Performance Evaluation of State-of-the-Art String Solvers

Kilian Lichtner

Contents

1 Setup

1.1 Enviroment

The experiment was run on the university's terminal server. Which is equipped with a Intel(R) Xeon(R) Gold 6354 and 1 Terrabyte of RAM, running Rocky Linux version 8.10. Since this is a server available to most students a certain background noise should be considered, even though most of the time only a small amount of the servers resources are actually used.

1.2 Solvers

We selected a total of 6 string solvers namely:

- Ostrich version 1.4
- Ostrich on the modular proof rules branch(commit a54d244f16413490343fac6a0fd1ab4b2ddbe91e)
- CVC5 version 1.2.0
- Z3 version 4.13.3
- Z3-Noodler version 1.3.0(compiled using the zig compiler due to libc issues)
- Z3Alpha commit 6458d1251f5edbcf6b7412e5d54741c0db8f9b34

Most solvers were given no flags with the exception of a timeout flag, and ostrich/ostrich modular to turn off verbose outputs and enable the use of the new features provided by ostrich modular (-stringSolver=ostrich.OstrichStringTheory:+eager,+forwardPropagation,+backwardPropagation, +nielsenSplitter,-length=on).

Also to note here is that ostrich was not run in server mode, so there might be significant overhead due to JVM startup times.

1.3 Benchmarking

The benchmarking was done using the perf utility invoked by a custom python script on the entire QF_S* problem set, which is a total of 103405 problems. Perf was chosen because it allows us to obtain precise timings on the actual calculation time on the CPU each solver had. Furthermore other statistics that could be of interest were recorded. If a solver used several threads the calculation time of each thread was sumed.

Recorded metrics are:

- Sat/Unsat
- task-clock (total time spent on the cpu in userspace)
- context-switches
- cpu-migrations
- page-faults
- cycles
- instructions
- branches
- branch misses
- elapsed
- user (time spent in userland)
- sys (time spent in system space)

For more info on these stats check the manpage of perf.

Each solver was passed a timeout flag of 60 seconds(per problem). After 70 seconds the solver was hard killed by the benchmarking script.

2 Categorizing

For categorizing I decided on a system in which I tag problems depening on operations/problem structure, and then group them via a combination of these tags.

2.1 Tags

Let's first explore the meaning of each tag:

- regular-constraints: The problem contains regular expressions and or regular membership
- str-replace: The problem contains the str.replace or str.replace_all operators
- word-equations: The problem contains simple word equations, eg. string equality and or string variables
- lia: The problem contains non relational operations from linear integer arithmetic, eg addition, substraction or multiplication or relations between non lengths

- length-constraints: The problem contains length constraints and or relations on them, but no operation on these lengths(eg. addition, substraction ...)
- substrings: The problem contains substring operations such as prefixof/suffixof/substr
- search: The problem has some searching property, eg. str.at str.contains, str.indexof
- re-replace: The problem contains re.replace or re.replace_all
- nia: The problem contains some none integer arithmetic, eg. floating point variables, their addition/comparison etc.
- logic: The problem contains boolean variables, or extensive use of logical operators especially ite(if then else)

2.2 Categories

I then came up with a total of 18 different categories(tagsets):

- (word-equations) Fundamental for testing core string concatenation and equality solving.
- (regular-constraints) These constraints involve regular expressions or regular membership tests. They play a critical role in validating inputs.[1]
- (str-replace, regular-constraints) Models real-world sanitization (e.g., escaping characters in regex-constrained strings). Solvers must track replacements while preserving regex validity. [4]
- (length-constraints, regular-constraints) It is hard to say if length constraints should be contained in LIA. But it is interesting to see how the introduction of some simple length constraints impacts the performance on these problems(see eg. z3noodler)
- (lia, regular-constraints, length-constraints) Adding full blown LIA to the problem introduces an entire second theory to the solver, which leads to a combinatorial explosion. [2]
- (word-equations, length-constraints) As with regular expressions it is interesting to see how the introduction of these simple length constraints makes it harder to solve the problem on hand.
- (length-constraints, substrings, regular-constraints) Solvers must reason about partial matches and their lengths. This is also used alot for verification and security analyses.[3]
- (lia, substrings, search, length-constraints) Common in program analysis[3], requires joint reasoning about positions, lengths, and substrings.

