Z3str4 String Solver: System Description

SMT-COMP 2020

Murphy Berzish*, Federico Mora†, Mitja Kulczynski‡, Dirk Nowotka‡ and Vijay Ganesh*
*University of Waterloo, Ontario, Canada
†University of California, Berkeley, USA
‡Kiel University, Germany

***Abstract*—In this document we introduce and describe our Z3str4 string solver submitted to the SMTCOMP competition 2020. We briefly review the key insights that underpin the algorithmic design of Z3str4, as well as its setup.**

I. System Overview

Z3str4 is a *multi-armed* solver that incorporates 3 sub-solvers, namely, Z3str3, the length abstraction solver LAS, and Z3seq (a string solver from the Z3 team at Microsoft Research). Of these, the Z3str3 and the LAS solvers were developed by the authors. Z3str3 is built on top of the Z3 theorem prover, from Microsoft Research. The Z3str4 solver additionally makes use of other existing, unmodified components of Z3 [1], namely, the core, bit-vector, and linear arithmetic solvers.

The architecture of Z3str4 is illustrated in Fig. 1. Our solver includes two pre-defined "arms", or sequences of algorithms. The algorithms in these arms are always executed in sequence (as shown), with the possibility of clauses learnt from the previous solver passed on as input to the subsequent one in the sequence. Further, only one arm is ever executed in a run of Z3str4. A arm is chosen using a heuristic we refer to as a "probe", that analyzes the input formula and predicts which of the two arms would have a smaller runtime.

If an algorithm in an arm answers SAT or UNSAT, this answer is returned by Z3str4. Otherwise, a timeout has been reached for that algorithm and the next algorithm in the sequence is called, first augmenting the input formula with certain learned constraints from the previous algorithm in the arm. This allows each algorithm to benefit from the work done by earlier ones, even if they were unable to solve the problem.

As described above, each arm includes three possible algorithms: a length abstraction solver (LAS), described in the following section; Z3str3, which uses the well-known "arrangement" method for solving strings [2], [3] and Z3seq, with certain modifications as described below.

II. Arm Selection

The arm selection method or the "probe" uses static features of the instance to determine which arm to invoke. At a high level, the method checks whether any of the following terms appear in the input formula: string disequalities; negated `prefixof` and `suffixof` terms; and any `contains`, `replace`, or regular expression terms. The intuition here is that the occurrence of these terms generally produce formulas that are hard for the bit-vector solver to handle due to disjunctions of constraints. If these terms do not occur, an arm is chosen in which algorithms that use the bit-vector reduction (LAS and Z3str3) are called before algorithms that don't (Z3seq). Otherwise, the opposite priority is used.

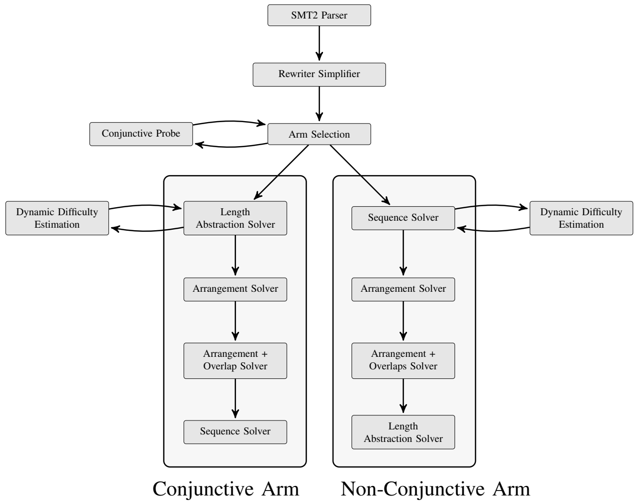
SMT2 Parser
Rewriter Simplifier
Conjunctive Probe
Arm Selection
Dynamic Difficulty Estimation
Length Abstraction Solver
Arrangement Solver
Arrangement + Overlap Solver
Sequence Solver
Sequence Solver
Arrangement Solver
Arrangement + Overlaps Solver
Length Abstraction Solver
Dynamic Difficulty Estimation
Conjunctive Arm
Non-Conjunctive Arm

Fig. 1. Architecture of the Z3str4 tool.

