Z3-Noodler: An Automata-based String Solver

Yu-Fang Chen[1], David Chocholatý[2], Vojtěch Havlena[2],
Lukáš Holík[2], Ondřej Lengál[2], and Juraj Síč[2]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
[2] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Abstract. Z3-Noodler is a fork of Z3 that replaces its string theory solver with a custom solver implementing the recently introduced stabilization-based algorithm for solving word equations with regular constraints. An extensive experimental evaluation shows that Z3-Noodler is a fully-fledged solver that can compete with state-of-the-art solvers, surpassing them by far on many benchmarks. Moreover, it is often complementary to other solvers, making it a suitable choice as a candidate to a solver portfolio.

1 Introduction

Recently, many tools for solving string constraints have been developed, motivated mainly by techniques for finding security vulnerabilities such as SQL injection or cross-site scripting (XSS) in web applications [34,35,36]. String solving has also found its applications in, e.g., analysis of access user policies in Amazon Web Services [26,8,39] or smart contracts [7]. Solvers for string constraints are usually implemented as string theory solvers inside SMT solvers, such as cvc5 [9] or Z3 [31], allowing combination with other theories, most commonly the theory of integers for string lengths. Other well known string solvers include Z3str3RE [13,12], Z3-Trau [1], Z3str4 [30], OS-TRICH [19], and others.

In this paper, we present Z3-Noodler 1.0.0 [47], a fork of Z3 4.12.2 where the string theory solver is replaced with the *stabilization-based procedure* for solving string (dis)equations with regular and length constraints [14,20]. The procedure makes heavy use of *nondeterministic finite automata* (NFAs) and operations over them, for which we use the efficient Mata library for NFAs [23,29].

The presented version implements multiple improvements over a previous Z3-Noodler prototype from [20]. Firstly, it extends the support for string predicates from the SMT-LIB string theory standard [11] by (1) applying smarter and more specific axiom saturation and (2) adding support for their solving inside the decision procedure (e.g., for the ¬contains predicate). It also implements various optimizations (e.g., for regular constraints handling) and other decision procedures, e.g., the *Nielsen transformation* [32] for quadratic equations and a procedure for regular language (dis)equations; moreover, we added heuristics for choosing the best decision procedure to use.

We compared Z3-Noodler with other string solvers on standard SMT-LIB benchmarks [10,42,43]. The results indicate that Z3-Noodler is competitive, superior especially on benchmarks containing mostly regular constraints and word (dis)equations, and that the improvements since [20] had a large impact on the number of solved instances as well as its overall performance.

2   Architecture

Z3-Noodler replaces the string theory solver in the DPLL(T)-based SMT solver Z3 [31] (version 4.12.2) with our string solver Noodler [14], which is based on the *stabilization algorithm* (cf. Section 3). DPLL(T)-based solvers in general combine a SAT solver providing satisfying assignments to the Boolean skeleton of a formula with multiple theory solvers for checking conjunctions of theory literals.

Z3-Noodler still uses the infrastructure of Z3, most importantly the parser, string theory rewriter and the *linear integer arithmetic* (LIA) solver. The Z3 parser takes formulae in the SMT-LIB format [10], where Z3-Noodler can handle nearly all predicates/functions (such as `substr`, `len`, `at`, `replace`, regular membership, word equations, etc.) in the string theory as defined by SMT-LIB [11].

Even though we do use the string theory rewriter of Z3, we disabled those rewritings that do not benefit our core string solver. For instance, we removed rules that rewrite regular membership constraints to other types of constraints since solving regular constraints and word equations using our stabilization-based approach is efficient.

The interaction of the Noodler solver with Z3 is shown in Fig. 1 and works as follows. Upon receiving a satisfying Boolean assignment from the SAT solver (❶), we first remove irrelevant assignments (using Z3's relevancy propagation), which allows us to work with smaller instances and return more general theory lemmas. A theory assignment obtained from the Boolean assignment consists of string (dis)equations, regular constraints, and, possibly, predicates that were not *axiom-saturated* before (cf. Section 3).
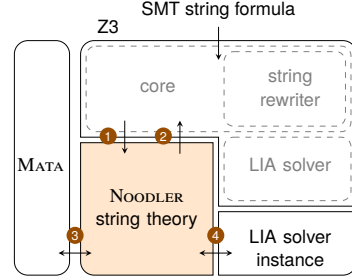

