more than one uninterpreted atom in the premise of the clause).

We have considered the following tracks, which includes benchmarks of linear and nonlinear clauses combining various background theories.

- **LIA-lin**: Linear Integer Arithmetic – linear clauses.
- **LIA-nonlin**: Linear Integer Arithmetic – nonlinear clauses.
- **LIA-lin-Arrays**: Linear Integer Arithmetic and Arrays – linear clauses.
- **LIA-nonlin-Arrays**: Linear Integer Arithmetic and Arrays – nonlinear clauses.

- **ADT-nonlin**: Algebraic Data Types – nonlinear clauses.
- **LIA-nonlin-Arrays-nonrecADT**: Linear Integer Arithmetic, Arrays and nonrecursive Algebraic Data Types – nonlinear clauses.

Moreover, we have also considered two additional tracks including a syntactically restricted class of linear clauses, that is, transition systems. Benchmarks in this track have exactly one uninterpreted relation symbol, and exactly three linear clauses encoding initial states, transitions, and error states.

- **LRA-TS**: Linear Real Arithmetic – Transition Systems.
- **LRA-TS-par**: Linear Real Arithmetic – Transition Systems – parallel.

2.2 Technical Resources

CHC-COMP 2022 was run on StarExec (`https://www.starexec.org/`) using the same technical resources used in the 2021 edition [7]. For the sake of completeness, we summarize the details of technical resources available to run the competition.

StarExec made available to CHC-COMP 2022 two queues: `chcpar.q` and `chcseq.q`, consisting of 15 and 20 nodes, respectively, each of which equipped with two quadcore CPUs. The machine specifications (see `https://www.starexec.org/starexec/public/machine-specs.txt`) are:

```
# Starexec stats nodes 001 - 192:
Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz (2393 MHZ)
    10240  KB Cache
    263932744 kB main memory

# Software:
OS:       CentOS Linux release 7.7.1908 (Core)
kernel:   3.10.0-1062.4.3.el7.x86_64
glibc:    glibc-2.17-292.el7.x86_64
          gcc-4.8.5-39.el7.x86_64
          glibc-2.17-292.el7.i686
```

Running a solver on a track was performed by submitting a job to a StarExec node. A job is a pair consisting of a solver (with a track-specific configuration) and a benchmark.

Each job of LRA-TS-par was run on a node of the `chcpar.q` queue, while two jobs of all other tracks were run in parallel on a node of the the `chcseq.q` queue.

2.3 Test and Competition Runs

Similar to CHC-COMP 2021 [7], the submitted solvers were evaluated twice by performing a **test** run and a **competition** run. For the sake of completeness, in the following we summarize the main features of the two kinds of runs (see also the report [7] of the 2021 edition).

In the **test** runs, the (optional) pre-submissions of the solvers were evaluated to check their configurations and identify possible inconsistencies. In these tests a small set of randomly selected benchmarks was used, and each job was limited to 600s CPU time, 600s wall-clock time, and 64GB memory.

In the **competition** runs, the final submissions of the solvers were evaluated to determine the outcome of the competition, that is, to rank the solvers entering CHC-COMP 2022. The ranking method is presented in Section 2.4, while the process for selecting the benchmarks is described in Section 3.5. In the competition runs of LRA-TS-par each job was limited to 1800s wall-clock time, and 64GB memory. For all other tracks, each job was limited to 1800s CPU time, 1800s wall-clock time, and 64GB memory.

2.4 Evaluation of the Competition Runs

The competing solvers were evaluated using the same approach as the 2021 edition [7].

The evaluation of the competition runs were done using the `summarize.py` script available at `https://github.com/chc-comp/scripts`; the script takes as input the 'job information' CSV file produced by StarExec at job completion.

The ranking of solvers in each track is based on the **Score** obtained by the solvers in the competition run for a track. The **Score** is computed on the basis of the **result** provided by the solver on the benchmarks for that track. The **result** can be *sat*, *unsat*, or *unknown* (which includes solvers giving up, running out of resources, or crashing), and the **Score** is given by the number of *sat* or *unsat* results.

If two solvers reached an equal **Score**, the ranking was determined by using the **CPU time** for all tracks except the LRA-TS-par track, where the **Wall-clock time** is used instead of the **CPU time**. The **CPU time** is the total CPU time needed by a solver to produce a **result** in some track; the **Wall-clock time** is the total wall-clock time needed by a solver to produce its answers in some track.

The tables in Appendix A also report in column '#unique' the number of *sat* or *unsat* results produced by a solver for benchmarks for which all other solvers returned *unknown*. The 'job information' files also include data about the space and memory consumption, which we consider less relevant and therefore are not reported in the tables (see also the CHC-COMP 2021 report [7]).

3 Benchmarks

3.1 Format

CHC-COMP accepts benchmarks in the SMT-LIB 2.6 format [1]. All benchmarks have to conform to the format described at `https://chc-comp.github.io/format.html`. Conformance is checked using the `format.py` script available at `https://github.com/chc-comp/scripts`.

