Z3-Noodler

System description for SMT-comp 2023

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

[1] Brno University of Technology, Brno
[2]Academia Sinica, Taipei

Abstract

This is a brief overview of the string solver Z3-Noodler entering SMT-comp 2023. It is based on the SMT solver Z3 [1] in which it replaces the solver for the theory of strings. It is built around the equation stabilisation algorithm from [2], implemented on top of the automata library Mata.

1 Overview

Z3-Noodler[1] is a string solver that targets string constraints such as those that occur at analysis of programs, regular filters, policy descriptions, etc. It is built on top of the SMT-solver Z3 [1] and automata library Mata[2]. The core of the string solver is the *equation stabilisation* algorithm from [2].

From the SMT-lib string language, the core solver Noodler handles the basic string constraints, word equations, regular constraints, linear arithmetic (LIA) constraints on string lengths, and extended string predicates such as `str.replace`, `str.substring`, `str.at`, `str.indexof`, `str.prefixof`, `str.suffixof`, `str.contains`, `str.replace_re`, `str.contains`, and has a limited support for negated `str.contains`. These extended predicates are to the basic string constraints. The core solver does not support the `str.replace_all` predicate, conversions between strings and integers, and transducer constraints, but Z3-Noodler may still handle predicates unsupported by the core solver if they are eliminated by the theory rewriter of Z3.

Z3-Noodler is implemented in C++, as well as Z3 and Mata. It is a complete reimplementation of the Python prototype presented in [2]. It is in early stages of the development, as well as the Mata automata library.

2 Core Algorithm

The core of Z3-Noodler is the *stabilisation algorithm* introduced in [2] that solves word equations combined with regular membership constraints. In a nutshell, a word equation $x_1 \dots x_m = x_{m+1} \dots x_n$ and a set of regular membership constraints $x \in L_x$ are said to be *stable* if the concatenation of the languages on the left, $L_{x_1} \cdots L_{x_m}$, is equal to the concatenation of the languages on the right, $L_{x_{m+1}} \cdots L_{x_n}$. It is shown in [2] that a stable system has a solution. The stabilisation algorithm uses extensions of classical nondeterministic automata constructions, implemented in Mata, to refine the languages until stability is achieved or some of the languages becomes empty, indicating unsatisfiability. The strength of the stabilisation algorithm is that it makes a good use of symbolic automata representation of equations, it does not require equation splitting (enumerating alignments of the left and right hand side), and it eliminates much of redundant automata constructions needed for instance in [3] and derived approaches. It leverages the efficiency of Mata (efficiency of an early prototype of Mata has been measured in [4]). The core solver combines the stabilisation algorithm with the older *equation alignment and automata splitting* of [3] in order to derive LIA formulae characterising lengths of strings solutions. As alignment and splitting is more expensive, it is used only when it is necessary to isolate a string variable involved in length constraints from word equations. The algorithm is complete for the *chain-free* combinations of equations, regular constraints, and LIA over string lengths, a largest known decidable fragment of these types of constraints, introduced in [5].

3 High Level Architecture

The core string solving algorithm replaces the string theory solver of Z3. Z3-Noodler still uses the infrastructure of Z3 and the theory rewriter. However, since the core string solver is quite different from the original Z3 string solver, some of the rewritings are undesired and we disable them. For instance, we remove rules that rewrite regular membership constraints to other types of constraints (as solving them in our approach is very efficient), and we eliminate

[1] `https://github.com/VeriFIT/z3-noodler`
[2] `https://github.com/VeriFIT/mata`

rewriting rules that produce if-then-else predicates, not supported by the core string solver.

The interaction of the core solver with Z3 is organized as follows. Upon receiving a satisfying assignment from the SAT-solver (a conjunction of string literals), the core string solver reduces the string conjunction to a LIA constraint over string lengths, and returns it to Z3 as theory lemma, to be solved together with the rest of the input arithmetic constraints by the Z3's internal LIA solver. As an optimization of this process, when the string constraint reduces into a disjunction of LIA length constraints, then each disjunct is passed to Z3 individually— the current solver context is cloned and queried about satisfiability of the LIA constraint conjoined with the disjunct. The disjuncts are generated lazily on demand.

4 Preprocessing

The core string solver uses a set of simple rewriting rules that infer length constraints form equations (such as $|x| = 0$ when x must be the empty string or $|x| + |y| = |u| + |v|$ for an equation $xy = uv$), eliminate trivial equations such as $x = y$, simplify equations when a string variables is known to equal the empty string, transform equations to regular membership constraint when possible ($x = uv$ becomes $x \in L_u \cdot L_v$ if u, v do not appear elsewhere and are constraint by the languages L_u and L_v, respectively), and simplify equations such as $xyz = xuz$ into $y = u$. Besides the semantics preserving rewriting rules, we use one under-approximating rule that replaces a membership of a variable in a co-finite language by a length constraint that excludes all the lengths of words outside the language.

5 Noodler at SMT-comp 2023

We are submitting a version 0.1.0 of Z3-Noodler to participate in the single-query track division QF-S and QF-SLIA. The submitted version is linked against version 0.58.0 of Mata and version 4.12.0 of Z3.

References

[1] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3.

[2] Frantisek Blahoudek et al. "Word Equations in Synergy with Regular Constraints". In: *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*. Ed. by Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker. Vol. 14000. Lecture Notes in Computer Science. Springer, 2023, pp. 403–423. DOI: 10.1007/978-3-031-27481-7_23. URL: https://doi.org/10.1007/978-3-031-27481-7%5C_23.

[3] Parosh Aziz Abdulla et al. "String Constraints for Verification". In: *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*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 150–166. DOI: 10.1007/978-3-319-08867-9_10. URL: https://doi.org/10.1007/978-3-319-08867-9%5C_10.

[4] Tomáš Fiedor et al. *Reasoning about Regular Properties: A Comparative Study*. Accepted at CADE'23. 2023. arXiv: 2304.05064 [cs.FL].

[5] Parosh Aziz Abdulla et al. "Chain-Free String Constraints". In: *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*. Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza. Vol. 11781. Lecture Notes in Computer Science. Springer, 2019, pp. 277–293. DOI: 10.1007/978-3-030-31784-3_16. URL: https://doi.org/10.1007/978-3-030-31784-3%5C_16.