Fig. 1: Architecture of Z3-Noodler

The core Noodler string decision procedure then reduces the conjunction of string literals to a LIA constraint over string lengths, and returns it to Z3 as a theory lemma (❷), to be solved together with the rest of the input arithmetic constraints by Z3's internal LIA solver. Noodler implements a couple of decision procedures (discussed in Section 3), heavily employing the Mata automata library (version 0.109.0) [29] (❸). As an optimization of the theory lemma generation, when the string constraint reduces into a disjunction of LIA length constraints, we check the satisfiability of individual disjuncts (generated lazily on demand) separately in order to get a positive answer as soon as possible. For testing the disjuncts, the current solver context is cloned and queried about satisfiability of the LIA constraint conjoined with the disjunct (❹).

3   String Theory Core

In this section, we provide details about Z3-Noodler's string theory implementation, including initial axiom saturation, proprocessing, the core procedure, and limitations.

Axiom Saturation. In order to best utilize the power of Z3's internal LIA solver during the generation of a satisfiable assignment, we saturate the input formula with length-aware

theory axioms and axioms for string predicates (this happens during Z3's processing of the input formula, before the main SAT solver starts generating assignments). We can then avoid checking SAT assignments that trivially violate length conditions. Most importantly, we add length axioms $\mathtt{len}(t_1) \geq 0$, $\mathtt{len}(t_1.t_2) = \mathtt{len}(t_1) + \mathtt{len}(t_2)$ where t_1, t_2 are arbitrary string terms, and $\mathtt{len}(t_1) = \mathtt{len}(t_2)$ for the word equation $t_1 = t_2$.

Moreover, for string functions/predicates, Noodler saturates the original formula with an equivalent formula composed of word (dis)equations and length/regular constraints, which are more suitable for our core procedure (e.g., for $\neg\mathtt{contains}(s, \mathtt{"abc"})$ in the input formula, we add the regular constraint $s \notin \Sigma^*\mathrm{abc}\Sigma^*$). We use different saturation rules for instances of predicates with concrete values. For instance, for $\mathtt{substr}(s, 4, 1)$, we add just the term $\mathtt{at}(s, 4)$. On the other hand, for $\mathtt{substr}(s, t_i, t_j)$, where s is a string term and t_i, t_j are general integer terms (possibly containing variables), we need to add a more general formula talking about the prefix and suffix of s of given lengths. The original predicate occurrence is then removed from received assignments by Noodler (Z3 does not allow to remove parts of the original formula).

Decision Procedures. Z3-Noodler's string theory core contains several complementary decision procedures. The main one is the *stabilization-based* algorithm for solving word equations with regular constraints introduced in [14] and later extended with efficient handling of length constraints and disequations [20]. The stabilization-based algorithm starts, for every string variable, with an NFA encoding regular constraints on the variable and iteratively refines the NFA according to the word equations until the stability condition is achieved. The stability condition holds when, for every word equation, the language of the left-hand side (obtained as the language of the concatenation of NFAs for variables and string literals) equals the language of the right-hand side. When stability is achieved, length constraints of the solutions are generated and passed to the LIA solver. The algorithm is complete for the *chain-free* [5] combinations of equations, regular and length constraints, together with unrestricted disequations, making it the largest known decidable fragment of these types of constraints.

The stabilization-based decision procedure starts by inductively converting the initial regular constraints into NFAs. During the construction, we utilize eager simulation-based reduction [16,17] with on-demand determinization and minimization.

For an efficient handling of *quadratic* equations (systems of equations with at most two occurrences of each variable) with lengths, Noodler implements a decision procedure based on the *Nielsen transformation* [32]. The algorithm constructs a graph corresponding to the system and reasons about it to determine if the input formula is satisfiable or not [38,22]. If the system contains length variables, we also create a counter automaton corresponding to the Nielsen graph (in a similar way as in [28]). In the subsequent step, we contract edges, saturating the set of self-loops and, finally, we iteratively generate flat counter sub-automata (a flat counter automaton only allows cycles that are self-loops), which are later transformed into LIA formulae describing lengths of all possible solutions.

In order to solve *(dis)equations of regular expressions*, we reduce the problem to reasoning about the corresponding NFAs (similarly as for regular constraints handling). In particular, we use efficient NFA equivalence and universality checking from Mata, which implements advanced antichain-based algorithms [46,6].