3.2 Inventory

All benchmarks used for the competition are selected from repositories under `https://github.com/chc-comp`. Anyone can contribute benchmarks to this repository. This year, we got several new benchmarks for many of the tracks. Table 1 summarizes the number of benchmarks and unique benchmarks available in each repository. The organizers pick a subset of all available benchmarks for each year's competition. In the rest of this section, we explain the steps in this selection.

3.3 Processing Benchmarks without Algebraic Data Types or Reals

All benchmarks are processed using the `format.py` script, which is available at `https://github.com/chc-comp/scripts`. The command line for invoking the script is

```
> python3.9 format.py --out-dir <out-dir> --merge_queries True <smt-file>
```

The script attempts to put benchmark `<smt-file>` into CHC-COMP format. The `merge_queries` option merges multiple queries into a single query as discussed in previous editions of CHC-COMP [7].

After processing, benchmarks are categorized into one of 4 competition tracks: LIA-lin, LIA-nonlin, LIA-lin-Arrays, and LIA-nonlin-Arrays. The scripts for categorizing the benchmarks are available at `https://github.com/chc-comp/chc-tools`. Benchmarks that could not be put in CHC-COMP compliant format and benchmarks that could not be categorized into any tracks are not used for the competition.

Repository	LIA-lin	LIA-nonlin	LIA-lin-Arrays	LIA-nonlin-Arrays	LRA-TS	ADT-nonlin	LIA-nonlin-Arrays-nonrecADT
adt-purified						67/67	
aeval	54/54						
aeval-unsafe (*new*)	54/54						
chc-comp19			290/290		228/226		
eldarica-misc	149/136	69/66					
extra-small-lia	55/55						
hcai	101/87	133/131	39/39	25/25			
hopv	49/48	68/67					
jayhorn	75/73	7325/7224					
kind2		851/737					
ldv-ant-med			10/10	79/79			
ldv-arrays			3/2	822/546			
llreve	44/44	43/42	31/31				
quic3			43/43				
ringen (*updated*)						454/440	
sally					177/174		
seahorn	3379/2812	68/66					
solidity (*new*)							3571/3548
sv-comp	3150/2930	1643/1169	79/73	856/780			
synth/nay-horn		119/114					
synth/semgus				5371/4839			
tricera	405/405	4/4					
tricera/adt-arrays (*new*)							156/156
ultimate (*new*)		8/8		21/21			
vmt	906/803				99/98		
total	8421/**7501**	10331/**9628**	495/**488**	7174/**6290**	504/**498**	521/**507**	3727/**3704**

Table 1: Summary of benchmarks (total/unique).

3.4 Processing benchmarks with Algebraic Data Types and Reals

For benchmarks containing either ADTs or Reals, no processing is done. All benchmarks submitted to the ADT-nonlin track were already processed using the RINGEN tool [16] to encode all theory symbols using ADTs. The benchmarks submitted to the LRA-TS and LIA-nonlin-Arrays-nonrecADT track were already in compliance with the CHC-COMP format.

3.5 Rating and Selection

This section describes the procedure used to select benchmarks for the competition.

For the LIA-lin-Arrays and LRA-TS tracks, consisting of a small amount of benchmarks, all unique benchmarks were selected.

In all other tracks, consisting of (i) either a large amount of benchmarks (that is, LIA-lin, LIA-nonlin, LIA-nonlin-Arrays, and LIA-nonlin-Arrays-nonrecADT), or (ii) too few repositories (that is, ADT-nonlin, where we need to balance between the repositories), we followed a procedure similar to the past editions of the competition aiming at selecting a representative subset of the available benchmarks. In particular, we estimated how "easy" the benchmarks were and picked a mix of "easy" and "hard" instances. We say that a benchmark in a track is "easy" if it is solved by both the winner and the runner-up solvers in the corresponding track in CHC-COMP 2021, within a small time interval. Each benchmark was rated A/B/C/D based on how difficult the previous competition winners found them. A rating of "A" is given if both solvers solved the benchmark, "B" if only the winner solved it, "C" if only the runner-up solved it, and "D" if neither solved it, within the set timeout. The timeout was selected based on the solver:

- **Spacer** was run with a timeout of 5s for all configurations, and
- **Eldarica** was run with a timeout of 10s for all configurations.

Eldarica was run with a higher timeout to compensate for the delay caused by JVM start-up. All solvers were run using the same binaries and configurations submitted for CHC-COMP 2021. For the newly introduced LIA-nonlin-Arrays-nonrecADT track, the winners from ADT-nonlin track of CHC-COMP 2021 were used.