III. Length Abstraction Solver (LAS)

The length abstraction solver (LAS) is a novel CEGAR-style algorithm we developed for Z3str4 that can quickly solve string formulas based on abstractions and refinements of integer constraints implied by string equations. LAS abstracts word equations and other input constraints into length (in)equalities, and uses Z3's built-in arithmetic solver to solve them. Briefly, LAS first constructs an integer over-approximation of the input formula based on implied length constraints and checks if it is satisfiable. If it is unsatisfiable, then it follows that the input is unsatisfiable as well. Otherwise, the solver refines this over-approximation appropriately. This process is repeated until the solver converges to the correct satisfying assignment.

We note that, in the absence of string constraints to guide the search, LAS exhibits two behaviours, depending on the input formula: either it converges to the correct solution in a small number of iterations, or does not converge at all. We thus augmented LAS with a "dynamic difficulty estimation" heuristic to allow the more powerful algorithms to take over if it is determined at runtime that LAS is not performing well.

The heuristic measures the number of queries made to the bit-vector solver and the time taken by each query, and instructs LAS to give up if queries start taking too long to solve or if too many queries are made. Note that in the non-conjunctive arm, as LAS is the last algorithm used, this heuristic is not applied.

IV. Z3 String Solver (Z3str3)

Z3str3 is part of the Z3-str family of string solvers, including Z3-str and Z3str2. Z3str3 reduces the input string constraints to an arrangement (disjunction) of conjunctions of derived string equations. (This algorithm is described in more detail in our previous work [3].) Then, for each arrangement, the solver queries the Z3 arithmetic solver to obtain consistent length assignments to the string variables in them, and reduces the resulting fixed-length equations to a bit-vector representation. This formula is then solved by Z3's bit-vector solver. If the formula is satisfiable, the bit-vector solution can be translated directly to a satisfying assignment for the original string equation. Otherwise, the solver learns a clause that avoids the current length assignment in a conflict-driven clause learning loop, and continues searching for either a different length assignment or a different arrangement.

The hybrid approach we use in Z3str3 combines the efficiency of an unfolding-based strategy (reduction of fixed-length word equations to bit-vectors) with the ability of a word-based strategy to reason about string terms of unbounded length (the arrangement method).

V. Z3 Sequence Solver (Z3seq)

The Z3 sequence solver (Z3seq) is a system implemented by Nikolaj Bjørner and others at Microsoft Research as part of the Z3 SMT solver. Z3seq is a procedure for solving general constraints over "sequences", including the ability to reason about string constraints. We have used the sequence solver "as-is", except for the addition of a "dynamic difficulty estimation" heuristic similar in principle to the one we implemented for LAS. The sequence solver has over 20 rules that it follows when determining the satisfiability of an input. We have observed empirically that queries which are solved efficiently by the sequence solver rarely cause later rules to fail. Our dynamic difficulty estimation (DDE) heuristic keeps track of the index of the last rule which failed, and instructs the solver to give up if that index is above a certain threshold and certain timeout has been reached. We fix this index internally to correspond to the index of the second-last rule to be checked. The rule in question is called `branch_nqs`, which fails if there are any disequalities in the formula that need to be branched on. We have found through experimental evaluation that giving up on this check if it fails, and passing the formula to the next algorithm in the current arm, increases performance significantly. Note that this heuristic is not enabled when the sequence solver is the very last algorithm to be used, because there is no "next" algorithm to fall back onto.

References

[1] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[2] M. Berzish, V. Ganesh, and Y. Zheng, "Z3str3: A string solver with theory-aware heuristics," in *2017 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2017, pp. 55–59.

[3] Y. Zheng, V. Ganesh, S. Subramanian, O. Tripp, M. Berzish, J. Dolby, and X. Zhang, "Z3str2: an efficient solver for strings, regular expressions, and length constraints," *Formal Methods in System Design*, vol. 50, no. 2-3, pp. 249–288, 2017. [Online]. Available: https://doi.org/10.1007/s10703-016-0263-6