Preprocessing. Each decision procedure employs a sequence of preprocessing rules transforming the string constraint to a more suitable form. Our portfolio of rules includes transformations reducing the number of equations by a conversion to regular constraints, propagating epsilons and variables over equations, underapproximation rules, and rules reducing the number of disequations (cf. [20]). On top of that, Z3-Noodler employs information about length-equivalent variables allowing to infer simpler constraints (e.g., for $xy = zw$ with $\texttt{len}(x) = \texttt{len}(z)$, we can infer $y = w$). Z3-Noodler also checks for simple unsatisfiable patterns for early termination. A sequence of preprocessing rules is composed for each of the decision procedures differently, maximizing their strengths.

Supported String Predicates and Limitations. Z3-Noodler currently supports handling of basic string predicates `replace`, `substr`, `at`, `indexof`, `prefix`, `suffix`, `contains`, and a limited support for `¬contains`. From the set of extended constraints, the core solver currently does not support the `replace_all` function (and variants of replacement based on regular expressions) and `to`/`from_int` conversions. The decision procedures used in Z3-Noodler make it complete for the chain-free fragment with unbounded disequations and regular constraints [20], and quadratic equations. Outside this fragment, our theory core is sound but incomplete.

4   Experiments

Tools and environment. We compared Z3-Noodler with the following state-of-the-art tools: cvc5 [9] (version 1.0.8), Z3 [31] (version 4.12.2), Z3str3RE [13,12], Z3str4 [30], OSTRICH [19][3], and Z3-Noodler^{pr} (version 0.1.0 used in [20]). We did not compare with Z3-Trau [2] as it is no longer under active development and gives incorrect results on newer benchmarks. The experiments were executed on a workstation with an Intel Xeon Silver 4314 CPU @ 2.4 GHz with 128 GiB of RAM running Debian GNU/Linux. The timeout was set to 120 s, memory limit was set to 8 GiB.

Benchmarks. The benchmarks come from the SMT-LIB [10] repository, specifically categories `QF_S` [42] and `QF_SLIA` [43]. These benchmarks were also used in SMT-COMP'23 [41], in which Z3-Noodler participated (version 0.2.0). As Z3-Noodler does not support `to`/`from_int` conversions and `replace_all`-like predicates, we excluded formulae whose satisfiability checking needs their support. Based on the occurrences of different kinds of constraints, we divide the benchmarks into three groups:

Regex This category contains formulae with dominating regular membership and length constraints. It consists of AutomatArk [13], Denghang, StringFuzz [15], and Sygus-qgen benchmark sets. We excluded 1,568 formulae from StringFuzz that require support of the `to_int` predicate.

Equations The formulae in this category consist mostly of word equations with length constraints and a small amount of other predicates. It contains Kaluza [40,27], Kepler [25], Norn [3,4], Slent [44], Slog [45], Webapp, and Woorpje [24] benchmark sets. We excluded 414 formulae from Webapp that require support of `replace_all`, `replace_re`, and `replace_re_all` predicates.

---

[3] Latest commit `70d01e2d2`, run with `-portfolio=strings` option.

Table 1: Results of experiments on all benchmark sets. For each tool and benchmark set (as well as whole groups under Σ), we give the number of unsolved instances. Results for tools with the highest number of solved instances are in **bold**. Numbers with * contain also incorrect results.

	Regex					Equations								Predicates-small				PyEx
	Aut	Den	StrFuzz	Syg	Σ	Kal	Kep	Norn	Slent	Slog	Web	Woo	Σ	StrInt	Leet	StrSm	Σ	
Included	15,995	999	10,050	343	*27,387*	19,432	587	1,027	1,128	1,976	267	809	*25,226*	11,669	2,652	1,670	*15,991*	23,845
Unsupported	0	0	1,568	0	*1,568*	0	0	0	0	0	414	0	*414*	5,299	0	210	*5,509*	0
Z3-Noodler	62	**0**	**0**	**0**	*62*	259	**4**	**0**	5	**0**	**0**	243	*511*	4	4	55	*63*	4,424
cvc5	94	18	1037	**0**	*1149*	**0**	240	85	22	**0**	40	54	***441***	**0**	**0**	4	*4*	**34**
Z3	113	118	340	**0**	*571*	164	313	124	74	71	61	**25**	*832*	4	**0**	32	*36*	1,071
Z3str4	60	4	27	**0**	*91*	174	254	73	73	16	62	78	*730*	5	4	37	*46*	570
OSTRICH	**55**	15	229	**0**	*299*	288	387	1	130	7	65	53	*931*	37	26	*106	*169	12,290
Z3str3RE	66	27	*143	1	*237	*144	311	133	87	55	*104	*118	*952	64	192	*179	*435	17,764
Z3-Noodler^{pr}	86	1	*1,014	**0**	*1,101	508	575	**0**	6	**0**	*3	256	*1,348	40	29	*493	*562	*13,362