Once we labelled each benchmark from a repository r, we decided the maximum number of instances, N_r, to take from the repository. N_r number was decided based on the total number of unique benchmarks and our knowledge about the benchmarks in repository r. We picked at most $0.2 \cdot N_r$ benchmarks each with ratings A, B, and C. Then, we picked $0.4 \cdot N_r$ benchmarks with rating D. If we did not find enough benchmarks with rating A, we picked the rest of the benchmarks equally from ratings B and C. If we did not find enough benchmarks with rating B or C, we pick the remaining benchmarks from rating D. This way, we obtained a mix of "easy" and "hard" benchmarks with a bias towards benchmarks that were not easily solved by either of the best solvers from the previous year's competition. The number of instances with each rating is given in Tables 2 and 3. The number of instances picked from each repository is given in Table 4. To pick `<num>` benchmarks of rating `<Y>`, we used the command

```
> cat <rating-Y-benchmark-list> | sort -R | head -n <num>
```

The final set of benchmarks selected for CHC-COMP 2022 can be found in the github repository `https://github.com/chc-comp/chc-comp22-benchmarks`, and on StarExec in the public space `CHC/CHC-COMP/chc-comp22`.

	LIA-lin				LIA-nonlin				LIA-nonlin-Arrays			
Repository	#A	#B	#C	#D	#A	#B	#C	#D	#A	#B	#C	#D
aeval	11	9	2	32								
aeval-unsafe	11	5	0	38								
eldarica-misc	84	39	2	11	9	26	1	30				
extra-small-lia	13	22	3	17								
hcai	77	5	0	5	71	41	0	19	12	6	1	6
hopv	47	1	0	0	46	14	7	0				
jayhorn	73	0	0	0	1870	3441	1	1912				
kind2					54	660	0	23				
ldv-ant-med									0	15	0	64
ldv-arrays									0	112	0	434
llreve	34	4	2	4	10	20	1	11				
seahorn	678	1306	1	827	28	25	0	13				
sv-comp	2361	475	1	93	309	766	5	89	245	254	1	280
synth/nay-horn					25	45	0	44				
synth/semgus									136	2386	0	2317
tricera/svcomp20	15	27	1	362	4	0	0	0				
ultimate					0	0	0	8	0	0	0	21
vmt	26	680	0	97								
total	3430	2573	12	1486	2426	5038	15	2149	393	2773	2	3122

Table 2: The number of unique benchmarks with ratings A/B/C/D - Tracks: LIA-lin, LIA-nonlin, and LIA-nonlin-Arrays.

Repository	ADT-nonlin				LIA-nonlin-Arrays-nonrecADT			
Repository	#A	#B	#C	#D	#A	#B	#C	#D
adt-purified	5	32	1	29				
ringen	11	17	3	409				
solidity					1033	1849	68	598
tricera/adt-arrays					2	29	0	125
total	16	49	4	438	1035	1878	68	723

Table 3: The number of unique benchmarks with ratings A/B/C/D – Tracks: ADT-nonlin, and LIA-nonlin-Arrays-nonrecADT.

Repository	LIA-lin	LIA-nonlin	LIA-nonlin-Arrays	ADT-nonlin	LIA-nonlin-Arrays-nonrecADT
adt-purified				67/52	
aeval	30/30				
aeval-unsafe	30/30				
eldarica-misc	45/31	30/30			
extra-small-lia	30/30				
hcai	45/19	60/43	15/13		
hopv	30/7	30/18			
jayhorn	30/6	90/90			
kind2		90/59			
ldv-ant-med			60/60		
ldv-arrays			90/90		
llreve	30/16	45/30			
ringen				134/131	
seahorn	90/90	45/31			
solidity					312/310
sv-comp	90/90	90/90	135/135		
synth/nay-horn		60/60			
synth/semgus			135/135		
tricera/svcomp20	60/60	3/0			
tricera/adt-arrays					156/155
ultimate		6/5	15/15		
vmt	90/90				
total	600/**499**	549/**456**	450/**448**	201/**183**	468/**465**

Table 4: The number of benchmarks to select and the number of selected benchmarks from each repository.

4 Solvers

Six solvers were submitted to CHC-COMP 2022: five competing solvers, and one solver *hors concours* (Spacer is co-developed by Hari Govind V K, who is one of the Organizers of the CHC-COMP 2022).

Table 5 lists the submitted solvers together with the configurations used to run them on the competition tracks. Detailed descriptions of the solvers are provided in Section 7. The binaries of the solvers are available on StarExec in the public space `CHC/CHC-COMP/chc-comp22`.

Solver	LIA-lin	LIA-nonlin	LIA-lin-Arrays	LIA-nonlin-Arrays	LRA-TS	LRA-TS-par	ADT-nonlin	LIA-nonlin-Arrays-nonrecADT
Eldarica	`def`	`def`	`def`	`def`	☐	☐	`def`	`def`
Golem	`lia-lin`	`lia-nonlin`	☐	☐	`lra-ts`	`lra-ts`	☐	☐
RInGen	☐	☐	☐	☐	☐	☐	`vampire`	☐
Ultimate TreeAutomizer	`default`	`default`	`default`	`default`	`default`	`default`	`default`	`default`
Ultimate Unihorn	`default`	`default`	`default`	`default`	`default`	`default`	☐	☐
Spacer	`LIA-LIN`	`LIA-NONLIN`	`LIA-LIN-ARRAYS`	`LIA-NONLIN-ARRAYS`	`LRA-TS`	`LRA- TS`	`ADT-LIN`	`ADT- NON-LIN`

Table 5: Submitted solvers and configurations used in the competition track; '☐' denotes that the solver did not enter the competition in that track. The configuration names have been taken as is from solver submissions.