- (lia, logic, regular-constraints, length-constraints)
- (logic, word-equations, lia, length-constraints, search) Tests solver capacity to interleave predicate logic, positional reasoning, and arithmetic – critical for vulnerability analysis where control flow depends on string inspections.
- (str-replace, substrings, search) Real-world workflows like sanitizing substrings found via indexof. Tests chaining of operations, which can create cascading constraints.[3]
- (substrings, search)
- (substring, str-replace, search) Highlights solver efficiency when substitutions alter subsequent search outcomes.
- (str-replace, length-constraints, regular-constraints) Replacements with length and regex constraints (e.g., replacing a with bb in a string of fixed length). Tests how solvers handle conflicting structural and numeric requirements.
- (length-constraints, regular-constraints, re-replace) Requires tracking regex transformations and their length effects.
- (length-constraints, str-replace, regular-constraints) Replacements may violate length bounds or regex structure. Forces solvers to jointly solve for replacement effects in both domains.
- (word-equations, length-constraints, logic) Even tho this is a very small class, it highlights solver generality beyond integer theories.
- (nia)

Regex-Length Interaction (4, 5, 7, 14–16): Regex-heavy tagsets reveal how well solvers avoid state-space explosions when combined with arithmetic.[2]

Search/Replace Workflows (8, 11–13): Common in sanitization, parsing, and refactoring. Solvers must handle cascading effects of substitutions (e.g., replacing a substring alters subsequent search results).[3]

Theory Integration (5, 9–10, 17): Tagsets with LIA/logic test how solvers handle the combination with other theorys.

Edge Cases (18): nia problems probe solver robustness beyond standard integer assumptions.

3 Evaluation

This section evaluates the performance of six string solvers on the complete QF_S* problem set(103405 problems). Figures 3 and 2 summarize each solver's overall performance, including the number of solved instances, timeouts, errors, and runtimes.

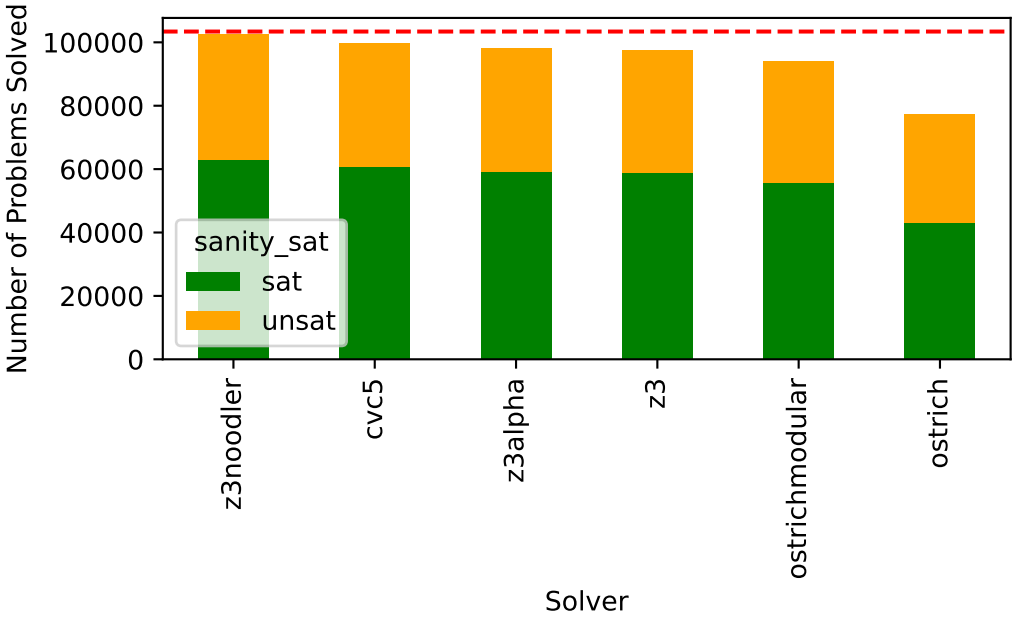


Figure 1: Performance on the entire problem set

Solver	Timeout	Errors	Solved	SAT	UNSAT	Total time(msec)
cvc5	3647	95	99661	60766	38895	44304962.59
ostrich	25474	710	77219	42922	34297	501431481.04
ostrichmodular	9165	59	94180	55756	38424	616537171.70
z3	5786	69	97548	58742	38806	76672930.87
z3alpha	5329	0	98074	59252	38822	52980512.81
z3noodler	833	0	102570	63078	39492	9972615.38

Figure 2: Summary of performance metrics for the entire benchmark set

Overall, Z3Noodler stands out by solving nearly all problems while maintaining the shortest runtime. CVC5, Z3Alpha, and Z3 exhibit comparable performance, with Ostrich Modular coming close behind—improving significantly

over its standard version by solving an additional 16,961 problems.

The regular constraints category, which includes 33,248 problems, is illustrated in Figure 3.