Predicates-small Although Z3-Noodler focuses mainly on word equations with length and regular constraints, the evaluation includes also a group consisting of smaller formulae that use string predicates such as `substr`, `at`, `contains`, etc. It is formed from FullStrInt, LeetCode, and StrSmallRw [33] benchmark sets. We removed 5,509 formulae containing the `to/from_int` predicates from FullStrInt and StrSmallRw.

We also consider the PyEx [37] benchmark, which we do not put into any of these groups, as it contains large formulae with complex predicates (`substr`, `contains`, etc.). We note that we omit the small Transducer+ [18] benchmark because it contains exclusively formulae with `replace_all`.

Results. We show the number of unsolved instances for each benchmark and tool (as well as whole groups) in Table 1. Some tools gave incorrect results (determined by comparing to the output of cvc5 and Z3) for some benchmarks. Usually, this was less than 10 instances, except for Z3str3RE on String-Fuzz and StrSmallRw (50 and 12 incorrect results respectively) and Z3-Noodler^{pr} on StrSmallRw (218 incorrect results). Table 2 then shows the average run times and their standard deviations for solved instances for each category and tool.

Table 2: Average run times (in seconds) of solved instances and their standard deviations.

	Reg		Eq		Pred	
	avg	std	avg	std	avg	std
Z3-Noodler	**0.11**	1.35	**0.11**	2.13	0.11	2.16
cvc5	1.17	8.51	**0.11**	2.15	0.03	0.15
Z3	1.92	9.71	0.18	2.83	0.04	0.42
Z3str4	0.35	2.00	0.25	3.40	0.02	0.31
OSTRICH	4.29	8.67	4.28	9.28	12.71	15.08
Z3str3RE	0.31	3.28	0.13	2.72	**0.01**	0.08
Z3-Noodler^{pr}	0.27	2.86	0.12	2.93	0.09	1.69

The results show that Z3-Noodler outperforms other tools on the **Regex** group (in particular on Denghang, StringFuzz, and Sygus-qgen) both in the number of solved instances and the average run time. Only on AutomatArk it cannot solve the most formulae (but it solves only 7 less than the winner OSTRICH, while being much faster).

On the **Equations** group, Z3-Noodler also outperforms other tools on most of the benchmarks. In particular on Kepler, Norn, Slent, Slog, and Webapp. On Kaluza, it is outperformed by other tools, but it still solves the vast majority of formulae. Z3-Noodler has worse performance on Woorpje, which seems to be a synthetic benchmark generated to showcase the strength of a specialized algorithm [24] (this benchmark is the reason for Z3-Noodler taking the second place in the whole group). With 0.11 s, Z3-Noodler and cvc5 have the lowest average run time.

(a) Z3-Noodler vs. cvc5   (b) Z3-Noodler vs. Z3   (c) Z3-Noodler vs. VBS
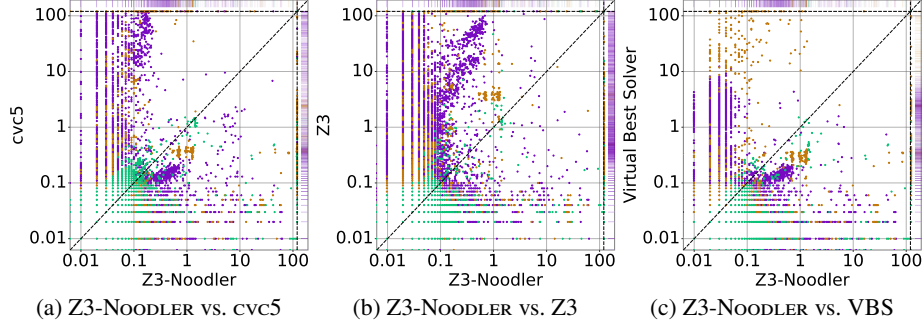
Fig. 2: Comparison of Z3-Noodler with cvc5, Z3, and the virtual best solver (VBS). Times are in seconds, axes are logarithmic. Dashed lines represent timeouts (120 s). Colours distinguish groups: ● **Regex**, ● **Equations**, and ● **Predicates-small**.