5 Results

The results of the CHC-COMP 2022 are presented in Table 6.

Eldarica and Ultimate TreeAutomizer were the only solvers that entered the competition in the 'LIA-nonlin-Arrays-nonrecADT' track. Ultimate TreeAutomizer did not provide any *sat* and *unsat* result, therefore it is not included as 2nd classified in the the final ranking.

Detailed results for the eight tracks are provided in Appendix A

5.1 Observed Issues and Fixes during the Competition Runs

In the competition runs of the LIA-lin-Arrays and LRA-TS tracks we detected 4 and 17 inconsistent results, respectively. **Ultimate Unihorn** reported *sat*, while other competing tools reported *unsat*. In particular, in the LIA-lin-Arrays track, we observed that both Eldarica and Ultimate TreeAutomizer reported *unsat* on 2 out of 4 *sat* results reported by Ultimate Unihorn, while on the other 2 *sat* results Ultimate TreeAutomizer reported *unsat* and Eldarica *unknown*. In the LRA-TS track, Golem reported 17 *unsat* results, while Ultimate TreeAutomizer reported 12 *unsat* and 5 *unknown* results, respectively.

	LIA-lin	LIA-nonlin	LIA-lin-Arrays	LIA-nonlin-Arrays	LRA-TS	LRA-TS-par	ADT-nonlin	LIA-nonlin-Arrays-nonrecADT
Winner	**Golem**	**Golem**	**Eldarica**	**Eldarica**	**Golem**	**Golem**	**RInGen**	**Eldarica**
2nd place	Eldarica	Eldarica	Ultimate Unihorn	Ultimate Unihorn	Ultimate TreeAutomizer	Ultimate TreeAutomizer	Eldarica	
3rd place	Ultimate Unihorn	Ultimate Unihorn	Ultimate TreeAutomizer	Ultimate TreeAutomizer	Ultimate Unihorn	Ultimate Unihorn		

Table 6: Results of the competition.

The inconsistencies were detected on March 24, 2022. We informed the authors of Ultimate Unihorn on March 25, 2022 by sending them six benchmarks on which we detected the inconsistencies: two LIA-lin-Arrays benchmarks and four LRA-TS benchmarks. The authors submitted a fixed version of their tool on March 28, 2022.

The results presented in this report were produces using the fixed version. In Table 7 we report the results before and after the fixes.

Ultimate	LIA-lin-Arrays		LRA-TS	
Unihorn	*#sat*	*#unsat*	*#sat*	*#unsat*
original	283	66	134	22
fixed	137	67	65	38

Table 7: Results produced by Ultimate Unihorn before (original) and after the bug fixes (fixed).

6 Conclusions and Final Remarks

We would like to congratulate the winners of the CHC-COMP 2022 (in alphabetical order): **Eldarica** (winner of the LIA-lin-Arrays and LIA-lin-Arrays tracks, and the newly introduced LIA-nonlin-Arrays-nonrecADT track), **Golem** (winner of the LIA-lin, LIA-nonlin, LRA-TS, and LRA-TS-par tracks), and **RInGen** (winner of the ADT-nonlin track).

We conclude with a few remarks on the open issues that should be discussed and addressed in the future editions of the CHC-COMP, which are based on our experience with running the competition and the observations made by the HCVS 2022 participants in the follow-up discussion we had after the presentation of the competition results.

- **Validation of results**. The ability of solvers to generate models and counterexamples is a recurrent request by our community members (this issue has been already discussed in the previous editions, see [7]). Thus, to encourage the developers to introduce this feature (some solvers already provide it), we could begin, as proposed in the CHC-COMP 2021 report [7], by introducing new tracks where this feature is taken into consideration in the computation of the score. For instance, the score could be weighted according to type of witness provided by the solver to support its result.

- **Status of benchmarks**. In order to assess the correctness of the result provided by the solvers, each submitted benchmark should explicitly declare the expected result of the satisfiability problem. We propose to use the `( set-info` ⟨*keyword*⟩ ⟨*attr-value*⟩ `)` command with the `:status` as *keyword*, and either `sat` or `unsat` as *attr-value*.

- **The LRA-TS and LRA-TS-par tracks**. As already discussed in [7], also in this edition, no solver requiring the syntactic restriction on the form of the clauses included in the LRA-TS track has been submitted. Hence, we propose to discontinue the LRA-TS and LRA-TS-par tracks starting from the CHC-COMP 2023, and to add more general LRA tracks, such as LRA-lin and LRA-nonlin.