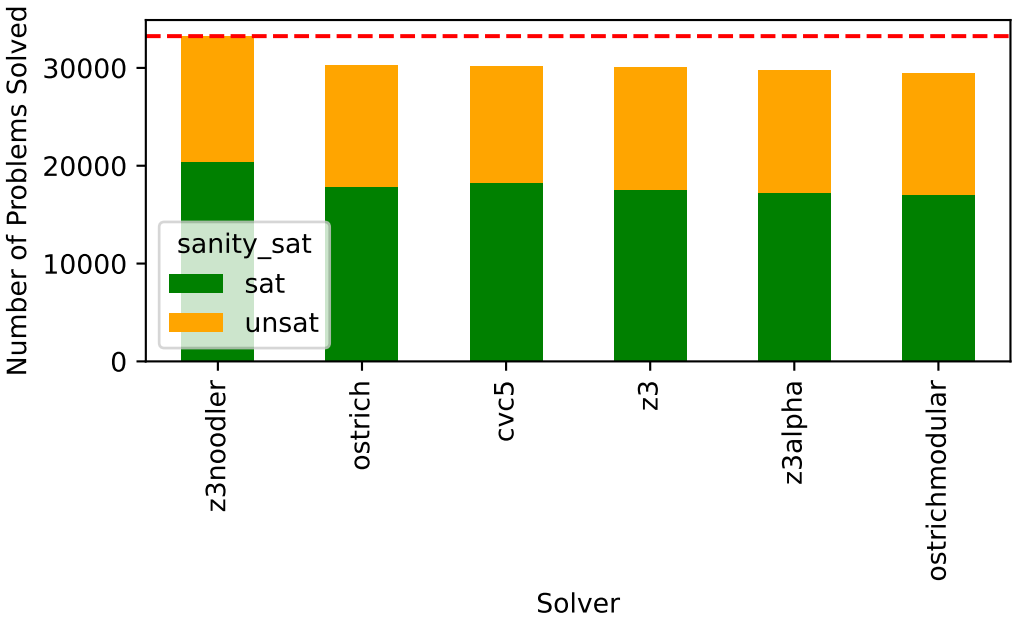


Figure 3: Regular constraints

Here, Z3Noodler is the clear leader, successfully solving every instance. The other solvers perform similarly, though Ostrich Modular shows a slight regression relative to its standard counterpart. It is also noteworthy that almost all unsolved instances are SAT.

3.1 Word Equations

In the word equations category (approximately 900 problems), Z3Noodler again solves nearly every instance. CVC5, Z3, and Z3Alpha follow closely, while both Ostrich and Ostrich Modular trail behind—with Ostrich Modular underperforming relative to the standard Ostrich. Introducing length constraints widens the performance gap between Z3Noodler and its competitors. Interestingly, the standard Ostrich solver shows improved performance compared to earlier categories. However, Ostrich Modular appears to struggle mainly with SAT instances in this context.

3.2 Combined Constraints

Length Constraints with Regular Constraints: For problems that combine length constraints with regular constraints, both Z3Noodler (which solves all instances) and Ostrich (solving all but one instance) lead the pack. They are followed by Ostrich Modular, with CVC5 ranking fourth and Z3 and Z3Alpha achieving similar results.

Substrings, Length Constraints, and Regular Constraints: This largest category, containing 36,400 problems, is led by CVC5. Z3Noodler, Z3Alpha, and Z3 follow closely, and here Ostrich Modular outperforms the standard Ostrich.

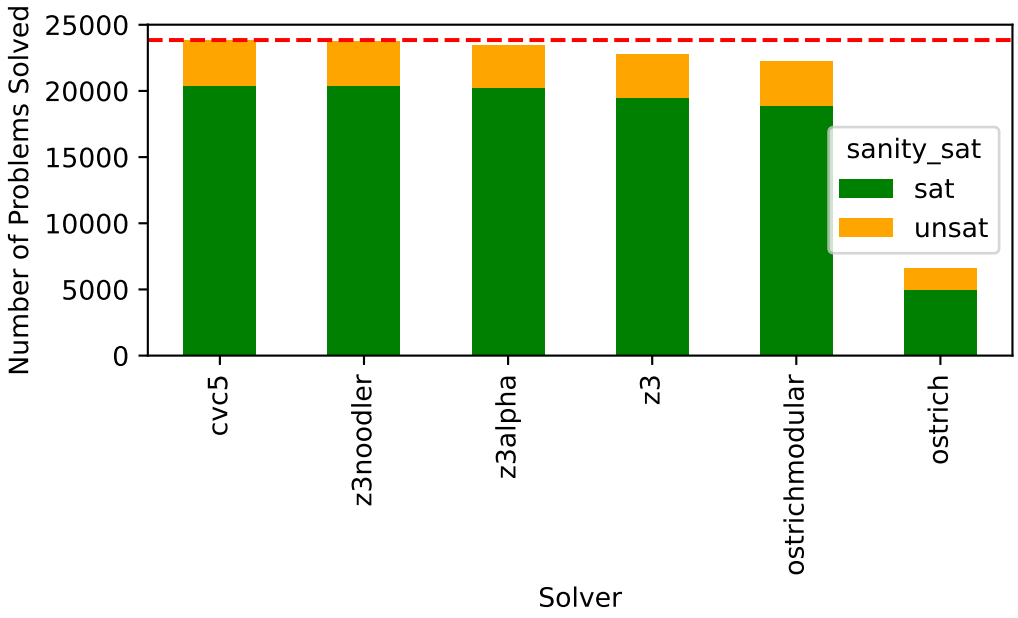


Figure 4: Substrings, length constraints, regular constraints

Substrings, LIA, Length Constraints, and Searches: In this category of 23,845 problems, CVC5 again performs best, followed by Z3Noodler, Z3Alpha, and Z3. Ostrich Modular shows a marked improvement over the standard Ostrich solver.