The winner of **Predicates-small** is cvc5. In particular, on FullStrInt and LeetCode the difference with Z3-Noodler is equally 4 instances and on StrSmallRw the difference is 51 cases. The average time of Z3-Noodler is also a bit higher, with 0.11 s for Z3-Noodler compared to the 0.03 s for cvc5. Similarly, Z3-Noodler is outperformed by cvc5, Z3, and Z3str4 on PyEx. Indeed, we have not optimized Z3-Noodler for formulae with large numbers of predicates yet. The results of Z3-Noodler could, however, be further improved by proper axiom saturation for predicates or lazy predicate evaluation.

In Fig. 2 we show scatter plots comparing running time of Z3-Noodler with cvc5, Z3, and virtual best solver (VBS; a solver that takes the best result from all tools other than Z3-Noodler) on all three benchmark groups. The plots show that Z3-Noodler outperforms the competitors on a vast number of instances, in many cases being complementary to them. To validate this claim, we also checked how different solvers contribute to a portfolio. That is, we took the VBS in-

Table 3: Evaluating solver contribution to a portfolio. Times are in seconds.

	Regex		Equations	
	Unsolved	Time	Unsolved	Time
VBS⁺	1	427	19	1,304
VBS⁺- Z3-Noodler	1	2,914	131	6,830
VBS⁺- cvc5	1	549	145	1,401
VBS⁺- Z3	1	430	29	1,579
VBS⁺- Z3str4	1	473	19	1,416
VBS⁺- OSTRICH	1	427	21	1,270
VBS⁺- Z3str3RE	1	510	20	1,307
cvc5 + Z3 + Z3-Noodler	1	608	22	1,471
cvc5 + Z3	278	27,916	303	2,805

cluding Z3-Noodler (VBS⁺) and then checked how well the portfolio works without each of the solvers. Table 3 shows the results on the **Regex** and **Equations** groups (we omit **Predicates-small**, where Z3-Noodler does not help the portfolio). The results show that on the two groups, Z3-Noodler is the most valuable solver in the portfolio. We also include results on the small portfolio of Z3 and cvc5 (with and without Z3-Noodler) showing that, on the two groups, using just these three solvers is almost as good as using the whole portfolio of all solvers.

Comparing with the older version Z3-Noodler^{pr} from [20], we can see that there is a significant improvement in most benchmarks, most significantly in AutomatArk, StringFuzz, Kepler, StrSmallRw, and Kaluza. We note that adding more complicated algorithm selection strategies significantly improved the overall performance of Z3-

Noodler, but, on the other hand, decreased the performance on Kaluza (cf. [20]). Better results in AutomatArk and StringFuzz stem from the improvements in Mata and from heuristics tailored for regular expressions handling. Including Nielsen's algorithm [32] has the largest impact on the Kepler benchmark. The improvement on predicate-intensive benchmarks is caused by optimizations in axiom saturation for predicates. The older version also had multiple bugs that have been fixed in the current version.

Acknowledgments

Data Availability Statement

An environment with the tools and data used for the experimental evaluation in the current study is available at [21].

References

1. Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Dolby, J., Janků, P., Lin, H., Holík, L., Wu, W.: Efficient handling of string-number conversion. In: Proc. of PLDI'20. pp. 943–957. ACM (2020). https://doi.org/10.1145/3385412, https://doi.org/10.1145/3385412

2. Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Holík, L., Rezine, A., Rümmer, P.: Trau: SMT solver for string constraints. In: Bjørner, N.S., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–5. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8602997, https://doi.org/10.23919/FMCAD.2018.8602997

3. Abdulla, P.A., Atig, M.F., Chen, Y., Holík, L., Rezine, A., Rümmer, P., Stenman, J.: String constraints for verification. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 150–166. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_10, https://doi.org/10.1007/978-3-319-08867-9_10

4. Abdulla, P.A., Atig, M.F., Chen, Y., Holík, L., Rezine, A., Rümmer, P., Stenman, J.: Norn: An SMT solver for string constraints. In: Kroening, D., Pasareanu, C.S. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9206, pp. 462–469. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_29, https://doi.org/10.1007/978-3-319-21690-4_29