- **The ADT-nonlin and LIA-nonlin-Arrays-nonrecADT tracks**. As already discussed in [7], the syntactic restrictions on the form of the clauses in ADT-nonlin track were meant to attract more solvers to enter the competition. Nowadays, there is an increasing interest in developing techniques for solving CHCs with constraints over ADTs and LIA/LRA [16, 5, 14], thereby increasing the availability of tools supporting these theories. In this regard, CHC-COMP 2022 introduced the LIA-nonlin-Arrays-nonrecADT track, which combines nonrecursive ADTs with LIA and Arrays, but only two solvers, that is, Eldarica and Spacer, were able to enter the competition in this track. We propose to try a more gradual combination of such theories, such as ADTs with LIA and ADTs with Arrays.

- **Generation of the benchmark suite for the competition runs**. The process for constructing the benchmark suite is based on the evaluation of the hardness of the encoded problems. This evaluation is performed by running the winners of the previous edition for a very limited amount of time (see Section 3.5) In the follow-up discussion at HCVS, it has been suggested to increase the amount of time given to the solvers to get a more accurate evaluation of the hardness of the benchmarks.

Finally, we would also to stress the fact that **a bigger set of benchmarks are needed**. Besides submitting their tools, all participants are invited to contribute with new benchmarks.

7 Solver Descriptions

The tool descriptions in this section were contributed by the participants, and the copyright on the texts remains with the individual authors.

7.1 Eldarica v2.0.8

Hossein Hojjat
University of Tehran, Iran

Philipp Rümmer
University of Regensburg, Germany

Algorithm. Eldarica [12] is a Horn solver applying classical algorithms from model checking: predicate abstraction and counterexample-guided abstraction refinement (CEGAR). Eldarica can solve Horn clauses over linear integer arithmetic, arrays, algebraic data-types, bit-vectors, and the theory of heaps. It can process Horn clauses and programs in a variety of formats, implements sophisticated algorithms to solve tricky systems of clauses without diverging, and offers an elegant API for programmatic use.

Architecture and Implementation. Eldarica is entirely implemented in Scala, and only depends on Java or Scala libraries, which implies that Eldarica can be used on any platform with a JVM. For computing abstractions of systems of Horn clauses and inferring new predicates, Eldarica invokes the SMT solver Princess [19] as a library.

Configuration in CHC-COMP 2022. Eldarica is in the competition run with the option `-portfolio`, which enables a simple portfolio mode: three instances of the solver are run in parallel, one with options `-splitClauses:0 -abstract:off`, one with options `-splitClauses:1 -abstract:off`, and one with default options.

`https://github.com/uuverifiers/eldarica`
BSD licence

7.2 Golem

Martin Blicha
Università della Svizzera italiana, Switzerland

Algorithm. Golem is a CHC solver under active development that provides several backend engines implementing various interpolation-based model-checking algorithms. It supports the theory of Linear Real or Integer Arithmetic and it is able to provide witnesses for both satisfiable and unsatisfiable CHC systems. The three engines of Golem are:

- `lawi` is our re-implementation of the IMPACT algorithm [17]
- `spacer` is our re-implementation of the SPACER algorithm [15] and allows Golem to solve non-linear systems.
- `tpa` is our new model-checking algorithm based on doubling abstractions using Craig interpolants [3].

Architecture and Implementation. Golem is implemented in C++ and built on top of the interpolating SMT solver OpenSMT [13] which is used for both satisfiability solving and interpolation. The only dependencies are those inherited from OpenSMT: Flex, Bison and GMP libraries.

New Features in CHC-COMP 2022. Compared to the previous year, Golem has two new backend engines, `spacer` and `tpa`. Additionally, Golem now uses basic preprocessing to simplify the input clauses before handing them over to the backend engine.

Configuration in CHC-COMP 2022. For LIA-nonlin track we used only `spacer` engine; the other engines cannot handle nonlinear system yet.

```
$ golem --logic QF_LIA --engine spacer
```

For LIA-lin and LRA-TS tracks, we used a trivial portfolio of all three engines running independently.

`https://github.com/usi-verification-and-security/golem`
MIT LICENSE

7.3 RInGen v1.2

Yurii Kostyukov
JetBrains Research, Russia

Dmitry Mordvinov
JetBrains Research, Russia

Algorithm. RInGen is a *R*egular *In*variant *Gen*erator and a first-order logic formula transformer. RInGen is based on the preprocessing approach presented in [16]. A system of constraint Horn clauses (CHCs) over algebraic datatypes (ADTs) is rewritten into a formula over uninterpreted function symbols. Crucial step is elimination of all disequalities, testers, and selectors from the clause bodies by introducing their Horn axioms. Then the satisfiability modulo theory of ADTs is reduced to satisfiability modulo theory of uninterpreted functions with equality (EUF) by replacing all ADT sorts with free sorts and and all constructors with free functions. After that, an off-the-shelf logical solver for many-sorted logic with quantifiers is called. As the proposed transformation gives a formula which is satisfiable modulo EUF iff the original CHC system is satisfiable modulo ADTs, the result of the logical solver is returned as is.