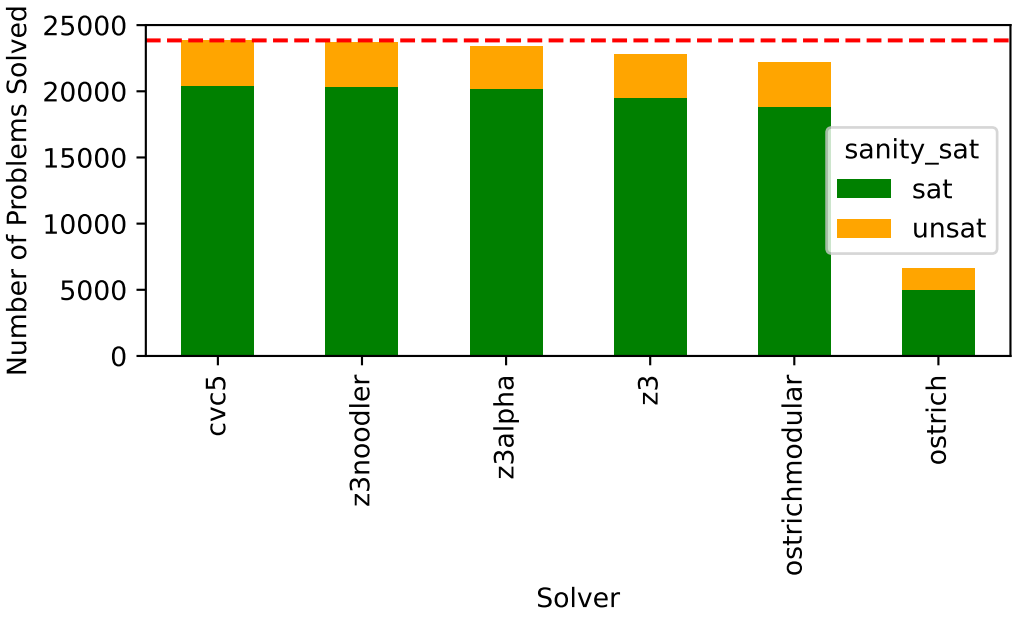


Figure 5: substrings, lia, length constraints, searches

Other Categories: The following categories exhibit a similar performance pattern—with either CVC5 or Z3Noodler taking the lead, and Z3 and Z3Alpha closely trailing:

- Logic, LIA, length constraints, and regular constraints.
- Length constraints, logic, LIA, search, and word equations.
- Substrings, search, and string replacement.
- Substrings and search.

In these cases, the relative performance between Ostrich Modular and standard Ostrich varies, with some categories showing improvements and others declines for Ostrich Modular.

Regular Expression Replacement For problems involving regular expression replacement in conjunction with length constraints and regular constraints (see Figure below), CVC5 leads, followed by Ostrich, Z3Alpha, Z3, Ostrich Modular and finally Z3Noodler. As expected, Z3Noodler underperforms here due to its incomplete support for regular expression replacement. The performance in the category combining length constraints, regular constraints, and string replacement is nearly identical.

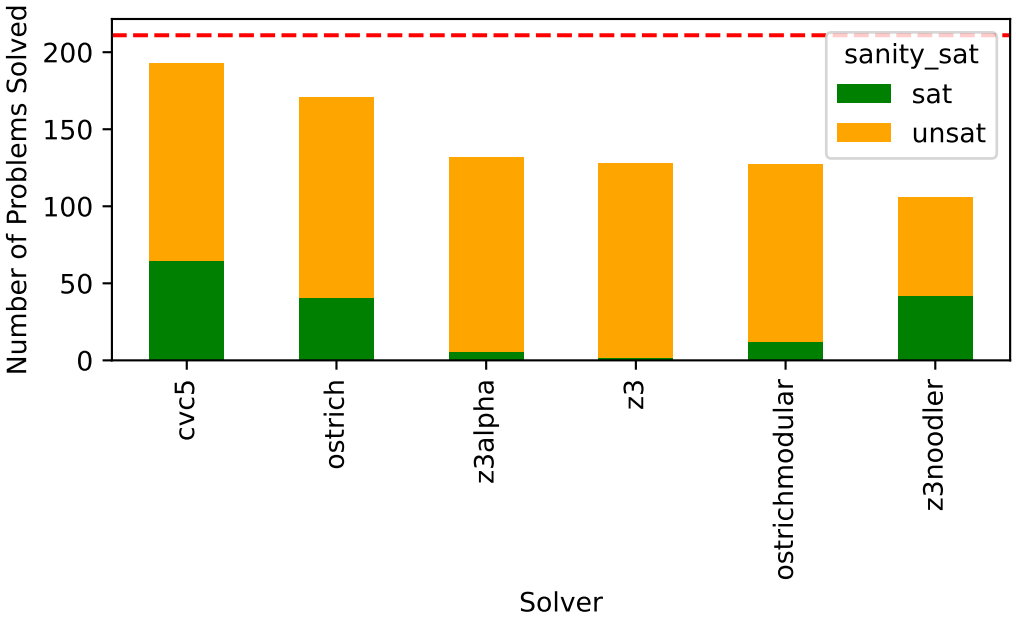


Figure 6: Performance on regular expression replacement with length and regular constraints

If you are interested in a specific category that I did not discuss, check the appendix A or checkout 4.4 to explore the data for yourself.