5. Abdulla, P.A., Atig, M.F., Diep, B.P., Holík, L., Janků, P.: Chain-free string constraints. In: Chen, Y., Cheng, C., Esparza, J. (eds.) Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11781, pp. 277–293. Springer (2019). https://doi.org/10.1007/978-3-030-31784-3_16, https://doi.org/10.1007/978-3-030-31784-3_16

6. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: TACAS'10. LNCS, vol. 6015, pp. 158–174. Springer (2010)

7. Alt, L., Blicha, M., Hyvärinen, A.E.J., Sharygina, N.: SolCMC: Solidity compiler's model checker. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13371, pp. 325–338. Springer (2022). `https://doi.org/10.1007/978-3-031-13185-1_16`, `https://doi.org/10.1007/978-3-031-13185-1_16`

8. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., Varming, C.: Semantic-based automated reasoning for aws access policies using smt. In: 2018 Formal Methods in Computer Aided Design (FMCAD). pp. 1–9 (2018). `https://doi.org/10.23919/FMCAD.2018.8602994`

9. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength smt solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer International Publishing, Cham (2022)

10. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)

11. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB): Strings. `https://smtlib.cs.uiowa.edu/theories-UnicodeStrings.shtml` (2023)

12. Berzish, M., Day, J.D., Ganesh, V., Kulczynski, M., Manea, F., Mora, F., Nowotka, D.: Towards more efficient methods for solving regular-expression heavy string constraints. Theor. Comput. Sci. **943**, 50–72 (2023). `https://doi.org/10.1016/j.tcs.2022.12.009`, `https://doi.org/10.1016/j.tcs.2022.12.009`

13. Berzish, M., Kulczynski, M., Mora, F., Manea, F., Day, J.D., Nowotka, D., Ganesh, V.: An SMT solver for regular expressions and linear arithmetic over string length. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12760, pp. 289–312. Springer (2021). `https://doi.org/10.1007/978-3-030-81688-9_14`, `https://doi.org/10.1007/978-3-030-81688-9_14`

14. Blahoudek, F., Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Word equations in synergy with regular constraints. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) Formal Methods. pp. 403–423. Springer International Publishing, Cham (2023)

15. Blotsky, D., Mora, F., Berzish, M., Zheng, Y., Kabir, I., Ganesh, V.: StringFuzz: A fuzzer for string solvers. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification. pp. 45–51. Springer International Publishing, Cham (2018)

16. Bustan, D., Grumberg, O.: Simulation based minimization. In: Proceedings of CADE-17. LNCS, vol. 1831, pp. 255–270. Springer (2000)

17. Cécé, G.: Foundation for a series of efficient simulation algorithms. In: LICS'17. pp. 1–12. IEEE Computer Society (2017)

18. Chen, T., Hague, M., He, J., Hu, D., Lin, A.W., Rümmer, P., Wu, Z.: A decision procedure for path feasibility of string manipulating programs with integer data type. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 325–342. Springer (2020). `https://doi.org/10.1007/978-3-030-59152-6_18`, `https://doi.org/10.1007/978-3-030-59152-6_18`

19. Chen, T., Hague, M., Lin, A.W., Rümmer, P., Wu, Z.: Decision procedures for path feasibility of string-manipulating programs with complex operations. Proc. ACM Program. Lang. **3**(POPL), 49:1–49:30 (2019). `https://doi.org/10.1145/3290362`, `https://doi.org/10.1145/3290362`

20. Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Solving string constraints with lengths by stabilization. Proc. ACM Program. Lang. **7**(OOPSLA2) (oct 2023). `https://doi.org/10.1145/3622872`

21. Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Z3-Noodler: An automata-based string solver (Oct 2023). `https://doi.org/10.5281/zenodo.10041441`, `https://doi.org/10.5281/zenodo.10041441`

22. Chen, Y.F., Havlena, V., Lengál, O., Turrini, A.: A symbolic algorithm for the case-split rule in solving word constraints with extensions. Journal of Systems and Software **201**, 111673 (2023). `https://doi.org/https://doi.org/10.1016/j.jss.2023.111673`, `https://www.sciencedirect.com/science/article/pii/S0164121223000687`

23. Chocholatý, D., Fiedor, T., Havlena, V., Holík, L., Hruška, M., Lengál, O., Síč, J.: Mata: A fast and simple finite automata library. In: Proc. of TACAS'24. LNCS, Springer (2024)