Architecture and Implementation. RInGen accepts input in the SMTLIB2 format and produces Horn clauses over pure ADT sorts in SMTLIB2 and Prolog. It takes conditions with a property and checks if the property holds, returning SAT and the safe inductive invariant if it does or terminates with UNSAT if it does not. We run Vampire as a backend many-sorted EUF solver. Vampire [18] searches for both refutations and saturations of it's input problem, which gives us both SAT and UNSAT results for CHC systems.

New Features in CHC-COMP 2022. The main feature of this year contribution is using our fine-tuned fork of Vampire as a backend.

Configuration in CHC-COMP 2022. The tool is run with the following arguments:
`--timelimit $tl -q -o "$2/" solve -s vampire --path "$input" -t --no-isolation`.
The tool runs our fork[3] of VAMPIRE (using `vampire --mode chccomp`) tuned for CHC problems as a backend solver.

`https://github.com/Columpio/RInGen/releases/tag/chccomp22`
BSD 3-Clause License

7.4 Ultimate TreeAutomizer 0.2.2-dev-f165340

Matthias Heizmann
University of Freiburg, Germany

Daniel Dietsch
University of Freiburg, Germany

Jochen Hoenicke
University of Freiburg, Germany

Alexander Nutz
University of Freiburg, Germany

Andreas Podelski
University of Freiburg, Germany

Frank Schüssele
University of Freiburg, Germany

Algorithm. The ULTIMATE TREEAUTOMIZER solver implements an approach that is based on tree automata [6]. In this approach potential counterexamples to satisfiability are considered as a regular set of trees. In an iterative CEGAR loop we analyze potential counterexamples. Real counterexamples lead to an *unsat* result. Spurious counterexamples are generalized to a regular set of spurious counterexamples and subtracted from the set of potential counterexamples that have to be considered. In case we detected that all potential counterexamples are spurious, the result is *sat*. The generalization above is based on tree interpolation and regular sets of trees are represented as tree automata.

Architecture and Implementation. TREEAUTOMIZER is a toolchain in the ULTIMATE framework. This toolchain first parses the CHC input and then runs the `treeautomizer` plugin which implements the above mentioned algorithm. We obtain tree interpolants from the SMT solver SMTInterpol[4] [11]. For checking satisfiability, we use the and Z3 SMT solver[5]. The tree automata are implemented in ULTIMATE's automata library[6]. The ULTIMATE framework is written in Java and build upon the Eclipse Rich Client Platform (RCP). The source code is available at GitHub[7].

[3]`https://github.com/Columpio/Vampire/tree/chc-comp22`
[4]`https://ultimate.informatik.uni-freiburg.de/smtinterpol/`
[5]`https://github.com/Z3Prover/z3`
[6]`https://ultimate.informatik.uni-freiburg.de/automata_library`
[7]`https://github.com/ultimate-pa/`

Configuration in CHC-COMP 2022. Our StarExec archive for the competition is shipped with the `bin/starexec_run_default` shell script calls the ULTIMATE command line interface with the `TreeAutomizer.xml` toolchain file and the `TreeAutomizerHopcroftMinimization.epf` settings file. Both files can be found in toolchain (resp. settings) folder of ULTIMATE's repository.

`https://ultimate.informatik.uni-freiburg.de/`
LGPLv3 with a linking exception for Eclipse RCP

7.5 Ultimate Unihorn 0.2.2-dev-f165340

Matthias Heizmann
University of Freiburg, Germany
Daniel Dietsch
University of Freiburg, Germany
Jochen Hoenicke
University of Freiburg, Germany
Alexander Nutz
University of Freiburg, Germany
Andreas Podelski
University of Freiburg, Germany
Frank Schüssele
University of Freiburg, Germany

Algorithm. ULTIMATE UNIHORN reduces the satisfiability problem for a set of constraint Horn clauses to a software verfication problem. In a first step UNIHORN applies a yet unpublished translation in which the constraint Horn clauses are translated into a recursive program that is nondeterministic and whose correctness is specified by an assert statement The program is correct (i.e., no execution violates the assert statement) if and only if the set of CHCs is satisfiable. For checking whether the recursive program satisfies its specification, Unihorn uses ULTIMATE AUTOMIZER [9] which implements an automata-based approach to software verification [10].

Architecture and Implementation. ULTIMATE UNIHORN is a toolchain in the ULTIMATE framework. This toolchain first parses the CHC input and then runs the `chctoboogie` plugin which does the translation from CHCs into a recursive program. We use the Boogie language to represent that program. Afterwards the default toolchain for verifying a recursive Boogie programs by ULTIMATE AUTOMIZER is applied. The ULTIMATE framework shares the libraries for handling SMT formulas with the SMTInterpol SMT solver. While verifying a program, ULTIMATE AUTOMIZER needs SMT solvers for checking satisfiability, for computing Craig interpolants and for computing unsatisfiable cores. The version of UNIHORN that participated in the competition used the SMT solvers SMTInterpol[8]and Z3[9]. The ULTIMATE framework is written in Java and build upon the Eclipse Rich Client Platform (RCP). The source code is available at GitHub[10].