3.3 Summary

Overall, Z3Noodler comes out on top— probably because its custom Automata Library really gives it a solid, fast foundation for most problem sets. However, when it comes to challenges involving substrings, searches, and replacements (both string and regex), it seems to struggle a bit, and CVC5 takes the lead in those cases. The default Z3 solver and z3Alpha perform very similarly, with z3Alpha offering only minor improvements. On the other hand, Ostrich Modular shows significant gains in the largest categories but loses ground in several others compared to the standard Ostrich.

4 Additional Analyses

4.1 How much SLIA is in SLIA?

To answer this question, let's first answer what is SLIA and what is not. With a similar approach as above, length constraints alone are not enough to qualify as SLIA. So using simple predicates on length and or constants(such as equality)

does not qualify as SLIA. Introducing arithmetic operations such as addition, multiplication etc. does however qualify as LIA. Additionally if the problem contains a variables from the Integer sort, it will be qualified as LIA. We assume all problems to actually have some theory of string operation present. Some problems may contain a definition for a variable that isn't used later on in the actual problem or contain artificial Arithmetic problems eg. $1 - 1 = 0$, to make it easier to classify problems we do count these as SLIA.

We then have to search for LIA propertys in our problem set, for this we can easily use the grep and count the files using wc:

```
grep -rlE '\(- |\(\+ |\(\* |\(div |\(mod |\(abs |Int' benchmarks/QF_SLIA/
|wc -l
```

To count the number of total problems we can use a similiar command but using find instead of grep `find benchmarks/QF_SLIA/ -type f | wc -l`

So, how much SLIA is in SLIA? The answer is out of 84417 problems a total of 51484 contain operations defined above.

That is about 61% of SLIA is actually SLIA.

If we do count simple length constraints we end up at 84398 SLIA problems.

4.2 Achieving Optimal Coverage: A Virtual Portfolio Analysis

Rather than relying on a single solver for every benchmark, this analysis explores a virtual portfolio approach—selecting, for each problem, the solver that performs best. This method establishes an optimistic upper bound on the number of problems that can be solved by today's string solvers.

Remarkably, when combining the strengths of all solvers, only 97 out of the 103,405 total problems remain unsolved.

tagset	count
'word_equations'	9
'regular_constraints'	8
'word_equations', 'length_constraints'	3
'substrings', 'regular_constraints', 'length_constraints'	7
'substrings', 'search'	5
'substrings', 'search', 'length_constraints', 'lia'	2
'str_replace', 'substrings', 'search'	4
'regular_constraints', 'str_replace', 'length_constraints'	56
'regular_constraints', 're_replace', 'length_constraints'	3

The table above shows that even some of the simpler categories—such as word equations or regular constraints—contain unsolved problems. The primary challenge appears to be with replacement operations, particularly when combined with length and regular constraints.

4.3 Non Trivial Benchmarks

By filtering out the universally solved instances, we focus on problems where performance differences become apparent. To keep this section short we will only look at the entire problem set. It might however be interesting to take a look at each category.