24. Day, J.D., Ehlers, T., Kulczynski, M., Manea, F., Nowotka, D., Poulsen, D.B.: On solving word equations using SAT. In: Filiot, E., Jungers, R.M., Potapov, I. (eds.) Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11674, pp. 93–106. Springer (2019). `https://doi.org/10.1007/978-3-030-30806-3_8`, `https://doi.org/10.1007/978-3-030-30806-3_8`

25. Le, Q.L., He, M.: A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In: Ryu, S. (ed.) Programming Languages and Systems. pp. 350–372. Springer International Publishing, Cham (2018)

26. Liana Hadarean: String solving at Amazon. `https://mosca19.github.io/program/index.html` (2019), presented at MOSCA'19

27. Liang, T., Reynolds, A., Tsiskaridze, N., Tinelli, C., Barrett, C., Deters, M.: An efficient SMT solver for string constraints. Formal Methods in System Design **48**(3), 206–234 (2016)

28. Lin, A.W., Majumdar, R.: Quadratic word equations with length constraints, counter systems, and presburger arithmetic with divisibility. In: Automated Technology for Verification and Analysis. pp. 352–369. Springer International Publishing, Cham (2018)

29. Mata: An efficient automata library (2023), `https://github.com/VeriFIT/mata`

30. Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V.: Z3str4: A multi-armed string solver. In: Huisman, M., Pasareanu, C.S., Zhan, N. (eds.) Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13047, pp. 389–406. Springer (2021). `https://doi.org/10.1007/978-3-030-90870-6_21`, `https://doi.org/10.1007/978-3-030-90870-6_21`

31. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS'08. LNCS, vol. 4963, pp. 337–340. Springer (2008), `https://doi.org/10.1007/978-3-540-78800-3_24`

32. Nielsen, J.: Die isomorphismen der allgemeinen, unendlichen gruppe mit zwei erzeugenden. Mathematische Annalen **78**(1), 385–397 (1917)

33. Nötzli, A., Reynolds, A., Barbosa, H., Niemetz, A., Preiner, M., Barrett, C., Tinelli, C.: Syntax-guided rewrite rule enumeration for SMT solvers. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing – SAT 2019. pp. 279–297. Springer International Publishing, Cham (2019)

34. OWASP: Top 10. `https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf` (2013)

35. OWASP: Top 10. `https://owasp.org/www-project-top-ten/2017/` (2017)

36. OWASP: Top 10. `https://owasp.org/Top10/` (2021)

37. Reynolds, A., Woo, M., Barrett, C., Brumley, D., Liang, T., Tinelli, C.: Scaling up DPLL(T) string solvers using context-dependent simplification. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 453–474. Springer International Publishing, Cham (2017)

38. Robson, J.M., Diekert, V.: On quadratic word equations. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 217–226. Springer (1999)

39. Rungta, N.: A billion SMT queries a day (invited paper). In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13371, pp. 3–18. Springer (2022). `https://doi.org/10.1007/978-3-031-13185-1_1`, `https://doi.org/10.1007/978-3-031-13185-1_1`

40. Saxena, P., Akhawe, D., Hanna, S., Mao, F., McCamant, S., Song, D.: Kaluza web site (2023), `https://webblaze.cs.berkeley.edu/2010/kaluza/`

41. SMT-COMP'23: `https://smt-comp.github.io/2023/` (2023)

42. SMT-LIB: `https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_S` (2023)

43. SMT-LIB: `https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_SLIA` (2023)

44. Wang, H.E., Chen, S.Y., Yu, F., Jiang, J.H.R.: A symbolic model checking approach to the analysis of string and length constraints. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. p. 623–633. ASE 2018, Association for Computing Machinery, New York, NY, USA (2018). `https://doi.org/10.1145/3238147.3238189`, `https://doi.org/10.1145/3238147.3238189`

45. Wang, H., Tsai, T., Lin, C., Yu, F., Jiang, J.R.: String analysis via automata manipulation with logic circuit representation. In: CAV'16. LNCS, vol. 9779, pp. 241–260. Springer (2016)

46. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.: Antichains: A new algorithm for checking universality of finite automata. In: CAV'06. LNCS, vol. 4144, pp. 17–30. Springer (2006)

47. Z3-Noodler: Automata-based string solver (2023), `https://github.com/VeriFIT/z3-noodler`