[8]`https://ultimate.informatik.uni-freiburg.de/smtinterpol/`
[9]`https://github.com/Z3Prover/z3`
[10]`https://github.com/ultimate-pa/`

Configuration in CHC-COMP 2022. Our StarExec archive for the competition is shipped with the `bin/starexec_run_default` shell script calls the ULTIMATE command line interface with the `AutomizerCHC.xml` toolchain file and the `chccomp-Unihorn_Default.epf` settings file. Both files can be found in toolchain (resp. settings) folder of ULTIMATE's repository.

`https://ultimate.informatik.uni-freiburg.de/`
LGPLv3 with a linking exception for Eclipse RCP

References

[1] Clark Barrett, Pascal Fontaine & Cesare Tinelli (2016): *The Satisfiability Modulo Theories Library (SMT-LIB)*. `www.SMT-LIB.org`.

[2] Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan & Andrey Rybalchenko (2015): *Horn Clause Solvers for Program Verification*. In Lev D. Beklemishev, Andreas Blass, Nachum Dershowitz, Bernd Finkbeiner & Wolfram Schulte, editors: *Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, Springer International Publishing, Cham, pp. 24–51, doi:`10.1007/978-3-319-23534-9_2`.

[3] Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen & Natasha Sharygina (2022): *Transition Power Abstractions for Deep Counterexample Detection*. In Dana Fisman & Grigore Rosu, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, Springer International Publishing, Cham, pp. 524–542, doi:`10.1007/978-3-030-99524-9_29`.

[4] Emanuele De Angelis, Fabio Fioravanti, John P. Gallagher, Manuel V. Hermenegildo, Alberto Pettorossi & Maurizio Proeitti (2021): *Analysis and Transformation of Constrained Horn Clauses for Program Verification*. *Theory and Practice of Logic Programming*, p. 1–69, doi:`10.1017/S1471068421000211`.

[5] Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi & Maurizio Proietti (2022): *Satisfiability of constrained Horn clauses on algebraic data types: A transformation-based approach*. *J. Log. Comput.* 32(2), pp. 402–442, doi:`10.1093/logcom/exab090`.

[6] Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz & Andreas Podelski (2019): *Ultimate TreeAutomizer (CHC-COMP Tool Description)*. In: *HCVS/PERR@ETAPS*, *EPTCS* 296, pp. 42–47, doi:`10.4204/eptcs.296.7`.

[7] Grigory Fedyukovich & Philipp Rümmer (2021): *Competition Report: CHC-COMP-21*. In Hossein Hojjat & Bishoksan Kafle, editors: *Proceedings 8th Workshop on Horn Clauses for Verification and Synthesis, HCVS@ETAPS 2021, Virtual, 28th March 2021*, *EPTCS* 344, Open Publishing Association, pp. 91–108, doi:`10.4204/EPTCS.344.7`.

[8] Arie Gurfinkel (2022): *Program Verification with Constrained Horn Clauses (Invited Paper)*. In Sharon Shoham & Yakir Vizel, editors: *Computer Aided Verification*, Springer International Publishing, Cham, pp. 19–29, doi:`10.1007/978-3-031-13185-1_2`.

[9] Matthias Heizmann, Yu-Fang Chen, Daniel Dietsch, Marius Greitschus, Jochen Hoenicke, Yong Li, Alexander Nutz, Betim Musa, Christian Schilling, Tanja Schindler & Andreas Podelski (2018): *Ultimate Automizer and the Search for Perfect Interpolants - (Competition Contribution)*. In: *TACAS (2)*, *Lecture Notes in Computer Science* 10806, Springer, pp. 447–451, doi:`10.1007/978-3-319-89963-3_30`.

[10] Matthias Heizmann, Jochen Hoenicke & Andreas Podelski (2013): *Software Model Checking for People Who Love Automata*. In: *CAV*, *Lecture Notes in Computer Science* 8044, Springer, pp. 36–52, doi:`10.1007/978-3-642-39799-8_2`.

[11] Jochen Hoenicke & Tanja Schindler (2018): *Efficient Interpolation for the Theory of Arrays*. In: *IJCAR*, *Lecture Notes in Computer Science* 10900, Springer, pp. 549–565, doi:`10.1007/978-3-319-94205-6_36`.

[12] Hossein Hojjat & Philipp Rümmer (2018): *The ELDARICA Horn Solver*. In: *2018 Formal Methods in Computer Aided Design, FMCAD*, pp. 1–7, doi:10.23919/FMCAD.2018.8603013.

[13] Antti E. J. Hyvärinen, Matteo Marescotti, Leonardo Alt & Natasha Sharygina (2016): *OpenSMT2: An SMT Solver for Multi-core and Cloud Computing*. In Nadia Creignou & Daniel Le Berre, editors: *Theory and Applications of Satisfiability Testing – SAT 2016*, Springer International Publishing, Cham, pp. 547–553, doi:10.1007/978-3-319-40970-2_35.