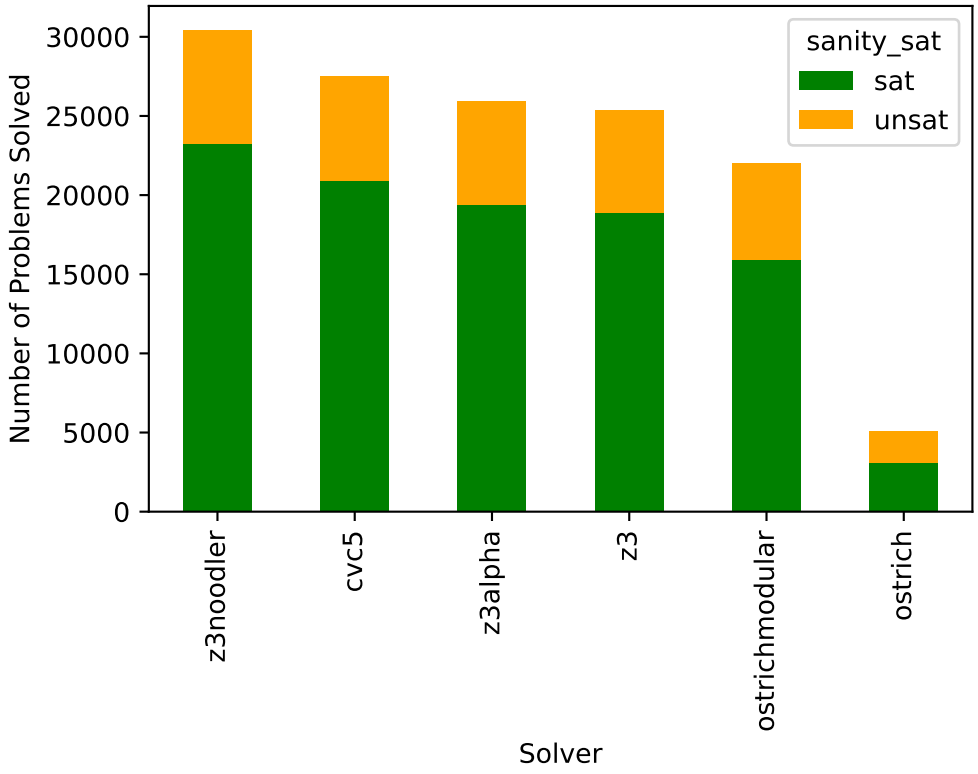


Figure 7: Non trivial problems solved

The general trend of our analysis continues with Z3Noodler leading, followed by CVC5, Z3Alpha, and Z3. Notably, Ostrich Modular shows significant improvements over the standard Ostrich, effectively catching up with the other solvers. This suggests that the recent optimizations in Ostrich Modular are effective in tackling more challenging benchmarks.

4.4 Ostrich vs. Ostrich modular

	ostrich total	ostrich-modular total	unique ostrich	unique ostrich-modular	difference	% change
1	30331	29500	964	133	-831	-2.740000
2	67	21	46	0	-46	-68.660000
3	586	481	110	5	-105	-17.920000
4	46	46	0	0	0	0.000000
5	1886	1866	21	1	-20	-1.060000
6	334	204	131	1	-130	-38.920000
7	32169	34301	755	2887	2132	6.630000
8	6625	22238	11	15624	15613	235.670000
9	1004	958	59	13	-46	-4.580000
10	2471	2651	0	180	180	7.280000
11	629	627	54	52	-2	-0.320000
12	1126	809	334	17	-317	-28.150000
13	414	340	91	17	-74	-17.870000
14	171	127	48	4	-44	-25.730000
15	414	340	91	17	-74	-17.870000
16	65	65	0	0	0	0.000000
17	4	5	0	1	1	25.000000

With categories in order:

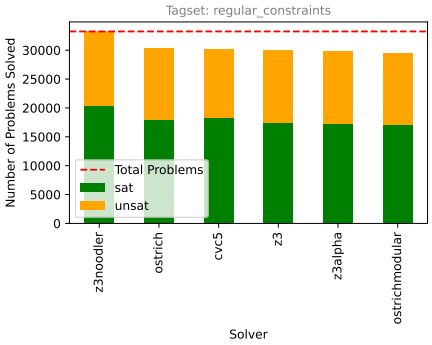
1. (regular-constraints)
2. (regular-constraints, str-replace)
3. (word-equations,)
4. (lia, length-constraints, regular-constraints)
5. (length-constraints, regular-constraints)
6. (length-constraints, word-equations)
7. (substrings, length-constraints, regular-constraints)
8. (lia, length-constraints, substrings, search)
9. (lia, length-constraints, regular-constraints, logic)
10. (search, length-constraints, logic, lia, word-equations)
11. (substrings, search, str-replace)
12. (substrings, search)
13. (length-constraints, regular-constraints, str-replace)
14. (re-replace, length-constraints, regular-constraints)

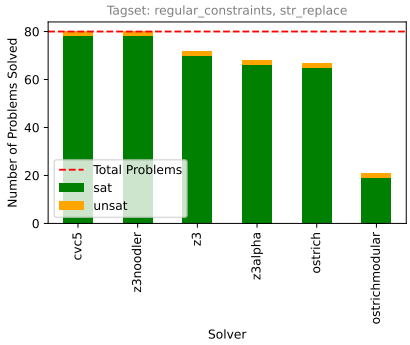
15. (length-constraints, word-equations, logic)
16. (nia)

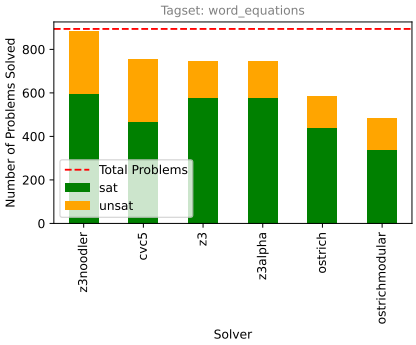
5 Repository and Data

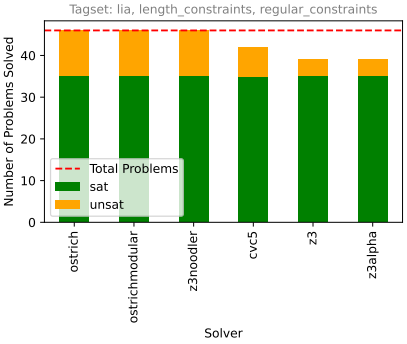
The tools used in this project—including the benchmarking scripts, data sets, and analysis utilities—are publicly available on **GitHub**

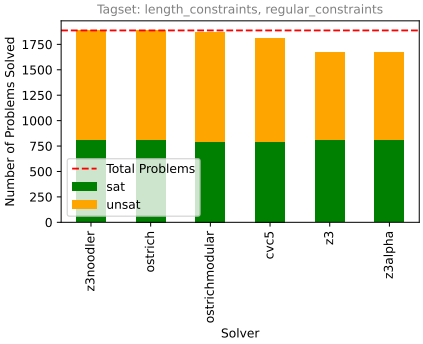
A Full category overview

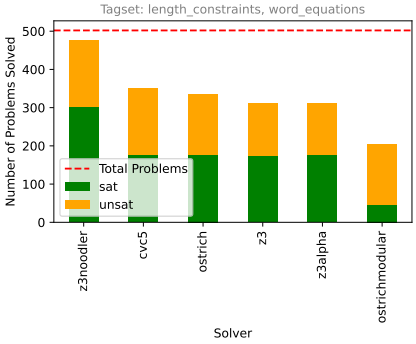

Tagset: regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
z3noodler
ostrich
cvc5
z3
z3alpha
ostrichmodular
Solver

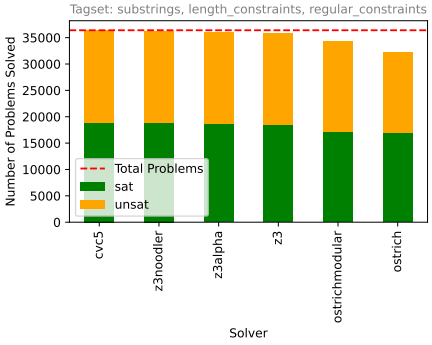

Tagset: regular_constraints, str_replace
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3noodler
z3
z3alpha
ostrich
ostrichmodular
Solver

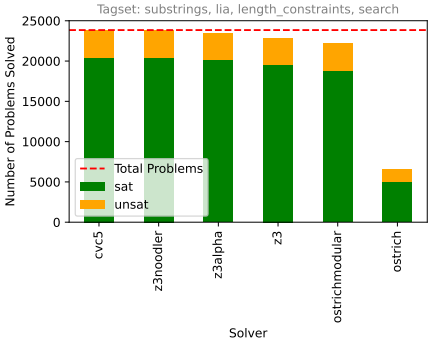

Tagset: word_equations
Number of Problems Solved
Total Problems
sat
unsat
z3noodler
cvc5
z3
z3alpha
ostrich
ostrichmodular
Solver

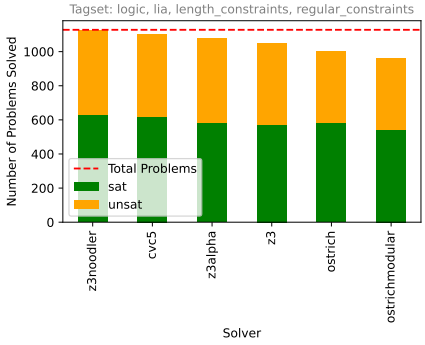

Tagset: lia, length_constraints, regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
ostrich
ostrichmodular
z3noodler
cvc5
z3
z3alpha
Solver

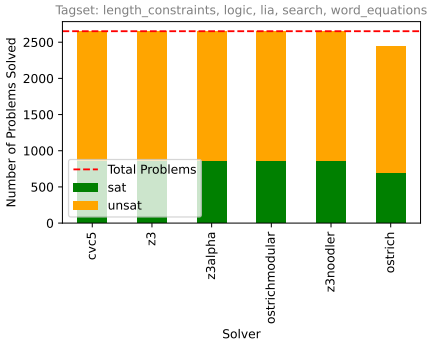

Tagset: length_constraints, regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
z3noodler
ostrich
ostrichmodular
cvc5
z3
z3alpha
Solver

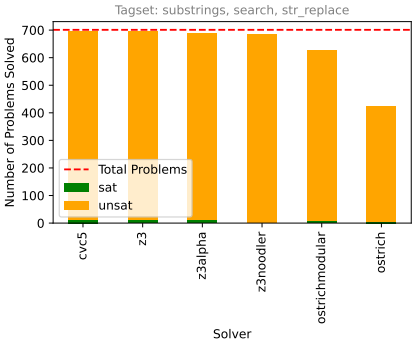

Tagset: length_constraints, word_equations
Number of Problems Solved
Total Problems
sat
unsat
z3noodler
cvc5
ostrich
z3
z3alpha
ostrichmodular
Solver

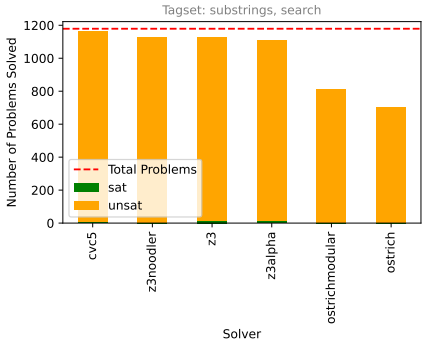

Tagset: substrings, length_constraints, regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3noodler
z3alpha
z3
ostrichmodular
ostrich
Solver