[14] Hari Govind V. K., Sharon Shoham & Arie Gurfinkel (2022): *Solving constrained Horn clauses modulo algebraic data types and recursive functions*. *Proc. ACM Program. Lang.* 6(POPL), pp. 1–29, doi:10.1145/3498722.

[15] Anvesh Komuravelli, Arie Gurfinkel & Sagar Chaki (2016): *SMT-based Model Checking For Recursive Programs*. *Formal Methods in System Design* 48(3), pp. 175–205, doi:10.1007/s10703-016-0249-4.

[16] Yurii Kostyukov, Dmitry Mordvinov & Grigory Fedyukovich (2021): *Beyond the Elementary Representations of Program Invariants over Algebraic Data Types*. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, ACM, p. 451–465, doi:10.1145/3453483.3454055.

[17] Kenneth L. McMillan (2006): *Lazy Abstraction with Interpolants*. In Thomas Ball & Robert B. Jones, editors: *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 123–136, doi:10.1007/11817963_14.

[18] Alexandre Riazanov & Andrei Voronkov (2002): *The Design and Implementation of VAMPIRE*. *AI Commun.* 15(2,3), p. 91–110.

[19] Philipp Rümmer (2008): *A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic*. In: *Proceedings, 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, *LNCS* 5330, Springer, pp. 274–289, doi:10.1007/978-3-540-89439-1_20.

[20] Aaron Stump, Geoff Sutcliffe & Cesare Tinelli (2014): *StarExec: A Cross-Community Infrastructure for Logic Solving*. In Stéphane Demri, Deepak Kapur & Christoph Weidenbach, editors: *Automated Reasoning*, Springer International Publishing, Cham, pp. 367–373, doi:10.1007/978-3-319-08587-6_28.

A Detailed results

Table 8: Solver performance on LIA-lin track

Solver	**Score**	**#sat**	**#unsat**	**CPU time/s**	**Wall-clock/s**	**#unique**
Spacer	338	235	103	299420	149835	36
Golem	309	215	94	374736	142604	25
Eldarica	307	219	88	372231	134933	38
U. Unihorn	169	107	62	551859	466284	0
U. TreeAutomizer	139	81	58	633917	605367	0

Table 9: Solver performance on LIA-nonlin track

Solver	**Score**	**#sat**	**#unsat**	**CPU time/s**	**Wall-clock/s**	**#unique**
Spacer	421	286	135	75414	39303	40
Golem	365	240	125	196890	196913	2
Eldarica	358	229	129	215589	76099	7
U. Unihorn	204	123	81	485808	391416	1
U. TreeAutomizer	50	13	37	691499	648778	0

Table 10: Solver performance on LIA-lin-Arrays track

Solver	**Score**	**#sat**	**#unsat**	**CPU time/s**	**Wall-clock/s**	**#unique**
Spacer	288	213	75	355686	178328	85
Eldarica	220	149	71	481857	163636	10
U. Unihorn	204	137	67	479998	393423	2
U. TreeAutomizer	170	113	57	491989	470080	0

Table 11: Solver performance on LIA-nonlin-Arrays track

Solver	Score	#sat	#unsat	CPU time/s	Wall-clock/s	#unique
Spacer	342	197	145	180474	95392	115
Eldarica	215	129	86	449249	177427	10
U. Unihorn	168	88	80	368162	297747	2
U. TreeAutomizer	89	19	70	618917	488550	5

Table 12: Solver performance on LRA-TS track

Solver	Score	#sat	#unsat	CPU time/s	Wall-clock/s	#unique
Spacer	317	234	83	355136	181996	37
Golem	311	235	76	364678	121607	19
U. TreeAutomizer	155	114	41	646968	619460	4
U. Unihorn	103	65	38	725651	613022	1

Table 13: Solver performance on LRA-TS-parallel track

Solver	Score	#sat	#unsat	CPU time/s	Wall-clock/s	#unique
Spacer	341	256	85	627685	319147	35
Golem	333	256	77	988621	329628	22
U. TreeAutomizer	155	114	41	671801	641980	3
U. Unihorn	103	65	38	1072345	719996	1

Table 14: Solver performance on ADT-nonlin track

Solver	Score	#sat	#unsat	CPU time/s	Wall-clock/s	#unique
RInGen	92	50	42	165240	163756	27
Eldarica	60	29	31	223335	73011	4
Spacer	55	25	30	226205	226257	5
U. TreeAutomizer	0	0	0	1881	745	0

Table 15: Solver performance on LIA-nonlin-Arrays-nonrecADT track

Solver	Score	#sat	#unsat	CPU time/s	Wall-clock/s	#unique
Eldarica	395	242	153	125994	46440	103
Spacer	298	179	119	131079	131124	6
U. TreeAutomizer	0	0	0	4825	1894	0