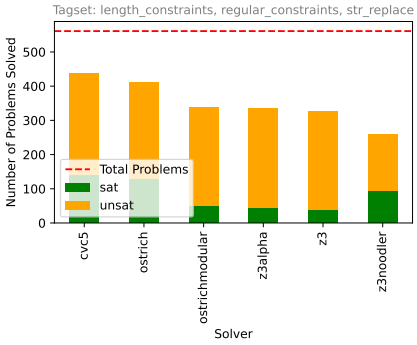

Tagset: substrings, lia, length_constraints, search
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3noodler
z3alpha
z3
ostrichmodular
ostrich
Solver

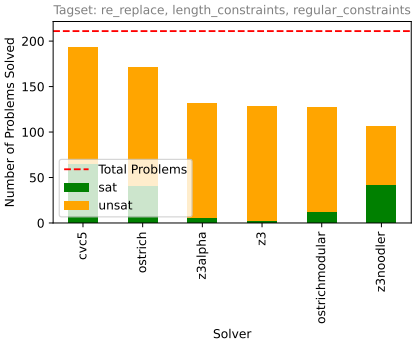

Tagset: logic, lia, length_constraints, regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
z3noodler
cvc5
z3alpha
z3
ostrich
ostrichmodular
Solver

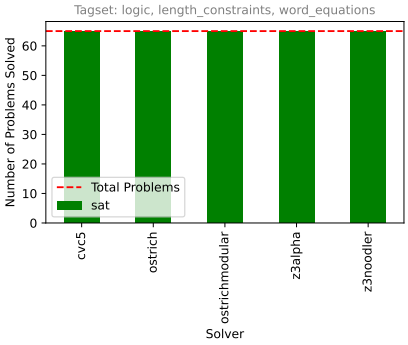

Tagset: length_constraints, logic, lia, search, word_equations
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3
z3alpha
ostrichmodular
z3noodler
ostrich
Solver

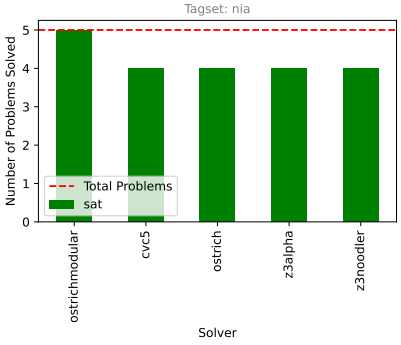

Tagset: substrings, search, str_replace
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3
z3alpha
z3noodler
ostrichmodular
ostrich
Solver


Tagset: substrings, search
Number of Problems Solved
Total Problems
sat
unsat
cvc5
z3noodler
z3
z3alpha
ostrichmodular
ostrich
Solver


Tagset: length_constraints, regular_constraints, str_replace
Number of Problems Solved
Total Problems
sat
unsat
cvc5
ostrich
ostrichmodular
z3alpha
z3
z3noodler
Solver


Tagset: re_replace, length_constraints, regular_constraints
Number of Problems Solved
Total Problems
sat
unsat
cvc5
ostrich
z3alpha
z3
ostrichmodular
z3noodler
Solver


Tagset: logic, length_constraints, word_equations
Number of Problems Solved
Total Problems
sat
cvc5
ostrich
ostrichmodular
z3alpha
z3noodler
Solver


Tagset: nia
Number of Problems Solved
Total Problems
sat
ostrichmodular
cvc5
ostrich
z3alpha
z3noodler
Solver

References

[1] Murphy Berzish et al. "An SMT Solver for Regular Expressions and Linear Arithmetic over String Length". In: *Computer Aided Verification*. Ed. by Alexandra Silva and K. Rustan M. Leino. Cham: Springer International Publishing, 2021, pp. 289–312. ISBN: 978-3-030-81688-9.

[2] Yu-Fang Chen et al. "Solving String Constraints with Lengths by Stabilization". In: *Proc. ACM Program. Lang.* 7.OOPSLA2 (Oct. 2023). DOI: 10.1145/3622872. URL: https://doi.org/10.1145/3622872.

[3] Tianyi Liang et al. "An efficient SMT solver for string constraints". In: *Formal Methods in System Design* 48.3 (June 2016), pp. 206–234. ISSN: 1572-8102. DOI: 10.1007/s10703-016-0247-6. URL: https://doi.org/10.1007/s10703-016-0247-6.

[4] Fang Yu et al. "Optimal sanitization synthesis for web application vulnerability repair". In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ISSTA 2016. Saarbrücken, Germany: Association for Computing Machinery, 2016, pp. 189–200. ISBN: 9781450343909. DOI: 10.1145/2931037.2931050. URL: https://doi.org/10.1145/2931037.2931050.